Державний вищий навчальний заклад
"Чернівецький індустріальний коледж"

ОСНОВИ ПРОГРАМУВАННЯ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

Конспект лекцій та матеріалів самостійного вивчення
для студентів спеціальності
151 “Автоматизація та комп’ютерно-інтегровані технології”

м.Чернівці, 2018 рік

РОЗРОБЛЕНО ТА ВНЕСЕНО: Державний вищий навчальний заклад «Чернівецький індустріальний коледж»

Розробник; Баляснікова О.А., викладач спец дисциплін ДВНЗ ЧІК

Схвалено на засіданні циклової комісії Видавничої поліграфії та комп'ютерної інженерії

Протокол від "___"___201__ року № ___

Голова циклової комісії _____ (Н.В. Циба)

Обговорено та рекомендовано методичною радою Державного вищого навчального закладу «Чернівецький індустріальний коледж»

Протокол від "___"___ 201__ року № ____

"___"___ 201__ року Голова _____(А.В.Васкан)

ЗМІСТ

РОЗДІЛ І. ОСНОВНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

ЛЕКЦІЯ 1. IBM-СУМІСНІ КОМП'ЮТЕРИ. РОЗВИТОК ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

1.1 Поява і поширення IBM-сумісних комп'ютерів

Поширення персональних комп'ютерів до кінця 70-х років привело до деякого зниження попиту на великі ЕОМ і міни ЕОМ. Це стало предметом серйозного занепокоєння фірми IBM (International Bussines Machines Corporation) – провідної компанії по виробництву великих ЕОМ, і в 1979 році фірма IBM вирішила спробувати свої сили на ринку персональних комп'ютерів. Однак керівництво фірми недооцінило майбутню важливість цього ринку і розглядало створення комп'ютера усього лише як дрібний експеримент. Щоб не витрачати на цей експеримент занадто багато грошей, керівництво фірми надало підрозділу, відповідальному за даний проект, небачену у фірмі волю. Зокрема йому було дозволено не конструювати персональний комп'ютер «з нуля», а використовувати блоки, виготовлені іншими фірмами. І цей підрозділ сповна використовував наданий шанс. Насамперед, як основний мікропроцесор комп'ютера, був обраний новітній тоді 16-розрядний мікропроцесор Intel-8088. Його використання дозволило значно збільшити потенційні можливості комп'ютера, тому що новий мікропроцесор дозволяв працювати з 1 Мб пам'яті, а всі комп'ютери, що тоді існували, були обмежені 64 Кб. У комп'ютері були використані й інші комплектуючі різних фірм, а його програмне забезпечення було доручено розробити невеликій фірмі Microsoft.

У серпні 1981 року новий комп'ютер за назвою IBM PC був офіційно представлений публіці і незабаром після цього він придбав велику популярність у користувачів. Через один-два років комп'ютер IBM PC зайняв ведуче місце на ринку, витиснувши моделі 8-бітових комп'ютерів. Фактично IBM PC став стандартом персонального комп'ютера. Зараз такі комп'ютери ("сумісні з IBM PC") складають близько 90% усіх вироблених у світі персональних комп'ютерів.

Головною перевагою IBM PC була так звана «відкрита архітектура» – спосіб побудови комп'ютера з окремих стандартизованих частин (компонентів). Випуск окремих компонентів є значно дешевшим, ніж побудова цілої обчислювальної машини і багато різних фірм налагодили виробництво окремих компонентів для IBM PC. Це за лічені роки створило новий ринок – IBM-сумісних комп'ютерів, жорстка конкуренція на якому сприяла стабільному зниженню цін і одночасно зростанню якості і потужності персональних комп'ютерів.

Найбільшу ж вигоду від «відкритої архітектури» отримував користувач. Тепер він міг підібрати собі комп'ютер з окремих компонент згідно зі своїми забаганками та фінансовими можливостями, комбінуючи між собою компоненти різних виробників і поєднуючи їх в одну машину. Крім того, якщо надалі у користувача виникало бажання вдосконалити власний комп'ютер, він міг замінювати лише окремі компоненти, оновлювати їх (робити «upgrade») витрачаючи достатньо помірні кошти.

Тому не дивно, що дуже швидко персональні комп'ютери стали використовуватися не лише в домашніх умовах – вони витіснили великі і середні комп'ютери з крупних підприємств, навчальних закладів та урядових установ. І сьогодні майже всюди, де потрібно використання комп'ютерної техніки, використовуються саме персональні комп'ютери.

1.2 Етапи розвитку комп'ютерної техніки. Перші обчислювальні машини

Вважається, що перший у світі ескізний малюнок тринадцяти розрядного десятинного сумуючого пристрою на базі коліщаток з десятьма зубцями був виконаний Леонардо да Вінчі в одному з його щоденників (вчений почав вести цей щоденник ще до відкриття Америки 1492 р.).

1623 року німецький вчений Вільгельм Шиккард запропонував свою модель шестирозрядного десятинного обчислювача, який мав складатися також із зубчатих коліщаток та міг би виконувати додавання, віднімання, а також множення та ділення. Винаходи да Вінчі та Шиккарда були знайдені лише в наш час і залишилися тільки на папері.

1642 року 19-річний французький математик Блез Паскаль сконструював першу в світі працюючу механічну обчислювальну машину, відому як підсумовуюча машина Паскаля («Паскаліна»). Ця машина являла собою комбінацію взаємопов'язаних коліщаток та приводів. На

коліщатках були зображені цифри від 0 до 9. Якщо перше коліщатко робить повний оберт від 0 до 9, автоматично починає рухатись друге коліщатко. Якщо і друге коліщатко доходить до цифри 9, починає обертатися третє і так далі. Машина Паскаля могла лише додавати та віднімати.

1673 року німецький математик Готфрід Вільгельм фон Лейбніц сконструював свою обчислювальну машину. На відміну від Паскаля, Лейбніц використав у своїй машині циліндри, а не коліщатка та приводи. На циліндри було нанесено цифри. Кожен циліндр мав дев'ять рядків виступів та зубців. При цьому перший ряд мав один виступ, другий ряд – два виступи і так до дев'ятого ряду, який мав відповідно дев'ять виступів. Циліндри з виступами були пересувними, оператор надавав їм певного положення. Машина Лейбніца, на відміну від підсумовуючої машини Паскаля, була значно складнішою за конструкцією. Вона була здатна виконувати не тільки додавання та віднімання, але й множення, ділення та обчислювання квадратного кореня.

Винахід першої програмованої обчислювальної машини належить видатному англійському математику Чарлзу Бебіджу (1830 р.). Він присвятив майже все своє життя цій праці, але так і не створив діючу модель. Бебідж назвав свій винахід «Аналітична машина». За планом машина мала діяти завдяки: силі пару. При цьому вона була б здатна сприймати команди, виконувати обчислення та видавати необхідні результати у надрукованому вигляді. Програми в свою чергу мали кодуватися та переноситись на перфокарти. Ідея використання перфокарт була запозичена Бебіджем у французького винахідника Жозефа Жаккара (кінець XVIII ст.). Для контролю ткацьких операцій Жаккар використовував отвори, пробиті в картках. Картки з різним розташуванням отворів давали різні візерунки на плетінні тканини. По суті, Бебідж був першим, хто використав перфокарти стосовно обчислювальних машин.

У своїй машині Бебідж використав також технологію обчислень, запропоновану наприкінці XVIII сторіччя французьким вченим Гаспаром де Проні. Він розділив обчислення на три етапи: розробка чисельного методу, створення програми послідовності арифметичних дій, проведення обчислень шляхом арифметичних операцій над числами згідно зі створеною програмою.

Серед учених, які зробили значний внесок у розвиток обчислювальної техніки, була математик леді Августа Лавлейс – дочка видатного англійського поета лорда Байрона. Саме вона переконала Бебіджа у необхідності використання у його винаході двійкової системи обчислення замість десяткової. Вона також розробила принципи програмування, що передбачали повторення послідовності команд та виконання цих команд за певних умов. Ці принципи використовуються і в сучасній обчислювальній техніці.

Чарлз Бебідж вперше висловив ідею використання перфокарт в обчислювальній техніці, але реалізовано цю ідею було тільки 1887 року Германом Холерітом. Його машина була призначена для обробки результатів перепису населення США. Також Холеріт уперше застосував для організації процесу обчислення електричну силу. Картки використовувались для кодування даних перепису, при цьому на кожну людину була заведена окрема картка. Кодування велося за допомогою певного розташування отворів, що пробивалися в картці по рядках та колонках. Коли картка, що мала розмір банкноти в один долар, пропускалася крізь машину, вона прощупувалась системою голок. Якщо навпроти голки з'являвся отвір, то голка проходила крізь нього і доторкалася до металевої поверхні, що була розташована під карткою. Контакт, який відбувався при цьому, замикав електричний ланцюг, завдяки чому до результату обчислення додавалася одиниця.

1.3 Перші електронно-обчислювальні машини

Перші електронні комп'ютери з'явилися в першій половині XX ст. На відміну від попередніх, вони могли виконувати задану послідовність операцій за програмою, що була задана раніше, або послідовно розв'язувати задачі різних типів. Перші комп'ютери були здатні зберігати інформацію в спеціальній пам'яті.

1934 року німецький студент Конрад Цузе, який працював над дипломним проектом, вирішив створити у себе вдома цифрову обчислювальну машину з програмним управлінням та з використанням (вперше в світі) двійкової системи числення. 1937 року машина 21 (Цузе 1) запрацювала. Вона була 22-розрядною, з пам'яттю на 64 числа і працювала на суто механічній (важільній) базі.

Необхідність у швидких та точних обчисленнях особливо зросла під час Другої світової

війни (1939–1945 рр.) перш за все для розв'язання задач балістики, тобто науки про траєкторію польоту артилерійських та інших снарядів до цілі.

1937 року Джон Атанасов (американський вчений, болгарин за походженням) вперше запропонував ідею використання електронних ламп як носіїв інформації.

В 1942–1943 роках в Англії була створена за участю Алана Тьюрінга обчислювальна машина «Колос». В ній було 2000 електронних ламп. Машина призначалася для розшифрування радіограм німецького вермахту. «Колос» вперше в світі зберігав та обробляв дані за допомогою електроніки, а не механічно.

1944 року під керівництвом професора Гарвардського університету Говарда Айкена було створено обчислювальну машину з автоматичним керуванням послідовністю дій, відому під назвою Марк 1. Ця обчислювальна машина була здатна сприймати вхідні дані з перфокарт або перфострічок. Машина Марк 1 була електромеханічною, для зберігання даних використовувались механічні прилади (коліщатка та перемикачі). Машина Айкена могла виконувати близько однієї операції за секунду та мала величезні розміри: понад 15 м завдовжки та близько 2,5 м заввишки і складалася більш ніж із 750 тисяч деталей.

1946 року групою інженерів під керівництвом Джона Моучлі та Дж. Преспера Еккерта на замовлення військового відомства США було створено машину ЕНІАК, яка була здатна виконувати близько 3 тисяч операцій за секунду. За розмірами ЕНІАК був більшим за Марк 1: понад 30 метрів завдовжки, його об'єм становив 85 м3. Важив ЕНІАК 30 тонн. Замість тисяч механічних деталей Марка 1, в ЕНІАКу було використано 18 тисяч електронних ламп.

Суттєвий внесок у створення ЕОМ зробив американський математик Джон фон Нейман, що брав участь у створенні ЕНІАКа. Фон Нейман запропонував ідею зберігання програми в пам'яті машини. Такі ЕОМ були значним кроком уперед на шляху створення більш досконалих машин. Вони були здатні обробляти команди в різному порядку.

Принципи побудови і функціонування цифрового комп'ютера, сформульовані Джоном фон Нейманом у 1945–46 рр., надовго визначили магістральний шлях розвитку комп'ютерної техніки.

Принцип функціонування

Після завантаження програми (алгоритму й даних для обробки) в запам'ятовуючий пристрій, машина фон-Неймана може працювати автоматично, без втручання оператора. Кожна комірка пам'яті машини має унікальний номер – адресу, а спеціальний механізм, найчастіше – лічильник команд – забезпечує автоматичне виконання необхідної послідовності команд, і визначає на кожному етапі адресу комірки, з якої необхідно завантажити наступну команду.

Перед початком виконання програми в лічильник записується адреса її першої команди. Визначення адреси наступної команди відбувається за одним з наступних сценаріїв:

1. Якщо поточна команда не є командою передачі управління (тобто це просто арифметична або логічна операція над даними), то до поточного значення лічильника додається число, яке дорівнює довжині поточної команди в мінімально адресованих одиницях інформації (зрозуміло, що це можливо за умови, якщо звичайні команди в блоках, не розділених командами передачі управління, розташовуються послідовно в пам'яті, інакше адреса наступної команди може зберігатись, наприклад, безпосередньо в команді)

2. Якщо поточна команда – команда передачі управління (команда умовного або безумовного переходу), яка змінює послідовний хід виконання програми, то в лічильник примусово записується адреса тої команди, яка була замовлена при виконанні переходу, де б вона не знаходилась.

У 50-х – 60-х роках XX століття стало ясно, що класична фон-нейманівська архітектура має багато «вузьких місць». Стали говорити про необхідність її модифікації, а також про необхідність відходу від неї. З'явився новий термін – ненейманівська машина, тобто машина, побудована за принципами, відмінними від фон-нейманівських.

Кардинальний відхід від фон-нейманівської архітектури пов'язаний з появою багатопроцесорних машин, здатних здійснювати паралельні обчислення. Нові архітектури вимагали розробки спеціальних методик організації взаємодії між процесорами та керування ними. В основі ж однопроцесорних комп'ютерних систем, як і раніше, лежать принципи фон-Неймана, хоча і значно модифіковані.

Перша ЕОМ, яка зберігала програми у пам'яті, дістала назву ЕДСАК (Electronic Delay Storage Automatic Calculator – електронний калькулятор з пам'яттю на лініях затримки). Вона була створена в Кембріджському університеті (Англія) 1949 року. З того часу всі ЕОМ є комп'ютерами з програмами, які зберігаються у пам'яті.

1951 року в Києві під керівництвом С. Лєбєдєва незалежно було створено МЕОМ (Мала Електрична Обчислювальна Машина). 1952 року ним же було створено ШЕОМ (Швидкодіюча Електрична Обчислювальна Машина), яка була на той час кращою в світі та могла виконувати близько 8 тисяч операцій за секунду.

1951 року компанія Джона Моучлі та Дж. Преспера Еккерта створила машину UNIVAC (Universal Automatic Computer – універсальна автоматична обчислювальна машина). Перший екземпляр ЮНІВАКа було передано в Бюро перепису населення США. Потім було створено багато різних моделей ЮНІВАКа, які почали застосовуватися у різних сферах діяльності. Таким чином, ЮНІВАК став першим серійним комп'ютером. Крім того, це був перший комп'ютер, в якому замість перфострічок та карток було використано магнітну стрічку.

1.4 Покоління комп'ютерів

Такі комп'ютери, як ЕНІАК, ЕДСАК, ШЕОМ та ЮНІВАК, являли собою. лише перші моделі ЕОМ. Упродовж десятиріччя після створення ЮНІВАКа було виготовлено та введено в експлуатацію в США близько 5000 комп'ютерів.

Гігантські машини на електронних лампах 50-х років склали перше покоління комп'ютерів.

Друге покоління комп'ютерів з'явилося на початку 60-х років, коли на зміну електронним лампам прийшли транзистори. Винайдені у 1948 р. транзистори, як виявилось, були спроможні виконувати всі ті функції, які до цього часу виконували електронні лампи. Але при цьому вони були значно менші за розмірами та споживали набагато менше електроенергії. До того ж транзистори дешевші, випромінюють менше тепла та більш надійні, ніж електронні лампи. І все ж таки найдивовижнішою властивістю транзистора є те, що він один здатен виконувати функції 40 електронних ламп та ще й з більшою швидкістю, ніж вони. В результаті швидкодія машин другого покоління виросла приблизно в 10 разів порівняно з машинами першого покоління, обсяг їх пам'яті також збільшився. Водночас із процесом заміни електронних ламп транзисторами вдосконалювалися методи зберігання інформації. Магнітну стрічку, що вперше було використано в ЕОМ ЮНІВАК, почали використовувати як для введення, так і для виведення інформації. А в середині 60-х років набуло поширення зберігання інформації на дисках.

Поява інтегральних схем започаткувала новий етап розвитку обчислювальної техніки – народження машин третього покоління. Інтегральна схема, яку також називають кристалом, являє собою мініатюрну електронну схему, витравлену на поверхні кремнієвого кристала площею приблизно 10 мм^2. Перші інтегральні схеми (ІС) з'явилися 1964 року. Одна така схема здатна замінити тисячі транзисторів, кожний 3 яких у свою чергу уже замінив 40 електронних ламп. Інакше кажучи, один крихітний, але складний кристал має такі ж самі обчислювальні можливості, як і 30-тонний ЕНІАК!

Швидкодія ЕОМ третього покоління збільшилася приблизно в 100 разів порівняно з машинами другого покоління, а розміри набагато зменшилися.

Четверте покоління – ЕОМ на великих інтегрованих схемах.

Розвиток мікроелектроніки дав змогу розміщати на одному кристалі тисячі інтегрованих схем. Так, 1980 р. центральний процесор невеликої ЕОМ вдалося розташувати на кристалі площею 1,6 см^2. Почалася епоха мікрокомп'ютерів. Швидкодія сучасної ЕОМ в десятки разів перевищує швидкодію ЕОМ третього покоління на інтегральних схемах, в 100 разів – швидкодію ЕОМ другого покоління на транзисторах та в 10 000 разів швидкодію ЕОМ першого покоління на електронних лампах.

П’яте покоління комп'ютерів

Нині створюються та розвиваються ЕОМ п'ятого покоління – ЕОМ на надвеликих інтегрованих схемах. Ці ЕОМ використовують нові рішення у архітектурі комп'ютерної системи та принципи штучного інтелекту.

ЛЕКЦІЯ 2. АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА

2.1 Складові апаратного забезпечення ПК

Будь-який IBM–сумісний персональний комп'ютер включає в себе дві основні складові:

1. Апаратне забезпечення: дві частини ПК та додаткові пристрої.
2. Програмне забезпечення: операційні системи, прикладні програми та мови програмування. Апаратне забезпечення має такі складові:

– системний блок;
– носії інформації та приводи;
– периферійні пристрої.

Системний блок розміщений у корпусі, що має дві основні функції: фізичне кріплення компонент системного блоку та їх захист від зовнішнього середовища. Корпуси бувають двох типів горизонтальна стійка – DeskTop та вертикальна стійка, які поділяють за розміром: mini-tower, midi-tower, big-tower.

У системному блоці, зібраному в корпусі з блоком живлення кріпляться материнська плата зі всіма її компонентами, контролери і адаптери, носії інформації та приводи.

Основним елементом системного блоку є *материнська плата* (MainBoard). На ній кріпиться більшість елементів системного блоку. Зверніть увагу на те, що характеристики материнської плати суттєво впливають на швидкодію комп'ютера в цілому. Основним електронним компонентом самої материнської плати є чіпсет (chipset) – набір мікросхем, що забезпечує взаємодію центрального процесора з іншими компонентами системного блоку. Від типу чіпсету залежать такі характеристики:

– частоти материнської плати і процесора;
– максимальний об'єм оперативної пам'яті;
– кількість і тип дочірніх плат, які можна встановити на материнську плату;
– наявність інтегрованих компонентів.

Основним елементом, що кріпиться на материнській платі є процесор. Поряд з ним може знаходитись співпроцесор, а в більш нових моделях – співпроцесор вбудований в процесор.

Процесор – це велика інтегральна мікросхема, яка є основним елементом ПК і виконує такі функції:

– арифметично-логічні операції;
– узгоджує роботу всіх пристроїв ПК та керує ними;
– є лічильником команд.

В сучасних ПК процесор встановлюється на материнську плату в спеціальний роз'єм (в більш старих моделях він впаяний безпосередньо в плату), що дозволяє в разі необхідності модернізувати ПК.

В зв'язку з високою концентрацією електронних компонентів, в сучасних мікропроцесорах використовують спеціалізовані системи охолодження – радіатори з вентиляторами. Якість та характеристики вентилятора не варто недооцінювати, оскільки при його неефективній роботі відбувається перегрів і відповідно вихід з ладу самого процесора.

Одним з основних елементів, що встановлюють на материнській платі, є мікросхема BIOS (Basic Input Output System) – базова система введення–виведення. BIOS призначений для:

– тестування всіх складових ПК на виявлення помилок;
– збереження інформації про конфігурацію ПК;
– визначення нових пристроїв, підключених до ПК ;
– початкового завантаження ПК;
– завантаження внутрішніх драйверів;
– виконання переривань BIOS.

Переривання – це зупинка роботи однієї програми, або пристрою ПК для виконання дій іншим пристроєм, або програмою. Кожне переривання має свій номер (адресу) від 0 до 20 - переривання BIOS, а 21 і вище – операційної системи.

У BIOS входить підпрограма POST (Power–On Self Test), що здійснює тестування всіх

основних пристроїв апаратної частини в момент включення живлення (перед початком завантаження ПК). У випадку виявлення фатальних помилок P0ST припиняє завантаження і видає код помилки на дисплей та звуковий сигнал. Цей сигнал видається як сукупність довгих і коротких гудків, за якими можна визначити, яка частина апаратного забезпечення вийшла з ладу. По закінченні POST–тесту BIOS здійснює пошук системного диску та початкове завантаження ПК.

Більша частина інформації про конфігурацію ПК знаходиться в CMOS – абревіатура способу виготовлення мікросхем з малим енергоспоживанням. У конфігурацію ПК входять параметри пристроїв апаратної частини. Дані в CMOS–пам'яті, на відміну від постійної пам'яті, можна дуже легко змінити за допомогою програми CMOS–SETUP, що також є частиною BIOS. Завантажити CMOS–SETUP можна лише відразу при початку завантаження ПК, найчастіше за допомогою клавіші DELETE (може використовуватись й інша клавіша в залежності від фірми-виробника BIOS). Якщо викликаємо CMOS–SETUP, відкривається меню з командами, що дозволяє змінювати параметри. Оскільки інформація, що знаходиться в CMOS при вимкненні живлення знищується, то на мікросхему BIOS постійно подається напруга від спеціального акумулятора, що встановлений на материнській платі. Від нього також живиться таймер (системний годинник).

Сучасні версії BIOS використовують спеціальні підпрограми для автоматичного визначення нових зовнішніх пристроїв (стандарт Plug and PIay – "вмикай і працюй"). Специфікація визначає засоби і способи взаємодії периферійних пристроїв з BIOS ПК та операційною системою, що дозволяє вирішувати конфлікти через системні ресурси з мінімальною участю користувача. Практично це означає непотрібність на додаткових платах будь-яких перемичок і перемикачів. Зверніть увагу, що для повної реалізації технології Plug and PIay, необхідними є наявність чотирьох основних компонентів: BIOS, що підтримує цю технологію, відповідної операційної системи, плати розширення, та відповідного програмного забезпечення.

Важливою функцією BIOS є те, що вона містить внутрішні драйвери основних компонент ПК (клавіатури, дисководу, портів і т.д.), без яких неможливе початкове завантаження ПК.

Контролер чи адаптер – це спеціалізована плата, що служить елементом зв'язку між процесором та зовнішніми пристроями, або виконує власні спеціалізовані функції.

Тоді, як контролер є керуючим елементом периферійного пристрою, то адаптер – лише узгоджуючий елемент, що виконує функції дешифратора базових адрес зовнішнього пристрою.

На материнській платі знаходяться мікросхеми оперативної пам'яті (ОП).

2.2 Оперативна пам'ять

Оперативна пам'ять (RAM – Random Access Memory) – це пам'ять, в яку поміщаються завантажені програми та дані, що ними обробляються. Власне з цією пам'яттю працює процесор. При завантаженні прикладної програми вона копіюється з носія інформації в ОП.

Програма, яка постійно знаходиться в ОП, аж до перезавантаження ПК, називається резидентною.

Оперативна пам'ять характеризується високою швидкодією та коротким часом доступу.

Нині існує декілька типів оперативної пам'яті, основними відмінностями між якими є різниця в швидкодії, методах регенерації, передачі та зберіганні даних:

– динамічна. В мікросхемах динамічної пам'яті кожна комірка – це конденсатор, що поступово розряджається (звідси і назва "динамічна"). Отож потрібна постійно їх регенерація (перезарядка). На це витрачається багато часу, тому швидкодія даної пам'яті є найнижчою (тактова частота 66 МГц);

– статична пам'ять. Кожна комірка статичної пам'яті, це напівпровідниковий діод, який не вимагає постійної регенерації. Вона є досить дорогою і тому використання великих об'ємів статичної пам'яті на ПК недоцільне. В основному її використовують як кеш-пам'ять (її швидкодія аналогічна процесору);

– синхро-динамічна, яка є пришвидшеним варіантом динамічної пам'яті. Вона може працювати на тактовій частоті 100,133 і більше МГц;

– подвійна, коли для передачі даних використовується прямий та зворотній хід сигналу. В

результаті цього, при збереженні частоти роботи шини об'єм корисних даних, що передаються за один такт синхронізаційного імпульсу збільшується вдвічі. Сучасні версії працюють з збільшеними частотами системної шини;

– RDRAM. Досить новий тип динамічної пам'яті, який використовується у ПК на базі Pentium 4. Основна відмінність – збільшена частота роботи 400 та 533 МГц (800 і 1066МГц при двосторонньому ході сигналу).

Оперативна пам'ять, за адресацією логічно поділена на два основних типи: основна до 1М і розширена – вище 1М. Цей поділ склався історично в процесі розвитку ПК. Необхідність досягнення сумісності між системою і периферійними пристроями заставляє розробників і в сучасних системах використовувати практично такий же розподіл пам'яті, як і у перших ПК.

Одним з елементів материнської плати є спеціальні роз'єми, що містять контакти для шини керування, шини даних та адресної шини. Вони мають назву слоти розширення. Використання слотів розширення підтверджує принцип відкритої архітектури IBM–сумісних ПК. В слоти розширення встановлюються додаткові плати (дочірні), які називають контролерами та адаптерами.

2.3 Інтерфейси ПК

Порти (інтерфейси) – це стандартні двонаправлені канали, якими відбувається передача даних між ПК і зовнішніми пристроями.

Порти поділяють на:

– послідовні – C0M1/C0M4 – передача даних здійснюється побітно (послідовно сигнал за сигналом).

– паралельні – LPT1/LPT4 – передача здійснюється по 8 біт одночасно (8 сигналів ідуть паралельно).

Послідовні порти використовують для під'єднання повільних зовнішніх пристроїв (маніпулятор «миша», модем та ін.) До паралельних портів під'єднують принтери, сканери, цифрові камери.

Найнеобхіднішим серед всіх адаптерів є відео адаптер який перетворює цифрові сигнали процесора у відеосигнали для дисплею.

Електричні сигнали і лінії, якими вони передаються до слотів розширення, називають шинами. В зв'язку з еволюцією ПК, на нинішній день існує декілька стандартів шин:

– ISA,

– EISA (не зважаючи на те, що шина досить стара, її інколи використовують ще й нині),

– PCI (вона забезпечує високу швидкодію, гнучкість)

Наприкінці 90-х років значно зросли вимоги до комп'ютерної графіки. Вона стала такою об'ємною, що шина PCI вже не справлялась з таким великим потоком інформації. Так з'явились шини

– AGP (призначена для встановлення лише відеоадаптера)

– USB (з портами для підключення IBM-сумісних мишей, клавіатур, відеокамер та інших периферійних пристроїв). Новий стандарт полегшує роботу систем, що перевантажені периферією.

Останнім часом поширена високошвидкісна (до 1 Gbit/c) послідовна шина FireWire, яка використовується для під'єднання високошвидкісної периферії, повністю підтримує технологію Plug & Play та "гарячу" заміну пристроїв, максимальна кількість яких – 63.

В системному блоці розміщено також блок живлення, який перетворює змінну напругу від електромережі у постійну напругу величиною 12В, 5В, та 3,3В. Живлення до материнської плати та всіх інших пристроїв подається за допомогою спеціальних кабелів, які закінчуються типовими роз'ємами.

Вмикач живлення виведено на передню панель системного блоку - кнопка POWER. Більшість сучасних блоків живлення має стандартну конструкцію, але різну вихідну потужність від 150 до 500Вт. Чим більша вихідна потужність, тим більшу кількість пристроїв можна встановити в системний блок чи під'єднати до нього.

2.4 Периферійні пристрої

Периферійні пристрої ПК поділяють на дві основні групи:

– пристрої введення інформації,
– пристрої виведення інформації.

Пристрої введення інформації призначені для внесення інформації в ПК та керування його роботою. До пристроїв введення інформації відносять:

– клавіатури,
– маніпулятори «миша» та трекболи,
– дігітайзери,
– сканери та ін.

Клавіатури

З функціональної точки зору клавіатура являє собою сукупність певних давачів, що сприймають натискування на клавіші і замикають таким чином певні електричні кола. Створюються послідовності сигналів (по 1 байту), які передають процесору коди клавіш.

Класична комп'ютерна клавіатура зазвичай має 102 клавіші. Сучасні клавіатури, як правило, оснащені додатковими клавішами. Додаткові клавіші використовуються для управління мультимедіа–програвачами, для навігації в мережі інтернет–браузерах, запуску додатків та ін.

Хід клавіш – відстань, яку проходить клавіша при натисненні до моменту зіткнення контактів. Для швидкого друку зазвичай вибирають клавіатури з невеликою довжиною ходу (3 мм і менше).

Комп'ютерні миші

За допомогою миші зручно керувати об'єктами, вікнами, працювати з меню, кнопками, піктограмами і т.д.

У нижній частині оптичної миші встановлений спеціальний світлодіод, який підсвічує поверхню, по якій переміщається миша. Мініатюрна камера «фотографує» поверхню понад тисячу разів за секунду, передаючи ці дані процесору, який і робить висновки про зміну координат.

Недоліками даної миші є:

– складність її одночасної роботи з графічними планшетами, останні через свої апаратні особливості іноді втрачають справжній напрямок сигналу при русі пера і починають спотворювати траєкторію руху інструменту при малюванні. Щоб вирішити цю проблему рекомендується використовувати лазерні маніпулятори;

– також, до недоліків оптичних мишей деякі люди відносять світіння таких мишей навіть при вимкненому комп'ютері.

Більш досконалим різновидом оптичного датчика, який використовує для підсвічування напівпровідниковий лазер, є оптичні лазерні миші. Для підсвічування поверхні використовується лазер. Лазер, на відміну від світлодіода, випускає вузькоспрямований пучок світла, завдяки чому одержувані сенсором зображення більш контрастні, а позиціонування курсора досягають високої точності. Оптичні лазерні миші менш вимогливі до робочої поверхні.

Особливим типом інтерфейсу є зв'язок на основі інфрачервоних портів. Перевагою такого з'єднання є безкабельний зв'язок, але при цьому необхідно дотримуватись зони прямої видимості.

Миші так само, як і клавіатури бувають провідні, що підключаються через USB або PS/2 порт, і бездротові.

За видом з'єднання з комп'ютером бездротові миші бувають:

– з інфрачервоним зв'язком – між мишею і спеціальним прийомним вузлом, підключеним до комп'ютера. Істотним недоліком є необхідність відсутності перешкоди між мишею і базою.

– з радіозв'язком – даний вид зв'язку дозволив позбутися недоліків інфрачервоної і повністю витіснив її.

– індукційні – живляться від спеціального робочого килимка або графічного планшета. Таким чином, миша вільна від проводу, але не працює без індукційного майданчика.

– bluetooth – такі миші не потребують приймального блоку й додаткових драйверів, проте відрізняються високим енергоспоживанням.

Сканери

Сканером називають пристрій, що дозволяє вводити в комп'ютер образи зображень, які пропонуються у вигляді тексту, малюнків, слайдів, фотографій та іншої графічної інформації. Сканери класифікують за конструкцією механізму руху та типом введеного зображення, способом під'єднаня до системного блоку.

За конструкцією сканери поділяють на: ручні, настільні і барабанні. Для того, щоб ввести в комп'ютер будь-яке зображення за допомогою ручного сканера, потрібно без різких рухів провести сканером по зображенню. Основними перевагами ручних сканерів є невеликі розміри і порівняно низька ціна.

Настільні сканери називають по-різному: сторінкові, планшетні, автосканери. Такі сканери дозволяють вводити зображення формату від А4 до А1. Існують три різновиди настільних сканерів: планшетні, рулонні і проекційні.

Особливістю планшетних сканерів є те, що скануюча голівка переміщається відносно аркуша паперу за допомогою крокового двигуна. Такі сканери досить дорогі, але вони мають найбільшу серед настільних роздільну здатність. Вони трохи нагадують копіювальні машини. Для сканування необхідно відкрити кришку сканера, покласти сканований листок на скляну пластину зображенням вниз, і закрити кришку. Все подальше керування процесом сканування здійснюється з використанням спеціального програмного забезпечення. Зрозуміло, що розглянута конструкція сканера дозволяє сканувати не лише окремі листки, а й сторінки журналів і книг.

Робота рулонних сканерів нагадує роботу факсу. Окремі листки документів проходять всередині сканера, при цьому здійснюється їх сканування. В даному випадку скануюча голівка фіксована і відносно неї рухається папір.

Третій різновид настільних сканерів – проекційні сканери, що нагадують своєрідний проекційний апарат. Документ кладуть на поверхню, догори сканованим зображенням, блок сканування знаходиться при цьому також зверху. Переміщується лише скануючий пристрій. Основною особливістю цих сканерів є можливість сканування проекційних тривимірних предметів.

Принцип роботи планшетного сканера полягає в наступному. Скановане зображення освітлюється спеціальним світлом. Як джерело світла використовують лампу з холодним катодом. Відбиті від поверхні світлові промені потрапляють, через систему дзеркал, на матрицю фоточутливих напівпровідникових елементів давачів, що розміщені в один ряд для чорно-білого сканера, і в три ряди для кольорового. В залежності від елементу сканованого зображення, ці промені мають різну інтенсивність і створюють на чутливих елементах напругу певної величини, а вона перетворюється в цифрову форму в аналого–цифровому перетворювачі (АЦП). Цей принцип використовується й в інших настільних та ручних сканерах.

Дисплей

Найважливішою з периферійних систем є відеосистема, що призначена для виводу текстової та графічної інформації. Відеосистема складається, в основному, з двох частин: відеоадаптера і дисплею. Відеоадаптер – це електронна схема, яка взаємодіючи з процесором, формує зображення. Дисплей візуалізує сформоване зображення на екрані.

Дисплеї за принципом роботи поділяють, на такі що діють:

– на основі електронно-променевих трубок;
– на рідких кристалах.

Електронно-променева трубка (ЕПТ) була створена у 1897 році німецьким вченим Карлом Фердинандом Брауном. Принцип дії моніторів на базі ЕПТ полягає в тому, що пучок електронів, що вилітають з електронної пушки, потрапляючи на екран, вкритий люмінофором, викликає його світіння. На шляху пучка електронів переважно знаходяться допоміжні електроди: відхиляюча система, що дозволяє змінити напрям пучка і модулятор, який регулює яскравість зображення.

Будь-яке текстове чи графічне зображення на екрані монітора комп'ютера складається з великої кількості дискретних точок люмінофора, які називають пікселями. Піксель є найменшим елементом зображення. Тому такі дисплеї ще називають растровими (растр – сукупність пікселів). Електронний промінь в такому випадку періодично сканує весь екран, утворюючи на ньому близько розміщені елементи зображення.

Екрани LCD-моніторів (Liquid Crystal Display, рідкокристалічні монітори) зроблені з речовини ціанофеніл, яка знаходиться в рідкому стані, але при цьому має деякі властивості, притаманні кристалічним тілам. Робота рідкокристалічних матриць заснована на такій властивості світла, як поляризація. Звичайне світло є неполяризованим, тобто амплітуди його хвиль лежать у безлічі площин. Однак існують речовини, здатні пропускати світло тільки з однієї площини. Ці речовини називають поляризаторами, оскільки пройшовши крізь них світло стає поляризованим лише в одній площині. Якщо взяти два поляризатора, площини поляризації яких розташовані під кутом 90° один до одного, світло через них пройти не зможе. Якщо ж розташувати між ними щось, що зможе повернути вектор поляризації світла на потрібний кут, ми отримаємо можливість керувати яскравістю світіння, гасити і запалювати світло так, як нам хочеться. Такий, якщо описувати коротко, принцип роботи РК-матриці.

У кольорових матрицях кожен піксель формується з трьох кольорових точок (червоної, зеленої і синьої), тому додається ще й кольоровий фільтр. У кожен момент часу кожна з трьох комірок матриці, що становлять один піксель, знаходиться або в увімкненому, або у вимкненому положенні. Комбінуючи їх стан, отримуємо відтінки кольору, а вимикаючи всі одночасно – білий колір.

Глобально матриці діляться на пасивні (прості) і активні. В пасивних матрицях управління проводиться попіксельно, тобто по порядку від комірки до комірки в рядку. Проблемою, що встає при виробництві РК-екранів за цією технологією, стало те, що при збільшенні діагоналі збільшуються і довжини провідників, по яких передається струм на кожен піксель. По-перше, поки буде змінений останній піксель, перший встигне втратити заряд і згаснути. По-друге, велика довжина вимагає більшої напруги, що призводить до зростання перешкод і наведень. Це різко погіршує якість картинки і точність передачі кольору. Через це пасивні матриці застосовуються тільки там, де не потрібні велика діагональ і висока щільність відображення. Для подолання цієї проблеми були розроблені активні матриці. Основою став винахід технології, відомої всім по абревіатурі TFT, що означає Thin Film Transistor – тонкоплівковий транзистор. Завдяки TFT з'явилася можливість управляти кожним пікселем на екрані окремо. Це різко скорочує час реакції матриці і робить можливими великі діагоналі матриць. Транзистори ізольовані один від одного і підведені до кожної клітинки матриці. Вони створюють поле, коли їм наказує драйвер матриці. Для того, щоб комірка не втратила заряд передчасно, до неї додається невеликий конденсатор, який грає роль буферної ємності. За допомогою цієї технології вдалося радикально зменшити час реакції окремих осередків матриці.

Важливими характеристиками дисплею є :

– роздільна здатність;
– кадрова частота;
– крок (розмір) пікселів;
– розмір екрану по діагоналі.

Принтери

Принтери – це пристрої для виведення інформації на тверді копії (папір, плівки, тканину та ін). Всі друкуючі пристрої поділяють на:

– послідовні - друкують на твердій копії посимвольно;
– стрічкові, друк здійснюється пострічково;
– сторінковні, друк здійснюється посторінково.

В свою чергу в кожній групі можна виділити принтери ударної і безударної дії. Принтери ударної дії поділяють на матричні й символьні (на нинішній день практично не використовуються). Коли говорять про матричні принтери, то мова йде про пристрої ударної дії, наприклад всім відомі моделі.

В послідовних ударних матричних принтерах зображення формується з точок, які утворюються внаслідок ударів голок по фарбуючій стрічці.

До пристроїв безударної дії відносять струменеві та лазерні принтери, оскільки в цих пристроях виконуючий механізм не торкається паперу. Вони працюють практично безшумно, що є однією з переваг у порівнянні з ударними. Струменеві чорнильні принтери використовують

«бульбашкову» технологію, або п'єзоефект.

У чорнильних струменевих принтерах, як і в послідовних матричних, друкуюча головка рухається тільки в горизонтальному напрямку, а папір кроково - вертикально. На друкуючій головці розміщені сопла (вертикальні отвори), через які розбрискуються мікрокраплі чорнила. Кількість сопел у різних моделях принтерів, як правило, може коливатися від 12 до 64. Оскільки діаметр кожного сопла досить малий, то зображення отримуємо значно чіткіше, порівняно з матричними.

В наш час великого поширення набули лазерні принтери. Вони використовують електрографічний принцип створення зображення. Процес створення такого зображення включає в себе створення макету зображення на барабані чи планшеті і його перенесення на папір. Найбільш важливими частинами лазерного принтера є фоточутливий елемент (друкуючий барабан, планшет), напівпровідниковий лазер і прецизійна оптико-механічна система, що переміщує лазерний промінь.

Напівпровідниковий лазер генерує тонкий світловий промінь, який, відбиваючись від дзеркала, формує зображення на світлочутливому фотоприймальному барабані. Барабанові заздалегідь надається негативний статичний заряд за допомогою заряджаючих щіток або коротрону. Коли лазерний промінь потрапляє на барабан, він нейтралізує негативний заряд маленької частинки барабану і таким чином формує образ майбутнього зображення. Для отримання зображення лазер повинен включатися і виключатися, що забезпечується спеціальною керуючою електронікою принтера. Дзеркало, що обертається, повертає промінь лазера на новий рядок. Після формування кожного нового рядка спеціальний прецизійний двигун повертає барабан так, щоб можна було формувати наступний рядок.

Далі, при повертанні барабану, частина його поверхні, що оброблена лазером потрапляє в блок розподілу тонера. Тонер – це спеціальний дуже дрібнозернистий чорний порошок. Вал розподілу тонера обертається дуже близько від фоточутливого барабану і частинки тонеру притягуються до ділянок, що були нейтралізовані лазерним променем і таким чином формується зображення на барабані. Барабан продовжує обертатись і доторкається своєю поверхнею до поверхні аркуша паперу. Швидкість подачі паперу відповідає швидкості обертання фоточутливого барабану. Під аркушем паперу знаходиться ще один коротрон, за допомогою якого поверхня паперу заряджається і тоді частинки тонеру переносяться з барабану на папір.

Далі зображення закріплюється на твердій копії за рахунок нагрівання частинок тонера спеціальним барабаном до температури плавлення.

На нинішній день важливою характеристикою принтерів є можливість кольорового друку. Принципи кольорового друку, в струменевих і лазерних принтерах, схожі з принципом традиційного чорно-білого друку. Особливістю кольрового друку є використання декількох основних кольорів фарби чи тонеру, що при змішуванні утворюють різні кольори та відтінки.

Якість принтера в основному визначається його роздільною здатністю. Роздільна здатність визначається в dpi, що відповідає кількості крапок зображення на 1 дюйм. Для деяких моделей принтерів роздільна здатність визначається окремо по вертикалі і горизонталі аркуша.

До переваг лазерних принтерів належать висока швидкодія, що визначається кількістю роздрукованих сторінок за хвилину, а також можливість друку не тільки на папері, а й на плівках та інших твердих копіях.

ЛЕКЦІЯ 3. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА

3.1 Програми та програмне забезпечення

В основу роботи комп'ютерів покладено програмний принцип керування, який полягає в тому, що комп'ютер виконує дії за заздалегідь заданою програмою. Цей принцип забезпечує універсальність використання комп'ютера: у певний момент часу розв'язується задача відповідно до вибраної програми. Після її завершення у пам'ять завантажується інша програма і т.д. *Програма* – це запис алгоритму розв'язання задачі у вигляді послідовності команд або операторів мовою, яку розуміє комп'ютер. Кінцевою метою любої комп'ютерної програми є керування апаратними засобами.

Для нормального розв'язання задач на комп'ютері потрібно, щоб програма була налагоджена, не потребувала дороблень і мала відповідну документацію. Тому стосовно роботи на комп'ютері часто використовують термін *програмне забезпечення* (software), під яким розуміють сукупність програм, процедур і правил, а також документації, що стосуються функціонування системи оброблення даних.

Програмне та апаратне забезпечення у комп'ютері працюють у нерозривному зв'язку та взаємодії. Склад програмного забезпечення обчислювальної системи називається програмною конфігурацією. Між програмами існує взаємозв'язок, тобто багато програм працюють, базуючись на програмах нижчого рівня. Міжпрограмний інтерфейс - це розподіл програмного забезпечення на декілька пов'язаних між собою рівнів.

3.2 Рівні програмного забезпечення

Рівні програмного забезпечення являють собою піраміду, де кожен вищій рівень базується на програмному забезпеченні попередніх рівнів.

Прикладний рівень
Службовий рівень
Системний рівень
Базовий рівень

Базовий рівень

Цей рівень є найнижчим рівнем програмного забезпечення. Відповідає за взаємодію з базовими апаратними засобами. Базове програмне забезпечення міститься у складі базового апаратного забезпечення і зберігається у спеціальних мікросхемах постійного запам'ятовуючого пристрою (ПЗП), утворюючи базову систему введення-виведення BIOS. Програми та дані записуються у ПЗП на етапі виробництва і не можуть бути змінені в процесі експлуатації.

Системний рівень

Системний рівень – є перехідним. Програми цього рівня забезпечують взаємодію інших програм комп'ютера з програмами базового рівня і безпосередньо з апаратним забезпеченням. Від програм цього рівня залежать експлуатаційні показники всієї обчислювальної системи. При під'єднанні до комп'ютера нового обладнання, на системному рівні повинна бути встановлена програма, що забезпечує для решти програм взаємозв'язок із цим пристроєм. Конкретні програми, призначені для взаємодії з конкретними пристроями, називають драйверами.

Інший клас програм системного рівня відповідає за взаємодію з користувачем. Завдяки йому є можливість вводити дані у обчислювальну систему, керувати її роботою й отримувати результат у зручній формі. Це засоби забезпечення користувацького інтерфейсу, від них залежить зручність та продуктивність роботи з комп'ютером.

Сукупність програмного забезпечення системного рівня утворює ядро операційної системи комп'ютера. Наявність ядра операційної системи – є першою умовою для можливості практичної роботи користувача з обчислювальною системою. Ядро операційної системи виконує такі функції: керування пам'яттю, процесами введення-виведення, файловою системою, організація взаємодії та диспетчеризація процесів, облік використання ресурсів, оброблення команд і т.д.

Службовий рівень

Програми цього рівня взаємодіють як із програмами базового рівня, так і з програмами системного рівня. Призначення службових програм (утиліт) полягає у автоматизації робіт по перевірці та налаштуванню комп'ютерної системи, а також для покращення функцій системних програм. Деякі службові програми (програми обслуговування) відразу додають до складу операційної системи, доповнюючи її ядро, але більшість є зовнішніми програмами і розширюють функції операційної системи. Тобто, у розробці службових програм відслідковуються два напрямки: інтеграція з операційною системою та автономне функціонування.

Класифікація службових програмних засобів

1. Диспетчери файлів (файлові менеджери). За їх допомогою виконується більшість операцій по обслуговуванню файлової структури копіювання, переміщення, перейменування файлів, створення каталогів (папок), знищення об'єктів, пошук файлів та навігація у файловій структурі. Базові програмні засоби містяться у складі програм системного рівня і встановлюються разом з операційною системою

2. Засоби стиснення даних (архіватори). Призначені для створення архівів. Архівні файли мають підвищену щільність запису інформації і відповідно, ефективніше використовуються носії інформації.

3. Засоби діагностики. Призначені для автоматизації процесів діагностування програмного та апаратного забезпечення. Їх використовують для виправлення помилок і для оптимізації роботи комп'ютерної системи.

4. Програми інсталяції (встановлення). Призначені для контролю за додаванням у поточну програмну конфігурацію нового програмного забезпечення. Вони слідкують за станом і зміною оточуючого програмного середовища, відслідковують та протоколюють утворення нових зв'язків, загублені під час знищення певних програм. Прості засоби управління встановленням та знищенням програм містяться у складі операційної системи, але можуть використовуватись і додаткові службові програми.

5. Засоби комунікації. Дозволяють встановлювати з'єднання з віддаленими комп'ютерами, передають повідомлення електронної пошти, пересилають факсимільні повідомлення тощо.

6. Засоби перегляду та відтворення. Переважно для роботи з файлами, їх необхідно завантажити у "рідну" прикладну систему і внести необхідні виправлення. Але, якщо редагування не потрібно, існують універсальні засоби для перегляду (у випадку тексту) або відтворення (у випадку звука або відео) даних.

7. Засоби комп'ютерної безпеки. До них відносяться засоби пасивного та активного захисту даних від пошкодження, несанкціонованого доступу, перегляду та зміни даних. Засоби пасивного захисту - це службові програми, призначені для резервного копіювання. Засоби активного захисту застосовують антивірусне програмне забезпечення. Для захисту даних від несанкціонованого доступу, їх перегляду та зміни використовують спеціальні системи, базовані на криптографії.

Прикладний рівень

Програмне забезпечення цього рівня являє собою комплекс прикладних програм, за допомогою яких виконуються конкретні завдання (від виробничих до творчих, розважальних та навчальних). Між прикладним та системним програмним забезпеченням існує тісний взаємозв'язок. Універсальність обчислювальної системи, доступність прикладних програм і широта функціональних можливостей комп'ютера безпосередньо залежать від типу наявної операційної системи, системних засобів, що містяться у її ядрі й взаємодії комплексу людина-програма-обладнання.

3.3 Класифікація прикладного програмного забезпечення

1. *Текстові редактори*. Основними функціями є введення та редагування текстових даних.

2. *Текстові процесори*. Дозволяють форматувати, тобто оформлювати текст. Основними засобами текстових процесорів є засоби забезпечення взаємодії тексту, графіки, таблиць та інших об'єктів, що складають готовий документ, а також засоби автоматизації процесів редагування та форматування.

3. *Графічні редактори*. Широкий клас програм, що призначені для створення та обробки графічних зображень. Розрізняють три категорії:

– растрові редактори;
– векторні редактори;
– 3-D редактори (тривимірна графіка).

У растрових редакторах графічний об'єкт представлений у вигляді комбінації точок (растрів), що мають свою яскравість та колір. Такий підхід ефективний, коли графічне зображення має багато кольорів і інформація про колір елементів набагато важливіша за інформацію про їх форму. Це характерно для фотографічних та поліграфічних зображень. Застосовують для обробки зображень, створення фотоефектів і художніх композицій.

Векторні редактори відрізняються способом представлення даних про зображення. Об'єктом є не точка, а лінія. Кожна лінія розглядається, як математична крива III порядку і представлена формулою. Таке представлення компактніше за растрове, дані займають менше місця, побудова об'єкта супроводжується підрахунком параметрів кривої у координати екранного зображення, і відповідно, потребує більш продуктивних обчислювальних систем. Широко застосовуються у рекламі, оформленні обкладинок поліграфічних видань.

Редактори тривимірної графіки. Використовують для створення об'ємних композицій. Мають дві особливості: дозволяють керувати властивостями поверхні в залежності від властивостей освітлення, а також дозволяють створювати об'ємну анімацію.

4. *Системи управління базами даних* (СУБД). Базою даних називають великі масиви даних організовані у табличні структури. Основні функції СУБД:

– створення пустої структури бази даних;
– наявність засобів її заповнення або імпорту даних із таблиць іншої бази;
– можливість доступу до даних, наявність засобів пошуку й фільтрації.

У зв'язку з поширенням мережевих технологій, від сучасних СУБД вимагається можливість роботи з віддаленими й розподіленими ресурсами, що знаходяться на серверах Інтернету.

5. *Електронні таблиці*. Надають комплексні засоби для збереження різних типів даних та їх обробки. Основна особливість електронних таблиць полягає у автоматичній зміні вмісту всіх комірок при зміні відношень, заданих математичними або логічними формулами.

6. *Системи автоматизованого проектування* (CAD-системи). Призначені для автоматизації проектно-конструкторських робіт. Застосовуються у машинобудуванні, приладобудуванні, архітектурі. Окрім графічних робіт дозволяють проводити прості розрахунки та вибір готових конструктивних елементів з існуючої бази даних. САПР є необхідним компонентом для гнучких виробничих систем (ГВС) та автоматизованих систем управління технологічними процесами (АСУ ТП).

7. *Настільні видавничі системи*. Автоматизують процес верстання поліграфічних видань. Займає проміжний стан між текстовими процесами та САПР. Видавничі системи відрізняються розширеними засобами управління взаємодії тексту з параметрами сторінки і графічними об'єктами, але мають слабші можливості по автоматизації вводу та редагування тексту. Їх доцільно застосовувати до документів, що попередньо оброблені у текстових процесорах та графічних редакторах.

8. *Редактори HTML* (Web-редактори). Особливий клас редакторів, що об'єднують у собі можливості текстових та графічних редакторів. Призначені для створення і редагування Web-сторінок Інтернету.

9. *Браузери* (засоби перегляду Web-документів). Програмні засоби призначені для перегляду електронних документів, створених у форматі HTML. Відтворюють окрім тексту та графіки, також музику, людську мову, радіопередачі, відеоконференції і дозволяють працювати з електронною поштою.

10. *Системи автоматизованого перекладу*. Розрізняють електронні словники та програми перекладу мови. Електронні словники – це засоби для перекладу окремих слів у документі. Потрібні для професійних перекладачів, які самостійно перекладають текст. Програми автоматичного перекладу отримують текст на одній мові і видають текст на іншій, тобто автоматизують переклад. При автоматизованому перекладі неможливо отримати якісний вихідний текст, оскільки все зводиться до перекладу окремих лексичних одиниць. Програми автоматичного

перекладу доцільно використовувати:

– при абсолютному незнанні іноземної мови;
– при необхідності швидкого ознайомлення з документом;
– для перекладу на іноземну мову;
– для створення чернетки, що потім буде підправлено повноцінним перекладом.

11. *Інтегровані системи діловодства*. Засоби для автоматизації робочого місця керівника. Зокрема, це функції створення, редагування і форматування документів, централізація функцій електронної пошти, факсимільного та телефонного зв'язку, диспетчеризація та моніторинг документообігу підприємства, координація дій підрозділів, оптимізація адміністративно-господарської діяльності й поставка оперативної та довідкової інформації.

12. *Бухгалтерські системи*. Містять у собі функції текстових, табличних редакторів та СУБД. Призначені для автоматизації підготовки початкових бухгалтерських документів підприємства та їх обліку, регулярних звітів по підсумках виробничої, господарської та фінансової діяльності у формі прийнятної для податкових органів, позабюджетних фондів та органів статистичного обліку.

13. *Фінансові аналітичні системи*. Використовують у банківських та біржових структурах. Дозволяють контролювати та прогнозувати ситуацію на фінансових, торгівельних та ринків сировини, виконувати аналіз поточних подій, готувати звіти.

14. *Експертні системи*. Призначені для аналізу даних, що містяться у базах знань і видачі результатів, при запиті користувача. Такі системи використовуються, коли для прийняття рішення потрібні широкі спеціальні знання. Використовуються у медицині, фармакології, хімії, юриспруденції. З використанням експертних систем пов'язана область науки, що зветься інженерією знань.

15. *Геоінформаційні системи* (ГІС). Призначені для автоматизації картографічних та геодезичних робіт на основі інформації, отриманої топографічним або аерографічними методами.

16. *Системи відеомонтажа*. Призначені для цифрової обробки відеоматеріалів, монтажу, створення відеоефектів, виправлення дефектів, додавання звуку, титрів та субтитрів. Характерною особливістю є підвищені вимоги до мультимедійної складової.

17. *Інструментальні мови та системи програмування*. Ці засоби служать для розробки нових програм. Комп'ютер "розуміє" і може виконувати програми у машинному коді. Кожна команда при цьому має вигляд послідовності нулів й одиниць. Писати програми машинною мовою дуже незручно, а їх надійність низка. Тому програми розробляють мовою, зрозумілою людині (інструментальна мова або алгоритмічна мова програмування), після чого спеціальною програмою, яка називається транслятором, текст програми перекладається (транслюється) на машинний код.

Транслятори бувають двох типів:

– інтерпретатори;
– компілятори.

Інтерпретатор читає один оператор програми, аналізує його і відразу виконує, після чого переходить до оброблення наступного оператора. *Компілятор* спочатку читає, аналізує та перекладає на машинний код усю програму і тільки після завершення всієї трансляції ця програма виконується. Інструментальні мови поділяються на мови низького рівня (близькі до машинної мови) та мови високого рівня (близькі до мови людини). До мов низького рівня належать асемблери, а високого – Pascal, Basic, C/C++, мови баз даних і т.д. Систему програмування, крім транслятора, складають текстовий редактор, компонувальник, бібліотека стандартних програм, налагоджувач, візуальні засоби автоматизації програмування. Прикладами таких систем є Delphi, Visual Basic, Visual C++ та ін.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: ІНСТРУМЕНТАЛЬНІ МОВИ І СИСТЕМИ ПРОГРАМУВАННЯ

Покоління та класифікація мов програмування

Мови програмування прийнято поділяти на п'ять поколінь. У перше покоління входять мови, які створено на початку 50-х років, коли з'явилися перші комп'ютери. Першою мовою був асемблер**,** створений за принципом «одна інструкція – одна команда».

Друге покоління мов програмування з'явилося у кінці 50-х на початку 60-х років, коли було створено символьний асемблер, у якому з'явилося поняття змінної. Він став першою повноцінною мовою програмування.

Трете покоління мов програмування з'явилося у 60-ті роки. У цей час виникли універсальні мови високого рівня, за допомогою яких стало можливим розв'язання задач у різних областях. Такі якості нових мов, як відносна простота, незалежність від конкретного комп'ютера та можливість використання складних синтаксичних конструкцій, полегшили роботу програмістів.

Починаючи з 70-х років почався період розвитку мов програмування четвертого рівня. Ці мови програмування призначені для реалізації складних проектів та орієнтовані на спеціалізовані області застосування, де позитивний результат досягається лише при використанніпроблемно-орієнтованих мов які оперують конкретними поняттями вузької предметної області.

Народження мов п'ятого покоління припадає на 90-ті роки. До них відносяться також системи автоматичного створення прикладних програм за допомогою візуальних засобів розробки, без володіння мовами програмування.

Інструментальні мови поділяють на мови низького рівня (близькі до машинної мови) та мови високого рівня (близькі до мови людини). До мов високого рівня належать:

FORTRAN - перша компілюєма мова, створена Джимом Беку сем у 50-ті роки. ALGOL - створена у 1960 році для заміну FORTRAN, але через складну структуру не отримав широкого використання.

PASCAL - мову програмування створено у кінці 70-років Ніколасом Віртом.

BASIC - створено у60-тіроки в якості навчальної мови, дуже легкий у вивченні.

C++ (Си++) – об’єктно - орієнтована мова програмування створена у 1980 році Бьярном Страуструпом.

JAVA (Ява) — мову було створено компанією SUN на початку 90-хроків на основі Си++.

Інструментальні середовища програмування. Їх класифікація

Інструментальні середовища програмування містять насамперед текстовий редактор, що дозволяє конструювати програми заданою мовою програмування, інструменти, що дозволяють компілювати або інтерпретувати програми на цій мові, а також тестувати і налагоджувати отримані програми. Крім того, можуть бути й інші інструменти, наприклад, для статичного або динамічного аналізу програм. Взаємодіють ці інструменти між собою через звичайні файли за допомогою стандартних можливостей файлової системи.

Розрізняють наступні класи інструментальних середовищ програмування:

– середовища загального призначення
– мовно-орієнтовані середовища

Інструментальні середовища програмування *загального призначення* містять набір програмних інструментів, що підтримують розробку програм на різних мовах програмування (наприклад, текстовий редактор, редактор зв'язків або інтерпретатор мови цільового комп'ютера) і звичайно являють собою деяке розширення можливостей використовуваної операційної системи. Для програмування в такому середовищі на якій-небудь мові програмування будуть потрібні додаткові інструменти, орієнтовані на цю мову (наприклад, компілятор).

Класифікація інструментальних середовищ програмування:

– *мовно-орієнтоване* інструментальне середовище програмування призначене для підтримки розробки ПЗ на якому-небудь одній мові програмування й знання про цю мову істотно використовувалися при побудові такого середовища. Внаслідок цього в такому середовищі можуть бути доступні досить потужні можливості, що враховують специфіку даної мови. Такі

середовища розділяються на два підкласи: інтерпретуючі середовища, синтаксично – керуємі середовища.

– *інтерпретуюче* інструментальне середовище програмування забезпечує інтерпретацію програм даною мовою програмування, тобто містить насамперед інтерпретатор мови програмування, на який це середовище орієнтоване. Таке середовище необхідне для мов програмування інтерпретуючого типу (таких, як Лисп), але може використовуватися й для інших мов (наприклад, на інструментальному комп'ютері).

– *синтаксично-кероване* інструментальне середовище програмування базується на знанні синтаксису мови програмування, на який вона орієнтована. У такому середовищі замість текстового використовується синтаксично – керований редактор, що дозволяє користувачеві використовувати різні шаблони синтаксичних конструкцій (у результаті цього розроблювальна програма завжди буде синтаксично правильною). Одночасно із програмою такий редактор формує (у пам'яті комп'ютера) її синтаксичне дерево, що може використовуватися іншими інструментами.

Компоненти інструментальних систем

Інструментальні системи служать для розроблення програм. У загальному випадку для створення програми на довільній мові програмування необхідно мати наступні компоненти.

Текстовий редактор. Текст програми створюється за допомогою ключових слів та стандартних символів для запису операцій. Формування тексту можна здійснювати убудь-якомутекстовому редакторі, але краще використовувати спеціалізовані редактори орієнтовані на конкретну мову програмування.

Транслятор. Кожна команда що виконується комп'ютером має вигляд послідовності нулів й одиниць. Писати програми машинною мовою незручно, а їх надійність низька. Тому програми розробляють мовою, зрозумілою людині (інструментальна мова), після чого спеціальною програмою (транслятором) текст програми перекладається на машинний код (транслюється).

Транслятори бувають двох типів: *інтерпретатори і компілятори*. Інтерпретатор читає один оператор програми, аналізує його в контексті вже працюючої програми і потім його виконує, після чого переходить до оброблення наступного оператора. Компілятор спочатку читає, аналізує та перекладає на машинний код усю програму, і тільки після завершення всієї трансляції ця програма виконується.

Оскільки при інтерпретації програма виконується частинами, на комп'ютерах з малою оперативною пам'яттю можна виконати, хоча і повільно, досить великі програми. Компілятори під час аналізу всієї програми оптимізують її. З цієї причини, а також завдяки тому, що програма читається в оперативну пам'ять відразу вся, при компіляції вона виконується швидше, ніж: при інтерпретації.

Бібліотека стандартних програм що поставляються разом з компілятором та реалізує стандартні функцій.

Компонувальник. Виконує зв'язування об'єктних модулів та машинного коду стандартних функцій, знаходячи їх у бібліотеках та формує на виході працездатний додаток.

Налагоджувач. У сучасних інтегрованих системах є компонент, який дозволяє аналізувати роботу програми під час її виконання. З його допомогою можна послідовно виконувати окремі оператори вихідного тексту по крокам, спостерігаючи при цьому, як змінюються значення різних змінних.

Засоби автоматизації програмування (дизайнери, майстри).

Інтегроване середовище автоматично готує всю необхідну послідовність команд, виконує їх, отримує результат, повідомляє про можливі помилки.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: ФАЙЛОВІ СИСТЕМИ

Поняття про файлові системи

Нерідко у користувача комп'ютера виникає необхідність провести форматування носія інформації. Діалогове вікно операційної системи Windows пропонує вибрати файлову систему і спосіб форматування (швидке або глибоке), додатково можна встановити розмір кластера, але цю операцію здійснює не кожен користувач, оскільки немає розуміння, для чого потрібен той чи інший розмір. Деякі можуть задаватися питаннями: що таке файлова система? Для чого вона потрібна? У чому відмінності однієї від іншої?

Файлова система – це сукупність умов і правил, що визначають спосіб організації файлів на носіях інформації. Основні функції файлових систем такі:

– розташування на диску інформації у вигляді файлів,
– присвоєння імен файлів і атрибутів,
– видалення інформації за запитом користувача,
– захист інформації від втрати у випадку збоїв і несанкціонованого доступу до даних.

Запускаються на комп'ютері програми не знають, що таке файлова система і як нею користуватися, в якому місці шукати той чи інший файл. Взаємодія програм і ФС забезпечують драйвери файлових систем. З їх допомогою відбувається зіставлення імен потрібних файлів зі списком зайнятих секторів диска, після чого драйверу диска передається команда на зчитування даних з певних секторів.

*Види файлових систем***.** У процесі розвитку операційних систем і накопичувачів створювалося безліч файлових систем. Сьогодні для роботи з жорсткими дисками і флеш-накопичувачами часто використовуються наступні:

– *FAT32*. Максимальний підтримуваний розмір диска – 8 Тб. Файлова система працює з файлами розміром більше 4 Гб.
– *NTFS*. Максимальний підтримуваний розмір диска – 16 Еб (ексабайт). Підтримує роботу з файлами розміром до 16 Тб.
– *Ext3, ext4*. Використовуються в операційних системах Linux. Максимальний розмір тома становить 32 Тб для ext3 і 1 Еб для ext4. Підтримується робота з файлами розміром до 2 і 16 Тб відповідно.
– *HFS Plus*. Використовується в системах OS X. Максимальний розмір тома – 8 Еб, максимально підтримуваний розмір файлу теж становить 8 Еб.

Для роботи з лазерними дисками використовуються:

– *ISO9660*. Максимальний розмір файлу – 2 Гб.
– *UDF*. Максимальний розмір файлу – 1 Еб.

*Особливості FAT32 і NTFS***.** Найбільш часто використовуваними файловими системами є FAT32 і NTFS з причини широкого розповсюдження операційної системи Windows. Для рядового користувача може бути не важливо, яким чином влаштована і працює та і інша ФС, для нього в контексті побутового використання можуть бути принципові максимальні підтримувані розміри тома і файлу. За цими параметрами FAT32 поступається NTFS, як зазначено вище. Крім того, при використанні FAT32 можлива втрата даних при відключенні подачі електричного живлення на диск (сама файлова система часто залишається цілою).

*Розподіл інформації на диску***.** Дані, записані на носії інформації, що містяться в так звані кластери. Вони являють собою осередки, створені віртуальним об'єднанням секторів жорсткого диска або комірок пам'яті немагнітного носія інформації. Розмір кластера залежить від встановленого при форматуванні значення. Оптимально ставити розмір осередків у відповідності з розмірами найбільш часто використовуваних файлів: чим більший розмір кластера, тим швидше будуть читатися файли великого розміру і навпаки. Особливість запису даних полягає в тому, що в один кластер може бути поміщений тільки один файл або його частину; файл в 1 байт зробить кластер розміром 256 байт недоступним для запису іншої інформації.

Файлову систему слід вибирати відповідно цілям і умовам використання носія. Так, незважаючи на недоліки по відношенню до NTFS файлова система FAT32 може успішно використовуватися в Efi-розділах на сучасних комп'ютерах.

Логічна організація дисків

Сектора

Будь-який жорсткий диск можна представити як величезний «чистий аркуш», на який можна записувати дані і звідки потім їх можна вважати. Щоб орієнтуватися на диску, всі його простір розбивають на невеликі «клітинки» – сектору. Сектор – це мінімальна одиниця зберігання даних на диску, зазвичай його розмір становить 512 байт. Всі сектора на диску нумеруються: кожен з n секторів отримує номер від 0 до n-1. Завдяки цьому будь-яка інформація, записана на диск, отримує точну адресу – номери відповідних секторів. Так що диск ще можна представити як дуже довгу рядок (стрічку) із секторів. Можете порахувати, скільки секторів на вашем диску розміром в N гігабайт.

Розділи

Представляти жорсткий диск як єдиний «лист» не завжди буває зручно: іноді корисно «розрізати» його на кілька незалежних аркушів, на кожному з яких можна писати і стирати що завгодно, не побоюючись пошкодити написане на інших аркушах. Логічно записувати окремо дані більшої і меншої важливості або просто належать до різних речей.

Звичайно, над жорстким диском слід виробляти не фізичне, а логічне розрізання, для цього вводиться поняття розділ (partition). Вся послідовність (дуже довга стрічка) секторів розрізається на декілька частин, кожна частина стає окремим розділом. Фактично, нам не доведеться нічого розрізати (та й навряд чи б це вдалося), досить оголосити, після яких секторів на диску знаходяться межі розділів.

Таблиця розділів

Технічно розбиття диска на розділи організовано наступним чином: заздалегідь певна частина диска відводиться під таблицю розділів, в якій і написано, як розбитий диск. Стандартна таблиця розділів для диска IBM-сумісного комп'ютера – HDPT (Hard Disk Partition Table) – розташовується в кінці самого першого сектора диска, після предзагрузчіка (Master Boot Record, MBR) і складається з чотирьох записів виду «тип початок кінець», по одній на кожен розділ. Початок і кінець – це номери тих секторів диска, де починається і закінчується розділ. За допомогою такої таблиці диск можна поділити на чотири або менше розділів: якщо розділу немає, тип встановлюється в 0.

Однак чотирьох розділів рідко коли буває достатньо. Куди ж поміщати додаткові поля таблиці розбиття? Творці IBM PC запропонували універсальний спосіб: один з чотирьох основних розділів оголошується розширеним (extended partition); він, як правило, є останнім і займає все залишився диска.

Розширений розділ можна розбити на підрозділи тим же способом, що і весь диск: на самому початку – на цей раз не диска, а самого розділу – заводиться таблиця розділів, з записами для чотирьох розділів, які знову можна використовувати, причому один з підрозділів може бути , знову-таки, розширеним, зі своїми підрозділами і т. д.

Розділи, згадані в таблиці розділів диска, прийнято називати основними (primary partition), а всі підрозділи розширених розділів – додатковими (secondary partition). Так що основних розділів може бути не більше чотирьох, а додаткових – скільки завгодно.

Щоб не ускладнювати цю схему, при розмітці диска дотримуються два правила: по-перше, додаткових розділів в таблиці розбиття диска може бути не більше одного, а по-друге, таблиця розбиття розширеного розділу може містити або один запис – опис додаткового розділу, або дві – опис додаткового розділу та опис вкладеного розширеного розділу.

Тип розділу

У таблиці розділів для кожного розділу зазначається тип, який визначає файлову систему, яка міститиметься в цьому розділі. Кожна операційна система розпізнає певні типи і не розпізнає інші, і, відповідно, відмовиться працювати з розділом невідомого типу.

ЛЕКЦІЯ 4. ОСНОВИ ОС MS-DOS

4.1 Модулі операційної системи

Операційна система – це набір програм, які забезпечують роботу обчислювальної системи. Вона здійснює діалог з користувачем, керування комп'ютером, його ресурсами (оперативною пам'яттю, місцем на диску і т. д.), запускає інші програми на виконання.

Основна причина необхідності операційної системи полягає в тому, що елементарні операції для роботи з пристроями комп'ютера і керування ресурсами комп'ютера — це операції дуже низького рівня, тому дії, які необхідні користувачу і прикладним програмам, складаються з декількох сотень чи тисяч таких елементарних операцій.

Наприклад, дисковод «розуміє» лише такі елементарні операції, як включити/виключити двигун, встановити читаючу головку на певний циліндр, прочитати інформацію з певної доріжки в комп'ютер і т. д. навіть для виконання такої нескладної дії, як копіювання файлу з однієї дискети на іншу, необхідно виконати тисячі операцій по запуску команд дисководу, перевірці їх виконання, пошуку і обробці інформації в таблиці розміщення файлів і т.д. Операційна система ховає від користувача всі ці складні і непотрібні деталі і надає йому зручний інтерфейс для роботи.

Операційна система MS-DOS однією з найпопулярніших ОС для IBM-сумісних ПК на основі 80386, чи більш раннього процесора. Вона пред'являє невисокі вимоги до апаратного забезпечення ПК, зокрема:

- процесор 8086, або новіший;
- 512К оперативної пам'яті;
- необов'язковим є ЖМД;
- будь-який тип відеосистеми.

ОС MS-DOS випробувана часом (див. попередній розділ) і тому є досить надійною в роботі. Поряд з цим, основними недоліками MS-DOS, є однозадачність та відсутність графічного інтерфейсу користувача, тобто користувач працює з командним рядком, набираючи всі команди вручну з клавіатури. Ця операційна система є морально застарілою, але, як ми вже говорили, досвідчений користувач повинен знати основи MS-DOS. MS-DOS в своєму розвитку пройшла шість основних етапів (версій).

Часто персональний комп'ютер працює під керуванням операційної системи MS-DOS.

Операційна система MS-DOS складається з наступних частин (модулі):

Базова система введення-виведення (BIOS), яка знаходиться в постійній пам'яті комп'ютера. знаходиться на нульовій стороні, нульовій доріжці, в першому секторі будь-якого диску і займає завжди один сектор (512 байт). Її призначення – виконання найбільш простих і універсальних послуг ОС, зв'язаних зі здійсненням введення-виведення. Базова система введення-виведення містить також тест функціонування комп'ютера, який перевіряє роботу пам'яті і пристроїв комп'ютера при його включенні. Крім того, базова система введення-виведення містить програму виклику завантажувача операційної системи.

Завантажувач операційної системи – це дуже коротка програма, яка знаходиться в першому секторі кожного диска з операційною системою MS-DOS.

Дискові файли IO.SYS і MSDOS.SYS. Вони завантажуються в пам'ять завантажувачем операційної системи і постійно залишаються в пам'яті комп'ютера. Файл *IO.SYS* являє собою доповнення до базової системи введення-виведення. Файл *MSDOS.SYS* реалізує основні високорівнені послуги MS-DOS.

Командний процесор DOS обробляє команди, які вводить користувач. Командний процесор знаходиться в файлі COMMAND.COM на диску, з якого завантажується операційна система. Деякі команди, наприклад TYPE, DIR чи COPY, процесор виконує сам, такі команди називаються *внутрішніми.* Для виконання решти (зовнішніх) команд процесор шукає на диску програму з відповідним іменем, і якщо знаходить її, то завантажує в пам'ять і передає їй керування.

Зовнішні команди MS-DOS – це програми, які додаються до операційної системи у вигляді окремих файлів. Ці програми виконують дії обслуговуючого характеру, наприклад – форматування дискети, перевірка дискети на «погані» сектори і т. д.

Драйвери пристроїв – це спеціальні програми, які доповнюють систему введення-виведення MS-DOS і забезпечують обслуговування нових чи нестандартне використання наявних пристроїв. Наприклад, за допомогою драйверів можлива робота з «електронним диском», тобто частиною пам'яті комп'ютера, з якою можна працювати так само, як і з диском. Драйвери завантажуються в пам'ять комп'ютера при завантаженні операційної системи, їх імена вказуються в спеціальному файлі CONFIG.SYS. Така схема полегшує додання нових пристроїв і дозволяє робити це, не чіпаючи системних файлів MS-DOS.

Для виконання початкового завантаження MS-DOS необхідно, щоб в дисководі А була встановлена дискета з записаною операційною системою або щоб комп'ютер мав вінчестер з записаною на ньому MS-DOS. На початку завантаження працює програма тестування. Після закінчення її роботи програма початкового завантаження читає завантажувач операційної системи. Після цього завантажувач зчитує в пам'ять комп'ютера модулі операційної системи (файли *IO.SYS, MSDOS.SYS*) і передає їм керування. Далі читається файл конфігурації системи CONFIG.SYS і у відповідності з вказівками, які містяться в цьому файлі, завантажуються драйвери пристроїв і встановлюються параметри операційної системи. Після цього з диска, з якого завантажується операційна система, читається командний процесор (файл COMMAND.COM) і йому передається керування. Командний процесор виконує командний файл AUTOEXEC.BAT. В цьому файлі вказують команди і програми, які необхідно виконувати при кожному вмиканні комп'ютера. Після його виконання процес завантаження операційної системи закінчується. MS-DOS видає запрошення, що означає її готовність до роботи. В запрошенні, як правило, міститься інформація про поточний дисковод і символ «>».

4.2 Внутрішні команди

Всі команди, що використовуються в М5-005 поділяють на дві групи:

– внутрішні команди;
– зовнішні команди.

Внутрішні команди записані в командний процесор (файл COMMAND.COM). Оскільки, цей файл при завантаженні ПК автоматично завантажується в оперативну пам'ять і постійно знаходиться там, то внутрішні команди виконуються завжди, за винятком тих випадків, коли синтаксис команд є однозначно неправильним.

Зовнішні команди, на відміну від внутрішніх, записані в окремих файлах, імена яких співпадають з назвою команд. Тобто для їх виконання необхідна наявність однойменних файлів на диску. Ці команди розширюють можливості ОС, виконуючи додаткові та сервісні функції.

Варто зауважити, що для того, щоб дана зовнішня команда виконувалась, потрібно, щоб відповідний їй файл не просто знаходився на ПК, а був присутній в активному каталозі, або в кореневому каталозі системного диску, або щоб шлях до нього був вказаний у спеціальній команді файлу autoexec.dat

Нагадаємо, що команди складаються з імені команди і, хоч це необов'язково, параметрів, розділених пробілами. Ім'я команди і параметри можна вводити як великими, так і малими латинськими буквами. Будемо відмічати дужками [ ] необов'язкові елементи команд.

КОМАНДА ЗМІНИ ДИСКОВОДУ

Для зміни дисководу потрібно набрати ім'я дисководу, з яким хочемо працювати, і потім двокрапку. Наприклад:

A: – перехід на дисковод А;
B: – перехід на дисковод В;
C: – перехід на дисковод С.

Після введення команди потрібно натиснути клавішу ENTER.

КОМАНДА ЗМІНИ КАТАЛОГУ

Для зміни каталогу існує команда CD. Формат команди:

cd *[дисковод:] шлях*

Якщо задано дисковод, то каталог зміниться на цьому дисководі, в іншому випадку зміна відбудеться на тому, з яким ми працювали.

Приклади:

cd\ – перехід в кореневий каталог даного диска;
cd\exe\dos — перехід в каталог \exe\dos\;
cd.. – перехід в надкаталог.

ПЕРЕГЛЯД КАТАЛОГУ

Для виведення змісту каталогу існує команда DIR. Формат команди:
DIR *[дисковод:] [шлях] [ім 'я каталогу]*

В імені файла можна використовувати символи * і ?. Символ * позначає будь-яку кількість будь-яких символів в імені файла чи в розширенні імені файла. Символ ? позначає один будь-який символ або відсутність символу в імені файла чи в розширенні імені файла.

Якщо в команді не вказано дисковод чи шлях, то це означає поточний дисковод і поточний каталог.

Для кожного файла команда видає його основне ім'я, розширення імені, розмір файла в байтах, дату і час створення чи останнього поновлення файла. Підкаталоги помічаються <dir>. В кінці видачі повідомляється про розмір вільного місця на диску.

СТВОРЕННЯ КАТАЛОГУ

Для створення нового каталогу існує команда md (Make direktory). Формат команди:
md *[дисковод:] шлях*
Приклади:
Md OLK – створення підкаталогу в поточному каталозі;
Md a:\bgt – створення підкаталогу в кореневому каталозі диска А.

ЗНИЩЕННЯ КАТАЛОГУ

Для знищення (порожнього) каталогу існує команда RD. Формат команди:
RD *[дисковод:] шлях*
Приклади:
RD OLK – знищення підкаталогу в поточному каталозі;
RD a:\BGT – знищення підкаталогу в кореневому каталозі диска А.

Відмітимо, що знищити можна лише порожній каталог, тобто каталог, в якому немає ні файлів, ні підкаталогів.

РОБОТА З ФАЙЛАМИ

Файли можна поділити на дві категорії – текстові і двійкові. Текстові файли призначені для читання людиною. Вони складаються з рядків символів, причому кожний рядок закінчується двома спеціальними символами «повернення каретки» і «новий рядок». При редагуванні і перегляді текстових файлів цих спеціальних символів, як правило, не видно. В текстових файлах зберігаються різні документи, тексти програм, командних файлів 008 і т. д. Файли, які не є текстовими, називаються двійковими. Текстовий файл, який містить лише символи з кодами до 127 (тобто без українських букв і символів псевдографіки), називається ASCII-файлом.

СТВОРЕННЯ ТЕКСТОВИХ ФАЙЛІВ

Щоб створити текстовий файл, найкраще скористатися будь-яким редактором, який може працювати з текстовими файлами

Невеликі текстові файли можна набрати і без редакторів. Для цього слід ввести наступну команду:
copy con *ім 'я файла*

Після введення цієї команди потрібно рядок за рядком вводити текст. В кінці кожного рядка потрібно натискати клавішу Enter, а після введення останнього – натиснути клавішу F6, що означатиме, що це вже кінець файла, і потім – Enter. І лише після цього буде створено текстовий файл.

КОПІЮВАННЯ ФАЙЛІВ

Для копіювання файлів є команда Copy. Формат команди:
copy *[дисковод:] [шлях\] ім 'я файла [дисковод:] [шлях\] [ім 'я файла]*
В іменах файлів можна використовувати символи * і ?.

З каталогу, вказаного в першому параметрі команди, копіюються файли, задані іменем файла в цьому параметрі команди. Дисковод і шлях в другому параметрі команди вказують каталог, в який

копіюються файли. Якщо в другому параметрі ім'я файла відсутнє, то імена файлів при копіюванні не змінюються. Якщо в другому параметрі команди задано ім'я файла, то воно вказує нове ім'я файла, в який копіюється інформація. Символи * і ? в імені файла в другому параметрі команди вказують, що відповідні символи в іменах файлів, що копіюються, при копіюванні не змінюються.

Приклади:

copy a:*.* – копіювання всіх файлів з кореневого каталогу дисководу А в поточний каталог;

ПЕРЕЙМЕНУВАННЯ ФАЙЛІВ

Для перейменування файлів існує команда REN (RENAME). Формат команди:

ren *[дисковод:] [шлях\] ім 'я файла ім 'я файла*

Перше ім'я файла в команді задає ім'я (імена) файлів, що перейменовуються, друге – нове ім'я (імена) файлів.

Параметри дисковод і шлях вказують, в якому каталозі перейменовуються файли. Якщо дисковод і шлях не вказані, то перейменування відбувається в поточному дисководі і каталозі.

В іменах файлів можна використовувати символи * і ?.

ЗНИЩЕННЯ ФАЙЛІВ

Для знищення файлів існує команда DEL (Delete). Формат команди:

del *[дисковод:] [шлях\] ім 'я файла*

В імені файла можна використовувати символи * і ?.

ВИВЕДЕННЯ ФАЙЛА НА ЕКРАН

Для виведення текстового файла на екран можна скористатися командою Type. Формат команди:

type *ім 'я файла*

4.3 Зовнішні команди

Зовнішні команди реалізовані в exe- чи corn-файлах, які є на диску в деякому каталозі» найчастіше з назвою DOS. Розглянемо найважливіші з них:

msd – вивести інформацію про комп'ютер;
edit <назва> – редагувати файл з даною назвою;
scandisk a: – перевірити (сканувати) диск а: і усунути :
defrag a: – оптимізувати (усунути фрагментацію) диска ас;
format a: – форматувати дискету а:;
deltree <назва> – вилучити не порожній каталог;
xcopy <назва1> <назва2> – копіювати каталог;
mem /c /p – показати розподіл оперативної пам'яті;
memmaker – оптимізувати розподіл оперативної пам'ятні;
backup – копіювати твердий диск на дискети;
restore – перенести файли з дискет на диск;
sys a: – скопіювати операційну систему на дискету а:;
undelete – відновити вилучені файли.

ЛЕКЦІЯ 5. ОСНОВНІ ХАРАКТЕРИСТИКИ ОС WINDOWS. СТАНДАРТНІ ДОДАТКИ ТА УТИЛІТИ

5.1 Загальні відомості операційної системи Windows

З кінця 90-х рр XX ст. стандартом ОС для ПК стала система WINDOWS. Вона має ряд особливостей:

1. Зручний для користувача графічний інтерфейс. Він дає змогу досить просто керувати роботою комп'ютера, використовуючи такі поняття, як «Мій комп´ютер», «Кнопка «Пуск»», «Мережене оточення», «Ярлик», «Контекстне меню» і т. ін.
2. Вона є об'єктно-орієнтованою ОС для оброблення документів. В ній використовуються такі офісні аналогії, як «робочій стіл», «папка», «документ», «кошик».
3. Допускає підключення до локальних і глобальних комп´ютерних мереж (електронна пошта, факс, Internet).
4. Містить довгі імена файлів (до 255 символів).
5. Допускає роботу з програмами, розробленими в інших ОС (MS DOS, Windows 3.1, Windows 95, Windows NT), і їх виконання.
6. Працює в багатозадачному режимі. Використовує процесну форму (паралельно виконує кілька програм) та потокову форму (паралельно виконуються різні частини однієї програми).
7. Застосовує пряму адресацію ОП, завдяки чому у програмах (додатках) можуть використовуватися до 4ГБ віртуальної пам'яті (ОП і пам´ять на ЖД).
8. Підтримує обмін даними між додатками за допомогою OLE-технології зв´язування та вбудування об´єктів. Наприклад, таблиці, а також діаграми, побудовані в табличному процесорі Excel, можуть використовуватися в документі, створеному в текстовому редакторі Word).
9. Містить ряд стандартних програм (Блокнот, графічний редактор Paint, калькулятор).

5.2 Завантаження Windows

Щоб розпочати роботу з комп'ютером, на якому встановлена ОС Windows, досить увімкнути живлення. Невдовзі на екрані з'являється заставка, а потім – *робочий стіл* Windows.

Зовнішній вигляд екрана може змінюватися, однак у будь-якому разі на робочому столі можна побачити кілька піктограм, які відображають різні об'єкти Windows. У нижній частині екрана розташовується панель задач, у лівій частині якої знаходиться кнопка «Пуск». Вона використовується для відкриття головного меню, за допомогою якого можна здійснювати запуск програм, пошук файлів, входити до довідкової системи тощо.

Відразу після запуску системи ми бачимо на екрані курсор мишки, який здебільшого має форму нахиленої вліво стрілки. Курсор мишки може змінювати свою форму залежно

від того, на який об'єкт або частину об'єкта він вказує. Форма курсора говорить про те, що саме в даний момент можна зробити з об'єктом.

За термінологією, прийнятою у Windows, файлові каталоги називаються *папками.* Для забезпечення зручного ефективного доступу до ресурсів комп'ютера через стандартний графічний інтерфейс поняття папки було розширене: спеціальні системні папки, такі як «Мій комп´ютер» чи «Мережене оточення», є контейнерами, що містять папки таких системних ресурсів, як диски, дисководи, принтери, під'єднанні до мережі комп'ютера тощо.

5.3 Вікна у системі Windows

При запуску програми чи відкритті папки на робочому столі з'являється вікно, а на панелі задач – відповідна кнопка. Вікно, в якому працює користувач, називається активним. Воно знаходиться на передньому плані. Щоб перейти до якогось вікна, слід підвести курсор до видимої частини потрібного вікна або його кнопки на панелі задач і клацнути лівою кнопкою миші.

Вікна у Windows бувають чотирьох типів:

1. *Вікна папок* вміщують значки (піктограми) інших об'єктів Windows і елементи управління вікном.
2. *Вікна прикладних програм* вміщують робочу інформацію, із якої працюють ці програми, а також елементи управління вікном
3. *Діалогові вікна* вміщують тільки елементи управління

4. *Вікна довідкової системи* вміщують допоміжну довідкову інформацію для роботи з операційною системою та прикладними програмами, а також елементи управління довідковою системою

Усі типи вікон мають подібну структуру. Заголовок, що знаходиться вгорі вікна, містить його назву, а також кілька кнопок управління в правому кінці рядка заголовку. Вікна програм та папок мають три кнопки (згортання вікна, розгортання–відновлення вікна, закриття вікна). Безпосередньо під заголовком знаходиться рядок меню. Якщо розмір вікна недостатній для розміщення всього зображення, справа та внизу з'являються смужки прокручування. Розміри та положення вікна можна змінювати, «перетягуючи» мишкою межі рамки та «буксируючи» його в потрібне місце. Змінювати параметри вікна можна також за допомогою меню, яке з´являється при встановленні курсора в будь-якому місці заголовка і клацанні правою кнопкою миші (контекстні меню.)

У рядку стану, що розміщується у нижній частині вікна, виводиться інформація про виділений об'єкт або пункт меню.

Рамка визначає межі вікна і використовується для зміни його розмірів.

5.4 Провідник і Мій комп'ютер

Файлова система ОС Windows має деревоподібну ієрархічну структуру. Під час переміщення, наприклад, файла з папки, розташованої на диску, в іншу, розміщену на іншому диску, необхідно послідовно відкрити папки на першому диску, щоб досягти вихідної папки, а потім – на другому, щоб на екрані з´явилася цільова папка. Тому при виконанні операції з інформаційними об´єктами (папками, файлами та ярликами) бажано мати швидкий доступ до цих об´єктів з урахуванням ієрархічності їх розміщення у файловій системі.

Такий наочний перехід з однієї гілки дерева файлової системи на іншу реалізовано у програмі «Провідник», призначеної для спрощення виконання операцій з інформаційними об´єктами (створення папок, ярликів, переміщення, перейменування, знищення), що розміщуються як на даному ПК, так і на інших комп´ютерах, підключених до локальної мережі.

Виклик програми:

– У меню Пуск, Програми, Провідник.

– Правою клавішею мишки на одній з папок «Мій комп´ютер» , або по кнопці Пуск. Потім з контекстного меню вибрати Провідник.

– При натиснуті клавіши Shift двічі клацнути мишею на значку будь-якої папки («Мій комп´ютер»)

Вікно програми «Провідник» складається з двох частин: лівої (панель ресурсів комп´ютера) і правої (панель вмісту вибраної папки).

На найвищому рівні у панелі ресурсів розміщується робочий стіл, оскільки з нього є доступ до решти ресурсів: «Мій комп´ютер», «Мережне оточення», «Кошик». Таким чином, ці три компоненти знаходяться на одному рівні ієрархії.

Якщо з нижньої частини значка якогось об´єкта виходить вертикальна штрихована лінія, то всі значки, з´єднанні з нею горизонтальними лініями, є елементами папки «Мій комп'ютер».

Ліворуч від значка об´єкта у прямокутній рамці може бути знак «+» або «-«. При клацанні мишею на позначці «+» згортається вузол, показуючи, які гілки виходять із нього (наприклад, які папки є на диску С:). При цьому знак «+» замінюється знаком «-». При клацанні мишею на позначці «-» відбувається зворотній процес – вузол згортається (гілки вилучаються).

Якщо розгорнуто багато вузлів, то для перегляду довгої деревоподібної структури можна скористатися вертикальною лінійкою прокручування.

Переміщення по дереву у панелі ресурсів не змінює вміст правої панелі, що є звичайним вікном папки. Для зміни вмісту тут досить клацнути мишею на значку папки, яка цікавить, у панелі ресурсів.

Таке переміщення по дереву для вибору об´єктів відбувається швидше, ніж послідовне відкривання вкладених папок.

Мій комп'ютер - програма, яка використовується для роботи з файлами і папками, що зберігаються на дисках комп'ютера. Моє мережне оточення – програма, яка використовується для роботи з мережевими ресурсами в робочій групі.

5.5 Стандартні додатки та утиліти ОС

Набір додатків, від версії до версії Windows зазнає ряд істотних змін, але група додатків, об'єднаних в папку «Стандартні» практично незмінна, і лише доповнюється новими.

До складу стандартних додатків Windows включено безліч додатків:

– додатки, дозволяють відтворювати звук, анімацію і відео через програвачі компакт-дисків або звукові адаптери;

– сервісні прикладні програми для зв'язку з іншими комп'ютерами та інтерактивними службами;

– службові прикладні програми, що дозволяють обслуговувати диски, виробляти архівацію і зберігати дані і файли, відображати вміст буфера проміжного зберігання, виводити інформацію щодо системних ресурсів, проводити спостереження за сервером мережі та іншими мережевими підключеннями, проводити системні політики на основі мережевих груп;

– збільшити обсяг дискового простору на жорстких дисках великої місткості, стиснути дані на дисках, спостерігати за продуктивністю системи, вставляти в документи різні символи;

– спеціальні прикладні програми з набором стандартних і нових засобів для людей з обмеженими можливостями;

– набір програм для роботи з Internet;

– стандартні програми і додаткові компоненти системи (Paint, WordPad, Швидкий перегляд, Ігри, Калькулятор, Програми-заставки, Перегляд малюнків, Сервіс сценаріїв, Покажчики миші, Фонові малюнки, Шаблони документів і Теми робочого столу)

Індикатор системних ресурсів

Для виклику необхідно натиснути кнопку Пуск, далі до папки Службові і запустити програму Індикатор ресурсів. За допомогою цієї програми ви можете оцінити системні ресурси вашого комп'ютера. Наприклад, ви зможете дізнатися скільки оперативної пам'яті потрібно того чи іншого додатка для роботи в Windows.

Перетворення диска в FAT32

Якщо ви хочете більше ефективно використовувати простір жорсткого диска і не ділити його на логічні диски, необхідно викликати меню Пуск, далі вибрати папку Службові і запустіть утиліту Перетворення диска в FAT32. За допомогою цієї утиліти ви можете здійснити перехід з FAT16 в систему FAT32 без переформатування диска. Перш, ніж ви перетворюєте ваш диск в FAT32, переконайтеся в тому, що нова дискова система вам дійсно потрібна. Мінімальний розмір кластера в FAT16 для одного гігабайтного диска є 32 кілобайти, а в FAT32 він становить всього 4 кілобайти. Звичайно, це вигідно, тим більше, що FAT32 не обмежується 2-гігабайтним лімітом дискового простору. З іншого боку, в FAT32 може некоректно працювати, або ж зовсім не працювати з DOS-додатками.

Якщо ви вже перетворили вашу систему в FAT32, то єдиний спосіб повернути все на місце - переформатування диска. У будь-якому випадку, як при форматуванні, так і при перетворенні системи FAT, ви втрачаєте дані.

Дефрагментація дисків

Дефрагментація – це правильне упорядкування файлів і ланцюжків файлів, для оптимізації роботи комп'ютера. Дефрагментація дозволяє більш швидко отримувати доступ до файлів, розміщених на диску. Корисно проводити дефрагментацію раз у 3-6 місяців, можна і частіше, але не рідше.

Сам процес дефрагментації почнеться після того, як користувач натисне кнопку з аналогічною назвою. Також можна оцінити наскільки вихідна інформація вже дефрагментувати. Для цього необхідно вибрати пункт Аналіз. Після нетривалої паузи буде виведена діаграма.

Архівація даних

Часом виникає ситуація коли розмір файлу занадто великий і його необхідно зменшити, наприклад – файл не поміщається на дискету. На допомогу в даній ситуації приходять програми -

архіватори, які дозволяють стиснути дані, використовуючи спеціальні алгоритми стиснення.

Різноманітність архіваторів велике, але, найбільш зручним, і дає найкращі результати стиснення є архіватор WinRar. Програма підтримує окрім свого формату rar, також і інші формати: Zip, tar, gz, arj та інші.

Існує можливість створити архів, що саморозпаковується, з розширенням. exe, для розпакування якого достатньо лише просто його запустити на виконання. Також є можливість створити багатотомний архів, для цього необхідно задати розмір вихідного файлу, і програма автоматично розділить архів на томи.

Найкраще упаковуються текстові файли і документи.

Після додавання файлів в архів на виході формується архівний файл. Зовні він нічим не відрізняється від інших файлів, але атрибути у даного файлу «архівний».

Необхідно викликати меню Пуск, далі папка Службові і запустити утиліту Архівація. За допомогою цієї утиліти ви можете резервувати файли жорсткого диска, а також складати завдання на автоматичне резервування таких файлів. За допомогою утиліти Архівація даних ви можете також відновити останню повну копію системи та файли реєстру.

Таблиця символів

Необхідно викликати меню Пуск, далі папка Службові і запустити програму Таблиця символів. Відкрита Таблиця символів дозволить переглянути весь набір символів того чи іншого встановленого шрифту і, в разі необхідності, ввести цей символ в оформлюваний документ.

Калькулятор

Якщо ви хочете використовувати у своїй роботі калькулятор, то необхідно викликати меню Пуск, потім вибрати папку Стандартні і запустити програму Калькулятор. Є 2 режими роботи калькулятора:

– звичайний – служить для простих арифметичних розрахунків

– інженерний – призначений для обчислення складних арифметичних та тригонометричних функцій.

Графічний редактор Paint

Якщо ви хочете щось намалювати, необхідно викликати меню Пуск, потім вибрати папку Стандартні та запустіть графічний редактор Paint. Графічний редактор дозволяє створювати зображення використовуючи різний інструментарій:

– пензлик;

– розпилювач;

– олівець і т.д.

Текстовий редактор WordPad

Якщо ви хочете створити простий текстовий документ, без використання таблиць, і інших складних елементів текстових редакторів, то можна скористатися текстовим редактором WordPad. Необхідно вибрати меню Пуск, далі папку Стандартні і запустити WordPad.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: ОБМІН ДАНИМИ МІЖ WINDOWS – ПРОГРАМАМИ ПРИ ОФОРМЛЕННІ ЗВІТНОЇ ДОКУМЕНТАЦІЇ ТЕХНІКА–ЕЛЕКТРОМЕХАНІКА

Тенденція розвитку прикладного ПЗ загального призначення

Варто зазначити, що для прикладних програм важливим є спрощення роботи користувача, виконання операцій по опрацюванню інформації раціонально із мінімальним затратами. В зв'язку з цим в електронних засобах формування результуючих документів опрацювання результатів різних видів інформації постає питання визначення поняття документа. Документ в прикладній програмі з точки зору ОС Windows є сховище специфічних об'єктів, де об'єктом може бути все від простого текстового файлу до відео ролика чи анімації. Крім того створення і збереження в документ зв'язків між об'єктами передбачає принцип робити роботу тільки один раз. Крім того процес створення і подання інформації перетворюється у вибір найкращого об'єкта для формування того чи іншого документа.

Використання модульних архітектур в розвитку ПЗ спрямоване на полегшення процесу перенесення об'єктів із однієї прикладної програми в іншу. Це поклало основу для створення інтегрованих пакетів, що призначені для опрацювання різних видів інформації. Для вирішення проблеми перенесення даних із одних прикладних програм в інші були реалізовані технології обміну даними між цими програмами.

1. В перших версіях Windows було реалізовано вбудований буфер проміжного збереження даних, який постійно залишався активним і доступним всім Windows-програмам.

Clipboard – ця технологія мала ряд недоліків:

– Неможливість накопичення даних в буфері
– Обмеження обсягу інформації
– Вставлені в документ-приймач через буфер обміну дані не поновлювались при їх зміні в документі-джерелі

Проте перевагою цієї технології стало можливість поєднання в одному документі об'єктів створених різними програмами. Це забезпечувалося тим, що для всіх Windows-програм встановлено ряд стандартних форматів, в яких можуть подаватися дані і при операції буфером обміну перетворених даних відбувається автоматично і не помітно для користувача.

2. DDE – динамічний обмін даними, яка починаючи з Windows 3.0 визначена була як стандарт на організацію взаємодії між різними програмами. Суть в тому, що в сеансі зв'язку одна програма названа клієнтом посилає запити у вигляді роману, а друга – сервер – у відповідь посилає дані. При цьому зберігається зв'язок вставлення об'єкта з оригіналу. Ця технологія не знайшла широкого розповсюдження, оскільки навіть враховуючи переваги DDE була складною у функціонуванні і користувачі продовжували використовувати обмін даними через буфер.

3. OLE – технологія зв'язування втілення об'єктів. Вперше була реалізована для Windows 3.1. Основною новизною її стала можливість активізації вбудованих об'єктів, зокрема при обміні даними в попередніх технологіях для вияснення змін у вбудований об'єкт потрібно було запускати програму, в якому створювався цей об'єкт, відкривати відповідний файл, змінювати і зберігати. Потім через буфер обміну виправлений об'єкт знову копіювався в потрібне місце, але при цьому необхідно було знищити стару версію об'єкта в документі. Технологія DDE передбачає динамічну зв'язування, але процес був складним. В технології OLE використовувалась та ж послідовність дій. Проте для вставленого об'єкта засіб активізації об'єктів Windows в полі об'єкта активізував цей об'єкт. Після завершення такої роботи ця програма закривається, а змінений об'єкт залишається в основному документі. Ця технологія визначила новий зміст поняття об'єкта. Зокрема це поєднання даних будь-якого вигляду у внутрішньому форматі програмного створювача поданого в одному з стандартних форматів Windows, а також інформацію про програму що створила цей об'єкт, його розмір, час створення. Особливістю цієї технології став метод *drag-and-drop*, який використовується замість копіювання через буфер обміну.

Технології OLE і DCOM

Зараз широко застосовуються спеціалізовані програми, що надають користувачу великі можливості при складанні текстових документів, при роботі з аудіо- і відеофайлами. Документи, створені в різних програмах, мають різний формат. Вони можуть обмінюватися даними один з одним, використовуючи технологію зв'язування і впровадження об'єктів (OLE). Цю технологію підтримують усі програми Windows.

По термінології OLE будь-які дані (текст, малюнок і ін.), що переносяться з документа, створеного за допомогою однієї програми, у документ, створений в іншій програмі, називаються об'єктом. Об'єктом може вважатися весь документ, окремий його фрагмент чи символ.

Зв'язаним об'єктом називається об'єкт (дані), створений в одному файлі і вставлений в інший файл із підтримкою зв'язку між файлами. Файл, у якому знаходиться вихідний об'єкт, і додаток, у якому він створений, є відповідно *файлом-джерелом* (вихідним файлом) і програмою-джерелом. Файл, що містить вставлений об'єкт, називається *складеним документом* (кінцевим файлом). У складеному документі зберігається інформація про програму, у якій був створений об'єкт. Щоб не порушити зв'язок, документ-джерело не можна переміщати, видаляти чи змінювати ім'я файлу.

Зв'язані об'єкти використовуються у випадку, якщо необхідно, щоб дані в кінцевому файлі обновлялися при зміні даних у вихідному файлі.

При встановленні зв'язку між об'єктами дані фізично продовжують знаходитися в програмі, де вони створювалися (у документі-сервері). Зв'язаний об'єкт не є частиною файлу, у який він уставлений. Недолік операції зв'язування виявляється при переносі файлу документа-клієнта на інший комп'ютер: порушується його зв'язок з документами-серверами.

Якщо не встановити прапорець *Зв'язок* у вікні *Вставка об'єкта*, то вставлені в документ дані, створені в іншій програмі, будуть впроваджені. Розмір документа при цьому збільшиться. Інформація про те, звідки були узяті дані, не збережеться, зв'язок між документами встановлена не буде. У такому випадку зміна даних у вихідному документі не стане відбиватися на інших документах. Подібні документи можна переносити на інші комп'ютери, не піклуючись про файли, з яких були узяті дані.

OLE є частиною моделей COM (Common Object Model – єдина об'єктна модель) і DCOM (Distributed COM). Остання дозволяє програмам, розподіленим по різних вузлах мережі, обмінюватися даними на основі OLE.

Таблиця символів дозволяє вставляти в текст документа спеціальні і математичні символи, різні знаки, не представлені на клавіатурі. Вона відображає набори символів Windows, DOS і Unicode .

Щоб відкрити таблицю символів, натисніть кнопку *Пуск*, виберіть *Програми, Стандартні*, *Службові*, а потім команду *Таблиця символів*.

Клік мишею символу дозволяє побачити його в збільшеному масштабі. Утримуючи натиснутої ліву кнопку миші і переміщаючи покажчик, можна переглянути всі символи в збільшеному масштабі.

Для вибору символу з клавіатури натисніть кілька разів клавішу *Tab*, поки курсор не виявиться в області розташування символів, і клавішами зі стрілками виберіть потрібний символ.

Таблицю можна згорнути до розмірів кнопки на панелі задач, перемістити по екрану, але не можна змінити її розміри й одержати її зображення у весь екран. У нижній області вікна знаходиться рядок стану.

Набір символів міняється в залежності від обраної в списку *Шрифт*, що розкривається, назви, який розташовано в лівому верхньому куті вікна програми.

Шрифт Symbol містить букви грецького алфавіту, математичні, хімічні і логічні символи. Ряд надрядкових і підрядкових символів з нижнього ряду таблиці можна об'єднати для одержання нового знака, наприклад інтеграла.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: МЕТОДИ АРХІВУВАННЯ

Поняття технології стиснення даних

Розробка алгоритмів стиснення інформації відноситься до однієї з галузей прикладної математики. В основі цих алгоритмів лежить принцип усунення природної надмірності.

Стиснення даних – це процедура перекодування даних, яка проводиться з метою зменшення їхнього обсягу, розміру, об'єму. Стиснення базується на усуненні надлишку інформації, яка міститься у вихідних даних. Стиснення даних, які не мають властивості надлишку (наприклад випадковий сигнал чи шум), неможливе. Також неможливо стиснути зашифровану інформацію.

Методи стиснення даних діляться на два класи: *стиснення без втрати інформації* (коли можливо відновлення вихідних даних без спотворень) і *стиснення з втратою інформації* (відновлення можливе з незначними спотвореннями).

Стиснення без втрат – метод стиснення даних, при використанні якого закодована інформація може бути відновлена з точністю до біта. Для кожного з типів цифрової інформації, як правило, існують свої алгоритми стиску без втрат. Стиснення без втрат використовується при обробці та збереженні комп'ютерних програм і даних.

Стиснення з втратами інформації – метод стиснення даних, при якому розпакований файл відрізняється від оригіналу, проте може бути корисним для використання. Стиснення із втратами найчастіше використовується для мультимедіа даних (аудіо, відео, зображення), особливо для потокової передачі даних та і телефонії. В цьому контексті такі методи часто називаються *кодеками*.

Перевага методів стиснення із втратами над методами стиску без втрат полягає в тому, що перші істотно перевершують по ступені стиску, продовжуючи задовольняти поставленим вимогам. Методи стиску із втратами часто використаються для стиску звуку або зображень. У таких випадках розпакований файл може дуже сильно відрізнятися від оригіналу на рівні порівняння «біт у біт», але практично не відрізняється для людського вуха або ока в більшості практичних застосувань.

Багато методів фокусуються на особливостях будови органів почуттів людини. Психоакустична модель визначає те, наскільки сильно звук може бути стиснений без погіршення сприйманої якості звуку. Помітні для людського вуха або ока недоліки, що виникли через стиснення із втратами, відомі як артефакти стиску.

До алгоритмів стиснення з втратою інформації відносяться такі алгоритми як JPEG (використовуються при стисненні фотозображень) і MPEG (використовуються при стисненні відео і аудіо).

Величиною допустимої втрати при стисненні зазвичай можна управляти, що дозволяє досягти оптимального співвідношення «розмір/якість». На фотографічних ілюстраціях, призначених для відтворення на екрані, втрата 5% інформації зазвичай некритична, а в деяких випадках можна допустити втрату і в 20-25%.

Методи стиснення без втрати інформації застосовуються при роботі з текстовими документами і програмами і не можуть допустити втрату інформації. Вони засновані тільки на усуненні її надлишку.

Програми, що виконують стиснення інформації, можуть вводити своє кодування (різну для різних файлів) і приписувати до стислого файлу деяку таблицю (словник), з якої програма, що розпаковує, дізнається, як в цьому файлі закодовані ті або інші символи або їх групи.

Наявність фрагментів, що повторюються, – третя основа для надлишку. У текстах це зустрічається рідко, але в таблицях і в графіці повторення кодів — звичайне явище. Так, наприклад, якщо число 0 повторюється 20 разів підряд, то немає сенсу ставити 20 нульових байтів. Замість них ставлять один 0 і коефіцієнт 20. Такі алгоритми, що засновані на виявленні повторів, називають методами кодування довжин серій (*Run Length Encoding*, RLE). Цей підхід досить ефективний для графічних зображень у форматі «байт на піксел» (наприклад, формати PCX або BMP).

При створенні резервних копій на жорстких дисках є ще одна можливість отримання виграшу в робочому просторі при стисненні файлів, яка пов'язана не з надлишком інформації, а з тим, як організована файлова система комп'ютера. Суть її полягає в тому, що будь-який файл, великий або маленький, може займати на диску тільки ціле число кластерів. У файловій системі FAT16 на жорсткому диску не може бути більше 65536 кластерів (2^{16}). А це означає, що для дисків розміром від 1 до 2 Гбайт, розмір кластера становить 32 Кбайт.

При ущільненні великої групи файлів в один файл економія складає мінімум по 16 Кбайт на кожному файлі тільки за рахунок скорочення втрат від нераціональної організації файлової системи.

Для FAT32 виграш виявляється менше, але і в цьому випадку мінімальний розмір кластера дорівнює 4 Кбайт.

Попри те, що існує немало різних методів стиснення, є деякі принципи і правила, які є загальними для усіх методів стиснення. Їх потрібно знати і правильно використовувати.

1. У всякого стиснення є межа, тобто ущільнення раніше ущільненого файлу у кращому разі не дає виграшу, а у гіршому разі може привести і до програшу у розмірі результуючого файлу.

2. Для всякого методу стиснення можна підібрати файл, стосовно якого цей метод є найкращим.

3. Програми-пакувальники до початку роботи повинні виконувати попередній перегляд оброблюваних файлів і вибирати той метод упаковки, який в даному випадку дає найкращий результат.

Типи архіваторів

Стискуватися можуть як один, так і декілька файлів, які в стислому виді поміщаються в так званий *архівний файл* або *архів*.

Архів — це файл, що містить у собі один або декілька файлів та метадані. Файли можуть бути як стиснені (без втрат), так і мати початковий розмір та структуру. Метадані можуть містити інформацію про початковий розмір файлів, інформацію про формат файлів, структуру директорій, коментарі до файлів, інформацію для відновлення архіву і т.д.

Архіви файлів створюються за допомогою спеціалізованих програм — архіваторів, які можуть бути як окремими програмами, так і частиною інших програм.

Архіватор – програмне забезпечення, що використовується для стиснення інформації.

Початковий файл – файл, що піддається стисненню.

Пакувальник – це програма, що перетворює масив символів в деякому алфавіті в інший, бажано меншого розміру. Часто в ролі цього масиву виступає безструктурний двійковий файл, а в ролі символів вхідного алфавіту – 256 можливих значень байта.

Розпакувальник – програма, що здійснює зворотне однозначне перетворення, тобто розпаковує архів.

Багатотомний архів – архів, розбитий на кілька частин (томів). Багатотомні архіви широко використовуються при завантаженні файлів з Інтернету.

Стиснення з можливістю завдання обсягу томів має ряд переваг:

– дозволяє завантажити файл по частинах; у випадку пошкодження одного, або кількох томів, проблема може бути усунена без повторного завантаження всіх томів;

– дозволяє стискати дані за розміром тому, для сумісності зі знімними накопичувачами, наприклад, розбиття архіву на частини по 700 Мб полегшить запис на CD.

Недоліком багатотомних архівів є те, що при пошкодженні одного з томів, неможливо відкрити повний архів.

Безперервний архів – архів, упакований таким чином, що всі стиснені файли розглядаються як один безперервний потік даних. При упаковці кожного файлу (крім першого) використовується інформація, що міститься в попередніх файлах.

До переваг безперервного архіву слід віднести потенційне збільшення ступеня стиснення. При цьому чим менший середній розмір файлів, більше самих файлів і більше схожих один на одного файлів, тим більший рівень стиснення.

Недоліки безперервного архіву:

– зміна безперервного архіву (тобто додавання або вилучення з нього файлів) відбувається повільніше, ніж звичайного;

– витяг окремого файлу з середини або кінця архіву відбувається повільніше, ніж з його початку, оскільки для цього доводиться аналізувати всі попередньо упаковані файли;

– якщо безперервний архів виявиться пошкоджений, то не вдасться витягти не лише пошкоджений файл, але і всі файли, що йдуть після нього, тому під час створення неперервних архівів має сенс завжди додавати інформацію для відновлення.

Критерієм для вибору є *надійність носія* стислої копії. Якщо використовується досить надійний носій: жорсткий диск, магнітооптичний диск, ZIP -накопитель, JAZZ -накопитель і т.п., можна використовувати безперервний архівний файл. Якщо використовується ненадійний накопичувач: гнучкий магнітний диск або магнітна стрічка стримера, застосування безперервних архівів не рекомендується.

Саморозпакувальний архів (*self-extracting archive,* SFX) — файл, комп'ютерна програма, що поєднує в собі архів і виконуваний код для його розпакування. Він має розширення імені .EXE. Такі архіви, на відміну від звичайних, не вимагають окремої програми для їх розпакування (отримання вихідних файлів, з яких вони створені), якщо виконуваний код можна виконати у зазначеній операційній системі. Це зручно, коли невідомо, чи є у користувача, якому передається архів, відповідна програма розпакування.

Дані в архіві можуть бути зашифровані в будь-який спосіб. При використанні універсальних архіваторів зазвичай використовується просто шифрування за паролем.

Коефіцієнт стиснення **(**K_C) – характеристика міри стиснення, яка показує, в скільки разів зменшився об'єм початкового файлу. Визначається як відношення об'єму стислого файлу V_C до об'єму початкового файлу $V_П$, виражене у відсотках.

Міра стиснення залежить від використовуваної програми, методу стиснення і типу початкового файлу. Найдобріше стискуються графічні, текстові файли і файли даних, для яких міра стиснення може досягати 5-40 %, менше стискуються файли виконуваних програм і завантажувальних модулів – 60-90 %. Майже не стискуються архівні файли.

Основні формати упаковки даних

Нині застосовується декілька десятків програм-архіваторів, які відрізняються переліком функцій і параметрами роботи, проте кращі з них мають приблизно однакові характеристики. З числа найбільш відомих програм можна виділити ARJ, ICE, HYPER, CAB, ZIP, PAK, ZOO, EXPAND, RAR, 7Z.

Існує ще один цікавий клас програм, які також можна віднести до архіваторів. Це *пакувальники виконуваних файлів* (тобто файлів з розширенням .com і .exe). Після упаковки виконувані файли залишаються працездатними, і їх можна запустити без яких-небудь додаткових операцій. Міра стиснення досягає 10-50 %. Останнім часом виробники програмних продуктів упаковують виконувані файли. Крім того, є *утиліти стиснення диска*, що упаковують файли «на льоту», у момент запису на диск, і розпаковують їх в процесі читання. Ці програми прозорі для користувача, і при роботі з ОС видимих змін не відбувається.

Серед усієї безлічі різних форматів упаковки можна виділити формати, що найчастіше зустрічаються .ZIP, .RAR і .7Z.

ZIP – формат стиснення та архівації даних. Файл цього формату зберігає у стиснутому або не стиснутому вигляді один або декілька файлів. Використовує LZW-стиснення, яке не вносить спотворень і втрат. ZIP є стандартним форматом в ОС Windows.

ZIP – достатньо простий формат, що окремо стискує кожен файл. Через це є можливість видобувати окремі файли без читання всього архіву; в теорії це дозволяє отримати краще стиснення, використовуючи різні алгоритми для певних типів файлів. Проте, недоліком цього методу є те, що упакований архів з великою кількістю файлів буде значно більшим, ніж якби він був стиснений як один файл.

Разом з безліччю утиліт, що працюють з zip-файлами з командного рядка, в середині 1990-х років з'явилися і графічні zip-програми. Серед них однією з найпопулярніших стала WinZip.

ZIP став де-факто стандартом для компресії даних. Безліч конкуруючих архіваторів, крім свого власного, також підтримують формат ZIP. Цей спосіб стиснення також широко використовується в інших програмах і навіть в деяких форматах файлів.

RAR – поширений формат стиснення даних і програма-архіватор, розроблений російським програмістом Євгенієм Рошалом. Він написав програму-архіватор для пакування/розпаковування RAR, спочатку під DOS, потім і для інших платформ. Версія для Microsoft Windows розповсюджується у складі багатоформатного архіватора з графічним інтерфейсом WinRAR.

Програма WinRAR за споживчими властивостями об'єктивно перевершує усі інші архіватори. У міжнародному секторі Інтернету цей формат використовують професіонали. Оскільки WinRAR дозволяє працювати також з архівами у форматах .ZIP, .ARJ і деяких інших, він загалом задовольняє більшість потреб користувача в засобах стискування інформації.

7-Zip – файловий архіватор з високим ступенем стиснення. Програма вільно поширюється. Вона підтримує формати: повністю – 7z, ZIP, gzip, bzip2, tar; частково (тільки розпаковування і перегляд) – CAB, RAR, ARJ, cpio, RPM, deb, ar, Z, LZH, Compiled HTMLHelp, SPLIT.

Є можливості створення саморозпакувальних архівів, багатотомних архівів, шифрування архівів.

У більшості випадків ступінь стиснення вищий, ніж в RAR, за винятком деяких мультимедіа даних. Швидкість стиснення при цьому нижче, але не критично (як правило, не більше ніж на 30 %).

Програма WinRAR забезпечує:

– повну підтримку архівів RAR I ZIP;
– високий ступінь стиснення інформації завдяки високоефективному алгоритму стиснення даних;
– стиснення мультимедіа файлів за допомогою спеціального алгоритму;
– підтримка технології Drap and Drop;
– керування архівами форматів CAB, ARJ, LZH;
– підтримку неперервних архівів (ступінь стиснення інформації в них на 10-15% більший, ніж звичайними методами стиснення);
– підтримку багатотомних архівів;
– створення звичайних і багатотомних архівів, що розпаковуються (SFX);
– відновлення фізично пошкоджених архівів;
– підтримку додаткових функцій (шифрування, додання архівних коментарів, протоколювання помилок та ін.).

Архіватор працює в режимі керування файлами або архівами. При завантаженні програми WinRAR активним є режим керування файлами. Для входження в режим керування архівами треба двічі клацнути мишею на імені архіву, знаходячись у режимі керування файлами.

ЛЕКЦІЯ 6. КЛАСИФІКАЦІЯ ВІРУСІВ ТА ЗАПОБІГАННЯ ЗАРАЖЕННЮ ДИСКІВ. АНТИВІРУСНІ ПРОГРАМИ

6.1 Комп'ютерний вірус

Комп'ютерний вірус – це програма, створена людиною. Жоден вірус, що «працює» на ПК, не може виникнути ні з чого. Він створюється програмістами. Вірус може знаходитись не лише у файлі, з яким переважно асоціюється поняття програми – це може бути просто сукупність машинного коду, що поширюється по комп'ютерній мережі. Але, все ж таки, у всіх випадках цей код створений людиною.

Причин створення вірусів дуже багато. Для декого віруси є їхнім бізнесом. До того ж, не тільки для авторів, але й для тих, хто з цими вірусами бореться. Для інших – це хобі. Хобі збирання вірусних колекцій і хобі написання вірусів. З останнього, до речі, починав відомий програміст, що займається саме боротьбою з вірусами – Ігор Данилов. Для третіх – створення вірусів – просто спосіб показати свій успіх і незалежність, оскільки в певних колах подібна діяльність просто необхідна для підняття престижу. Ще для когось віруси – покликання, адже бувають лікарі за покликанням, отож, може бути і комп'ютерний лікар за покликанням. Для декого віруси служать приводом пофілософствувати на теми виникнення і розвитку «комп'ютерного життя». Віруси – це навіть стаття Кримінального кодексу.

В багатьох країнах, наприклад, тільки за написання шкідливої програми можливе позбавлення волі терміном до п'яти років, а подекуди і більше. Та для більшості користувачів комп'ютерні віруси – це щоденний головний біль, причина порушень в роботі комп'ютера - ворог номер один.

Вірус – це програма, хоча деякі початківці до цього часу про це не знають. І, очевидно, що шкодити вона може лише програмно, але ніяк не апаратно. Страшні казки про віруси, які вбивають і позбавляють розуму користувачів за допомогою виведення на екран смертельної кольорової гами, були і залишаються тільки казками. Так само вірус не може пошкодити жодного пристрою ПК. Отже, не потрібно при виході з ладу певного вузла ПК шукати причину у вірусах. Не існувало, не існує і не з'явиться такий вірус, який би фізично пошкодив апаратну частину ПК. Про це потрібно завжди пам'ятати. Єдина непряма шкода, яку може завдати вірус апаратній частині ПК – перепрограмувати BIOS. Наприклад, вірус "Чорнобиль", завдяки високій активізації своєї діяльності 26 квітня, в день аварії на ЧАЕС, що в кінці 90-х років приніс справжню епідемію в комп'ютерний світ.

Отже однією із характеристик комп'ютерного вірусу є його здатність наносити шкоду програмному забезпеченню ПК. Але це не єдина характеристика вірусів, адже існують віруси, які не займаються нічим, крім само розмноження.

Основною характеристикою будь-якого комп'ютерного вірусу є здатність копіювати себе. Навіть якщо програма взагалі не приносить ніякої шкоди, тільки розмножує себе – це вже вірус. І навпаки, якщо програма робить велику шкоду для програмного забезпечення, але не копіює себе - така програма вірусом не вважається.

Отже, комп'ютерний вірус – це спеціально створена програма або сукупність машинного коду, яка здатна розмножуватись і, як правило, виконує на ПК певні деструктивні дії.

6.2 Історія виникнення комп'ютерних вірусів

Перші комп'ютерні віруси з'явились на початку 80-х років. Поняття «комп'ютерний вірус» вперше ввів відомий англійський програміст Фред Коуен. Цей термін прозвучав у вересні 1984 році на конференції з безпеки інформації, яка проходила в США. Він провів ряд експериментів на системі, що працювала під ОС. Вірус був імплантований на початок утиліти і протягом 30 хвилин йому надавались права «суперюзера». В експерименті вірус проявив надзвичайно високий ступінь розмноження (півсекунди на одне зараження).

Хоча, потрібно відмітити, що ідея саморозмноження своїм корінням сягає в далекі 50-ті роки. її досліджував ще в 1951 році один із засновників теоретичних принципів ЕОМ Джон фон Нейман. Вже в 60-ті роки створено гру, в якій декілька асемблерних програм, що мали назву «організми», завантажувались в пам'ять ЕОМ і повинні були знищувати «організми» супротивника. Переможцем

вважався той учасник, «організми» якого захоплювали всю пам'ять.

На початку 80-х з'являється перший завантажувальний вірус для ПЕОМ Apple II, що мав назву EIк CIопег. А в 1982 році програмісти дослідницького центру Xerox розробили перший в історії мережевий вірус «черв'як» і провели над ним ряд експерементів.

Перші, офіційно зареєстровані, комп'ютерні віруси з'явились в 1986, а перша епідемія, що охопила декілька тисяч комп'ютерів, зареєстрована в 1988 році.

Кількість комп'ютерних вірусів збільшується з кожним роком. В 1990 році було відомо близько 500 вірусів, в 1992 - 1500, в 1994 - 4000, в 1996 - 8000, 1998 - 15000, 2000 - 40000, на кінець 2001 -близько 60000. На цьому розвиток вірусів не зупиняється.

Ситуація з вірусами суттєво змінилася кілька років тому. Якщо до того часу кожен турбувався про безпеку власного комп'ютера і власних даних, то із збільшенням кількості машин, з появою корпоративних мереж, із поширенням НЕТА проблема набула нових характеристик. Раніше віруси проникали на робочі місця хіба що з не ліцензованим програмним забезпеченням, або з комп'ютерними іграми. Зараз шляхи розповсюдження вірусів значно розширились. З'явилось поняття макровірусу, тобто вірусу, що розповсюджується через макроси написані для таких програм як Word і Excel (поняття про макроси буде розглянуте пізніше).

При достатньо активному документо обороті, як із зарубіжними партнерами, так і всередині країни, макровіруси здатні повністю паралізувати роботу великої компанії. На Заході таке вже траплялося. Друга проблема виникла завдяки НЕТА Практично немає ніякої гарантії, що на поштових або файлових серверах разом із потрібною інформацію ви не отримаєте декілька вірусів, при цьому користувач про це може і не здогадуватись. Чим більше ПК підключається до комп'ютерних мереж, тим важче контролювати розповсюдження вірусів. Відомі випадки, коли через мережу вірус паралізував за короткий час роботу тисяч ПК, швидко долаючи великі відстані між материками. Наприклад, ще на початку 1989 р. вірусом, створеним американським студентом Моррісом, були заражені і виведені з ладу близько 7 тисяч комп'ютерів, в тому числі й ті, які належали міністерству оборони США. Автор вірусу був засуджений до 3-х місяців тюрми та оштрафований на 270 тис. доларів. Покарання могло бути більш суворим, але суд врахував, що вірус не зіпсував даних, а лише розмножувався.

6.3 Запобігання зараженню дисків

Програма всередині якої знаходиться вірус, називається зараженою. Коли така програма починає роботу, то спочатку отримує управління вірус. Він знаходить і заражає інші програми, а також виконує певні шкідливі дії (наприклад, псує файли або таблицю розміщення файлів на диску, засмічує оперативну пам'ять і т.д.) Після того, як вірус виконає ці дії, він передає управління тій програмі, в якій він знаходиться, і вона працює так, як звичайно. Тим самим зовнішня робота зараженої програми виглядає так, як і незараженої.

Багато різновидів вірусів побудовані так, що при запуску зараженої програми вірус залишається постійно в пам'яті комп'ютера (точніше, до перевантаження ОС) і час від часу заражає програми та виконує шкідливі дії на комп'ютері. Всі дії вірусу можуть виконуватися досить швидко і без видачі будь-яких повідомлень, тому користувачу дуже важко помітити, що в комп'ютері відбувається щось незвичайне.

Поки на комп'ютері заражено відносно мало програм, наявність вірусу може бути практично непомітна. Однак через деякий час починають відбуватися нетипові дії, наприклад:

- деякі програми припиняють роботу, або починають працювати неправильно;
- на екран виводяться сторонні повідомлення або символи і т.д.
- робота на комп'ютері суттєво сповільнюється;
- деякі файли виявляються зіпсутими.

До цього моменту, як правило, вже досить багато (навіть більшість) тих програм, якими ви користуєтесь, виявляються зараженими вірусом, а деякі файли та диски - зіпсутими. Більше того, зараженими за допомогою дискет або по локальній чи глобальній мережі виявляються цілий ряд комп'ютерів з якими «спілкувався» даний ПК.

Деякі різновидності вірусів поводяться ще більш зухвало. Вони спочатку непомітно заражають велику кількість програм або дисків, а потім наносять дуже серйозні пошкодження, наприклад,

форматують увесь жорсткий диск на комп'ютері.

Для того, щоб програма-вірус була постійною, вона повинна бути невеликою. Тому, як правило, віруси пишуться на мовах низького рівня досить висококваліфікованими програмістами. Хоча останнім часом в Інтернеті вільно поширюються програми, з допомогою яких без особливих зусиль можна створювати найрізноманітніші типи вірусів за готовими шаблонами. Вони дали новий поштовх у розвитку комп'ютерних вірусів, адже тепер вірус може створити навіть початківець у програмуванні.

Комп'ютерний вірус може зіпсувати будь-який файл, але тільки деякі види файлів вірус може заразити. Це означає, що вірус може «вкорінитися» в ці файли, тобто змінити їх так, що вони міститимуть вірус, який за певних обставинах може почати свою роботу. Слід відмітити, що тексти програм та текстових документів, інформаційні файли баз даних не можуть бути зараженні вірусом, він може їх тільки зіпсувати. Види файлів, які можуть бути заражені вірусом:

– виконуючі файли, тобто файли з розширеним ім'ям COM і EXE, а також оверлейні файли, завантажені іншими програмами. Вірус в заражених виконуючих файлах починає свою роботу при завантаженні тієї програми, в якій він знаходиться;

– файли документів та шаблонів, створених програмами Word, Excel, Access та іншими офісними програмами, а точніше макроси, що використовуються там. Цей тип вірусів порівняно молодий і називають його макровірусами. Деякі з вірусів цього типу є надзвичайно шкідливими. Наприклад вірус W97M.Thus при активізації 13 грудня здатний знищити всі файли на диску С, зберігаючи при цьому структуру каталогів (папок).

– блок початкового завантаження операційної системи і головний завантажувальний запис жорсткого диску. Вірус, який заразив ці ділянки, як правило, складається з 2-х частин, оскільки на цих ділянках диску, важко розмістити програму вірусу вцілому. Частина вірусу, що не поміщається в них, розташована на іншій ділянці диску, який оголошується дефектним. Такий вірус починає свою роботу при початковому завантажуванні операційної системи і є резидентним, тобто постійно знаходиться в пам'яті комп'ютера. Відомі випадки, коли вірус форматує додаткову доріжку диску, куди і записує основну частину програми;

– таблиці файлової системи та каталоги. Як відомо, в кожен каталог записуються імена файлів, дата та час створення, номер першого кластера файлу, а також резервні байти, що ОС не використовуються. Віруси цього типу, записавшись в кластери, помічають їх як пошкоджені, а тоді реорганізовують файлову систему. При цьому інформація про перші кластери деяких виконуючих файлів записується у резервні біти, а на її місце поміщається посилання на тіло віруса. Тому, при спробі користувача завантажити відповідну програму - вірус отримує керівництво. Цей тип вірусів, з'явившись в 1991 році, викликав в Росії справжню епідемію, яку можна порівняти із чумою.

– драйвери пристроїв, тобто файли, які здійснюють програмне керування зовнішнім пристроєм. Вірус, який знаходиться в цих файлах, починає свою роботу при кожному звертанні до відповідного пристрою;

– системні файли, тобто файли IO. SYS і MSDOS. SYS. Це досить небезпечно, оскільки вони, як і у випадку зараження блоків початкового завантаження дисків, починають діяти при кожному завантаженні ПК.

Як правило, кожна конкретна різновидність вірусу може заразити тільки один або два типи файлів. На даний час частіше всього зустрічаються макровіруси, тоді як в 90-ті роки найпоширенішими були віруси, що заражали COM–файли, а на другому місці – EXE–файли.

До загальних засобів, що допомагають запобігти зараженню та його руйнівних наслідків належать:

– резервне копіювання інформації (створення копій файлів і системних областей жорстких дисків);

– уникнення користування випадковими й невідомими програмами. Найчастіше віруси розповсюджуються разом із комп'ютерними вірусами;

– перезавантаження комп'ютера перед початком роботи, зокрема, у випадку, якщо за цим комп'ютером працювали інші користувачі;

– обмеження доступу до інформації, зокрема фізичний захист дискети під час копіювання файлів із неї. роблено свій антивірус. Однак, багато сучасних антивірусних пакетів мають у своєму складі спеціальний програмний модуль, який називається евристичний аналізатор, і який здатний досліджувати вміст файлів на наявність коду, характерного для комп'ютерних вірусів. Це дає змогу вчасно виявляти та попереджати про небезпеку зараження новим вірусом.

6.4 Вибір антивірусу

Слід зауважити, що вибір одного "найкращого" антивірусу є вкрай помилковим рішенням. Рекомендується використовувати декілька різних антивірусних пакетів одночасно. Вибираючи антивірусну програму слід звернути увагу на такий параметр, як кількість розпізнаючих сигнатур (послідовність символів, які гарантовано розпізнають вірус). Другий параметр - наявність евристичного аналізатора невідомих вірусів, його присутність дуже корисна, але суттєво уповільнює час роботи програми. На сьогоднішній день існує велика кількість різноманітних антивірусних програм. Розглянемо коротко найбільш поширені в Україні.

DRWEB

Один з кращих антивірусів із сильним алгоритмом знаходження вірусів. Поліфаг, здатний перевіряти файли в архівах, документи Word і робочі книги Excel, виявляє поліморфні віруси, котрі в останній час, отримують все більше поширення. Достатньо сказати, що епідемію дуже небезпечного вірусу OneHalf зупинив саме DrWeb. Евристичний аналізатор DrWeb, досліджуючи програми на наявність фрагментів коду, характерних для вірусів, дозволяє знайти майже 90% невідомих вірусів. При завантаженні йти майже 90% невідомих вірусів. При завантаженні програми в першу чергу DrWeb перевіряє самого себе на цілісність, після чого тестує оперативну пам'ять. Програма може працювати у діалоговому режимі, має дуже зручний інтерфейс користувача, який можна настроювати.

ADINF

Антивірус-ревізор диска ADINF (Avanced DiskINFoscope) дозволяє знаходити та знищувати, як існуючі звичайні, stealth- і поліморфні віруси, так і зовсім нові. Антивірус має в своєму розпорядженні лікуючий блок ревізору ADINF – Adinf Cure Module – який може знешкодити до 97% всіх вірусів. Цю цифру наводить «ДіалогНаука», виходячи з результатів тестування, котре відбувалося на колекціях вірусів двох визнаних авторитетів в цій області – Д.Н.Лозинського й фірми Dr.Solomon's (Великобританія).

Якщо вірус є файловим, то тут на допомогу приходить лікуючий блок Adinf Cure Module, який на основі звіту основного модуля про заражені файли порівнює нові параметри файлів із попередніми, які зберігаються в спеціальних таблицях. При виявленні розбіжностей ADINF відновлює попередній стан файлу, а не знищує тіло вірусу, як це роблять поліфаги.

AVP Антивірус AVP (AntiVirus Program) відноситься до поліфагів, у процесі роботи перевіряє оперативну пам'ять, файли, в тому числі архівні, на гнучких, локальних, мережних і CD-ROM дисках, а також системні структури даних, такі як завантажувальний сектор, таблицю розділів і т.д. Програма має евристичний аналізатор, котрий, за твердженнями розробників антивірусу здатний знаходити майже 80% усіх вірусів.

Eset NOD32 Antivirus – популярна антивірусна програма, що швидко працює та ефективно захищає від усіх видів вірусів, шпигунських і рекламних програм.

Norton AntiVirus знешкоджує всі види вірусів, захищає комп'ютер під час роботи в інтернеті, перевіряє електронну пошту і мережні файли.

Антивирус Касперского – антивірус, що володіє (за рахунок спеціального алгоритму) дуже високим відсотком визначення вірусів, включно з невідомими.

avast! Free Antivirus – безкоштовна антивірусна програма яка розроблена для повноцінного захисту домашнього ПК.

Security Essentials – безкоштовна антивірусна програма яка надійно захищає від вірусів, шпигунських програм, руткітів та троянів.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: КЛАСИФІКАЦІЯ АНТИВІРУСНИХ ЗАСОБІВ

На сьогодні відомі десятки тисяч вірусів, які в цілому мають конкретну класифікацію. Спробуємо детальніше розглянути основні групи, на які поділяються комп'ютерні віруси.

I. Поділ вірусів за середовищем їх розповсюдження:

Завантажувальні віруси – це найбільш небезпечна група вірусів, що заражають Boot Record та Master Boot Record логічних та фізичних дисків. Про ці віруси ми вже говорили попередньо.

Файлові віруси. Ці віруси поширюються, заражаючи файли різних типів, як вже було сказано, – найчастіше це виконуючі файли та файли оверлеїв. До цієї групи слід також віднести макровіруси, хоч інколи їх виділяють як окремий клас вірусів.

Завантажувально–файлові віруси здатні вразити як код завантажувальних секторів, так і код файлів, як правило системних.

Віруси сімейства Dir використовують інформацію про файлову структуру та вміст каталогів

Multipartition – віруси можуть вражати одночасно виконуючі файли, BIOS – сектор, МВК, FAT і каталоги і є найбільш небезпечними, особливо, якщо вони ще й володіють поліаморфними властивостями і елементами невидимості.

Мережеві віруси – це віруси, що поширюються як сукупність машинного коду в комп'ютерних мережах.

Поштові віруси – на сьогодні досить нова але надзвичайно поширена група вірусів, що розповсюджуються разом із поштовими повідомленнями у вигляді прикріплених до них файлів із програмним кодом. Як правило, такі віруси досить швидко розмножуються і час від часу викликають вірусні епідемії

II. Класифікація комп'ютерних вірусів за алгоритмом роботи:

Віруси «паразити» найпростіші віруси що використовують «тіло» інших файлів (виконуючих), записуючи туди себе. Вони можуть бути досить легко виявлені і знешкоджені.

Віруси супутники створюють копію exe-файлу з розширенням сот і записують туди себе. Коли з командного рядка Р05 завантажують такий файл, то як правило розширення не вказують, а за правилами 005, першим завантажується сот файл, тобто вірус.

Віруси «черв'яки» (віруси-реплікатори) не створюють собі файлу, а поширюються лише в комп'ютерних мережах та в оперативній пам'яті у вигляді певного машинного коду. Вони ніби черв'яки проникають в оперативну пам'ять ПК через комп'ютерну мережу, пронизуючи системи захисту.

– Студентські віруси – це віруси, які мають в собі багато помилок і написані, як правило, початківцями.

– Віруси «невидимки» фальсифікують інформацію, перехоплюючи звертання антивірусної програми, до заражених ділянок диску і направляючи її на незаражені. Вірус перехоплює вектор переривання. Ця технологія використовується, як у файлових, так і в завантажувальних вірусах.

– Віруси «мутанти» («привиди») або поліморфні – не мають постійної сигнатури (машинного коду), за якою можна було б виявити вірус. Вони міняють сигнатуру з кожною копією і тому з ними важко боротись. Виявляють такі віруси лише за допомогою евристичного аналізу, коли антивірусна програма «прокручує» алгоритм роботи виконуючих файлів і в разі підозрілих операцій приймає це за вірус. Таким же чином антивірусні програми шукають невідомі ще їм віруси.

– Ретровіруси – це звичайні файлові віруси, які прагнуть заразити антивірусні програми, знищуючи їх або роблячи непрацездатними. Тому практично всі антивіруси в першу чергу перевіряють свої власні розміри і контрольну суму.

«Троянські» віруси здійснюють шкідливі дії замість оголошених легальних функцій або разом з ними. Вони переважно не здатні на саморозповсюдження і передаються тільки при копіюванні користувачем. Часто ці віруси використовують в якості «шпигунів». Проникаючи по мережі на ПК, вони стараються «затаїтись» і «вкрасти» паролі користувача (особливо виходу в Інтернет) і передати їх господарю. Деякі троянські віруси готують ґрунт на зараженому ними ПК для проникнення без перешкод інших вірусів, що слідують за ними. Боротись з такими вірусами (особливо новими)

досить важко, адже в їх коді немає ніякої диструктивної дії (не міняється розмір інших файлів, не форматуються диски), а навпаки вони стараються ніяк себе не проявити. Для боротьби з такими вірусами використовуються спеціальні програми (фаєрволл) – мережеві екрани, які під час підключення до мережі слідкують чи не пробує якась програма на ПК вийти в Інтернеті. Якщо така спроба відбулась, то вона блокується і виводиться повідомлення, із запитом дозволу на таку операцію. Корисною функцією фаєрволл є те, що він може захистити не лише від троянських вірусів, але й від хакерських атак із зовні.

Потрібно відмітити, що існує багато відомих троянських вірусів, які не лише виступають в ролі «шпигунів» але й самі несуть досить високу деструктивну дію

III. Поділ вірусів за деструктивною дією:

Нешкідливі віруси – це віруси, які не приносять жодної шкоди, а просто себе копіюють багато разів, заповнюючи диски, або загромаджуючи оперативну пам'ять.

Не небезпечні віруси схожі до попередніх, але крім цього їх дія супроводжується різними спецефектами (відео та звуковими).

Небезпечні віруси – це віруси дія яких призводить до серйозних збоїв в роботі ПК, таких як зависання комп'ютера і т.д.

Дуже небезпечні віруси – це віруси, дія яких супроводжується знищенням інформації (файлів, каталогів, форматування цілих дисків). Це найбільш небезпечні віруси, що несуть реальну загрозу комп'ютерній системі.

IV. Класифікація вірусів за принципом дії:

Резидентні – це віруси, що завантажуються в оперативну пам'ять і постійно там знаходяться, аж до виключення живлення чи перезавантаження ПК.

Нерезидентні – це віруси, які короткочасно завантажуються в пам'ять, виконують потрібні їм дії і вивантажуються з пам'яті.

V. Поділ вірусів за місцем втілення у файли:

На початку файлу.

– Всередині файлу.
– В кінці файлу.

Для виявлення, видалення і захисту від комп'ютерних вірусів розроблено декілька видів спеціальних програм, які дозволяють виявляти і знищувати віруси. Такі програми називаються антивірусними. Розрізняють такі види антивірусних програм:

– Програми–детектори;
– Програми–доктори, або фаги;
– Програми–ревізори;
– Програми–фільтри;
– Програми–вакцини, або іммунізатори.

Програми–детектори здійснюють пошук характерної для конкретного вірусу сигнатури в оперативній пам'яті й у файлах і при виявленні видають відповідне повідомлення. Недоліком таких антивірусних програм є те, що вони можуть знаходити тільки ті віруси, які відомі розробникам таких програм.

Програми–лікарі, або фаги, а також програми–вакцини не тільки знаходять заражені вірусами файли, але і «лікують» їх, тобто видаляють з файлу тіло програми-вірусу, повертаючи файли в початковий стан. На початку своєї роботи фаги шукають віруси в оперативній пам'яті, знищуючи їх, і тільки потім переходять до «лікування» файлів. Серед фагів виділяють поліфаги, тобто програми–доктори, призначені для пошуку н знищення великої кількості вірусів. Найбільш відомі з них: Kaspersky Antivirus, Norton AntiVirus, Doctor Web.

У зв'язку з тим, що постійно з'являються нові віруси, програми–детектори і програми–доктори швидко застарівають, і потрібне регулярне оновлення версій.

Програми–ревізори належать до найнадійніших засобів захисту від вірусів. Ревізори запам'ятовують вихідний стан програм, каталогів і системних областей диска тоді, коли комп'ютер не заражений вірусом, а потім періодично або за бажанням користувача порівнюють поточний стан з вихідним. Виявлені зміни виводяться на екран монітора. Як правило, порівняння станів

проводять відразу після завантаження операційної системи. При порівнянні перевіряються довжина файла, код циклічного контролю (контрольна сума файла), дата і час модифікації, інші параметри. Програми–ревізори мають досить розвинуті алгоритми, виявляють стелс–віруси і можуть навіть відрізнити зміни версії перевіреній програми змін, внесених вірусом. До числа програм-ревізорів належить широко поширена програма Kaspersky Monitor.

Програми–фільтри або «сторожи» являють собою невеликі резидентні програми, призначені для виявлення підозрілих дій при роботі комп'ютера, характерних для вірусів. Такими діями можуть бути:

- Спроби корекції файлів з розширеннями COM. EXE;
- Зміна атрибутів файла;
- Прямий запис на диск з абсолютного адресу;
- Запис у завантажувальні сектори диска;
- Завантаження резидентної програми.

При спробі будь-якої програми здійснити вказані дії «сторож» посилає користувачеві повідомлення і пропонує заборонити або дозволити відповідну дію. Програми-фільтри вельми корисні. оскільки здатні виявити вірус на ранній стадії його існування, до розмноження. Однак вони не «лікують» файли і диски.

Для знищення вірусів потрібно застосувати інші програми, наприклад фаги. До недоліків програм-сторожів можна віднести їх «настирливість» (наприклад, вони постійно видають попередження про будь-якій спробі копіювання виконуваного файлу), а також можливі конфлікти з іншим програмним забезпеченням.

Вакцини або іммунізатори – це резидентні програми. Запобігають зараженню файлів. Вакцини застосовують, якщо відсутні програми-доктори, «лікують» цей вірус. Вакцинація можлива тільки від відомих вірусів. Вакцина модифікує програму або диск таким чином, щоб це не відбивалося на їх роботі, а вірус буде сприймати їх зараженими і тому не впровадитеме. В даний час програми–вакцини мають обмежене застосування.

Своєчасне виявлення заражених вірусами файлів і дисків, повне знищення виявлених вірусів на кожному комп'ютері дозволяють уникнути поширення вірусної епідемії на інші комп'ютери.

ЛЕКЦІЯ 7. ОСНОВНІ ХАРАКТЕРИСТИКИ MICROSOFT WORD

7.1 Основні характеристики MICROSOFT WORD

Microsoft Office Word (MS Word) – текстовий процесор, що випускається компанією Microsoft і входить до складу офісного пакету «Microsoft Office».

Microsoft Office – це офісний пакет, набір програм, створених корпорацією Microsoft для операційних систем Microsoft Windows і Apple Macintosh. До складу цього пакету входить програмне забезпечення для роботи з різними типами документів: текстами, електронними таблицями, базами даних тощо.

В основу функціонування текстового процесора MS Word покладено принцип «що ви бачите, те й одержуєте». Системний інтерфейс процесора Word не потребує спеціальних знань у сфері комп'ютерної техніки та інформатики і дає змогу бачити результат роботи в тому вигляді, в якому він буде надрукований на папері.

За допомогою текстового процесора Word користувачі можуть здійснювати:

– введення, перегляд, редагування та форматування тексту;
– вибір і створення стилю та шаблону документа;
– збереження документа на диску у вигляді файлів із певним ім'ям і розширенням
– відкривання та завантаження файлу з диска в оперативну пам'ять;
– підключення до порталу Microsoft SharePoint, на якому можна зберігати документи та отримувати відомості про нові версії окремих із них;
– перевірка правопису, створення словників користувача, здійснення рецензування, виносок, приміток та розсилки;
– формування, редагування, оброблення і сортування таблиць та створення тримірних діаграм;
– вставлення в текст документа ілюстрацій з інших додатків, що входять до складу програмного середовища Microsoft Office без втрати форматування;
– швидка зміна зовнішнього вигляду за допомогою експрес-стилів та тем;
– створення цифрового підпису документів для покращення їх захисту тощо.

В середовищі Windows реалізовано технологію, завдяки якій в документи можна вставляти різноманітні об'єкти, створені за допомогою інших програм-додатків. При цьому вмонтований об'єкт стає частиною поточного документа. У Word, як правило, входять програми–додатки, що підтримують цю технологію: Microsoft WordArt – програма введення текстових спец ефектів; Microsoft Graph – програма створення ділової графіки; Microsoft Equation Editor – програма введення математичних формул і рівнянь та ін.

Текстовий процесор Word дає змогу користувачу реалізувати механізм гіпертекстових посилань, забезпечуючи доступ до потрібних матеріалів, розташованих в файлах, що зберігаються на запам'ятовуючих пристроях комп'ютера, Інтернеті або в локальній мережі. Такі посилання можуть вказувати на документи, що зберігаються у Web-вузлах, на файловому сервері або на дисках персонального комп'ютера.

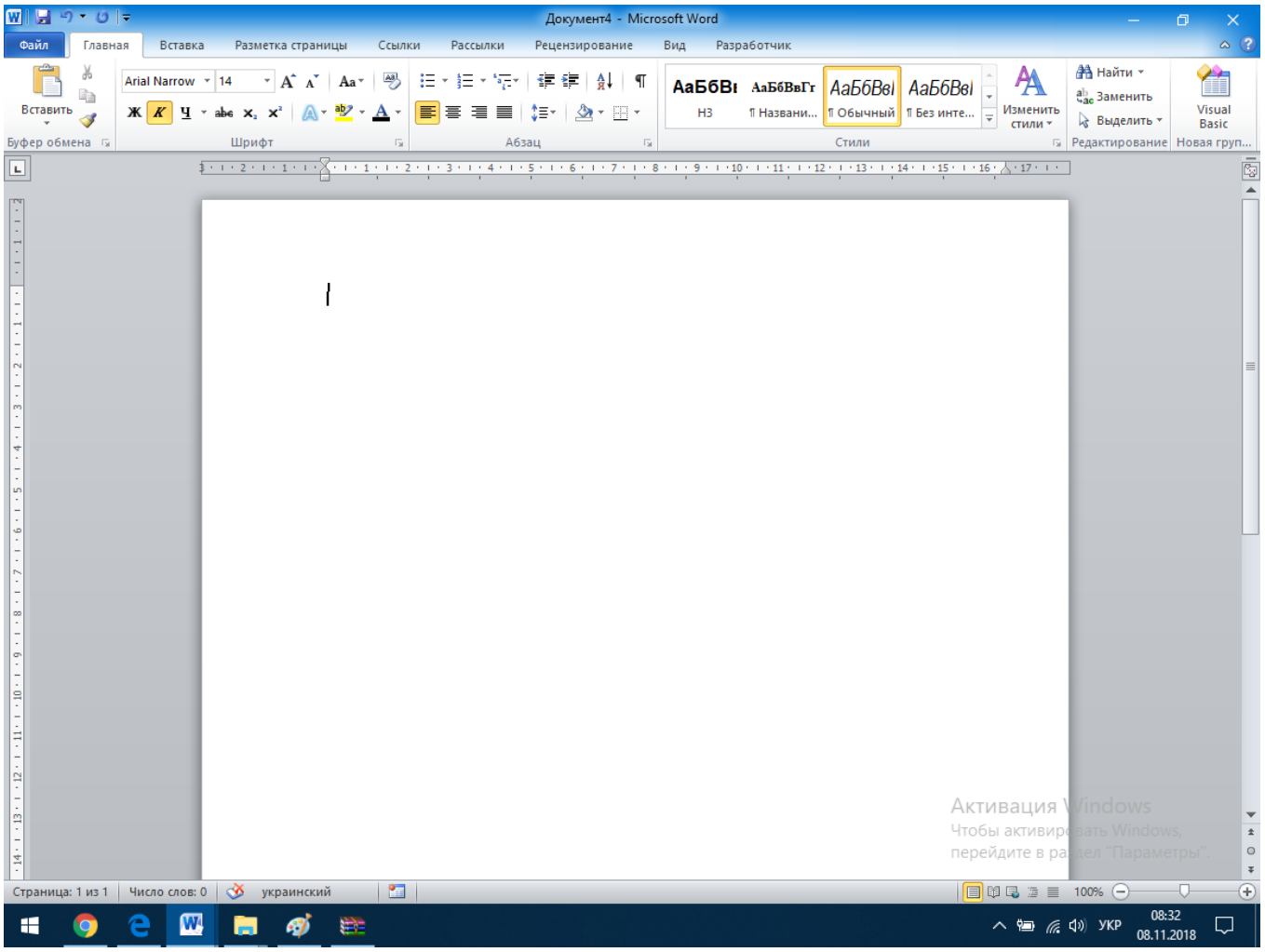
Word дає змогу реалізувати більшість можливостей настільної видавничої системи DTP (Desktop Publishing), призначеної для редагування книг, журналів, газет, рекламних оголошень тощо. За допомогою зазначеного програмного забезпечення можна створювати управлінські документи та бланки, що за якістю не поступаються друкованим. Як додатки до Word можуть використовуватися сучасні програми для опрацювання табличних даних та системи управління базами даних (СУБД), наприклад, Microsoft Excel, Lotus, Quattro Pro, Microsoft Access, Fox Pro та ін.

7.2 Робоче вікно MICROSOFT WORD

Після вмикання комп'ютера і введення відповідного пароля, якщо він є, автоматично завантажується операційна система Windows. Щоб завантажити текстовий процесор Word

користувачу належить звернутися до послуги Пуск; вибрати в головному меню системи Програми, а потім – Microsoft Word, або *Пуск – Програми – Microsoft Office – Microsoft Office Word.*

Після завантаження процесора на екрані монітора буде відображено його основне вікно.



У *верхній* частині вікна Word розміщуються: кнопка "Office", рядок головного меню, панель швидкого доступу, стрічка заголовка, горизонтальна лінійка форматування.

У *нижній* частині вікна: горизонтальний рядок стану системи.

У *правій* частині – вертикальна смуга прокручування, а в *лівій* – вертикальна лінійка форматування.

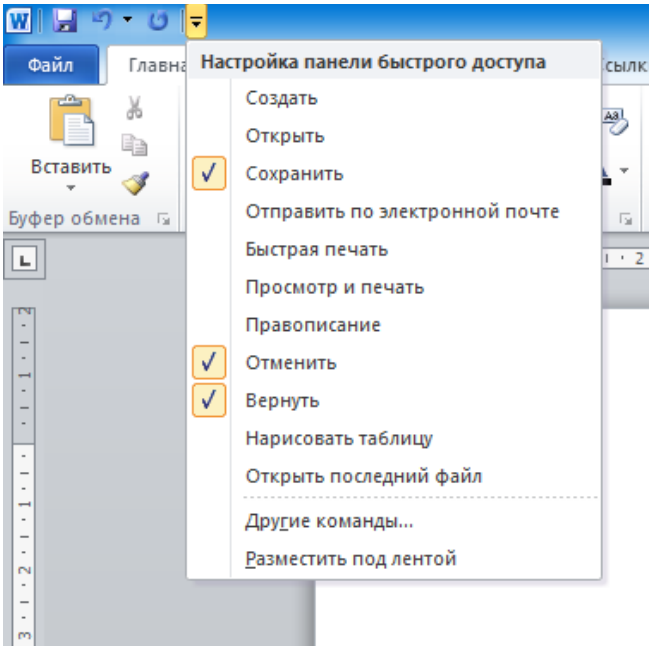
Розглянемо *основні елементи вікна Word.*

Кнопка "Office". Іконка кнопки розташована у верхньому лівому куті. При натисненні на ній лівої кнопки миші відкривається спадне меню, що дає змогу користувачеві створювати, відкривати, перетворювати, зберігати, друкувати, надсилати, публікувати та закривати документи.

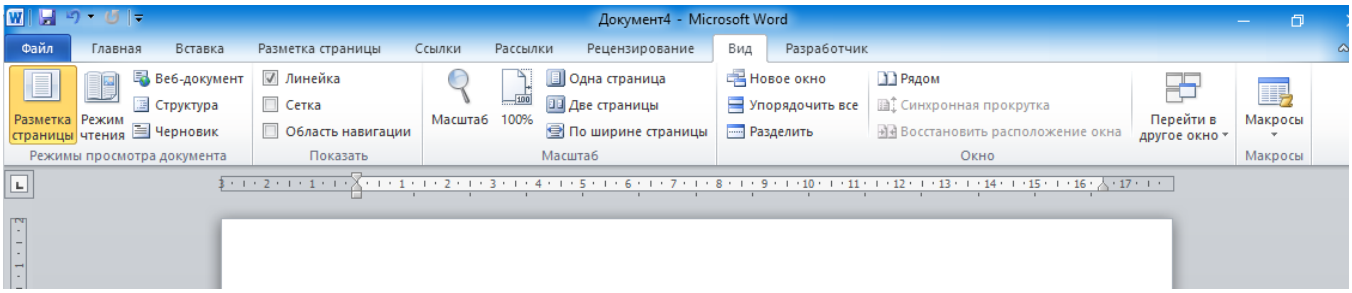
Рядок головного меню. Містить три основні компоненти: вкладки (призначені для окремого виду роботи), групи (зі спорідненими завданнями), команди (кнопка, поле для введення інформації або меню).

Використовується інтерфейс користувача Office Fluent, в якому інструменти групуються за призначенням, який містить вісім вкладок: Основне, Вставлення, Розмітка сторінки, Посилання, Розсилки, Рецензування, Вигляд, Надбудови. Кожна вкладка орієнтується на виконання певного завдання, а групи створенні в ній розділяють на під-завдання. Піктограми, що знаходяться в кожній з підгруп виконують якусь команду, або відкривають спадне меню. Відкривається кожна із вкладок за допомогою натискання правої клавіші миші.

Панель швидкого доступу. Це набір кнопок-піктограм, за допомогою яких здійснюється швидкий і наочний вибір та виконання команд. Зазвичай на ній відображаються найуживаніші команди *(Зберегти, Скасувати* та Повторити). При підготовці процесора до роботи користувач може самостійно налаштувати на панелі доступу необхідні піктограми. Для цього необхідно скористатись послугою *Настроїти панель швидкого доступу*.



Лінійки форматування. За допомогою «лінійки форматування» і «миші» можна швидко встановити відступи абзаців, розмір поля сторінки, розміри колонок на сторінках та в таблицях, а також точки табуляції тексту. Вмикання (вимикання) лінійок виконується через послугу *Лінійка* з меню Вигляд.



Смуги прокрутки. Ці смуги призначені для переміщення користувачем вмісту робочої ділянки вікна за допомогою миші по вертикалі та горизонталі. Натискання на ліву кнопку миші, при наведенні на кнопки зі стрілками-трикутниками вертикальної прокрутки, дозволяє перемістити документ на один рядок вгору або вниз. Перехід між сторінками забезпечується натисканням лівої кнопки миші на іншій парі кнопок зі спареними трикутниками. Натискання лівої кнопки миші на смузі прокрутки вище або нижче індикатора-бігунка переміщає документ на висоту екрана, відповідно, назад або вперед. Порядок використання горизонтальної смуги прокрутки є таким самим.

Рядок стану. В ньому виводяться різноманітні повідомлення та довідкова інформація, наприклад, номери поточної сторінки, загальна кількість сторінок, номер поточного рядка і позиція курсору в ньому, а також ярлики режимів перегляду та масштаб документу. Налаштування відображення послуг здійснюється за допомогою натискання на ліву кнопку миші на самому рядку стану.

У Word використовується п'ять основних режимів перегляду документу на екрані монітора.

«Розмітка сторінки» – забезпечує таке посторінкове зображення документа на екрані, яке він матиме на папері після друкування. Тільки в цьому режимі можна переглянути на екрані рисунки, ілюстрації, діаграми і т. ін.

«Читання в повноекранному режимі» – призначений для читання документа з екрана комп'ютера. У повноекранному режимі читання також є параметр перегляду документа так, як він би виглядав надрукованим.

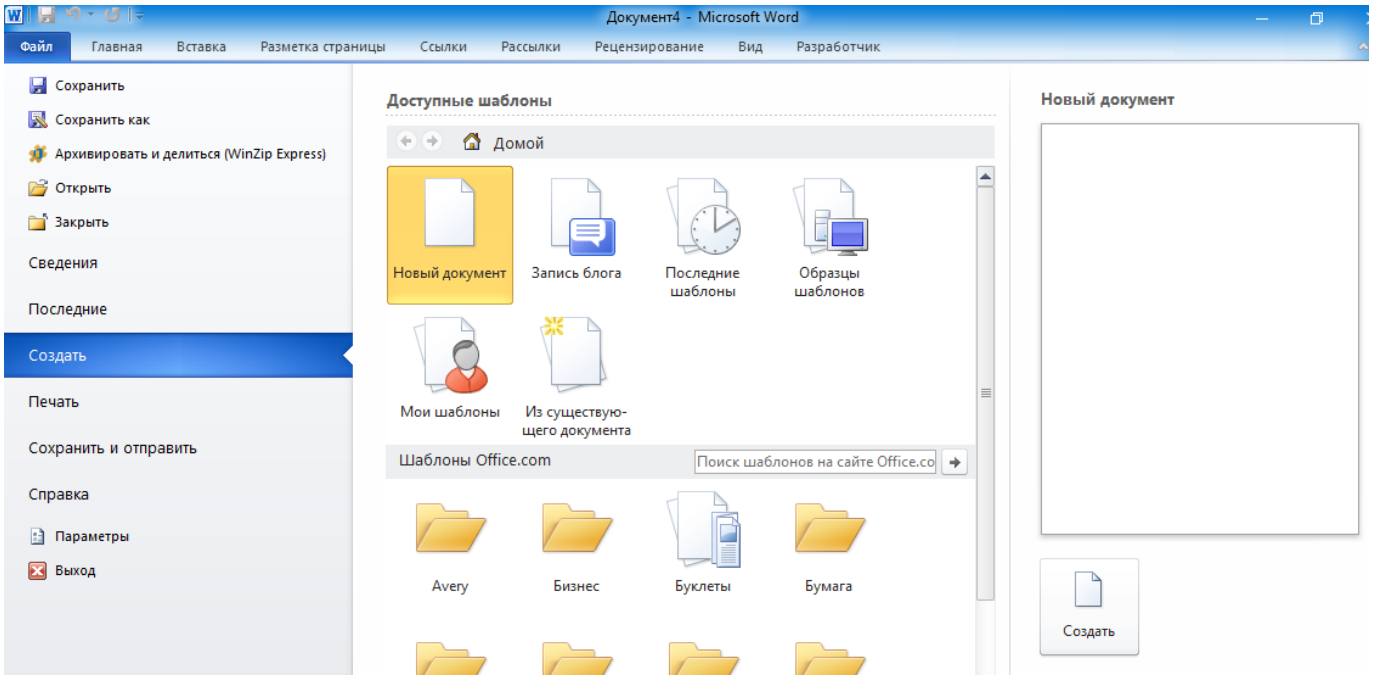
«Веб-документ» – у ньому на екрані в збільшеному масштабі відображається тільки текст документа, решта елементів середовища Word (меню, панелі, смуги прокрутки тощо) вимикаються.

«Структура» – у цьому режимі на екрані відображається тільки ескіз усього документа, тобто ієрархія його частин і заголовків. Вибираючи та переміщуючи рівні ієрархії, можна рухатися по тексту документа і змінювати положення його окремих фрагментів.

«Чернетка» – призначений для прискореного переглядання та друкування документів, які містять великі обсяги форматування.

Перемикання режимів здійснюється за допомогою послуги меню Вигляд або кнопок, розташованих у лівому нижньому куті вікна документа: Розмітка сторінки, Читання в повноекранному режимі, Веб-документ, Структура, Чернетка.

7.3 Створення документа Word



Для створення будь-якого документа необхідно скористатися кнопкою «Office» звернувшись до послуги Створити, яка відкриває поле Створення документа

У полі *Створення документа,* що з'являється на екрані, є ціла низка меню, які містять шаблони, призначені для створення

документів певного типу. Запропоновані системою шаблони користувач може модифікувати та пристосовувати для власних потреб.

Створення документів Word ґрунтується на стандартному шаблоні-файлі Новий документ (Normal.dot), що знаходиться у вкладці Чисті та недавні. Ця вкладка розміщена першою. Створення нового документа здійснюється без закриття старого.

Для швидкого створення документа можна скористатися панеллю швидкого доступу.

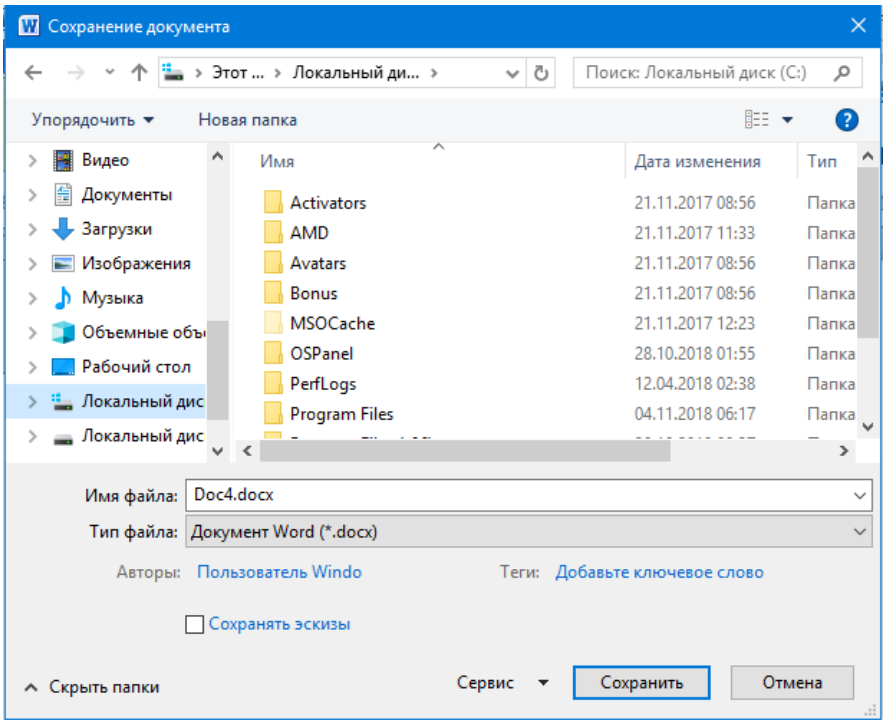
7.4 Збереження документа Word

Зберігається поточний документ на магнітному диску через послуги Зберегти та *Зберегти як,* що відкриваються за допомогою кнопки «Office». Ці послуги дозволяють користувачу відкрити вікно *Збереження документа.*

Звертання до послуги *Зберегти* дозволяє зберігати документ-файл з його початковим ім'ям.

Звертання користувача до послуги Зберегти як дозволяє зберігати документ-файл з ім'ям, що його задає сам користувач.

Ім'я файлу задається в однойменному полі робочого вікна Збереження документа. Після цього треба вибрати папку, де буде зберігатися документ, ввести його ім'я, обрати тип файлу та натиснути кнопку Зберегти.



Для збереження існуючого файлу-документу, після його перегляду і редагування досить натиснути на ліву кнопку миші, курсор якої вказує на послугу *Зберегти* на панелі швидкого доступу або натиснути клавіші *Ctrl+S.*

7.5 Відкриття документа Word

Відкривання будь-якого збереженого файла-документа виконується за допомогою послуги *Відкрити,* що активізується кнопкою "Office" і відкриває вікно *Відкриття документа*. При цьому користувач вибирає робочу папку з потрібним документом.

Якщо в списку *Тип файлів* встановлено параметр *Усі документи Word,* то у відповідному списку відображаються імена всіх файлів, що мають розширення зазначені у дужках. Тоді досить вибрати ім'я потрібного файлу і вказавши курсором миші на кнопці *Відкрити* натиснути на її ліву кнопку. У вікні *Відкриття документа* можна вибрати і відкрити кілька файлів одночасно, натиснувши ліву кнопку миші на їхніх іменах утримуючи клавішу Ctrl. Вікно відкривання документа Word можна активізувати також за допомогою однойменної кнопки-піктограми *Відкрити* на панелі швидкого доступу Word або при натисненні клавіш Ctrl+O.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: ФОРМАТУВАННЯ АБЗАЦІВ У MICROSOFT WORD. АВТОМАТИЧНА НУМЕРАЦІЯ ТА МАРКУВАННЯ АБЗАЦІВ

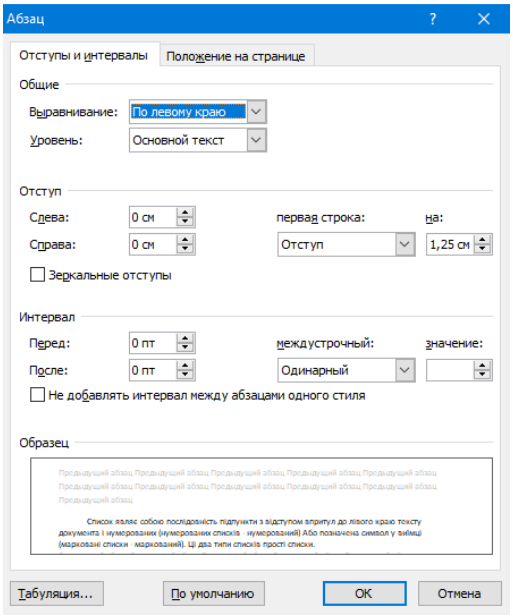
Місце, з якого вводиться текст, визначається положенням курсору. Після введення тексту, як правило, його редагують, тобто корегують, вилучають і переміщають слова, речення, абзаци та блоки, змінюють параметри шрифту тощо. Маніпулюючи клавішами *Backspace* та *Delete*, користувач може вилучати окремі символи або фрагменти тексту. У процесі редагування тексту виникає необхідність виділити (підсвітити) його фрагменти або весь текст, що здійснюється за допомогою натискання на кнопки миші або клавіатури.

Форматування абзаців у Microsoft Word.

Форматування тексту зазвичай починають із форматування абзацу. *Абзац* - це будь-який фрагмент документа, за яким розміщується маркер кінця абзацу 1|. Абзац вводять за допомогою клавіші Enter.

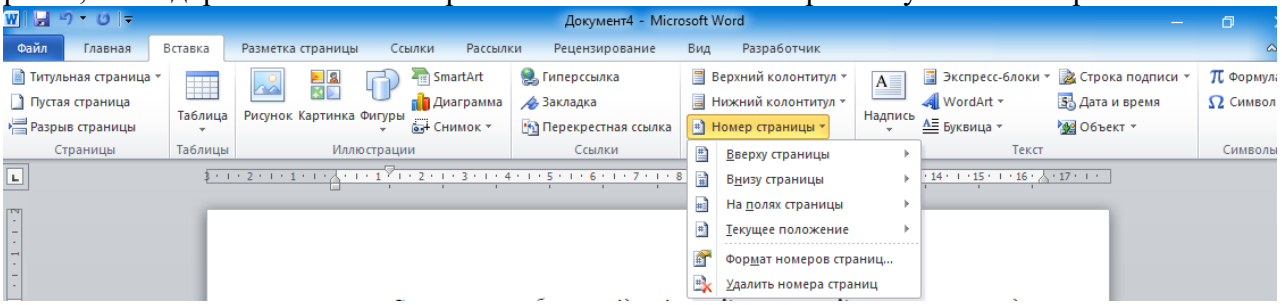
Форматування абзаців передбачає: вирівнювання абзаців; задавання відступів; установлення інтервалів між рядками й абзацами; контроль "висячих" рядків; форматування табуляцією та ін.

Заздалегідь виділені абзаци форматують за допомогою послуги Розмітка сторінки - Абзац, що активізує однойменне вікно з двома вкладками Відступи та інтервали та *Розташування на сторінці*.



Щоб надати тексту більшої виразності, окремі його абзаци та заголовки іноді вкладають у рамку з тінню і фоном. Таку операцію виконують послугою *Розмітка сторінки - Тло сторінки - Межі сторінок,* яка викликає на екран однойменне вікно з трьома вкладками *Межі, Сторінка* та Заливка.

Нумерація сторінок виконується за допомогою послуги Вставлення - Колонтитули - Номер сторінки, яка відкриває меню з запропонованими способами розташування номера.



Послугою Формат номера сторінки і її вікном користуються, щоб вибрати формат номера (арабські або римські цифри, латинські літери тощо), встановити початок нумерації (із зазначеного номера сторінки або продовжувати нумерацію).

При створенні документу автоматично утворюється один розділ (частина докумена, з певними значеннями параметрів форматування сторінок). Громіздкі документи поділяють на розділи, або створюють нові.

Розділи при необхідності зміни окремих параметрів сторінок (орієнтації, полів, колонтитулів, нумерації). Для створення нового розділу, необхідно скористатись послугою *Розмітка сторінки – Розриви-Розриви розділів*, попередньо розмістивши курсор в необхідному місці і обрати параметр, що вказує звідки починатиметься розділ. При цьому вставляється мітка з позначкою кінця розділу та відомостями про параметри його форматування, які можна змінювати та застосовувати лише до поточного розділу.

Автоматична нумерація та маркування абзаців.

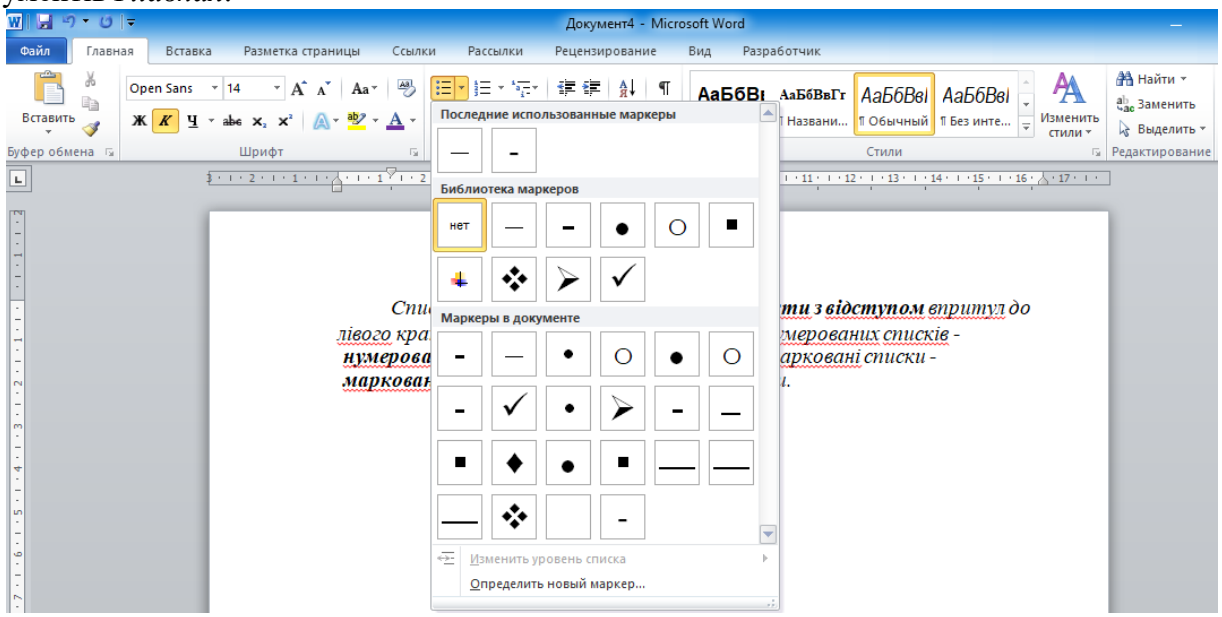
Список являє собою послідовність підпунктів з відступом впритул до лівого краю тексту документа і нумерованих (нумерованих списків – нумерований) або позначених символом (марковані списки - маркований). Ці два типи списків – прості списки.

Багаторівневий список (схема Номерні) – список, який містить в собі один або декілька інших списків.

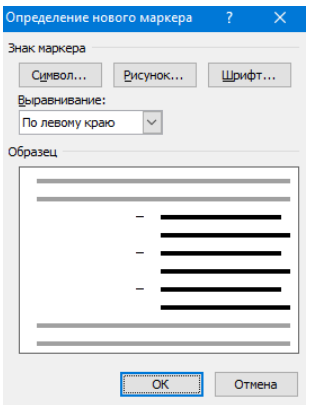
Списки створюють натисканням кнопки або вибравши конкретні варіанти:

– після введення і вибору потрібних пунктів;
– у перший пункті списку, після <ENTER> символ списку вводиться автоматично.

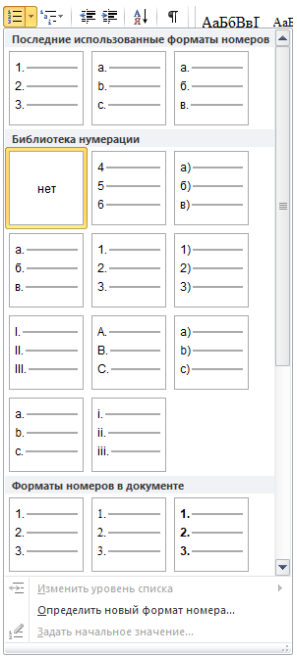
Маркований список може бути створений шляхом активації відповідної кнопки на панелі інструментів *Главная*.



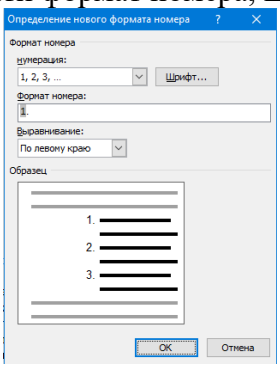
При необхідності можна змінити маркування за допомогою опції *Определить новый маркер* у цьому ж вікні. Це дозволяє, через відповідні поля, вибрати інший символ вставити малюнок, змінити шрифт та вирівнювання.



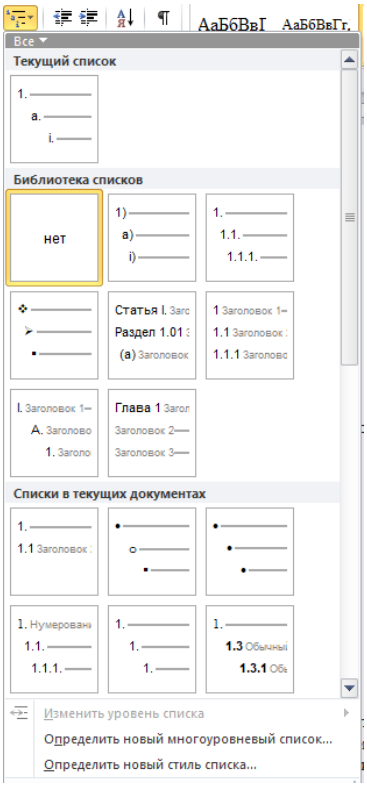
Нумерований список може бути створений за допомогою кнопки *нумерація* панелі інструментів *Главная*.



Вона містить десять визначених типів нумерації. В цьому ж вікні можна налаштувати нумерований список. Наприклад, вибрати формат номера, шрифт та вирівнювання списка.

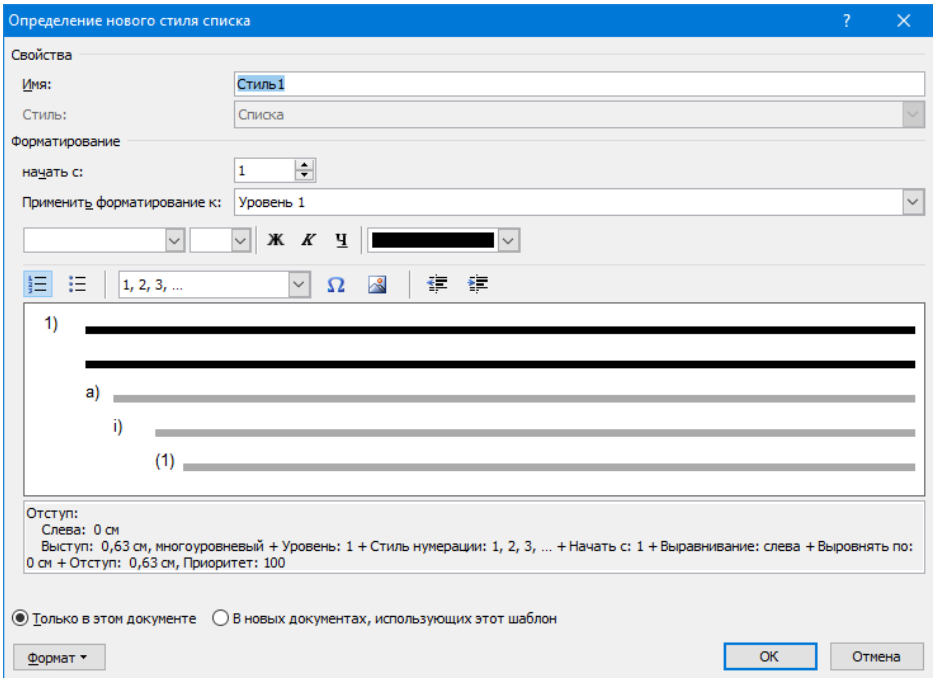


Багаторівневий список може мати до дев'яти рівнів. Поля можуть бути створені за допомогою кнопки на панелі інструментів *Главная*.

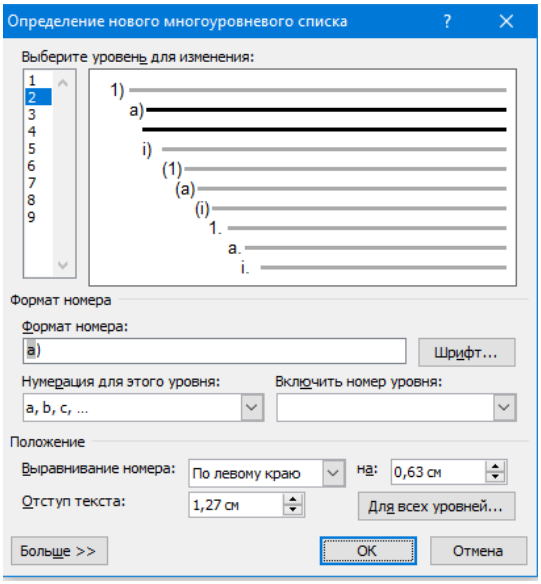


Він надає користувачам сім визначених типів таких списків, але список можливостей може бути змінений в діалоговому вікні *Определить новый многоуровневый список*.

Налаштування багаторівневого списку відбувається при виборі опції *Определить новый стиль списка*. Тут можна вибрати ім'я стилю та призначити йому необхідний спосіб форматування. В межах кожного рівня можна визначити зміст полів, формат номера, номер позиції і попередній перегляд.

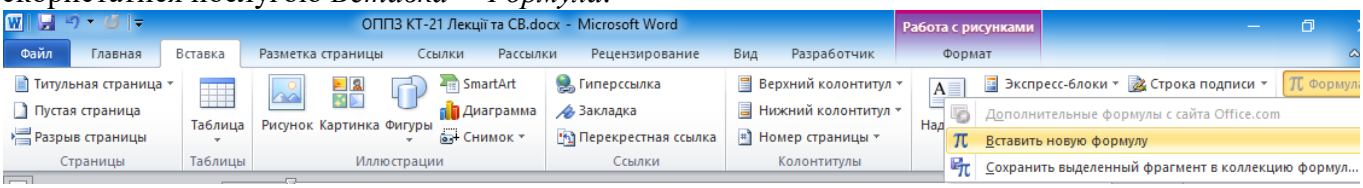


Різні рівні списку представлені різними вирівнюванням (у напрямку до лівого краю); для цієї мети можуть застосовуватися кнопки відступу на панелі інструментів форматування пунктів у списку.

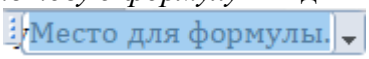


ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: СТВОРЕННЯ ТА РЕДАГУВАННЯ МАТЕМАТИЧНИХ ФОРМУЛ

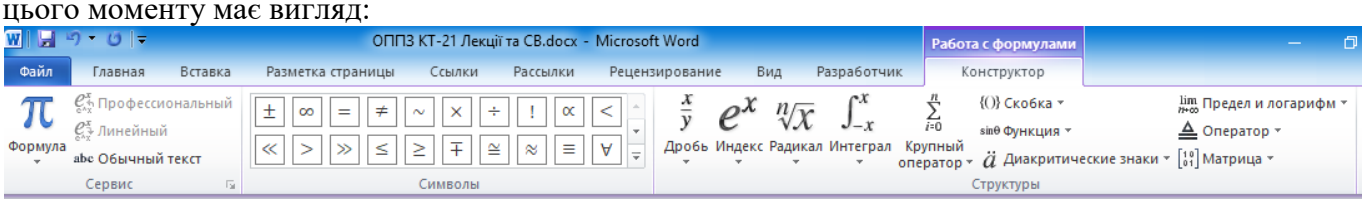
Для вставлення нової формули чи рівняння в текст, починаючи з позицій курсору, слід скористатися послугою *Вставка – Формула.*



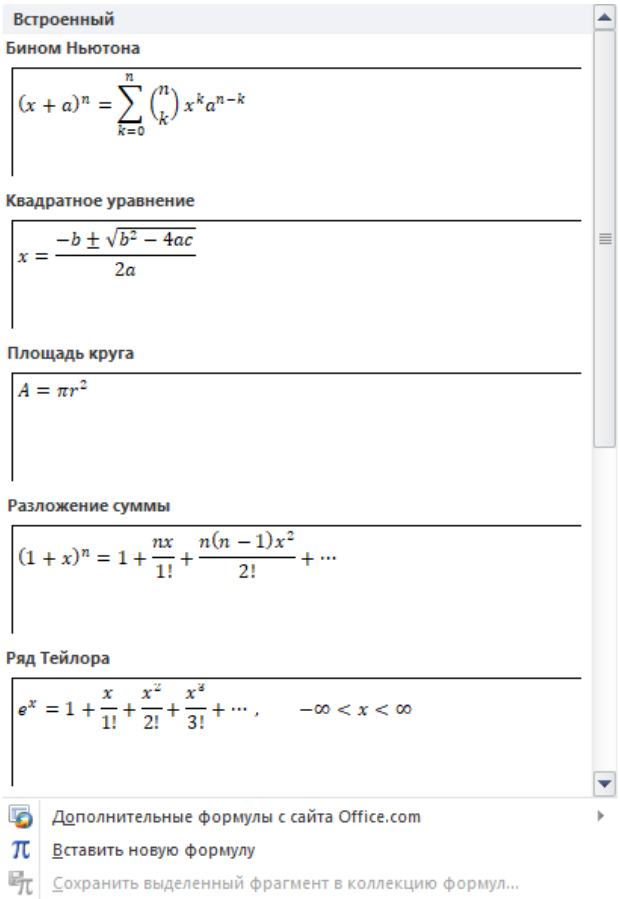
У вікні слід вибрати *Вставить новую формулу* . Тоді в тексті з'явиться вікно вигляду



В ньому слід набирати формулу, вибравши потрібний елемент на панелі інстументів, яка до цього моменту має вигляд:



Можна також обрати одну із вбудованих формул або переглянути ів вибрати формулу на сайті Office.com.



Для вставлення у формулу будь-якого символу, літери, оператора достатньо скористатися спадним меню *Конструктор – Символи – Основні математичні знаряддя,* обравши необхідну послугу.

Скориставшись *Параметрами формули* послуги *Знаряддя,* формулам надається формат, необхідний користувачу.

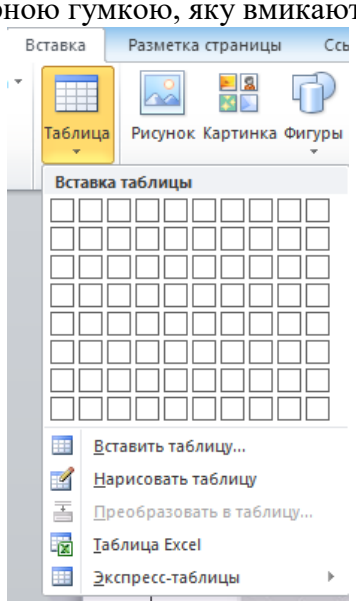
ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: ДІЛОВА ТА ІЛЮСТРАТИВНА ГРАФІКА У MICROSOFT WORD

Word надає користувачеві багатий набір засобів для швидкого створення двовимірних таблиць будь-якої складності і конфігурації. Він має засоби оброблення табличних даних.

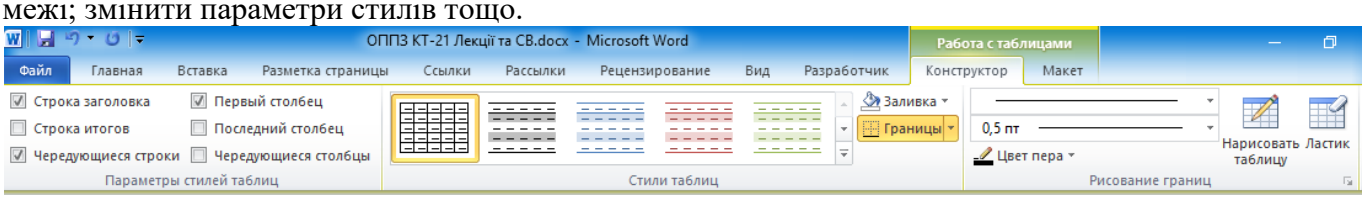
Двовимірні таблиці можна створити у такі способи.

1) За допомогою піктограми *Накреслити таблицю,* що знаходиться на панелі швидкого доступу і дозволяє олівцем намалювати необхідну таблицю. При цьому активується стрічка Табличні знаряддя.

Цю панель викликають на екран також, вдаючись до послуги Вставлення – Таблиця – Накреслити таблицю. Після цих дій покажчик миші набуває вигляду олівця. Для формування контуру таблиці необхідно встановити олівець на її початок і натиснути ліву кнопку миші. Далі штриховий прямокутник, що з'явиться на екрані, розтягнути до розмірів бажаної таблиці, утримуючи кнопку миші. Розмежувальні лінії рядків і стовпців таблиці проводять олівцем. Непотрібні лінії вилучають мініатюрною гумкою, яку вмикають кнопкою Гумка.



За допомогою інших груп команд *Конструктор* можна: задати стилі таблиці, її затінення, межі; змінити параметри стилів тощо.



Послугою *Вставлення – Таблиця – Вставити таблицю.*

2) Цією командою послуговуються, щоб викликати на екран вікно Вставлення таблиці, у якому задають кількість рядків та стовпців (автоматично – 5 і 2), а також ширину стовпця таблиці. Спочатку ширина всіх стовпців однакова, і таблиця займає все поле набору (Авто).

3) На основі наявного тексту

4) Word дає змогу досить просто перетворити текст на таблицю.

Для цього необхідно:

– Розділити текст на стовпці за допомогою знаку абзацу Ц (або клавіші Enter), табуляції (або клавіші Tab), крапки з комою або будь-якого символу, вибраного користувачем.

– Виділити перетворений текст і активізувати команду Вставлення – Таблиця – Перетворити на таблицю, яка викличе на екран вікно *Перетворити на таблицю.*

– Встановити кількість стовпців таблиці у цьому вікні, зазначити вид роздільника та активізувати команду ОК.

4) За допомогою послуги *Вставлення - Таблиця - Вставлення таблиці.*

При цьому з'являється спадаюче меню у вигляді клітинок майбутньої таблиці, кількість яких можна змінювати розтягуванням за допомогою натиснутої лівої клавіші миші.

Введення даних до таблиці та їх форматування

Дані вводять у клітинки таблиці, починаючи з позиції курсору. В міру заповнення клітинки її розміри по вертикалі автоматично збільшуються. Переміщення між комірками таблиці здійснюється за допомогою миші або різних комбінацій клавіш керування курсором, які наведені нижче.

Комбінації клавіш	Виконувана операція
Tab	Перехід до наступної клітинки
Shift+ Tab	Перехід до попередньої клітинки
Ctrl+ Tab	Вставлення символу табуляції
Alt+Home	Перехід до першої клітинки рядка
Alt+End	Перехід до останньої клітинки рядка
Alt+PgUp	Перехід до верхньої клітинки стовпця
Alt+PgDn	Перехід до нижньої клітинки стовпця

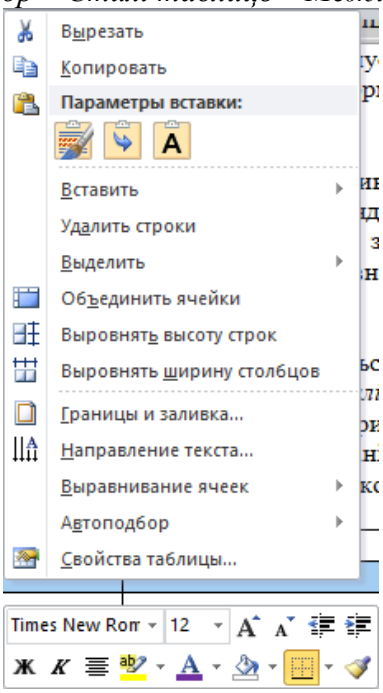
Форматування табличних даних виконується аналогічно форматуванню звичайного тексту. Спочатку їх виділяють, а потім форматують, використовуючи команди *Основне - Шрифт/Абзац/Стилі* тощо.

Редагування таблиці

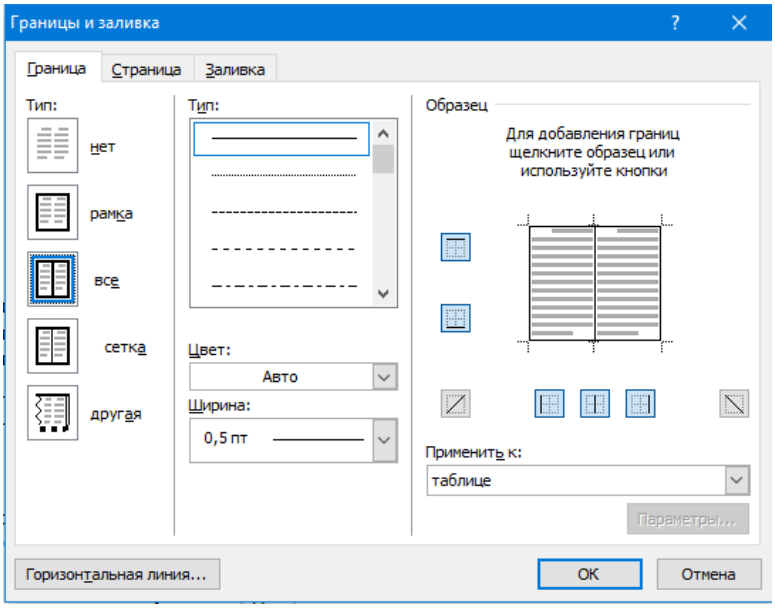
Таблиці редагують, щоб надати їм привабливішого і досконалішого вигляду. Редагування включає: зміну ширини стовпців та висоти рядків; вставлення окремих клітинок, рядків, стовпців і вилучення їх; форматування даних таблиці; зовнішнє оформлення таблиці тощо. Усі процедури редагування виконують при виділених певних елементах таблиці натисненням правої клавіші миші

Оформлення таблиць

Для оформлення таблиць користуються засобами відомого вікна Межі й заливка, що активізується послугою *Конструктор – Стилі таблиць – Межі* та тіні.



Для обрамлення таблиці можна використовувати штрихові, одинарні, подвійні, потрійні, напівжирні, комбіновані й інші лінії. Тип ліній вибирають зі списку *Стиль,* товщину – зі списку Ширина. Лініям можна надати певного кольору (зазвичай вони чорні), вибравши його зі списку *Колір.*

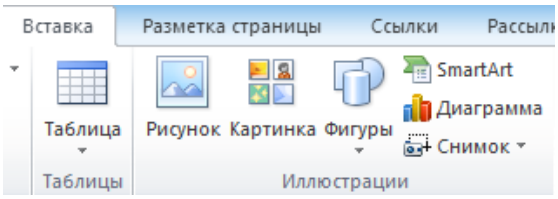


Останній штрих в оформленні таблиці додають, застосовуючи заливання її клітинок, рядків або стовпців. Колір заливання можна встановити, скориставшись вкладкою *Заливка* або *Конструктор – Стилі таблиць – Затінення.*

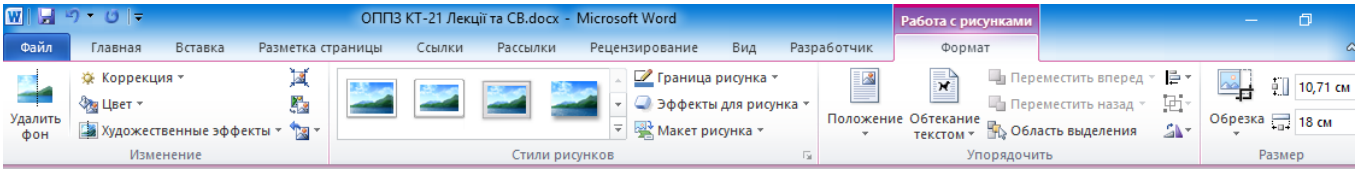
Вставка зображень у документ

Вставлення різних ілюстрацій виконується послугою Вставлення - Зображення, на стрічці якої відображається набір команд для вибору потрібного графічного об'єкта.

За допомогою послуги *Графіка* можна відкрити вікно *Microsoft Clip Gallery* з набором кольорових малюнків.

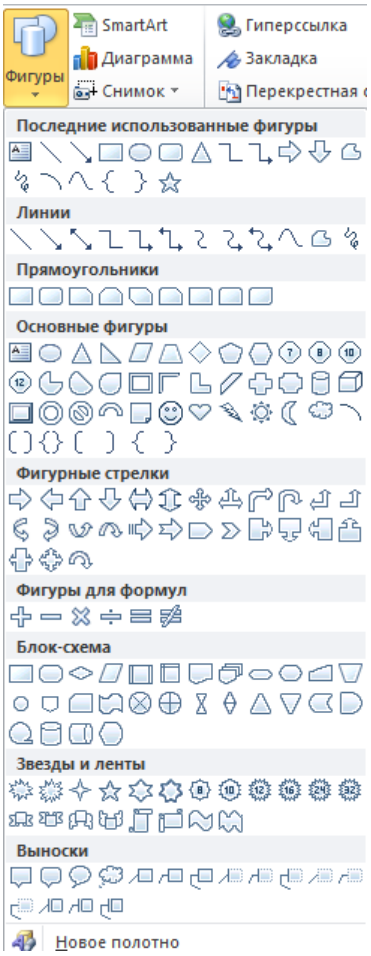


Будь-який з них можна вставити у документ на місце курсору. Зображення можна редагувати, тобто змінювати розмір і колір, додавати та забирати окремі елементи і навіть розбирати на складові частини, перефарбовувати, задавати певні форми, ефекти та розташування.



Обравши послугу *Рисунок* можна вставити у документ раніше підібрані малюнки.

Скориставшись послугою *Фігури* на екрані відкривається підменю, що містить множину вмонтованих геометричних фігур, розділених на сім груп: *Нещодавно використані фігури, Основні фігури, Фігурні стрілки, Блок-схема, Виноски, Зірки та стрічки, Створити полотно*. Їх використовують залежно від потреби користувача. При виборі послуги *Створити* полотно у документі виділяється певна його частина, при цьому активуються стрічка *Засоби малювання – Формат*, яка дозволяє вставити фігури, задати їм певного стилю (залити різними способами, задати контури чи змінити саму фігури, змінити формат полотна), додати тіньових ефектів та за необхідністю перетворити фігуру на об'ємну, зміни розміри й впорядкувати відносно тексту.



Послуга SmatArt, з меню *Вставлення – Зображення,* дає змогу вставити об'єкт для візуального сприйняття інформації. Обравши потрібний рисунок, активується додаткова стрічка знарядь для рисунків.

За допомогою послуги *Вставлення – Зображення – Діаграма* на екрані відкривається вікно Вставлення діаграми із зображеннями різних типів діаграм.

Послідовність вставлення в текст графічних об'єктів залежить від їх характеру та середовища створення.

Найпростіше у Word-тексти вставляти робочі вікна. Для цього необхідно:

– викликати на екран монітора потрібне вікно;
– натисненням на клавіші *Alt+PrtScr* занести його в буфер обміну Clipboard;
– активізувати текстовий файл;
– послугою *Правка – Вставити* вставити графічний об'єкт із буфера пам'яті на позицію курсору;
– виділити межу-кадр вставленого вікна, задати його розміри та розмістити в потрібному місці.

Таке вставлення робить положення вікна в тексті фіксованим, оскільки воно вважається в системі окремим символом.

Аналогічно у Word-текст можна вставляти й об'єкти, сформовані безпосередньо в додатках до Windows. Будь-який із таких об'єктів спочатку виділяють і, послуговуючись командою *Копіювати (Вирізати),* заносять у буфер обміну. Потім командою Основне - Вставити - Спеціальне вставлення добавляють у текст. Зазвичай графічні об'єкти вставляють у текст послугою *Основне – Вставити.*

У текстовому процесорі Word є можливість вводити математичні об'єкти (формули та рівняння) до тексту документа. Ці математичні об'єкти вводять до тексту і редагують безпосередньо в ньому або в спеціальному вікні.

ЛЕКЦІЯ 8. ОСНОВНІ ХАРАКТЕРИСТИКИ MICROSOFT EXCEL.

8.1 Поняття про табличний процесор. Інтерфейс програми MS EXCEL. Вікно робочої книги

Microsoft Excel – потужний табличний процесор, створений фірмою Microsoft Corporation з використання найновіших програмних технологій.

Табличний процесор – це пакет програм, призначений для створення, редагування і обробки електронних таблиць (ЕТ). На екрані дисплея ЕТ подається у вигляді матриці, що складається зі стовпців та рядків, на перетині яких утворюються комірки. Стовпці і рядки мають ідентифікатори, тому кожну комірку можна визначити однозначно. У комірках розміщують текст, числа, формули, що задають залежність однієї комірки від іншої.

Файли, з якими працюють в MS Excel і де зберігаються дані (вони мають розширення .xls), називають робочими книгами**.**

Робоча книга – це набір декількох електронних таблиць, об'єднаних в одному файлі. Кожна така таблиця називається *робочим аркушем***.** Аркуші поділяються на:

– робочі аркуші – звичайні аркуші з електронними таблицями;

– аркуші модулів – аркуші, на яких пишуть програми мовою Visual Basic for Applications;

– аркуші діаграм – аркуші, на яких за допомогою майстра діаграм можна будувати діаграми.

Книга може складатись з одного або кількох робочих листів. Перехід від одного до іншого здійснюється при активізації ярликів. Сама таблиця складається із стовпців і рядків, перетин яких дає комірку. Кожна комірка має свою адресу – ім'я стовпця та номер рядка. На робочому аркуші Excel існує 65536 рядки, які іменуються арабськими цифрами. Кожен робочий аркуш Excel містить 256 колонок, які іменуються латинськими літерами. (А4, К18, CZ6).

Діапазон комірок **–** це адреси виділених комірок. Формат:

<Комірка1> : <Комірка2>

<Комірка1> – адреса лівого верхнього кута прямокутної ділянки, що займають виділені комірки (діапазон комірок).

<Комірка2> – адреса правого нижнього кута прямокутної ділянки, що займають виділені комірки (діапазон комірок).

Існує багато галузей застосування табличних процесорів: інженерні розрахунки, математичне моделювання економічних процесів, статистична обробка масивів даних, фінансові розрахунки, керування базами даних тощо. Використовуючи Excel, на підприємстві можна розраховувати заробітну плату і податки, вести облік кадрів і витрат, планувати виробництво та управляти збутом.

До переваг табличного процесора Excel відносять:

– Велику кількість функціональних можливостей для створення якісних таблиць і графіків;

– Зручний інтерфейс для користувача;

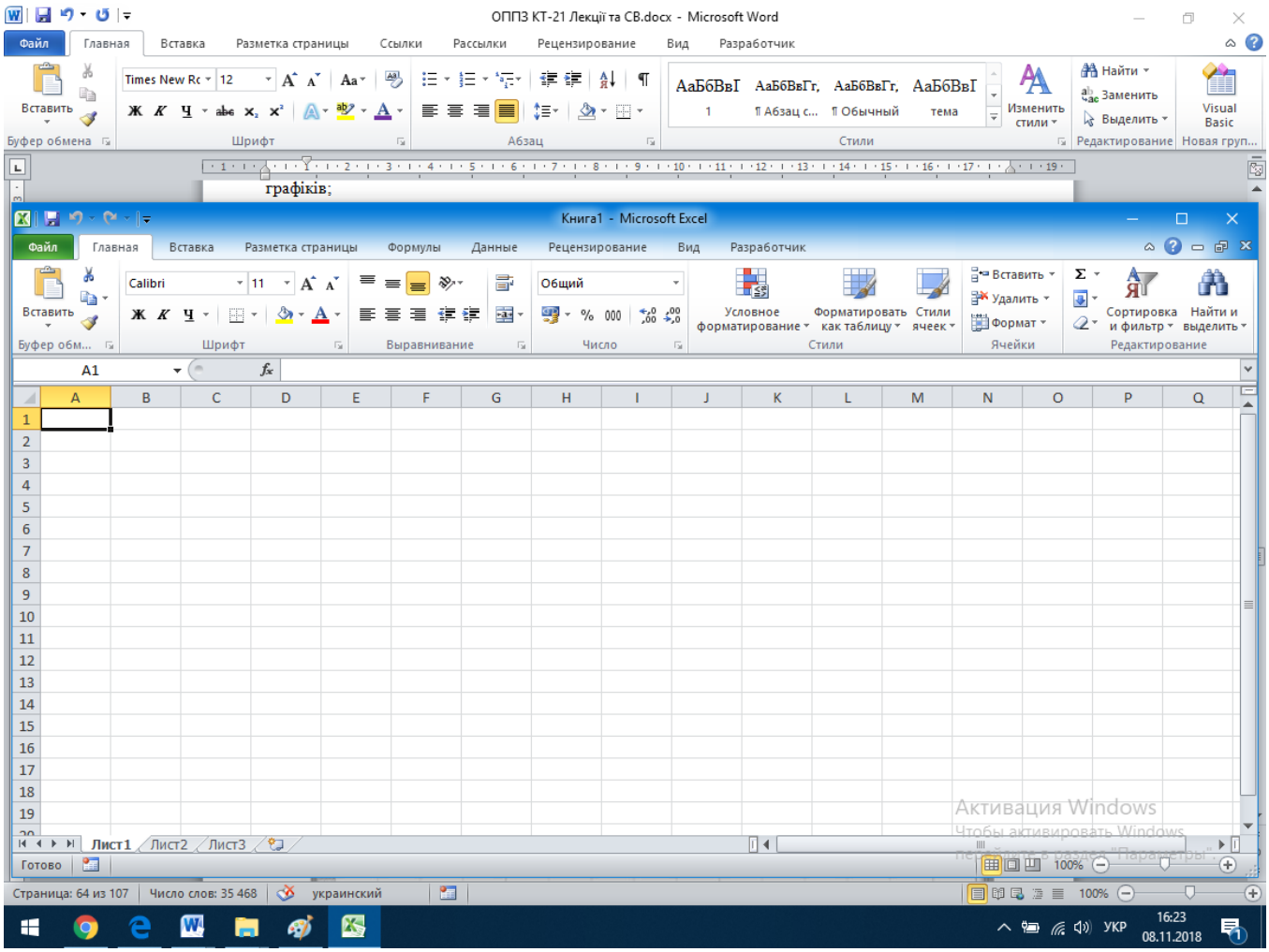
– Приналежність до сім'ї MS Office, що дає можливість обробки різноманітної інформації;

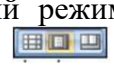
– Вбудовану систему програмування Visual Basic for Applications, яка дозволяє досить просто створювати мікропрограми для автоматизації ЕТ.

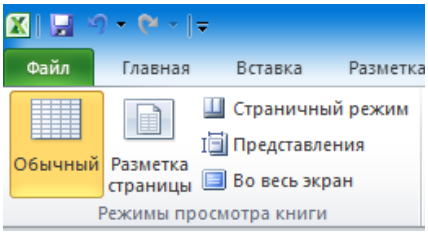
Для завантаження Excel, як і будь-якої програми, що входить до пакету Microsoft Office потрібно скористатись послугою Пуск; в головному меню системи вибрати команду Програми - Microsoft Office – Microsoft Excel. Можна також скористатись ярликом Excel, якщо він є на робочому столі, двічі натиснути ліву клавішу миші на ньому.

Для створення ярлика Excel належить: за допомогою послуги Пуск в головному меню системи вибрати команду Програми; в каскадному меню накласти покажчик курсору на піктограму *Microsoft Excel,* натиснути клавішу Ctrl і, не відпускаючи її, перетягнути піктограму на вільне місце робочого столу.

Після завантаження на екрані з'явиться основне вікно Excel.



Щоб отримати новий режим, необхідно скористатись піктограмою *Макет сторінки* на панелі інструментів Вигляд  внизу вікна з правого боку. Також можна відкрити вкладку *Вид* і обрати послугу *Макет сторінки* у групі Режими перегляду книги. У режимі макета сторінки аркуш з усіх боків оточено білими полями сторінки, а окремі аркуші відокремлюються один від одного вузькою синьою смугою.



Угорі та ліворуч містяться лінійки, за допомогою яких можна відрегулювати розмір полів. Якщо лінійки не потрібні, їх відображення можна вимкнути (кнопка ***Лінійка*** у групі Відобразити або приховати на вкладці Вид). Завдяки цьому новому режиму немає потреби вмикати попередній перегляд друку, щоб скоригувати вигляд аркуша перед друкуванням.

Стрічка Excel має у своєму складі дев'ять вкладок: *Основне, Вставлення, Розмітка сторінки, Формули, Дані, Рецензування, Вигляд, Надбудови.* Кожна з них виконує певні функції, містить по декілька груп елементів, схожих за функціями. Найголовніші команди зібрано на вкладці *Основне.* Це команди, які, за дослідженнями Майкрософт, найчастіше використовуються для виконання елементарних дій з аркушами.

Для перегляду значного обсягу даних збільшено кількості рядків і стовпців *у Microsoft Office Excel:* підтримується до 1 мільйона рядків і до 16 тисяч стовпців на одному робочому аркуші. Зокрема, сітка Office Excel 2007 має розмір 1 048 576 рядків на 16 384 стовпці. Під час роботи на екрані монітора відображається тільки частина електронної таблиці. Переміщуючи вікно, можна переглядати необхідні частини таблиці і таблицю в цілому.

Рядки і стовпці створюють *клітинки.* Декілька клітинок утворюють діапазон клітинок. Клітинка таблиці, в якій перебуває курсор, називається робочою, поточною. Робоча клітинка має контрастне обрамлення. Тільки в неї користувач може вводити потрібні дані (число, текст або формулу).

Табличний курсор – робоча, поточна клітинка, виокремлена рамкою.

Автоматично початкова робоча книга складається з 3 аркушів зі стандартними іменами *Аркуші, Аркуш2, Аркуш 3.* Їх «перегортають» за допомогою однойменних кнопок-ярликів, які розміщені у нижній частині екрана. Стандартні імена аркушів можна поміняти на більш інформативні таким чином: двічі натиснути на стандартному імені Аркуш 1 (або *2* чи 3); ввести нове ім'я; натиснути клавішу *Enter*. Або: натиснути правою клавішею миші на імені аркуша; вибрати в меню, що відкриється, команду *Перейменувати;* ввести нове ім'я; натиснути клавішу *Enter*. За допомогою того ж меню можна: *Додати, Видалити, Перемістити або копіювати Захистити аркуш, Приховати, Переглянути код, Виділити всі аркуші,* а також задати *Колір вкладки*.

8.2 Операції автозаповнення і автоповтору

Автозаповнювання та автоповторення – гнучкий і зручний інструмент автоматичного введення числових і текстових даних, що змінюються або копіюються в межах заданого інтервалу.

До таких даних належать порядкові номери, послідовність цілих чисел, дати, дні тижня, місяці року та ін.

Автозаповнювання реалізується однойменною програмою і виконується користувачем таким чином:

– до вибраного елемента таблиці вводять перше значення початкового інтервалу, наприклад, «Понеділок»;

– покажчик миші поєднується з маркером заповнення і перетворюється на чорний хрестик;

– «буксуванням» нового покажчика виділяють діапазон клітинок стовпця або рядка, який за розміром відповідає заданому інтервалу даних.

Перехід до режиму автозаповнювання здійснюється послугою Основне – Редагування – Заповнити – далі за вибором самого користувача.

8.3 Імена комірок і діапазонів. Правила запису формул

Для виділення *будь-якої клітинки* робочого аркуша, наприклад, клітинки*A1,* достатньо помістити в неї курсор і натиснути ліву кнопку миші. Поява жирної рамки навколо клітинки свідчить про те, що вона стала робочою і до неї можна вводити дані або формулу. Посилання на виділену клітинку відображається в панелі імені робочого аркуша.

На рисунку відображено варіанти виділення клітинок робочого аркуша Excel.

Для того щоб виділити *множину клітинок* окремого рядка або стовпця, потрібно натиснути ліву кнопку миші на номері відповідного рядка або стовпця. Посилання на виділений рядок або стовпець відображається в панелі імені робочого аркуша у вигляді адреси першої клітинки рядка або стовпця.

Для того щоб виділити *діапазон суміжних клітинок* робочого аркуша, потрібно помістити курсор в першу клітинку, клацнути мишею і протягнути курсор до останньої клітинки. Або помістити курсор в першу клітинку, натиснути ліву кнопку миші і утримуючи клавішу Shift, помістити курсор в останню клітинку й знову натиснути ліву кнопку миші. Можна також скористатись комбінаціями клавіш *Shift* + ← - ↑-↓- →. Посилання на виділений діапазон клітинок відображається в панелі імені робочого аркуша у вигляді адреси першої клітинки діапазону.

Для того щоб виділити кілька *несуміжних клітинок* або *діапазонів* робочого аркуша, потрібно скористатися «буксуванням» покажчика миші при натиснутій клавіші Ctrl.

Для виділення *всього робочого аркуша* досить натиснути ліву кнопку миші на перетині заголовків стовпців і рядків.

У табличному процесорі Excel можна виконувати з даними безліч різних операцій – математичних, логічних, статистичних, текстових, фінансових та ін.

Формула – записана послідовність дій з операндами. Будь-яку формулу, як і текст або число, вводять до вибраної клітинки робочого аркуша вручну. Кожна формула, що

використовується для обчислень в Excel, починається зі знаку «дорівнює». Формула повністю *відображається* в рядку формул і легко редагується.

До формули можна також включати імена стандартних функцій, вибираючи їх зі спеціальної вкладки *Формули – Бібліотека функцій*.

Вікно *Вставлення функції* автоматизує процес введення формул, залишаючи за користувачем тільки вибір функції та введення деяких констант.

Введення функції завершується натисканням клавіші Enter.

В Excel формули можна *копіювати* з автоматичним настроюванням їх за новим місцеположенням. Цю процедуру виконують або «буксуванням» клітинки з формулою, або за допомогою послуг Копіювати – Вставити. Копіювання формули – *це процес поширення дії формули, введеної в одну клітинку, на інші клітинки.*

При зміні вхідних даних результати у всій таблиці будуть перераховуватись автоматично. Копіювання формул та автоматичне переобчислення табличних даних – основні засоби автоматизації обчислень у електронних таблицях.

8.4 Абсолютні та відносні адреси комірок. Посилання на комірки

У формулах для посилання на відповідні значення використовують адреси клітинок. В Excel використовують два типи адрес (посилань) клітинок: відносні та абсолютні.

Відносні адреси – це адреси, які в процесі копіювання змінюють своє значення (посилання на іншу клітинку) відповідно до нової позиції формули при її копіюванні. Їх адреси позначаються звичним чином та використовувати їх не завжди зручно.

Абсолютні адреси – адреси, які під час копіювання не змінюють своє значення відповідно до нової позиції формули при її копіюванні. Позначаються абсолютні адреси символом $ і застосовуються, якщо у формулу треба ввести значення з фіксованої клітинки. Під час переміщення (копіювання) формул абсолютні адреси залишаються незмінними.

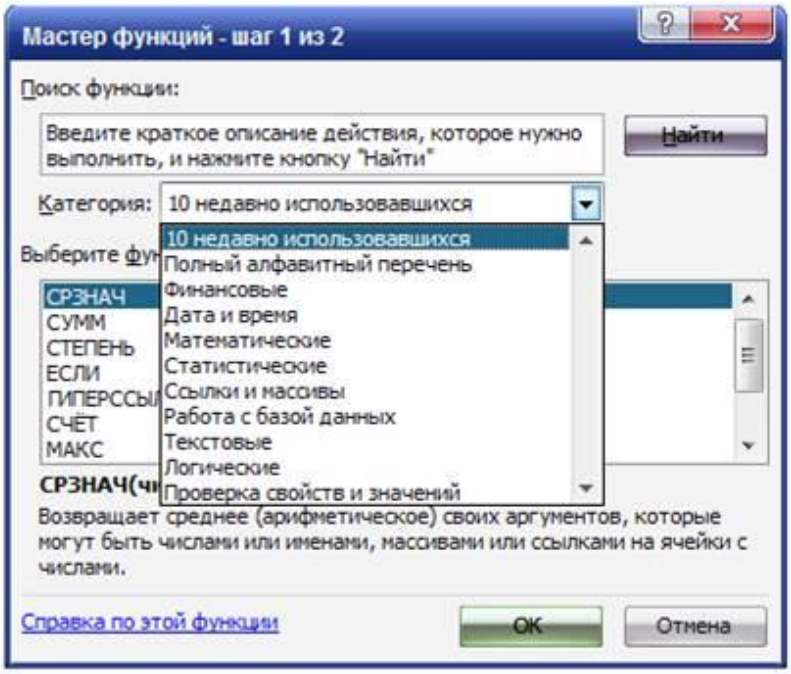
Якщо посилання відображається записом адрес крайніх клітинок певної частини стовпця, рядка чи аркуша, говорять що це посилання на інтервал клітинок.

Інколи, для зручності, замість посилань на клітинки чи їх інтервали використовують не адреси, а імена (умовно присвоєні позначення), які не містять пробілів, спеціальних символів та розділових знаків. Відображаються всі створені імена у списку імен, який належить книзі, що дозволяє виконувати посилання на ім'я на довільному аркуші. При переміщенні таких інтервалів формула імені налагоджується автоматично у відповідності до розміщення інтервалу.

ТЕМА САМОСТІЙНОГО ВИВЧЕННЯ: МАЙСТЕР ФУНКЦІЙ EXCEL. ПОБУДОВА ДІАГРАМ. ФІЛЬТРАЦІЯ ДАНИХ

Майстер функцій

Майстер функцій – це спеціальна програма, за допомогою якої можна вибрати потрібну функцію і виконати її, вказавши всі потрібні параметри. Майстер функцій можна викликати таким чином: натискання кнопки *Мастер функций* (f_x), що розміщена на панелі інструментів *Стандартная*. Вікно *Мастера функций* складається з трьох частин .



У першій можна ввести опис дії, яку необхідно виконати і натиснути кнопку *Найти*. Цей метод використовується, якщо користувач не знає чи не пам'ятає, як називається потрібна йому функція. Нижче є поле для вибору категорії функцій. Для спрощення роботи з великим обсягом вбудованих функцій всі вони розділені на категорії залежно від призначення. Це значно спрощує пошук потрібної функції. Є окремо виділені категорії: *10 недавно используемых функций* та *Полный алфавитный перечень* для спрощення пошуку функцій. У третій частині є можливість вибору функцій відповідно до категорії. При виборі функцій в нижній частині вікна відображається коротка інформація про призначення цієї функції. Після вибору функції з'являється ще одне діалогове вікно для визначення аргументів визначеної функції. Це можна зробити шляхом введення потрібних даних із клавіатури або безпосередньо зазначенням адреси у таблиці за допомогою миші. Верхня частина вікна містить перелік аргументів та поля для їх введення. У нижній частині – короткий опис функції. Якщо розмістити курсор мишки в полі для введення деякого аргументу, в нижній частині з'являється пояснення до цього аргументу та його тип. Всі обов'язкові аргументи виділені напівжирним шрифтом.

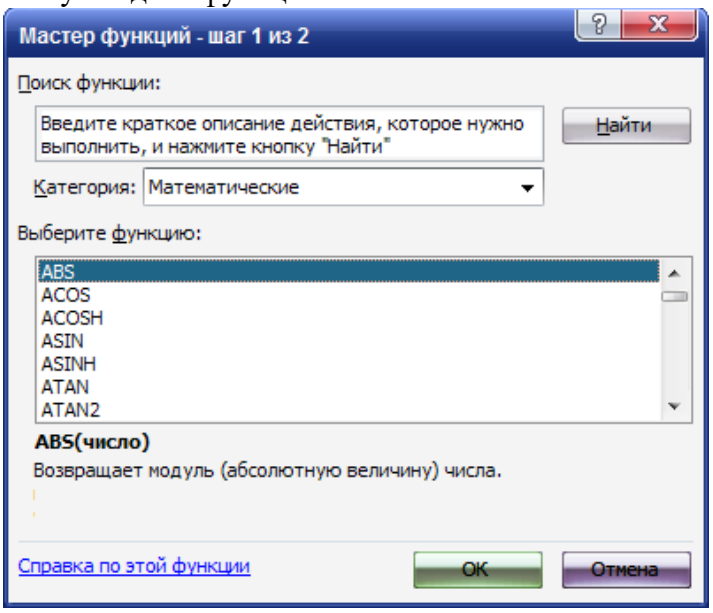
Довідку про необхідну функцію можна одержати, якщо вибрати її зі списку у довідковій системі Excel. Довідку про функції під час її введення в комірку робочого листка можна одержати за допомогою *Помощника*. Якщо ви знаходитеся у вікні діалогу *Мастера функций*, то на панелі інструментів необхідно вибрати інструмент, який позначається знаком «?» для виклику *Помощника*. Якщо ви вводите формулу безпосередньо в комірку робочого листа, то просто наведіть курсор мишки в рядок формул та натисніть F1. Під час діалогу деталізуйте тему довідки – введіть ім'я функції або її частину. У наступному діалоговому вікні необхідно виділити одну з запропонованих функцій, початок яких збігається з введеним фрагментом.

Математичні функції використовують різноманітні математичні дії. Вони спрощують різного роду математичні обчислення, наприклад арифметичні та тригонометричні. Розглянемо деякі із них.

СУММ – додає аргументи.

КОРЕНЬ – повертає додатне значення квадратного кореня.
COS, SIN, TAN – тригонометричні функції cos, sin і tg.
ACOS, ATAN – зворотні тригонометричні функції arccos, arctg.
ГРАДУСЫ – перетворює радіани в градуси.
LN – натуральний логарифм числа.
ABS – модуль числа.
ПИ – повертає число Пі (π=3.14).
ЗНАК – повертає знак числа.
ПРОИЗВЕД – повертає добуток аргументів.
СТЕПЕНЬ – повертає результат піднесення до степеня.
12 ОКРУГЛ – закруглює число до заданої кількості десяткових розрядів.
ОСТАТ – повертає залишок від ділення.
СЛЧИС – повертає випадкове число в інтервалі від 0 до 1.
РИМСКОЕ – перетворює число в арабському записі до числа в римському як текст.
СУММЕСЛИ – повертає суму вмісту комірок, яке задовольняє заданий критерій;
СУММКВ – повертає суму квадратів аргументів.
МОБР, МУММНОЖ, МОПРЕД – зворотна матриця, добуток та визначник матриці.

В електронній таблиці Excel вибрати математичні функції можна з використанням *Мастера функций*, де в полі Категорія необхідно вибрати *Математические* і тоді можна буде вибрати необхідну математичну функцію. Якщо виділити курсором мишки будь-яку функцію, то внизу буде написано, що розраховує подана функція та її синтаксис.



Функції категорії *Математические*

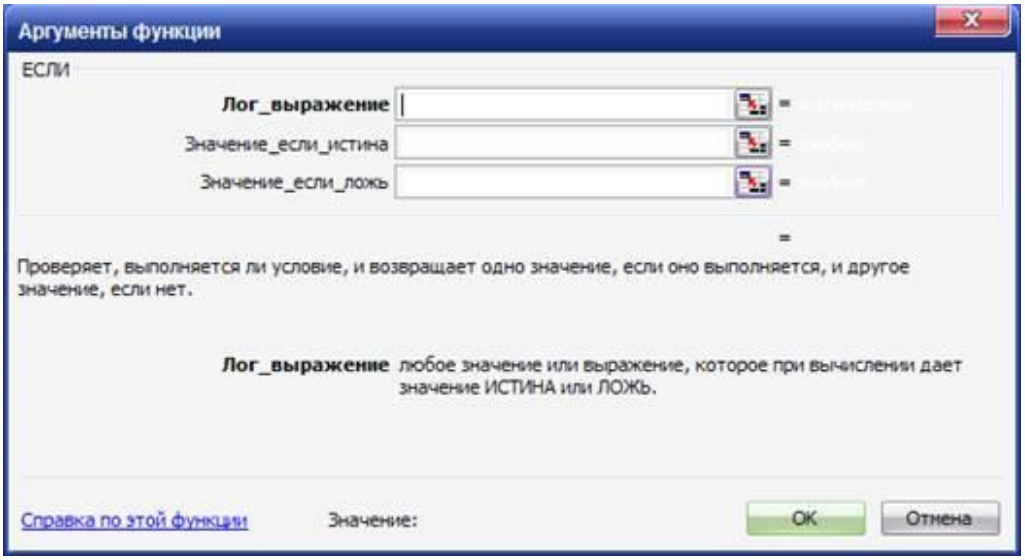
Окрему групу становлять функції призначені, призначені для роботи з матрицями. В їх застосуванні є особливості: аргументами таких функцій є діапазон комірок. При введенні функцій, аргументами яких є масиви (матриці) і які повертають як результат матрицю, необхідно перед введенням функції виділяти не одну комірку, куди буде розміщений результат, а діапазон.

Статистичні функції призначені для проведення статистичного аналізу. Крім того, їх можна використовувати для факторного та регресійного аналізу. Спочатку розглянемо найуживаніші:

СРЗНАЧ – визначає середнє значення.
МИН, МАКС – визначає мінімальне та максимальне значення.
СЧЕТ – визначає кількість числових аргументів.

Ці функції винесені на панель інструментів *Стандартная*.

Логічні функції допомагають створити складні формули, що залежно від виконання тих, чи інших умов, роблять різні види обробки даних. Ці функції приймають логічні значення «Істина» або «Хибно».



Якщо при обчисленні формули сталася помилка, то в комірку виводиться повідомлення про помилку, яке починається із символу #. Excel виводить такі повідомлення про помилки.

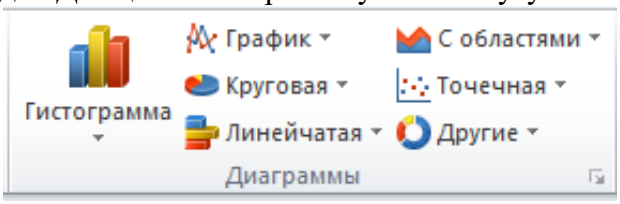
Повідомлення про помилку	Пояснення
#дел0	Спроба поділити на нуль або на порожню комірку
#имя ?	Формула використовує неіснуюче ім'я
#н/д	Формула посилається на комірку з невизначеними даними
# число !	Помилка в числі, число неможливо подати в Excel
# ссьл !	Формула посилається на неіснуючу комірку
# знач !	Помилка при обчисленні функції

Поняття про діаграми

Табличний процесор Excel дає змогу подавати табличні дані в наочній та зручній для сприйняття графічній формі. Такі ілюстрації використовують для показу функціональної залежності однієї величини від іншої або для порівняння двох і більше величин тощо.

Діаграма – графічне відображення числових даних.

Табличний процесор Excel дозволяє побудувати 12 стандартних типів діаграм, кожен із яких має ще кілька різновидів. Для цього використовують послугу *Вставлення – Діаграми*.



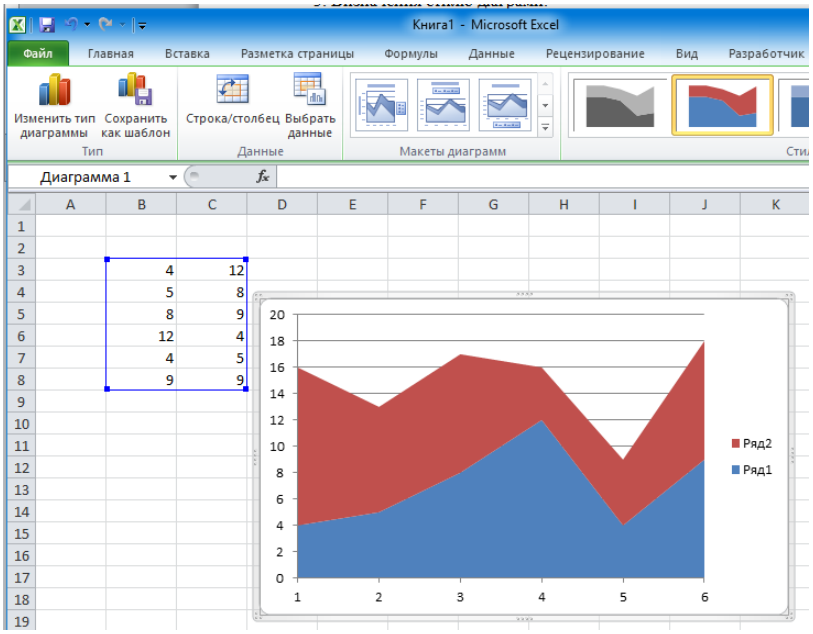
Створення будь-якої діаграми розпочинається з виділення діапазону даних, що підлягають відображенню на ній. Початковий діапазон даних можна виділяти пізніше. Його попереднє виділення пояснюється тільки прагненням мати зразок діаграми вже після вибору її типу та вигляду.

Процес створення діаграм за вкладки *Діаграми* складається з п'яти характерних кроків, які відображені у відповідних групах стрічки *Діаграми*.

Вибір типу та вигляду діаграми, а також перегляд її зразка.

1. Зміна або вибір діапазону даних, на основі яких буде побудовано діаграму, та визначення способу формування її рядів.
2. Вибір необхідного макету діаграми.
3. Визначення стилю діаграми.
4. Вибір варіанту розташування діаграми (на поточному або на окремому аркуші).

При правильному призначенні початкового діапазону даних послуга *Діаграма* здатна сформувати її практично за перший крок, оскільки багато параметрів призначаються за замовчуванням.



Процес створення графіка або діаграми розпочинається з активізації послуги Діаграми та обрання її типу. Якщо тип обраної діаграми незадовільняє користувача, його можна змінити, скориставшись послугою *Знаряддя для діаграм – Конструктор – Тип – Змінити тип діаграми*.

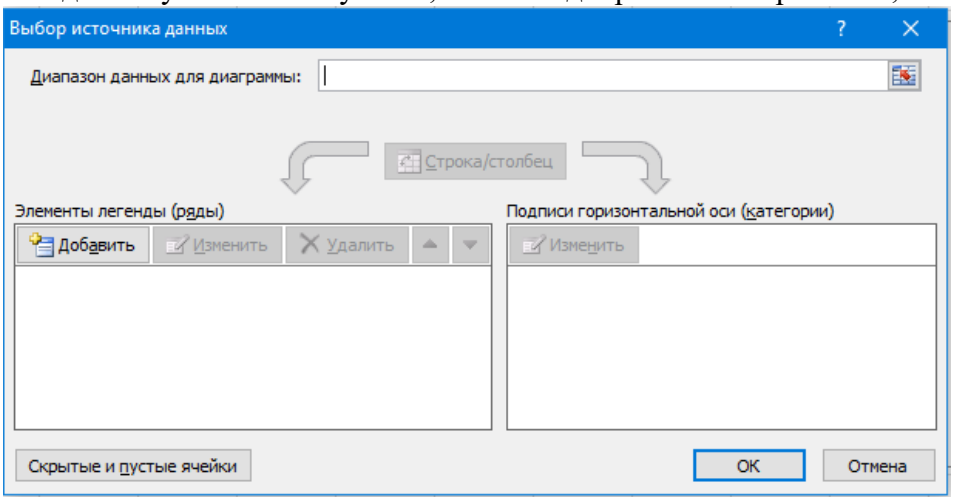
Після чого із 11 запропонованих типів, а саме: стовпчаста, лінійчата, секторна, гістограма, з областями, точкова, біржова, поверхнева, кільцева, пелюсткова бульбашкова, обирається необхідний тип. За умови, що певний тип був раніше створений і збережений у шаблонах, користувач може обрати його із меню *Шаблони*, що міститься у групі *Змінити тип діаграми*. Якщо ж користувачем створено діаграму з особливими параметрами, яку необхідно використовувати надалі, то її доцільно зберегти як шаблон, скориставшись послугою *Конструктор – Тип – Зберегти як шаблон*.

Зміна або вибір діапазону даних, на основі яких буде побудовано діаграму, і визначення способу формування її рядів здійснюються через групу *Дані*, що знаходиться у вкладці Конструктор.

Скориставшись піктограмою *Перехід рядок/стовпець* – зміна способу формування рядів діаграми здійснюється автоматично і одразу відображається на аркуші.

При виборі піктограми Вибір даних, відкривається вікно *Вибір джерела даних*, яке дозволяє обрати діапазон даних діаграми, здійснити перехід рядок/стовпець, редагувати записи легенди та підписи осей.

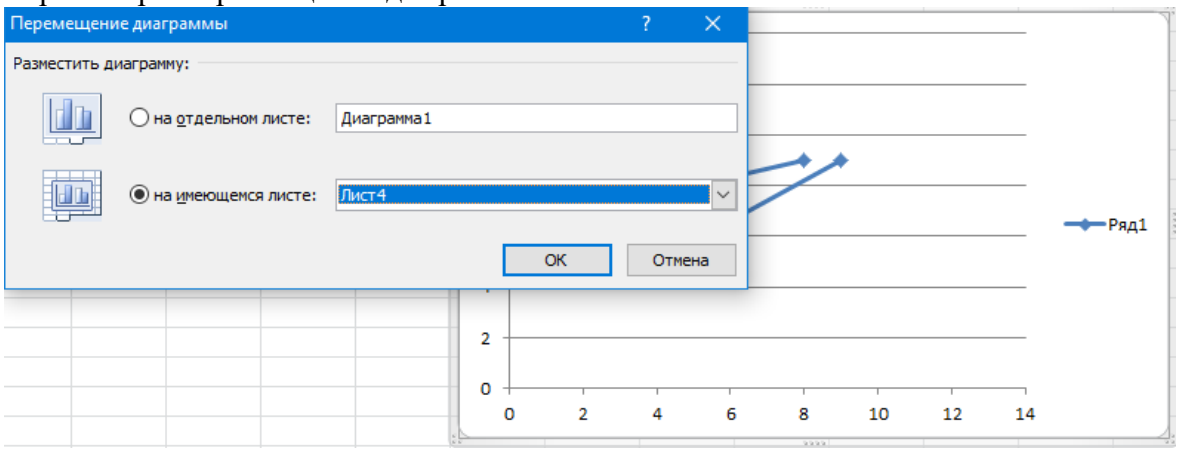
Якщо в діапазоні даних наявні пусті клітинки, їх за допомогою послуги *Приховані та пусті клітинки*, що знаходиться у зазначеному вікні, можна відображати як проміжок, або як нуль.



Послуговуючись групою піктограм, розміщених у вкладці *Макети діаграм*, користувач має змогу обрати необхідне оформлення діаграми: назву, підписи, легенду.

Для зміни візуально стилю діаграми доцільно скористатися послугою *Знаряддя для діаграм – Конструктор – Стилі діаграм.*

Цей крок здійснюються за допомогою вкладки *Конструктор – Розташування – Перемістити діаграму*. При цьому на екрані відкривається вікно *Переміщення діаграми,* в якому можна обрати варіант розміщення діаграми.



Скориставшись одним з двох перемикачів зазначеного вікна – окремому чи *наявному,* діаграму можна розмістити на окремому робочому аркуші книги, або, як вбудований графічний об'єкт, на поточному аркуші.

Щоб розмістити діаграму на окремому робочому аркуші, досить увімкнути відповідний перемикач, а потім (за бажанням) замінити системне ім'я *Діаграма 1* на ім'я, задане користувачем.

Для розміщення діаграми як вбудованого графічного об'єкта на одному з аркушів робочої книги потрібно ввімкнути перемикач наявному, а потім вибрати ім'я цього аркуша зі списку, що активується.

Оформлення таблиці. Фільтрація даних

Інколи таблиці містять велику кількість даних, поданих у вигляді списку. При їх опрацюванні зручно користуватися сортуванням та фільтруванням. Списки необхідно оформляти грамотно: дані розміщувати однотипні, не залишати порожніх рядків та стовпців, при наявності заголовків застосовувати до них інший формат. Сортування або впорядкування списків значно полегшує пошук даних. Після сортування записи відображаються в порядку, зазначеному користувачем (у алфавітному, зростання/спадання). Сортування здійснюється послугою Дані - Сортувати. При цьому у вікні Сортування користувач має змогу задати необхідні параметри сортування, порядок тощо.

При необхідності вибору даних, які відповідають певним умовам користуються фільтром. Основною відмінністю фільтрування від сортування є те, що під час фільтрування записи, які не відповідають заданим умовам відбору тимчасово не відображаються (але не видаляються). Фільтри розділяють на звичайний (авто фільтр) і розширений. Для застосування автофільтра користуються послугою *Дані – Фільтр.* У стовпцях списку при цьому відображаються кнопки зі стрілочками, обравши які користувач може налаштовувати параметри фільтра. Для використання розширеного фільтра застосовується послуга *Дані – Фільтр – Додатково.* За її допомогою відкривається вікно *Розширений фільтр* у якому користувач задає необхідні параметри фільтрування. Розширений фільтр зручно використовувати при необхідності розміщення результатів відбору окремо від основного списку.

ЛЕКЦІЯ 9. СУБД MICROSOFT ACCESS

9.1 Основні поняття бази даних

Сприйняття реального миру можна співвіднести з послідовністю різних, хоча іноді і взаємозв'язаних, явищ. З давніх часів люди намагалися описати ці явища (навіть тоді, коли не могли їх зрозуміти). Такий опис називають *даними*.

Активна діяльність по відшуканню прийнятних способів усуспільнення безперервно зростаючого об'єму інформації привела до створення на початку 60-х років спеціальних програмних комплексів, званих с*истеми управління базами даних (СУБД)*.

СУБД – це програмна система, що підтримує наповнення і маніпулювання даними, що представляють інтерес для користувачів при вирішенні прикладних завдань. Іншими словами, СУБД є інтерфейсом між базою даних і прикладними завданнями.

Основна особливість СУБД – це наявність процедур для введення і зберігання не тільки самих даних, але і описів їх структури. Файли, забезпечені описом що зберігаються в них даних і СУБД, що знаходяться під управлінням, почали називати банки даних, а потім бази даних (БД).

Існує велика кількість визначень поняття бази даних. Приведемо декілька визначень.

База даних – сукупність взаємозв'язаних даних, що зберігаються разом даних за наявності такої мінімальної надмірності, яка допускає їх використання оптимальним чином для одного або декількох застосувань.

База даних – це реалізована за допомогою комп'ютера інформаційна структура (модель), що відображає стан об'єктів і їх відношення.

База даних (БД) – це засіб накопичення і організації великих масивів інформації про об'єкти деякої предметної області (ПО). БД повинна відображати поточні дані про предметну область, накопичувати, зберігати інформацію і надавати різним категоріям користувачів швидкий доступ до даних. Для цього дані в базі мають бути структуровані відповідно до деякої моделі, що відображає основні об'єкти ПО, їх властивості і зв'язки між ними. БД є частиною складної системи, званої банком даних або системою баз даних (СБД).

Досвід використання баз даних дозволяє виділити загальний набір їх робочих характеристик:

– *повнота* – чим повніше база даних, тим ймовірніше, що вона містить потрібну інформацію (проте, не повинно бути надмірної інформації);

– *правильна організація* – чим краще структурована база даних, тим легко в ній знайти необхідні відомості;

– *актуальність* – будь-яка база даних може бути точною і повною, якщо вона постійно оновлюється, тобто необхідно, щоб база даних в кожен момент часу повністю відповідала стану об'єкту, що відображався нею;

– *зручність для використання* – база даних має бути проста і зручна у використанні і мати розвинені методи доступу до будь-якої частини інформації.

Нижче перераховані основні функції СУБД.

1. *Визначення даних* – визначити, яка саме інформація зберігатиметься в базі даних, задасть властивості даних, їх тип (наприклад, число цифр або символів), а також вказати, як ці дані зв'язані між собою. В деяких випадках є можливість задавати формати і критерії перевірки даних.

2. *Обробка даних* - дані можуть оброблятися самими різними способами. Можна вибирати будь-які поля, фільтрувати і сортувати дані. Можна об'єднувати дані з іншою, пов'язаною з ними, інформацією і обчислювати підсумкові значення.

3. *Управління даними* – можна вказати, кому дозволено знайомитися з даними, коректувати їх або додавати нову інформацію. Можна також визначати правила колективного доступу.

Вхідні до складу сучасних СУБД засоби спільно виконують наступні функції:

– опис даних, їх структури (звичайний опис даних і їх структури відбувається при ініціації нової бази даних або додаванні до існуючої бази нових розділів (стосунків); опис даних необхідний для контролю коректності використання даних, для підтримки цілісності бази даних);

– первинне введення, поповнення інформації в базі даних;

– видалення застарілої інформації з бази даних;
– коректування даних для підтримки їх актуальності;
– впорядкування (сортування) даних по деяких ознаках;
– пошук інформації по деяких ознаках (для опису запитів є спеціальна мова запитів, він забезпечує також інтерфейс між базою даних і прикладними програмами користувачів, дозволяє цим програмам використовувати бази даних);
– підготовку і генерацію звітів (засоби підготовки звітів дозволяють створювати і роздруковувати зведення по заданих формах на основі інформації бази даних);
– захист інформації і розмежування доступу користувачів до неї (деякі розділи бази даних можуть бути закриті для користувача зовсім, відкриті тільки для читання або відкриті для зміни; крім того, при многопользовательськом режимі роботи з базою даних необхідно, щоб зміни вносилися коректно; для збереження цілісності даних служить механізм трансакцій при маніпулюванні даними – виконання маніпуляцій невеликими пакетами, результати кожного з яких у разі виникнення некоректності операцій «відкатуються» і дані повертаються до початкового стану);
– резервне збереження і відновлення бази даних, яке дозволяє відновити втрачену при збоях і аваріях апаратури інформацію бази даних, а також накопичити статистику роботи користувачів з базою даних;
– підтримку інтерфейсу з користувачами, який забезпечується засобами ведення діалогу (у міру розвитку і вдосконалення СУБД цей інтерфейс стає все більш дружнім; дружність існуючих засобів інтерфейсу припускає наявність розвиненої системи допомоги (підказки), до якої у будь-який момент може звернутися користувач, не перериваючи сеансу роботи з комп'ютером і базою даних;
– захист від необдуманих дій, застережливий користувача і запобігаючу втрату інформації у разі поспішних або помилкових команд;
– наявність декількох варіантів виконання одних і тих же дій, з яких користувач може вибрати найбільш зручні для себе, відповідні його підготовці, кваліфікації, звичкам;
– ретельно продуману систему ведення людино-машинного діалогу, відображення інформації на дисплеї, використання клавіш клавіатури).

Розрізняють три типи СУБД:
– ієрархічна;
– мережева;
– реляційна.

Ієрархічна модель – це модель даних, у якій зв’язки між даними мають вигляд ієрархій.

В ієрархічній базі файли будуть пов‟язані між собою фізичними покажчиками або полями даних, доданих до окремих записів.

Мережна модель – це модель, коли кожний запис може бути підпорядкований записам більше, ніж з одного файлу.

Для зв’язування даних використовують фізичні покажчики.

Реляційна модель – це модель, в основі якої лежить математичне поняття відношення.

Відношення подається у вигляді двовимірних таблиць. Отже, в реляційній моделі дані організовані у формі двовимірної таблиці по колонках і рядках. Тут дані пов’язані відповідно до їхніх внутрішніх логічних взаємовідносин.

9.2 Створення зв’язку із зовнішніми даними

Таблицю можна створити за допомогою імпортування або зв’язування з даними, які зберігаються в іншому місці. Наприклад, можна імпортувати або створити зв’язок із даними в робочому аркуші *Excel*, списку *SharePoint*, файлі *XML*, іншій базі даних *Access*, папці *Microsoft Office Outlook* і деяких інших джерелах. Під час імпортування даних створюється їх копія в новій таблиці в поточній базі даних. На відміну від цього, якщо створюється зв’язок із даними, у поточній базі даних створюється зв’язана таблиця, яка представляє постійний зв’язок із наявними даними, які зберігаються в іншому місці. Таким чином, якщо змінюються дані в зв’язаній таблиці,

вони також змінюються у вихідному джерелі (за деякими винятками). Якщо змінюються дані вихідного джерела за допомогою іншої програми, ця зміна відображається в зв'язаній таблиці.

У деяких випадках не можна внести зміни до джерела даних через зв'язану таблицю, особливо, якщо джерело даних – це робочий аркуш *Excel*.

9.3 Типи даних у таблицях Access

У програмі Office Access автоматично визначається тип даних для поля, що створюється в поданні таблиці. Якщо потрібно безпосередньо визначити тип даних і формат поля, щоб перезаписати параметри, вибрані програмою Office Access, це можна зробити за допомогою команд у групі *Тип даних і форматування* на вкладці *Режим таблиці*.

У нижченаведеній таблиці описано доступні типи даних для полів в Office Access.

Тип даних	Зберігає	Розмір
Текст	Алфавітно-цифрові символи Використовується для тексту або для тексту й чисел, які не використовуються в обчисленнях (наприклад ідентифікатор товару).	Не більше 255 символів.
Примітка	Алфавітно-цифрові символи (довжиною більше 255 символів) або текст із форматуванням RTF. Використовується для текстових фрагментів, які містять більше 255 символів, або для форматованого тексту. Прикладами використання поля «Примітка» можуть бути примітки, об'ємні описи й абзаци, що містять таке форматування, як жирний шрифт або курсив.	Не більше 1 Гбайт символів або не більше 2 Гбайт дискового простору (по 2 байти на символ), з яких в елементі керування можна відобразити 65 535 символів.
Число	Числові значення (цілі або дробові). Використовується для збереження чисел, призначених для обчислень, за винятком грошових значень (для грошових значень використовується тип даних «Грошова одиниця»).	1, 2, 4 або 8 байтів, або 16 байтів у разі використання для ідентифікатора реплікації.
Дата й час	Значення дати й часу. Використовується для збереження значень дати й часу. Зверніть увагу на те, що кожне збережене значення включає компонент дати і компонент часу.	8 байтів.
Грошова одиниця	Грошові значення. Використовується для збереження грошових (валютних) значень.	8 байтів.
Автонумерація	Унікальне числове значення, яке у програмі Office Access 2007 автоматично вставляється в разі додавання запису. Використовується для створення унікальних значень, які можуть застосовуватись в якості первинного ключа. Зверніть увагу, що поля з типом даних «Автонумерація» можуть послідовно збільшуватися на задане значення або значення вибираються довільно.	4 байти або 16 байтів у разі використання для ідентифікатора реплікації.
Так/Ні	Логічні значення.	1 біт (8 бітів = 1 байт).

Тип даних	Зберігає	Розмір
	Використовуються для полів, які можуть містити одне з двох можливих значень, наприклад: «Так/Ні» або «True/False».	
Об'єкт OLE	Об’єкти OLE або інші двійкові дані. Використовується для збереження об’єктів OLE з інших застосунків Microsoft Windows.	Не більше 1 Гбайт.
Вкладення	Рисунки, зображення, двійкові файли, файли застосунків Office. Це основний тип даних для збереження цифрових зображень і будь-яких типів двійкових файлів.	Для стиснених вкладень 2Гб; для нестиснених вкладень приблизно 700 Кб, залежно від коефіцієнта можливого стиснення вкладення.
Гіперпосилання	Гіперпосилання. Використовується для збереження гіперпосилань, які надають безпосередній доступ до веб-сторінок за допомогою URL-адреси або до файлів із використанням імені у форматі UNC (універсальна угода про іменування). Також можна створювати зв’язки з об’єктами Access, які зберігаються в базі даних.	Не більше 1 Гбайт символів або не більше 2 Гбайт дискового простору (по 2 байти на символ), з яких в елементі керування можна відобразити 65 535 символів.
Майстер підстановок	Фактично не є типом даних, натомість викликає майстер підстановок. Використовується для запуску майстра підстановок із метою створення поля, значення якого видобувається за допомогою поля зі списком з іншої таблиці, запиту або списку значень.	На основі таблиці або запиту: розмір приєднаного стовпця. На основі значення: розмір текстового поля, яке використовується для зберігання значення.

9.4 Створення зв’язків. Первинний ключ

Первинний ключ таблиці складається з одного або кількох полів, які визначають унікальним чином кожен рядок, який зберігається в таблиці. Часто первинним ключем є унікальний ідентифікаційний номер, наприклад номер ідентифікатора, серійний номер або код. Наприклад, у таблиці «Клієнти» кожен клієнт може мати унікальний номер ідентифікатора. Поле ідентифікатора клієнта є первинним ключем цієї таблиці.

Поле, яке найкраще підходить для первинного ключа, має кілька характеристик. По-перше, значення поля унікальним чином ідентифікує кожний рядок. По-друге, воно ніколи не буває пустим або нульовим – завжди містить значення. По-третє, поле рідко (найкраще – ніколи) змінюється. У програмі Access поля первинних ключів використовуються для швидкого зведення даних із кількох таблиць.

Для таблиці завжди потрібно вказувати первинний ключ. У програмі Access автоматично створюється індекс первинного ключа, який допомагає прискорити виконання запитів та інших операцій. Також у програмі Access кожен запис має в полі первинного ключа значення, яке є унікальним.

Під час створення нової таблиці в поданні таблиці програмою Access автоматично створюється первинний ключ, якому призначається ім’я поля типу даних «Ідентифікатор» і «Автонумерація». За промовчанням у поданні таблиці поле приховано, але його можна побачити, якщо перейти до подання конструктора.

В Access можуть пов’язуватися між собою окремі таблиці БД.

Зв'язок – спосіб, за допомогою якого інформація з однієї таблиці пов'язується з інформацією іншої таблиці.

Як правило, зв'язують ключове поле однієї таблиці з відповідним йому полем іншої таблиці, яке називають полем зовнішнього ключа. Для встановлення зв'язку між таблицями вибирають команду *Сервис–Схема данных* або натиснути кнопку *Схема данных,* у якому можна встановити та переглянути зв'язки між таблицями

Для зв'язування таблиць потрібно мишкою перемістити поле первинного ключа головної таблиці до відповідного поля (зовнішнього ключа) підпорядкованої таблиці. На екрані з'явиться діалогове вікно *Изменение связей*.

Між таблицями можуть установитися такі типи відношень: «*один-до-одного*» або «*один-до-багатьох*» із забезпеченням цілісності даних. Тому схема даних базується відповідно до інформаційно-логічної моделі.

Параметр *Обеспечение целосности* означає виконання для взаємозв'язаних таблиць таких умов коригування даних:

1) у підпорядковану таблицю не можна додати запис з неіснуючими у головній таблиці значеннями ключа зв'язку;

2) у головній таблиці не можна вилучити запис, якщо не вилучені пов'язані з нею записи в підпорядкованій таблиці;

3) зміна значень ключа зв'язку в головній таблиці повинна призводити до зміни відповідних значень у записах підпорядкованої таблиці;

4) встановлювати зв'язки між таблицями типу 1:1 або 1:М і задавати для них параметри цілісності даних можна тільки за таких умов:

– зв'язані поля можуть мати різні імена, але тип даних і значення характеристик повинні бути однаковими;
– обидві таблиці повинні зберігатися в одній базі даних;
– головна таблиця зв'язується з підпорядкованою за первинним ключем.

9.5 Форми та звіти

Форма – це об'єкт БД, призначений для введення і відображення інформації. Форми дозволяють виконати перевірку коректності даних при введенні, проводити обчислення, забезпечують доступ до даним в зв'язаних таблицях за допомогою підлеглих форм.

Робота з формами може відбуватися в трьох режимах: у режимі Форми, в режимі Таблиці, в режимі Конструктора. Вибрати режим роботи можна за допомогою кнопки Вигляд панелі інструментів Конструктор форм або за допомогою команди меню Вигляд.

У режимах Форми і Таблиці можна здійснювати додавання, видалення і редагування записів в таблиці або в запиті, що є джерелом даних для форм.

У режимі Конструктора можна проводити зміну зовнішнього вигляду форми, додавання і видалення елементів управління, розробку.

У Access можна створити форми наступних видів:

– форма в стовпець або повноекранна форма;
– стрічкова форма;
– таблична форма;
– форма головна/підлегла;
– зведена таблиця;
– форма-діаграма.

Форма в стовпець є сукупністю певним чином розташованих полів введення з відповідними ним мітками і елементами управління. Найчастіше ця форма використовується для введення і редагування даних.

Стрічкова форма служить для відображення полий групи записів. Поля не обов'язково розташовуються у вигляді таблиці, проте для одного поля відводиться стовпець, а мітки поля розташовуються як заголовки стовпців.

Таблична форма відображає дані в режимі таблиці.

Форма головна/підлегла є сукупністю форми в стовпець і табличною. Її має сенс створювати при роботі із зв'язаними таблицями, в яких встановлений зв'язок типу один-до-багатьох.

Форма Зведена таблиця виконується майстром створення звідних таблиць Excel на основі таблиць і запитів Access (майстер звідних таблиць є об'єктом, упровадженим в Access, щоб використовувати його в Access, необхідно встановити Excel). Звідна таблиця є перехресною таблицею даних, в якій підсумкові дані розташовуються на перетині рядків і стовпців з поточними значеннями параметрів.

Форма з діаграмою. У Access у форму можна вставити діаграму, створену Microsoft Graph. Graph є упроваджуваним OLE-застосунком і може бути запущений з Access. З упровадженою діаграмою можна працювати так само, як і з будь-яким об'єктом OLE.

Будь-яка форма може включати наступні розділи: заголовок форми – визначає верхню частину форми і може містити текст, графіку і інші елементи управління;

– верхній колонтитул – розділ відображається тільки в режимі попереднього перегляду і зазвичай містить заголовки стовпців;

– область даних – визначає основну частину форми, що містить поля, отримані з джерела даних;

– нижній колонтитул – розділ відображається тільки в режимі попереднього перегляду в нижній частині екранної сторінки і зазвичай містить номер сторінки, дату і т. д.;

– примітка форми – відображається внизу останньої екранної сторінки форми.

Звіти – це форми «навпаки». З їхньою допомогою дані видають на принтер у зручному і наочному виді.

Звіти, які формуються в Access, мають ті самі області, що і форми: область заголовку і приміток, області нижнього і верхнього колонтитулів. Відмінність звітів і форм полягає в тому, що вони створюються лише для видачі на друк і не мають управляючих елементів для введення даних. Звіти можна побудувати на основі значень таблиць і запитів і розрахувати проміжні підсумки для груп значень.

Є такі засоби створення звітів:

1) конструктор (не для початківців);

2) майстер звітів;

3) автозвіти.

Складові частини звіту (не всі обов’язкові):

– Заголовок. Інформація на початку першої сторінки.

– Верхній колонтитул. Інформація на початку кожної сторінки(заголовки стовпців таблиць)

– Область даних. Відображення даних із таблиць або запитів

– Примітка групи. Інформація в кінці групи даних(підсумок за групою)

– Нижній колонтитул. (Інформація в кінці кожної сторінки(номер сторінки)

– Область приміток звіту. Інформація в кінці останньої сторінки звіту (підсумкові обчислення по всім записам звіту).

За допомогою елементів керування у звіт можна додавати рисунки, діаграми, інші об'єкти. Дані редагувати у звіті не можна.

Для створення звіту у вікні бази даних активізують вкладку *Отчеты*, вибирають *Создать\Мастер отчетов*. У наступному вікні *Таблицы и запросы* вибирають ім’я потрібної таблиці чи запита. Після цього з групи *Доступные поля* за допомогою кнопок вибирають потрібні поля, які переміщуються автоматично в групу *Выбранные*.

Структуру звіту можна змінити у режимі конструктора. Перемикаючи режими *Конструктор і Образец* можна оглянути результати кожного кроку при зміні структури звіту.

.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: СТВОРЕННЯ ЗАПИТІВ РІЗНИХ ТИПІВ У БД

Запити

Запити – це гнучкий і зручний засіб доступу до даних, важливою властивістю якого є те, що при створенні результуючої таблиці можна не тільки вибирати інформацію з бази, але й обробляти її.

При роботі запиту дані можуть упорядковуватися (сортуватися), фільтруватися (відсіюватися), об'єднуватися, розділятися, змінюватися, і при цьому ніяких змін у базових таблицях може не відбуватися. Наприклад: на великому підприємстві є база даних «Кадри», що містить докладні відомості про кожного співробітника.

Крім формальної інформації база може містити і конфіденційну, наприклад відомості про заробітну плату. Вся ця інформація зберігається в базових таблицях. Працювати з базою даних «Кадри» можуть різні підрозділи підприємства, і усім їм потрібні різні дані. Не все те, що дозволено знати службі безпеки підприємства, повинно бути доступно головному лікарю, і навпаки. Тому доступ користувачів до базових таблиць закривають. У такому випадку зручно використовувати запити. Для однієї і тієї ж таблиці можна створити багато різних запитів, кожний із який зможе добувати саме ту частину інформації, що у даний момент необхідна. Так, у бухгалтера має бути запит, що дозволить визначити скільки днів у році через хворобу був відсутнім той або інший працівник, але в нього не повинно бути запиту, що дозволяє дізнатись, чим він хворів і де лікувався, а в головного лікаря такий запит має бути.

Види запитів Access

1. Запити на вибірку. Метою запиту на вибірку є створення результуючої таблиці, у якій відображаються тільки потрібні за умовою запиту дані з базових таблиць.

2. Запити за зразком. Бланк запиту за зразком має дві панелі. На верхній панелі розташовані списки полів тих таблиць, на яких засновується запит. Рядки нижньої панелі визначають структуру запиту, тобто структуру результуючої таблиці, у якому будуть міститися дані, отримані за результатами запиту.

3. Запити з параметром. Використовується тоді, коли користувачу треба надати можливість вибору того, що він хоче знайти в таблицях бази даних.

4. Підсумкові запити. Дозволяють не тільки відбирати потрібну інформацію з таблиць і обробляти її шляхом створення нових полів, що обчислюються, але іробити так називані підсумкові обчислення.

5. Запити на зміну. Дозволяють автоматично створювати нові таблиці або змінювати вже наявні шляхом створення тимчасової результуючої таблиці.

Способи створення запитів Access

Запити на вибірку

Створення запиту до бази починається з відкриття вкладки «Запросы» діалогового вікна «База данных» і натиснення лівої клавіші миші на кнопці «Создать». У вікні «Новый запрос» задають режим створення запиту вибором пункту «Конструктор». Вибір таблиць виконують у діалоговому вікні «Добавление таблицы». Їх заносять у верхню половину бланка «запиту за зразком» натисненням лівої клавіші миші на кнопці «Добавить».

Запити за зразком.

Рядок «Поле» заповнюють перетягуванням назв полів із таблиць у верхній частині бланка. Кожному полю майбутньої результуючої таблиці відповідає один стовпець бланка запиту за зразком. Рядок «Ім'я» таблиці заповнюється автоматично при перетягуванні поля. Якщо натиснути на рядок «Сортировка», з'явиться кнопка списку, щорозкривається, який містить види сортування. Якщо призначити сортування по якомусь полю, дані в остаточній таблиці будуть відсортовані по цьому полю.

Запити з параметром.

Припустимо, що треба створити запит, за допомогою якого користувач може визначити, у якому році та або інша команда займала перше місце у чемпіонатах світу з футболу. Для цього служить спеціальна команда мови SQL, що виглядає так: LIKE [...]. У квадратних скобках можна

записати будь-який текст, звернений до користувача, наприклад: LIKE [Введіть назву країни]. Команду LIKE треба помістити в рядку «Условие отбора» і в те поле, по якому робиться вибір. У нашому випадку це стовпець збірних, що займали перші місця в чемпіонатах світу з футболу. Після запуску запиту відкривається діалогове вікно, у якому користувачу пропонується ввести параметр. Якщо в якості параметра ввести слово «Бразилія», те видається результуюча таблиця, що містить запису по тим чемпіонатам, коли збірна Бразилії ставала чемпіоном.

Підсумкові запити.

Їх створюють на основі бланка запиту за зразком, у якому з'являється додатковий рядок – «Групування». Для введення цього рядка в треба натиснути на кнопку «Групові операції» на панелі інструментів програми Access. У тих полях, по яких робиться групування, треба установити (або залишити) функцію «Групування» і вибрати одну з підсумкових функцій. Натиснення лівої клавіші миші на кнопці «Вид» запускає запит і видає результуючу таблицю з необхідними підсумковими даними.

Запити на зміну.

Натиснення лівої клавіші миші на кнопці «Вид» дозволяє переконатися, що запит працює як треба і створює результуючу таблицю, більш повну ніж базова. В меню «Запит», що доступно тільки в режимі «Конструктора» є команда для створення запитів на відновлення даних, на додавання записів і на вилучення записів. Всі вони відносяться до запитів на зміну і працюють аналогічно, змінюючи базові таблиці відповідно до даних результуючих таблиць.

Перехресний запит – підсумовує в електронній таблиці дані з однієї або декількох таблиць. Вони використовуються для аналізу даних, створення діаграм;

Запит SQL – заснований на інструкціях SQL (Structured Query Language – мова структурованих запитів). Мова SQL є стандартом для більшості СУБД. У форматі SQL у базі даних зберігаються всі запити.

ЛЕКЦІЯ 10. ПРОЦЕСИ ПРОЕКТУВАННЯ, КОНСТРУЮВАННЯ І ПІДГОТОВКИ ВИРОБНИЦТВА ТА ЇХ АВТОМАТИЗАЦІЯ. ЗАСОБИ РЕДАГУВАННЯ КРЕСЛЕНЬ

10.1 Проектування технічного об'єкту

Створення будь-яких виробів промисловості починається з розробки конструкторської документації. Рівень її виконання значною мірою впливає на скорочення строків створення та освоєння виробів, зниження трудомісткості їх виробництва, підвищення надійності та якості.

Розвиток обчислювальної техніки, розповсюдження персональних комп'ютерів і графічних дисплеїв як технічних засобів відображення графічної інформації привели до появи засобів генерації графічних зображень і автоматизованого виконання креслень – комп'ютерної графіки. Комп'ютерна графіка – сукупність методів і способів перетворення за допомогою комп'ютера даних у графічне зображення і графічного зображення у дані (ДСТУ 2939-94. «Система оброблення інформації. Комп'ютерна графіка. Терміни та визначення»).

Застосування комп'ютерної графіки та САПР дозволяє більшу частину рутинної роботи з проектування передати комп'ютеру і цим самим вивільнити час інженера-конструктора для творчої діяльності, суттєво підвищуючи при цьому якість результатів та скорочуючи строки проектування.

Проектування технічного об'єкту – це створення, перетворення і представлення в зрозумілій формі образу цього ще не існуючого об'єкту на основі виконання комплексу робіт дослідницького, розрахункового і конструкторського характеру.

Образ об'єкту або його складових частин може створюватися в уяві людини в результаті творчого процесу або генеруватися у відповідності з деякими алгоритмами в процесі взаємодії людини і комп'ютера. У будь-якому випадку інженерне проектування починається за наявності вираженої потреби суспільства в деяких технічних об'єктах, якими можуть бути об'єкти будівництва, промислові вироби або процеси.

Проектування включає розробку технічної пропозиції і технічного завдання (ТЗ), що відбивають ці потреби, і реалізацію ТЗ в виді проектної документації. Зазвичай ТЗ представляють у вигляді деяких документів, і воно являється початковим (первинним) описом об'єкту.

Результатом проектування, як правило, служить повний комплект документації, що містить достатні відомості для виготовлення об'єкту в заданих умовах. Ця документація і є проект, точніше, остаточний опис об'єкту.

Зміст технічних завдань на проектування включає в себе:

1. Призначення об'єкту.

2. Умови експлуатації. Разом з якісними характеристиками (представленими у вербальній формі) є числові параметри, для яких вказані області допустимих значень (температура довкілля, зовнішні сили, навантаження і т.п.).

3. Вимоги до вихідних параметрів, тобто до величин, що характеризують властивості об'єкту, які цікавлять споживача.

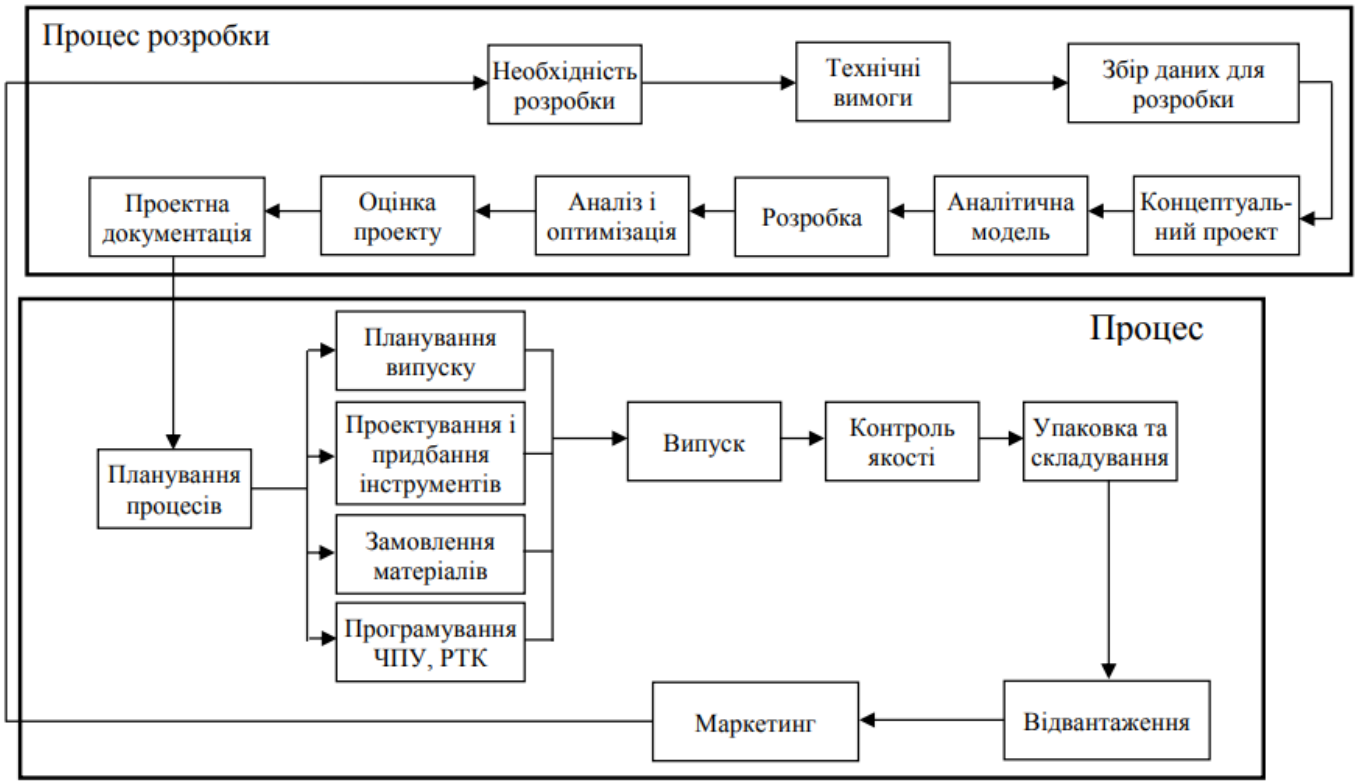
10.2 Автоматизація процесів проектування, конструювання та підготовки виробництва

Проектування, при якому усі проектні рішення або їх частину отримують шляхом взаємодії людини і комп'ютера, називають автоматизованим, на відміну від ручного (без використання комп'ютера) або автоматичного (без участі людини на проміжних етапах).

Система, що реалізовує автоматизоване проектування, є системою автоматизованого проектування (САПР). Автоматичне проектування можливе лише в окремих часткових випадках для порівняно нескладних об'єктів. На сьогодні переважаючим є автоматизоване проектування. Всі завдання, які доводиться вирішувати і виконувати в процесі розробки і виготовлення продукту, називаються життєвим циклом продукту. Приклад життєвого циклу продукту приведений на рисунку.

Системи автоматизованого проектування відносяться до числа найбільш складних сучасних штучних систем. Їх проектування і супровід неможливі без системного підходу. Основний загальний принцип системного підходу полягає в розгляді частин явища або складної системи з

урахуванням їх взаємодії. Системний підхід включає виявлення структури системи, типізацію зв'язків, визначення атрибутів, аналіз впливу зовнішнього середовища.



При структурному підході, як різновиді системного, проводиться синтез варіантів системи з стандартних компонентів (блоків) і оцінюються варіанти при їх частковому переборі з попереднім прогнозуванням характеристик компонентів.

Блоково-ієрархічний підхід до проектування використовує ідеї декомпозиції складних описів об'єктів на ієрархічні рівні і аспекти, вводить поняття стилю проектування (висхідне і низхідне), встановлює зв'язок між параметрами сусідніх ієрархічних рівнів.

В об'єктно-орієнтованому підході до проектування виражено ряд важливих структурних принципів, що використовуваних при розробці інформаційних систем і програмного забезпечення.

Для усіх підходів до проектування складних систем характерні також наступні особливості:

1. Структуризація процесу проектування, що виражається декомпозицією проектних завдань і документації, виділенням стадій, етапів, проектних процедур. Ця структуризація є суттю блоково-ієрархічного підходу до проектування.

2. Ітераційний характер проектування.

3. Типізація і уніфікація проектних рішень і засобів проектування.

У загальному випадку виділяють стадії науково-дослідних робіт (НДР), ескізного, технічного, робочого проектів, випробувань дослідних зразків.

Стадії (етапи) проектування підрозділяють на складові частини, що називаються проектними процедурами. Прикладами проектних процедур можуть служити підготовка креслень окремих деталей, аналіз кінематики, моделювання перехідного процесу тощо. У свою чергу, проектні процедури можна поділити на дрібніші компоненти, що називаються проектними операціями, наприклад побудова сітки скінченних елементів, вибір або розрахунок зовнішніх дій, представлення результатів моделювання. Таким чином проектування зводиться до виконання деяких послідовностей проектних процедур.

Як і будь-яка складна система, САПР складається з підсистем. Розрізняють підсистеми проектуючі і обслуговуючі.

Проектуючі підсистеми безпосередньо виконують проектні процедури. Прикладами проектуючих підсистем можуть служити підсистеми геометричного тривимірного моделювання об'єктів, виготовлення конструкторської документації, аналізу трасувань з'єднань в друкованих платах тощо.

Обслуговуючі підсистеми забезпечують функціонування проектуючих підсистем, їх сукупність часто називають системним середовищем (або оболонкою) САПР. Типовими обслуговуючими підсистемами є підсистеми управління проектними даними, підсистеми допомоги та навчання користувачів, вводу-виводу інформації.

Структуризацію САПР по різних аспектах обумовлює поява видів забезпечення САПР. Прийнято виділяти сім видів забезпечення САПР:

1) технічне – включає різні апаратні засоби (ЕОМ, периферійні пристрої, мережеве комутаційне устаткування, лінії зв'язку, вимірювальні засоби);

2) математичне – об'єднує математичні методи, моделі і алгоритми для виконання проектування;

3) програмне, що представляється комп'ютерними програмами САПР;

4) інформаційне, що складається з бази даних проектної інформації та СКБД;

5) лінгвістичне, вираженне мовою спілкування між проектувальником і комп'ютером, мовами програмування і мовами обміну даними між технічними засобами САПР;

6) методичне – включає різні методики проектування;

7) організаційне, представлене штатними розкладами, посадовими інструкціями і іншими документами, що регламентують роботу проектного підприємства.

10.3 Класифікація САПР

Класифікацію САПР здійснюють по ряду ознак, наприклад по застосуванню, цільовому призначенню, масштабах (комплексності вирішуваних завдань).

По застосуванню найбільш показними і широко використовуваними являються наступні групи САПР:

1. САПР для застосування в галузях загального машинобудування (AutoCAD, Inventor, Компас).

2. САПР для радіоелектроніки (P-CAD, OrCAD, Mentor).

3. САПР в області архітектури і будівництва (ArchiCAD, Allplan, Revit Structure).

Крім того, відоме велике число спеціалізованих САПР, що виділяються у вказаних групах або представляють самостійну гілку класифікації.

Прикладами таких систем є САПР великих інтегральных схем (ВІС); САПР літальних апаратів тощо.

За цільовим призначенням розрізняють САПР, що забезпечують проектування різних етапів життєвого циклу продукції:

1. Автоматизоване проектування (computer-aided design – CAD) – технологія, що полягає у використанні комп'ютерних систем для полегшення створення, зміни і аналізу проектів.

Найосновніша функція CAD – визначення геометрії конструкції (деталі механізму, архітектурних елементів, електронні схеми, плани будівель і т.п.), оскільки геометрія визначає усі наступні етапи життєвого циклу продукту. Для цієї мети зазвичай використовуються системи розробки робочих креслень і геометричного моделювання. (AutoCAD, ArchiCAD, Компас).

2. Автоматизоване виробництво (computer-aided manufacturing – CAM) – це технологія, що полягає у використанні комп'ютерних систем для планування, управління і контролю операцій виробництва через інтерфейс з виробничими ресурсами підприємства.

Одним з найбільш зрілих підходів до автоматизації виробництва є числове програмне управління, що полягає у використанні запрограмованих команд для управління верстатами.

Ще одна важлива функція систем автоматизованого виробництва – програмування роботів, які можуть працювати на гнучких автоматизованих дільницях чи конвеєрах. (CAM350, T-Flex).

3. Автоматизоване конструювання (computer-aided engineering – CAE) – це технологія, що полягає у використанні комп'ютерних систем для аналізу геометрії, моделювання і вивчення поведінки продукту для удосконалення і оптимізації його конструкції. Засоби CAE можуть здійснювати багато різних варіантів аналізу – кінематичний розрахунок, динамічний аналіз, аналіз логіки електронних ланцюгів, розрахунок напружено-деформованого стану конструкцій.

Таким чином, технології CAD/CAM/CAE полягають в автоматизації і підвищенні ефективності конкретних стадій життєвого циклу продукту. Розвиваючись незалежно, ці системи

ще не до кінця реалізували потенціал інтеграції проектування і виробництва. Перспективним є сценарій використання баз даних для інтеграції систем CAD, CAE і CAM.

Така інтеграція всіх процесів та інформаційних моделей в будівництві отримала назву BIM-технології (Building Information Model).

Інформаційне моделювання споруд (BIM) – процес колективного створення та використання інформації про споруду, що формує надійну основу для всіх рішень на протязі життєвого циклу об'єкту (від концепції до робочого проекту, будівництва, експлуатації та демонтажу).

Системи проектування умовно можна розділити на графічні системи, які працюють тільки в двомірній системі, в двох і трьох системах вимірювання, і лише в тривимірних системах. Останні дві схожі між собою. Просторовим моделям виробниками спеціальних програмних продуктів надано назву «твердотільні моделі».

Проектування відбувається на рівні твердотільних електронних (математичних) моделей із залученням могутніх конструкторсько-технологічних бібліотек, з використанням сучасного математичного апарату для проведення розрахунків. Крім того, ці системи дозволяють за допомогою засобів анімації імітувати переміщення в просторі робочих органів виробу (наприклад, маніпуляторів робота). Вони відстежують траєкторію руху інструменту при розробці і контролі технологічного процесу виготовлення спроектованого виробу. Все це робить просторове моделювання невід'ємною частиною спільної роботи САПР/АС ТПВ (Системи Автоматизованого Проектування / Автоматизовані Системи Технологічної Підготовки Виробництва).

Проект – результат розумової діяльності людини або колективу у сфері інформації.

Виріб – результат діяльності про сферу матеріальних об'єктів, тобто конкретна субстанція.

Процес виготовлення є процесом додання матеріальним об'єктам бажаних властивостей.

Особливість виробу як технічного засобу виявляється у взаємодії його з іншими об'єктами, тобто його відношенням до інших об'єктів. Так, особливості промислового робота, як технічного засобу з гнучкими технологічними властивостями виявляються у взаємодії його з об'єктами маніпулювання, технологічними машинами, технологічним оснащенням (накопичувачами, живильниками об'єктів маніпулювання, орієнтуючими пристроями і т.д.), а також з іншими промисловими роботами.

Елемент – частина цілого, яка в різних сукупностях може виступати як відносно ціле, причому властивості цієї щодо цілого залишаються постійними.

Конструкція – сукупність структур і стану виробу, яка визначає клас деякої безлічі виробів і тим самим встановлює можливість існування або існування виробів з однаковими властивостями

Конструювання – процес добору конструктивних характеристик, що визначають логічну основу конструкції.

Проектування – вибір деякого способу дії або складання опису, необхідного для створення об'єкту на основі первинного опису цього об'єкту і (або) алгоритму його функціонування, перетворення первинного опису (інколи багаторазове), автоматизація завдання характеристик об'єкту і алгоритму його функціонування, усунення некоректності первинного опису і його послідовності, описи на різних мовах.

Технічне проектування – проектування технічної системи як логічної основи дії технічного засобу або технічного комплексу, а також визначення характеристик конструктивного виду матеріальних комплексів.

Автоматизоване проектування – проектування, при якому окремі перетворення об'єкту і (або) алгоритму його функціонування, а також представлення опису на різних мовах здійснюються взаємодією людини і ЕОМ.

Системне проектування – проектування частини цілого з погляду цілого, з урахуванням інтересів цілого.

10.4 Засоби редагування креслень

AutoCAD – найвідоміший із продуктів компанії Autodesk, універсальна система автоматизованого проектування, що поєднує у собі функції двовимірного креслення й тривимірного моделювання.

AutoCAD прискорює щоденну роботу зі створення креслень і підвищує швидкість і точність їхнього виконання. Середовище концептуального проектування забезпечує легке й інтуїтивне створення і редагування твердих тіл і поверхонь. AutoCAD дозволяє легко й швидко створювати на основі моделі розрізи й проекції, ефективно формувати комплекти креслень і керувати ними: групувати їх по розділах проекту та інших логічних категорій, створювати переліки аркушів, керувати видами креслень, архівувати комплекти проектної документації та організовувати спільну роботу фахівців. Наявні в AutoCAD засоби візуалізації, такі як анімація й реалістичне тонування, допомагають виявити будь-які вади на ранніх етапах проектування.

Існують спеціалізовані галузеві різновиди AutoCAD для архітектури, дорожнього будівництва та землевпорядження, електротехніки, машинобудування тощо. Для фахівців, яким не потрібні функції роботи з 3D графікою, існує полегшена версія AutoCAD, призначена для створення двовимірних креслень – AutoCAD LT.

AutoCAD Mechanical – продукт на платформі AutoCAD для промислового виробництва. Він допомагає прискорити процес проектування, маючи у своєму складі бібліотеки ДСТУ, стандартних деталей і функції автоматизації типових завдань, забезпечує значний виграш у продуктивності при проектуванні.

AutoCAD Electrical – це AutoCAD для проектування електричних систем керування. Спеціалізовані функції й великі бібліотеки умовних позначень дозволяють підвищити продуктивність, усунути ризик виникнення помилок і забезпечити точність інформації, переданої у виробництво.

AutoCAD Inventor Suite – для проектування та конструювання в промисловому виробництві. Рішення сполучать у собі інтуїтивне середовище 3D моделювання деталей і виробів з інструментами. Ці інструменти містять у собі автоматичне створення інтелектуальних компонентів, таких як деталі із пластмаси, сталеві каркаси та обертові механізми.

CATIA – система автоматизованого проектування французької фірми Dassault Systems. Можливості – тривимірне моделювання і колективна робота в реальному часі. Відкриває можливість використовувати інтелектуальні результати онлайнового взаємозв’язку. Для зв'язку між людьми, що перебувають у різних точках світу, передбачені засоби простого підключення до Web.

Pro/Engineer – CAD система високого рівня. Містить у собі всі необхідні модулі для твердотілого моделювання деталей і створення креслярської документації. Має убудовані можливості для проектування зварених конструкцій.

SolidWorks – продукт компанії SolidWorks Corporation, система автоматизованого проектування у трьох вимірах, працює під керуванням Microsoft Windows. Розроблена як альтернатива для двомірних програм САПР. Придбала популярність завдяки простому інтерфейсу. Основний продукт SolidWorks включає інструменти для тривимірного моделювання, створення креслень, роботи з листовим металом, звареними конструкціям і поверхнями довільної форми.

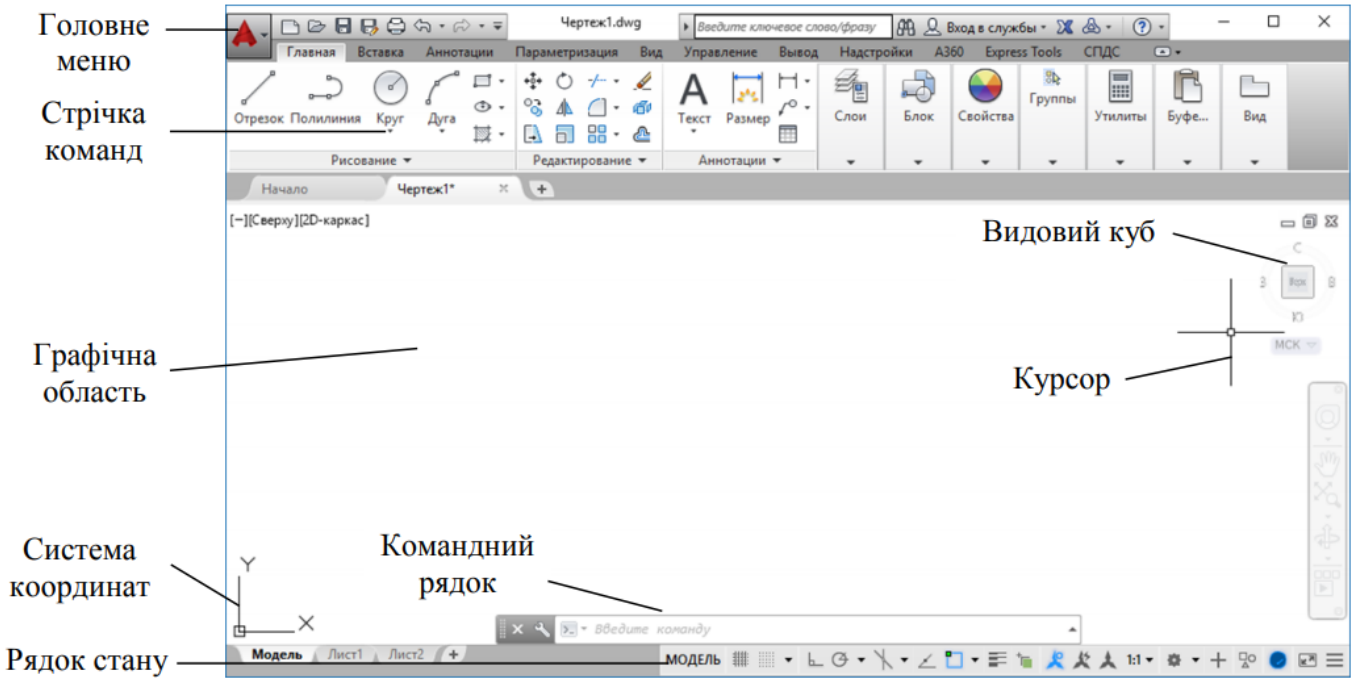
КОМПАС – система автоматизованого проектування, розроблена російською компанією «АСКОН» з можливостями оформлення проектної й конструкторської документації відповідно до стандартів серії ЕСКД і СПДБ (Система проектної документації для будівництва). Існує у двох версіях: Компас-Графік і КОМПАС-3D, відповідно призначених для плоского креслення і тривимірного проектування.

MechaniCS – додаток до AutoCAD або Autodesk Inventor, призначене для оформлення креслень відповідно до ЕСКД, проектування систем гідропневмоелементів, зубчастих зачеплень, валів, інженерного аналізу, розрахунку розмірних ланцюгів, створення користувальницьких бібліотек.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: НАЛАШТУВАННЯ ПАРАМЕТРІВ РОБОЧОГО СЕРЕДОВИЩА САПР AUTOCAD

Екранний інтерфейс

AutoCAD – універсальний графічний пакет, який працює не з зображенням, а з геометричним описом об'єктів, що обумовлено задачами систем автоматизованого проектування (САПР). В AutoCAD користувач створює креслення з використанням рядкового інтерфейсу.



Після запуску *Auto*CAD на екрані з'явиться діалогове вікно *Start Up*, в якому запропоновані наступні варіанти дій:

Open a drawing (відкриття малюнку) – дає змогу вибрати зі списку один з чотирьох раніше створених малюнків;

Start from scretch (без шаблону) – дає змогу створити нове креслення з параметрами, встановленими за замовченням;

Use a template (використовувати шаблон) – для створення креслення за заданим шаблоном;

Use a Wizard (виклик Майстра) – встановлення параметрів нового креслення.

За *Select Wizard* (вибір Майстра) AutoCAD пропонує два режими з автоматичними установками робочого середовища: *Advanced Setup* (детальна установка) і *Quick Setup* (швидка установка).

У діалоговому вікні *Advanced Setup* можна задати наступні параметри робочого середовища: лінійні та кутові одиниці, напрям нульового кута, орієнтації відліку кутів, межі креслення.

У діалоговому вікні *Quick Setup* можна вибрати одиниці вимірювання довжини і визначити межі креслення.

На початку програма пропонує створити нове креслення з параметрами, встановленими за замовченням. Для того, щоб його відкрити, необхідно натиснути на кнопку «Ок». Відкриється чистий файл з ім'ям Drawing1.dwg, яке пізніше можна буде змінити.

Вікно *Auto*CAD розподілено на наступні частини.

Графічна зона – центральна частина вікна, в якій буде розміщене створене нами креслення. Вона займає найбільшу частину екрану.

У верхній частині вікна знаходиться: рядок падаючого меню. Одразу під ним – панелі інструментів Standard (стандартна), Object Properties (властивості об'єктів) та Layers (шари).

За допомогою панелі Standard можна виконувати такі операції як відміна дій, зміна масштабу перегляду, друк, виклик центру управління *Auto*CAD, виклик вікна «Властивості об'єктів», виклик довідки *Auto*CAD і таке інше.

Панель *Object Properties* вміщує команди для присвоєння властивостей знову створеним об'єктам, а також для зміна поточних властивостей обраних об'єктів. Панель *Layers* вміщує елементи, що забезпечують роботу зі шарами *Auto*CAD.

Загалом панелей більше 20-ти, залежно від версії *Auto*CAD. Серед інших панелей – *Styles* (вміщує елементи для створення та вибору стилів, текстів, розмірів та таблиць), Workspaces (дозволяє переключатися між різними настройками робочого середовища *Auto*CAD) та інші.

Зліва та справа від графічної зони розташовуються панелі інструментів, що містять набори піктограм команд побудови графічних об'єктів та їх редагування. За замовченням відображені панелі *Draw* (креслення) та *Modify* (редагування). В панелі Draw зібрані кнопки з командами для створення основних об'єктів *Auto*CAD. Усі інші панелі можна відкрити у разі їх необхідності.

У нижній частині графічної зони знаходиться піктограма системи координат користувача. Напрямок її стрілок вказує на позитивний напрям осей X та Y. Нижче знаходяться вкладки стану компонувань аркуша. Вкладки Layout використовують, готуючи розроблені креслення до друку.

У нижній частині екрана розташоване окреме вікно командного рядка, у якому записуються найменування команд та опції для їх виконання. Для відображення введених команд відкривають текстове вікно на весь екран за допомогою натискання на клавіатурі функціональної клавіші «F2».

Найнижче знаходиться рядок стану: зліва відстежуються координати курсора, а по центру знаходяться кнопки управління режимами креслення.

Використовуючи вкладки *Model* і *Layout* можна швидко переключатися із простору моделі (*Model*) в простір аркуша (*Layout*). Звичайно проектування ведеться в просторі моделі, а в просторі аркуша створюються та розміщуються різноманітні види моделей, призначені для виводу на друк.

Встановлення режимів креслення

У нижній частині вікна AutoCAD розміщується рядок стану. У ньому відображаються поточні координати графічного курсору, а також знаходиться ряд кнопок, призначених для встановлення режимів креслення. Всього цих кнопок на панелі за замовчанням десять, але будь-яку з них можна або прибрати, або додати на панель. Вони працюють як перемикачі – зображення натиснутої кнопки відповідає увімкненому стану відповідного режиму, а зображення ненатиснутої кнопки – вимкненому стану.

Функціональне призначення кнопок у порядку їх розташування:

МОДЕЛЬ (MODEL/PAPER) – служить для перемикання між простором моделі і простором аркуша.

СЕТКА (GRID) – вмикає або вимикає відображення на екрані фонової допоміжної сітки з точок. Цій кнопці відповідає функціональна клавіша F7.

ШАГ (SNAP) – задає параметри крокової прив'язки, тобто управляє режимом прив'язки до точок сітки з певним кроком – F9.

ОРТО (ORTHO) – вмикає або вимикає режим ортогонального креслення, при якому система проводить лінії тільки паралельно координатним осям – F8.

ИЗООРТО (ISOORTHO) – вмикає або вимикає режим ізометричних координатних площин та забезпечує переключення між ними – F5.

ОТС-ПОЛЯР (POLAR) – вмикає або вимикає режим полярного трекінгу (відстеження), при якому система відображає на екрані тимчасові допоміжні нескінченні прямі (лінії вирівнювання), направлені під кутами, кратними кутові, вказаному користувачем – F10.

ОТС-ОБЪЕКТ (OTRACK) – вмикає або вимикає режим відслідковування об'єктної прив'язки, що дозволяє точно розміщувати нові об'єкти відносно проміжної точки, що вказується за допомогою об'єктної прив'язки – F11.

ПРИВЯЗКА (OSNAP) – вмикає або вимикає режим об'єктної прив'язки, який дозволяє користувачеві вибирати характерні точки в процесі редагування об'єктів – F3.

ВЕС (LWDISPLAY) – вмикає або вимикає відображення ваги (товщини) ліній на екрані.

Управління екранним відображенням

Часто в процесі роботи необхідно змінити масштаб перегляду креслення, переміститися до певного його місця і т. д. Для управління екранним відображенням в AutoCAD використовується декілька основних команд.

Панорамування – дає можливість переміщати видиму область креслення. Актуальне при роботі над проектами великого розміру, коли усе креслення не може уміщатися на екрані. Виконується натисканням середньої кнопки миші або за допомогою смуг прокрутки.

Масштабування – дає можливість збільшити певну ділянку креслення і розглянути її ближче або ж, навпаки, віддалити зображення, щоб усе креслення поміщалося на екрані. Виконується прокручуванням колеса миші.

Для збільшення зручності користування система AutoCAD дозволяє змінити велику кількість налаштувань робочого середовища, таких як колір фону та курсора, розміщення файлів на диску, гладкість відображення кривих на екрані і т. п.

За замовчанням файли креслень записуються у кореневий каталог системи AutoCAD.

Файли креслень слід зберігати в окремих папках, створених спеціально з цією метою. За умовчанням в AutoCAD значення інтервалу між автоматичними збереженнями становить 30 хвилин. Зменшити або збільшити цей інтервал можна відповідною зміною значення системної змінної SAVETIME. Зробити це можна як з клавіатури (набравши savetime, а далі у відповідь на запит системи: Новое значение SAVETIME <30>: ввести потрібне значення), так і через діалогове вікно Настройка (Options).

Файл, що створюється в результаті автоматичного збереження, має розширення .sv$. Щоб скористатися ним, його потрібно попередньо перейменувати. Він є тимчасовим і після закриття креслення видаляється. При збереженні креслення вручну AutoCAD за замовчанням автоматично зберігає стару версію під поточним ім’ям з розширенням .bak. У разі потреби повернутися до цієї копії її потрібно перейменувати, присвоївши розширення .dwg. Автоматичне створення резервної копії можна заборонити, встановивши значення змінної ISAVEBAK в 0 або за допомогою діалогового вікна Настройка (Options).

В AutoCAD відстань між точками вимірюється в умовних одиницях, які можуть відповідати будь-яким одиницям вимірювання довжини (футам, метрам, дюймам тощо). Завдяки цьому при кресленні можна оперувати реальними розмірами. Масштабування різних частин зображення відповідно до формату документа здійснюється при виведенні на друк шляхом задання співвідношення між умовними одиницями файлу креслення і міліметрами креслення на аркуші.

Положення будь-якого елемента креслення визначається за допомогою координат. За умовчанням AutoCAD використовує свою внутрішню тривимірну прямокутну декартову систему координат, що називається Світовою (МСК) – World Coordinate System (WCS). Напрямок осей X та Y відображає піктограма в лівому нижньому кутку графічного поля. Вісь Z згідно з правилом правої руки направлена на користувача. Признаком МСК на піктограмі осей є квадрат в точці початку координат. Користувач може створювати свої власні системи координат – ПСК (UCS). Системи координат користувача використовують для зручнішого задання геометрії моделі, вони можуть бути довільним чином орієнтовані відносно МСК. В одному кресленні можна створювати та зберігати необмежену кількість систем координат користувача.

Велика кількість команд AutoCAD потребує вибору об’єктів, про що повідомляє підказка Выберите объекты: командного рядка. Після появи запиту курсор миші набуває вигляду маленького квадратика. За допомогою цього квадратного маркера можна послідовно вибрати потрібну кількість об’єктів. Вибрані об’єкти відображаються пунктирною лінією (стають виділеними). Щоб закінчити процес вибору, необхідно натиснути Enter. Якщо якийсь з об’єктів вибрано помилково, його можна видалити з набору, помістивши на нього квадратний маркер і натиснувши ліву кнопку миші, утримуючи при цьому натиснутою клавішу Shift. Квадратний маркер є режимом (методом) вибору за умовчанням.

Вводити команду можна лише тоді, коли в командному рядку відображається запрошення «Команда:». Повторний виклик останньої команди можна здійснити, якщо у відповідь на запрошення «Команда:» натиснути клавішу Enter або Space, для послідовного перебору історії введених команд – клавіші управління курсором.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: КОМАНДИ НАНЕСЕННЯ ШТРИХУВАНЬ

Штрихування

Штрихуванням називають заповнення трафаретом замкнутих областей рисунка. Звичайно штрихування являє собою лінії, розташовані на певній відстані одна від одної і які мають обраний кут і тип нахилу. Але штрихування може бути представлене також у вигляді символів або суцільного заливання кольорами. При заповненні простору усередині об'єкта штрихування може змінюватися в процесі його редагування (наприклад, зміни розмірів). Таке штрихування прийнято називати асоціативним.

Мета штрихування – більш наочне представлення деяких областей рисунка, нанесення додаткової інформації про тип заштрихованої області (зрізи, перетини, різні типи поверхонь та ін.).

В AutoCAD існує досить багато зразків штрихування, кожний з яких відповідає певному матеріалу: сталі, цегельній кладці й т.д. Використовуючи ці зразки, можна без особливих труднощів заштриховувати виділені об'єкти, не вимальовуючи кожну лінію вручну. Крім заготовлених зразків штрихувань, у програмі передбачені можливості створення власних.

Штрихування виконується командою bhatch або у пункті «Штриховка» меню «Рисование», чи після натиснення курсором відповідної піктограми з панелі інструментів «Рисование».

Після цього з’являється діалогове вікно «Штрихова» (Boundary Hatch), у якому вказується зразок штрихування, границі області, параметри штрихування. Для штрихування можна використати такі типи зразків: стандартний (Predefined), користувача (User defined), замовлені (Custom). Стандартні зразки створені у відповідності до стандартів ANSI (American National Standard Institute), ISO (International Standard Organization), або деяких загальноприйнятих стандартів, що використовуються у світі.

Назва відповідних зразків починається з префіксів ANSI чи ISO. Зразки з префіксами ISO базуються на метричній системі одиничних вимірів. Для стандартних зразків можна змінювати параметри кута повороту ліній зразка (Angle) та масштабу чи відстані між лініями (Scale). Деякі зразки створені у реалістичному масштабі і при рисуванні у масштабі 1:1 показують абсолютні відстані між власними елементами.

Зразки, що визначаються користувачем, складаються з одного чи двох наборів паралельних ліній, інтервал між якими та кут їх нахилу можна змінювати. Другий набір паралельних ліній, що є перпендикулярним до першого набору, створюється за допомогою опції «Двойные».

Зразки, створені користувачем, не є дуже різноманітними, але швидко і зручно визначаються.

Замовлені зразки визначаються як окремі файли з розширенням pat (pattern – зразок), або можуть додаватись до існуючих файлів Acad.pat чи Acadiso.pat. Ці файли редагуються окремо, не під час роботи в AutoCad. Установки масштабу та кута нахилу функціонують аналогічно цим же установкам для стандартних зразків.

Зміна типу зразка або його імені здійснюється у вкладці «Быстрый» (Quick) натисненням екранного курсору на відповідних полях діалогового вікна. Після натиснення кнопки з трьома точками викликається діалогове вікно палітри штрихувань, де можна обрати один з усіх доступних зразків. У діалоговому вікні штрихування є кнопка «Наследуемые свойства» (Inherit Properties). Після натиснення на неї екранним курсором можна продублювати параметри вже нанесеного штрихування в інших об’єктах рисунка. Діалогове вікно тимчасово приховується і на екрані з’являється курсор з піктограмою. Цим курсором обирається штрихування, яке треба копіювати. Після цього обирається внутрішня точка області, до якої потрібно застосувати обраний зразок. Тоді натисненням «Ввод» здійснюється перехід до діалогового вікна штрихування, де після натиснення «ОК» операція завершується.

Існує два способи визначення контурів штрихування: «область», коли повністю заповнюється область, замкнена одним чи декількома граничними об’єктами (Select objects); «внутрішня точка» (Pick inherit point), коли вказується точка в області, обмеженій декількома різнорідними об’єктами, що перетинаються.

Командою bhatch (hatch) можна створювати асоціативні та неасоціативні штрихування. Асоціативні штрихування пов'язані з граничними об'єктами і автоматично змінюють свою форму при зміні форми граничного контуру. Звичайно використовується саме цей тип штрихування. Неасоціативні штрихування не пов'язані з об'єктом, який їх обмежує.

Острови. Способи їх розпізнавання

При деяких формах контуру, зокрема, якщо він не замкнений, асоціативне штрихування не може бути створено, тоді як неасоціативне – може заповнити певну внутрішню частину цього контуру. Іноді всередині області штрихування можуть знаходитись інші замкнені області (острови). Ці острови також можуть містити в собі інші, менші острови. Наприклад, текст чи мультитекст розглядаються як островки в області штрихування.

Якщо для визначення області штрихування вказується внутрішня точка, то AutoCad автоматично визначає острови. Якщо ж використовується метод вибору граничних об'єктів, треба явно вказати всі внутрішні об'єкти, які треба врахувати як острови, інакше вони також будуть заштриховані. Острови та стилі їх обробки визначаються у закладці «Дополнительно» діалогового вікна штрихування.

Існує три стилі розпізнавання островів (Island Detection Style):

– Обычный (Normal),
– Внешний (Outer),
– Пропуск (Ignore).

Згідно з першим із стилів штрихування виконується від зовнішнього контуру всередину області. Якщо зустрічається замкнений контур всі лінії зразка закінчуються на ньому. Якщо ж всередині цього острова є також острови, штрихування поновлюється.

Згідно зі стилем «Внешний» штрихування виконується від зовнішнього контуру і при наявності внутрішнього контуру припиняється. Згідно з останнім стилем всі внутрішні контури при штрихуванні ігноруються. Стиль «Внешний» доцільно використовувати при заповненні островів різними зразками штрихування.

В закладці «Дополнительно» вказується також тип об'єкта, який обмежує штрихову, а також чи потрібно зберігати границі (Retain Boundaries). У множині границь (Boundary Set) вказані всі можливі набори контурів. Одночасно може існувати два набори: останній, визначений користувачем та завжди доступний, який складається з усіх наявних об'єктів рисунку. При створенні нового набору він заміщує старий. За наявністю великого числа об'єктів і контурів для штрихування можна явно обрати ті об'єкти і контури, які визначаються як границі штрихування. Створені контури звичайно зберігаються у поточному шарі, але можуть зберігатись окремо. Для визначення островів використовуються такі способи: «Поток» (Flood), «Трассировка лучей» (Ray Casting).

Згідно з першим способом островки включаються до контуру як об'єкти, згідно з другим способом – від вказаної точки до найближчого об'єкта проводиться лінія, після чого визначається контур у напрямку стрілки годинника з виключенням островів з числа потенціальних граничних об'єктів. Для керування штриховою використовуються кнопки: «Удалить острова» (Remove Islands) – вилучення окремих островів з набору контурів із в казанням точки всередині острова. «Показати контури» (View selection) – відображення визначених у даний момент контурів з виділенням граничних об'єктів. «Перегляд» (Preview) – відобразити визначені у даний момент контури з поточними установками штрихування. Редагування зразків штрихувань виконується командою hatchedit, або у пункті «Штриховка» меню «Изменить». Крім того, всі властивості створених штрихувань можуть бути змінені за натисненням правої кнопки миші. Після виклику команди hatchedit з'являється діалогове вікно, подібне до вікна штрихування, у якому заблоковано декілька параметрів. Доступними, зокрема, є параметри типу зразка та стилю розпізнавання островів. При зміні форми граничних контурів, у тому числі островів, асоціативна штрихова змінює форму. При вилученні одного з граничних об'єктів штрихова втрачає асоціативність і більше не узгоджується зі своїм контуром. Після втрати властивість асоціативності неможливо поновити – це можна зробити негайною командою undo.

Керування виглядом штрихувань

Для керування виглядом штрихувань використовуються такі можливості:

1) вирівнювання відповідних елементів зразків у сусідніх областях з однаковим масштабом та кутом нахилу. Точка прив'язки для всіх штрихувань звичайно (0, 0, 0). Змінити цю точку можна, задавши нові значення системній змінній SnapBase;

2) розбиття штрихування на окремі лінійні об'єкти командою explode, після чого властивість асоціативності втрачається, а окремі об'єкти можна редагувати;

3) відключення видимості штрихувань або системною змінною FillMode (якщо має нульове значення, всі штрихування невидимі) або у вкладці «Экран» вікна «Опції», меню «Инструменты». При цьому стають невидимими і внутрішні розширені частини поліліній та мультиліній.

При копіюванні, переміщенні, дзеркальному відображенні можна обирати асоціативне штрихування та контур як разом, так і окремо. Якщо системна змінна PickStile дорівнює одиниці, штрихова і об'єкти вибираються окремо, якщо нулю – то разом.

В AutoCad 2000 збережена команда Hatch з попередніх версій, яка створює лише неасоціативні штрихування. У цій команді є опція Direct Hatch, де можна вказати область штрихування, не визначивши граничні контури. Для визначення областей і островів може бути використана команда Boundary. Після цього з'являється діалогове вікно, аналогічне до вікна штрихування, але тільки з закладкою «Дополнительно» (Advanced). Після цієї команди створюються і зберігаються лише контури штрихування.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: РЕЖИМИ РОБОТИ В САПР AUTOCAD

AutoCAD надає допоміжні засоби, що забезпечують додаткові зручності в процесі креслення, особливо при використанні інтерактивного методу задання координат. До них належать режими: Сетка (Grid), Привязка к сетке (Grid Snap), Полярная привязка (Polar Snap), Полярное отслеживание (Polar Tracking), Ортогональный (Ortho).

Відображення піктограмами:

Відображення текстом:

Режим сітки (Grid) призначений для візуалізації на екрані дисплея вузлів фонової допоміжної координатної сітки, крок якої встановлюється користувачем. Режим вмикається/вимикається кнопкою СЕТКА (GRID), розміщеною в статусному рядку, або клавішею F7. Змінити крок фонової допоміжної сітки можна через діалогове вікно Режимы рисования (Drafting Settings). У діалоговому вікні встановлюється значення інтервалу між вузлами сітки вздовж осі X та Y. Фонова допоміжна сітка не є об’єктом креслення і на друк не виводиться.

При роботі з кресленням можна ввімкнути або вимкнути режими, що спрощують побудову. Одним з таких режимів є режим об'єктної прив'язки, який дозволяє пов'язувати точки створюваного об'єкта з точками раніше побудованого. Точками прив'язки можуть бути кінцеві або центральні точки об'єктів, точки явного або передбаченого перетину і т.д.

Визначення необхідних точок відбувається без визначення їх координат. Після увімкнення режиму об'єктної прив'язки необхідно вибрати спосіб прив'язки і помістити курсор поблизу об'єкта. Координати необхідної точки будуть визначені автоматично.

Об'єктна прив'язка використовується при виконанні операцій побудови і редагування у відповідь на запит програми вказати наступну точку.

Вибір режиму прив'язки здійснюється таким способом:

1. Вибрати режим, натиснувши на панелі Object Snap відповідну кнопку. Для виклику панелі використовують контекстне меню будь-якої панелі інструментів розташованих на екрані.

2. Ввести в командному рядку перші три букви назви режиму у відповідь на запит системи вказати точку.

3. Вибрати режим прив'язки на вкладці Object Snap діалогового вікна Drafting Settings. Для виклику вікна натиснути ПКМ, утримуючи при цьому затиснутою клавішу Shift.

Режим прив’язки до сітки (Grid Snap) має два режими – прив’язка до сітки (Grid Snap) та полярна прив’язка (Polar Snap). У певний момент часу може бути активізований тільки один з них. Вибір режиму здійснюється встановленням відповідного перемикача на вкладці Шаг и сетка (Snap and Grid) діалогового вікна Режимы рисования (Drafting Settings). Вмикання/вимикання обох режимів здійснюється кнопкою ШАГ (SNAP) або клавішею F9. Перемикатися з одного режиму на інший можна також вибором відповідного пункту в контекстному меню, що викликається на кнопці ШАГ (SNAP). При увімкненому режимі прив’язки до сітки графічний курсор переміщується по вузлах уявної сітки із заданим кроком (за умовчанням він дорівнює 10 одиницям). Це дозволяє точно задавати довжину відрізків на кресленні. Зміна кроку сітки та її орієнтації здійснюється на вкладці Шаг и сетка (Snap and Grid) діалогового вікна Режимы рисования (Drafting Settings).

Режим Полярная прив’язка (Polar Snap) призначений для використання разом з режимом Полярное отслеживание (Polar Tracking). Він забезпечує переміщення курсору з фіксованим кроком уздовж напрямів, установлених для режиму полярного відстежування. За замовчанням значення кроку переміщення дорівнює 10 одиницям. Змінити його можна шляхом введення нового значення на вкладці Шаг и сетка (Snap and Grid) діалогового вікна Режимы рисования (Drafting Settings).

Режим полярного відстежування (Polar Tracking) спрощує побудову відрізків, а також

виконання деяких інших команд у разі, коли подібні операції виконуються із заданим приростом кута. Режим вмикається/вимикається клавішею F10 або кнопкою ОТС-ПОЛЯР (POLAR), розміщеною в статусному рядку.

При увімкненому режимі система відображає на екрані тимчасові допоміжні нескінченні прямі (лінії вирівнювання), направлені під кутами, кратними кутові, вказаному користувачем. За замовчанням крок кута полярного відстеження (трекінгу) дорівнює 90°. Змінити значення кроку кута можна на вкладці Полярное отслеживание (Polar Tracking) діалогового вікна Режимы рисования (Drafting Settings).

Пропонується наступний стандартний набір значень кроку кута полярного відстеження: 90, 45, 30, 22.5, 18, 15, 10.5, можна також задати власні значення кроку кута полярного відстеження.

Ортогональний режим (Ortho) призначений для виконання ортогональних побудов. Вмикання/вимикання режиму здійснюється клавішею F7 або кнопкою ОРТО (ORTHO) в статусному рядку При увімкненому режимі ОРТО відрізки автоматично будуть розміщуватися по горизонталі або вертикалі (за умови, що не змінено встановлену за замовчуванням орієнтацію невидимої сітки).

Режим Object Snap Tracking використовується спільно з режимом об'єктної прив'язки. Після увімкнення режиму об'єктного відстеження – точному позиціонуванню черговий точки допомагають тонкі пунктирні лінії, які перетинають об'єкт в точках прив'язки – лінії трасування. Цей режим розширює і доповнює можливості об'єктної прив'язки, дозволяє задати точне положення об'єктів відносно один одного. Забезпечується дотриманням точних геометричних побудов без попереднього побудови допоміжних ліній. Режим генерує будь-яку кількість ліній трасування на підставі будь-якої кількості точок і параметрів об'єктної прив'язки.

Об’єктне відстежування є механізмом, що полегшує вибір точок, що лежать на лініях відстежування і об’єктів, що проходять через точки, що вказуються за допомогою об’єктної прив’язки. Об’єктне відстежування розширює і доповнює можливості об’єктної прив’язки. Цей режим включається кнопкою ОТС-ОБЪЕКТ (OTRACK) в рядку стану або функціональною клавішею F11. На перший погляд може здатися, що полярне і об’єктне відстежування абсолютно схожі між собою, тому як в обох режимах відображуються лінії відстежування. Насправді схожість дійсно є, проте при використанні об’єктного відстежування лінії генеруються не відносно поточного, а відносно інших побудованих об’єктів. Для роботи об’єктного відстежування необхідно активізувати режим об’єктної прив’язки, за допомогою якого відбувається захоплення вже побудованих точок малюнка. Щоб почати відстежування відносно деякої точки об’єкту, необхідно підвести покажчик миші до точки і затримати його на деякий час, поки у вказаній точці не з’явиться маленький плюс – він сигналізує, що AutoCAD захопив вказану точку і готовий до відстежування. Одночасно програмі можна вказати до семи таких точок. Після захоплення точки AutoCAD будуватиме вертикальні, горизонтальні і полярні лінії відстежування, які проходитимуть через цю точку.

РОЗДІЛ ІІ. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ АВТОМАТИКИ НА МОВІ С

ЛЕКЦІЯ 11. АЛГОРИТМИ ТА ЇХ ВЛАСТИВОСТІ. СХЕМИ АЛГОРИТМІВ. ФОРМИ ПОДАННЯ

11.1 Алгоритм та їх властивості

Використовуючи мікрокалькулятор для обчислень по формулі, інколи корисно наперед скласти і записати на папері програму, тобто перелік дій, які повинні бути зроблені користувачем у процесі розрахунку. Таку програму особливо корисно мати тоді, коли потрібно провести декілька обчислень по одній і тій же формулі. В цьому випадку спочатку складається і перевіряється програма обчислень, а потім користувач може використовувати її механічно, вже не вникаючі в послідовність виконуваних дій, що полегшує і прискорює процес обчислень.

Більш сучасні обчислювальні машини можуть запам´ятовувати програми обчислень. І тоді користувачу залишається тільки складати програму і помістити її в пам´ять машини, а все інше – розв´язок задачі – машина зробить автоматично, тобто без втручання людини. Такі обчислювальні машини називаються програмно-керуючими.

Основою програми є алгоритм розв´язку даної задачі, тобто точні вказівки про послідовність дій, які повинні бути зроблені для отримання результата. Алгоритм є більш узагальненим розумінням, ніж програма.

Одним із базових понять інформатики і ОТ є поняття алгоритму як деякого правила перетворення інформації. Він указує, які операції оброблення даних і в якій послідовності необхідно виконати, щоб одержати розв’язок задачі.

Алгоритм – точне розпорядження, що визначає обчислювальний процес, який веде від початкових даних, що змінюються, до шуканого результату.

Алгоритм має такі властивості:

1. Зрозумілість. Щоб виконавець міг досягти поставленої перед ним мети, використовуючи даний алгоритм, він повинен уміти виконувати кожну його вказівку, тобто розуміти кожну з команд, що входять до алгоритму.
2. Визначеність (однозначність). Зрозумілий алгоритм все ж таки не повинен містити вказівки, зміст яких може сприйматися неоднозначно. Крім того, в алгоритмах неприпустимі такі ситуації, коли після виконання чергового розпорядження алгоритму виконавцю не зрозуміло, що потрібно робити на наступному кроці.

Отож, точність – це властивість алгоритму, що полягає в тім, що алгоритм повинен бути однозначно витлумачений і на кожному кроці виконавець повинен знати, що йому робити далі.

3. Дискретність. Як було згадано вище, алгоритм задає повну послідовність дій, які необхідно виконувати для розв'язання задачі. При цьому, для виконання цих дій їх розбивають у визначеній послідовності на прості кроки. Виконати дії наступного розпорядження можна лише виконавши дії попереднього. Ця розбивка алгоритму на окремі елементарні дії (команди), що легко виконуються даним виконавцем, і називається дискретністю.
4. Масовість. Дуже важливо, щоб складений алгоритм забезпечував розв'язання не однієї окремої задачі, а міг виконувати розв'язання широкого класу задач даного типу. Наприклад, алгоритм покупки якого-небудь товару буде завжди однаковий, незалежно від товару, що купується. Або алгоритм прання не залежить від білизни, що *переться*, і таке інше. Отож, під масовістю алгоритму мається на увазі можливість його застосування для вирішення великої кількості однотипних завдань.
5. Результативність. Взагалі кажучи, очевидно, що виконання будь-якого алгоритму повинне завершуватися одержанням кінцевих результатів. Тобто ситуації, що в деяких випадках можуть призвести до так званого «зациклення», повинні бути виключені при написанні алгоритму.

Наприклад, розглянемо таку ситуацію: роботу дано завдання залишити кімнату (замкнутий простір), не виконуючи руйнівних дій. У цьому випадку, якщо роботу не дати вказівки відкрити двері (що, можливо, закриті), то спроби залишити приміщення можуть бути безуспішними.

6. Скінченність.

11.2 Форми подання та схеми алгоритмів

Для задання алгоритмів використовують такі способи, як:

– словесний опис послідовності обчислень;
– аналітичний (у вигляді формул);
– графічний (у вигляді схем і діаграм);
– псевдокод;
– запис алгоритмічною мовою.

Запис алгоритму алгоритмічною мовою потребує точного дотримання правил цієї мови, оскільки він має бути зрозумілим не тільки людині, а й комп´ютеру.

Псевдокод займає проміжне місце між словесним описом алгоритму і його записом алгоритмічною мовою. У цьому способі вживаються конструкції, близькі до алгоритмічної мови, але не вимагається повного дотримання всіх її правил, оскільки він призначений для сприйняття людиною.

Велике поширення дістав графічний спосіб задання алгоритму у вигляді схем.

Схема алгоритму – це графічне зображення його структури, в якому кожний етап процесу перероблення даних подається у вигляді різних геометричних фігур (символів).

Ці фігури з’єднуються між собою лініями потоку, які для кожного етапу вказують можливих наступників. Усередині фігури дається опис відповідного етапу, якщо він не є занадто громіздким. У противному разі такий опис наводиться у додатку до схеми, а замість нього у відповідній фігурі записується номер або яке-небудь позначення цього етапу. Біля фігури можуть бути деякі зауваження, наприклад такі, що показують, у якому випадку вибір наступника буде робитися відповідно до лінії потоку.

Символам присвоюють порядкові номери, які проставляються в розриві лінії контуру в лівій частині верхнього боку зображення символу. Лінії потоку проводять паралельно лініям зовнішньої рамки схеми. Напрямок лінії потоку зверху вниз і зліва направо прийнято за основний і, якщо вони не мають зламів, стрілки їх можна не позначати. В інших випадках їхній напрямок обов´язково позначають стрілкою. Лінію потоку, як правило, підводять до середини символу.

Відстань між паралельними лініями потоку має бути не меншою від 3 мм, між іншими символами – не менше від 5 мм. Лінію потоку можна обривати, використовуючи на місці обриву з´єднувачі, якщо схему виконано на двох аркушах, або якщо символи, які з´єднуються, розташовано на значній відстані один від одного.

Запис у середині символу або поруч із ним потрібно виконувати машинописом з одним інтервалом або креслярським шрифтом.

Перевагою схем є те, що з їх допомогою можна наочно зобразити структуру алгоритму в цілому, відобразивши його логічну суть (показати розгалуження шляхів розв´язання задачі залежно від виконання деякої умови, відобразити багаторазове повторення окремих етапів обчислювального процесу). Особливо це важливо для задач економічного характеру і задач управління. Вони містять велику кількість операцій порівняння, логічних, арифметичних й інших операцій, і тому відразу важно встановити їх послідовність у процесі розв´язування задачі.

Графічне зображення алгоритму у вигляді схем полегшує складання програми для розв´язання задачі на комп’ютері.

Опис алгоритмів за допомогою блок-схем

Термінатор – відображає вхід із зовнішнього середовища або вихід у нього (початок і кінець програми), всередині фігури записують відповідну дію: початок або кінець – овал.

Процес (обчислення) – виконання однієї або кількох операцій, опрацювання даних, всередині фігури записують безпосередньо самі операції — прямокутник з горизонтальними та вертикальними сторонами.

Дані – перетворення даних у форму, придатну для опрацювання (введення) або відображення (виведення). Цей символ не визначає носія даних. Для вказання типу носія даних використовують спеціальні позначення, які тут не розглянуто – паралелограм, у якому відстань між горизонтальними сторонами удвічі більша за проекцію похилої сторони на горизонталь.

Рішення – показує рішення: функцію перемикання з одним входом і двома або більше виходами, з яких лише один буде обрано після обчислення умов, записаних всередині елемента. Вхід в елемент позначають лінією, що входить зазвичай у верхню вершину. Якщо виходів два чи три, то зазвичай кожен вихід позначають лінією, що виходить з решти вершин (бічних і нижній). Якщо виходів більше трьох, то їх показують однією лінією, що виходить з вершини (частіше нижньої) і потім розгалужується. Відповідні результати обчислень записують поруч з лініями, що відображають ці шляхи – ромб з однаково нахиленими (до горизонталі) сторонами.

Зумовлений процес – виконання процесу, що складається з однієї або кількох операцій, визначених в іншому місці програми (у підпрограмі, модулі). Всередині символу записують назву процесу й передані в нього дані – прямокутник з горизонтальними та вертикальними сторонами, у якому проведено 2 вертикальні відрізки без порушення симетрії.

Межа циклу – складається з двох частин – відповідно, початок і кінець циклу. Операції, що виконують усередині циклу, записують між ними. Умови циклу і збільшення записують усередині символу початку або кінця циклу залежно від типу організації циклу. Часто для зображення на блок-схемі циклу замість цього символу використовують символ рішення, вказуючи в ньому умову, а одну з ліній виходу замикають вище в блок-схемі (перед операціями циклу) – прямокутник з горизонтальними та вертикальними сторонами, у якого згори відрізано рівнобедрені трикутники, і результат симетрії такого 6-кутника відносно горизонталі.

З'єднувач – символ відображає вихід з частини схеми і вхід у іншу частину цієї схеми. Використовують для обриву лінії та продовження її в іншому місці для поділу блок-схеми, що не можливо розташувати на одній сторінці. Відповідні сполучні символи повинні мати одне (при тому унікальне) позначення – круг.

Коментар – використовують для подання інформації про кроки, процесу або групи процесів. Опис розташовують з боку квадратної дужки, яка охоплює його по всій висоті. Пунктирна лінія йде до описуваного елементу, або групи елементів, яку виділяють замкнутою пунктирною лінією, – ліва квадратна дужка, з середини якої виходить пунктирна лінія ліворуч.

Стрілками блок-схемах вказують переходи між кроками виконання.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: ГРАФІЧНЕ ЗОБРАЖЕННЯ РІЗНИХ ВИДІВ ОБЧИСЛЮВАЛЬНИХ ПРОЦЕСІВ

Графічне зображення лінійних обчислювальних процесів

Обчислювальні процеси, що виконуються, за даним алгоритмом, поділяються на три основні види:

– лінійні;
– розгалужені;
– циклічні.

У лінійному обчислювальному процесі всі операції виконуються послідовно у порядку їх запису.

Типовим прикладом такого процесу є стандартна обчислювальна схема, що складається з трьох етапів:

– введення початкових даних;
– обчислення за формулами;
– виведення результату.

Графічне зображення розгалужених обчислювальних процесів

Обчислювальний процес називається розгалуженим, якщо для здобуття кінцевого результату передбачається вибір одного з кількох можливих напрямів обчислень (гілок) залежно від результату перевірки деякої умови.

Розгалужений обчислювальний процес, що складається з двох гілок, називається простим, а з більшої кількості гілок – складним. Напрям обчислень вибирається перевіркою, внаслідок якої можливі два виходи:

– «Так» – умову виконано;
– «Ні» – умову не виконано.

Умова вказується всередині символу «Розв'язування».

Графічне зображення циклічних обчислювальних процесів

В алгоритмах розв'язування багатьох задач потрібно виконати одну або кілька команд більше ніж один раз. Для цього такі алгоритми мають містити команди, що визначатимуть, які команди повинні виконатися неодноразово і скільки саме разів.

Фрагмент алгоритму, у якому одна або кілька команд можуть виконуватися більше ніж один раз, називають циклом. Алгоритм, який містить цикл, називають алгоритмом із циклом, або алгоритмом з повторенням.

Цикл «Повторити N разів» – це фрагмент алгоритму, який містить послідовність команд (тіло циклу), яка виконуватиметься вказану кількість разів, після чого знову виконується команда перевірки умови; якщо результат виконання команди перевірки умови хиба, виконання циклу з передумовою припиняється, після чого виконується перша команда наступного фрагмента алгоритму.

Цикл «Повторити N разів»

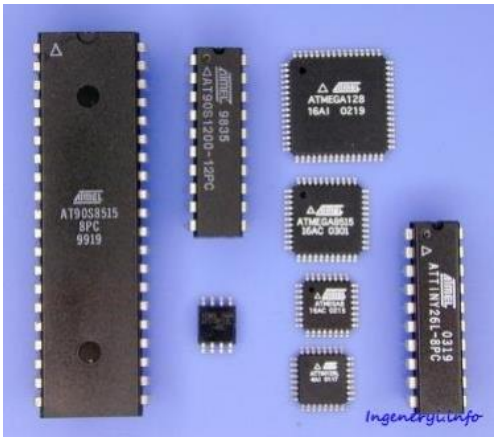
Команда циклу з передумовою виконується так: відбувається перевірка умови; якщо результат виконання цієї команди істина, то виконавець виконує команди тіла циклу, після чого знову виконує команду перевірки умови; якщо ж результат виконання команди перевірки умови хиба, то виконання команд тіла циклу не відбувається і виконавець переходить до виконання першої команди наступного фрагмента алгоритму.

Цикл з передумовою (стандартний)

ЛЕКЦІЯ 12. АРХІТЕКТУРА МІКРОКОНТРОЛЕРІВ AVR

12.1 Мікроконтролери AVR

AVR – це назва популярного сімейства мікроконтролерів, яке випускає компанія Atmel. Зовнішній вигляд:



AVR являють собою потужний інструмент для створення сучасних високопродуктивних і економічних багатоцільових контролерів. На даний момент співвідношення «ціна-продуктивність-енергоспоживання» для AVR є одним з кращих на світовому ринку 8-розрядних мікроконтролерів. Можна вважати, що AVR стає ще одним індустріальним стандартом серед 8-розрядних мікроконтролерів загального призначення.

Основою будь-якого МК є обчислювальне ядро, яке також виконує і керування інтегрованою периферією. У всіх моделях AVR воно однакове, тому легко забезпечується переносимість коду в межах цілої лінійки. Ядро AVR побудоване за Гарвардською архітектурою, згідно якої розділені не тільки види пам’яті (адресний простір пам’яті програм та пам’яті даних), але і шини доступу до цих видів пам’яті.

Ядро AVR використовує вдосконалену (enhanced) RISC систему команд. Більшість арифметичних та логічних команд виконуються за 1 такт, але деякі команди (наприклад, команди виклику переривань та підпрограм, а також повернення з них) виконуються за 4-5 тактів. Розмір команди фіксований: 1 комірка пам’яті програм (16 біт) для переважної більшості команд та 2 комірки для команд, у яких один з операндів є 16-ти розрядною адресою.

Сама ідея створення нового RISC-ядра народилася в 1994 році в Норвегії. У 1995 році два його винахідника Альф Боген (Alf-Egil Bogen) і Вегард Воллей (Vegard Wollen) запропонували корпорації Atmel випускати новий 8-розрядний RISC-мікроконтролер як стандартний виріб і забезпечити його Flash-пам'яттю програм на кристалі. Керівництво Atmel Corp. прийняло рішення інвестувати даний проект. У 1996 році був заснований дослідний центр в місті Тронхейм (Норвегія). Обидва винахідника стали директорами нового центру, а мікроконтроллерне ядро було запатентовано і отримало назву AVR (**A**lf-Egil Bogen + **V**egard Wollen + **R**ISC). Перший дослідний кристал 90S1200 був випущений на стику 1996-1997 років, а з 3 кварталу 1997 року корпорація Atmel приступила до серійного виробництва нового сімейства мікроконтролерів і до їх рекламної та технічної підтримки.

AVR функціонують в широкому діапазоні живлячої напруги від 1,8 до 6,0 вольт. Енергоспоживання в активному режимі залежить від величини напруги живлення, від частоти, на якій працює AVR і від конкретного типу мікроконтролера. Температурні діапазони роботи мікроконтролерів AVR – комерційний (0C ... 70C) і індустріальний (-40C ... + 85C).

Існує три види мікроконтролерів AVR:

1. AVR 8-bit.
2. AVR 32-bit.
3. AVR xMega

Найпопулярнішим вже більше десятка років є саме 8-бітове сімейство МК. Вивід або ніжка мікроконтролера називається «піном». Звідси слово «розпіновка» – тобто інформація про призначення кожної з ніжок.

Мікроконтролери AVR 8-bit в свою чергу діляться на два популярних сімейства:

– Attiny – з назви видно, що молодші (tiny – юний, молодий, молодший), в основному мають від 8 пінів і більше. Обсяг їх пам'яті і функціонал зазвичай скромніше, ніж в наступних.

– Atmega – більш просунуті МК, мають більшу кількість пам'яті, виводів і різних функціональних вузлів.

– Найпотужнішою підродиною мікроконтролерів є xMega – ці МК випускаються в корпусах з величезною кількістю пінів, від 44 до 100.

12.2 Загальна будова мікроконтролерів

Основу мікроконтролера становить процесорне ядро, яке складається з трьох основних пристроїв:

1. АЛУ – арифметико-логічний пристрій (мікропроцесор), який виконує всі обчислення.
2. Пам'ять – призначена для зберігання програм та даних.
3. Порти вводу–виводу, за допомогою яких мікроконтролер спілкується з «зовнішнім світом».

Залежно від моделі мікроконтролера в ньому можуть бути присутніми додаткові (периферійні) пристрої.

1. Аналоговий компаратор – пристрій порівняння. Присутній у всіх моделях мікроконтролерів. Основне завдання компаратора – це порівняння двох напруг: одна з них зразкова (з чим порівнюємо), а друга – вимірювана (порівнювана). Якщо порівнювана напруга більше зразкової – компаратор виробляє сигнал логічної одиниці. Якщо порівнювана напруга менше зразкової – компаратор формує на своєму виході логічний нуль.

2. АЦП – перетворювач аналогового напруги в цифрову форму. Мають не всі МК.

Аналогова напруга – це напруга, яка непрерервно змінюється в часі. Наприклад, синусоїдальний сигнал з виходу генератора частоти, напруга в побутової розетки, звуковий сигнал на колонках.

АЦП постійно аналізує на своєму вході величину напруги і видає на своєму виході цифровий код, відповідний вхідній напрузі.

3. Таймер/лічильник присутній у всіх моделях мікроконтролерів, але в різних кількостях – від 1 до 4, і з різними можливостями.

Таймер – пристрій, який дозволяє формувати часові інтервали. Він є цифровим лічильником, який рахує імпульси або від внутрішнього генератора частоти, або від зовнішнього джерела сигналу.

За допомогою таймера/лічильника можна:

– відраховувати і вимірювати часові інтервали;
– підраховувати кількість зовнішніх імпульсів;
– формувати ШІМ-сигнали

ШІМ – широтно-імпульсний модулятор, призначений для управління середнім значенням напруги на навантаженні.

ШІМ – один з варіантів роботи таймера/лічильника, що дозволяє генерувати на виході мікроконтролера прямокутну імпульсну напругу з регульованою тривалістю між імпульсами (скважністю), який застосовується в різних пристроях:

– регулювання частоти обертання електродвигуна;
– освітлювальні прилади;
– нагрівальні елементи

4. Сторожовий таймер має тільки одне завдання – проводити скидання (перезапускати програму) МК через певний проміжок часу. Є у всіх моделях МК. Може бути включений або вимкнений.

При нормальній роботі мікроконтролера і включеному сторожовому таймері, програма повинна періодично проводити скидання сторожового таймера (періодичний скид треба передбачити в програмі) ще до того, як він повинен спрацювати і перезапустити МК. Якщо

програма «зависла», то скидання сторожового таймера не буде, і через певний проміжок часу він перезапустить МК.

5. Модуль переривань.

Переривання – сигнал, що повідомляє процесору про настання якої-небудь події. При цьому виконання поточної програми призупиняється і управління передається обробнику переривання, який реагує на подію і обслуговує його (виконується програма, яку повинен виконати МК при настанні відповідної події – переривання), після чого повертається в перервану програму.

Переривання бувають внутрішні і зовнішні.

Внутрішні переривання можуть виникати при роботі периферійних пристроїв МК (АЦП, компаратор, таймер і т.д.)

Зовнішнє переривання – подія, яка виникає при наявності сигналу на одному зі спеціальних входів МК (таких спеціальних входів для зовнішніх переривань у МК може бути кілька).

12.3 Архітектура мікроконтролерів AVR

Мікроконтролер AVR містить: швидкий RISK-процесор, два типи енергонезалежної пам'яті (Flash-пам'ять програм і пам'ять даних EEPROM), оперативну пам'ять RAM, порти вводу/виводу та різні периферійні інтерфейсні схеми.

1. Мікропроцесор

Серцем мікроконтролерів AVR є 8-бітове мікропроцесорне ядро або центральний процесорний пристрій, побудоване на принципах RISK-архітектури. Основою цього блоку служить арифметико-логічний пристрій. При системному тактовому сигналі з пам'яті програм відповідно до вмісту лічильника команд вибирається чергова команда і виконується АЛП. Під час вибору команди з пам'яті програми відбувається виконання попередньої обраної команди, що і дозволяє досягти швидкодії 1 MIPS на 1 МГц.

АЛП підключено до регістрів загального призначення, яких всього 32. Вони мають байтовий формат, тобто кожен з них складається з восьми біт. РЗП знаходяться на початку адресного простору оперативної пам'яті, але фізично не є її частиною. Тому до них можна звертатися двома способами (як до регістрів і як до пам'яті). Таке рішення є особливістю AVR і підвищує ефективність роботи і продуктивність мікроконтролера.

Відмінність між регістрами і оперативною пам'яттю полягає в тому, що з регістрами можна робити будь-які операції (арифметичні, логічні, бітові), а в оперативну пам'ять можна лише записувати дані з регістрів.

2. Пам'ять

У мікроконтролерах AVR реалізована Гарвардська архітектура, відповідно до якої розділені не тільки адресні простори пам'яті програм і пам'яті даних, а й шини доступу до них. Кожна з областей пам'яті даних (оперативна пам'ять і EEPROM) також розташована в своєму адресному просторі.

Вся програмна пам'ять AVR-микроконтроллерів виконана по технології FLASH і розміщена на кристалі. Вона являє собою послідовність 16-розрядних комірок і має ємність від 512 слів до 256K слів залежно від типу кристала.

Поділ шин доступу до FLASH пам'яті й SRAM пам'яті дає можливість мати шини даних для пам'яті даних і пам'яті програм різної розрядності, а також використати технологію конвейеризації. Конвеєризація полягає в тім, що під час виконання поточної команди програмний код наступної вже вибирається з пам'яті й дешифрирується.

У випадку використання конвеєризації тривалість машинного циклу можна скоротити. Тривалість машинного циклу AVR становить один період кварцового резонатора, тому AVR-мікроконтролери здатні забезпечити задану продуктивність при більш низькій тактовій частоті. Саме ця особливість архітектури й дозволяє AVR-мікроконтролерам мати найкраще співвідношення енергоспоживання/продуктивність, тому що споживання мікросхем визначається їхньою робочою частотою.

3. Периферія

Периферія мікроконтролерів AVR включає: порти (від 3 до 48 ліній введення і виведення), підтримку зовнішніх переривань, таймери-лічильники, сторожовий таймер, аналогові

компаратори, 10-розрядний 8-канальний АЦП, інтерфейси UART, JTAG і SPI, пристрій скидання по зниженню живлення, широтно-імпульсні модулятори.

Система переривань – одна з найважливіших частин мікроконтролера. Всі мікроконтролери AVR мають багаторівневу систему переривань. Переривання припиняє нормальний хід програми для виконання пріоритетного завдання, визначається внутрішнім або зовнішнім подією.

Таймери/лічильники можна використовувати для точного формування часових інтервалів, підрахунку імпульсів на висновках мікроконтролера, формування послідовності імпульсів, тактирования прийомопередавача послідовного каналу зв'язку.

Сторожовий таймер (WatchDog Timer) призначений для запобігання катастрофічних наслідків від випадкових збоїв програми.

Аналоговий компаратор (Analog Comparator) порівнює напруги на двох виводах (пінах) мікроконтролера.

Універсальний асинхронний або універсальний синхронно/асинхронний приймач (Universal Synchronous / Asynchronous Receiver and Transmitter – UART або USART) – зручний і простий послідовний інтерфейс для організації інформаційного каналу обміну мікроконтролера із зовнішнім світом. Здатний працювати в дуплексному режимі (одночасна передача і прийом даних). Він підтримує протокол стандарту RS-232, що забезпечує можливість організації зв'язку з персональним комп'ютером.

Послідовний периферійний трьохпровідний інтерфейс SPI (Serial Peripheral Interface) призначений для організації обміну даними між двома пристроями. З його допомогою може здійснюватися обмін даними між мікроконтролером і різними пристроями, такими, як цифрові потенціометри, ЦАП / АЦП, FLASH-ПЗУ та ін За допомогою цього інтерфейсу зручно проводити обмін даними між декількома мікроконтролера AVR.

Крім того, через інтерфейс SPI може здійснюватися програмування мікроконтролера.

Двохпровідний послідовний інтерфейс TWI (Two-wire Serial Interface) дозволяє об'єднати разом до 128 різних пристроїв з допомогою двобічної шини, що складається з лінії тактового сигналу (SCL) і лінії даних (SDA).

Інтерфейс JTAG використовується для тестування друкованих плат, внутрішньосхемного налагодження, програмування мікроконтролерів.

4. Живлення

AVR функціонують при напругах живлення від 1,8 до 6,0 Вольт. Струм живлення в активному режимі залежить від величини напруги живлення і частоти, на якій працює мікроконтролер, і становить менше 1 мА для 500 кГц, 5 ... 6 мА для 5 МГц і 8 ... 9 мА для частоти 12 МГц.

AVR можуть бути переведені програмним шляхом в один з трьох режимів зниженого енергоспоживання.

– Режим холостого ходу (IDLE). Припиняє роботу тільки процесор і фіксується вміст пам'яті даних, а внутрішній генератор синхросигналів, таймери, система переривань і сторожовий таймер продовжують функціонувати.

– Стоповий режим (POWER DOWN). Зберігається вміст реєстрового файлу, але зупиняється внутрішній генератор синхросигналов, і, отже, зупиняються всі функції, поки не надійде сигнал зовнішнього переривання або апаратного скидання.

– Економічний режим (POWER SAVE). Продовжує працювати тільки генератор таймера, що забезпечує збереження часової бази. Всі інші функції відключені.

Скидання при зниженні напруги живлення (BOD)

Схема BOD (Brown-Out Detection) відстежує напругу джерела живлення. Якщо схема включена, то при зниженні живлення нижче деякого значення, вона переводить мікроконтроллер в стан скидання. Коли напруга живлення знов збільшиться до порогового значення, запускається таймер затримки скидання. Після формування затримки внутрішній сигнал скидання знімається і відбувається запуск мікроконтролера.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: ОГЛЯД, ХАРАКТЕРИСТИКИ, ПРИЗНАЧЕННЯ ВИВОДІВ МІКРОКОНТРОЛЕРІВ

Огляд і характеристики виводів мікроконтролерів

Порти мікроконтролера – це пристрої вводу–виводу мікроконтролера, що дозволяють йому передавати або приймати дані.

Стандартний порт мікроконтролера AVR має вісім розрядів даних, які можуть передаватися або прийматися паралельно. Кожному розряду (або біту) відповідає вивід (ніжка) мікроконтролера. Ніжки мікроконтролера також називають пінами. Для позначення портів використовуються латинські літери A, B, C і т.д. Кількість портів вводу/виводу варіюється в залежності від моделі мікроконтролера.

При передачі інформації мікроконтролер виставляє на своїх виводах відповідні логічні рівні (0 або 1). При прийомі інформації мікроконтролер зчитує з цих виводів логічні рівні, які виставлені зовнішнім пристроєм.

Порти вводу/виводу AVR мають число незалежних ліній «вхід/вихід» від 3 до 53. Кожна лінія порту може бути запрограмована на вхід або на вихід. Потужні вихідні драйвери забезпечують струмову навантажувальну здатність 20 мА на лінію порту при максимальному значенні 40 мА, що дозволяє, наприклад, безпосередньо підключати до мікроконтролера світлодіоди і біполярні транзистори. Загальне струмове навантаження на всі лінії одного порту не повинне перевищувати 80 мА (всі значення наведено для напруги живлення 5 В).

Архітектурна особливість побудови портів вводу/виводу в AVR полягає в тому, що для кожного фізичного виводу (піна) існує 3 біта контролю/управління, а не 2, як у поширених 8-розрядних мікроконтролерів (Intel, Microchip, Motorola і т.д.). Це дозволяє уникнути необхідності мати копію вмісту порту в пам'яті для безпеки і підвищує швидкість роботи мікроконтролера при роботі із зовнішніми пристроями, особливо в умовах зовнішніх електричних перешкод.

Мікроконтролери AVR мають в своєму складі від одного до семи паралельних портів вводу–виводу, які призначені для обміну даними із зовнішніми пристроями. Кожен розряд такого порту приєднаний до одного з виводів (контактів) МК. Порти можуть бути повні і неповні. Повний порт містить 8 розрядів. У неповних портах можуть бути задіяні сім, шість або навіть три розряди. Але для центрального процесора порти залишаються 8-розрядними, бо він завжди записує в такі порти і читає з них повноцінний байт даних. Розряди, які не використовуються, при записуванні просто втрачаються, а при читанні їх значення дорівнює нулю. Кожен порт має своє ім'я, яким є латинська буква від A до G.

Управління виводами мікроконтролерів

Для керування кожним портом вводу–виводу використовується три спеціальних регістра вводу–виводу. Це регістри PORTx, DDRx і PINx. Під «x» тут мається на увазі конкретна буква – ім'я порту. Наприклад, для порту A імена регістрів управління будуть такими: PORTA, DDRA і PINA.

Розглянемо призначення кожного з цих регістрів:

– PORTx – регістр даних (використовується для виведення інформації);
– DDRx – регістр напряму передачі інформації;
– PINx – регістр введення інформації.

Окремі розряди цих регістрів також мають свої імена. Розряди регістра PORTx зазвичай іменуються як Pxn, де «n» – це номер розряду. Наприклад, розряди регістра PORTA іменуватимуться таким чином: PA0, PA1, PA2–PA7. Розряди регістру DDRx іменуються як DDxn, а розряди регістру PINx іменуються як PINxn.

Будь-який порт вводу–виводу МК AVR влаштований таким чином, що кожен його розряд може працювати як на введення, так і на виведення. Для перемикання напряму передачі служить регістр DDRx. Кожен розряд регістра DDRx керує своїм розрядом порту. Якщо в якому-небудь розряді регістра DDRx записаний нуль, то відповідний розряд порту працює як вхід. Якщо ж в цьому розряді одиниця, то розряд порту працює як вихід.

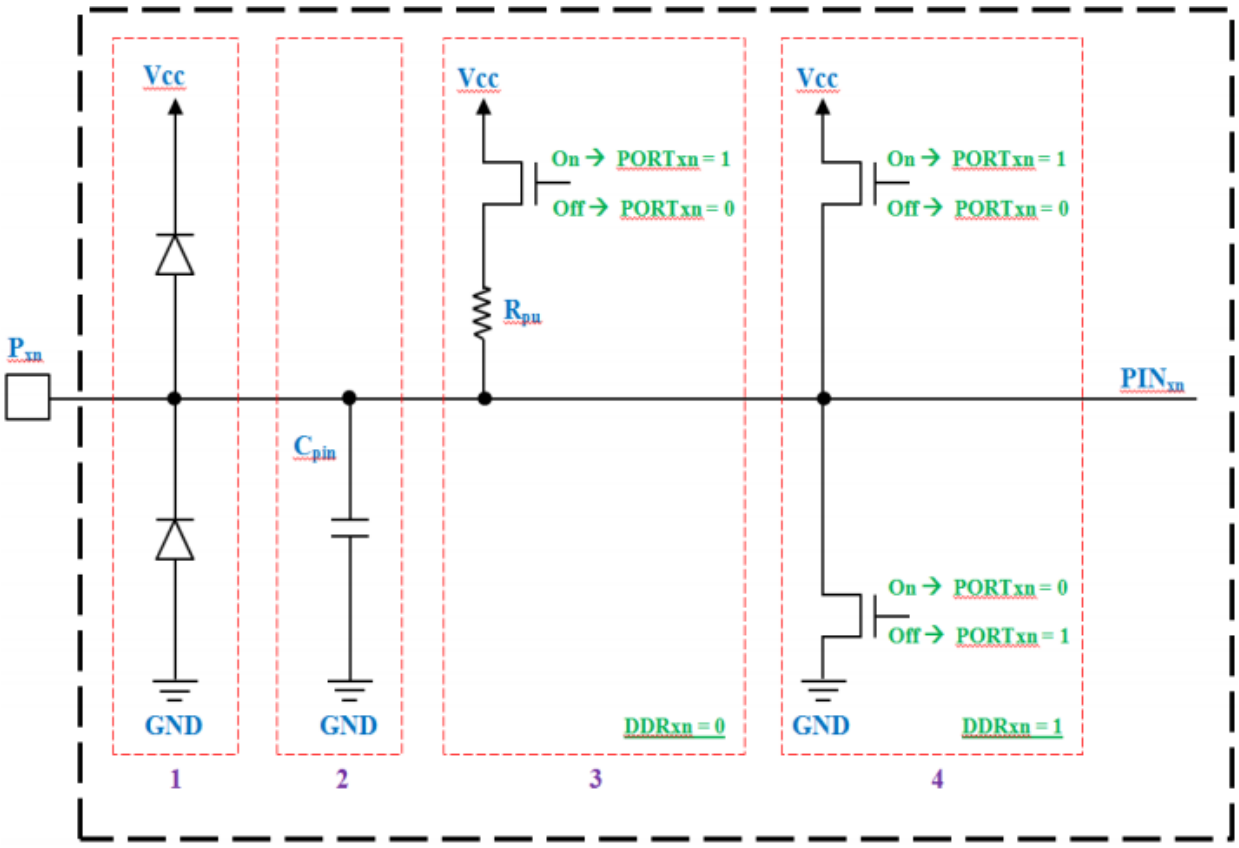
Для того, щоб видати інформацію на зовнішній контакт МК, потрібно у відповідний розряд DDRx записати логічну одиницю, а потім записати байт даних в регістр PORTx.

Для того, щоб прочитати інформацію із зовнішнього контакту МК, потрібно спочатку перевести потрібний розряд порту в режим введення, тобто записати у відповідний розряд регістра DDRx нуль. Тільки після цього на даний контакт МК можна подавати цифровий сигнал від зовнішнього пристрою. Далі МК читає байт з регістра PINx.

Порти вводу–виводу МК AVR мають ще одну корисну функцію. У режимі введення інформації вони можуть при необхідності підключати до кожного виводу порту внутрішній резистор навантаження. Внутрішній резистор дозволяє значно розширити можливості порту. Такий резистор створює для зовнішніх пристроїв струм, що витікає. Завдяки цьому резистору спрощується підключення зовнішніх контактів і кнопок. Включенням і відключенням внутрішніх резисторів керує регістр PORTx, якщо порт знаходиться в режимі введення. Це видно з таблиці, в якій показані всі режими роботи порту.

Конфігурація порту введення-виведення				
DDxn	Pxn	Режим	Резистор	Примітка
0	0	Вхід	Відключений	Вивід відключено від схеми
0	1	Вхід	Підключений	Вивід є джерелом струму
1	0	Вихід	Відключений	На виході «0»
1	1	Вихід	Відключений	На виході «1»

Спрощена схема порту вводу–виводу подана на рисунку.



Pxn – ім'я ніжки порту мікроконтролера, де x буква порту (A, B, C або D), n номер розряду порту (7 ... 0).

Cpin – паразитна ємність порту.

VCC – напруга живлення.

Rpu – відключається навантажувальний верхній резистор (pull-up).
PORTxn – біт n регістра PORTx.
PINxn – біт n регістра PINx.
DDRxn – біт n регістра DDRx.
Розглянемо, що являє собою вивід мікроконтролера.

На вході мікроконтролера стоїть невеликий захист з двох діодів (1), призначений для захисту виводу мікроконтролера від короткочасних імпульсів напруги, що перевищують напругу живлення. Якщо напруга буде вище живлення, то верхній діод відкриється і цю напруга буде направлено на шину живлення, де з нею буде вже боротися джерело живлення і його фільтри.

Якщо на вивід потрапить негативна (нижче нульового рівня) напруга, то вона буде нейтралізована через нижній діод і погаситься на землю. Цей захист допомагає тільки від мікроскопічних імпульсів і перешкод. Якщо ж на ніжку мікроконтролера подати 6-7 Вольт при 5 Вольтах живлення, то внутрішні діоди його не врятують.

Конденсатор (2) – це паразитна ємність виводу. Зазвичай її не враховують. Ключі управління (3,4) підпорядковані логічній умові: коли умова виконується, ключ замикається. Кожен порт мікроконтролера AVR (вони зазвичай мають імена A, B та іноді C або навіть D) має 8 розрядів, кожен з яких прив'язаний до певної ніжки корпусу.

Кожен порт має три спеціальні регістра DDRx, PORTx і PINx (де x відповідає букві порту A, B, C або D).

– У регістрі PINxn міститься інформація про реальний поточний логічний рівень на виводах порту. Причому існує дві границі: межа гарантованого нуля і межа гарантованої одиниці – пороги, за якими ми можемо однозначно визначити поточний логічний рівень. При зниженні напруги від максимуму до мінімуму біт в регістрі PINx переключиться з 1 на 0; а коли напруга наростає від мінімуму до максимуму, перемикання біта з 0 на 1 буде тільки після досягнення напруги в 1.8 вольта.

– DDRxn – це регістр напрямку порту. Порт в конкретний момент часу може бути або входом або виходом (але для стану бітів PINxn це значення не має). DDRxy = 0 – вивід працює як ВХІД. DDRxy = 1 – вивід працює на ВИХІД.

– PORTxn – режим управління станом виводу. Коли ми налаштовуємо вивід на вхід, то від PORTx залежить тип входу. Коли ніжка налаштована на вихід, то значення відповідного біта в регістрі PORTx визначає стан виводу. Якщо PORTxn = 1 то на виводі логічна 1; якщо PORTxn = 0, то на виводі логічний 0.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: ОРГАНІЗАЦІЯ ПАМ'ЯТІ. ПАМ'ЯТЬ ПРОГРАМ, ОПЕРАТИВНА ПАМ'ЯТЬ І РЕГІСТРИ

Види пам’яті мікроконтролерів. Регістри

Мікроконтролери AVR мають в своєму складі три види пам'яті. По-перше, це оперативний запам'ятовувальний пристрій ОЗП (пам'ять даних). У документації фірми Atmel ця пам'ять називається SRAM. Об'єм ОЗП для різних мікроконтролерів змінюється від повної її відсутності до 2 Кбайт.

Другий вид пам'яті – це постійний запам'ятовувальний пристрій ПЗП (пам'ять програм). Вона виконана за Flash-технологією і призначена для зберігання керуючої програми. Об'єм програмної пам'яті в різних мікросхемах складає від 1 до 64 Кбайт. Пам'ять програм допускає стирання записаної інформації і повторний запис. Проте кількість циклів запису/стирання обмежена (до 1000 циклів).

Третій вид пам'яті – це енергонезалежна пам'ять даних. Вона також виконана за Flash-технологією, але в технічній документації вона називається EEPROM. Основне призначення цього виду пам'яті – довготривале зберігання даних навіть при вимкненому джерелі живлення. Пам'ять EEPROM допускає до 100000 циклів запису/стирання, а кількість циклів читання не обмежена. Об'єм пам'яті EEPROM порівняно невеликий і складає від 64 байт до 4 Кбайт.

Кожен з цих трьох видів пам'яті має свій власний адресний простір, і доступ до різних видів пам'яті здійснюється незалежно один від одного. Така побудова мікроконтролерів називається архітектурою Гарвардського типу.

В адресному просторі SRAM (пам’яті даних) розміщені регістри загального призначення, але для швидкої роботи з ними винесені фізично за межі SRAM.

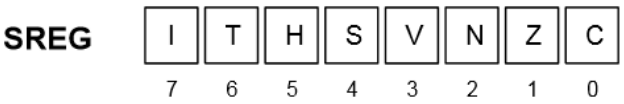
Регістри загального призначення можна поділити на 3 групи:

– Молодші R0 … R15. Ці регістри не можуть працювати з командами, що оперують з константами, наприклад, ми не можемо в R0 записати число (константу), але можемо скопіювати в нього число з будь-якого іншого регістра.

– Старші R16 … R31. Повноцінні регістри, що працюють майже зі всіма командами.

– Індексні R26 …R31. Вони можуть використовуватися як звичайні регістри загального призначення, але крім цього можуть утворювати регістрові 16-розрядні пари X(R26:R27), Y(R28:R29), Z(R30:R31), що використовуються як вказівники для непрямої адресації пам’яті.

Під час виконання арифметичних, логічних та порозрядних операцій ALU формує певні інформативні ознаки результату виконання у вигляді бітів регістру стану SREG (Status Register), які потім можуть бути використані в програмі для подальших арифметично-логічних операцій чи команд умовних переходів. Такі інформативні біти часто називають прапорцями.



I – загальний дозвіл переривань. При встановленні цього прапорця в одиничний стан дозволяється робота всієї системи переривань. Якщо прапорець скинутий (тобто встановлений в нуль) всі переривання заборонені. Прапорець I апаратно скидається відразу після виклику відповідної процедури обробки переривання й встановлюється при виконанні команди RETI, дозволяючи наступні переривання. Прапорець I може бути також встановлений і скинутий програмно за допомогою команд SEI і CLI відповідно.

T – біт користувача для тимчасового зберігання інформації. Біт використовується командами BLD (завантаження біта T) і BST (читання біта T) як елемент пам’яті для тимчасового зберігання інформації. Будь-який біт регістра загального призначення може бути скопійований у T, а потім вміст T може бути скопійований в будь-який інший біт того ж або іншого регістра.

H – прапорець половинного перенесення. Цей прапорець встановлюється в одиницю, якщо мало місце перенесення із молодшої половини байта (з 3-го розряду в 4-й) або позика зі старшої половини байта при виконанні деяких арифметичних операцій.

S – прапорець знаку. Цей прапорець є результатом операції «ВИКЛЮЧАЛЬНЕ АБО» (XOR) між прапорцями N (від'ємний результат) і V (переповнення числа в додатковому коді). Відповідно, цей прапор установлюється в одиницю, якщо результат виконання арифметичної операції менше нуля.

V – прапорець переповнення додаткового коду. Цей прапорець використовується при роботі зі знаковими числами (числами, які представлені у додатковому коді). Прапорець встановлюється в одиницю, якщо в результаті арифметичної операції відбувається переповнення числа, представленого у додатковому коді.

N – прапорець від'ємного значення. Цей прапорець встановлюється в одиницю, якщо в результаті арифметичної операції старший розряд результату дорівнює одиниці. Якщо старший розряд результату дорівнює нулю, то прапорець N теж дорівнює нулю.

Z – прапорець нуля. Цей прапорець встановлюється в одиницю, якщо результат операції, яка виконана, дорівнює нулю.

C – прапорець перенесення. Цей прапорець встановлюється в одиницю, якщо має місце переповнення результату (перенесення у старший розряд) при виконанні арифметичної операції.

Для збереження тимчасових даних та адрес повернення після виконання переривань чи підпрограм використовується стек. Стек розміщується в SRAM, як правило, у її кінцевій частині, та при заповненні росте у напрямку початку оперативної пам'яті. Вказівник стеку вказує на його початок, та представляє собою два 8-бітних регістра SPH:LPL. При використанні підпрограм та переривань вказівник стеку обов'язково має бути проініціалізований.

Модуль переривань має свої керуючі регістри з додатковим прапорцем дозволу глобального переривання у регістрі стану SREG. Усі переривання мають свої вектори переривань у таблиці векторів переривань, яка розташована на початку пам'яті програм (FLASH). Порядок пріоритетів переривань відповідає місцю знаходження вектора переривань у таблиці – переривання з найменшою адресою вектора має найвищий пріоритет.

Пам'ять програм (FLASH) у МК AVR містить команди для керування процесором, а також може використовуватися для зберігання константних табличних даних. FLASH має 16-бітну організацію пам'яті та її розмір сягає від 1 до 256 КБайт, у залежності від моделі. FLASH пам'ять допускає до 10 тис. разів перезапису.

EEPROM (енергонезалежна пам'ять даних) використовується для тривалого зберігання усяких налаштувань, переналаштувань, зібраних даних і т.п., тобто всіх даних, які слід зберегти до наступного запуску МК. Кількість циклів перезапису сягає 100 тис., що є небагато у порівнянні зі SRAM. Читання даних триває близько 4 тактів, однак запис відбувається дуже довго – близько 2-9 мсек. (це час виконання декількох тисяч команд).

Організація пам'яті

Адресація пам'яті даних здійснюється побайтно та повністю лінійна, без будь-яких поділів на сторінки, сегменти чи банки. Регістри заг. призначення, регістри керування та периферії, внутрішня оперативна пам'ять знаходяться в одному адресному просторі

Адресний простір SRAM поділений на декілька областей:

– $0000-$001F – знаходяться 32 регістри загального призначення. В асемблері ми звертаємося до них за їхніми безпосередніми назвами R0, R1 і т.д., однак можемо зчитувати та записувати у них дані за їхніми повними адресами у SRAM за допомогою групи команд LOAD/STORE.

– $0020-$005F – знаходяться 64 регістри вводу/виводу (порти, таймери, АЦП, регістри керування і т.п.). Ці регістри також мають свою внутрішню адресацію від $00 до $3F, яка використовується разом зі швидкими командами IN/OUT. Але можемо також працювати з ними через команди LOAD/STORE, використовуючи їхні повні адреси у SRAM.

– $0060-RAMEND – безпосередньо розміщується внутрішня статична оперативна пам'ять даних. Значення RAMEND залежить від моделі МК. Розміщувати наші дані в пам'яті можемо лише починаючи з адреси $0060.

Слід зазначити, що регістри загального призначення та регістри вводу/виводу не віднімають простір у пам'яті даних, а лише відсувають початок її адресації. Наприклад, модель

ATmega32A має на борту 2 КБайти SRAM (2048 байт чи 0x800). Відповідно, її значення RAMEND = 0x85F= 0x1F(рег.заг.призн.) + 0x3F(I/O порти)+ 0x800 (SRAM).

У деяких «нафаршированих» периферією моделях може не вистачати виділеної області адресації для регістрів вводу/виводу. Тому для таких моделей одразу після області регістрів вводу/виводу зарезервована додаткова область пам'яті 0x0060 - 0x00FF для додаткових 160 регістрів вводу/виводу. Слід зазначити, що з ними не працюють швидкі команди IN/OUT, і єдиний спосіб роботи з ними через команди LOAD/STORE. Відповідно, початок внутрішньої пам'яті даних вже починається з адреси 0x0100.

Ще один важливий момент стосується команд встановлення/очищення окремих бітів регістрів вводу/виводу SBI/CBI та команд перевірки стану біта у регістрі вводу/виводу SBIS/SBIC. Вони працюють лише з молодшими 32 регістрами вводу/виводу, що мають внутрішню адресу $00-$1F. Встановлення додаткової зовнішньої пам'яті RAM передбачено лише на окремих моделях, наприклад ATmega162.

Область пам'яті з адреси $0020 по адресу $005F суміщена з регістрами введення-виведення. Ці регістри дозволяють центральному процесору обмінюватися інформацією з вбудованими периферійними пристроями самого МК, такими, як таймери, компаратори, канали послідовної передачі інформації, система переривань, АЦП і т.п. Максимально можлива кількість РВВ рівна 64. Реальна кількість РВВ майже завжди значно менше їх максимально можливої кількості. Проте дана область пам'яті завжди використовується тільки для цієї мети. Якщо регістр існує, то існує і відповідна комірка пам'яті. Решта комірок цієї області ОЗП просто відсутня.

Енергонезалежна пам'ять даних (EEPROM) – це спеціальна внутрішня пам'ять, яка виконана за Flash-технологією і призначена для довготривалого зберігання даних. У сучасних мікроконтролерних пристроях часто виникає необхідність у зберіганні таких даних. Для подібних завдань зазвичай не вимагається великих об'ємів пам'яті, тому МК AVR мають об'єм EEPROM від 64 байт до 4 Кбайт.

До цієї пам'яті центральний процесор МК звертається не так, як до решти видів пам'яті. Для ЦП не існує адресного простору EEPROM і до неї він звертається за допомогою регістрів введення-виведення. Для МК з об'ємом EEPROM менше 256 байт таких регістрів три:

– EEAR – регістр адреси EEPROM. Працює тільки на запис. За допомогою цього регістра МК вибирає комірку, куди потрібно записати або звідки потрібно прочитати дані;

– EEDR – регістр даних EEPROM. Працює як на запис, так і на читання. Через цей регістр в EEPROM поступає байт, який записується. Через нього ж процесор отримує байт при читанні з EEPROM;

– EECR – регістр керування, який визначає режими роботи EEPROM. Саме через нього подаються команди читання і запису EEPROM.

Якщо об'єм EEPROM перевищує 256 байт, то замість одного регістра адреси (EEAR) такий МК має два регістри: EEARH і EEARL. Регістри доступу до EEPROM мають наступні номери:

EEAR - $1E; EEARL - $1E; EEARH - $1F; EEDR - $1D; EECR - $1C.

Способи програмування Flash- і eeprom- пам'яті

Мікроконтролери AVR допускають декілька способів програмування Flash- і EEPROM-пам'яті. Основні способи такі:

– паралельне програмування (Self-Prog);

– послідовне програмування з використанням SPI-интерфейса.

При паралельному програмуванні програматор передає в мікросхему записувані дані побайтно за допомогою 8-розрядної шини.

При послідовному програмуванні використовується спеціальний послідовний інтерфейс, що отримав назву SPI. За допомогою цього інтерфейсу дані передаються в мікросхему послідовно, біт за бітом, з використанням всього трьох ліній. Послідовний спосіб набагато повільніший, ніж паралельний. Але він більш універсальний і допускає програмування мікросхеми без виймання ВІС з плати. У табл. 2.1 у стовбці «ISP (I), Self-Prog (S)» для кожної мікросхеми показані підтримувані способи програмування.

Записувати інформацію в EEPROM можна також за допомогою програматора. Причому для

запису інформації в пам'ять програм і в EEPROM використовується один і той же програматор. Такий порядок доступу до пам'яті дозволяє при необхідності відмовитися від програмного перезапису EEPROM і використовувати цю пам'ять для зберігання будь-яких незмінних констант, що збільшує гнучкість системи.

Покажчик стеку призначений для організації стекової пам'яті. Запис в стекову пам'ять і читання з неї проводиться за принципом «останній зайшов – перший вийшов». У МК AVR застосовується широко поширений спосіб організації стекової пам'яті, коли стек є частиною ОЗП. Для реалізації принципу «останній зайшов – перший вийшов» і служить регістр – покажчик стеку.

Покажчик стеку у всіх МК AVR виконаний у вигляді двох 8-розрядних регістрів введення-виведення. Число фактично використовуваних розрядів для кожної моделі МК різне. У деяких моделях обсяг пам'яті даних настільки малий, що для покажчика стеку використається тільки молодший з регістрів покажчика стека SPL, а регістр SPH у них відсутній. Для МК з ОЗП великих розмірів до регістра SPL додається ще один регістр SPH. Разом вони складають один 16-розрядний покажчик стеку.

Вміст покажчика стеку зменшується на одиницю, коли дані записуються в стек за допомогою команди PUSH, і зменшується на два, коли в стек записується адреса повернення з підпрограми або процедури обробки переривання.

Покажчик стека збільшується на одиницю, коли дані читаються зі стеку за допомогою команди POP, і збільшується на два, коли дані читаються зі стека при виході з підпрограми (команда RET) або завершенні процедури обробки переривання (команда RETI).

Перед початком роботи в покажчик стеку необхідно записати адресу вершини стеку. Це деяка адреса комірки ОЗП, яка є старшою коміркою області пам'яті, виділеної під стек. Визначати розмір стекової пам'яті і адресу її вершини повинен сам програміст. Зазвичай вершину стеку встановлюють рівною адресі старшої комірки ОЗП.

ЛЕКЦІЯ 13. ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ AVR. ОБРОБКА ПЕРЕРИВАНЬ. СКИДАННЯ. РЕЖИМИ РОБОТИ ПРОЦЕСОРА

13.1 Програмування мікроконтролерів AVR

Система команд AVR є досить розвинута і нараховує до 118 (а в останніх розробках і до 137) різних інструкцій. Майже всі команди мають фіксовану довжину в одне слово (16 біт), що дозволяє в більшості випадків поєднувати в одній команді і код операції, і операнд(и). В останніх версіях кристалів mega реалізовано функцію апаратного множення.

Програмна модель мікроконтролерів включає частину структури мікроконтролера, що доступна програмісту за допомогою системи команд. В програмній моделі мікроконтролерів є декілька груп регістрів з різним функціональним призначенням: регістри загального призначення (РЗП), регістри введення/виведення (РВВ), статичний оперативний запам'ятовуючий пристрій (СОЗП), який у технічній літературі англійською мовою називають SRAM, ємністю 128 байт та постійний запам'ятовуючий пристрій (ПЗП) FLASH-типу, ємністю 4 Кбайт.

В усіх моделях мікроконтролерів, за винятком AT90S1200, використовується програмний стек. У цьому разі стек розміщується в ОЗП, і його глибина визначається тільки розміром вільної області ОЗП.

Пам'ять програм призначено для зберігання команд, які керують функціонуванням мікроконтролера. У пам'яті програм зберігаються також різні константи, що не змінюються під час роботи програми.

Пам'ять даних мікроконтролерів родини classic розділено на три частини: регістрова пам'ять, оперативна пам'ять (статичний ОЗП) і енергонезалежна пам'ять EEPROM. Регістрова пам'ять містить 32 регістри загального призначення (РЗП), об'єднаних у регістровий файл і службові регістри введення/виведення (РВВ). Розмір регістрової пам'яті фіксовано і для всіх моделей становить 96 байт, відповідно під РЗП виділяється 32 байти, а під РВВ – 64 байти. В області регістрів введення/виведення розташовані різні службові регістри (регістр вказівника стека, регістр стану і т. ін.), а також регістри керування периферійними пристроями, що входять до складу мікроконтролера. Загальна кількість РВВ залежить від конкретної моделі мікроконтролера. Для зберігання змінних також може використовуватися статичний ОЗП ємністю від 128 до 512 байт. Крім того, мікроконтролери AT90S4414 і AT90S8515 мають можливість підключення зовнішнього статичного ОЗП ємністю до 64 Кбайт. Для зберігання даних, що можуть змінюватися в процесі налаштовування і функціонування готової системи (калібрувальні константи, серійні номери, ключі тощо), може використовуватись EEPROM-пам'ять. Її ємність становить для різних моделей від 64 до 512 байт. Ця пам'ять лежить в окремому адресному просторі, а доступ до неї здійснюється за допомогою певних РВВ.

Тип звернення (адресації) до операндів (даних, що беруть участь в операції) називають способом адресації. Наявних способів адресації всього чотири: неявна, безпосередня, пряма, і непряма адресації. При неявній адресації в команді відсутня адреса операнда в явному вигляді. Інформацію про потрібну адресу операнда контролер отримує з коду операції команди, тобто кажуть, що операнд адресується неявно. При безпосередній адресації операнд входить в саму команду та читається із пам'яті програм у складі машинного коду команди. При прямій адресації адреси операндів містяться безпосередньо в слові команди. При непрямій адресації адреса комірки пам'яті (для AT90S1200 – регістра) міститься в одному з індексних регістрів X, Y і Z.

В мікроконтролерах AVR практично всі команди займають одну комірку пам'яті. Більшість команд виконується за один машинний цикл (1 такт). Всю множину команд мікроконтролерів AVR родини classic можна розділити на декілька груп:

- команди логічних операцій;
- команди арифметичних операцій і команди зсуву;
- команди операцій з бітами;
- команди пересилання даних;
- команди передачі керування;
- команди керування системою.

Усі мікроконтролери родини mega мають можливість самопрограмування, тобто самостійної зміни вмісту своєї пам’яті програм. Ця особливість дозволяє створювати на їхній основі дуже гнучкі системи, алгоритм роботи яких буде змінюватися самим мікроконтролером залежно від яких-небудь внутрішніх умов або зовнішніх подій. Для підтримки самопрограмування всю область пам’яті програм логічно розділено на дві секції – секцію прикладної програми (Application Section) і секцію завантажника (Boot 48 Loader Section). Програмування пам’яті програм здійснюється програмою-завантажником, яка розташована в однойменній секції.

13.2 Система переривань. Алгоритм роботи

Система переривань мікроконтролерів AVR призначена для обслуговування декількох джерел переривань, які поділяються на внутрішні і зовнішні.

Внутрішнім перериванням називається переривання, викликане одним з вбудованих периферійних пристроїв самого МК (наприклад, переривання по таймеру, аналоговому компаратору, АЦП і так далі).

Зовнішнім перериванням називається переривання, викликане сигналом, що поступає від зовнішнього джерела на спеціальний вхід МК.

Керування системою переривань здійснюється за допомогою спеціальних регістрів введення-виведення. Визначальним регістром тут є регістр статусу SREG, який розглядався вище. Сьомий біт цього регістру (біт I) є прапорцем загального дозволу переривань. Коли цей прапорець скинутий (тобто містить логічний нуль), всі переривання у МК заборонені. Для дозволу переривань потрібно встановити цей прапорець в одиницю.

Проте частіше всі види переривань не потрібні одночасно. Для того, щоб заборонити одні переривання і дозволити інші, застосовуються так звані маскуючі регістри (регістри маски). Регістр маски – це звичайний регістр введення-виведення, який призначений для керування окремими джерелами переривань. Кожному біту в регістрі маски відповідає одне джерело. Якщо біт скинуто в нуль, переривання цього виду заборонено. Якщо біт встановлений в одиничний стан, переривання дозволено.

В МК AVR застосовуються два регістри маски. Регістр GIMSK керує всіма видами переривань, окрім переривань від таймерів. В деяких МК сімейства Mega цей регістр називається GICR. Для керування перериваннями від таймерів є спеціальний регістр TIMSK.

Окрім регістрів маски для керування процесом виконання переривань існують ще два регістри. Це регістри прапорців переривань. Кожний біт такого регістра – це прапорець одного з видів переривань. Під час надходження запиту на переривання прапорець встановлюється в одиницю. За станом прапорця програма може визначити наявність запиту.

В певних режимах після встановлення прапорця процедура (підпрограма) обробки переривання викликається автоматично. Відразу після виклику процедури відповідний прапорець скидається. Мікроконтролери AVR мають два регістри прапорців: регістр GIFR (обслуговує ті ж переривання, що і регістр GIMSK) і регістр TIFR (прапорці переривань від таймерів).

Загальний алгоритм роботи системи переривань наступний. Після скидання мікроконтролера всі переривання заборонені (прапорці дозволу скинуті). Якщо програміст планує використовувати один з видів переривань, він повинен в своїй програмі встановити прапорець I загального дозволу переривань регістра статусу SREG в одиницю і дозволити це переривання шляхом встановлення у одиницю відповідного прапорця дозволу у регістрі маски.

Під час надходження запиту на переривання встановлюється прапорець відповідного переривання. Прапорець встановлюється навіть в тому випадку, якщо переривання заборонено. Якщо переривання дозволено, то МК приступає до його виконання. Поточна програма тимчасово припиняється, і керування передається за адресою відповідного вектора переривання на підпрограму обробки переривання.

В той же момент прапорець I автоматично скидається, забороняючи обробку інших переривань. Прапорець, який відповідає викликаному перериванню, також скидається, сигналізуючи про те, що мікроконтролер вже приступив до його обробки. Підпрограми обробки переривання обов'язково повинні закінчуватися командою повернення з переривання (RETI). По

цій команді керування передається в ту точку основної програми, у якій перервалася її робота. Прапор I при цьому автоматично встановлюється в одиницю, дозволяючи нові переривання.

Без вживання спеціальних заходів неможливі вкладені переривання. Поки обробляється одне переривання, вся решта переривань заборонена. Проте жодне переривання не залишається без обробки. При отриманні запиту на переривання відповідний прапорець обов'язково встановлений. В цьому стані він знаходитиметься до тих пір, поки дане переривання не буде оброблено.

Після закінчення обробки чергового переривання відбувається перевірка решти прапорців, і якщо є хоч одне необроблене переривання, МК переходить до його обробки. Якщо необроблених переривань виявиться декілька, то застосовується закон пріоритетів. Зі всіх переривань вибирається те переривання, пріоритет якого вище. Чим менше адреса вектора переривання, тим вище його пріоритет.

13.3 Режими роботи процесора

Всі мікроконтролери AVR мають багато режимів роботи. Деякі з режимів неможливо перемкнути програмним шляхом, використовуючи регістри керування. Для подібної мети фірма Atmel додала в свої мікроконтролери новий елемент настроювання МК – програмовані перемикачі режимів. Ці перемикачі виконані у вигляді спеціальних конфігураційних комірок, які, по суті, є ще одним видом перепрограмованої енергонезалежної пам'яті. Всі конфігураційні комірки об'єднуються в байти. Різні мікросхеми AVR мають від однієї до трьох байтів конфігураційних комірок.

Кожний конфігураційний перемикач має своє певне ім'я і призначений для того, щоб змінювати який-небудь параметр або режим роботи мікроконтролера. Деякі біти конфігураційних комірок з'єднані в групи. Різні моделі мікроконтролерів мають різні набори конфігураційних комірок. По термінології фірми Atmel конфігураційні комірки називаються Fuse Bits. Тому для зручності ці комірки часто називають Fuse-комірками**.**

Запис і читання конфігураційних комірок можливий тільки за допомогою програматора в режимі програмування. Всі незапрограмовані Fuse-комірки містять одиницю. При програмуванні в комірку записується нуль. Деякі з комірок програмуються ще на заводі. Стан всіх конфігураційних комірок для кожного конкретного МК наводиться в його документації.

Мікроконтролери AVR мають програмовані комірки захисту інформації. Це спеціальні комірки, подібні конфігураційним. Кожний МК має, як мінімум, дві захисні комірки: LB1 і LB2. Запис і читання цих комірок можливий тільки в режимі програмування. Запис нуля в LB1 блокує запис даних у Flash- і EEPROM-пам’ять. Одночасно блокується можливість змінювати конфігураційні комірки.

Якщо записати нуль ще і в LB2, то блокується і можливість читання всіх даних. Після цього скопіювати програму з Flash-пам’яті буде неможливо.

В мікроконтролерах сімейства Mega є додаткові комірки захисту BLB02, BLB01, BLB12 та BLB11. Вони призначені для обмеження доступу до різних областей пам'яті програм.

Ще одна група комірок – це комірки ідентифікації. Будь-який мікроконтролер має три комірки ідентифікації. Ці комірки доступні тільки для читання і містять інформацію про виробника і модель мікроконтролера.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: ТАЙМЕР. ЛІЧИЛЬНИКИ. СТОРОЖОВИЙ ТАЙМЕР

Таймери-лічильники

Будь-який мікроконтролер AVR містить декілька вбудованих таймерів. Причому по своєму призначенню їх можна розділити на таймери загального призначення та сторожовий таймер.

Сторожовий таймер призначений для перезапуску програми у разі збою у ході її виконання. Програма, що працює без збоїв, періодично скидає сторожовий таймер, не допускаючи його переповнювання. Сторожовий таймер має свій власний генератор, що працює на частоті 1 МГц. На вході таймера підключений передподільник вхідної частоти з програмованим коефіцієнтом ділення, що дає змогу регулювати часовий інтервал переповнення таймера і скидання МК. Сторожовий таймер може бути відключений програмним способом під час роботи МК.

Таймери загального призначення використовуються для формування різних інтервалів часу і прямокутних імпульсів заданої частоти. Крім того, вони можуть працювати в режимі лічильника і підраховувати тактові імпульси заданої частоти, вимірюючи таким чином тривалість зовнішніх сигналів, а також при необхідності підраховувати кількість будь-яких зовнішніх імпульсів.

З цієї причини такі таймери називають таймерами-лічильниками. В МК AVR застосовуються як 8-розрядні, так і 16-розрядні таймери-лічильники. Їх кількість для різних МК змінюється від одного до чотирьох. Всі таймери позначаються числами від 0 до 3. В технічній літературі їх часто іменують скорочено – T0, T1, T2, T3. Таймери T0 і T2 в більшості МК – 8-розрядні, а таймери T1 і T3 – 16-розрядні. Таймер T0 є в будь-якому МК AVR. Інші додаються по мірі ускладнення моделі.

Кожний 8-розрядний таймер є одним 8-розрядним регістром, який для МК є регістром введення-виведення. Цей регістр зберігає поточне значення таймера і називається рахунковим регістром. 16-розрядні таймери мають 16-розрядний рахунковий регістр. Кожний рахунковий регістр має своє ім'я.

Рахунковий регістр 8-розрядного таймера іменується TCNTx, де «x» – це номер таймера. Так, для таймера T0 регістр називається TCNT0, а для таймера T2 – TCNT2. 16-розрядні регістри іменуються схожим чином. Відмінність в тому, що кожний 16-розрядний рахунковий регістр для МК являє собою два регістри введення-виведення.

Один призначений для зберігання старших бітів числа, а другий - для зберігання молодших бітів. До імені регістра старших розрядів додається буква H, а для регістра молодших розрядів додається буква L. Таким чином, рахунковий регістр таймера T1 – це два регістри введення-виведення: TCNT1H і TCNT1L. Рахунковий регістр таймера T3 – це два регістри TCNT3H і TCNT3L.

МК може записати в будь-який рахунковий регістр будь-яке число у будь-який момент часу, а також у будь-який момент прочитати вміст будь-якого рахункового регістра. Коли таймер вмикається в режим рахунку, то на його вхід починають надходити рахункові імпульси. Рахунковими імпульсами можуть служити як спеціальні тактові імпульси, що виробляються усередині самого МК, так і зовнішні імпульси, що поступають на спеціальні входи МК. При переповненні рахункового регістра його вміст становиться рівним нулю, і рахунок починається спочатку.

Будь-який таймер зав'язаний із системою переривань. Викликати переривання може цілий ряд подій, пов'язаних з таймером. Наприклад, існує переривання по переповнюванню таймера, або по спрацьовуванню спеціальної схеми порівняння. Окремі переривання може викликати сторожовий таймер.

Режими роботи таймерів МК AVR

Таймери МК AVR можуть працювати в декількох режимах. Для вибору режимів роботи існують спеціальні регістри – регістри керування таймерами. Для простих таймерів використовується один регістр керування. Для більш складних – два регістри. Регістри керування таймером називаються TCCRx (де «x» – номер таймера). Наприклад, для таймера T0 використовується один регістр з ім'ям TCCR0. Для керування таймером T1 використовується два регістри: TCCR1A і TCCR1B. За допомогою регістрів керування проводиться не тільки вибір

відповідного режиму, але і більш тонка настройка таймера. Нижче розглянуті всі основні режими роботи таймерів.

Режим Normal – в цьому режимі таймер робить підрахунок імпульсів, що приходять на його вхід (від тактового генератора або зовнішнього пристрою) і викликає переривання по переповненню. Цей режим є єдиним режимом роботи для 8-розрядних таймерів більшості МК сімейства Tiny і для частини МК сімейства Mega.

Режим «Захоплення» – суть цього режиму полягає в збереженні вмісту рахункового регістра таймера в певний момент часу. Запам'ятовування відбувається або по сигналу, що поступає через спеціальний вхід МК, або від сигналу з виходу вбудованого аналогового компаратора.

Цей режим є зручним у тому випадку, коли потрібно вимірювати тривалість зовнішнього процесу (наприклад час, за який напруга на конденсаторі досягне певного значення). В цьому випадку напруга з конденсатора подається на один з входів компаратора, а на другий його вхід подається опорна напруга. В той момент, коли напруга на конденсаторі порівняється з опорною напругою, логічний рівень на виході компаратора зміниться на протилежний. По цьому сигналу поточне значення рахункового регістра запам'ятовується в спеціальному регістрі захоплення. Ім'я цього регістра ICRx (для таймера T0 це буде ICR0, для T1 – ICR1 і т. д.). Одночасно виробляється запит на переривання.

Режим «Скидання при співпаданні» – для роботи в цьому режимі використовується спеціальний регістр – регістр співпадання. Якщо МК містить декілька таймерів, то для кожного з них існує свій окремий регістр співпадання. Причому для 8-розрядних таймерів регістр співпадання – це один 8-розрядний регістр, а для 16-розрядних таймерів - це два восьмирозрядні регістри.

Регістри співпадання також мають свої імена. Наприклад, регістр співпадання таймера T1 складається з двох регістрів: OCR1L і OCR1H. Якщо регістр співпадання 16-розрядний, то фізично він складаються з двох регістрів введення-виведення.

Ці регістри включаються в роботу тільки тоді, коли вибраний режим «Скидання при співпаданні». В цьому режимі, як і в попередньому, таймер виконує підрахунок вхідних імпульсів. Поточне значення таймера з його рахункового регістра постійно порівнюється з вмістом регістрів співпадання.

Якщо таймер має два регістри співпадання, то для кожного з цих регістрів проводиться окреме порівняння. Коли вміст рахункового регістра співпаде з вмістом одного з регістрів співпадання, відбудеться виклик відповідного переривання. Окрім виклику переривання, у момент співпадання може відбуватися одна з наступних подій:

скидання таймера (виконується тільки для регістрів співпадання OCR1 і OCR1A);

зміна стану одного з виводів мікроконтролера (виконується для всіх регістрів).

Режим «Швидкодіючий шім» (Fast pwm) – сигнал з широтно-імпульсною модуляцією (ШІМ) часто використовується в пристроях керування електродвигунами постійного струму, нагрівальними елементами, освітлювальними приладами і т.п. Для цього замість постійної напруги на ці пристрої подається прямокутна імпульсна напруга. Змінюючи скважність імпульсів, можна змінювати середню напругу, прикладену до пристрою. Перевага імпульсного керування – у високому коефіцієнті корисної дії. Імпульсні керуючі елементи розсіюють набагато менше паразитної потужності, ніж керуючі елементи, що працюють в аналоговому режимі.

Сигнал з ШІМ формується на спеціальному виході МК. На вхід таймера подаються імпульси від системного генератора. Таймер знаходиться в стані безперервного рахунку. При переповнюванні таймера його вміст скидається в нуль, і рахунок починається спочатку. В режимі ШІМ переповнювання таймера не викликає переривань.

В деяких моделях мікроконтролерів таймер може працювати в асинхронному режимі. В цьому режимі на вхід таймера подається або частота від внутрішнього кварцового генератора, або від зовнішнього генератора. Лічильник не виробляє ніяких переривань і додаткових сигналів. В цьому режимі він працює як годинник реального часу. МК може встановлювати вміст рахункового регістра. А потім у будь-який момент він може прочитати цей вміст, отримавши, таким чином, поточне значення реального часу.

ЛЕКЦІЯ 14. РОБОТА З ПОРТАМИ ВВОДУ ТА ВИВОДУ. ПАРАЛЕЛЬНИЙ ТА ПОСЛІДОВНИЙ ВВІД-ВИВІД

14.1 Порти в архітектурі AVR

Всі порти в архітектурі AVR мають так звану Read-Modify-Write функціональність. Це значить, що в будь-який момент часу будь-яка ніжка порту може бути переключена на ввід або на вивід без будь-якого впливу на інші ніжки порту за допомогою інструкцій CBI і SBI.

З кожним набором ліній (який називається «портом») зв’язано три адреси вводу-виводу, які дозволяють визначати значення даних, записаних в порт, напрямок передачі даних(«1» – вивід, «0» – ввід) і реальне значення сигналу на зовнішньому виводі. Внаслідок цього є можливість «підтягнути» виводи порту до високого потенціалу для роботи в режимі вводу даних, читання даних може бути виконано або безпосередньо з зовнішнього виводу, або з виходу регістру даних порту. Така можливість є важливою відмінністю роботи порту. Якщо зовнішня лінія перевантажена або випадково закорочена на «землю», то стан зовнішнього виходу ніколи не буде мінятися. Ось чому в деяких випадках дуже важливо мати можливість прочитати вміст регістру порту і порівняти його з реальним станом зовнішнього виводу.

Хоча лінії вводу-виводу AVR працюють подібно до аналогічних ліній в інших мікроконтролерах, вони мають одну суттєву відмінність. Вона полягає в тому, що «підтягування» зовнішнього виводу до високого потенціалу управляється не окремими бітами регістра, а для цього використовується спеціальна комбінаційна схема. Ця схема дозволяє «підтягування» тільки тоді, коли зовнішній вивід працює в режимі вводу даних.

Взаємодія з портом здійснюється за допомогою трьох розташованих в просторі I/O регістрів: регістру даних – PORTx, регістру напрямку передачі даних, і регістру читання стану входу – PINx. Регістр вводу PINx доступний тільки для читання, регістри даних PORTx і напрямку передачі даних DDRx доступні як для читання, так і для запису. Всі виводи порту оснащені індивідуально підключеними вбудованими підтягуючими резисторами. Вихідні буфери виводів порту здатні забезпечити струм до 20 мА, що достатньо для прямого керування LED індикатором.

Якщо виводи порту використовуються в якості входів і зовнішні підтягнуті до низького рівня (нуля), то це може викликати зайве енергоспоживання у випадку активних внутрішніх підтягуючи резисторів до високого рівню (одиниці).

14.2 Послідовні порти вводу-виводу

Деякі мікроконтролери AVR мають:

– вбудований універсальний послідовний асинхронний приймач-передавач (UART);
– універсальний послідовний синхронно/асинхронний приймач-передавач (USART).

Канали UART (USART) призначені для обміну інформацією між МК і будь-яким зовнішнім пристроєм. Протокол UART (USART) – це досить поширений протокол послідовної передачі інформації. Такий протокол, зокрема, використовує послідовний порт комп'ютера. За допомогою UART (USART) можна організовувати лінію зв'язку не тільки між двома МК, але і між МК та комп'ютером.

Для обміну інформацією UART (USART) використовує дві лінії: RxD і TxD. Одна лінія використовується для приймання інформації, інша – для передавання. В модулях UART посилка може бути 8– або 9–розрядною. В модулі USART її довжина може складати від 5 до 9 розрядів. Крім того, модулі можуть виробляти і контролювати розряд парності.

Швидкість передавання визначається спеціальним внутрішнім програмованим подільником і частотою тактового генератора МК. Коефіцієнт ділення подільника може змінюватися від 2 до 65536. Для обміну інформацією через послідовний порт необхідно так підібрати коефіцієнт ділення і частоту тактового генератора, щоб отримати одну із стандартних швидкостей передачі інформації (2400, 4800, 9600, 14400, 19200, 28800 біт в секунду).

Послідовний периферійний інтерфейс (SPI) – це спеціальний послідовний інтерфейс, розроблений для зв'язку мікроконтролерів між собою. Канал SPI використовує для передачі інформації три лінії:

– лінію MISO (Master Input / Slave Output);
– лінію MOSI (Master Output / Slave Input);
– лінію SCK (Тактовий сигнал).

В мікроконтролерах AVR канал SPI може виконувати дві функції. По-перше, за допомогою цього інтерфейсу можна не тільки організувати послідовний канал обміну інформацією між двома МК, але і між МК і будь-яким периферійним пристроєм, що має SPI-інтерфейс. Існує цілий набір подібних пристроїв: цифрові потенціометри, ЦАП/АЦП, зовнішня Flash-пам'ять та ін.

Друге призначення каналу SPI – програмування мікроконтролера. Саме через цей канал здійснюється послідовне програмування пам'яті програм і пам'яті EEPROM. Перевага програмування через SPI полягає в тому, що такий спосіб дозволяє програмувати МК, не виймаючи його з пристрою. Це так зване внутрішньосхемне програмування. Зазвичай на платі пристрою, який налагоджується, передбачають спеціальні контакти, куди і підключається програматор.

Послідовний двохдротовий інтерфейс (TWI) є повним аналогом шини I^2C фірми Philips, що отримала широке розповсюдження в різних системах керування побутовою і промисловою технікою. Інтерфейс дозволяє об'єднати разом до 128 пристроїв, підключивши їх до однієї двохдротової шини. Шина I^2C містить лінію тактового сигналу SCL та лінію передавання даних SDA.

Інтерфейс дозволяє обмінюватися даними між ведучим пристроєм, яким звичайно є МК, і будь-яким із зовнішніх пристроїв, підключених до двохдротової лінії. При цьому ведучий пристрій може як передавати дані на відомий пристрій, так і приймати дані з нього.

Наявність інтерфейсу для роботи з I^2C шиною дозволяє застосовувати МК в системах керування телевізорів, радіоприймачів, магнітол, рідкокристалічних дисплеїв на і т.п.

14.3 Паралельні порти вводу-виводу

Паралельні порти вводу/виводу є основною складовою будьякого МК AVR. Ноги портів, окрім основного призначення – побайтного чи побітного вводу/виводу, можуть також використовуватися для роботи з певною внутрішньою периферією МК (у залежності від того, які альтернативні функції прикріплені до них).

Модель ATmega32A має в себе на борту 4 повних порти вводу/ виводу: A, B, C та D. Налаштування та керування портом можемо виконувати як усім одразу, так і кожною його ногою окремо.

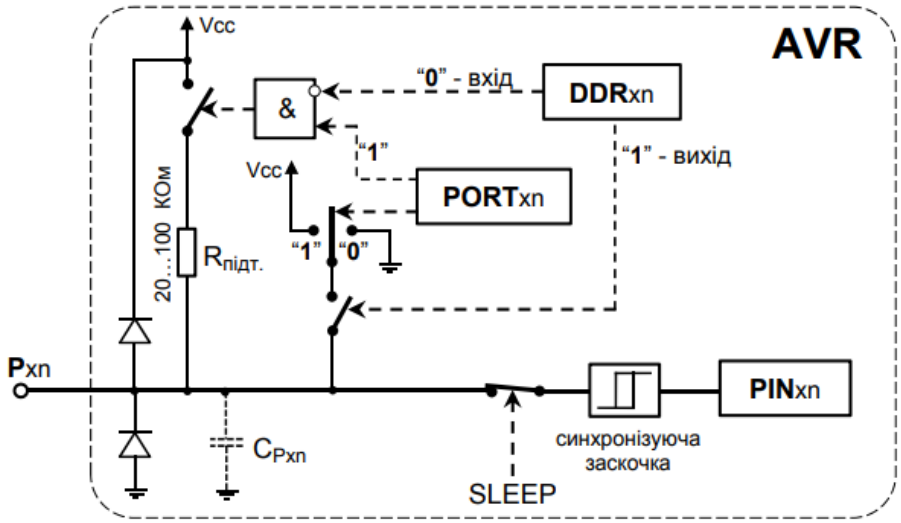
Усі ноги портів мають внутрішній діодний захист, який захищає від стрибків підвищеної напруги (верхній діод відкривається при напрузі вищій за напругу живлення МК, і з перепадом напруги вже борються фільтри БЖ) та від’ємної напруги (яка нейтралізується на землю нижнім діодом). Цей діодний захист призначений лише від імпульсних завад, і при перевищеній напрузі нога порту ймовірно вийде з ладу.

Варто мати на увазі, що на кожній нозі присутня незначна внутрішня паразитна ємність. За керування ногами портів МК AVR відповідає набір регістрів DDRx, PORTx та PINx (літера x відповідає назві регістра A, B, C чи D).

DDRx – ці регістри визначають напрямок роботи портів: на вхід чи на вихід. При цьому ми можемо налаштувати індивідуально кожну ногу вибраного порту. Якщо біт у регістрі, що відповідає потрібній нозі порту, дорівнює «0», тоді ця нога працює на вхід, якщо ж «1» – тоді на вихід.

PORTx – значення бітів цих регістрів визначають налаштування виводів портів у залежності від значення, встановленого у регістрі DDRx. Якщо вивід працює на вихід, тобто біт у

регістрі DDRx встановлений в «1», тоді значення відповідного біта регістра PORTx вказує на рівень напруги на нозі. Значення «1» встановлює на нозі напругу високого рівня (Vcc, напруга живлення МК), значення «0» встановлює низький рівень (цифрова земля). Якщо вивід працює на вхід, тобто біт у регістрі DDRx встановлений в «0», тоді значення «1» відповідного біта регістра PORTx підключає внутрішній резистор (Rпідт.), який підтягує цей вивід до напруги живлення МК. Номінал цього підтягуючого резистора є не точним і знаходиться в межах 20-100 КОм. Значення «0» залишає цей вивід у високоімпедансному стані Hi-Z (без підтягуючого резистора), тобто повний опір входу є дуже високим та не впливає на зовнішнє підключення. У стані Hi-Z входи МК використовуються для підключення до сторонніх шин даних, не заважаючи при цьому їхній роботі.



PINx – ці регістри призначені лише для читання та, незалежно від налаштування портів вводу/виводу, завжди відображають поточні стани виводів МК. Для стабільності зчитування станів виводів МК є присутня синхронізуюча ланка, що складається з тригерної заскочки та розряду PINxn. Значення сигналу на виводі МК фіксується заскочкою при низькому рівні сигналу та перезаписується потім у розряд PINxn з наростаючим фронтом тактового сигналу. Відповідно, між операціями запису у порт та зчитуванням його стану є присутня затримка. Тому у програмах між командами out та in одного порту необхідно вставляти пустий оператор nop.

Значення окремих бітів регістрів PINx відповідають реальним рівням сигналів на виводах портів. «1» – при високому рівні, «0» – при низькому рівні. Якщо виводи налаштовані як високоімпедансні входи, та немає підключення зовнішніх сигналів, тоді значення відповідних бітів PINx будуть рівними «0». Однак, від найменших завад, наприклад дотик пальця до корпусу МК, на виході може з'явитися «1». Саме по цій причині непідключені входи до шин сигналів необхідно підтягувати через резистор до напруги живлення чи землі.

ЛЕКЦІЯ 15. АНАЛОГОВО-ЦИФРОВЕ ПЕРЕТВОРЕННЯ. АНАЛОГОВИЙ КОМПАРАТОР

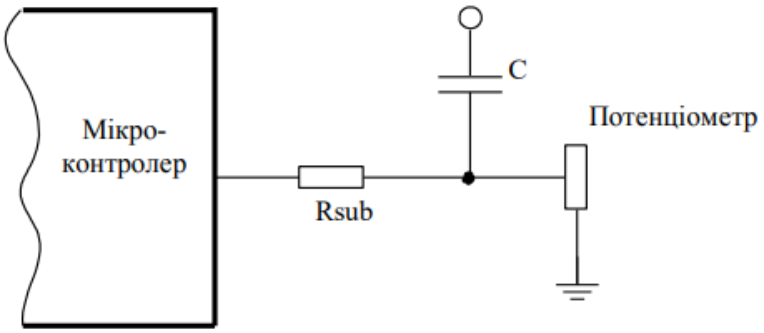
15.1 Аналогове введення-виведення

Світ, що оточує мікроконтролер, складається не тільки з одиниць і нулів, у дійсності там є багато значень між нулем і одиницею. Часто мікроконтролер повинен взаємодіяти з аналоговими пристроями, які працюють з сигналами, що мають рівень між напругою живлення Ucc і «землею», вводити і виводити такі аналогові сигнали.

Багато моделей мікроконтролерів у різних сімействах містять аналогово-цифрові (ADC - Analog-to-Digital Converter) і цифрово-аналогові (DAC – Digital-to-Analog Convener) перетворювачі.

Існує три способи введення аналогового сигналу в мікроконтролер. Перший спосіб – використання датчика, за допомогою якого мікроконтролер визначає фізичне положення движка потенціометра. Другий спосіб – увімкнення аналогового компаратора, що визначає чи знаходиться значення напруги, що надходить, вище чи нижче заданого рівня (опорної напруги). Третій тип – використання мікроконтролера з інтегрованим на кристалі аналого-цифровим перетворювачем (АЦП), що забезпечує вимірювання значення напруги, що надходить на вхід. Кожний з цих джерел має визначені переваги для різних областей застосування.

У першому способі аналого-цифрове перетворення фактично не реалізується, а поточне значення опору потенціометра визначається за допомогою введення-виведення цифрових даних. Для визначення опору потенціометра до виводу мікроконтролера вмикається простий RC-ланцюг. Опір визначається шляхом вимірювання часу, протягом якого потенціал на конденсаторі залишається більшим за поріг перемикання. Чим більший опір, тим більше число "1" буде підраховано на вході за час вимірювання. Щоб виконати вимірювання, вивід паралельного порту переводиться в режим виходу, на якому встановлюється "1" (високий потенціал). Конденсатор розряджається через опір Rsub, що обмежує струм, запобігаючи коротке замикання на початку розряду. Коли конденсатор цілком розрядився, вихідний драйвер закривається, і конденсатор починає заряджатися через потенціометр. Вимірювання закінчується, коли напруга на виводі упаде нижче порога перемикання. Звичайно, для вимірювання часу використовується таймер.



15.2 Аналоговий компаратор напруг

Наступний спосіб аналого-цифрового перетворення, який звичайно використовується у мікроконтролерах, – це аналоговий компаратор напруг. Компаратор являє собою просту схему, що порівнює дві напруги: вхідну й опорну (Uref), і встановлює на виході 1, якщо вхідна напруга більша, ніж опорна. Цей спосіб найбільш зручно використовувати в таких пристроях, як термостати, де необхідно контролювати досягнення визначеного рівня вимірюваної величини, що задається значенням вхідної напруги.

Часто в мікроконтролерах, що використовують компаратори, опорну напругу формують всередині за допомогою резистивного подільника й аналогового мультиплексора, що робить вибір необхідної вихідної напруги. Така схема забезпечує одержання деякого набору опорних напруг. Використовується простий алгоритм перебирання набору різних опорних напруг до спрацьовування компаратора. Опорна напруга, при якій відбувається перемикання компаратора, відповідає значенню вхідної напруги, що надходить. Дана схема дає наближене значення вхідної напруги, тому що звичайна напруга Uref задається з досить великим кроком.

Наприклад, якщо схема забезпечує 8 рівнів опорної напруги, то при діапазоні напруг 5 В

крок складе більше 700 мВ.

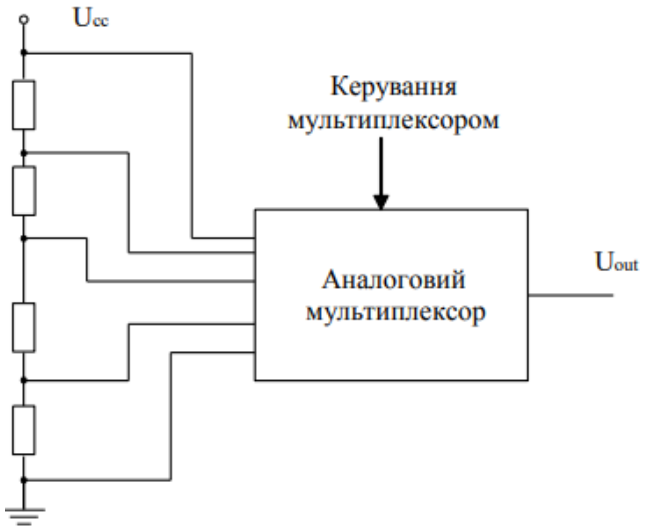


Рис. Отримання опірних напруг за допомогою подільника

Інший спосіб аналого-цифрового перетворення – це використання паралельно увімкнутих компараторів. Цей метод є найбільш швидким у порівнянні з іншими методами перетворення ADC. Час перетворення визначається затримкою компараторів і пріоритетного дешифратора. Цей спосіб є відносно дорогим, тому що вимагає використання великого числа компараторів. Наприклад, щоб одержати 8-розрядну точність перетворення буде потрібно 256 компараторів.

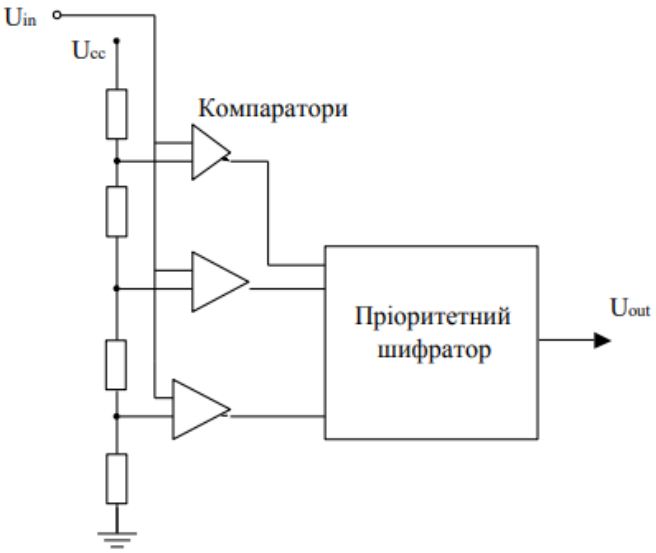


Рис. Паралельний цифроаналоговий перетворювач

В останньому способі аналого-цифрового перетворення використовується компаратор і аналогове джерело напруги (генератор пилкоподібної напруги), що лінійно збільшується, починаючи з 0 В и до Ucc. Ця схема називається інтегрувальним АЦП. На початку перетворення відбувається скидання таймера, і на виході генератора встановлюється 0 В. Потім відбувається запуск таймера і генератора. Коли напруга на виході генератора перевищить значення Uin, таймер зупиняється, і виробляється сигнал «кінець перетворення» (ADC stop), що може викликати переривання мікроконтролера. При цьому вміст таймера буде пропорційний значенню напруги Uin.

Цей метод забезпечує дуже високу точність перетворення. Однак при його реалізації виникає ряд проблем. Перша – це час, необхідний для перетворення. Чим вища напруга Uin, тим більший час перетворення.

Інша проблема – зміна значення вхідного сигналу під час перетворення. Ця проблема

вирішується шляхом використання конденсатора, що швидко заряджається до рівня вхідної напруги. Потім напруга на конденсаторі порівнюється з напругою на виході генератора.

15.3 Аналогово-цифровий перетворювач

Аналого-цифровий перетворювач (АЦП) призначений для перетворення аналогового сигналу в цифрову форму. Перетворювач вимірює величину сигналу і видає на виході цифровий код, який відповідає цій величині. АЦП застосовуються в мікропроцесорних системах керування різними аналоговими пристроями та процесами. Наприклад, мікропроцесорний стабілізатор напруги, цифровий вольтметр і т.п.

В МК AVR застосовується 10-розрядний АЦП послідовного наближення. МК, що мають в своєму складі вбудований АЦП, обов'язково мають роздільне живлення для цифрової і для аналогової частин схеми. Тому вони мають два виводи живлення і два виводи загального дроту.

Крім того, один з виводів зарезервований для подачі на мікросхему зовнішньої опорної напруги. Опорна напруга використовується в схемі АЦП для оцінки рівня вхідного сигналу. Від стабільності опорної напруги залежить точність вимірювання.

Кожний АЦП забезпечений багатоканальним аналоговим комутатором (мультиплексором), який дозволяє вимірювати аналогову напругу з декількох різних входів. Кількість входів АЦП у різних МК різна і становить від 4 до 16.

Сигнал, що вимірюється, прикладається між відповідним входом АЦП і аналоговим загальним дротом. Такі входи називаються несиметричними. В деяких мікроконтролерах є режим, в якому входи АЦП об'єднуються попарно і утворюють диференціальні входи. Диференціальні входи відрізняються від звичайних тим, що сигнал, що виміряється, прикладається між двома входами: прямим і інверсним. Перешкоди, що при цьому наводяться, компенсуються, а корисний сигнал проходить без змін. Такі входи називаються симетричними.

Процес перетворення напруги в код займає 13 або 14 тактів. За цей час відбувається підбір коду методом послідовних наближень. Після закінчення процесу перетворення виробляється запит на переривання. Результат перетворення записується в пару регістрів ADCH, ADCL. З шістнадцяти розрядів цієї регістрової пари використовуються тільки 10, а інші дорівнюють нулю. Причому можуть використовуватися або десять старших розрядів (ADCH7-ADCH0, ADCL7, ADCL6), або десять молодших розрядів (ADCH1, ADCHO, ADCL7-ADCL0). Це залежить від вибраного режиму роботи.

АЦП можуть працювати як в одиночному режимі, так і в безперервному. В безперервному режимі цикли перетворення йдуть один за одним. В одиночному режимі процес перетворення запускається однократно від однієї з наступних подій: переривання від аналогового компаратора; зовнішнього переривання INT0; переривання по події «Співпадання» одного з таймерів; переривання по переповненню одного з таймерів; переривання по події «Захоплення» одного з таймерів.

Керування всіма режимами роботи АЦП здійснюється за допомогою двох спеціальних регістрів: ADMUX і ADCSR. Регістр ADMUX призначений для керування вхідним аналоговим мультиплексором, а регістр ADCSR – для вибору режиму роботи АЦП.

Процес перетворення в АЦП синхронізується від внутрішнього генератора мікроконтролера. Тактовий сигнал від генератора поступає на АЦП з попереднього подільника з програмованим коефіцієнтом ділення. Коефіцієнт ділення залежить від значення розрядів ADPSO, ADPS1 і ADPS2 регістра ADCSR і може приймати значення 2, 4, 8, 16, 32, 64 і 128.

Найбільша точність перетворення досягається тоді, коли тактова частота перетворення знаходиться в діапазоні 50–200 кГц. Тому рекомендується вибирати такий коефіцієнт ділення, щоб тактова частота модуля АЦП знаходилася в цьому діапазоні.

ЛЕКЦІЯ 16. МОВА АСЕМБЛЕРА МІКРОКОНТРОЛЕРІВ. ОГЛЯД КОМАНД

16.1 Мова Асемблер

Перш ніж почати розробку якого-небудь пристрою на базі мікроконтролера, дуже важливо ознайомитися з основами програмування мовою Асемблера. Якщо Ви вивчили мову Асемблер для якого-небудь мікропроцесора чи мікроконтролера й освоїли процес розробки асемблерних програм, то у Вас не виникнуть великі проблеми при написанні програм для іншого процесора.

Навіть якщо Ви розробляєте програмне забезпечення мовою високого рівня, то знання Асемблера дозволить визначити, які команди реалізує процесор, щоб зрозуміти, що відбувається в системі при виконанні програми.

Щоб процес вивчення мови, написання і налагодження програм на Асемблері був більш простим і зрозумілим, використовується два прийоми. Перший – візуалізація процедур виконання команд процесором. Другий – використання методів структурного програмування, щоб зробити програми більш простими для читання і розуміння.

Візуалізацію виконання команд найкраще здійснити, використовуючи структурну схему процесора чи мікроконтролера, на якій відзначається проходження даних при виконанні кожної команди. В результаті забезпечується гарне візуальне зображення процесу виконання команд.

Для програмування мікроконтролерів можна використовувати різні мови високого рівня. Термін «мова високого рівня» слугує для означення мов, використовуваних для написання програм, що легко читаються, і які конвертуються (компілюються) у мову асемблера, а потім перетворюються в об’єктний код (біти і байти) для їхнього виконання мікроконтролером.

Це звичайні мови загального призначення, що конвертовані для створення об’єктного коду, виконуваного наявним у мікроконтролері процесором, і забезпечують реалізацію деяких специфічних функцій мікроконтролера.

Є безліч компіляторів, розроблених для різних мікроконтролерів, причому ефективність об’єктного коду, що вони генерують, варіюється від дуже поганої до дуже гарної. Кращими є компілятори, що надають користувачу велику частину ресурсів мікроконтролера. Ефективність отриманого об’єктного коду визначається обсягом зайнятої пам’яті програм, необхідним обсягом оперативної пам’яті для збереження даних, і кількістю ресурсів, необхідних для підтримки відкомпільованого коду. Найбільш популярні мови високого рівня – це C, BASIC і Forth.

У мікроконтролерах програмний код повинний мати доступ до апаратних регістрів. Звичайно, це робиться двома методами.

Перший – дозволити вставку асемблерних інструкцій у тіло програми, написаної мовою високого рівня.

Другий – дозволити користувачу визначити деякі позиції в адресному просторі даних для звертання до цих регістрів. Обидва методи є відхиленням від стандарту мови, прийнятого для персональних комп’ютерів і робочих станцій. Але ці методи, звичайно, підтримуються мовами високого рівня, використовуваними для програмування мікроконтролерів.

Застосування процедур макросів і умовної компіляції дозволяє спростити розробку і читання (а виходить, і розуміння) програм. Ці засоби дозволяють писати узагальнений програмний код, що може бути використаний для багатьох специфічних випадків, замість того, щоб створювати спеціальне програмне забезпечення для кожного з цих випадків окремо. Макрос може розглядатися як функція, що заміщає в програмі її оператор (макровиклик), у той час як підпрограма розміщується поза основним програмним кодом.

16.2 Система команд мікроконтролерів AVR

Система команд МК AVR налічує до 133 різних команд. Розрізняють наступні групи команд AVR: команди логічних операцій, арифметичних операцій, команди операцій з розрядами, команди порівняння, команди зсуву, команди пересилання даних, команди керування МК, команди умовного та безумовного передавання керування. За кількістю реалізованих команд МК AVR більше схожі на CISC, ніж на RISC-мікроконтролери.

Систему команд МК AVR наведено у таблиці, де використано такі позначення:

SREG – регістр статусу;

РЗП – регістр загального призначення;
РВВ – регістр введення-виведення;
Z, C, N, V, H, T, I, S – прапорці регістру статусу;
Rd – регістр призначення у регістровому файлі;
Rr – регістр-джерело у регістровому файлі;
K – байт даних (константа);
k – величина зсуву відносно стану лічильника команд при відносних переходах;
b – біт в регістровому файлі або у регістрі введення-виведення;
s – біт в регістрі статусу;
X, Y, Z – регістри непрямої адресації (X=R27:R26, Y=R29:R28, Z=R31:R30).
Для прикладу вибрані деякі команди.

Система команд мікроконтролерів AVR				
Назва команди	Мнемокод	Операція	Цикли	Прапорці
Команди логічних операцій				
Логічне І двох РЗП	AND Rd, Rr	Rd ← Rd Λ Rr	1	Z, N,V
Логічне АБО двох РЗП	OR Rd, Rr	Rd ← Rd V Rr	1	Z, N,V
Логічне АБО РЗП і константи	ORI Rd, K	Rd← Rd V K	1	Z, N,V
Перетворення у зворотний код вмісту РЗП	COM Rd	Rd ← 0FFH - Rd	1	Z, C, N, V
Перетворення у додатковий код вмісту РЗП	NEG Rd	Rd ← 00H - Rd	1	Z, C, N, V, H
Скидання розрядів РЗП	CLR Rd	Rd ← 00H	1	Z, N,V
Команди арифметичних операцій				
Додавання двох РЗП	ADD Rd, Rr	Rd ← Rd + Rr	1	Z, C, N, V, H
Віднімання двох РЗП	SUB Rd, Rr	Rd ← Rd - Rr	1	Z, C, N, V, H
Декремент РЗП	DEC Rd	Rd ← Rd - 1	1	Z, N,V
Інкремент РЗП	INC Rd	Rd ←Rd + 1	1	Z, N,V
Команди операцій з розрядами				
Скидання розряду(ів) РЗП	CBR Rd, K	Rd ←Rd Λ (0FFH - K)	1	Z, N,V
Встановлення розряду(ів) РЗП	SBR Rd, K	Rd←Rd Λ K	1	Z, N,V
Команди порівняння				
Порівняння двох РЗП	CP Rd, Rr	?(Rd - Rr)	1	Z, N, V, C, H
Команди зсуву				
Арифметичний зсув праворуч	ASR Rd	Rd7→ d6→Rd5→ Rd4→Rd3→Rd2→Rd1→Rd0	1	Z, C, N, V

Продовження таблиці				
Команди пересилання даних				
Пересилання між РЗП	MOV Rd,Rr	Rd←Rr	1	-
Запис у програмну пам'ять	SPM	{Z}←R1:R0	-	-
Пересилання із РВВ у РЗП	IN Rd, P	Rd<-P	1	-
Пересилання з РЗП у РВВ	OUT P, Rr	P←Rr	1	-
Завантаження байта у стек	PUSH Rr	STACK ← Rr	2	-
Витягування байта зі стеку	POP Rd	Rd ← STACK	2	-
Команди керування МК				
Скидання сторожового таймера	WDR	-	1	-
Припинення програми	BREAK	Застосовується для відладки	-	-
Команди передавання керування (безумовне передавання керування)				
Відносний безумовний перехід	RJMP	PC ← PC + k + 1	2	-
Непрямий безумовний перехід	IJMP	PC←Z	2	-
Відносний виклик підпрограми	RCALL	PC ← PC + k + 1	3	-
Непрямий виклик підпрограми	ICALL	PC←Z	3	-
Повернення з підпрограми	RET	PC ← STACK	4	-
Повернення з підпрограми обробки переривань	RETI	PC ← STACK	4	I
Команди передавання керування (пропускання наступної команди за умовою)				
Порівняння й пропускання наступної команди при рівності	CPSE Rd, Rr	Якщо Rd = Rr	1/2/3	-
Пропускання наступної команди, якщо розряд РЗП скинутий	SBRC Rr, b	Якщо Rr.b = 0	1/2/3	-
Команди передавання керування за умовою				
Перехід, якщо прапорець s регістра SREG скинутий	BRBC s, k	Якщо SREG.s = 0	1/2	-
Перехід, якщо прапорець s регістра SREG встановлений	BRBS s, k	Якщо SREG.s = 1	1/2	-
Перехід за умовою «дорівнює»	BREQ k	Якщо Z = 1	1/2	-

ЛЕКЦІЯ 17. ВИКОНАННЯ АСЕМБЛЕРНОГО КОДУ У КОДІ МОВИ С

17.1 Середовища для програмування на мові С

Для мікроконтролерів AVR існують різні мови програмування, але, мабуть, найбільш придатними є асемблер і С, оскільки в цих мовах в найкращій мірі реалізовані всі необхідні можливості по управлінню апаратними засобами мікроконтролерів.

Асемблер – це низькорівнева мова програмування, що використовує безпосередній набір інструкцій мікроконтролера. Асемблер програє С в швидкості і зручності розробки програм, але має помітні переваги в розмірі кінцевого виконуваного коду, а відповідно, і швидкості його виконання. С дозволяє створювати програми з набагато більшим комфортом, надаючи розробнику всі переваги мови високого рівня.

Архітектура і система команд AVR створювалася за безпосередньої участі розробників компілятора мови С і в ній враховані особливості цієї мови.

Щоб перетворити вихідний текст програми у файл прошивання мікроконтролеру, застосовують компілятори. Фірма Atmel поставляє потужний компілятор асемблера, який входить в середовище розробки AVR Studio, що працює під Windows. Поряд з компілятором, середовище розробки містить відладчик і емулятор. AVR Studio абсолютно безкоштовна і доступна на сайті Atmel .

В даний час представлено досить багато компіляторів С для AVR. Найпотужнішим з них вважається компілятор фірми IAR Systems. IAR C Compiler має широкі можливості по оптимізації коду і поставляється в складі інтегрованого середовища розробки IAR Embedded Workbench (EWB), що включає в себе також компілятор асемблера, лінкер, менеджер проектів і бібліотек, а також відладчик.

Фірмою Image Craft випускається компілятор мови С, який отримав досить широку популярність. Image Craft C Compiler володіє непоганим рівнем оптимізації коду і досить низькою ціною Не меншу популярність завоював Code Vision AVR C Compiler. Компілятор поставляється разом з інтегрованим середовищем розробки, в яке, крім стандартних можливостей, включена досить цікава функція – CodeWizardAVR Automatic Program Generator. Наявність в середовищі розробки послідовного терміналу дозволяє виробляти налагодження програм з використанням послідовного порту мікроконтролера.

Інтегроване середовище розробки WinAVR включає потужні компілятори С і асемблера, програматор AVRDUDE, відладчик, симулятор і безліч інших допоміжних програм і утиліт. WinAVR прекрасно інтегрується з середовищем розробки AVR Studio від Atmel. Асемблер ідентичний по вхідному коду асемблеру AVR Studio. Компілятори С і асемблера мають можливість створення налагоджувальних файлів у форматі COFF, що дозволяє застосовувати не тільки вбудовані засоби, але і використовувати потужний симулятор AVR Studio. Ще одним важливим плюсом є те, що WinAVR поширюється вільно без обмежень (виробники підтримують GNU General Public License).

17.2 Області пам’яті AVR та їх використання в мові С

Мікроконтролери AVR реалізовані за принципом гарвардської архітектури, де пам'ять програм і пам'ять даних, як і шини доступу до них, розділені. Такий поділ дозволяє, наприклад, одночасно зчитувати нову асемблерну інструкцію з пам'яті програм по одній шині і в той же момент записувати результат попередньої команди в пам'ять даних мікроконтролера (ОЗП/ПВВ/РЗП) за іншою. Такий «поділ праці» називається конвеєром (pipeline) і дозволяє виконувати по одній команді за такт.

Одна з ключових складових AVR мікроконтролера є 8-бітна Шина Даних (8-bit Data Bus), що зв'язує між собою ОЗП, лічильник команд, регістр стану, регістри периферійних пристроїв а також RALU (Регістри загального призначення + АЛП) мікроконтролера. Пам'ять даних являє собою сукупність регістрів загального призначення (РЗП), регістрів вводу/виводу а також статичного ОЗП (SRAM).

У AVR є кілька наборів асемблерних команд які призначені спеціально для адресації, до регістрів вводу/виводу (зі своїм адресним простором), а також набір команд для адресації через

шину даних до всієї області пам'яті даних (зі своїм адресним простором).

При програмуванні на мові С, програміст може оперувати декількома типами даних:

Char – 8-бітна змінна.

Int, short – 16-бітові змінні.

Long, float, double – 32-бітові змінні.

А також struct, union.

Якщо ми оголосимо змінну всередині функції, тобто зробимо її локальною, то компілятор виділить для неї один або кілька регістрів загального призначення. Якщо в даний момент не вистачає РЗП, компілятор помістить змінну в оперативну пам'ять (точніше в Стек), що надалі вимагатиме додаткові такти для зчитування змінної з ОЗП/Стека. Якщо ми оголосимо змінну за межами функції (зробимо її глобальною) або визначимо її як static, то компілятор не роздумуючи виділить для неї місце в ОЗП пам'яті.

Тому слід намагатися, по можливості, оголошувати змінні всередині функцій, тим самим надавши компілятору можливість для вибору найбільш оптимального розміщення змінних. Також в мові С можна рекомендувати (необов'язковий для виконання) компілятору помістити ту чи іншу змінну в РЗП за допомогою ключового слова register.

register unsigned char value; // може бути поміщена в РЗП

static unsigned int data; // буде поміщена в ОЗП

Оскільки AVR, як і більшість 8-бітних мікроконтролерів, не вміє апаратно працювати з 16-/32-/64-бітними змінними, для таких випадків розроблені бібліотечні функції, що входять до складу компіляторів високорівневих мов, таких як С. Ці функції реалізують велику кількість різноманітних математичних/логічних і ін. операцій, таких як множення/ділення 16-/32-/64-бітних змінних зі знаком і без, робота з числами з плаваючою комою, робота зі структурами, бітовими полями і т. д.

Але використання змінних даного типу, а значить і пов'язані з ними функції, призводить до збільшення числа тактів необхідних для читання/запису а також регістрів загального призначення, використовуваних при адресації до таких змінних, а найголовніше, хоча і невелике, але додаткове місце в пам'яті програм, для зберігання цих функцій.

Додаткові модифікатори:

eeprom – розмістити змінну в EEPROM. Це незалежна пам'ять – значення таких змінних зберігається при виключенні живлення і при перезавантаженні МК.

Приклад:

eeprom unsigned int x;

Якщо це перша змінна в EEPROM, то її молодший байт буде поміщений в клітинку 1 EEPROM, а старший в клітинку 2.

Щоб її використовувати, потрібно додати наступний заголовки: #include <avr / eeprom.h>

volatile – використовують, якщо потрібно запобігти можливості пошкодження вмісту змінної в перериванні, і не дозволити компілятору спробувати викинути її при оптимізації коду. Приклад: volatile unsigned char x;

Програма мікроконтролера по суті являє собою один великий масив, кожна осередок якого містить одну асемблерну інструкцію, в свою чергу одна асемблерна інструкція займає 2 байта пам'яті програм. Термін дії пам'яті програм в AVR мікроконтролери розрахований на 10000 циклів запису / стирання і може зберігати інформацію протягом 20 років при температурі в 85 ° С і до 100 років при температурі в 25 °С.

В мові С виконується тільки той код, що міститься в головній функції (main function). Стосовно мікроконтролерів це не зовсім так. Найпростіша програма, написана на мові С:

```
int main ( void )
{
  return 0;
}
```

У AVR GCC компілятора є одна особливість: він вважає за краще, щоб функція main повертала значення цілого типу (int). В іншому випадку, компілятор генерує попередження.

Нижче наведено приклад підключення заголовних файлів (для використання периферії і переривань) в AVR Studio і ImageCraft IDE. Приклад для AVR Studio:

```
#include <avr / io.h>            //підключення бібліотеки для використання периферії
#include <avr / interrupt.h>  //підключення бібліотеки для використання переривань

int main (void)
{
  unsigned global_data;              // оголошення змінної
  peripheral_initialization ();      // ініціалізація периферії
  sei ( );                           // дозволяємо всі переривання
  while (1)
    {
     super_code ( );                 // власне програма
     }
 return 0 ;                          // щоб не було warning'ів
 }
```

Рекомендації з програмування мікроконтролерів:

1. Не використовувати багато бібліотечних функцій. Вони роздувають розмір програми. Іноді краще навіть написати вузькоспеціалізований аналог для якоїсь функції.
2. Відмовитися від динамічного виділення пам'яті.
3. Пам'ятати, що математичного співпроцесора немає і дійсний тип float емулюється. Це теж сильно роздуває код.
4. Пам'ятати, що архітектура 8 біт і не захоплюватися int і long.
5. Пам'ятати, що пам'яті не багато і не використовувати багато змінних.
6. Не забивати стек великим числом переданих змінних. Іноді краще оголосити щось глобально. Так само не перестаратися в самих функціях, тому що пам'ять для локальних змінних з функції виділяється теж в стеку.

17.3 Суміщення С та асемблера в одному проекті

При написанні коду для мікроконтролера досить часто зустрічаються окремі місця, де можливості асемблера дійсно необхідні. У таких випадках існують два виходи: перший – асемблерні вставки, другий – написання функції окремо і цілком на асемблері з подальшим викликом її з С.

Асемблерні вставки дозволяють майже повністю забути про питання планування регістрів - компілятор бере це на себе.

Асемблерна вставка – це фрагмент тексту, який напряму передається компілятору мови Assembler, разом зі згенерованим асемблерним текстом в результаті трансляції програми мовою С.

Для оголошення асемблерних вставок в мові С використовується вбудована функція asm () або asm{ } (ще зустрічаються варіації на тему __asm__, але зазвичай ключові слова і функції, що починаються з двох підкреслення маються на увазі для внутрішнього використання). «Вбудована» значить, що для її використання не потрібно підключати ніякі бібліотеки.

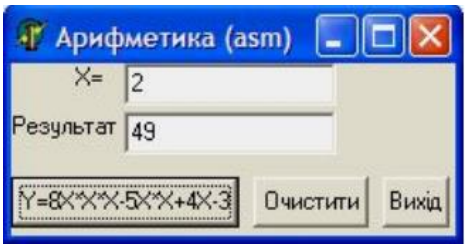
Команди мови асемблер складаються з таких полів:

[ім’я або позначка] операція операнди [; коментар]

Ім’я або позначка може складатися з латинських літер, цифр та спеціальних знаків (., ?, _, @, $), але не може розпочинатися з цифри. Якщо використовується крапка (.), то вона повинна бути першим знаком. Позначка у полі команди відокремлюється від неї двокрапкою (:). Значенням позначки є її адреса. Операція – мнемонічний код операції асемблера. Операнди – один чи кілька операндів, які відокремлюються один від одного комами.

Зазвичай змінні в асемблері оголошуються і набувають початкових значень у сегменті даних. Адреса цього сегмента зберігається в сегментному регістрі DS або ES. У вбудованому асемблері розмірність змінної та тип, тобто кількість байтів займаної нею пам’яті, визначають за директивами DB (Define Byte – визначити байт), DW (Define Word – визначити слово) і DD (Define Double Word – визначити подвійне слово).

Приклад. Обчислити значення виразу $y = 8x^3 - 5x^2 + 4x - 3$, використовуючи програмне забезпечення C++ Builder. Введення даних, підрахунок і виведення результату організуємо у формі:




```
void __fastcall TForm1::Button1Click(TObject *Sender)
{
int y, x = StrToInt(Edit1->Text);
    asm
    {
    mov eax, x
    push eax
    mov ecx, eax
    imul eax
    push eax
    imul ecx shl eax, 3
    mov esi, eax
    pop eax
    mov ecx, 5
    imul ecx
    sub esi, eax
     pop eax
    shl eax, 2
    add eax, esi
    sub eax,3
    mov y, eax
    }
Edit2->Text = IntToStr(y);
}
```

У таблиці подано стан регістрів процесора за покрокового налагоджування

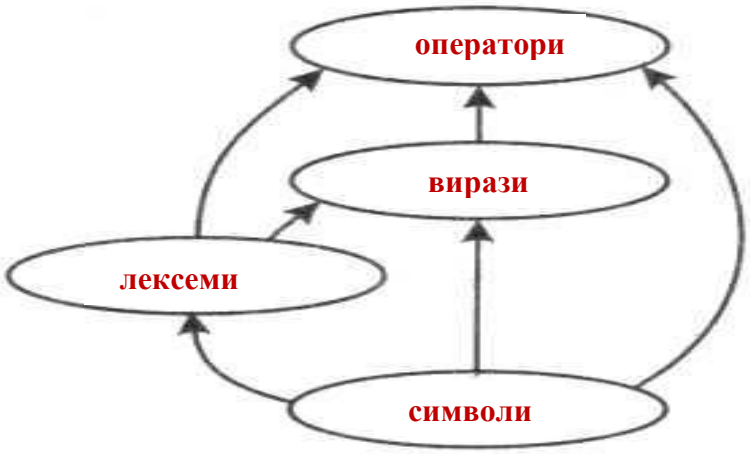
Команди	Стан використовуваних регістрів				Стек
	EAX	EDX	ECX	ISI	
mov eax, x	2				
push eax	2				2
mov ecx, eax	2		2		
imul eax	4	0	2		
push eax	4	0	2		4, 2
imul ecx	8	0			4, 2
shl eax, 3	40 (=64)	0	2		4, 2
mov esi, eax	40	0	2	40	4, 2
pop eax	4	0	2	40	2
mov ecx, 5	4	0	5	40	2
imul ecx	14 (=20)	0	5	40	2
sub esi, eax	14	0	5	2C (=44)	2
pop eax	2	0	5	2C	
shl eax, 2	8	0	5	2C	
add eax, esi	34 (=52)	0	5	2C	
sub eax,3	31 (=49)	0	5	2C	
mov y, eax	31	0	5	2C	

РОЗДІЛ ІІІ. ПРОГРАМУВАННЯ НА МОВІ С++

ЛЕКЦІЯ 18. ОСНОВНІ ВІДОМОСТІ ПРО МОВУ С++. ОПИС СЕРЕДОВИЩА. ТИПИ ДАНИХ. ВВЕДЕННЯ–ВИВЕДЕННЯ ДАНИХ

18.1 Склад алгоритмічної мови

У тексті на будь-якій мові можна виділити чотири основні елементи: символи, слова, словосполучення і речення. Подібні елементи містить й алгоритмічна мова, тільки слова називають лексемами (елементарними конструкціями), словосполучення – виразами, а речення – операторами. Лексеми утворюються із символів, вирази – з лексем і символів, а оператори – із символів, виразів та лексем:



Склад алгоритмічної мови

– алфавіт мови, або його символи, — це основні неподільні знаки, за допомогою яких пишуться всі тексти мовою;

– лексема, або елементарна конструкція, — мінімальна одиниця мови, що має самостійний зміст;

– вираз задає правило обчислення деякого значення;

– оператор задає закінчений опис деякої дії.

Для опису складної дії потрібна послідовність операторів. Оператори можуть бути об'єднані в складовий оператор, або блок. Блоком у мові С++ вважається послідовність операторів, яка взята у фігурні дужки { }. Оператори можуть бути *виконуючі* та *не виконуючі*. Виконуючі оператори задають дії над даними; не виконуючі оператори служать для опису даних, тому їх часто називають операторами опису, або просто описами.

Кожний елемент мови визначається синтаксисом і семантикою. Синтаксичні визначення встановлюють правила побудови елементів мови, а семантика визначає їх значення й правила використання.

Для того, щоб виконати програму, необхідно перекласти її мовою, зрозумілою процесору – в машинні коди. Цей процес складається з декількох етапів. Спочатку програма передається *препроцесору,* який виконує *директиви,* що містяться в її тексті (наприклад, включення в текст так званих заголовних файлів – текстових файлів, котрі містять описи елементів, які використовуються в програмі). Потім текст програми надходить на вхід *компілятора,* який виділяє лексеми, а потім на основі граматики мови розпізнає вирази та оператори. При цьому компілятор виявляє синтаксичні помилки й у разі їх відсутності будує *об'єктний модуль. Компонувальник,* або редактор зв'язків, формує *виконуючий модуль* програми, підключаючи до об'єктного модуля інші об'єктні модулі, у тому числі й ті, які містять функції бібліотек. Якщо програма складається з декількох вихідних файлів, вони компілюються окремо та об'єднуються на етапі компонування. Виконуючий модуль має розширення .exe і запускається на виконання звичайним способом.

Алфавіт С++ включає:

– прописні й рядкові латинські букви і знак підкреслення;

– арабські цифри від 0 до 9;
– спеціальні знаки: " {} , [] () + - / % * . \ ‘ : ? < = > ! & # □ ¦ ^ ;
– пробільні символи: пропуск, символи табуляції, символи переходу на новий рядок.

Із символів алфавіту формуються лексеми мови:
– ідентифікатори;
– ключові (зарезервовані) слова;
– знаки операцій;
– константи;
– роздільники (дужки, крапка, кома, пробільні символи).

Межі лексем визначаються іншими лексемами, такими, як роздільники або знаки операцій.

Ідентифікатор — це ім'я програмного об'єкта. В ідентифікаторі можуть використовуватися латинські букви, цифри й знак підкреслення. Прописні та рядкові букви розрізняються, наприклад, sysop, SySoP і SYSOP — три різні імені. Першим символом ідентифікатора може бути буква або знак підкреслення, але не цифра. Пропуски всередині імен не допускаються. Довжина ідентифікатора за стандартом не обмежена, але деякі компілятори і компонувальники накладають на неї обмеження. Ідентифікатор створюється на етапі оголошення змінної, функції, типу тощо, після цього його можна використовувати в подальших операторах програми.

При виборі ідентифікатора необхідно враховувати, що:

− ідентифікатор не повинен збігатися з зарезервованими словами й іменами стандартних об'єктів;

− не рекомендується починати ідентифікатори із символу підкреслення, оскільки вони можуть збігтися з іменами системних функцій або змінних;

− на ідентифікатори, що використовуються для визначення зовнішніх змінних, накладаються обмеження компонувальника.

Ключові слова — це зарезервовані ідентифікатори, що мають спеціальне значення для компілятора. Їх можна використовувати тільки в тому змісті, в якому вони визначені. Список ключових слів C++ наведений у таблиці.

Список ключових слів C++

asm	else	new	this
auto	enum	operator	throw
bool	explicit	private	true
break	export	protected	try
case	extern	public	typedef
catch	false	register	typeid
char	float	reinterpret_cast	typename
class	for	return	union
const	friend	short	unsigned
const_cast	goto	signed	using
continue	if	sizeof	virtual
default	inline	static	void
delete	int	static_cast	volatile
do	long	struct	wchar_t
double	mutable	switch	while
dynamic_cast	namespace	template	

Знак операції — це один або більше символів, які визначають дію над операндами. Всередині знака операції пропуски не допускаються. Один і той самий знак може інтерпретуватися по-різному залежно від контексту. Всі знаки операцій, за винятком [ ], ( ) і ? : є окремими лексемами.

Константами називають незмінні величини. Розрізняються цілі, дійсні, символьні та рядкові константи. Компілятор, виділивши константу як лексему, відносить її до одного з типів за її зовнішнім виглядом.

Формати констант, відповідні кожному типу, подані в таблиці.

Константа	**Формат**	**Приклади**
Ціла	*Десятковий*: послідовність десяткових цифр, що починається не з нуля, якщо це не число нуль *Вісімковий*: нуль, за яким слідують вісімкові цифри (0, 1, 2, 3, 4 ,5, 6, 7) *Шістнадцятковий*: 0x або 0X, за яким слідують шістнадцяткові цифри (0, l, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)	*8, 0, 199226* *01, 020, 07155* *0xA, 0x1 B8, 0X00FF*
Дійсна	*Десятковий*: [цифри].[цифри] *Експоненціальний*: [цифри][.][цифри]{E|e}[+|-][цифри]	*5.7, .001, 35.* *0.2E6, 11e-3, 5E10*
Символьна	Один або декілька символи, взятих в апострофи	*'A', 'ю', '*', 'db', '\0', '\n', '\012', '\x07\x07'*
Рядкова	Послідовність символів, взятих у лапки	*"Тут був Vasia" , "\tЗначение r=\0xF5\n"*

Символ зворотної косої риски використовується для представлення:

− кодів, що не мають графічного зображення (наприклад *\a* — звуковий сигнал *\n* — переведення курсору в початок наступного рядка);

− символів апострофа *(')*, зворотної косої риски *(\ )*, знака питання *(?)* і лапок *(");*

− будь-якого символу за допомогою його шістнадцяткового або вісімкового коду, наприклад *\073 \0xF5*. Числове значення повинне знаходитися в межах від 0 до 255.

Послідовності символів, що починаються із зворотної косої риски, називають *керуючими*, або *escape-послідовностями*. В таблиці наведені їх допустимі значення.

Керуючі послідовності

Зображення	Найменування
\a	Звуковий сигнал
\b	Повернення на крок
\f	Переведення сторінки (формату)
\n	Переведення рядка
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\\	Зворотна коса межа
\'	Апостроф
\"	Лапка
\?	Знак питання
\0ddd	Вісімковий код символу
\0xddd	Шістнадцятковий код символу

Наприклад, якщо всередині рядка необхідно записати лапки, тоді перед ними потрібно поставити косу риску, за якою компілятор відрізняє її від лапок, котрі обмежують рядок:

"видавництво \"Магнолія\""

У кінець кожного рядкового літерала компілятором додається *нульовий символ,* що представляється керуючою послідовністю *'\0'*. Тому довжина рядка завжди на одиницю більша від кількості символів в його записі. Таким чином, порожній рядок *" "* має довжину 1 байт. Слід звернути увагу на різницю між рядком з одного символу, наприклад, *"A",* і символьною константою *'A'*. Порожня символьна константа недопустима.

Коментар або починається з двох символів «пряма коса риска» (//) й закінчується символом переходу на новий рядок або розміщується між символами-дужками /* і */. Всередині коментаря можна використовувати будь-які допустимі на комп'ютері символи, оскільки компілятор коментарі ігнорує.

18.2 Типи даних

Цілий тип даних, призначений для представлення в пам'яті комп'ютера звичайних цілих чисел. Основним і найбільш уживаним цілим типом є тип *int*. Набагато рідше використовують його різновиди: *short* (коротке ціле) і *long* (довге ціле). Також до цілих типів відноситься тип *char* (символьний). Крім того, при необхідності можна використовувати і тип *long long* (довгі-довгі!), який хоча і не визначений стандартом, але підтримується багатьма компіляторами C++. За замовчуванням усі цілі типи знаковими, тобто старший біт в таких числах визначає знак числа: 0 - число додатне, 1 - число від'ємне. Крім знакових чисел на C++ можна використовувати беззнакові. В цьому випадку всі розряди беруть участь у формуванні цілого числа. При описі беззнакових цілих змінних додається слово *unsigned* (без знаку).

Зведена таблиця знакових цілих типів даних:

Тип даних	Розмір, байт	Діапазон значень
char	1	-128 ... 127
short	2	-32768 ... 32767
int	4	-2147483648 ... 2147483647
long	4	-2147483648 ... 2147483647
long long	8	-9223372036854775808 ... 9223372036854775807

Зведена таблиця беззнакових цілих типів даних:

Тип даних	Розмір, байт	Діапазон значень
unsigned char	1	0 ... 255
unsigned short	2	0 ... 65535
unsigned int (можна просто unsigned)	4	0 ... 4294967295
unsigned long	4	0 ... 4294967295
unsigned long long	8	0 ... 18446744073709551615

Запам'ятовувати граничні значення, особливо для 4-х або 8-мибайтових цілих, навряд чи варто, достатньо знати хоча б якогось порядку можуть бути ці значення, наприклад, тип *int* - приблизно $2 \cdot 10^9$.

На практиці рекомендується скрізь використовувати основний цілий тип, тобто *int*. Справа в тому, що дані основного цілого типу практично завжди обробляються швидше, ніж дані інших цілих типів. Короткі типи (*char, short*) підійдуть для зберігання великих масивів чисел з метою економії пам'яті за умови, що значення елементів не виходять за граничні для цих типів. Довгі типи необхідні у ситуації, коли не досить типу *int*.

Символьні типи

У стандарті C++ немає типу даних, який можна було б вважати дійсно символьним. Для представлення символьної інформації є два типи даних, придатних для цієї мети, - це типи *char і wchar_t* хоча обидва ці типу по своїй суті взагалі-то є цілими типами. Наприклад, можна взяти символ 'A' і поділити його на число 2. До речі, а що вийде? Підказка: символ пробілу. Для «нормальних» символьних типів, наприклад, у Паскалі або C#, арифметичні операції для символів заборонені.

Тип *char* використовується для представлення символів у відповідності з системою кодування ASCII (American Standard Code for Information Interchange - Американський стандартний код обміну інформацією). Це семибітний код, його достатньо для кодування 128 різних символів з кодами від 0 до 127. Символи з кодами від 128 до 255 використовуються для кодування національних шрифтів, символів псевдографіки та ін.

Тип *wchar_t* призначений для роботи з набором символів для кодування яких недостатньо 1 байта, наприклад, Unicode. Розмір типу *wchar_t* зазвичай дорівнює 2 байтам. Якщо в програмі необхідно використовувати рядкові константи типу *wchar_t,* то їх записують з префіксом *L* наприклад, *L"Слово".*

Логічний тип

Логічний (булевий) тип позначається словом *bool*. Дані булевого типу можуть приймати тільки два значення: *true* і *false*. Значення *false* зазвичай дорівнює числу 0, значення *true* - числа 1. Під дані булевого типу відводиться 1 байт.

Дійсні типи

Особливістю дійсних чисел є те, що в пам'яті комп'ютера вони практично завжди зберігаються наближено, а при виконанні арифметичних операцій над такими даними накопичується обчислювальна погрішність. Є три дійсних типу даних: *float, double* і *long double*. Основним вважається тип *double*. Так, усі математичні функції за замовчуванням працюють саме з типом *double*. У таблиці нижче наведені основні характеристики дійсних типів:

Тип даних	Розмір, байт	Діапазон абсолютних величин	Точність, кількість десяткових цифр
float	4	від 3.4E-38 до 3.4E+38	7
double	8	від 1.7E-308 до 1 .7E+308	15

Тип *long double* в даний час, як правило, збігається з типом *double* і на практиці зазвичай не застосовується. При використанні старих 16-ти розрядних компіляторів дані типу *long double* мають розмір 10 байт і забезпечують точність до 19 десяткових цифр.

Рекомендується скрізь використовувати тільки тип *double*. Робота з ним завжди ведеться швидше, менше ймовірність помітної втрати точності при великій кількості обчислень. Тип *float* може знадобитися тільки для зберігання великих масивів за умови, що для вирішення поставленого завдання буде достатньо цього типу.

Тип void

Тип *void* – самий незвичайний тип даних мови C++. Множина значень цього типу порожньо, тобто не можна змінної такого типу привласнити яке-небудь значення. Більш того, не можна навіть описати змінну цього типу.

Він використовується:

– для визначення функцій, які не повертають результату своєї роботи;
– для вказівки того, що список параметрів функції порожній;

Цей тип також є базовим для роботи з покажчиками. Досить сказати, що все програмування з використанням Win32 API побудовано на застосуванні покажчиків на тип *void.*

Змінна – це іменована область пам'яті, в якій зберігаються дані певного типу. У змінної є ім'я і значення. Ім'я служить для звернення до області пам'яті, в якій зберігається значення. Під час виконання програми значення змінної можна змінювати. Перед використанням будь-яка змінна повинна бути описана.

Приклад опису цілої змінної з ім'ям a і дійсної змінної *x*:

int a; float x;

Загальний вигляд *оператора опису* змінних:

[клас пам'яті] [const] тип ім'я [ініціалізатор];

Необов'язковий клас пам'яті може приймати одне із значень *auto, extern, static* та *register*.

Модифікатор const показує, що значення змінної змінювати не можна. Таку змінну називають *іменованою константою*, або просто *константою*. *Константа* повинна бути ініціалізованою при оголошенні.

При описі змінної можна присвоїти їй початкове значення змінної. Це називається *ініціалізацією*. Ініціалізатор можна записувати у двох формах – зі знаком рівності: “= значення” або в круглих дужках: (значення).

В одному операторі можна описати кілька змінних одного типу, розділяючи їх комами.

Приклади:
short int a = 1; // ціла змінна a
const char 3 = 'C' ; // символьна константа 3

18.3 Команда введення даних

У С++ немає вбудованих команд уведення-виведення даних. Для організації введення-виведення тут реалізована концепція *потоків*, яка визначена в спеціальних модулях. У модулі *istream.h* описані команди введення, у модулі *ostream.h* – команди виведення, а у модулі *iostream.h* – команди введення і виведення.

Під потоком розуміють процес введення-виведення інформації в файл. Периферійні пристрої введення-виведення, такі, як клавіатура, монітор, принтер, розглядаються як текстові файли. Під час виконання будь-якої програми автоматично підключаються стандартні потоки для введення даних з клавіатури (*cin*), виведення на екран (*cout*), виведення повідомлення про помилки (*cerr*) і допоміжний потік (*clog*).

Стандартні потоки використовують команди введення (>>) та виведення (<<) даних. За замовчуванням стандартним пристроєм для потоків виведення даних і повідомлень про помилки є монітор користувача, а для потоку введення даних – клавіатура. Однак потоки можна перенаправляти, наприклад, можна зчитувати вхідну інформацію для програми не з клавіатури, а з деякого файлу на диску.

Надавати значення змінним можна двома способами: за допомогою команди присвоєння або команди введення даних із клавіатури. Остання дає змогу виконувати програму для різних вхідних даних.

Це робить її більш універсальною. Команда >> описана у бібліотеці *iostream.h* і має вигляд:

```
cin >> <змінна>;
```

Дія команди. Виконання програми зупиняється. Система переходить в режим очікування введення даного (екран темний, миготить курсор). Користувач набирає на клавіатурі значення змінної і натискає на клавішу вводу. В результаті виконання цієї команди змінній буде присвоєне конкретне значення (те, яке користувач введе з клавіатури).

Якщо необхідно ввести значення відразу для декількох змінних, то можна або використати декілька потоків введення, або записати всі змінні в одному потоці *cin*, застосувавши для цього декілька команд >>, а саме:

```
cin >> <змінна 1> >> <змінна 2> >>…>><змінна N>;
```

Приклад. Уведення значень змінних *a, b, c* можна здійснити так:

```
cin >> a;
cin >> b;
cin >> c;
```

або так:

```
cin >> a >> b >> c;
```

В першому випадку після введення кожного значення слід натискати клавішу вводу, а в другому – введення значень можна здійснювати в одному рядку через пробіл. Якщо у списку введення (який набрали на клавіатурі) даних більше, ніж змінних, то зайві дані будуть зчитані наступною командою введення. Якщо така команда відсутня в програмі, то дані будуть проігноровані.

Програми необхідно складати так, щоб ними могли користуватись не лише укладачі (розробники), але й інші особи. Тобто програми мають бути масовими і зрозумілими. Перед командою введення даних варто записувати команду виведення на екран текстового повідомлення-підказки про те, що саме слід ввести.

Команда виведення даних

Для виведення на екран повідомлень і результатів обчислень використовують стандартний потік виведення даних *cout* і команду <<, які визначені в бібліотеці *iostream.h*:

```
cout << <Вираз 1> << <Вираз 2> <<…<< <Вираз N>;
```

У списку виведення можуть бути сталі, змінні або вирази. Елементи списку в потоці *cout*

відокремлюють командами <<.

Дія команди. Сталі, значення змінних та виразів виводяться на екран у вікно виведення.

Текстові повідомлення у команді виведення записують у лапках. Лапки на екран виводитися не будуть.

Приклад. Якщо в програмі обчислено значення площі *s=6* та периметру *p=12* трикутника, то ці результати можна вивести на екран таким чином:

```
cout <<"Периметр="<<p;
cout <<"Площа="<<s;
```

На екрані з'явиться рядок:

Периметр=12Площа=6

Рядок буде суцільним (без пробілів). Для того, щоб дані виводилися в потрібному для користувача форматі, використовують керуючі послідовності.

Керуючі послідовності

Керуючі послідовності – це комбінації спеціальних символів, які використовуються для введення та виведення даних. Керуюча послідовність складається із символу слеш "\" і спеціально означеного символу. Вони призначені для форматованого виведення результатів обчислень на екран, наприклад, для переходу на новий рядок, подання звукового сигналу, а також для виведення на екран деяких спеціальних символів: апострофа, лапок тощо. Основні керуючі послідовності наведені в таблиці.

Керуючі послідовності

Символи керуючих послідовностей	Коментар
\a,\7	Подати звуковий сигнал
\b	Знищити попередній символ
\f	Перейти на нову сторінку
\n	Перейти на новий рядок
\r	Повернути курсор на початок рядку
\t	Перевести курсор до наступної позиції табуляції
\v	Вертикальна табуляція
\\	Вивести символ похилої риски
\'	Вивести символ апострофу
\"	Вивести символ лапок
\?	Вивести знак запитання

Керуючі послідовності разом з коментарями записують у лапках. Приклади:

```
cout << "Увага!\a\n"; cout << "Дане \"a\" введене некоректно\n";
cout << "Виконайте команду ще раз";
```

При виконанні цих команд буде подано звуковий сигнал і на екрані з'являться такі рядки:

```
Увага!
Дане "a" введене некоректно
Виконайте команду ще раз
```

Якщо використати послідовність, невизначену в мові С++ (наприклад, *\u*), то компілятор пропустить символ послідовності (похилу риску) і виведе на екран лише символ, записаний після риски (в даному випадку літеру *u*). Замість керуючої послідовності *\n* можна використовувати команду *endl* – кінець рядку. Наприклад, наступні дві команди рівносильні:

```
cout << "f = " <<f << "\n";
cout << "f = " <<f<< endl;
```

Функція *clrscr()* в IDE C++Builder знищує з екрану всі попередні результати. Після її виконання екран монітору стає чистим. Для використання її в програмі слід приєднати файл заголовку *conio.h*. В IDE Visual Studio ця функція не працює.

В бібліотеці *conio.h*. також міститься й функція *getch()* (в IDE C++Builder) *_getch()* (в IDE Visual Studio) для затримки на екрані результатів виконання програми, доки не буде натиснута довільна клавіша на клавіатурі.

ЛЕКЦІЯ 19. СКЛАДАННЯ НАЙПРОСТІШИХ ПРОГРАМ НА МОВІ ПРОГРАМУВАННЯ С++

19.1 Структура програми

Процесор – це програма, яка опрацьовує директиви. *Директиви процесора* – це команди компілятора відповідної мови програмування, які виконуються на початку компіляції програми. Директиви мови С++ починаються з символу #. Найчастіше використовуються наступні директиви.

Директива *#include* означає, що до програми необхідно приєднати програмний код із зазначеного після неї файлу. Файли, які приєднуються директивою *#include*, називаються *файлами заголовків* (*header*-файлами, бібліотеками, модулями). У таких файлах зазвичай оголошують сталі, змінні, заголовки (сигнатури) функцій тощо.

Усі стандартні команди та функції мови С++ визначені у файлах заголовків. Щоб приєднати модуль до програми користувача, директиву процесора необхідно зазначити на початку програми так:

#include <назва файлу.розширення>

або так:

#include "шлях до файлу\ назва файлу.розширення"

Зазвичай усі стандартні бібліотеки розміщені у папці INCLUDE середовища С++. У такому випадку назва файлу є параметром директиви, її зазначають в кутових дужках <назва>, наприклад:

#include <math.h>.

Якщо ж потрібний файл розміщений не у папці INCLUDE, то назву файлу із зазначенням шляху пишуть у лапках "...". Наприклад, якщо деякий файл *MyBib.h* є у папці stud на диску *d:*, то треба написати так:

#include "d:\stud\MyBib.h".

Директива *#define* має подвійне значення. По-перше, вона може задати стале значення (оголошує сталу). Наприклад, якщо в програмі задано #define N 25, то N під час виконання програми буде мати значення 25. По-друге, вона дає змогу описати макроси – короткі команди (перевизначити команди) чи записати функції, наприклад, так:

#define D(a,b,c) ((b)(b)-4*(a)*(c)).*

Тепер скрізь для обчислення дискримінанта замість команди *d=b*b-4*a*c* можна записувати *d= D(a,b,c).* Директива *#undef* скасовує дію директиви *#define*. Наприклад,

```
#define D(a,b,c) ((b)*(b)-4*(a)*(c))
#undef D
#define D(a,b,c) ((a)*(b)*(c))
```

Решта директив можна знайти в довідниках по С++.

Головна функція програми С++

Суттєвою особливістю мови С++ порівняно з іншими мовами є те, що програми складаються з функцій, які відіграють роль підпрограм в інших мовах. *Головна функція*, яка має бути в кожній програмі, - це функція вигляду

```
main(void)
{
тіло функції з командою return 0;
}
```

де *main()* – заголовок функції. Ключове слово *void* означає, що функція не залежить від параметрів, його записувати не обов'язково. Функції з параметрами розглядатимуться пізніше. В тілі програми містяться команди та викликаються інших функцій. Команди одна від одної відокремлюють символом "**;**" (крапка з комою). Текст функції закінчується командою *return*. Тіло функції (усі команди після заголовка) записуються у фігурних дужках.

В якості прикладу далі наведено просту програму, результатом виконання якої буде виведення на екран повідомлення "Привіт! Я С++".

```
//Програма 1
#include <iostream.h>
int main()
   {
    cout << "Привіт! Я C++";
    return 0;
    }
```

У першому рядку є коментар. *Коментар* – це фрагмент тексту програми, який слугує для пояснення призначення програми чи окремих команд і не впливає на їх виконання. Його записують так: //*текст коментарю* або так: /* *текст коментарю* */. У першому випадку коментар повинен розташовуватися або в кінці рядку або бути єдиним у рядку. Другий спосіб більш універсальний: коментар можна записувати будь-де, не розриваючи лексем.

Директива *#include <iostream.h>* під'єднує бібліотечний файл *iostream.h*. Саме в цьому файлі описані функції, які дають змогу виконувати операції введення-виведення даних.

Далі в програмі записана обов'язкова функція *main()*. Ключове слово *int* означає, що ця функція повертатиме в точку виклику результат цілого типу.

Конструкція *cout* << забезпечує виведення на екран монітору повідомлення "Привіт! Я C++".

Команда *return* слугує для виходу з функції *main()*. Числовий параметр після *return* є результатом (значенням) функції (у цій програмі - 0).

Якщо функція має тип *void*, то вона не повертає жодних значень, тому команду return писати не треба.

Загальна структура програми

В загальному випадку програма мовою C++ має такий вигляд:

```
//Коментарі
#include <назва бібліотечного файлу 1>
...
#include < назва бібліотечного файлу N>
<інші директиви процесора>
...
<оголошення глобальних змінних>;
< оголошення глобальних сталих>;
< оголошення та створення функцій користувача>;
...
<тип результату функції> main(<опис формальних параметрів>)
 {
 < оголошення локальних змінних >;
 < оголошення локальних сталих>;
 <команди>;
 }
```

Слід звернути увагу на те, що в кутових дужках < > записані значення параметрів директиви, які в програмі пропускати не можна. Крім того, в кутових дужках словами описані загальні конструкції мови, замість яких у реальній програмі будуть конкретні команди. В цьому разі кутові дужки не пишуться.

Розрізняють глобальні та локальні дані. Дані, визначені для всіх функцій C++-програми, називаються *глобальними*, а дані, які використовуються лише в окремих функціях або блоках, - *локальними*.

19.2 Операції інкременту та декременту

Операції інкременту (++) та декременту (--) є унарними операціями присвоювання. Вони відповідно збільшують або зменшують значення операнда на одиницю. Операнд може бути цілого або плаваючого типу або типу покажчика і повинен бути модифікується. Операнд цілого або

плаваючого типу збільшуються (зменшуються) на одиницю. Тип результату відповідає типу операнда. Операнд адресного типу збільшується або зменшується на розмір об'єкта, який він адресує. У мові допускається префіксна або постфіксна форми операцій збільшення (зменшення), тому значення виразу, що використовує операції збільшення (зменшення) залежить від того, яка з форм зазначених операцій використовується.

Якщо знак операції стоїть перед операндом (префіксна форма запису), то зміна операнда відбувається до його використання в вираженні і результатом операції є збільшене або зменшене значення операнда.

У тому випадку, якщо знак операції варто після операнда (постфіксна форма запису), то операнд спочатку використовується для обчислення виразу, а потім відбувається зміна операнда.

Приклади:

int t = 1, s = 2, z, f;

*z = (t++) * 5;*

Спочатку відбувається множення *t * 5*, а потім збільшення *t*. В результаті вийде *z=5, t=2.*

f = (++s)/3;

Спочатку значення s збільшується, а потім використовується в операції ділення. В результаті отримується *s=3, f=1*.

У разі, якщо операції збільшення та зменшення використовуються як самостійні оператори, префіксна та постфіксна форми запису стають еквівалентними.

z++; / еквівалентно */ z++;*

19.3 Присвоєння

Операція простого присвоєння використовується для заміни значення лівого операнда значення правого операнда. При присвоєнні здійснюється перетворення типу правого операнда до типу лівого операнда за раніше наведеними правилами. Лівий операнд повинен бути модифікований.

Приклад:

int t;

char f;

long z;

t = f + z;

Значення змінної *f* перетвориться до типу *long*, обчислюється *f + z* ,результат перетвориться до типу *int* і потім присвоюється змінній *t*.

Крім простого присвоєння, є ціла група операцій, які об'єднують просте присвоювання з однієї з бінарних операцій. Такі операції називаються *складовими операціями присвоювання* і мають вигляд:

(операнд_1) (бінарна операція) = (операнд_2) .

Складене присвоювання за результатом еквівалентно простому присвоювання:

(операнд_1) = (операнд_1) (бінарне операція) (операнд_2) .

Вираз складеного присвоювання з точки зору реалізації не еквівалентним простому присвоювання, так як в останньому *операнд_1* обчислюється двічі.

Кожна операція складеного присвоювання виконує перетворення, які здійснюються відповідної бінарної операцією. Лівим операндом операцій (+=) (-=) може бути покажчик, у той час як правий операнд повинен бути цілим числом.

Приклади:

double arr[4]={ 2.0, 3.3, 5.2, 7.5 } ;

double b=3.0;

b+=arr[2]; / еквівалентно b=b+arr[2] */*

arr[3]/=b+1; / еквівалентно arr[3]=arr[3]/(b+1) */*

При другому присвоєнні використання складеного присвоювання дає більший виграш у часі, так як лівий операнд є індексним виразом.

19.4 Приклади простих програм

Задача 1 (про прямокутний трикутник). Нехай задано катети прямокутного трикутника a = 3, b = 4. Знайти периметр і площу трикутника.

```
// Програма Трикутник1
#include <iostream.h>
#include <math.h>
void main()
{
int a = 3, b = 4, c, p, s;
c = sqrt(a * a + b * b)
p =a+ b + c;
s =a * b / 2;
cout << "p =" << p<< "\n";
cout << "s =" << s << "\n";
cout <<"Виконав Квакін В."
 }
```

Задача 2. Складемо програму, яка обчислює значення функції $y = \sqrt[5]{x^2 + 7.2} - |x - 5| + sin\frac{\pi x}{3}$ для x=2

```
#include <iostream.h>
#include <math.h>
void main()
{
const float pi = 3.1415926;
float x = 2, y;
y = pow((x*x+7.2),5) - fabs(x-5) + sin(pi*x/3);
cout << "y=" <<y<< "\n";
}
```

Результат виконання програми такий: y= -0.512748.

Задача 3 (про трикутник, заданий координатами вершин). Дано координати трьох вершин трикутника A(1; 1), B(2; 2) та C(-1; 2). Обчислити медіану b та радіус описаного кола *r*.

```
#include <iostream.h>                    // Трикутник2
#include <math.h>                        //Бібліотека математичних функцій
#include <conio.h>
void main()
  {
  clrscr();                              // Очищаємо екран
  float x1, x2, x3, y1, y2, y3;          // Оголошуємо змінні та вводимо дані
  cout << "Уведіть координати точки A";
  cin >> x1 >> y1;
  cout << "Уведіть координати точки B";
  cin >> x2 >> y2;
  cout << "Уведіть координати точки C";
  cin >> x3 >> y3;
  float a, b, c, x, y, mb, p, s, r;      // Оголошуємо додаткові змінні
                                         // Обчислюємо довжини сторін трикутника
  a = sqrt(pow((x3 - x2), 2) + pow((y3 - y2), 2));
  b = sqrt(pow((x1 - x3), 2) + pow((y1 - y3), 2));
  c = sqrt(pow((x1 - x2), 2) + pow((y1 - y2), 2));
  x = (x1 + x3) / 2;                     // Обчислюємо координати
  y = (y1 + y3) / 2;                     // середини сторони b
  mb = sqrt(pow((x - x2), 2) + pow((y - y2), 2));   // Обчислюємо медіану mb
  p = (a + b + c) / 2;                   // Обчислюємо периметр
  s = sqrt(p * (p - a) * (p - b) * (p - c));       // Обчислюємо площу
  r = a * b * c / (4 * s); // Обчислюємо радіус
  cout << "mb = " << mb << "\n";         //Виводимо значення медіани
  cout << "r = " << r << "\n";           //Виводимо радіус
  getch();                               //Затримуємо результати на екрані
  }
```

ЛЕКЦІЯ 20. РОЗГАЛУЖЕННЯ НА МОВІ С++, ЇХ ОПИС ТА ЗАСТОСУВАННЯ

20.1 Вибір із двох альтернатив

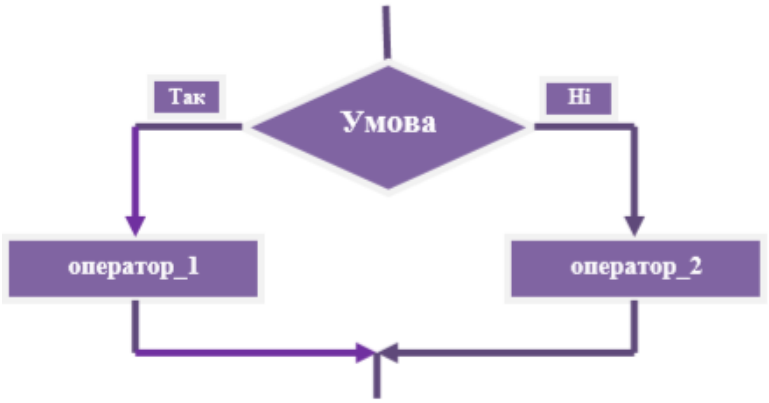
Алгоритмічна конструкція, що дозволяє вибрати ту чи іншу послідовність дій залежно від певних умов, називається *розгалуженням* або *конструкцією вибору альтернатив*.

Конструкція вибору з двох альтернатив дозволяє вибирати один з двох варіантів дій залежно від істинності деякої умови. У мові С++ альтернативні розгалуження реалізуються умовним оператором (*оператором розгалуження*) і *умовним виразом*. Синтаксис умовного оператора є таким:

```
if (<умова>) <оператор_1;> [else <оператор_2;>]
```

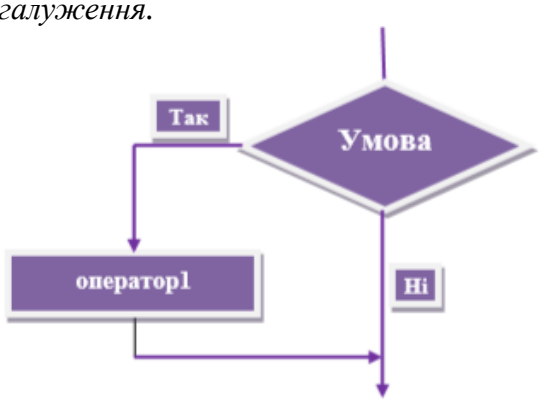
Тут *if, else* – ключові слова, що перекладаються як «*якщо*», «*інакше*», *<умова>* - довільний логічний вираз; *<оператор_1>, <оператор_2>* - довільні оператори.

Виконання умовного оператора починається з обчислення значення булевого виразу *<умова>*. Якщо це вираз є істинним, то виконується *<оператор_1>* і керування передається наступному за умовним оператору, а *<оператор_2>* пропускається. Якщо вираз *<умова>* є хибним, то *<оператор_1>* пропускається, а виконується лише *<оператор_2>* і на цьому дія умовного оператора вважається завершеною.



Слід пам’ятати, що «;» є символом, що завершує оператори, тому запис крапки з комою перед *else* є обов’язковим. Усі умови записуються в дужках.

В синтаксисі умовного оператора *фразу else <оператор_2>* записано в квадратних дужках. Це означає, що ця фраза не є обов’язковою. Скорочена форма умовного оператора (без фрази *else*) реалізує *одноальтернивне розгалуження*.



Якщо вираз *<умова>* в одно альтернативному розгалуженні є істинним, то оператор *<оператор_1>* виконується, якщо хибним – не виконується і на цьому дія умовного оператора вважається завершеною.

Умова (логічний вираз), що записується після слова *if* може бути представлена як окремим порівнянням або окремою булевою змінною, так і поєднанням порівнянь або булевих виразів з допомогою логічних операцій *&&*, *||* та *!*.

Наприклад, умова $0 \le x \le 5$ мовою С++ записується так:

```
x <= 0 && x <= 5
```

а умова $0 \leq x \geq 5$ – так:

```
x <= 0 || x>=5
```

Умови, записані за допомогою логічних операцій &&, || та !, називаються *складеними*, а умови, записані без таких операцій – *простими*.

Приклад. Визначити, чи належить значення дійсної змінної *x* проміжку *[0;1]*, і вивести відповідне повідомлення.

Ці дії виконує такий фрагмент програми:

```
if (x <= 0 && x <= 1)  cout << "x belong to [0;1]";
  else  cout << "x does not belong to [0;1]";
```

20.2 Команда ?

Команда *?* є аналогом команди розгалуження *if*. Загальний вигляд команди *?* такий:

<логічний вираз> ? <команда або вираз 1> : < команда або вираз 2 >

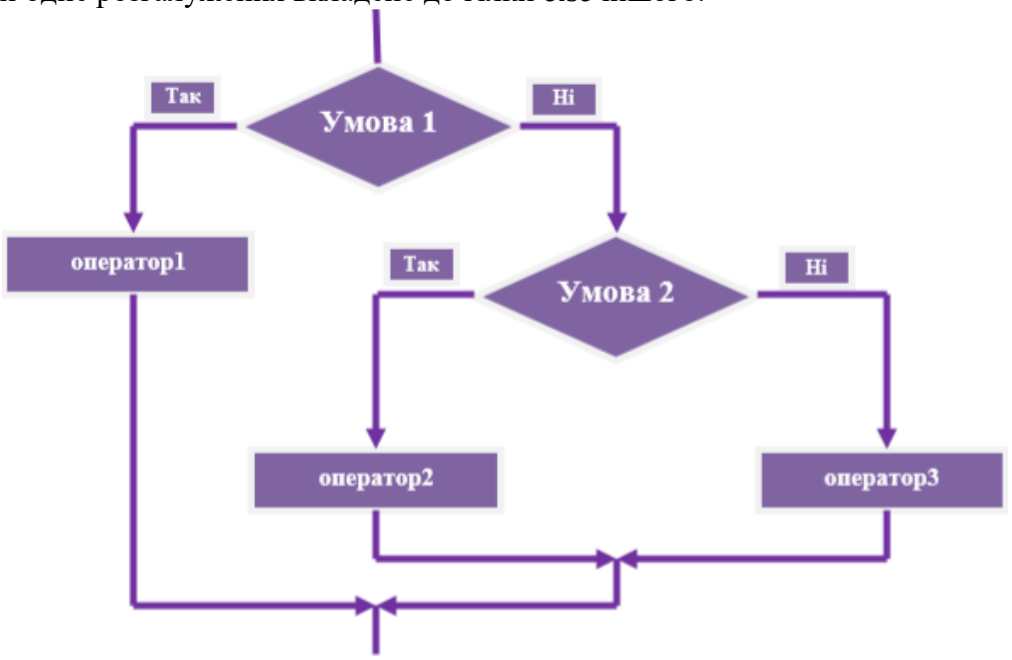
Дія команди: обчислюється значення логічного виразу; якщо воно істинне, то виконується команда 1 або обчислюється вираз 1, інакше - команда або вираз 2.

Приклад. Порівняти значення змінних x та y. Вивести більше з них.

```
x > y ? cout << x  : cout << y;
```

20.3 Вкладеність конструкцій вибору

Гілки розгалуження можуть містити інші розгалуження. Для простоти далі розглядається той випадок, коли одне розгалуження вкладене до гілки *else* іншого.



Синтаксис такої конструкції має вигляд:

```
if (<умова_1>) <оператор_> ;
  else  if (<умова_2>) <оператор_2> ;
        else  <оператор_3>
```

Піраміди вкладених розгалужень завжди можуть бути реалізовані послідовними операторами розгалуження за рахунок ускладнених умов. Цю ж саму конструкцію виконують такі оператори:

```
if (<умова_1>) <оператор_1>;
if (! <умова_1> && <умова_2>) <оператор_2> ;
if (<умова_1> && ! <умова_2>) <оператор_3> ;
```

Але слід мати на увазі, що вкладені умовні конструкції працюють значно швидше, ніж серія умовних операторів у скороченій формі, завдяки тому, що робота всієї конструкції завершується автоматично після виконання однієї з умов. Додаткове прискорення може бути досягнуте за рахунок запису умов, що перевіряються частіше, нагорі піраміди вкладених розгалужень.

Приклад. Обчислити значення функції *y(b,c,x):*

$$y = \begin{cases} bx + c, & x \le -4 \\ \frac{bx}{c}, & x \ge 4, c \ne 0 \\ bx^2, & -4 < x < -4 \\ 1, & x \ge 4, c = 0 \end{cases}$$

Програмний код розв'язання цієї задачі такий:

```
#include <iostream.h>
#include <conio.h>
void main()
{
  float y, b, c, x;
  clrscr();
  cout << "b="; cin >> b;
  cout << "c="; cin >> c;
  cout << "x="; cin >> x;
  if (x <= -4)   y=b*x+c;                          //перша формула
    else if (x > -4 && x <4 )  y = b * pow(x,2);    //третя формула
          else if (c != 0) y = b *x / c;            //друга формула
                else y = 1;                         //четверта формула
  cout << "result:\n";
  cout << "y=" <<y << "\n";
  getch();
}
```

20.4 Операторний блок

Розглянутий синтаксис умовного оператора дозволяє виконати лише один оператор у разі істинності певної умови і один оператор у разі її хибності. Часто виникає потреба у розгалуженнях, гілки яких містять більше одної інструкції. Тоді застосовують операторні блоки.

Операторний блок, або складений оператор, – це послідовність операторів, що оточені фігурними дужками { }. Операторний блок може перебувати в будь-якому місці програми, де синтаксисом мови припускається наявність оператора. Усередині операторного блоку можуть міститися довільні оператори, у тому числі й складені, вони виконуються у порядку запису.

Синтаксис операторного блоку має такий вигляд:

```
{
  <оператор_1>;
  <оператор_2>;
  ……….
  <оператор_n>;
}
```

20.5 Оператор switch

Узагальненням альтернативного розгалуження є алгоритмічна конструкція поліваріатного (мультиальтернативного, множинного) вибору, що дозволяє виконувати одну з декількох алгоритмічних гілок залежно від значення деякого виразу. У мові С++ цю алгоритмічну конструкцію реалізовано *оператором перемикання.* Він має такий синтаксис:

```
switch (<селектор>)
 {
  case <список констант 1>:<оператор 1>; break;
  case <список констант 1>:<оператор 1>; break;
 ……..
  case <ознака N>:<команда N>; break;
  [default: < команда N+1 >; break;]
        }
```

Тут:

– *switch, case, default, break* – це зарезервовані слова, що перекладаються як «*перемикати*», «*випадок*», «*за замовчуванням*», «*перериватиі*»;

– <*селектор*> - змінна або вираз, який має довільний перелічувальний тип;

– <*список констант*> - перелік розділених комами значень того самого типу, що і селектор;

– <*оператор*> - будь-який оператор мови С++;

– дужки *{* та *}* означають початок та кінець тіла оператора *switch*.

Списки констант відокремлюються від операторів двокрапками (**:**).

Оператор перемикання виконується за таким алгоритмом. Спочатку обчислюється значення виразу-селектора. Потім вибирається той список констант, до якого належить отримане значення, виконується відповідний оператор і на цьому дія оператора *switch* завершується. Якщо поточне значення селектора не збігається з жодною з констант перемикання, то виконується гілка *default*, а якщо її немає, то виконання оператора перемикання завершується завдяки наявності оператора *break* у кожній гілці *case*. Відсутність оператора *break* призводить до виконання всіх операторів, починаючи з гілки, позначеної даним списком констант, до кінця оператора *switch*.

Слід пам’ятати, що тип селектора повинен бути перелічуваним. Неприпустимим є використання селекторів дійсних або структурованих типів.

Приклад. Нехай населені пункти позначені номерами від *1* до *8*. Вартість одного квитка до населеного пункту *k* визначається так:

$$cina = \begin{cases} 22, k = 1, \\ 25, k = 2,3,4, \\ 30, k = 5,6, \\ 35, k = 7,8. \end{cases}$$

Скільки коштуватимуть *m* квитків до населеного пункту номер *k*, значення якого вводять з клавіатури?

```
#include <stdafx.h>
#include <iostream>
using namespace std;
void main()
{
//k - номер зони населеного пункт
//m - кількість квитків
//cina - ціна квитка
int k, m; float cina;
cout << "k="; cin >> k;
cout << "m="; cin >> m;
switch ( k )
  {
    case 1: cina=22; break;
    case 2:
    case 3:
    case 4: cina=25; break;
    case 5:
    case 6: cina=30; break;
    case 7:
    case 8: cina=35; break;
    default: cout << "Error\n";
    cina=0;
   }
cout << "ticket prices: " << cina*m <<endl;
```

```
  system("pause");
}
```

Якщо під час виконання програми дані ввести так:

```
3  5,
```

то на екрані з’явиться результат:

```
ticket prices: 125
```

Поліваріантний вибір зручно використовувати в програмах для реалізації меню.

Приклад. Створити калькулятор, що виконує чотири арифметичних дії над дійсними числами. Користувач повинен мати змогу ввести знак операції та значення операндів, а програма має обчислити та вивести результат арифметичної дії.

Для зберігання значень операндів та результату виконання арифметичних операцій слід оголосити три дійсних змінних *(operand1*, *operand2*, *result*), для знаку операції – символьну змінну (*operetion*). Про введення некоректного символу операції та спроби ділення на нуль свідчитиме логічна змінна *flag*.

Програмний код розв’язання цієї задачі такий:

```
#include <iostream>
using namespace std;
void main()
{
  float operand1, operand2, result;
  char operation;
  bool flag;
  cout<<"enter operand 1"; cin>> operand1;
  cout<<"enter operation (+ - * /)"; cin>>operation;
  cout<<"enter operand 1"; cin>> operand2;
  flag=true;
 switch (operation)
  {
   case '+': result = operand1 + operand2; break;
   case '-': result  = operand1 -  operand2; break;
   case '*': result  = operand1 * operand2; break;
   case '/': if (operand2!=0) result  = operand1 / operand2;
             else {
                  cout<<"division by zero\n";
                  flag=false;              //помилка ділення на 0
                  }
              break;
   default:
              flag=false;                          //уведено помилковий символ операції
              cout<<"invalid operation\n";
              break;
  }
  if (flag=true) cout<<"result= "<<result<<"\n";
  else cout<<"result not defined\n";
  system("pause");
}
```

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: АДРЕСА ДАНИХ. ВКАЗІВНИКИ. ДИНАМІЧНА ПАМ'ЯТЬ ДЛЯ МОВИ ПРОГРАМУВАННЯ С++

Визначення адреси даного у пам'яті

Для розв'язування специфічних для мови С++ задач (робота з масивами, побудова графічних зображень і динамічних ефектів) потрібно знати не тільки значення деякої змінної, але й її адресу в оперативній пам' яті. Для визначення адреси даного у пам'яті є операція визначення адреси

&<назва даного>

Приклад 1. Розглянемо фрагмент програми

```
int a= 25;
cout << "Значення змінної a = " << a << "\n";
cout << "Адресою змінної a є " << &a;
```

У результаті виконання цих команд на екрані одержимо:

Значення змінної a = 25

Адресою змінної a є 0xaf72254

Адреси зображаються шістнадцятковими числами і під час кожного виконання програми можуть бути різними.

У мові С++ є ще один засіб визначення адреси даного - це вказівники. Вказівник вказує на початок області оперативної пам'яті комп'ютера, де зберігається дане. Вказівники дають змогу оперувати не з іменами даних, а безпосередньо звертатись до областей пам'яті комп'ютера. Вказівники утворюють так:

<тип даного> *<назва вказівника> ;

Можна створювати вказівники на сталі, змінні, функції, інші вказівники тощо. Особливо ефективною є робота з вказівниками на рядки та масиви.

Наприклад,

```
int *nomer;
float *rist, *prtA, *prtB;
```

Тут оголошено вказівник на цілий тип nomer і вказівники на дійсний тип rist, prtA, prtB. Надати значення вказівникам можна так:

<назва вказівника> = <адреса змінної>;

Приклад 2. Нехай у програмі оголошені змінні

int n = 10; float stud1, stud2, stud3;

Тоді можна визначити вказівники на ці змінні:

```
nomer = &n;
rist = &stud 1;
rist = &stud3;
```

Вказівники застосовують, зокрема, для надання значень змінним:

```
rist = &stud1;
*rist = 1.65;
```

Тут вказівник rist вказуватимемо на адресу змінної stud1, а власне змінній stud1 буде присвоєно значення 1.65.

Приклад 3. Обчислити довжини (в байтах) вказівників на різні типи даних можна так:

```
int *prt_i; double *prt_d;
char *prt_c;
cout << "\nНа цілий тип " « sizeof(prt_i);
cout << "\nНа дійсний тип " « sizeof(prt_d);
cout << "\nНа символьний тип " << sizeof(prt_c);
```

Вказівники .можна переадресовувати. Зокрема, у прикладі 2 вказівнику rist спочатку присвоєно адресу змінної stud1, а потім – змінній stud3. Це правило не діє, якщо вказівник є сталою. Сталий вказівник, що вказуватимемо на одну і ту ж адресу, оголошують так:

```
const int *rik;
```

Для вказівників залежно від операційної системи та версії компілятора резервується 2 або 4 байти в оперативній пам'яті. Над вказівниками визначені арифметичні операції та операції порівняння, описані в таблиці.

Операція	Приклади і пояснення
==, !=, >=, <=, >, <	Порівнює значення двох вказівників (адреси, на які вони вказують). Наприклад, якщо вказівники вказують на одне і те ж саме дане, то результатом порівняння vk1 == vk2 буде істина, інакше – хибність
–	vk1– vk2. Використовується для визначення кількості елементів, які наявні між двома вказівниками
+, –	vk1 + k, vk1 – k. Знаходить вказівник, який зміщений відносно даного на *k* одиниць

Динамічна пам'ять

Команди new і delete. Значення даних у пам'яті комп'ютера можна зберігати в області даних (статична пам'ять), у стеку або в динамічній пам'яті ("на купі"). Усі змінні, які ми розглядали досі, - статичні. Під час їх оголошення система автоматично надає для зберігання їхніх значень певний обсяг оперативної пам'яті, і цей розподіл пам' яті залишається незмінним протягом виконання програми. Пам'ять, надана цим змінним, резервується (фіксується) у середині exe-файлу відкомпільованої програми. Навіть якщо не всі змінні будуть використані у програмі, пам'ять буде зарезервована, тобто використана неефективно. Пам'ять, надана таким змінним, вивільняється лише після виконання програми. Тому в оперативній пам'яті можна розмістити лише обмежену кількість даних.

Однак є задачі, де заздалегідь невідомо, скільки змінних потрібно для їхнього розв'язування, а, отже, який обсяг пам'яті потрібно зарезервувати, або задачі, у яких, навпаки, заздалегідь відомо, що змінних буде багато, наприклад, задачі опрацювання масивів великих розмірів. У таких випадках застосовують динамічну організацію пам'яті.

Принцип динамічної організації пам’яті полягає у тому, що для змінних надається пам'ять за необхідністю (за вказівкою програміста). Далі ці змінні опрацьовують і в потрібний момент пам'ять вивільняють (знову за вказівкою програміста). Такі змінні називаються динамічними.

Для роботи з динамічними змінними використовують вказівники. Для виділення динамічної пам'яті застосовується команда new так:

```
<тип вказівника> *<назва> = new <тип змінної> ;
```

Дія команди *new*. Для відповідного типу змінної автоматично надається необхідна неперервна ділянка пам'яті. Команда new повертає обсяг цієї ділянки, а вказівник вказує на її початок. Наприклад, щоб зарезервувати у пам'яті комп'ютера область для зберігання значення цілого типу, застосовують таку команду:

*int *prt = new int;*.

Надати ділянку пам'яті й відразу занести у неї значення можна так:

*int *prt2 = new float(3.14);*.

У адресу, на яку показує prt2, буде занесено число 3,14.

З динамічною змінною можна виконувати операції, визначені для даних відповідного базового типу. Після опрацювання динамічних змінних пам'ять необхідно вивільнити, а відповідний вказівник занулити. Якщо цього не зробити, то пам' ять можна вичерпати. Вивільняють пам'ять за допомогою команди

```
delete <назва вказівника>;
```

Щоб вказівник не вказував на жодну ділянку пам'яті, його необхідно занулити такою командою:

```
<назва вказівника> = NULL;
```

Значенням (адресою) такого вказівника буде нульова адреса 0x00000000. Тут не може буте розміщене значення жодного даного.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: ПОДІЛ ПРОГРАМИ НА ЕЛЕМЕНТАРНІ ЧАСТИНИ ЗА ДОПОМОГОЮ ФУНКЦІЙ, ПРАКТИЧНЕ ЗАСТОСУВАННЯ ДЛЯ РОЗВ'ЯЗУВАННЯ ЗАДАЧ

Використання функцій

Програми, які досі розглядались, були єдиним, функціонально неподільним кодом. Алгоритм програми перебував у головної функції, причому інших функцій у програмі не було. Ці програми були невеликими, тому не було потреби в оголошенні своїх функцій. Для написання великих програм, досвід показує, що краще користуватися функціями. Програма буде складатися з окремих фрагментів коду, під окремим фрагментом коду розуміється функція. Окремим, тому, що робота окремої функції не залежить від роботи якої-небудь іншої. Тобто алгоритм у кожній функції є функціонально достатнім і не залежним від інших алгоритмів програми. Одного разу написавши функцію, її можна буде з легкістю переносити в інші програми.

Функція – це фрагмент коду або алгоритм, реалізований на якійсь мові програмування, з метою виконання певної послідовності операцій. Отже, функції дозволяють зробити програму модульною, тобто розділити програму на кілька маленьких підпрограм (функцій), які в сукупності виконують поставлену задачу. Ще один величезний плюс функцій полягає в тому, що їх можна багаторазово використовувати. Дана можливість дозволяє багаторазово використовувати один раз написаний код, що в свою чергу, набагато скорочує обсяг коду програми.

Завжди після імені функції ставляться круглі дужки, всередині яких, функції передаються аргументи, і якщо кілька аргументів, то вони відділяються один від одного комами. Аргументи потрібні для того, щоб функції передати інформацію. Наприклад, щоб звести число 3.14 у квадрат використовуючи функцію *pow()* треба якось цієї функції повідомити, яке число, і в яку ступінь його зводити. Ось саме для цього й придумані аргументи функцій, але бувають функції, в яких аргументи не передаються такі функції викликаються з порожніми круглими скобочками. Отже, для того, щоб скористатися функцією із стандартного заголовного файлу C++ необхідно виконати дві дії:

1. Підключити необхідний заголовковий файл;
2. Запустити потрібну функцію.

Крім виклику функцій із стандартних заголовних файлів, в мові програмування C++ передбачена можливість створення власних функцій. В мові програмування C++ є два типи функцій:

1. Функції, які не повертають значень
2. Функції, які повертають значення

Функції, що не повертають значення

Функції, що не повертають значення, завершивши свою роботу, жодної відповіді програмі не дають. Синтаксис оголошення таких функцій має вигляд:

```
void <ім'я функції>(<параметри функції>) // заголовок функції
{
  <тіло функції>
}
```

Перший рядок починається з зарезервованого слова *void* - це тип даних, який не може зберігати будь-які дані. Тип даних *void* говорить про те, що дана функція не повертає ніяких значень. Після зарезервованого слова *void* пишеться ім'я функції. Відразу за ім'ям функції ставляться круглі дужки (перша, що відкривається, а друга - закривається.

Якщо потрібно функції передавати якісь дані, то всередині круглих дужок оголошуються параметри функції, вони відокремлюються один від одного комами. Перший рядок називається *заголовком функції*. Після заголовка функції пишуться дві фігурні дужки, всередині яких знаходиться алгоритм, що називається *тілом функції*.

Приклад. Розробити програму, в якій оголошується функція знаходження факторіала, причому функція не повинна повертати значення.

```
#include <stdafx.h>
#include <iostream>
#include <clocale>
using namespace std;
// оголошення функції знаходження n!
void faktorial(int numb)// заголовок (прототип) функції
{
   int rezult = 1; // ініціалізація змінної rezult значенням 1
   for (int i = 1; i <= numb; i++) // цикл обчислення значення n!
      rezult *= i; // накопичення добутку в змінній rezult
   cout << numb << "! = " << rezult << endl; // відображення значення n!
}
void  main()
{
     setlocale(LC_ALL,"rus");
   int digit; // змінна для збереження значення n!
   cout << "Enter number: ";
   cin >> digit;
   faktorial(digit);// виклик функції знаходження факторіала
 system("pause");
 }
```

Функції, що повертають значення

Функції, що повертають значення, по завершенню своєї роботи надсилають програмі, яка їх викликає, певний результат. Такі функції можуть повертати значення будь-якого типу даних. Структура функцій, які повертають значення буде трохи відрізняться від структури функцій розглянутих раніше. Синтаксис оголошення таких функцій має вигляд.

```
// заголовок функції
<тип даних, що повертаються <ім'я функції>(<параметри функції>)
{
 < тіло функції>
  return <значення, що повертається>;
}
```

Структура оголошення функцій залишилася майже незмінною, за винятком двох рядків. У заголовку функції спочатку потрібно визначати тип даних, що повертається . Це може бути тип даних *int* якщо необхідно повернути ціле число або тип даних *float* - для чисел з плаваючою точкою. Загалом, будь-який інший тип даних, все залежить від того, що функція повинна повернути. Так як функція повинна повернути значення, то для цього має бути передбачений спеціальний механізм. За допомогою зарезервованого слова *return* можна повернути значення, по завершенні роботи функції. Все, що потрібно - це вказати змінну, що містить потрібне значення, або деяке значення, після оператора *return*. Тип даних значення, що повертається повинен збігатися з типом даних, зазначеним перед іменем функції.

Способи оголошення функцій

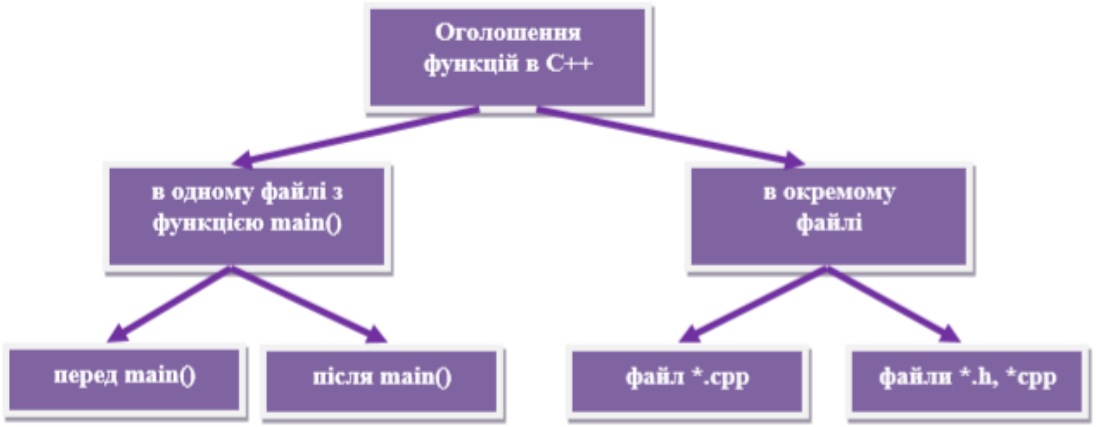
Оголошення розглянутих двох типів функцій виконувались в області програми, після підключення заголовних файлів, але до початку функції *main()*. Існує кілька способів оголошення функцій.

Функції можна оголошувати в двох областях, до початку функції *main()* та після функції *main()*. У розглянутому прикладі функція була оголошена в одному файлі, перед функцією *main()* - це найпростіший із способів.

```
#include <...>
/*область 1 - оголошення функцій до начатку main()
*/головна функція
int main()
```

```
{
 return 0;
}
```

Якщо функції оголошувати в *області 1* перед головною функцією, то прототипи цих функцій не потрібні.



Хорошому стилю програмування відповідає спосіб оголошення функцій після *main()*. В цьому разі структура оголошення функцій така:

```
#include <...>
 // місце для оголошення прототипів функцій
int main()
{
 return 0;
}
 /*область 2 - оголошення функцій після main()
```

Область оголошення функцій перемістилася в самий кінець програми, після *main()*. Що стосується самого способу оголошення функцій , то він не змінився. Але так як функції оголошені після main(), використовувати їх не вдасться, адже порядок оголошень змінився і функція main() просто не буде бачити функції оголошені після. Так от для того, щоб ці функції можна було побачити в *main()* існує поняття прототипу. *Прототип функції* – це заголовок функції, який оголошується перед функцією *main()*. І якщо оголосити прототип функції, тоді функцію можна буде побачити в *main()*. Синтаксис оголошення прототипу функції такий:

```
<тип значення, що повертається функцією> <ім'я функції> (<параметри функції>)
```

Структура оголошення прототипу дуже схожа зі структурою оголошення функції.

Під час оголошення змінних в оперативній пам'яті комп'ютера резервується місце для зберігання їхніх значень. Обсяг наданої пам'яті залежить від типів змінних та компілятора. Кожна змінна характеризується областю дії та областю видимості.

Область видимості ідентифікатора - це область програми, в якій можна посилатися на даний ідентифікатор. На деякі ідентифікатори можна посилатися в будь-якому місці програми, де синтаксисом мови це не заборонено, на інші – тільки в деяких частинах програми. У мові С++ припускається довільна послідовність і кількість блоків, в яких іменуються ті чи інші об'єкти. Існують такі області дії ідентифікатора:

- блок,
- функція,
- файл,
- прототип функції.

Взагалі область дії ідентифікатора поширюється від точки, де він оголошений, до кінця блоку, в якому це оголошення відбулося. Кінцем блоку вважається права фігурна дужка «}».

ЛЕКЦІЯ 21. СТРУКТУРА ОПЕРАТОРІВ ЦИКЛУ

21.1 Загальна структура операторів циклу

Часто постає потреба виконувати один і той самий оператор декілька разів. Для цього використовуються *оператори циклів*, що реалізують алгоритмічну конструкцію повторення. Цикл складається і *заголовка* та *тіла*. У заголовку циклу визначається умова завершення циклу, а тіло циклу являє собою блок операторів, що повторюються. Кожне виконання операторів тіла циклу супроводжується перевіркою умови повторення циклу і називається його *ітерацією*. Якщо умова повторення істинна, то тіло циклу виконується ще раз, якщо хибна, то виконання циклу припиняється і здійснюється перехід до виконання наступного за циклом оператора. Змінні. Значення яких модифікуються в тілі циклу і впливають на істинність умови повторення, називаються *параметрами циклу*. Виконанню будь-якого циклу має передувати ініціалізація його параметрів.

У мові С++ є три різновиди операторів циклу.

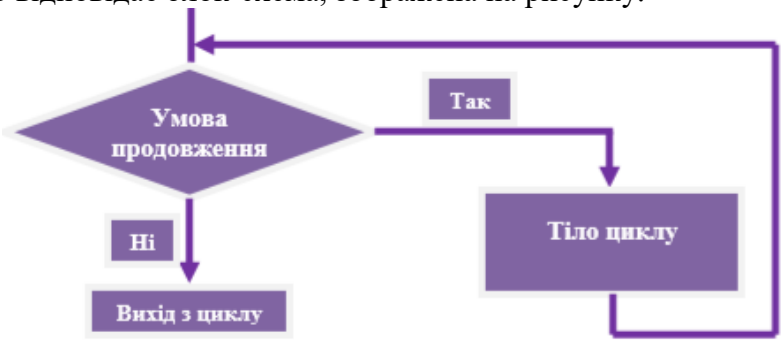
21.2 Цикл з передумовою

У циклі з передумовою перша перевірка умови продовження циклу відбувається ще до першого виконання його тіла. Це означає, що за деяких значень параметрів циклу його тіло може не виконуватися жодного разу. Саме за цією ознакою під час розв'язання певних задач віддають перевагу циклу з передумовою перед циклом із постумовою.

Синтаксис оператора циклу з передумовою такий:

```
while (<умова продовження циклу>)
<оператор>
```

Тут *while (<умова продовження циклу>)* є заголовком циклу, *<оператор>* – його тілом. Тіло циклу може бути операторним блоком і містить в собі будь-які оператори: циклу, вибору, присвоєння тощо. Умова продовження циклу повинна бути виразом булевого типу. Оператору циклу з передумовою відповідає блок-схема, зображена на рисунку.



Оператор циклу з передумовою виконується за таким алгоритмом. Спочатку обчислюється умова продовження циклу, що записана в його заголовку. Якщо вона істинна, то виконується тіло циклу, інакше виконання циклу припиняється. Після виконання тіла циклу буде знову перевірена умова його продовження. Чергування виконання тіла циклу та перевірки умови його продовження триває доти, доки умова не стане хибною. Для запобігання зациклення у тіло циклу слід включити оператор зміни значення його параметра.

Слід пам'ятати, що згідно з синтаксисом оператора *while* тіло циклу є одним оператором. Для того, щоб в циклі виконувалося декілька операторів, їх треба оточити операторними дужками { }.

Приклад. Слід перевірити, чи є задане натуральне число простим.

Як відомо, число *n* є простим, якщо воно ділиться лише на *1* і *n*, інакше *n* вважається складеним числом.

Алгоритм перевірки числа на простоту складається з таких дій, що повторюються:

– перевірка подільності числа *n* на число *k;*
– вибір наступного дільника шляхом збільшення *k* на *1*.

Таким чином, алгоритм є циклічним. Очевидною умовою продовження циклу є умова $k < n$. Оскільки на *1* ділиться будь-яке число, то найменше значення параметра *k* дорівнюватиме *2*.

Перебір усіх *n - 2* чисел від *2* до *n-2* є недоцільним, оскільки значення найбільшого дільника складеного *n* не може перевищувати величини $\sqrt{n}$. В такому разі достатньо перебрати лише цілі числа від *2* до $\lfloor\sqrt{n}\rfloor$ (символ $\lfloor\sqrt{x}\rfloor$ по означає цілу частину числа *x*).

Інформація про те, чи знайдено вже будь-який дільник числа *n*, зберігатиметься в змінні логічного типу *flag*. На початку циклу їй слід присвоїти значення *true*, що означатиме необхідність продовження пошуку. Щойно дільник знайдеться, цій змінній слід призначити значення *false*.

В разі знаходження хоча б одного дільника подальша перевірка чисел не потрібна, оскільки n тоді напевне буде складеним. Таким чином, цикл можна зупиняти не тільки по досягненні дільником *k* значення $\lfloor\sqrt{n}\rfloor$, а й внаслідок хибності значення змінної *flag*. Умова продовження циклу буде складеною:

(k<= (int) sqrt (double(n)) && flag),

де вираз *(int) sqrt (double(n))* дорівнює цілій частині кореня з *n*. Відповідно до синтаксису функції *sqrt()* її параметр і значення, що повертається, мають тип *double*. Використовуючи оператори перетворення типів *(double)* та *(int)* слід передати у функцію *sqrt()* число типу *double* і повернути з неї значення типу *int*. Програмний код розв'язання цієї задачі такий:

```
#include <stdafx.h>
#include <iostream>
#include <math.h>
#include <clocale>
using namespace std;
void main()
{
setlocale(LC_ALL,"rus");
int n, k;  bool flag;
cout<<"number simplicity\n";
cout<<"enter integer: ";  cin>>n;
k=2;
flag=true;
while (k<= (int) sqrt (double(n)) && flag)
  {
    if (n%k == 0) flag=false;
    k++;
  }
 if (flag) cout<<n<<" is simple\n";
 else cout<<n<<" is not simple\n";
  system("pause");
}
```

Оскільки цикл може почати роботу лише в разі істинності умови його продовження, а завершити роботу – лише в разі хибності цієї умови, то її істинність, а отже, і значення параметрів циклу повинні змінюватися під час його роботи. В іншому разі відбудеться «зациклення», тобто виникне ситуація, коли цикл ніколи не завершує своєї роботи.

21.3 Цикл з постумовою

Як і цикл з передумовою, цикл з постумовою застосовують тоді, коли кількість ітерацій циклу є невідомою до початку його виконання. Умова продовження циклу з постумовою записується після тіла циклу та вперше перевіряється після виконання операторів тіла. А отже, цикл з постумовою за будь-яких обставин буде виконано принаймні один раз – в цьому й полягає його головна відмінність від циклу з передумовою.

Синтаксис оператора циклу з постумовою такий:

```
do
{
 <оператор_1;>
……….
```

```
 <оператор_N;>
 }
while (<умова продовження циклу>)
```

Тут *do, while* – зарезервовані слова, *<оператор_1;>… <оператор_N;>* - тіло циклу, *<умова продовження циклу>* - деякий булів вираз. Дослівно ця мовна конструкція перекладається так:

```
виконувати послідовність операторів
доти, доки виконується
```

Оператор циклу з постумовою працює за таким алгоритмом. Спочатку виконуються оператори, що входять до складу тіла циклу. Потім обчислюється умова продовження циклу. Якщо вона хибна, цикл завершує свою роботу, інакше повторюється виконання його тіла. Процеси виконання тіла циклу та перевірки умови повторення чергуються доти, доки умова не стане хибною. Параметри циклу з постумовою, як і циклу з передумовою, повинні змінюватися під час виконання так, щоб не трапилось «за циклювання». Блок-схема циклу з постумовою має вигляд, наведений на рисунку.

Приклад. Скласти програму «вгадування числа». Програма генерує випадкове число з деякого діапазону, а користувач намагається його вгадати, вводячи значення з клавіатури. Якщо число вгадане, то програма виводить повідомлення про це і завершує свою роботу, інакше спроби відгадати число повторюються. На початку виконання програми встановлюється певний «призовий фонд», що зменшується з кожною невдалою спробою користувача.

Очевидно, що процес відгадування є циклічним, оскільки повторюються такі дії, як введення числа, його порівняння із згенерованим і зменшення призового фонду. Для моделювання цієї гри найзручнішою конструкцією буде саме цикл з постумовою, оскільки, по-перше, кількість спроб наперед невідома, і, по-друге, принаймні одна спроба повинна бути здійснена.

Для збереження «призового фонду» використовуватиметься змінна цілого типу *prize*. Число, що генерується комп'ютером, зберігатиметься в цілочисловій змінній y. Для чисел, що вводитиме користувач, призначена змінна цілого типу *x*. Штраф, на який зменшуватиметься «призовий фонд», зберігатиметься в змінній *penalty*.

Для генерування випадкового числа використовуються дві бібліотечні підпрограми. Функція *srand()* ініціалізує датчик випадкових чисел значенням, отриманим від системного таймеру, а функція *rand()* повертає згенероване цим датчиком випадкове ціле число з діапазону *0..32767*. Насправді згенеровані в такий спосіб числа не є випадковими з математичної точки зору, вони лише схожі на випадкові і називаються псевдовипадковими числами. Для генерації різних псевдовипадкових чисел при кожному запуску програми у функцію *srand()* слід передати аргумент, значення котрого також змінюватиметься. Таким значенням, яке змінюватиметься незалежно від користувача, є значення таймеру, яке повертається функцією *time()* з аргументом *(NULL)*. Діапазон генерації псевдовипадкових чисел можна зменшити, застосувавши операцію % визначення остачі від ділення цілих чисел.

Програмний код розв'язання цієї задачі такий:

```
#include <stdafx.h>
#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <math.h>
#include <clocale>
using namespace std;
void main()
{
setlocale(LC_ALL,"rus");
const int penalty=10;
 int x,y,prize;
cout<<"define number, which computer generate\n";
srand((unsigned) time(NULL));      //ініціалізувати генератор
y=rand()%1000;                     //не герувати випадкове число з діапазону 0..999
prize=100;                         //початкова призова сума

do                                 //цикл вгадування числа
{
 cout<<"enter number: "; cin>>x; //введення користувачем числа з клавіатури
 if (x>y)                                  //введене число більше заданого
   {
    cout<<"Wrong, enter smaller\n";
    prize=prize-penalty;
    }
 if (x<y)                                  // введене число менше заданого
   {
    cout<<"Wrong, enter bigger\n";
    prize=prize-penalty;
    }
}
while (x!=y);                              //цикл продовжується доки не буде вгадане задане
число
cout<<"You have guessed right!\n";
cout<<"You have won the prize"<<prize<<"\n";
 system("pause");
}
```

21.4 Синтаксис оператора циклу з лічильником

Поняття циклу з лічильником розглядатиметься на прикладі задачі обчислення факторіалу невід'ємного цілого числа за формулою:

$n!=1\times2\times\ldots\times(n-1)\times n$, де $0!=1$

Очевидно, що $n!=(n-1)!\times n$, тому обчислення можна звести до багаторазового виконання операції *factorial = factorial * i*, де $i=2,3,\ldots,n$, а початкове значення *factorial* дорівнює *1*. Отже, задачу можна розв'язати за допомогою циклічної структури. Тіло циклу складатиметься з однієї операції присвоєння, а кількість її повторень складатиме n. Таким чином, умовою завершення циклу буде виконання певної кількості ітерацій. Відстежити істинність такої умови дозволяє спеціальний різновид параметра циклу, *лічильник ітерацій*. *Лічильник* – це змінна, що під час кожного повторення збільшується або зменшується на певну величину. В операторі *циклу з лічильником* облік кількості виконаних ітерацій вказується в заголовку, і тому цей оператор є зручною формою запису циклів із наперед визначеною кількістю повторень.

Синтаксис оператора циклу з лічильником такий:

```
for ( [вираз1]; [вираз2]; [вираз3])
 <оператор>
```

Тут *for* – зарезервоване слово («*для*»); [*вираз1*] ініціалізує лічильник та виконується один раз на початку циклу; [*вираз2*] – деякий булів вираз, котрий визначає умову повторення циклу; [*вираз3*] змінює значення лічильника циклу. Найчастіше це просто операція інкременту або декременту; <*оператор*> - простий або складений оператор, що є тілом циклу. Фраза *for ( )* є заголовком циклу, а зазначений після круглих дужок оператор – тілом циклу.

Найчастіше [*вираз1*] записують у вигляді <*лічильник*>=<*початкове значення*>; [*вираз2*] є виразом відношення (<, >, <=, >=, !=, ==), в якому порівнюється поточне та кінцеве значення лічильника. Початкове та кінцеве значення лічильника – це вирази одного типу; лічильник – це змінна будь-якого простого типу, значення якої є узгодженим за присвоєнням з типом початкового та кінцевого значення.

Виконання оператора *for* здійснюється за таким алгоритмом. Спочатку обчислюються значення виразів <*початкове значення*> та <*кінцеве значення*>. Один раз на початку циклу лічильнику присвоюється початкове значення, яке порівнюється з його кінцевим значенням. Якщо умова повторення циклу істинна або не дорівнює нулю, то виконується тіло циклу. Після виконання тіла циклу обчислюється значення <*вираз3*>, внаслідок чого змінюється поточне значення лічильника та знову обчислюється значення <*вираз2*> умови повторення циклу. Цикл завершує свою роботу тоді, коли порівняння поточного значення лічильника циклу та його кінцевого значення дає хибний результат або дорівнює *0* (нульове значення в С++ вважається хибним).

Оператор *for* є циклом з передумовою, оскільки рішення про виконання тіла циклу приймається до початку його проходження. Наступний псевдокод аналогічний за своїм результатом оператору *for*:

```
<вираз1>;
while (<вираз2>)
{
 <оператор>;
 <вираз3>
}
```

Синтаксис оператора *for* дозволяє змінювати значення лічильника всередині циклу, але така практика призводить до прихованих логічних помилок. Отже, не слід змінювати значення лічильника всередині тіла циклу *for*.

Запис крапки з комою після заголовку циклу означає, що тіло циклу є порожнім оператором. Зазвичай – це синтаксична помилка.

Вирази в заголовку оператора *for* можна опустити. У такому разі отримується нескінчений цикл:

```
for ( ; ; ) <оператор;>
```

Завершити цикл, в заголовку якого не визначена умова його завершення, можливо внаслідок перевірки умови, що включена до тіла циклу.

21.5 Переривання циклу

У деяких програмах виникає потреба завершити цикл або його ітерацію передчасно. Це можна зробити, використовуючи оператора *break* та *continue*.

Передчасний вихід із циклу означає, що умова продовження циклу під час виходу нього є істинною. Для примусового переривання циклу слід виконати оператор *break*. Оператор continue здійснює пропуск усіх інструкцій, записаних після його виклику в тілі циклу. Таким чином, після виконання оператора continue завжди перевіряється умова продовження циклу.

Якщо використовуються вкладені цикли, то оператор *break* перериває той цикл, у якому він виконується. Після переривання внутрішнього циклу управління передається зовнішньому циклу для перевірки умови його продовження. Аналогічно працює і оператор continue: його виконання забезпечує дії нової ітерації того циклу, в якому цей оператор використовується.

ЛЕКЦІЯ 22. ПОНЯТТЯ МАСИВІВ, ЇХ ОПИС, СТРУКТУРА НА МОВІ С++

22.1 Одновимірні масиви в с++

Характерною ознакою простих типів даних є те, що вони атомарні, тобто не містять як складові елементи дані інших типів. Типи даних, що не задовольняють такій властивості називаються *структурованими*. У мові С++ означено такі структуровані типи:

– масиви,
– рядки,
– структури,
– файли,
– класи.

Структурований тип характеризується множиною елементів, з яких складаються дані цього типу. Елементи файлу називаються компонентами, елементи структур називаються полями, а елементи класів – їх членами. Отже змінна або константа структурованого типу містить декілька елементів.

У свою чергу, кожний елемент може належати до структурованого типу, що дозволяє говорити про вкладеність типів.

З простих елементів даних більш складні структури утворюють двома способами:

– об'єднання однорідних даних метою створення масивів,
– об'єднання різнорідних елементів з метою створення структур.

Визначення, оголошення, властивості

Потреба у масивах виникає тоді, коли в оперативній пам'яті зберігається велика, але визначена кількість однорідних даних. Наприклад, можна задати масив щоденних значень температури повітря протягом місяця з метою визначення середньомісячної температури або масив логічних значень, що зображуватиме наявність квитків на кіносеанс на всі місця у кінозалі.

Розрізняють одновимірні та багатовимірні масиви. Зокрема, наведений в прикладі масив температур є одновимірним, а масив квитків – двовимірним.

Одновимірний масив – це послідовність однотипних даних. Він поєднує в фактично поєднує в собі множину елементів і заданий на цій множині порядок. Усі елементи масиву мають один і той самий тип, що називається *базовим*. З іншого боку, порядок теж визначається набором значень одного й того самого типу, що називається *індексним*, а самі ці значення називаються *індексами.* Кожному елементу масиву відповідає певний індекс. Індексний тип має бути простим порядковим типом даних. Кількість елементів в одновимірному масиві називається його *розмірністю*, або *довжиною*.

Тип масиву – це структурований тип даних, множина допустимих значень котрого складається з усіх масивів, для яких зафіксовано:

– розмірність,
– базовий тип,
– індексний тип,
– множину значень індексу.

Наведене визначення типу масиву можна застосувати до типів як одновимірного, так і двовимірного масивів. Усі елементи одновимірного масиву записуються до розташованих поряд ділянок оперативної пам'яті. Тому і весь масив може розглядатися як одна нерозривна область пам'яті.

Змінну, що матиме тип масиву, можна оголосити з використанням такого синтаксису:

< тип елемента > <ім'я масиву > [<кількість елементів >];

У цьому оголошенні *< тип елемента >* - будь-який тип даних, окрім файлового; *<ім'я масиву >* - деякий ідентифікатор; *[<кількість елементів >]* – константа, що визначає розмір масиву. Межі діапазону допустимих значень індексу визначаються значеннями від 0 до *<кількість елементів > -1*.

За допомогою ключового слова *typedef* можна подати оголошення типу масиву:

```
typedef <базовий тип> <ім'я типу масиву > [<кількість елементів >];
```

Тут *<базовий тип>* - тип елементів масиву; *<ім'я типу масиву >* - деякий ідентифікатор*; [<кількість елементів >]* – розмір масиву.

Змінна, що матиме раніше одзначений тип, оголошується так:

```
<ім'я типу масиву > <ім'я масиву >;
```

Обсяг пам'яті, яка виділена для зберігання всіх оголошених змінних, не повинен перевищувати 64Кбайт. Тому є обмеження на максимальну кількість елементів у масиві. Так, максимальна кількість елементів типу *short* не може перевищувати *32767*, а елементів типу *float* – *16383*.

Приклади оголошень одновимірних масивів.

```
#include<iostream.h>
int main()
{
 const int start=10, finish=30;   //оголошення констант
 typedef int vector[10];          //перейменування простих типів
 typedef chsr STR[256]; //у структуровані
 vector a;                  // оголошення масиву 10 змінних типу int
 STR s;                           // оголошення масиву 256 символів
 int digit[15];                   // оголошення масиву 15 цілих чисел
 float arr[finish-start];         // оголошення масиву 20 змінних типу float
 double temperature[7]; // оголошення масиву 7 змінних типу double
}
```

Основні властивості масивів такі:

– однорідність – усі елементи належать одному типу;

– сталість – вимірність масиву задається під час його оголошення і не змінюється протягом роботи з ним;

– рівно доступність – спосіб доступу до всіх елементів є однаковим;

– послідовність розташування – усі елементи масиву розташовані в послідовних комірках оперативної пам'яті;

– індексованість – елементи однозначно ідентифікуються своїми індексами;

– упорядкованість індексу – індексним тип має бути простим порядковим типом даних.

Типові операції над одновимірними масивами

Будь-яка обробка масивів здійснюється шляхом виконання операцій над їх елементами. Двома найпростішими операціями над елементами одновимірного масиву є вибір певного елемента та зміна його значення. Для доступу до окремого елемента масиву застосовується операція індексування *[ ]*, за допомогою якої утворюють вирази *< ім'я масиву > < індексний вираз >*. Елемент масиву є окремою змінною, що ідентифікується таким виразом.

Приклади ідентифікації елементів масиву:

```
a[1], a[2], ..., a[10];
digit[5], digit[6], ..., digit[20];
```

Слід пам'ятати, що індексація елементів масиву починається з 0. Це означає, що перший елемент завжди матиме індекс 0, а останній – індекс на одиницю менший за кількість елементів.

Значення елементів масиву змінюються так само, як і значення інших змінних.

Наприклад, для першого елемента масиву a це можна зробити операцією присвоєння *a[0] = 5* або операцією введення даних *cin>>a[0]*.

Застосовуючи оператори вибору елемента та модифікації його значення, можна розв'язати досить широкий клас простих задач з обробки одновимірних масивів, які можуть вважатися базовими операціями в контексті складніших задач. Такими базовими операціями є:

– введення та виведення масиву;

– ініціалізація масиву;

– копіювання масиву;

- пошук максимального і мінімального елемента в масиві;
- обчислення узагальнюючих характеристик (сум елементів, середніх значень тощо);
- пошук заданого елемента;
- перестановка елементів або обмін значеннями між елементами масиву;
- вставка та видалення елемента масиву;
- упорядкування масивів.

Базові операції обробки масивів зручно реалізовувати у вигляді функцій, що згодом можуть бути використані як «архітектурні блоки» при розв'язанні більш складних задач.

22.2 Введення та виведення масиву

Мова С++ не має засобів введення та виведення масиву як цілісного об'єкту. Ця операція виконується поелементно за допомогою оператора циклу. Під час введення елементів масиву необхідно враховувати те, що їх кількість, тип та тип індексів задаються в оголошенні масиву на початку виконання програми і не можуть бути змінені. Якщо межі індексів масиву точно не відомі, їх добирають так, щоб введена кількість елементів масиву під час виконання програми не перевищувала верхньої межі індексу. Наприклад, після оголошення масиву *float a[100]* кількість елементів, що до нього вводяться, не повинна перевищувати 100.

Для зберігання значень елементів масиву на етапі компіляції виділяється оперативна пам'ять, обсяг якої дорівнює означеній кількості елементів, помноженій на обсяг пам'яті, що її потребує збереження одного елемента. Під час виконання програми користувач вводить реальну кількість елементів, яка не повинна перевищувати оголошеної кількості. Наведений далі програмний код ілюструє принцип використання операцій введення та виведення масиву. Елементи масиву слід вводити з клавіатури через пробіл.

```
#include "stdafx.h"
#include <iostream>
#include <clocale>
using namespace std;
int main()
{
  setlocale(LC_ALL,"rus");
 int mas[10];              //масив з 10 елементів
 int n;                    //кількість елементів масиву
 int i;                    //поточний індекс (лічильник циклу)
 cout<<"Ввести кількість елементів масиву (<=10): ";  cin>>n;
 cout<<"Input\n";
 cout<<"Ввести значення елементів масиву  <=\n";
 for (i=0; i<n; i++)       cin>>mas[i];          //введення масиву
 cout<<"Output\n";
 for (i=0; i<n; i++)       cout<<mas[i]<<"  ";//виведення масиву
 cout<<"\n";
 system("pause");                                //затримка зображення
}
```

22.3 Ініціалізація масиву

Введення значень елементів із клавіатури є одним із способів ініціалізації масиву. Інший спосіб ініціалізації масиву полягає у присвоєнні кожному її елементу деякого значення. Найбільш ефективно ця операція виконується за допомогою оператора *for*. Наприклад, у далі наведеному програмному коді десяти елементам масиву *arr* присвоюється значення квадратів цілих чисел від 0 до 9.

```
int arr[10]; int n=10;
for (i=0; i<n; i++) arr[i]=(int) pow(i, (float(2));
```

Одновимірні масиви-константи записуються за наведеним далі зразком як перелік значень їх елементів:

```
const int a[5] = (1,3,2,-5,6);
```

Така ініціалізація еквівалентна серії присвоювань

```
a[0]=1;  a[1]=3;  a[2]=2;  a[3]=-5;  a[4]=6;
```

22.4 Поняття багатовимірного масиву

Одновимірні масиви застосовуються для зберігання послідовностей. Однак для багатьох структур даних зображення у вигляді послідовностей є неприйнятним. Наприклад, результати матчів чемпіонату найзручніше подавати у вигляді квадратної таблиці. Для зберігання таких структур даних застосовують багатовимірні масиви, серед яких найбільш широко використовуються двовимірні масиви (матриці).

Оголошення багатовимірних масивів. Доступ до елементів

У математиці матриці представлені у вигляді прямокутної таблиці, що складається з $m \times n$ однотипних елементів, де m – кількість рядків, n – кількість стовпців. У програмуванні матричні структури називають двовимірними масивами.

Синтаксис оголошення змінної матричного типу в мові С++ виглядає так:

```
<тип елементів> <ім‘я матриці > [<кількість рядків>] [<кількість стовпців>];
```

У цьому оголошенні *<тип елементів>* - будь-який тип даних, окрім файлового; *<ім‘я матриці>* - деякий ідентифікатор; *[<кількість рядків>]* і *[<кількість стовпців>]* – константи, що визначають розмірність матриці. Межі діапазону допустимих значень індексів рядків і стовпців визначається значеннями від *0* до *<кількість рядків> -1* та від *0* до *<кількість стовпців> -1*. Масиви, що мають більш ніж два виміри, оголошуються в аналогічний спосіб.

Як і будь-який тип даних користувача, тип двовимірного масиву можна оголосити також за допомогою ключового слова *typedef* окремо від оголошення змінної матричного типу.

Обсяг сегмента пам'яті, де зберігатимуться дані програми, становить 64 Кбайт. Тому під час визначення розміру масиву слід враховувати можливість переповнення пам'яті.

Приклади оголошень багатовимірних масивів.

```
const int k=10;
const int m=20;
const int n=5;
typedef int TMatrix[k][m];
TMatrix a;
float b[n][n];
char c[4][3];
int d[256][2];
int mat[3][5][4];
```

Доступ до елементів матриці здійснюється операцією індексування *[ ]* за двома індексами, які визначають номер рядка та номер стовпця елемента. Синтаксис операції індексування є таким:

```
<ім‘я матриці> [<номер рядка>][<номер стовпця>]
```

Доступ до елементів масивів, які мають більш ніж два виміри, здійснюється також із застосуванням операції індексування. Але її операндами є більш ніж два індекси.

Приклади індексування елементів оголошених масивів.

```
c[3][2]=’C’;
mat[2][4][3]=a[k-1][m-1]+d[0][1];
```

Природне зображення багатовимірного масиву суттєво відрізняється від його зображення в оперативній пам'яті, адже оперативна пам'ять має лінійну структуру. Елементи матриці розташовується в пам'яті комп’ютера за таким принципом. Спочатку у послідовних комірках записуються всі елементи першого рядка, потім у подальших послідовних комірках записуються елементи другого рядка і т.д., поки не буде вичерпано всі елементи. Розмір оперативної пам'яті визначається при оголошенні багатовимірного масиву і не змінюється під час роботи з ним.

22.5 Базові операції обробки двовимірних масивів

До базових операцій над матрицями та їх елементами належать:

– введення та виведення матриць;

– створення нової матриці за заданим алгоритмом;
– пошук елементів матриці за певним критерієм;
– визначення, чи задовольняє матриця або окремі її елементи певним властивостям;
– виконання певних операцій над компонентами матриць (переставлення рядків і стовпців, множення матриць тощо).

Однотипну обробку всіх елементів деякої матриці найлегше здійснити шляхом її обходу за рядками або стовпцями. Тобто спочатку обробляються всі елементи першого рядка (стовпця), потім – всі елементи другого рядка (стовпця) і т.д. Найпростішим методом реалізації такого обходу є використання вкладених циклів *for*.

Введення матриці є достатньо очевидною операцією:

```
int a[5][5];
void main()
{
 for (int i=1; i<5; i++)          //зовнішній цикл по рядках
     for (int j=1; j<5; j++)      //внутрішній цикл по стовпцях
          cin>>a[i][j];           //із клавіатури елементи вводяться через пробіл
}
```

Для того, щоб елементи матриці відображалися у вигляді таблиці, слід переводити курсор на новий рядок після виведення всіх елементів поточного рядка:

```
int a[5][5];
void main()
{ зовнішній цикл по рядках
 for (int i=1; i<5; i++)
  { //внутрішній цикл по стовпцях
    for (int j=1; j<5; j++)        cout<<a[i][j]<<" ";          // виведення елементів рядка
    cout<<"\n";                                                 //переведення курсору на новий рядок
  }
```

Аналогічну структуру має код, що ініціалізує всі елементи матриці деякими значеннями:

```
int a[5][5];
viod main()
{
 for (int i=1; i<5; i++)
     for (int j=1; j<5; j++)
          a[i][j]=rand()%20;
}
```

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: РЯДКИ В МОВІ ПРОГРАМУВАННЯ С++

Поняття рядка та оголошення змінних рядкового типу

Один з різновидів одновимірних масивів – *масив символів*, або *рядок,* - посідає особливе місце у багатьох мовах програмування. І це невипадково, адже алгоритми перетворення рядків застосовуються для вирішення вкрай широкого кола задач: редагування та перекладу текстів, алгебраїчних перетворень формул, крипто аналізу, в інформаційно-пошукових системах, електронних словниках тощо.

Рядок – це скінчена послідовність символів, яку можна розглядати як особливу форму одновимірного масиву. Одна з характеристик масиву – це кількість його елементів, яка є фіксованою величиною і визначається під час оголошення масиву.

Рядок як масив символів теж характеризується довжиною, тобто кількістю символів. Але для рядка розрізняють поняття загальної та поточної довжини. Загальна довжина рядка визначається об'ємом оперативної пам'яті, що була надана рядку під час його оголошення. Поточна довжина рядка визначається кількістю символів у ньому в конкретний момент виконання програми, вона ніколи не перевищує загальної довжини. У разі оголошення рядка як змінної типу символьного масиву його поточна довжина фіксується спеціальним символом. Розташованим після останнього інформаційного символу рядка. Цей спеціальний символ називається *символом кінця рядка* або *нуль-символом* (NULL-'\0'). Поточна довжина рядка визначається автоматично під час його ініціалізації. Зокрема, при введенні рядка з клавіатури його довжина дорівнюватиме кількості символів, введених до натискання клавіші *Enter*. Рядок зберігається одним з трьох способів:

- як рядкова константа або рядковий літерал, що задається в тексті програми;
- як рядкова змінна, що зберігається у масиві символів фіксованої довжини;
- як покажчик на перший символ рядка.

Синтаксис оголошення змінної рядкового типу як масиву символів такий:

```
char < ім'я змінної > [<загальна довжина рядка>];
```

Після такого оголошення створюється масив символів, останнім елементом якого є нуль-символ *' \0 '*. Значення загальної довжини рядка потрібно вказувати обов'язково. Синтаксис оголошення змінної типу char без вказування довжини, тобто кількості символів, стосується тільки символьних змінних.

Оскільки будь-який символ займає один байт в пам'яті, то максимальне значення загальної довжини рядка буде обмежене максимальним значенням, яке можна записати в одному байті. Отже максимальне значення довжини рядка становить 255 символів. Обсяг пам'яті, що виділяється для збереження значення рядкової змінної, буде на один байт більше від вказаної в оголошенні загальної довжини рядка, оскільки для збереження значення його поточної довжини потрібен додатковий бай пам'яті. Оголошений масив символів повинен мати такий розмір, щоб зберегти найдовший рядок, що обробляється та символ кінця рядка *' \0 '*. Якщо рядок більший за масив символів, в якому він зберігається, символи, які виходять за його межі, будуть змінювати дані в комірках пам'яті, розташованих за масивом.

Рядок в С++ доступний через покажчик на його перший символ. Значенням рядка є адреса його першого символу. Отже рядок можна оголосити як покажчик на тип *char:*

```
char *<ім'я змінної>;
```

Таке оголошення створює змінну-покажчик, який вказує на рядок десь в оперативній пам'яті.

Приклади оголошень змінних рядкового типу:

```
const int length=50;
char str[255];              //резервується 256 байт
char name[15];              // резервується 16 байт
char address[length];       // резервується 51 байт
char *pointer;              // резервується 4 байти для покажчика
```

У мові С++ рядкові константи або рядкові літерали записують у подвійних лапках, наприклад, “Visual C++”, “avenue Victory, 37”. Символьні константи записують в одиночних лапках, наприклад *‘+’, ‘A’, ‘\n‘*.

Як і елементи будь-якого масиву, символи рядка зберігаються у поряд розташованих комірках оперативної пам'яті. Наприклад, якщо до оголошеної змінної name ввести з клавіатури рядок символів “Visual C++”, то зображення змінної name в оперативній пам'яті буде таким:

V	i	s	u	a	l		C	+	+	\0	\0	\0	\0	\0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

До елементів рядка з індексами від 10 до 14 можна звертатися: їм можна присвоїти значення будь-якого символу або прочитати їх. Але під час виведення рядка символи, що їх індекси перевищують поточну довжину, не відображатимуться.

Рядкові константи та ініціалізація рядків

Рядкові константи можуть застосовуватися в операторах виведення як повідомлення про виконані дії, наприклад:

```
cout<<”Bubble sort”;
```

Оголошення рядкових констант можна задати в директиві процесора #define, наприклад:

```
#define Title “Insertion sort”;
```

Рядкова константа *Title* має 14 символів, у тому числі пробіл, що є значущим символом. Для використання цієї константи достатньо задати її ім’я в відповідному місці програми. Означене в директиві *#define* значення не можна змінити.

Рядкову константу можна задати через покажчик на її перший символ, використовуючи синтаксис покажчиків. Оскільки ім’я масиву є покажчиком на його перший елемент, то еквівалентними будуть такі оголошення:

```
const char *name1=”Sort by Shell”;
const char  name2[ ]=”Sort by Shell”;
```

Використання ключового слова *const* не дозволяє змінити значення рядкового літералу операторами С++.

Ініціалізація рядкової змінної здійснюється під час її оголошення як масиву символів чи покажчика на перший символ рядка. На відміну від рядкової константи ініціалізована рядкова змінна може змінювати своє значення протягом роботи програми.

Приклад ініціалізації рядкової змінної через покажчик на ї перший символ:

```
char *author=”Deitel H.”;
```

Оголошеній як покажчик на тип char змінній присвоюється значення адреси її першого символу, тобто *‘D’*. Символи рядка зберігаються за фіксованою адресою, яка присвоюється змінній-покажчику author.

Рядок, ініціалізований як масив символів, оголошується без вказування їх кількості, наприклад:

```
char book[ ] = ”C++ how to programming”;
```

Символи з рядка ”C++ how to programming” копіюються в область пам'яті, що її зарезервовано для рядкової змінної. Об’єм пам'яті для рядка book визначається по кількості символів з урахуванням нуль-символу.

Масиви рядків

У мові С++ змінні рядкового типу можна обробляти двома способами. Перший спосіб дає можливість розглядати рядок як цілісний об’єкт, другий – обробляти його як об’єкт, що складається з окремих символів, тобто елементів типу char. Перший спосіб надає можливість за одну дію обробити усі символи рядка. Другий спосіб забезпечує доступ до окремих символів рядка за їх індексами – аналогічно тому, як здійснюється доступ до елементів одновимірного масиву. Для доступу до символу рядка застосовують операцію індексування:

```
<ім‘я змінної рядкового типу> [<індекс символу >];
```

Аналогічно цифровим двовимірним масивам можна побудувати й масив рядків. Текст, який включає більше, ніж один рядок, є *масив рядків*.

Очевидно, що масив рядків є двовимірним масивом, перший розмір якого визначає кількість рядків, а другий – максимальну кількість символів у рядку.

Приклад. Розглядається масив рядків, що містить в кожному з них назви днів тижня *“Monday”, “Tuesday”, “Wendesday”, “Thursday”, “Friday”, “Saturday”, “Sunday”*. Його можна оголосити так:

```
char Week[7][10];
```

Тут значення 10 відповідає найдовшій довжині днів “*Wednesday*” з урахуванням символу кінця рядку. Для доступу до елементів масиву, яким є рядок, використовується лише один його індекс:

```
<ім‘я масиву рядків> [<індекс рядка>];
```

Доступ до кожного символу з масиву рядків здійснюється за двома індексами:

```
<ім‘я масиву рядків> [<індекс рядка>][індекс символу];
```

У наведеному прикладі отримати значення “*Wednesday*” з масиву *Week* можна як *Week[2]*. Символ *‘s’* того самого елемента масиву *Week* можна отримати через звернення *Week[2][5]*.

Уведення та виведення рядків

Для введення та виведення рядків можна застосувати операції << - «взяти з потоку» та >> - «помітити в потік». Наприклад:

```
char word[20];
cin>>word;
```

Другий оператор зчитує символи доти, доки не зустрінеться символ пробілу, табуляції, нового рядка або покажчика кінця файлу. Отже, рядок, що вводиться операцією >>, складається з одного слова, припустимі розділові символи крапка, крапка з комою, кома та інші, крім раніше перелічуваних, які припиняють введення рядка. Символи рядка, що введені після символу пробілу, ігноруються. Слід також пам’ятати, що в наведеному прикладі рядок не повинен містити більше 19 символів, щоб залишити двадцятий символ для збереження нуль-символу.

Рядок можна вивести на екран оператором:

```
cout<<word;
```

У разі ініціалізації рядка константним значенням виводитимуться усі символи, у тому числі й пробіли. В іншому разі будуть зображені тільки символи, що введені до першого пробілу. Очевидно, що рядки, які не мають символів пробілу, є поодиноким випадком. Функції стандартної бібліотеки введення-виведення у мові С++ дозволяють вводити рядки з будь-якими символами.

Уведення та виведення рядка здійснюється за допомогою функцій *gets(char *s), puts(char *s)*. Параметром функцій уведення та виведення є покажчик на рядок. Під час уведення та виведення цикли перебирання символів рядка не потрібні. Натискання клавіші *Enter* в процесі введення формує символ кінця рядку. Функція *puts(char *s)* автоматично дописує символ нового рядку, що забезпечує переведення курсору на наступний рядок. Для використання цих функцій уведення та виведення рядків слід підключити заголовний файл *stdio.h*.

ЛЕКЦІЯ 23. ГОЛОВНІ ПРИЙОМИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ. ЕЛЕМЕНТИ ПРОГРАМИ BORLAND C++

23.1 Представлення програми у вигляді сукупності *об'єктів*

Об'єктно-орієнтоване програмування або *ООП (object - oriented programming)* - методологія програмування, заснована на представленні програми у вигляді сукупності *об'єктів*, кожен з яких є реалізацією визначеного типу, що використовує механізм пересилки повідомлень, і *класів*, організованих в ієрархію успадкування.

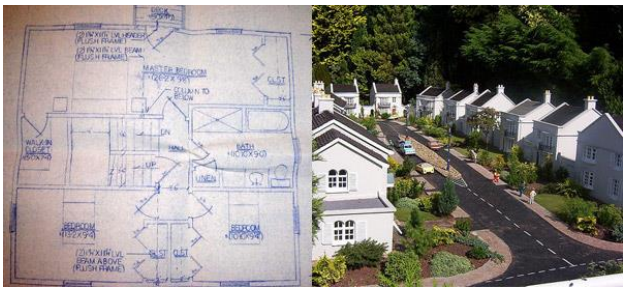
Центральний елемент ООП – *абстракція*. Дані за допомогою абстракції перетворюються в об'єкти, а послідовність обробки цих даних перетворюється на набір повідомлень, що передаються між цими об'єктами. Кожен з об'єктів має свою власну унікальну поведінку. До об'єктів можна звертатися як до окремих сутностей, які реагують на повідомлення, що наказують їм виконати якісь дії.

Абстрагування (abstraction) – метод розв'язання задачі, при якому об'єкти різного типу об'єднуються загальним поняттям (*концепцією*), а потім згруповані сутності розглядаються як елементи єдиної категорії. Абстрагування дозволяє відокремити логічний сенс фрагмента програми від проблеми його реалізації, розділивши зовнішній опис (*інтерфейс*) об'єкту і його внутрішню організацію (*реалізацію*).

Основними концепціями об'єктно-орієнтоване програмування є *інкапсуляція, успадкування* та *поліморфізм*, а базовими поняттями – *клас* та *об'єкт*.

Приклад.

Клас – це проект будинку. Він визначає на папері, як виглядатиме будинок, чітко описує всі взаємозв'язки між його різними часинами, навіть якщо будинку ще не існує в реальності.
А об'єкт – це реальний будинок, побудований у відповідності з проектом. Дані, що зберігаються в об'єкті, схожі на дерево, бетон, цеглу, з яких побудований будинок: без збірки у відповідності з проектом вони будуть лише купою матеріалів. Однак, зібрані разом вони стають зручним будинком.
Класи формують структуру даних і дій. Вони використовують цю інформацію для побудови об'єктів. З одного класу може бути побудовано багато об'єктів. В той самий час ці об'єкти будуть незалежними один від одного. Продовжуючи аналогію з будівництвом, цілий район може бути побудований по одному проекту: деяка кількість будинків, які зовні виглядають однаково, але в кожному з них живуть різні люди та внутрішнє оздоблення будівель різне.



Класом в ООП називається опис деякої структури програми, яка володіє набором внутрішніх змінних – *властивостей (атрибутів)*, і функцій, що мають доступ до властивостей – *методів*. Властивості та методи класу називають ще *членами класу*. Клас – всього лише опис, аналогічний опису типу даних, і не доступний для прямого використання в програмі.

Для отримання доступу до властивостей та методів класу необхідно створити *екземпляр класу*, який інакше називається *об'єктом*. В цьому є деяка подібність між класами та записами. Однак класи є набагато потужнішим інструментом програмування в силу притаманних їм, на відміну від записів, механізмів успадкування та властивостей поліморфізму.

Інкапсуляція (*encapsulation*) - це механізм, який об'єднує дані і код, який маніпулює цими даними, а також захищає дані і код від зовнішнього втручання або неправильного використання. В об'єктно-орієнтованому програмуванні код і дані можуть бути об'єднані разом. В цьому випадку говорять, що створюється так звана *чорна скриня*.

Усередині класу коди і дані можуть бути закритими (*private*). Закриті коди або дані

доступні тільки для інших частин цього класу. Таким чином, закриті коди і дані недоступні для тих частин програми, які існують поза класом. Якщо коди і дані є відкритими (*public*), то, незважаючи на те, що вони задані всередині класу, вони доступні і для інших частин програми. Характерною є ситуація, коли відкрита частина класу використовується для того, щоб забезпечити контрольований інтерфейс його закритих елементів.

Поліморфізм (*polymorphism*) - це властивість, яка дозволяє одне і те ж ім'я використовувати для вирішення двох або більше схожих, але технічно різних завдань. Метою поліморфізму, стосовно об'єктно-орієнтованого програмування, є використання одного імені для завдання загальних для класу дій. Виконання кожної конкретної дії визначатиметься типом даних. Тип даних, який використовується при виклику функції, визначає, яка конкретна версія функції дійсно виконується. Можна використовувати одне ім'я функції для множини різних дій. Це називається *перевантаженням функцій* (*function overloading*).

У більш загальному сенсі, концепцією поліморфізму є ідея: *один інтерфейс, множина методів*. Це означає, що можна створити загальний інтерфейс для групи близьких за змістом дій. Перевагою поліморфізму є те, що він допомагає знижувати складність програм, використовуючи один і той самий інтерфейс для визначення єдиного класу дій. Вибір конкретної дії, залежно від ситуації, покладається на компілятор. Програмісту не потрібно робити цей вибір самому. Потрібно тільки пам'ятати і використовувати загальний інтерфейс.

Ключовим у розумінні поліморфізму є те, що він дозволяє маніпулювати об'єктами різного ступеня складності шляхом створення спільного для них стандартного інтерфейсу для реалізації схожих дій.

Успадкування (*inheritance*) - це процес, за допомогою якого один клас може набувати властивостей іншого. Точніше, клас може успадковувати основні властивості іншого класу і додавати до них риси, характерні тільки для нього. Успадкування є важливим, оскільки воно дозволяє підтримувати концепцію ієрархії класів (*hierarchical classification*). Застосування ієрархії класів робить керованими великі потоки інформації.

Як приклад можна розглянути опис житлового будинку. Будинок - це частина загального класу, званого будовою. З іншого боку, будова - це частина більш загального класу - конструкції, який є частиною ще більш загального класу об'єктів, який можна назвати створенням рук людини. У кожному випадку породжений клас успадковує всі, пов'язані з батьківським класом, якості та додає до них свої власні визначальні характеристики. Без використання ієрархії класів, для кожного об'єкта довелося б задати всі характеристики, які б вичерпно його визначали. Однак при використанні успадкування можна описати клас шляхом визначення того загального класу (або класів), до якого він відноситься, з тими спеціальними рисами, які роблять його унікальним.

Множинне успадкування – одна з особливостей мови С++. В мовах, створених пізніше цей механізм не реалізований через велику кількість його недоліків.

Отже, суть множинного успадкування полягає в тому, що клас, який створюється, може отримати *у спадок* набір атрибутів і методів кількох базових класів.

Першим і основним недоліком реалізації такого механізму є виникнення колізій між методами або атрибутами базових класів, які називаються однаково. Для вирішення цією проблеми завжди потрібно вказувати базовий клас методу, який викликається (так би мовити класифікувати методи і атрибути за назвами їх класів).

Другим недоліком можна вважати те, що при побудові великих ланцюжків спадкувань виникають ситуації, коли один клас успадковує властивості іншого кілька разів (перший раз через проміжний клас, а другий – безпосередньо).

Програмне забезпечення стає занадто заплутаним і зв’язаним, що створює ряд проблем при його аналізі і спробах повторно використати чи модифікувати.

23.2 Структура класу. Створення та використання об’єктів

Клас в С++ створюється таким чином:

```
class TMyClass: public ParentClass // ParentClass — клас-предок, якщо клас є дочірнім
 {
  public:
```

{Описані в цій секції елементи доступні всім}

```
  <властивості>
  <заголовки методів>
protected:
```

{Описані в цій секції елементи доступні лише класу та його нащадкам}

```
  <властивості>
  <заголовки методів>
private:
```

{Описані в цій секції елементи не доступні ззовні (за межами класу}

```
  <властивості>
  <заголовки методів>
};
```

Після свого створення клас вважається повноцінним *типом даних*. Екземпляри класу створюються таким чином:

```
TMyClass  MyClass;
```

Звернення до членів класу здійснюється так:

```
MyClass.classmember                    // classmember – ім'я члена класу
```

Знищується екземпляр класу, як і будь-яка змінна, тільки в випадку, якщо функція, в якій він був створений, завершила роботу або якщо була примусово звільнена динамічна пам'ять, виділена під клас.

Програмний код, що створює та використовує клас *TKVD (квадрат)* на мові С++, такий:

```
//Створення класів та об'єктів
#include<iostream.h>
#include<conio.h>
#include<math.h>
//Оголошення класу TKVD (квадрат)
class TKVD
  {
   public:                    //модифікатор доступу - будь-яка частина модуля
   double a;                  //сторона квадрату - змінна
   double S();                //площа квадрату - функція
   double P();                //периметр квадрату - функція
   double D();                //діагональ квадрату - функція
  };
//реалізація методу обчислення площі квадрату
double TKVD::S()
{
 return a*a;
}
//реалізація методу обчислення периметру квадрату
double TKVD::P()
{
 return 4*a;
}
//реалізація методу обчислення діагоналі квадрату
double TKVD::D()
{
 return a*sqrt(2);
}
//головна функція
void main()
```

```
{
 TKVD K1, K2, *K3;   //об'єкти (екземпляри) класу KVD
//1-й екземпляр класу
 cout<<"Objet1:"<<endl;
 cout<<"=======";
 K1.a=1;
 cout<<"\na="<<K1.a;
 cout<<"\nS="<<K1.S();
 cout<<"\nP="<<K1.P();
 cout<<"\nD="<<K1.D();
//2-й екземпляр класу
 cout<<"\nObjet2:"<<endl;
 cout<<"=======";
 K2.a=3;
 cout<<"\na="<<K2.a;
 cout<<"\nS="<<K2.S();
 cout<<"\nP="<<K2.P();
cout<<"\nD="<<K2.D();
//3-й екземпляр класу
 cout<<"\nObjet3:"<<endl;
 cout<<"=======";
 K3->a=10;
 cout<<"\na="<<K3->a;
 cout<<"\nS="<<K3->S();
 cout<<"\nP="<<K3->P();
 cout<<"\nD="<<K3->D();
 getch();
}
```

```
Objet1:
=======
a=1
S=1
P=4
D=1.41421
Objet2:
=======
a=3
S=9
P=12
D=4.24264
Objet3:
=======
a=10
S=100
P=40
D=14.1421
```

Результати роботи програми, що демонструє створення та використання класу (консольний режим IDE C++Builder)

23.3 Переваги об'єктно-орієнтованого програмування

Найважливішим кроком на шляху до вдосконалення мов програмування стала поява об'єктно-орієнтованого підходу до програмування та відповідного класу мов. При об'єктно-орієнтованому підході програма являє собою опис об'єктів, їх властивостей (*атрибутів*), сукупностей (*класів*), відносин між ними, способів їх взаємодії та операцій над об’єктами (*методів*).

Безперечною перевагою даного підходу є концептуальна близькість до предметної області довільної структури та призначення. Механізм успадкування атрибутів і методів дозволяє будувати похідні поняття на основі базових і таким чином створювати модель як завгодно складної предметної області з заданими властивостями.

Ще одною теоретично цікавою і практично важливою властивістю об'єктно-орієнтованого підходу є підтримка механізму обробки подій, які змінюють атрибути об'єктів і моделюють їх взаємодію в предметній області. Переміщаючись по ієрархії класів від загальних понять предметної області до більш конкретних (або від більш складних - до більш простих) і навпаки, програміст отримує можливість змінювати ступінь абстрактності або конкретності погляду на модельований їм реальний світ.

Використання раніше розроблених (можливо, іншими колективами програмістів) бібліотек об'єктів і методів дозволяє значно заощадити трудовитрати по розробці програмного забезпечення, особливо, типового.

Об'єкти, класи і методи можуть бути поліморфними, що робить реалізоване програмне забезпечення більш гнучким і універсальним.

Складність адекватної (несуперечливої і повної) формалізації об'єктної теорії породжує труднощі тестування та верифікації створеного програмного забезпечення. Мабуть, ця обставина є одним з найбільш істотних недоліків об'єктно-орієнтованого підходу до програмування.

ЛЕКЦІЯ 24. КОРОТКА ХАРАКТЕРИСТИКА КОМПОНЕНТ BORLAND C++

24.1 Основні поняття

Технологія роботи у середовищі C++ Builder базується на ідеях об'єктно-орієнтованого та візуального програмування. Ідея об'єктно-орієнтованого програмування полягає в інкапсуляції (об'єднанні) даних і засобів їх опрацювання (методів) у тип, який називається класом. Конкретною змінною певного класу і є *об'єкт*. Прикладами об'єктів можуть бути елементи керування у вікні: кнопки, списки, текстові поля тощо. Середовище візуального програмування C++ Builder - це графічна автоматизована оболонка над об'єктно-орієнтованою мовою програмування C++ . Якщо у мові Сі структурними одиницями є дані та команди, то тут такою структурною одиницею є візуальний об'єкт, який називається *компонентом*. Автоматизація програмування досягається завдяки можливості переносити компонент на форму (у програму) з палітри компонентів і змінювати його властивості, не вносячи вручну змін до програмного коду. Формою називають компонент, який володіє властивостями вікна Windows і призначений для розташування інших компонентів. Компоненти на формі можуть бути видимими та невидимими. Видимі призначені для організації діалогу з користувачем. Це різні кнопки, списки, текстові поля, зображення тощо. Вони відображаються на екрані під час виконання програми. Невидимі компоненти призначені для доступу до системних ресурсів комп'ютера. *Проект* - це сукупність файлів, з яких складає'гься C++ Builder-програма.

24.2 Інструменти середовища C++ Builder

Вікно середовища містить головне меню, панелі інструментів (ToolBars), а також:

– палітру компонентів (Component Palette);
– вікно властивостей об'єктів (Object Inspector);
– вікно форми;
– редактор коду програми

Ці інструменти стають доступними після запуску C++ Builder: три знаходяться у головному вікні (верхня частина екрана), а решта - в окремих вікнах . Ці вікна також можна відкрити командами головного меню View ~ TooiBars ~ Component Palette, View ~ Object lnspector, View ~ Forms та View ~ Units. Почергова активізація вікна форми та коду програми здійснюється командою головного меню View ~ Toggle Form/Unit чи клавішею F12.

Головне меню складається з таких елементів: File, Edit, Search, View, Project, Run, Component, Database, Tools, Help. Меню File містить стандартні команди для роботи з файлами проекту. За допомогою цих команд можна створити новий проект (New Application), нову форму (New Form), відкрити чи закрити файл проекту (Open і Close), закрити всі відкриті файли (Ciose All), зберегти файл, проект або все відразу (Save, Save As, Save Project As, Save All)

За допомогою команд меню Edit можна вирівнювати компоненти відносно сітки та між собою (Align to Grid, Align), задавати порядок відображення компонентів, які перетинаються (Bring to Front, Send to Back), змінювати розмір вибраного компонента (Size), масштабувати візуальні компоненти (Scale) тощо. Меню Search містить стандартні команди пошуку та заміни фрагмента тексту (Find, Replace, Search Again, lncremental Search) та інші. У меню View знаходяться команди візуалізації елементів середовища. Меню Project містить команди керування проектом, зокрема команди додавання файлів до проекту (Add to Project) та команди компіляції (Compile Unit, Build AII Project). Меню Run містить команди налагодження та запуску програми.

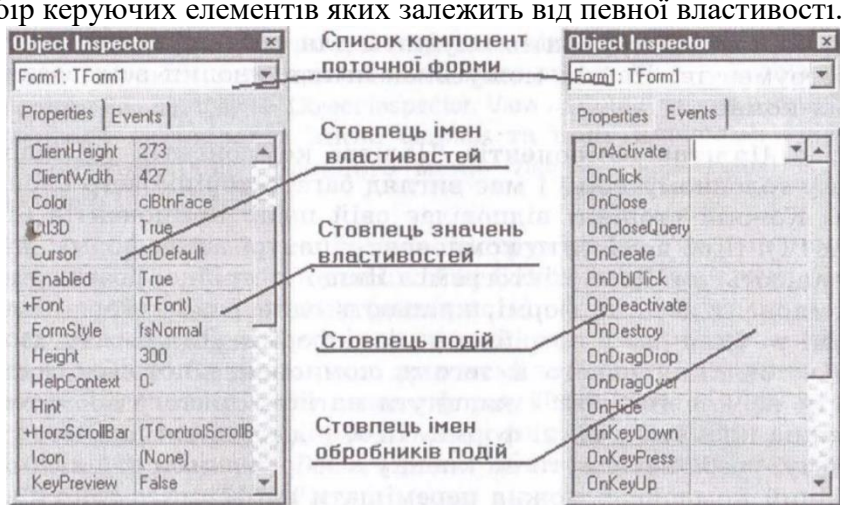
Меню Component використовують для створення та інсталяції нових компонентів. Меню Database містить команди виклику інструментів бази даних. У меню Tools містяться команди для задання параметрів середовища. Панель інструментів служить для розташування кнопок інструментів. На ній можуть міститися кнопки всіх зазначених команд.

Палітра компонентів. Палітра компонентів розташована у головному вікні і має вигляд багатосторінкового блокнота. Кожній сторінці відповідає свій набір компонентів (див.рис.1). Щоб помістити компонент у центрі вікна форми, двічі клацають на його піктограмі. Якщо потрібно розташувати компонент десь на формі, клацають один раз на його піктограмі та один раз у потрібному місці форми. Для багаторазової вставки одного й того ж компонента потрібно натиснути на клавішу Shift і клацнути на його піктограмі – тепер можна клацати у вікні форми. Щоб відмовитися від цього режиму, треба натиснути на кнопку з зображенням стрілки. Вибраний компонент можна переміщати на формі, а також змінювати розміри, перетягуючи його маркери.

Інспектор об'єктів. За допомогою інспектора об'єктів можна задавати початкові значення властивостей об'єкта та їхню реакцію на стандартні події. Вікно інспектора об'єктів містить список компонентів поточної форми, а також дві закладки: властивостей (Properties) та подій (Events). Щоб активізувати вікно інспектора об'єктів, використовують клавішу F11. Розглянемо це вікно.

Закладка властивостей складається з двох стовпців: лівий містить назви властивостей компонентів, а правий - їхні значення. Властивості можуть бути простими або комплексними.

Комплексні властивості складаються з набору інших властивостей. Такі властивості в іспекторі об'єктів позначені символом "+", наприклад *Font*. Закладка подій також має два стовпці. У лівому відображаються імена стандартних подій, на які об'єкт може реагувати, а в правому - імена методів-обробників (функцій), які реалізовуватимуть реакцію на подію. Кожній стандартній події відповідає назва методу, яка з'являється після подвійного клацання мишею у правому стовпці. У цей момент у вікно тексту програми додається шаблон базового коду (функції) для відповідного методу, який треба заповнити. Для введення значень властивостей числового і текстового типу (Width, Name тощо) використовується стандартне поле введення. Значення властивостей перерахованого типу (Align, Cursor, тощо) задаються комбінованим списком, звідки вибирають потрібне. Деякі комплексні властивості (Font, Picture, Glyph тощо) використовують діалогові вікна, набір керуючих елементів яких залежить від певної властивості.



Вікно форми. Форма - це вікно Windows, яке утворюється в одному з можливих для вікон стилів. Увесь внутрішній простір є робочою областю, яка має сітку вирівнювання для зручного розташування компонентів на формі. Для виконання групових операцій декілька компонентів можна об'єднувати. Для цього необхідно натиснути на ліву клавішу миші і переміщенням вказівника охопити всі потрібні компоненти. У групу долучаються компоненти, які хоча б частково потрапляють в охоплену область. Можна також долучити/вилучити окремий елемент. Для цього необхідно натиснути на клавішу Shift та, не відпускаючи її, вибрати мишею потрібний компонент на формі. Вилучення виокремлених компонентів чи групи виконується клавішею Delete, переміщення виокремленого компонента в межах форми мишею. Над компонентами та їхніми групами можна виконувати операції вирізання, копіювання в буфер обміну та вставляння з

буфера. Вирівнювати компоненти можна як відносно вікна форми, так і один відносно одного. Для цього використовується команда Edit => Align головного меню або палітра вирівнювання (команда View => Alignment Palette головного меню). Інша можливість - безпосередньо задати властивості Left і Top компонентів. Компоненти у групі вирівнюються відносно того компонента, який потрапив у групу першим.

24.3 Структура проекту

Проектом називають сукупність файлів, з яких C++ Builder створює готову для виконання програму. До складу кожного проекту обов' язково входять такі файли:

1. файл проекту *.bpr. Це невеликий файл, який містить посилання на всі файли проекту й ініціалізує програму;
2. файли опису всіх форм, які входять у проект: файл тексту програми *.cpp і файл форми *.dfm;
3. файл ресурсів програми *.res. У ньому описані ресурси, які не входять у форму, наприклад, піктограма програми;
4. файли заголовків "<.h. У них C++ Builder автоматично розміщує описи породжених класів, а користувач, декларації новостворених функцій;

Усі інші файли створюються після компіляції проекту. Для збереження проекту необхідно задати імена модулів (автоматично пропонуються імена Unitl.cpp, Unit2.cpp, ...) та ім'я проекту (Projectl.bpr). Ці імена можна змінити на власні. Для переміщення проекту на інший комп'ютер необхідно мати файли таких типів: *.bpr, *.dfm, *.cpp, *.res та '<.h. Інші файли створюються автоматично.

Редактор коду знаходиться в окремому вікні. Це вікно складається з двох половинок. Права половина організована як багатосторінковий блокнот відкритих у певний момент файлів. Ліва половина - вікно Class Explorer, у якому відображається ієрархія створених класів і функції проекту. Це вікно можна відкрити чи закрити командою головного меню View ~ Class Explorer а також командою View Explorer контекстового меню багатосторінкового блокнота. Під час створення нової форми у файл Unitl.cpp, який відповідає формі Form1, редактор автоматично заносить порожню заготовку конструктора TForm1::TForm1 класу TForm1. Цей клас породжується від інтегрованого у C++ Builder класу TForm (Форма), який володіє усіма властивостями стандартного вікна Windows. Опис наслідування класу користувача TForm1 від готового класу TForm знаходиться у заголовочному файлі Unitl.h. Посилання на цей файл (include), а також опис змінної Form1 типу TForm1 розміщені у модулі Umitl.cpp Під час додавання нових компонентів до форми у файл Unit.dfm автоматично заносяться описи параметрів цих компонентів (висота, ширина, розташування, стиль тощо). Застосування методу до певного об'єкта веде до появи заготовки базового коду відповідної функції у вікні редактора. Заготовка (шаблон) складається із заголовка функції та операторних дужок {}. Заготовку заповнює користувач. Файл Unitl.h має такий вигляд:

```
// Директиви препроцесора
class TForm1 : public TForm        // TForm1 породжується від TForm
  {
  __published:                     // Компоненти, видимі в Object Inspector,
                                   // та їхні методи
      TButton *Button1;
      TEdit *Edit1;
        . . .                      // Інші компоненти
      void __fastcall Button1Click(TObject *Sender);
      void __fastcall Edit1Change(TObject *Sender);
        . . .                      // Інші методи
  private:                         // Приватні оголошення
  public:                          // Загальнодоступні оголошення
                                   // Конструктор типу
      __fastcall TForm1(TComponent* Owner);
  };                               // Кінець опису типу TForm1
```

Файл Unitl.cpp має такий загальний вигляд:

```
// Директиви препроцесора, зокрема
   #include "Unit1.h"              // Додається опис типу TForm1
   #pragma resource "*.dfm"        // Додається файл з описом форми
   TForm1 *Form1;                  // Опис змінної Form1 типу TForm1

   //---------------------------------------------------------------------------
   __fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)             // Конструктор
        {
        }
   //---------------------------------------------------------------------------
   void __fastcall TForm1::Button1Click(TObject *Sender)
        {
        // Тут користувач записує тіло функції
        }
-------------------------------------------------------------------------------
   void __fastcall TForm1::Edit1Change(TObject *Sender)
        {
        // Тут користувач записує тіло функції
        }
   //---------------------------------------------------------------------------
   // . . . Реалізація інших методів
```

ЛЕКЦІЯ 25. ОСНОВИ СИСТЕМИ ЧИСЛЕННЯ

25.1 Принцип використання двійкової системи числення

Інформація (команди і дані) у комп'ютері кодуються у двійковій системі числення. Система числення – це система позначення чисел. У повсякденній практиці використовується десяткова система, в який числа записуються за допомогою десяти арабських цифр *0,1,2,3,…,9*.

У двійковій системі числення є лише дві цифри: 0 та 1. Зручність використання двійкової системи в обчислювальній техніці обумовлена тим, що перемикачі можуть перебувати тільки в одному з двох станів: увімкненому чи вимкненому. Ці стани можна кодувати двома цифрами: одиницею та нулем. Так само в двох станах у кожний окремий момент часу може перебувати канал передачі даних: «рівень напруги високий» або «рівень напруги низький». Крім того, логіка висловлень є двійковою логікою: будь-яке висловлення в кожний момент є істинним або хибним. Якщо висловлювання є істинним, йому відповідає значення 1, якщо хибним – значення 0.

Одна двійкова цифра називається *бітом* (від англ. *bi*nary digi*t* – двійкова цифра). За допомогою одного біта можна закодувати два інформаційних повідомлення, що умовно позначаються символами *0* та *1*. Із двох бітів можна утворити коди чотирьох інформаційних повідомлень: *00, 01, 10* та *11*; із трьох бітів – восьми. У загальному випадку за допомогою *n* бітів можна закодувати 2^n інформаційних повідомлень.

Послідовність, що складається з восьми бітів, у сучасній обчислювальній техніці прийнято називати *байтом*. За допомогою одного байта можна закодувати $2^8=256$ рівних повідомлень і відобразити, наприклад, значення цілих чисел від 0 до 255. Ці самі повідомлення можна розглядати і як числа від -128 до 127 (їх кількість теж дорівнює 256), символи, логічні значення тощо. Байт є одиницею обсягу пам'яті. Для позначення тисяч та мільйонів байтів використовуються такі одиниці як кілобайт ($1Кбайт = 2^{10} = 1\ 024$ *байт*), мегабайт ($1Мбайт = 2^{20} = 1\ 048\ 576$ *байт*) і гігабайт ($1Мбайт = 2^{30} = 1\ 073\ 741\ 824$ *байт*). Послідовностями двійкових цифр можна кодувати не лише числа, а й довільні інформаційні повідомлення. Зокрема, загальноприйнята система кодування ASCII ставить коди у відповідність 256 символам. Наприклад, латинській літері *«a»* відповідає код $97_{10} = 1100001_2$, а символу *«@»* - код $64_{10} = 1000000_2$. Складніші системи кодування дають можливість закодувати зображення, звук тощо.

25.2 Класифікація систем числення

Системою числення, або нумерацією, називається сукупність правил і знаків, за допомогою яких можна відобразити (кодувати) будь-яке невід'ємне число. До систем числення висуваються певні вимоги, серед яких найбільш важливими є вимоги однозначного кодування невід'ємних чисел 0, 1,… з деякої їх скінченної множини — діапазону P за скінченне число кроків і можливості виконання щодо чисел арифметичних і логічних операцій. Крім того, системи числення розв'язують задачу нумерації, тобто ефективного переходу від зображень чисел до номерів, які в даному випадку повинні мати мінімальну кількість цифр. Від вдалого чи невдалого вибору системи числення залежить ефективність розв'язання зазначених задач і її використання на практиці

Розрізняють такі типи систем числення:

– позиційні;
– змішані;
– непозиційні.

25.3 Позиційні системи числення

У позиційних системах числення одна і та ж цифра (числовий знак) у записі числа набуває різних значень залежно від своєї позиції. Таким чином, позиція цифри має вагу у числі. Здебільшого вага кожної позиції кратна деякому натуральному числу *b*, $b > 1$, яке називається *основою системи числення*.

Наприклад, якщо *b* - натуральне число ($b > 1$), то для представлення числа *x* у системі числення з основою *b* його подають у вигляді лінійної комбінації степенів числа *b*:

$$x = \sum_{k=0}^{n} a_k b^k \qquad (1)$$

де a_k – цілі, $0 \le a_k < b$.

Іншими словами, основа - це кількість символів, що використовуються при записуванні чисел.

Наприклад, число *204* представляється у десятковій системі числення у вигляді:

$204 = 2 \cdot 10^2 + 0 \cdot 10^1 + 4 \cdot 10^0$

Використовуючи позиційний принцип, можна зобразити будь-яке дійсне число за допомогою усього лише десяти цифр у їх різних комбінаціях.

Крім десяткової найбільш поширеними є системи числення з основами:

2 – двійкова (у дискретній математиці, інформатиці, програмуванні)

8 – вісімкова (у програмуванні)

16 – шістнадцяткова (поширена у програмуванні, а також для кодування шрифтів)

60 – шістдесяткова (для виміру кутів і, зокрема, довготи і широти)

За одночасної роботи із кількома системами числення для їх розрізнення основи систем зазвичай вказують нижнім індексом, який записується у десятковій системі:

453_{10} – це число *453* у десятковій системі числення;

111000101_2 – те ж число, але у двійковій системі;

705_8 – те ж число, але у вісімковій системі.

У деяких спеціальних галузях застосовуються особливі правила вказування основи. Наприклад, у програмуванні шістнадцяткова система позначається:

– у мові асемблера і записах загального роду, не прив'язаних до конкретної мови, буквою *h* (від *hexadecimal*) у кінці числа (синтаксис Intel) — *75h*;

– у PASCAL знаком *$* на початку числа;

– у C і багатьох інших мовах комбінацією *0x* або *0X* (від *hexadecimal*) на початку.

Двійкова система числення використовує для запису чисел тільки два символи, зазвичай *0* (нуль) та *1* (одиницю). Двійкова система числення є позиційною системою числення, база якої дорівнює двом. Завдяки тому, що таку систему доволі просто використовувати в електричних схемах, двійкова система отримала широке розповсюдження у світі обчислювальних пристроїв.

Двійкове число можна представити як послідовність будь-яких об'єктів, які можуть знаходитися в одному з двох можливих станів. Наприклад:

Числа, що можуть приймати значення *0* або *1: 1 0 1 0 0 1 1*.

Позиції, на яких можуть стояти хрестики або нулики: *x o x o o x x*.

Вузли електричної схеми, які може бути, а може не бути знеструмлено.

Ділянки магнітної смужки, які може бути, а може не бути намагнічено.

Зазвичай, для позначення двійкових чисел використовують нулі та одиниці. Перші персональні комп'ютери для відображення чисел мали ряд електричних лампочок (кожна з яких, зрозуміло, може або світитися, або бути вимкненою).

Лічити у двійковій системі не складніше, ніж у будь-якій іншій. У десятковій системі, коли число у поточному розряді сягає десяти, то розряд обтулюється і одиниця додається до старшого. Наприклад: *9+1=10, 44+7=51*. Аналогічним чином у двійковій системі: коли число в розряді сягає двох – розряд обтулюється і одиниця додається до старшого розряду. Тобто: *1+1=10*. *10* у цьому записі – двійкове число, у десятковій системі це число записується як *2*. А десяткове *9+1=10* у двійковій системі буде виглядати так: *1001+1=1010* (після додавання одиниці число в останньому розряді дорівнює двом, тож розряд обтулюється і одиниця додається до передостаннього(старшого) розряду).

Конвертування десяткових чисел у двійкові здійснюється за формулою (1), де *b=2*.

$$x = a_0 + a_1 \times 2 + a_2 \times 2^2 + a_{N-1} \times 2^{N-1} \quad (2)$$

Вісімкова система числення – позиційна цілочисельна система числення з основою *8*. Для представлення чисел в ній використовуються цифри *0* до *7*. Вона часто використовується в галузях, пов'язаних з цифровими пристроями. Характеризується легким переводом вісімкових чисел у двійкові і назад, шляхом заміни вісімкових чисел на триплети двійкових. Раніше широко використовувалася в програмуванні і взагалі комп'ютерної документації, проте в наш час майже повністю витіснена шістнадцятковою.

Таблиця переведення вісімкових чисел в двійкові

$0_8 = 000_2$	$1_8 = 001_2$	$2_8 = 010_2$	$3_8 = 011_2$
$4_8 = 100_2$	$5_8 = 101_2$	$6_8 = 110_2$	$7_8 = 111_2$

Для переведення вісімкового числа в двійкове необхідно замінити кожну цифру вісімкового числа на триплет двійкових цифр. Наприклад: $2541_8 = 010\ 101\ 100\ 001 = 010101100001_2$

Шістнадцяткова система числення — це позиційна система числення, кожне число в якій записується за допомогою 16-ти символів. Цю систему часто називають також Hex (початкові літери англ. *hexadecimal* – шістнадцятковий). Спочатку планувалось вживати латинське sexa замість hexa, проте це слово сприймалось неоднозначно. Для запису чисел в цій системі окрім 10 арабських цифр (від *0* до *9*) використовують 6 літер латинської абетки: *A, B, C, D, E, F*.

Запис числа формується за загальним принципом: на *n*-й позиції (справа наліво від *0*) стоїть цифра, що відповідає кількості *n*-х степенів шістнадцяти у цьому числі. Наприклад, число записане в десятковій системі як *1000*, в hex записується як *3E8*, де:

$3\times16^2 + 14\times16^1 + 8\times16^0 = 768 + 224 + 8 = 1000.$

Шістнадцяткова система числення широко застосовується в інформатиці, оскільки значення кожного байту можна записати у вигляді двох цифр шістнадцяткової системи. Таким чином значення послідовних байтів можна представити у вигляді списку двозначних чисел. В той же час запис 4 бітів можна представити однією шістнадцятковою цифрою.

0_{hex}	0_{dec}	0_{oct}	0	0	0	0
1_{hex}	1_{dec}	1_{oct}	0	0	0	1
2_{hex}	2_{dec}	2_{oct}	0	0	1	0
3_{hex}	3_{dec}	3_{oct}	0	0	1	1
4_{hex}	4_{dec}	4_{oct}	0	1	0	0
5_{hex}	5_{dec}	5_{oct}	0	1	0	1
6_{hex}	6_{dec}	6_{oct}	0	1	1	0
7_{hex}	7_{dec}	7_{oct}	0	1	1	1
8_{hex}	8_{dec}	10_{oct}	1	0	0	0
9_{hex}	9_{dec}	11_{oct}	1	0	0	1
A_{hex}	10_{dec}	12_{oct}	1	0	1	0
B_{hex}	11_{dec}	13_{oct}	1	0	1	1
C_{hex}	12_{dec}	14_{oct}	1	1	0	0
D_{hex}	13_{dec}	15_{oct}	1	1	0	1
E_{hex}	14_{dec}	16_{oct}	1	1	1	0
F_{hex}	15_{dec}	17_{oct}	1	1	1	1

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: МЕТОДИ ПЕРЕВОДУ ЧИСЕЛ З ОДНІЄЇ СИСТЕМИ ЧИСЛЕННЯ В ІНШУ

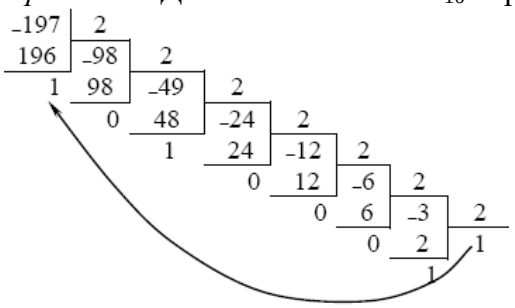
Переведення цілого числа з десяткової системи числення у будь-яку іншу

Метод ділення

Для перетворення цілого числа з десяткової системи числення в будь-яку іншу позиційну систему необхідно розділити десяткове число на основу нової системи числення, потім отриману частку знову розділити на основу нової системи числення і так доти, поки в частці не залишиться число менш, ніж основа нової системи числення.

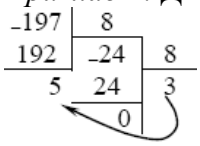
Число в новій системі числення запишеться у вигляді залишків від ділення, починаючи з останньої частки. Тобто перший залишок дає молодшу цифру, а останній – старшу.

Приклад 1. Десяткове число 197_{10} перетворити в двійкову систему числення.



Отже, $197_{10} = 11000101_2$.

Приклад 2. Десяткове число 197_{10} перетворити в вісімкову систему числення.



Отже, $197_{10} = 305_8$.

Приклад 3. Десяткове число 197_{10} перетворити в шістнадцяткову систему числення.

При перетворенні десяткового числа в шістнадцяткову систему треба враховувати, що алфавіт у шістнадцятковій системі числення, починаючи з 10-го символу, має букви *A, B, C, D, E, F*, тому якщо в результаті ділення отримуємо числа більш, ніж 9, їх треба переводити в символи шістнадцяткової системи.

Отже $197_{10} = C5_{16}$.

Метод віднімання

З десяткового числа віднімається найбільш можлива степінь двійки, у відповідний розряд двійкового числа записується одиниця, якщо різниця менше наступного степеня двійки, то далі записується нуль, а якщо більше – записується одиниця і знову проводиться віднімання, і так доти, поки вихідне число не зменшиться до нуля.

Приклад 4. Десяткове число 149_{10} перетворити в двійкове методом віднімання.

$149_{10} = 10010101_2$

$128 = 2^7$

21

$16 = 2^4$

5

$4 = 2^2$

1

$1 = 2^0$

0

Таким чином $149_{10} = 10010101_2$.

Приклад 5. Десяткове дробове число $685{,}5_{10}$ перетворити в двійкове методом віднімання.

$685{,}5_{10} = 1010101101{,}1_2$

$512 = 2^9$

173,5

$128 = 2^7$

45,5

$32 = 2^5$

13,5

$8 = 2^3$

5,5

$2 = 2^4$

3,5

$2 = 2^1$

1,5

$1 = 2^0$

0,5

$0{,}5 = 2^{-1}$

0

Таким чином $685{,}5_{10} = 1010101101{,}1_2$.

Метод множення

Даний метод застосовується для перетворення десяткових дробів, зокрема для чисел менших одиниці. При цьому число множиться на 2, якщо результат ≥1, то в старший розряд записується одиниця, якщо ні, то нуль. Множимо на 2 тільки дробову частину результату і повторюємо процедуру далі до отримання потрібного степеня точності або до обнулення результату.

Приклад 6. Десяткове число $0{,}321_{10}$ перевести в двійкове методом множення.

0,321 · 2 = 0,642
0,642 · 2 = 1,284
0,284 · 2 = 0,568
0,568 · 2 = 1,136
0,136 · 2 = 0,272
0,272 · 2 = 0,544

Отже $0{,}321_{10} = 0{,}010100_2$.

Приклад 7. Десяткове число $0{,}32812510_{10}$ перевести у вісімкове методом множення.

$0{,}328125 \cdot 8 = 2{,}625$	-2
$0{,}625 \cdot 8 = 5{,}00$	-5

Таким чином $0{,}32812510_{10} = 0{,}25_8$.

Приклад 8. Десяткове число $0{,}32812510_{10}$ перевести в шістнадцяткове методом множення.

$0{,}328125 \cdot 16 = 5{,}25$	-5
$0{,}25 \cdot 16 = 4{,}00$	-4

Таким чином $0{,}3281251010 = 0{,}54_{16}$.

Перетворення з двійкової, вісімкової, шістнадцяткової систем числення в десяткову

Для перетворення числа з системи числення з основою *p* в десяткову систему числення необхідно скористатися формулою 1.1 і кожній позиції числа присвоїти певну вагу. Потім значення ваги позиції множиться на коефіцієнт, що займає цю позицію. Результати операцій множення, виконаних для всіх позицій числа, підсумовуються.

Приклад 9. Двійкове число 11001100_2 перевести в десяткову систему числення.

$$\overset{7}{1}\ \overset{6}{1}\ \overset{5}{0}\ \overset{4}{0}\ \overset{3}{1}\ \overset{2}{1}\ \overset{1}{0}\ \overset{0}{0}_2$$

$$11001100_2 = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 =$$

$$= 128 + 64 + 8 + 4 = 204_{10}.$$

Приклад 10. Двійкове число $110111{,}11_2$ перевести в десяткову систему числення.

$$\overset{5}{1}\ \overset{4}{1}\ \overset{3}{0}\ \overset{2}{1}\ \overset{1}{1}\ \overset{0}{1},\ \overset{-1}{1}\ \overset{-2}{1}_2$$

$$110111{,}11_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$$

$$= 32 + 16 + 4 + + 2 + 1 + 0.5 + 0{,}25 = 55{,}75_{10}.$$

Приклад 11. Вісімкове число 537_8 перевести в десяткове.

$$\overset{2}{5}\ \overset{1}{3}\ \overset{0}{7}_8 = 5 \cdot 8^2 + 3 \cdot 8^1 + 7 \cdot 8^0 = 320 + 24 + 7 = 351_{10}.$$

Приклад 12. Вісімкове число $1172{,}25_{(8)}$ перевести в десяткове.

$$\overset{3}{1}\overset{2}{1}\overset{1}{7}\overset{0}{2},\overset{-1}{2}\overset{-2}{5}_8 = 1 \cdot 8^3 + 1 \cdot 8^2 + 7 \cdot 8^1 + 2 \cdot 8^0 + 2 \cdot 8^{-1} + 5 \cdot 8^{-2} =$$

$$= 634{,}328125_{10}.$$

Приклад 13. Шістнадцяткове число $3A2_{16}$ перевести в десяткове.

$$\overset{2}{3}\ \overset{1}{B}\ \overset{0}{2}_{16} = 3 \cdot 16^2 + 11 \cdot 16^1 + 2 \cdot 16^0 = 768 + 160 + 2 = 946_{10}.$$

Приклад 14. Шістнадцяткове число $27A{,}54_{16}$ перевести в десяткове.

$$\overset{2}{2}\overset{1}{7}\overset{0}{A},\overset{-1}{5}\overset{-2}{4}_{16} = 2 \cdot 16^2 + 7 \cdot 16^1 + 10 \cdot 16^0 + 5 \cdot 16^{-1} + 4 \cdot 16^{-2} =$$

$$= 634{,}328125_{10}.$$

Перетворення з двійкової системи числення в вісімкову і шістнадцяткову та навпаки

Для перетворення з двійкової системи числення у вісімкову, необхідно згрупувати (починаючи з молодшого розряду) по три біти (тріади), далі кожну групу записати однією вісімковою цифрою (табл.1).

Таблиця 1 – Перетворення двійкових тріад у вісімкові цифри

Двійкові тріади	000	001	010	011	100	101	110	111
Вісімкові цифри	0	1	2	3	4	5	6	7

Приклад 15. Двійкове число 1111011011001_2 перевести в вісімкове:

$$\underbrace{001}_{1}\underbrace{111}_{7}\underbrace{011}_{3}\underbrace{011}_{3}\underbrace{001}_{1},$$

$$001_2 = 1_8;\ 011_2 = 3_8;\ 011_2 = 3_8;\ 111_2 = 7_8;\ 001_2 = 1_8.$$

Отже $1111011011001_2 = 17331_8$.

З цього прикладу видно, що старші розряди двійкового числа треба доповнювати нулями до 3-х розрядів (тріад) у двійковому коді.

Для перетворення з двійкової системи числення в шістнадцяткову, необхідно згрупувати (починаючи з молодшого розряду) по чотири біта (тетради), далі кожну групу записати однією шістнадцятковою цифрою (табл. 2).

Таблиця 2 – Перетворення двійкових тетрад у шістнадцяткові цифри

Двійкові тетради	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
Шістнадцяткові цифри	0	1	2	3	4	5	6	7	8	9	*A*	*B*	*C*	*D*	*E*	*F*

Приклад 16. Двійкове число 111001011011001_2 перевести в шістнадцяткове.

$$\underbrace{0111}_{7}\underbrace{0010}_{2}\underbrace{1101}_{D}\underbrace{1001}_{9},$$

$$1001_2 = 9_{16};\ 1101_2 = D_{16};\ 0010_2 = 2_{16};\ 0111_2 = 7_{16}.$$

Отже $111001011011001_2 = 72D9_{16}$.

Для перетворення з вісімкової системи числення у двійкову, необхідно кожну цифру вихідного числа записати у вигляді еквівалентного трибітного двійкового числа (див. табл. 1.3).

Приклад 17. Вісімкове число 7265_8 перевести у двійкову систему числення.

$$\underbrace{7}_{111}\ \underbrace{2}_{010}\ \underbrace{6}_{110}\ \underbrace{5}_{101}$$

$$5_8 = 101_2;\ 6_8 = 110_2;\ 2_8 = 010_2;\ 7_8 = 111_2.$$

Отже $7265_8 = 111010110101_2$.

Для перетворення з шістнадцяткової системи числення в двійкову, необхідно кожну цифру вихідного числа записати у вигляді еквівалентного чотирибітного двійкового числа (див. табл. 1.3).

Приклад 18. Шістнадцятирічне число *A*2*E*5*F*(16) перевести в двійкову систему числення.

$$\underbrace{A}_{1010}\ \underbrace{2}_{0010}\ \underbrace{E}_{1110}\ \underbrace{5}_{0101}\ \underbrace{F}_{1111}$$

$$F_{16} = 1111_2;\ 5_{16} = 0101_2;\ E_{16} = 1110_2;\ 2_{16} = 0010_2;\ A_{16} = 1010_2.$$

Таким чином $A2E5F_{16} = 10100010111001011111_2$.

Перетворення з вісімкової системи числення в шістнадцяткову і навпаки відбувається за допомогою двійкового коду. Для перетворення вісімкового числа в шістнадцяткову систему числення спочатку це число перетворюють у двійкову систему, потім, розбиваючи на тетради, починаючи з молодшого біта, перетворюють у шістнадцяткову за допомогою табл. 1.3. Для перетворення числа з шістнадцяткової системи у вісімкову це число перетворюють у двійкову систему, потім розбивають його на тріади, починаючи з молодшого біта, і замінюють тріади відповідними еквівалентами у вісімковой системі.

Приклад 19. Вісімкове число 24738 перевести в шістнадцяткову систему числення.

$$\underbrace{2}_{010}\ \underbrace{4}_{100}\ \underbrace{7}_{111}\ \underbrace{3}_{011}$$

$$3_8 = 011_2;\ 7_8 = 111_2;\ 4_8 = 100_2;\ 2_8 = 010_2.$$

$$\underbrace{0101}_{5}\underbrace{0011}_{3}\underbrace{1011}_{B}$$

$$1011_2 = B_{16};\ 0011_2 = 3_{16};\ 0101_2 = 5_{16}.$$

Отже $2473_8 = 53B_{16}$.

Приклад 20. Шістнадцяткове число $CA5F_{16}$ перевести у вісімкову систему числення.

$$\underbrace{C}_{1100}\ \underbrace{A}_{1010}\ \underbrace{5}_{0101}\ \underbrace{F}_{1111}$$

$$F_{16} = 1111_2;\ 5_{16} = 0101_2;\ A_{16} = 1010_2;\ C_{16} = 1100_2.$$

$$\underbrace{001}_{1}\underbrace{100}_{4}\underbrace{101}_{5}\underbrace{001}_{1}\underbrace{011}_{3}\underbrace{111}_{7}$$

$$111_2 = 7_8;\ 011_2 = 3_8;\ 001_2 = 1_8;\ 101_2 = 5_8;\ 100_2 = 4_8;\ 001_2 = 1_8.$$

Таким чином $CA5F_{16} = 145137_8$.

ЛЕКЦІЯ 26. АВТОМАТИЗАЦІЯ ВИРОБНИЦТВА З ВИКОРИСТАННЯМ ПРОГРАМНИХ КОМПЛЕКСІВ

26.1 Головні напрямки автоматизації

Технологічний процес в промисловості нерозривно зв'язаний з її автоматизацією технологічних процесів. Автоматизація ефективно застосовується на сучасному етапі розвитку людства з метою досягнення зростання показників ресурсозбереження, поліпшення екології навколишнього середовища якості та надійності продукції. В зв'язку з бурхливим розвитком мікропроцесорної техніки і персонально електронно-обчислювальних машин, функціональні можливості яких дають змогу використовувати найдосконаліші методи в рамках сучасних складних систем управління. Мікропроцесорні пристрої та електронно-обчислювальних машини, пов'язані між собою обчислювальними та керуючими мережами з використанням загальних баз даних, дозволяють впроваджувати комп'ютерні технології у нетрадиційній сфері діяльності підприємства, що проявляється в інтеграції виробничих процесів та управління ними.

Головним напрямом автоматизації на сучасному етапі є створення комп'ютерно-інтегрованих виробництв. Основою систем автоматизації стали функціональні можливості мікропроцесорних систем управління, при створенні яких вирішальну роль відіграють такі фактори, як використання принципів інтеграції, розподіленого управління, програмних комплексів. При автоматизації виробництва об'єктом є не окремий технологічний процес чи агрегат, а технологічний комплекс із складними взаємозв'язками між його підсистемами.

Підвищити оперативність управління, максимально враховувати виробничу ситуацію дає можливість розширення функціональних можливостей сучасних мікропроцесорних систем управління пов'язано із значно зрослою кількістю видів і систем відображення технологічної інформації: використанням динамічних мікросхем; одержанням графіків технологічних параметрів за будь-який відрізок часу; формування передісторії і розвитку процесу; архівування за допомогою таблиць, звітних документів тощо.

При системному підході автоматизація виробництва дає кращі результати, коли досконало вивчаються властивості обєкта автоматизації, розробляється функціональна структура як сукупність виконуваних системою функцій.

Нині існує велика кількість визначень «система», оскільки в різних ситуаціях в нього вкладається різний зміст, але в будь-якому випадку система являє собою підмножину взаємоз'язаних елементів певної природи, залежно від розв'язуваного завдання.

При створенні систем автоматизації використовують багато контурні системи, в яких реалізуються принципи компенсації збурень, адаптації, досконалі структури типу каскадних систем з додатковими сигналами та інше.

26.2 Розвиток автоматики

Слово «автомат» у перекладі з грецького означає «самодіючий». У Древній Греції так називалися механізми і пристрої, що могли самостійно, без видимої участі людини виконувати будь-які дії. Створювалися механічні птахи, звірі і різні фантастичні тварини, що махали крилами, рухатися і ричати. Звичайно, практичної користі від таких «автоматів» було небагато, але саме вони стали попередникам сучасних автоматів.

Інтенсивний розвиток автоматики почався в XVIII-XIX ст. у зв'язку з промисловим переворотом в Європі, пов'язаним з використанням енергії пари.

Першим промисловим «регулятором» того часу був поплавковий «регулятор», розроблений І.І. Ползуновим , яку він побудував у 1765 році.

На принципі зміни керованих технологічних параметрів залежно від їх відхилення відносно заданого значення в 1784 році англійський механік Джорж Уатт побудував відцентровий «регулятор» швидкості парової машини. Принцип керування за відхиленням величини від заданого значення, відомий як принцип Ползунова - Уатта, дістав поширення в сучасній техніці.

У 1830 році Шиллінг у розробленому ним телеграфі запропонував перше електромагнітне реле, яке дістало практичне застосування в різних сферах промисловості.

У 1856 - 1871 році В.М. Чиколаєв розробив регулятори для дугових ламп, а в 1871 році

математик П.Л. Чебишев у своїй праці про відцентровий регулятор почав теоретичні дослідження автоматичних регуляторів.

Одним із фундаторів теорії автоматичного керування вважається професор Петербурзького практичного технологічного інституту І. О. Вишнєградський, який опублікував у 1876 і 1878 роках свої класичні праці «Про загальну теорію регуляторів» та «регулятори прямої дії».

Велике значення для розвитку теорії автоматичного керування мали дослідження академіка О.М. Ляпунова, який в 1892 році у своїй праці «Загальна задача про стійкість руху» заклав основи теорії стійкості нелінійних динамічних систем, а також обґрунтував вихідні положення лінійної теорії автоматичного керування.

Важливою подією було опубліковано М.Є. Жуковським у 1909 році першого російського підручника «Теорія регулювання ходу машин», в якому, крім узагальнення відомих положень, було наведено нові дослідження регулятора з сухим тертям, основи теорії переривчастого регулювання.

В XX столітті енергія пари дедалі більше змінювалась електричною енергією, і питанням автоматизації різних електроустановок приділялося більше уваги. У цей період виникають автоматичні електростанції, автоматизуються окремі промислові ділянки, цехи та цілі підприємства. Ставляться і вирішуються завдання комплексної автоматизації цілих промислових процесів і виробництв.

В 60-80- ті роки теорія автоматичного керування вирішує усе складніші питання з розробки нових систем, методів їх дослідження та синтезу.

Великою подією у розвитку теорії автоматичного керування була поява в 1948 і 1952 році праць американського вченого Н. Вінера, які стали основою нового напрямку розвитку - кібернетики. Академік А.М.Комогоров визначив кібернетику як вчення про способи добування, збереження, перетворення і використання інформації в машинах, живих організмах та їх обєднаннях. Принципи кібернетики як загальної науки про керування в найрізноманітніших умовах (системах) покладено в основу сучасних термінів і понять теорії автоматичного керування.

Нині в умовах науково–технічної революції автомати знаходять широке застосування в промисловості, на транспорті й у дослідницьких лабораторіях. Але яку б роботу не виконував автомат, він працює не сам по собі, його робота визначається програмою – визначеною послідовністю дій, що задається людиною. Програма роботи автомата може бути закладена в його конструкції. Наприклад, програма роботи часів міститься в пристрої спускового механізму і маятника, що одержують енергію від заводної пружини. У більш складних автоматах, як, наприклад, у верстатах із програмним керуванням, програма роботи задається ззовні у виді серії сигналів, записаних на магнітну стрічку, перфокарту або на спеціальний чіп. Пристрій, вмонтований у блок керування верстатом, «зчитує» ці сигнали і посилає їх на механізми, які виконують необхідно задані операції.

Автомати також мають «органи почуттів» – різноманітні чуттєві елементи, чи датчики, що сприймають зміни освітленості, тиску, переміщення, температури, звуку . Датчики виробляють сигнали, що по «нервових волокнах» – ланцюгам прямої і зворотного зв'язку – надходять у «центральну нервову систему» – пристрій керування автомата. У залежності від призначення автомата і складності його конструкції пристроєм керування може бути звичайне реле, а в складних автоматах – навіть ЕОМ.

У відповідь на сигнали датчиків пристрій керування посилає імпульсу-команди – виконавчим пристроям. З їхньою допомогою автомати пересувають важелі, поршні і заслінки робочих машин, пускають у хід високовольтні вимикачі, піднімають багатотонні вантажі, керують кермовими системами кораблів і літаків. Якщо сигнали слабкі, те їх підсилюють у спеціальних пристроях – підсилювачах.

Завдяки бурхливому розвитку техніки в XX ст. з'явилися енергетичні, технологічні, транспортні й інші машини й агрегати з автоматичним керуванням. Широке використання у виробничих процесах автоматичного й автоматизованого устаткування – це і є автоматизація виробництва. Якщо механізація звільняє людину тільки від важкої фізичної праці, те автоматизація передбачає передачу автоматичним пристроям також і функцій керування,

регулювання і контролю, що раніш виконував людину.

26.3 Основні поняття та терміни в автоматизації

Автоматизація – один з основних факторів сучасної науково-технічної революції. Її ціль – підвищення ефективності праці, поліпшення якості продукції, що випускається, створення сприятливих умов для найбільш раціонального використання всіх ресурсів виробництва.

В основі автоматизації виробництва лежить системний підхід до побудови і використання комплексу засобів автоматичного керування, регулювання і контролю. В автоматиці широко використовуються новітні досягнення в області науки і техніки, що дозволяє повніше розкрити можливості технологічного устаткування.

Автоматизована система управління (АСУ) – так називаються системи керування, у яких процес керування здійснюється частково автоматично, а частково при участі людини.

Людина координує роботу окремих ланок АСУ, оцінює результати обробки інформації, в екстрених випадках бере на себе оперативне керування.

Участь людини в роботі АСУ особливо необхідно тоді, коли ті чи інші дії в процесі керування здійснюються на підставі досвіду людини, його інтуїції і тому не можуть бути запрограмовані.

Найважливіша науково-технічна передумова для створення АСУ – можливість автоматизації різних інформаційних процесів на основі широкого використання обчислювальної техніки.

У загальному плані автоматизація виробництва – це етап машинного виробництва, що характеризується звільненням людини від безпосереднього виконання функцій управління виробничими процесами та передачею цих функцій технічним засобом – автоматичним пристроям і системам. В основі автоматизації виробництва лежить поняття «управління». Управління – цілеспрямована дія на процес (обєкт), яка забезпечує оптимальний чи заданий режим його роботи. Процес управління, з точки зору автоматичних систем, складаються з ряду елементарних операцій та етапів, які є спільними для технічних систем і систем живої природи.

Незалежно від мети, призначення, структури обєкта процес управління передбачає виконання таких операцій, як:

– одержання та попередня обробка інформації про фактичний стан обєкта, системи і навколишнього середовища;

– аналіз одержаної інформації, порівняння існуючої виробничої ситуації із даною;

– прийняття рішення про дію на обєкт у певному напрямку та оцінка можливості реалізації такої дії;

– реалізація управління, тобто формування дії за допомогою відповідних технічних засобів.

При здійсненні процесу управління часто доводиться спочатку відшуковувати потрібний режим роботи, а потім його підтримувати. В окремих випадках для простих обєктів значення технологічних параметрів задаються наперед, тоді системи називаються системами автоматичного регулювання САР. Сучасні автоматичні та автоматизовані системи є за своєю структурою розподіленими і базуються на мережевих технологій з використанням мікропроцесорних засобів.

Обєкт автоматизації – будь-який технологічний апарат, процес, машина, установка які підлягають автоматизації.

Сучасні системи автоматизації обєднуються у складні комп'ютерно-інтегровані системи. Розглядаючи їх, слід передусім, наголосити на тому, що сукупність взаємоз'язаних і взаємодіючих елементів у них призначена для досягнення певних цілей. сукупність елементів системи та характери зв'язків між ними визначаються структурою останньої. При створенні й аналізі систем автоматизації виділяють такі структури:

– функціональну – сукупність частин для виконання окремих функцій: одержання інформації, її обробки, передачі і т.д.;

– алгоритмічну – сукупність частин для виконання певних алгоритмів обробки інформації;

– технічну – сукупність необхідних технічних засобів як відображення функціональної та алгоритмічної структур.

Основні переваги автоматизації полягають у можливостях забезпечити:

– зростання продуктивності та поліпшення умов праці;

– виконання робіт в важкодоступних та взагалі недоступних для людини сферах (радіоактивні зони, космос окремі види металургійного та інших виробництв);

– підвищення точності, якості технологічних процесів і відповідних виробів;

– зростання надійності та техніко-економічних показників і загальної культури виробництва та кваліфікації обслуговуючого персоналу.

Автоматизація виробництва проводиться автоматичних пристроїв, які можна класифікувати за різними ознаками. Однією з найпоширеніших є класифікація за функціональним призначенням пристрою, згідно з якою виділяють такі автоматичні пристрої:

– автоматичного контролю та сигналізації;

– автоматичного захисту;

– обчислюванні;

– автоматичного керування.

Пристрої автоматичного контролю та сигналізації забезпечують контроль за перебігом технологічних процесів, станом приміщень та відповідно сигналізацію. При нормальних умовах процесів використовується оптична сигналізація, а при появі відхилень від цих умов – оптична та акустична сигналізація.

Пристрої автоматичного захисту забезпечують захист обєктів при появі загрози для обладнання, продукції або обслуговуючого персоналу.

Обчислювально-лічильні пристрої виконують самостійно складні розрахунки робіт супутників, ракет, найвигідніших технологічних режимів роботи, експрес-аналізу та ін.

Блокуючі пристрої мають призначення не допускати виконання хибних команд.

Пристрої автоматичних керування забезпечують бажані зміни в ході процесів. Це – найскладніші й дуже поширені пристрої автоматики, роботу яких вивчає «Теорія автоматичного керування». Управління – це цілеспрямована дія на об'єкт яка забезпечує оптимальний чи заданий режим його роботи. Процес управління складається з ряду елементарних операцій та етапів, які є спільними для технічних систем і систем живої природи. Незалежно від мети, призначення, структури обєкта процесу управління передбачає виконання таких операцій, як:

– одержання та попередня обробка інформації про фактичний стан обєкта, системи і навколишнього середовища;

– аналіз одержаної інформації, порівняння існуючої виробничої ситуації із заданою;

– прийняття рішення про дію на обєкт у певному напрямку та оцінка можливості реалізації такої дії;

– реалізація управління, тобто формування і здійснення дії за допомогою відповідних технічних засобів.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: SCADA - СИСТЕМИ

Термін SCADA-система використають для позначення програмно-апаратного комплексу збору даних (телемеханічного комплексу)

До основних завдань, що розв'язуються SCADA-системами, відносять:

– обмін даними в реальному часі з ПЗО (пристроєм зв'язку з об'єктом, що контролюється). Цим пристроєм може бути як промисловий контролер, так і плата вводу/виводу

– обробка інформації в реальному часі

– відображення інформації на екрані монітора в зрозумілій для людини формі (HMI — скор. від англ. Human Machine Interface

– людино-машинний інтерфейс)

– ведення бази даних реального часу з технологічною інформацією

– аварійна сигналізація й управління тривожними повідомленнями

– підготовка й генерування звітів про хід технологічного процесу

– створення архіву технологічної інформації (збір історії)

– забезпечення зв'язку із зовнішніми додатками (СУБД, електронними таблицями, текстовими процесорами й т.д.); у системі управління підприємством такими додатками найчастіше є додатки, що відносяться до рівня MES.

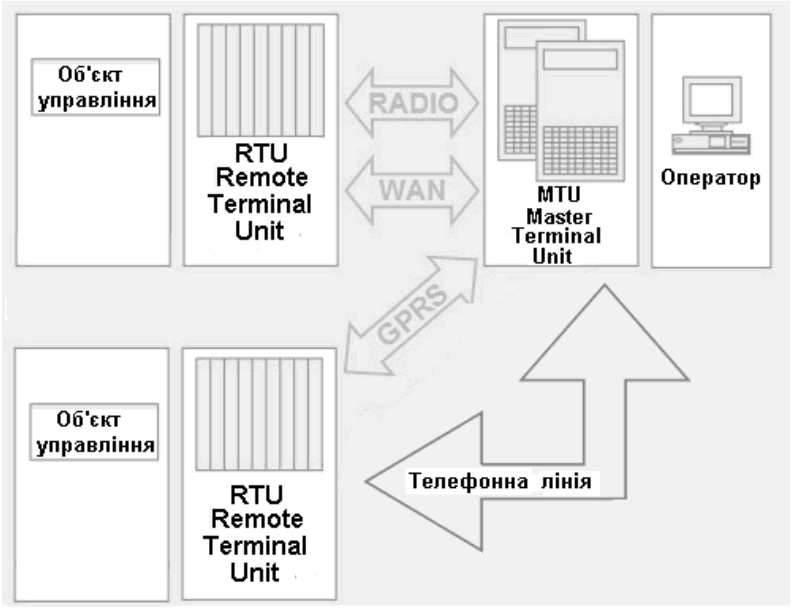
Іноді SCADA-системи комплектуються додатковим ПЗ для програмування промислових контролерів. Такі SCADA-системи називаються інтегрованими, і до них додають термін SoftLogic. Системи такого класу мають чітке призначення – вони надають можливість здійснювати моніторинг і диспетчерський контроль безлічі віддалених точок (від 1 до 10000 пунктів контролю, іноді на відстані в тисячі кілометрів один від одного) або одного територіально розподіленого об'єкта.

Основне завдання SCADA – це збір інформації про безліч віддалених об'єктів, що надходить із пунктів контролю, і відображення цієї інформації в єдиному диспетчерському центрі. Крім цього, SCADA повинна забезпечувати довгострокове архівування отриманих даних. При цьому диспетчер найчастіше має можливість не тільки пасивно спостерігати за об'єктом, але й обмежено їм управляти, реагуючи на різні ситуації.

Загальна структура SCADA-системи

Робота SCADA – це безперервний процес збору інформації реального часу з віддалених пунктів (об'єктів) для обробки, аналізу й можливого управління. Вимога обробки реального часу обумовлено необхідністю оперативної доставки (видачі) всіх повідомлень і даних на центральний інтерфейс оператора (диспетчера). У той же час поняття реального часу відрізняється для різних SCADA-систем.

Всі сучасні SCADA-системи включають три основних структурних компоненти.



Remote Terminal Unit (RTU) - віддалений термінал, що підключається безпосередньо до контрольованого об'єкта й здійснюючу обробку завдання (управління) у режимі реального часу. Спектр втілень RTU широкий: від примітивних датчиків, що здійснюють збір інформації з об'єкта, до спеціалізованих багатопроцесорних відмово стійких обчислювальних комплексів, що здійснюють обробку інформації й управління в режимі жорсткого реального часу.

Конкретна його реалізація визначається специфікою застосування. Використання пристроїв низько рівневої обробки інформації дозволяє знизити вимоги до пропускної здатності каналів зв'язку із центральним диспетчерським пунктом.

Master Terminal Unit (MTU), Master Station (MS) - диспетчерський пункт управління (головний термінал); здійснює обробку даних і управління високого рівня, як правило, у режимі м'якого (квазі-) реального часу. Одна з основних функцій – забезпечення людино-машинного інтерфейсу (між людиною- оператором і системою). Залежно від конкретної системи MTU може бути реалізований по-різному: від одиночного комп'ютера з додатковими пристроями підключення до каналів зв'язку до більших обчислювальних систем (мейнфреймів) і/або об'єднаних у локальну мережу робочих станцій і серверів. Як правило, і при побудові MTU використаються різні методи підвищення надійності й безпеки роботи системи. Пристрій MTU часто називають SCADA-сервером.

Communication System (CS) - комунікаційна система (канали зв'язку) між RTU і MTU. Вона необхідна для передачі даних з віддалених пунктів (RTU) на центральний інтерфейс диспетчера й передачі сигналів управління назад з MTU на RTU. В якості комунікаційної системи можна використати наступні канали передачі даних:

– виділені лінії - власні або орендовані; мідний кабель або оптоволокно;
– приватні радіомережі;
– аналогові телефонні лінії;
– цифрові *ISDN* мережі;
– стільникові мережі *GSM* (*GPRS*).

З метою дублювання ліній зв'язку пристрої можуть підключатися до декількох мереж, наприклад до виділеної лінії й резервного радіоканалу.

Особливості SCADA як процесу управління

Нижче перераховані деякі характерні риси процесу управління в сучасних диспетчерських системах:

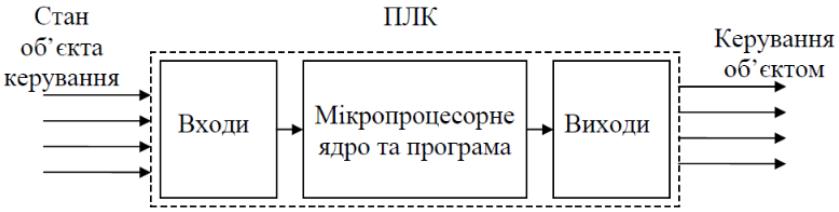
– у системах SCADA обов'язкова наявність людини (оператора, диспетчера);
– будь-який неправильний вплив може привести до відмови (втрати) об'єкта управління або навіть катастрофічних наслідків;
– диспетчер несе, як правило, загальну відповідальність за управління системою, що, при нормальних умовах, тільки зрідка вимагає підстроювання параметрів для досягнення оптимального функціонування;
– більшу частину часу диспетчер пасивно спостерігає за відображуваною інформацією; активна участь диспетчера в процесі управління відбувається нечасто, звичайно у випадку настання критичних подій - відмов, аварійних і позаштатних ситуацій та ін.;
– дії оператора в критичних ситуаціях можуть бути жорстко обмежені за часом (декількома хвилинами або навіть секундами).

РОЗДІЛ IV. ПРОГРАМУВАННЯ ПРОМИСЛОВИХ ЛОГІЧНИХ КОНТРОЛЕРІВ АСУ ТП

ЛЕКЦІЯ 27. КОМПОНЕНТИ ОРГАНІЗАЦІЇ ПРОГРАМ (POU)

27.1 Програмовані логічні контролери

Програмовані логічні контролери, скорочено ПЛК (PLC- Programmable Logic Controller) вперше з'являються у 1969р. для автоматизації автомобільної промисловості. Зараз ПЛК використовуються в енергетиці, хімічній промисловості, системах забезпечення безпеки, харчовому виробництві, машинобудуванні, транспорті тощо. Типовий ПЛК являє собою мікропроцесорний блок з деякою кількістю входів і виходів для підключення давачів та виконуючих механізмів. Логіка роботи описується програмно ідентичні ПЛК можуть виконувати зовсім різні задачі. Входи і виходи зазвичай роблять стандартними, тому при зміні алгоритму роботи не потрібно ніякої зміни апаратної частини.



Програмований контролер - це програмно керований дискретний автомат, що має певну кількість входів, підключених за допомогою давачів до об'єкта керування, і деяку кількість виходів, підключених до виконуючих пристроїв. ПЛК контролює стан входів і виробляє певні послідовності програмно заданих дій, що відображаються в зміні виходів.

Завданням прикладного програмування ПЛК є реалізація алгоритму керування конкретним об'єктом керування (машиною, агрегатом). Опитування входів і виходів контролер здійснює автоматично, незалежно від способу фізичного з'єднання. Цю роботу виконує системне програмне забезпечення. В ідеальному випадку прикладний програміст зовсім не цікавиться, як приєднані й де розмішені давачі й виконавчі механізми. Крім того, його робота не залежить від того, з яким контролером й якою фірмою він працює. Завдяки стандартизації мов програмування прикладна програма виявляється переносимою. Це означає, що її можна використати в будь-якому ПЛК, що підтримує даний стандарт.

ПЛК призначений для роботи в режимі реального часу в умовах промислового середовища і повинен бути доступним для програмування неспеціалістом в області інформатики.

З самого початку ПЛК призначалися для керування послідовними логічними процесами, що й обумовило слово «логічний» у назві ПЛК. Сучасні ПЛК крім простих логічних операцій здатні виконувати цифрову обробку сигналів, керування приводами, регулювання, функції операторського керування тощо. Конструкція ПЛК може бути найрізноманітнішою - від мініатюрних мікроПЛК до стояка заповненого апаратурою.

Задачі керування вимагають неперервного циклічного контролю. У будь- яких цифрових пристроях неперервність досягається за рахунок застосування дискретних алгоритмів, що повторюються через досить малі проміжки часу. Таким чином, обчислення в ПЛК завжди повторюються циклічно. Одна ітерація, що включає вимірювання і розрахунок впливу, називається робочим циклом ПЛК. Виконувані дії залежать від значення входів контролера, попереднього стану і визначаються користувацькою програмою.

При вмиканні живлення ПЛК виконує самотестування і настроювання апаратних ресурсів, очищення оперативної пам'яті даних (ОЗП), контроль цілісності прикладної програми

користувача. Якщо прикладна програма збережена в пам'яті, ПЛК переходить до основної роботи, що складається з постійного повторення послідовності дій, що входять до робочого циклу.

Робочий цикл ПЛК складається з декількох фаз:

1. Початок циклу.
2. Читання стану входів.
3. Виконання коду програми користувача.
4. Запис стану виходів.
5. Обслуговування апаратних ресурсів ПЛК.
6. Монітор системи виконання (системне програмне забезпечення ПЛК).
7. Контроль часу циклу.
8. Перехід на початок циклу.

27.2 Стандарт МЕК 61131

Стандарт МЕК 61131 було впроваджено у 1982 р з метою вирішення проблем уніфікації устаткування промислової автоматизації. Він охоплює вимоги до апаратних засобів, монтажу, тестування, документації, зв'язку і програмування ПЛК. Мови програмування відносяться до розділу МЕК 61131-3. Стандартом описується 5 мов програмування.

Текстові:

– Instruction List (IL) - список інструкцій;
– Structured Text (ST) - структурований текст.

Графічні:

– Sequential Function Chart (SFC) - послідовні функційні діаграми;
– Function Block Diagram (FBD) - функціонально-блокові діаграми;
– LadderDiagram (LD) - сходинкова діаграма, мова релейно-контактних схем.

Розглянемо кожну з цих мов детальніше.

Мова IL - список інструкцій - є типовим асемблером з акумулятором та переходами за позначками. Набір інструкцій стандартизовано, він не залежить від конкретної цільової платформи.Мова ILдозволяє працювати з будь-якими типами даних, викликати функції та функціональні блоки, реалізовані будь-якою мовою. Таким чином, на ILможна реалізувати алгоритм будь-якої складності, хоча текст буде досить громіздким.

Мова ST- структурований текст - це мова високого рівня. Синтаксично STявляє собою трохи адаптовану мову Паскаль. Замість процедур мови Паскаль в STвикористовуються компоненти програм стандарту МЕК. Для фахівців, знайомих з мовою С, освоєння STтакож не викличе ніяких складностей. У більшості комплексів програмування ПЛК мова STза замовчуванням пропонується для опису дій та умов переходів SFC. Це дійсно максимально потужний тандем, що дозволяє ефективно вирішувати будь-які задачи.

У сімействі мов МЕК SFC-діаграми стоять окремо, а точніше, вище відносно інших чотирьох мов. Діаграми SFCє високорівневим графічним інструментом. Графічна діаграма SFCскладається із кроків і переходів між ними. Дозвіл переходу визначається умовою. Із кроком пов'язані певні дії. Опис дії виконується будь-якою мовою МЕК. Сам SFCне містить керуючих команд ПЛК, дії необхідно описати на IL, ST, LDабо FBD.

Мова FBD - функціонально-блокові діаграми - нагадує принципову схему електронного пристрою на мікросхемах. Провідники в FBD можуть проводити сигнали (передавати змінні) будь-якого типу (логічний, аналоговий, час тощо). Виходи блоків можуть бути підімкнені на входи інших блоків або безпосередньо на виходи ПЛК. Самі блоки, подані на схемі як «чорні ящики» можуть виконувати будь-які функції. FBD-схеми дуже чітко описують взаємозв'язок входів і виходів діаграми. Якщо алгоритм добре описується з позиції сигналів, то його FBD-подання завжди виходить наочніше, ніж у текстових мовах.

Мова LD - сходинкова діаграма або мова релейно-контактних схем (РКС) - графічна мова, що реалізує структури електричних ланцюгів. Графічно LD- діаграма представлена у вигляді двох вертикальних шин живлення. Між ними розмішені ланцюги, утворені з'єднанням контактів. Навантаженням кожному ланцюгу служить реле. Кожне реле має контакти, які можна використати в інших ланцюгах. Послідовне (І), паралельне (АБО) з'єднання контактів та інверсія (НЕ)

утворюють базис Буля. У результаті LDідеально підходить не тільки для побудови релейних автоматів, але й для програмної реалізації комбінаційних логічних схем. Завдяки можливості включення в LDфункцій і функціональних блоків, виконаних іншими мовами, сфера застосування мови практично не обмежена.

Включення в стандарт п'яти мов пояснюється в першу чергу історичними причинами. Розроблювачі стандарту зіткнулися з наявністю величезної кількості різних варіацій схожих мов програмування ПЛК. Мови, що увійшли в стандарт, створені на основі найбільш популярних мов програмування,найпоширеніших у світі контролерів. Якщо взяти будь-який контролер, що працює в сучасному виробництві, то його програму можна перенести в середовище МЕК 61131-3 із мінімальними затратами.

27.3 Компоненти організації програм

Компоненти створюють код прикладного програмного забезпечення ПЛК. Саме на цьому рівні компонентів доступне сумісництво різник мов МЕК. В англомовних документах компоненти організації программ скорочено називаються POU – Program Organization Unit.

Компоненти організації програм є базовими елементами, з яких будується код проекту, аналогічним чином електронні пристрої складаються зазвичай з модулів. Наприклад, магнітофон містить: підсилювачі запису і програвання ведення, генератор підмагнічування і стирання, блок живлення і т. д. Кожен компонент програми має власну назву, певний інтерфейс і опис на одній з МЕК-мов. Один компонент може викликати інші компоненти. Виклик самого себе (рекурсія) в стандарті МЕК не дозволена. Комбінувати різні мови в одному проекті можна при описі різних компонентів, але окремий компонент повністю реалізується на одній мові МЕК. При виклиці компонента мова його реалізації значення не має.

До компонентів організації програм в МЕК-стандарті відносять функції, функціональні блоки і програми. Всі вони схожі, але мають певні особливості і різне призначення.

Компонент має властивість інкапсуляції - працює як «Чорний ящик», приховуючи деталі реалізації. Для роботи з компонентом досить знати його інтерфейс, що включає опис входів і виходів. Внутрішнє його пристрій знати необов'язково. У графічній формі представлення компонент виглядає як прямокутник з входами зліва і виходами справа. Локальні (Внутрішні) змінні компонента недоступні ззовні і в графічному поданні не відображаються.

Готовий компонент завжди можна розкрити, вивчити і поправити. Це, звичайно, відноситься тільки до призначених для користувача компонентів і відкритих бібліотек. Деякі стандартні компоненти включені в транслятор і не доступні для перегляду і змін.

27.4 Оголошення POU

Реалізації будь-якого POU завжди повинен передувати розділ оголошень. Оголошення функції, функціонального блоку і програми починається відповідно з ключових слів FUNCTION, FUNCTION_BLOCK і PROGRAM. За ним йде ідентифікатор (Ім'я компонента). Далі визначається інтерфейс POU. До інтерфейсу компонента відносяться входи VAR_INPUT, виходи VAR_OUTPUT і змінні типу вхід-вихід VAR_IN_OUT, завершають розділ оголошень локальні змінні VAR.

У функціях розділи VAR__OUTPUT і VAR_IN_OUT відсутні. Виходом функції служить єдина змінна, що співпадає з ім'ям функції. Тип значення, що повертається, вказаний при визначенні ідентифікатора через двокрапку.

Наприклад: FUNCTION iNearby: INT

Структура розділу оголошень POU наведена в таблиці

Тип POU	Функція	Функціональний блок	Програма
	FUNCTION Ім’я: ТИП	FUNCTION_BLOCK Ім’я	PROGRAM Ім’я
Інтерфейс	VAR_INPUT	VAR_INPUT	VAR_INPUT
		VAR_OUTPUT	VAR_OUTPUT
		VAR_IN_OUT	VAR_IN_OUT
Локальні змінні	VAR	VAR	VAR

Всі розділи змінних є не обов'язковими. Інтерфейс не потрібен, розділ оголошень буде містити тільки локальні змінні VAR.

Інтерфейс компонента утворюється вхідними та вихідними змінними. Інтерфейсні вхідні змінні називають формальними параметрами. При використанні компонента його формальні параметри зв'язуються з актуальними параметрами. І нарешті, при виклику параметри компонента набувають актуальні або поточні значення. Ці поняття необхідні для уникнення двозначності при описі техніки роботи з компонентами.

Пояснимо їх відмінності на прикладі. Візьмемо стандартний блок R_TRIG. Він має вхід з назвою CLK. Ми будемо використовувати його в програмі, в якій визначена якась змінна, наприклад bPulse. При виклику блоку з нашої програми ми подаємо bPulse на вхід CLK. Далі програма компілюється і завантажується в контроллер. Змінна bPulse набуває деяке значення, наприклад TRUE. Вхід CLK, природно, теж матиме значення TRUE. CLK - це формальний параметр, bPulse – актуальний параметр, а TRUE - фактичне значення. З формальними параметрами доводиться мати справу при проектуванні POU і описанні його інтерфейсу. Актуальні параметри працюють при використанні користувацького компонента. Поточні значення народжуються тільки в «Залізі» в процесі виконання.

При оголошенні POU можна зустріти такі заголовки:

Формальні вхідні параметри VAR_ _INPUT

Передаються POU за значенням шляхом копіювання, при виклику блоку такої змінної можна привласнити значення іншої змінної (сумісного типу), константи або виразу. Будь-які зміни такої змінної всередині POU ніяк не відображаються на дані компонента. Застосовується в будь-яких POU. Можуть мати значення за замовчуванням. Відображаються в графічному поданні з лівого боку компонента.

Формальні вихідні параметри VAR_OUTPUT відображають результати роботи компонента. Передаються з POU за значенням шляхом копіювання. Читання значення виходів звичайні, але має сенс після виконання блоку. Поза компонента параметри VAR_OUTPUT доступні тільки для читання. Не викоистовуються в функціях, оскільки функція має тільки одне значення, що повертається. Можуть мати початкові значення. Відображаються в графічному поданні справа.

Параметр типу VAR_IN_OUT одночасно є входом і виходом. Передача змінної екземляра блоку виконується за посиланням. Це означає, що зовнішня змінна як би сама працює всередині блоку на правах внутрішньої змінної. В компонент передається тільки адреса її розташування в пам'яті даних. Для змінної VAR_IN_OUT не можна:

- використовувати її в функціях;
- привласнювати початкове значення;
- звертатися як до елементу структури даних, через точку;
- привласнювати константу, як актуальний параметр.

Присвоєння зовнішньої змінної для VAR_IN_OUT можна здійснювати тільки при виклику блоку.

Найважливішою властивістю VAR_IN_OUT є відсутність копіювання зовнішніх даних. Параметри VAR INPUT і VAR_OUTPUT можуть оперувати з масивами і структурами, але всякий раз при зверненні до компоненту відбуватиметься повне копіювання даних. Це може забирати багато часу. Присвоєння одного масиву іншому для VAR_IN_OUT означає фактично перемикання компонента з одного масиву на інший. Локальна копія даних в цьому випадку не створюється.

Як і глобальні змінні, параметри VAR_IN_OUT порушує ідеологію незалежності компонентів. Правильний компонент не повинен мати можливості зіпсувати чужу пам'ять. Тому застосовувати їх потрібно дуже акуратно і тільки в випадках, коли це дійсно необхідно.

Локальні змінні VAR доступні тільки всередині компонента, поза компонента доступу немає. Можуть мати початкові значення. Для функцій локальні змінні розміщуються в динамічній пам'яті. Після закінчення роботи функції пам'ять звільняється і може використовуватися в інших функціях. У програмах і в функціональних блоків змінні VAR зберігають свої значення між викликами програм. У графічному поданні компонента локальні змінні не відображаються.

ЛЕКЦІЯ 28. СПЕЦІАЛІЗОВАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ЩО ПРИЗНАЧЕНЕ ДЛЯ ПІДГОТОВКИ КОРИСТУВАЦЬКИХ ПРОГРАМ ПРОГРАМОВАНОГО ЛОГІЧНОГО КОНТРОЛЕРА(CODESYS). ОСНОВНІ МОВИ ПРОГРАМУВАННЯ КОНТРОЛЕРІВ PLC

28.1 Комплекс CoDeSys

CoDeSys - це сучасний інструмент для програмування контролерів (CoDeSysутворюється від слів Controllers Development System). CoDeSys надає програмісту зручне середовище для програмування контролерів мовами стандарту МЕК 61131-3.

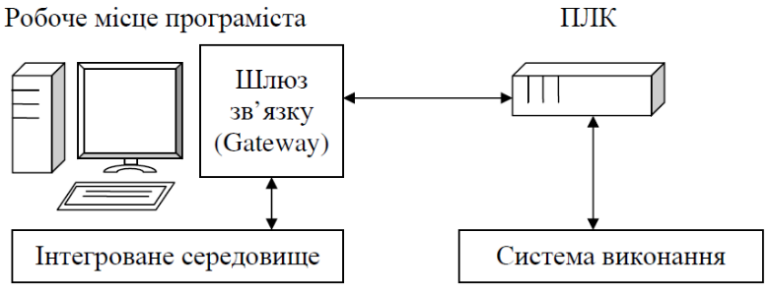
Комплекс CoDeSys не прив'язаний до конкретної апаратної платформи, існує кілька модифікацій спеціально оптимізованих під різні мікропроцесори. Для прив'язки до конкретного ПЛК потрібна адаптація програми до низкорівневих ресурсів - розподілу пам'яті, інтерфейсів зв'язку та інтерфейсів уводу-виводу.

Серед особливостей середовища CoDeSys можна відзначити такі:

– пряма генерація машинного коду, що забезпечує швидкодію програм користувача;
– повноцінна реалізація мов МЕК;
– «розумні» редактори мов, що не дають робити типові для початківців помилки;
– вбудований емулятор контролера, що дозволяє проводити налагодження проекту без додаткових апаратних засобів;
– вбудовані елементи візуалізації дають можливість створити модель об'єкту керування та проводити налагодження проекту без виготовлення засобів імітації;
– набір бібліотек і готових сервісних функцій.

Базовий склад комплексу програмування ПЛК складається із двох обов'язкових частин: системи виконання й робочого місця програміста. Система виконання функціонує в контролері. Крім безпосереднього виконання керуючої програми вона забезпечує завантаження коду прикладної програми і функції її налагодження. Система виконання повинна мати зв'язок з комп'ютером робочого місця програміста. Не важливо як фізично організований зв'язок ПК і ПЛК, у найпростішому випадку ПЛК підключається до комп'ютера через стандартний COM-порт (RS-232) нуль-модемним кабелем. В умовах цеху може використовуватися більш завадостійкий і далекобійний інтерфейс (RS-422, RS-485 або струмова петля).

У комплексі CoDeSys посередником між середовищем розробки та ПЛК слугує спеціальний додаток - шлюз зв'язку (gateway). Шлюз зв'язку взаємодіє з інтегрованим середовищем через Windows-сокет з'єднання, побудоване на основі протоколу TCP/IP. Таке з'єднання забезпечує однакову взаємодію додатків, що працюють на одному комп'ютері або в мережі. Завдяки цьому програміст може абсолютно повноцінно працювати на віддаленому комп'ютері. Причому віддаленість не обмежується рамками локальної мережі. ПК, що виконує роль шлюзу зв'язку, може одночасно взаємодіяти із ПК програміста через Інтернет і із ПЛК через модемне з'єднання.



За замовчуванням шлюз зв'язку настроєний на локальну роботу й запускається автоматично при встановленні зв'язку з ПЛК із інтегрованого середовища. Для з'єднання з ПЛК через COM-порт достатньо настроїти параметри драйвера інтерфейсу відповідно до керівництва по застосуванню ПЛК (порт, швидкість, контроль паритету та число стоп-бітів).

До складу будь-якого комплексу обов'язково входить посібник із застосування та електронна довідкова система. Асортименти додаткових додатків CoDeSys включають сервери даних (DDEй OPC), утиліти конфігурування комплексу, засоби керування проектами і версіями, текстові інструменти, спеціалізовані бібліотеки функцій та функціональних блоків.

28.2 Організація роботи у середовищі розробки CoDeSys

Програми у середовищі розробки CoDeSyszберігаються у виді проектів. Життєвий цикл проекту:

– визначення конфігурації ПЛК відповідно до апаратних засобів;
– створення програмних компонентів, необхідних для вирішення проблеми;
– написання програмного коду для створених компонентів;
– компіляція проекту, виправлення помилок;
– налагодження проекту у режимі симуляції;
– запис результатів компіляції і налагодження у ПЛК.

Проект містить ряд різнорідних об'єктів: програмні компоненти (POU – Program Organization Unit), дані різних типів, елементи візуалізації й ресурси. Кожний проект зберігається в окремому файлі.

До програмних компонентів (POU) відносяться функціональні блоки, функції й програми. Окремі POU можуть включати дії (підпрограми). Перший програмний компонент родміщується в новому проекті автоматично і дістає назву PLC_PRG. Саме з нього і починається виконання процесу (за аналогією з функцією main у мові С), з нього будуть викликатися інші програмні блоки (програми, функції й функціональні блоки). POU можуть викликати інші POU, але рекурсії неприпустимі.

Кожен програмний компонент складається з розділу оголошень і коду.

Для написання всього коду POU використовується тільки одна з мов МЕК програмування (IL, ST, FBD, SFC, LD або CFC).

CoDeSys підтримує всі описані стандартом МЕК компоненти. Для їхнього використання досить включити у свій проект бібліотеку standard.lib.

Головне вікно середовища розробки CoDeSys складається з таких елементів:

Меню.

*Панель інструмент*ів. На ній знаходяться кнопки для швидкого виклику команд меню.

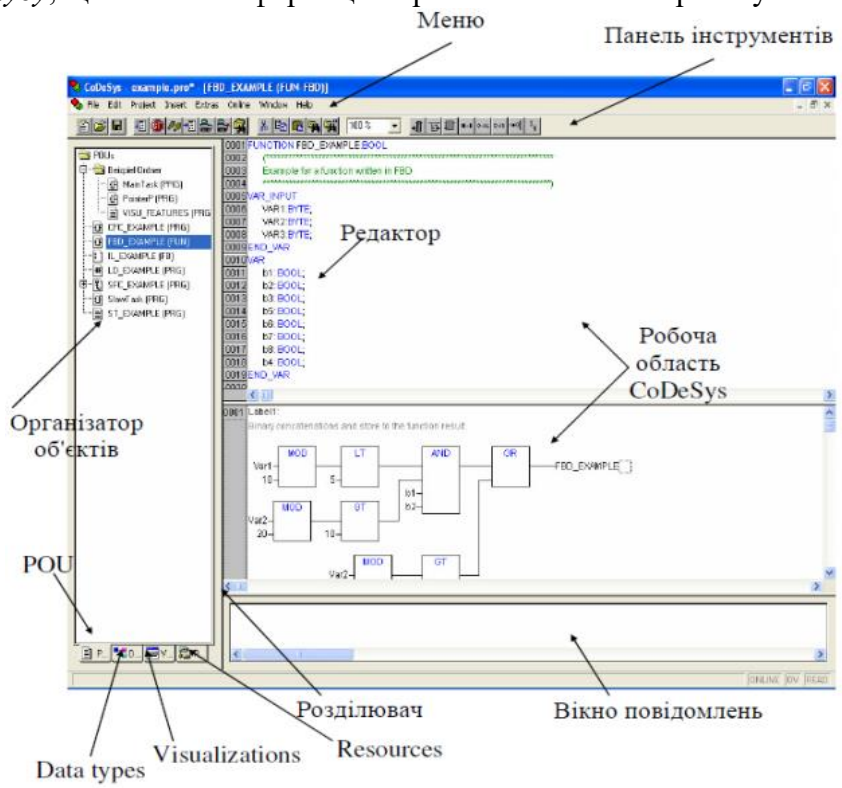
Організатор об'єктів, що має вкладки POU, Datatypes, Visualizationsi Resources.

Розділювач Організатора об'єктів і робочої області CoDeSys.

Робоча область, в якій знаходиться редактор.

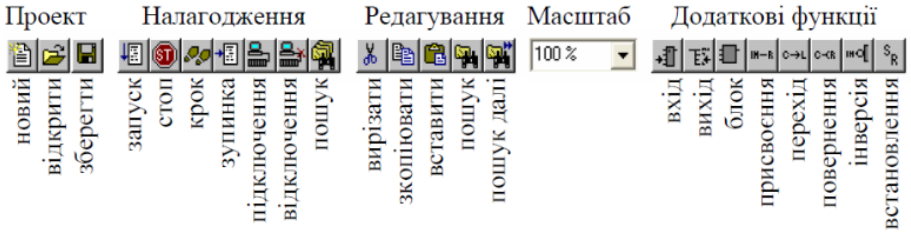
Вікно повідомлень.

Рядок статусу, що містить інформацію про поточний стан проекту.



Меню знаходиться у верхній частині Головного вікна. Воно містить всі команди CoDeSys.

Кнопки на Панелі інструментів забезпечують більш швидкий доступ до команд Меню. Викликана за допомогою кнопки на панелі інструментів команда автоматично виконується в активному вікні.



Кнопки на Панелі інструментів різні для різних редакторів CoDeSys. Одержати інформацію щодо призначення цих кнопок можна в описі редакторів.

Організатор об'єктів (Object Organizer) керує списком усіх об'єктів проекту. Він завжди знаходиться в лівій частині Головного вікна CoDeSys. У нижній частині організатора об'єктів знаходяться вкладки POUs, Datatypes (типи даних), Visualizations(візуалізації) і Resources (ресурси). Переключатися між об'єктами можна за допомогою миші або клавіш переміщення.

Розділювач екрана - це межа між двома непересічними вікнами. В CoDeSysє такі розділювачі: між організатором об'єктів і робочою областю, між розділом оголошень і розділом коду POU, між робочою областю й вікном повідомлень. Ви можете переміщати роздільники за допомогою миші, натиснувши її ліву кнопку.

Розділювач зберігає своє положення навіть при зміні розмірів вікна. Якщо Ви більше не бачите розділювича на екрані, це, означає, що потрібно змінити розміри вікна.

Робоча область знаходиться в правій частині головного вікна CoDeSys. Всі редактори, а також менеджер бібліотек відкриваються саме в цій області. Ім'я відкритого об'єкта знаходиться в заголовку вікна.

Вікно повідомлень відділене від робочої області розділювачем. Саме в цьому вікні з'являються повідомлення компілятора, результати пошуку й список перехресних посилань.

При подвійному клацанні мишею або при натисканні клавіші <Enter> на повідомленні буде відкритий об'єкт, до якого стосується дане повідомлення.

За допомогою команд “Edit->Next error” і ”Edit->Previous error” можна швидко переміщатися між повідомленнями про помилки.

Вікно повідомлень можна прибрати або включити за допомогою команди “Window Message”.

Статусний рядок знаходиться в нижній частині головного вікна CoDeSys і надає інформацію про проект і команди меню.

При виборі пункту меню його опис з'являється в лівій частині рядка статусу.

Якщо Ви працюєте в режимі online, то напис Online у рядку статусу виділяється чорними кольорами. В іншому випадку напис сірий.

За допомогою статусного рядка в режимі online можна визначити, у якому стані перебуває програма: SIM - у режимі емуляції, RUN- програма запущена, BP- установлена точка зупинки, FORCE- відбувається фіксація змінних.

При роботі в текстовому редакторі в рядку статусу вказується позиція, у якій знаходиться курсор (наприклад, Line:5, Col.:11). У режимі заміни напис “OV” виділяється чорними кольорами. Натискаючи клавішу <Ins>можна перемикатися між режимом вставлення і заміни.

При візуалізації в статусному рядку виводяться координати курсора X та Y, які відраховуються щодо верхнього лівого кута вікна. Якщо курсор миші знаходиться на елементі або над елементом і проводяться будь-які дії, тоді вказується номер цього елемента. При вставленні елемента в рядку статусу вказується його назва (наприклад, Rectangle).

Якщо помістити курсор на пункт меню, то в рядку статусу з'являється його короткий опис.

Замість того, щоб використовувати головне меню для виклику команд, можна скористатися контекстним меню. Це меню, викликуване правою кнопкою миші, містить найбільш часто використовувані команди.

28.3 Основні мови програмування контролерів PLC

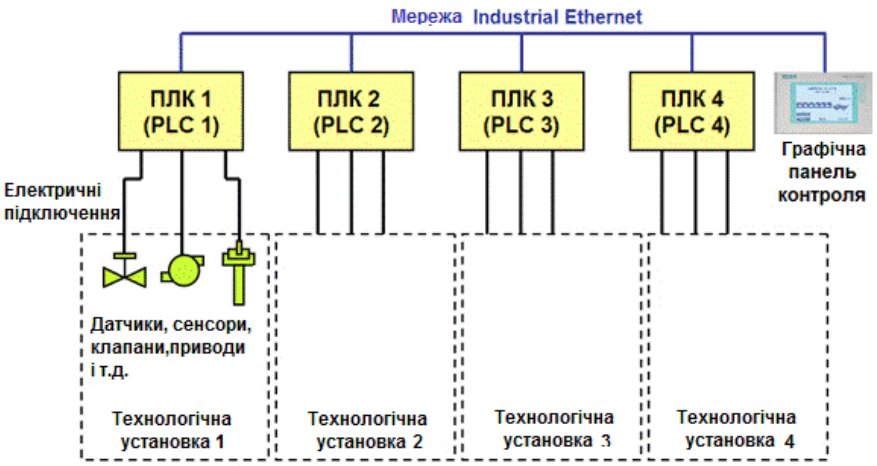
Основним компонентом системи є програмований логічний контролер - PLC. Системи класу PLC використовують для управління послідовністю технологічних операцій у процесі виготовлення різних виробів (не продукту, а саме виробу). Типовим прикладом застосування систем PLC є управління формувальною машиною для виготовлення склотари або, наприклад, управління апаратом по наклейці алюмінієвих кришок на пластикові стаканчики з йогуртом. Типові завдання систем PLC:

- управління конвеєрними виробництвами;
- управління робототехнікою;
- високошвидкісне управління приводами,
- управління позиціонуючими пристроями;
- сигналізація, оповіщення;
- управління комплектними технологічними машинами.

Для систем PLC характерно те, що вони не вимагають безперервного контролю з боку диспетчера (на відміну від SCADA і DCS), досить періодичної перевірки статусу. Рівень диспетчерського (операторського) управління розвинений слабко й зводиться, як правило, до установки кнопкового пульта управління для запуску/зупинки тієї або іншої технологічної ділянки й відображення аварійних сигналізацій. Більшу частину часу система PLC працює без нагляду з боку людини, тобто в автоматичному (автономному) режимі.

Структура системи PLC досить проста. Один або кілька програмувальних логічних контролерів, об'єднаних у мережу за допомогою цифрової шини. Обмінюючись по шині даними, контролери можуть взаємодіяти один з одним, що необхідно для їхньої погодженої роботи. Як уже було згадано, при необхідності до системи також можна підключити пульт локального управління (кнопковий або із рідкокристалічною панеллю - РК-панеллю).

На рисунку зображена типова структура системи PLC. Чотири програмувальних логічних контролери об'єднані з єдину мережу (у цьому випадку стандарту Industrial Ethernet). До мережі також підключена РК-панель для найпростішого локального управління й відображення аварійних сигналізацій.



Як видно, система структурована так, що кожна технологічна установка (машина, автомат) управляється своїм контролером. Така технологічна розбивка характерна для даного класу систем.

Як правило, у контролерів є електричні входи/виходи для підключення до них мережних датчиків, сенсорів, виконавчих механізмів (клапанів, позиціонуючих пристроїв, різних приводів), пристроїв оповіщення й сигналізації. Кількість входів/виходів може бути як фіксованою, так і розширюваною за допомогою модулів, що підключають додатково. Такі модулі називаються «модулями вводу/виводу» (I/O modules). Контролер безупинно виконує закладену в нього програму управління по наступному циклі: зчитування сигналів з датчиків, математична обробка даних у відповідності з певним алгоритмом, формування керуючої дії та її передача на виконавчі механізми. При цьому потрібна висока швидкодія – час виконання всього циклу звичайно не більше 10-20 мс.

ЛЕКЦІЯ 29. МОВА СТРУКТУРОВАНОГО ТЕКСТУ ST

29.1 Основні поняття

Мова ST є текстовою мовою викого рівня і дуже схожа за своєю структурою на Паскаль:
Приклад

```
IF Voltage>220 THEN
  Current:=Current - 10;  (*Якщо V>220 В, то зменшити струм 10*)
ELSE
  Current:=50; Speed:= ON;(*Встановити струм 50А і вкоючити двигун *)
END_IF;
```

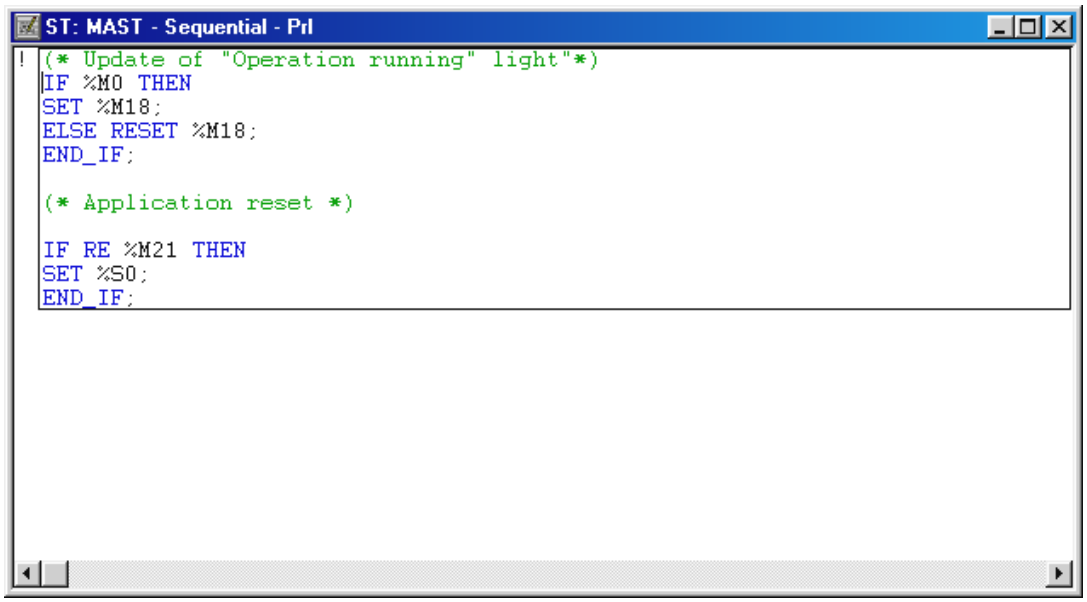
Мова ST розроблена спеціально для програмування ПЛК. Вона містить безліч конструкцій для присвоєння значень змінних, для виклику функцій і функціональних блоків, для написання виразів умовних переходів, вибору операторів, для побудови ітераційних процесів. Мова призначена в основному для виконання складних математичних обчислень, опису складних функцій, функціональних блоків і программ.

Мова Structured Text (ST) використовується для створення програм, написаних із програмними рядками, створених з буквено-цифрових знаків.

Твердження ST складає основну частину мови ST, серія тверджень використовується для визначення програми.

Головні команди в мові Structured Text такі:

- розрядні команди,
- арифметичні і логічні команди на слова і подвійні слова,
- арифметичні команди на числа з плаваючою точкою,
- чисельне порівняння на слова, подвійні слова і числа з плаваючою точкою,
- чисельні конверсії,
- команди на біт, слово, подвійне слово і таблиці чисел з плаваючою точкою,
- команди салінгових символів,
- алфавітно-цифровий компаратор,
- команди адміністрації часу,
- програмні команди,
- команди керування,
- команди стандартного функціонального блока,
- точні команди обміну,
- специфічні для програми команди (зв'язок, ПІД-керування, і т.д).



29.2 Команди мови ST

Розрядні команди:

Назва	Функція
:=	Розрядне присвоєння
OR	Логічний OR
AND	Логічний AND
XOR	Ексклюзивне логічне OR
NOT	Інверсія
RE	Передній фронт імпульсу
FE	Задній фронт імпульсу
SET	Установка в 1
RESET	Установка в 0

Чисельне порівняння для слів, подвійних слів і чисел з плаваючою точкою:<, >, =, <=, >=, <>.

Розрядові таблиці:

Назва	Функція
Table := Table	Присвоєння між двома таблицями
Table := Word	Присвоєння слова таблиці
Word := Table	Присвоєння таблиці слову
Table := Double word	Присвоєння подвійного слова таблиці
Double word := Table	Присвоєння таблиці до подвійному слову
COPY_BIT	Копіювання розрядної таблиці в розрядну таблицю
AND_ARX	AND між двома таблицями
OR_ARX	OR між двома таблицями
XOR_ARX	Ексклюзивне OR між двома таблицями
NOT_ARX	Заперечення таблиці
BIT_W	Копіювання розрядної таблиці в таблицю слова
BIT_D	Копіювання розрядної таблиці в таблицю подвійного слова
W_BIT	Копіювання таблиці слова в розрядну таблицю
D_BIT	Копіювання таблиці подвійного слова в розрядну таблицю

Цілочисельна арифметика для слів і подвійних слів:

Назва	Функція
+	Додавання
-	Віднімання
*	Множення
/	Цілочисельне ділення
REM	Залишкове значення від цілочисельного ділення
SQRT	Цілочисельний квадратний корінь
ABS	Абсолютне значення
INC	Збільшення
DEC	Зменшення

Арифметика для чисел з плаваючою точкою: +, -, *, /, SQRT (квадратний корінь), ABS (абсолютне значення).

Логічні команди для слів і подвійних слів:

Назва	Функція
AND	Логічний AND
OR	Логічне OR
XOR	Ексклюзивне логічне OR
NOT	Логічне доповнення
SHL	Логічний зсув ліворуч
SHR	Логічний зсув праворуч

Команди перетворення чисел:

Назва	Функція
INT_TO_REAL	Ціле число - Число з плаваючою точкою
DINT_TO_REAL	32-розрядне ціле число - Число з плаваючою точкою
REAL_TO_INT	Число з плаваючою точкою - Ціле число
REAL_TO_DINT	Число з плаваючою точкою - 32-розрядне ціле число

Команди для таблиць слів і подвійних слів:

Назва	Функція
Table := Table	Присвоєння між двома таблицями
Table := Word	Ініціалізація таблиці
+, -, *, /, REM	Арифметичні операції між таблицями
+, -, *, /, REM	Арифметичні операції між виразами і таблицями
EQUAL	Порівняння двох таблиць
SUM	Логічне доповнення таблиці
NOT	Інверсія таблиці
FIND_EQW, FIND_EQD	Знаходження першого елемента, рівного значенню
FIND_GTW, FIND_GTD	Знаходження першого елемента, більшого ніж значення
FIND_LTW, FIND_LTD	Знаходження першого елемента, меншого ніж значення
MAX_ARW, MAX_ARD	Знаходження максимального значення в таблиці
SORT_ARW, SORT_ARD	Сортування таблиці в порядку зростання чи спадання

Команди для таблиці чисел з плаваючою точкою:

Назва	Функція
Table := Table	Присвоєння між двома таблицями
Table := Floating point	Ініціалізація таблиці

Команди для символьних рядків:

Назва	Функція
STRING_TO_INT	ASCII - Ціле число
STRING_TO_DINT	ASCII - Подвійне ціле число
INT_TO_STRING	Ціле число - ASCII
DINT_TO_STRING	Подвійне ціле число - ASCII
FIND	Позиція підрядка у рядку символів
EQUAL_STR	Позиція першого різного символу
LEN	Довжина рядка символів
MID	Витяг підрядка із рядка символів
INSERT	Вставка підрядка в рядок символів
DELETE	Видалення підрядка з рядка символів
CONCAT	Зв'язати два рядки

Програмні команди:

Назва	Функція
HALT	Виконання зупинки програми
JUMP	Перехід до мітки
Sri	Запит підпрограми
RETURN	Повернення з підпрограми

Команди адміністрації часу:

Назва	Функція
RRTC	Читає дату системи
WRTC	Змінює дату системи
PTC	Читає дату і зупиняє код
ADD_TOD	Додає інтервал часу до часу дня
ADD_DT	Додає інтервал часу до дати і часу
DELTA_TOD	Різниця між вказаними годинами
DELTA_DT	Різниця між вказаними датами (із врахуванням годин)
DAY_OF_WEEK	Читає поточний день тижня
TRANS_TIME	Перетворює тривалість часу до дати
DATE_TO_STRING	Перетворює дату до рядка символів
TOD_TO_STRING	Перетворює час до рядка символів
DT_TO_STRING	Перетворює завершену дату до рядка символів
TIME_TO_STRING	Перетворює тривалість до рядка символів

29.3 Структура програми

Програма Structured Text організована у твердженнях. Кожні твердження ST складаються з таких елементів:

- мітка,
- коментарі,
- команди.

Кожний з цих елементів необов'язковий, тобто можливе існування порожнього твердження, твердження, що складається тільки з коментарів чи складається тільки з мітки. Кожне твердження починається з мітки виклику, яка генерується автоматично.

Приклад:

```
!%L2:  (* Твердження з міткою, коментарі *) SET %M0; %MW4 :=%MW2 + %MW9;
(* і кілька команд *)
%MF12 := SQRT (%MF14);
```

Коментар обмежений з обох кінців символами (* і *), він може бути поміщений в будь-якому місці у твердженні. Немає ніякого обмеження на число коментарів у твердженні. Він полегшує тлумачення твердження, для якого призначений, але є необов'язковим.

- Будь-які символи можна використовувати в коментарі.
- Число символів обмежене 256 на коментар.
- Вкладені коментарі не дозволяються.
- Коментар може бути кілька рядків у довжину.

Коментарі зберігаються в ПЛК і користувач може звертатися до них у будь-який момент. Через це вони споживають пам'ять програми.

Мітка використовується для виклику твердження в об'єкті програми (основна програма, підпрограма, і т.д) але є необов'язковою.

Мітка має такий синтаксис: %Li, де i=0...999 і зафіксована на початку твердження. Виклик мітки може бути здійснений тільки на одиночне твердження в межах того ж самого об'єкта програми (SR, головна програма, програмний модуль).

Мітки можуть бути розташовані в будь-якому порядку, залежно від того, як твердження введені і прийняті системою протягом сканування.

29.4 Структури керування в програмі

Програма складається з команд. Твердження ST може містити кілька команд. Кожна команда повинна закінчуватися знаком “;”.

Існує чотири структури керування:

– умовна дія IF,

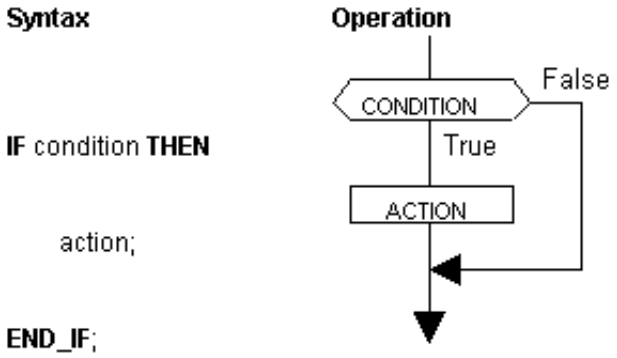
– умовні повторювані дії WHILE і REPEAT,

– повторювана дія FOR.

Кожна структура керування включена між ключовими словами, і вона починається і закінчується в тому самому твердженні. Можна вкласти структури керування одна в одну, незалежно від їхнього типу.

Умовна дія IF... END_IF;

Проста форма – команда виконує дію, якщо умова правдива (рисунок).

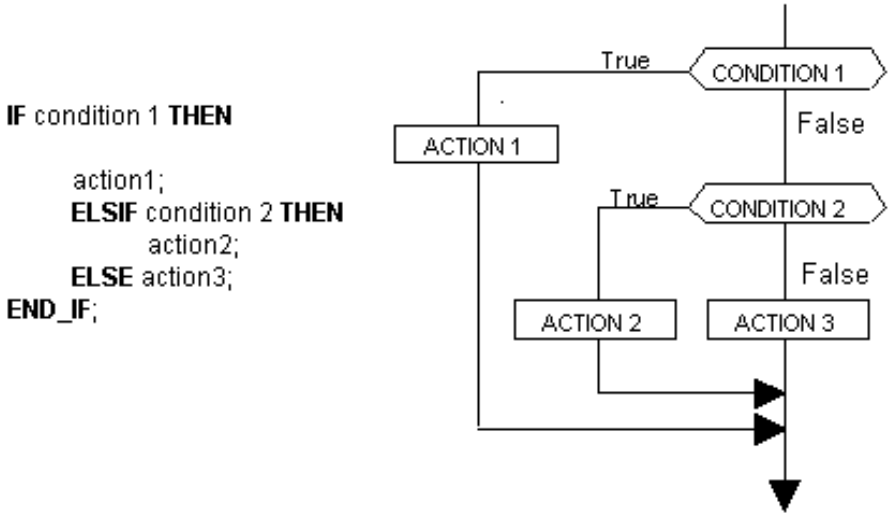


Приклад простої форми умовної дії IF... END_IF; :

```
IF %M0 AND %M12 THEN
    RESET %M0;
    INC %MW4;
    %MW10:=%MW8+%MW9;
END_IF;
```

Основна форма:

- Умови можуть бути множинними.
- Кожна дія представлена як список команд.
- Кілька "IF" структур керування можуть бути вкладеними.
- Немає ніякого обмеження на число команд ELSIF.
- Є максимум одна частина ELSE.



Приклад простої форми умовної дії IF... END_IF; :

```
IF %M0 AND %M1 THEN
    %M5:=%M3+%M4;
    SET %M10;
    ELSIF %M0 OR %M1 THEN
        %MW5:=%MW3-%MW4;
        SET%MW11;
    ELSE
        RESET %M10;
        RESET %M11;
END_IF;
```

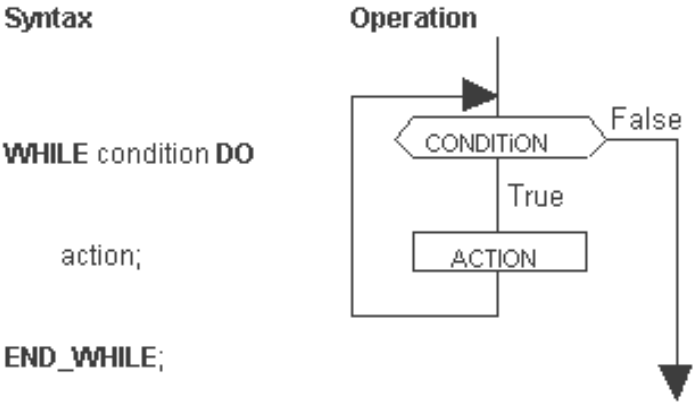
Умовна повторювана дія WHILE ... END_WHILE;

Команда виконує періодично повторювану дію, поки умова виконується.

Приклад умовної повторюваної дії WHILE ... END_WHILE; :

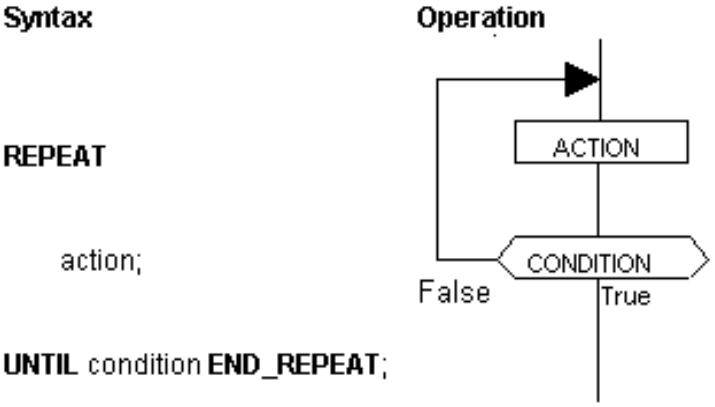
```
WHILE %MW4<12 DO
    INC %MW4;
    SET %MW25[%MW4];
END_WHILE;
```

- Умова може бути множинною.
- Дія представляє собою список команд.
- Умова випробовується перед виконанням дії. Дія виконується, поки умова істинна.
- Можуть бути вкладеними кілька структур керування WHILE.



Умовна дія, що триває REPEAT…END_REPEAT;

Команда виконує повторювану дію до перевіреної умови.



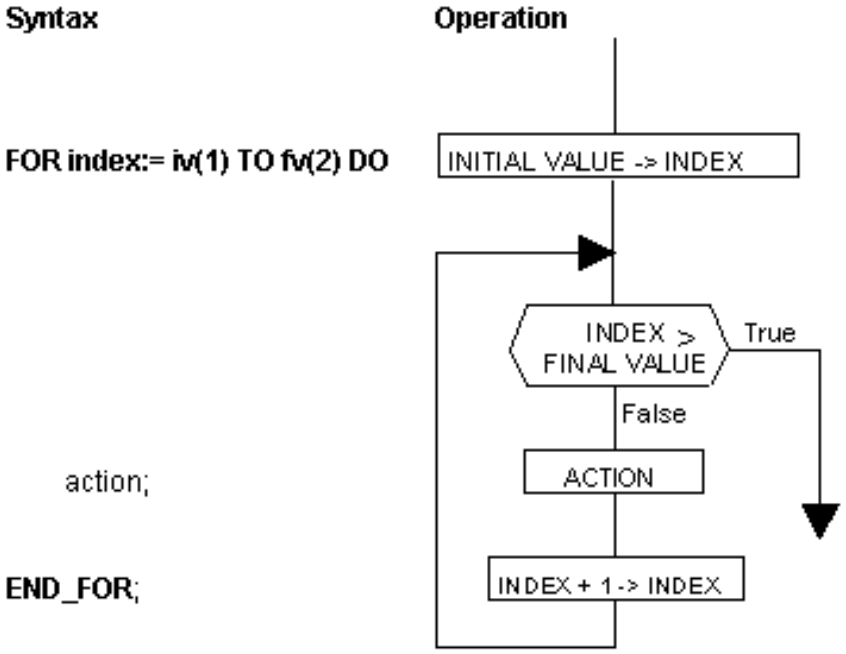
Приклад:

```
REPEAT
    INC %MW4.
    SET %MW25[%MW4];
UNTIL %MW4>12 END_REPEAT;
```

Повторювана дія FOR ... END_FOR;

Команда виконує операцію з обробки кілька разів, збільшуючи індекс на 1 в кожному циклі.

– Коли індекс більший ніж остаточне значення, виконання продовжується на наступній команді за ключовим словом END_FOR.
– Індекс збільшується автоматично.
– Дія представляє собою список команд.
– Початкове значення і остаточне значення повинні бути чисельним виразом типу слова.
– Індекс повинен бути об'єктом типу слова, який є доступним у режимі читання.
– Можуть бути вкладеними кілька структур керування FOR.



Приклад:

```
FOR %MW4:=0 TO %MW23+12 DO
   SET %M25[%MW4];
END_FOR;
```

29.5 Правила для виконання програм Structured Text

Програма на мові Structured Text виконується послідовно, команда за командою з врахуванням структур керування.

У випадку арифметичних чи логічних виразів, що складаються з кількох операторів, визначаються правила пріоритету між різними операторами.

Коли є конфлікт між двома операторами того ж самого пріоритету, перший оператор буде мати вищий пріоритет (оцінка виконується зліва направо).

Приклад:

%MW34 * 2 REM 6

У цьому прикладі %MW34 спочатку збільшується на 2, тоді результат використовується для знаходження модуля.

Круглі дужки використовуються для зміни порядку, у якому оператори оцінюються, наприклад, давати додаванню більш високий пріоритет, ніж множенню.

Приклад:

(%MW10 + %MW11) * %MW12

У цьому прикладі, додавання буде виконуватися перед множенням.

Круглі дужки можуть бути вкладеними; немає ніякого обмеження рівням вкладення. Круглі дужки можуть також використовуватися для запобігання неправильного тлумачення програми.

Приклад:

NOT %MW2 <> %MW4 + %MW6

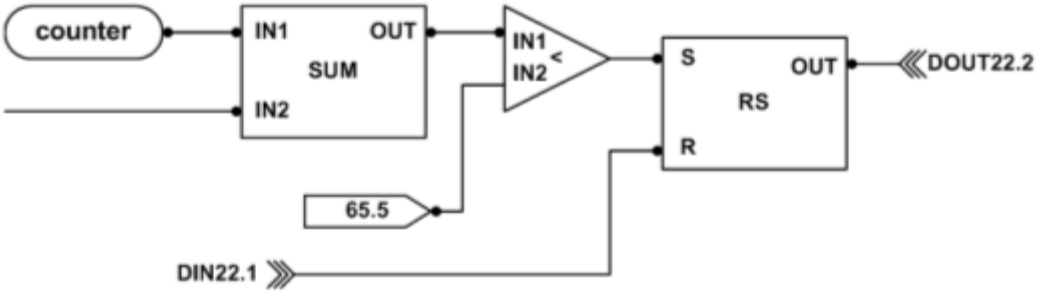
ЛЕКЦІЯ 30. МОВА ДІАГРАМ ФУНКЦІОНАЛЬНИХ БЛОКІВ (FBD). МОВА РЕЛЕЙНИХ ДІАГРАМ (LD)

30.1 Мова функціональних блокових діаграм

Мова функціональних блокових діаграм FBD (Function Block Diagrams) дозволяє створювати програмну одиницю практично будь-якої складності на основі стандартних «цеглинок» (арифметичні, тригонометричні, логічні блоки, PID-регулятори, блоки, що описують деякі закони управління, мультиплексори і т.д.). Цей мовний засіб використовує технологію інкапсуляції алгоритмів обробки даних та законів регулювання. Все програмування зводиться до «склеювання» готових компонентів. У результаті виходить максимально наочна і добре контрольована програмна одиниця. Функціональні блокові діаграми дозволяють візуально визначати як логічні, так і аналогові вирази.

FBD є графічною мовою і застосовується для побудови комплексних процедур, що складаються з різних функціональних бібліотечних блоків - арифметичних, тригонометричних, регуляторів, мультиплексорів і т.д. Він підходить також для управління безперервними процесами і процесами регулювання. При цьому здійснюється уявлення функцій за допомогою блоків, пов'язаних між собою. З'єднання між виходами функціональних блоків в явному вигляді можуть бути відсутніми, а вихід блоку може з'єднуватися зі входами одного або декількох блоків.

Основними об'єктами мови FBD є елементарні функції та елементарні функціональні блоки (ФБ). Вони знаходяться в бібліотеці, логіка їх роботи (програма) написана на мові С і не може бути змінена в редакторі FBD (змінювати можна тільки їх параметри). Крім них можна використовувати функції та ФБ користувача, які конструюються користувачем з елементів мови FBD. Практика показує, що FBD є найбільш поширеним мовою стандарту IEC 61131-3. Графічна форма подання алгоритму, простота використання, повторне використання функціональних діаграм та бібліотеки функціональних блоків роблять мову FBD незамінною при розробці програмного забезпечення ПЛК. Приклад фрагменту програми мовою FBD наведено на рисунку.



Разом з тим, не можна не помітити і деякі недоліки FBD. Хоча FBD забезпечує легке подання функцій обробки як «безперервних» сигналів, зокрема, функцій регулювання, так і логічних функцій, у ньому незручним і неочевидним чином реалізуються ті ділянки програми, які було б зручно предста- вити у вигляді кінцевого автомата.

Розробка програми здійснюється за допомогою графічного редактора за допомогою формування блок-схеми з перерахованих вище компонентів, які об'єднуються один з одним або за допомогою зовнішніх (фактичних) параметрів (змінні, відповідні входах і виходах), або безпосередньо лініями зв'язку - графічними зв'язками.

Ідеологія програмування на мові FBD має на увазі, що час виконання кожної програми має бути цілком визначеним, тобто детермінованим. Іншими словами жодна програма не має права зациклитися на невизначений час, наприклад на очікуванні якої-небудь події.

FBD - програма дуже нагадує функціональну схему електронного пристрою. Кожен ФБ має фіксовану кількість вхідних точок зв'язку і фіксовану кількість вихідних точок зв'язку.

FBD - програма описує функцію між вхідними та вихідними змінними. Ця функція представляється сукупністю елементарних ФБ. Тип кожної змінної повинен бути тим же, що і тип відповідного входу. Входом FBD - блоку може бути константа, будь-яка внутрішня, вхідні або вихідна змінна.

Дана мова програмування, крім іншого, може використовуватися для опису кроків і

переходів в мові SFC. Функціональні блоки інкапсулюють дані і методи, ніж подібні об'єктно-орієнтованим мовам програмування, але вони не підтримують успадкування і поліморфізм.

30.2 Мова релейних діаграм (LD)

Програми, написані мовою Ladder Diagram (LD), складаються із серії блоків, які виконуються послідовно ПЛК.

Блок складається з ряду графічних елементів, обмежених ліворуч і праворуч шинами живлення.

Вони представляють:

– I/O ПЛК (кнопки, сенсори, реле, індикаторні лампи і т.д).
– Функції Standard Control System (таймери, лічильники і т.д).
– Арифметичні, логічні оператори і оператори дій.
– Внутрішні змінні ПЛК.

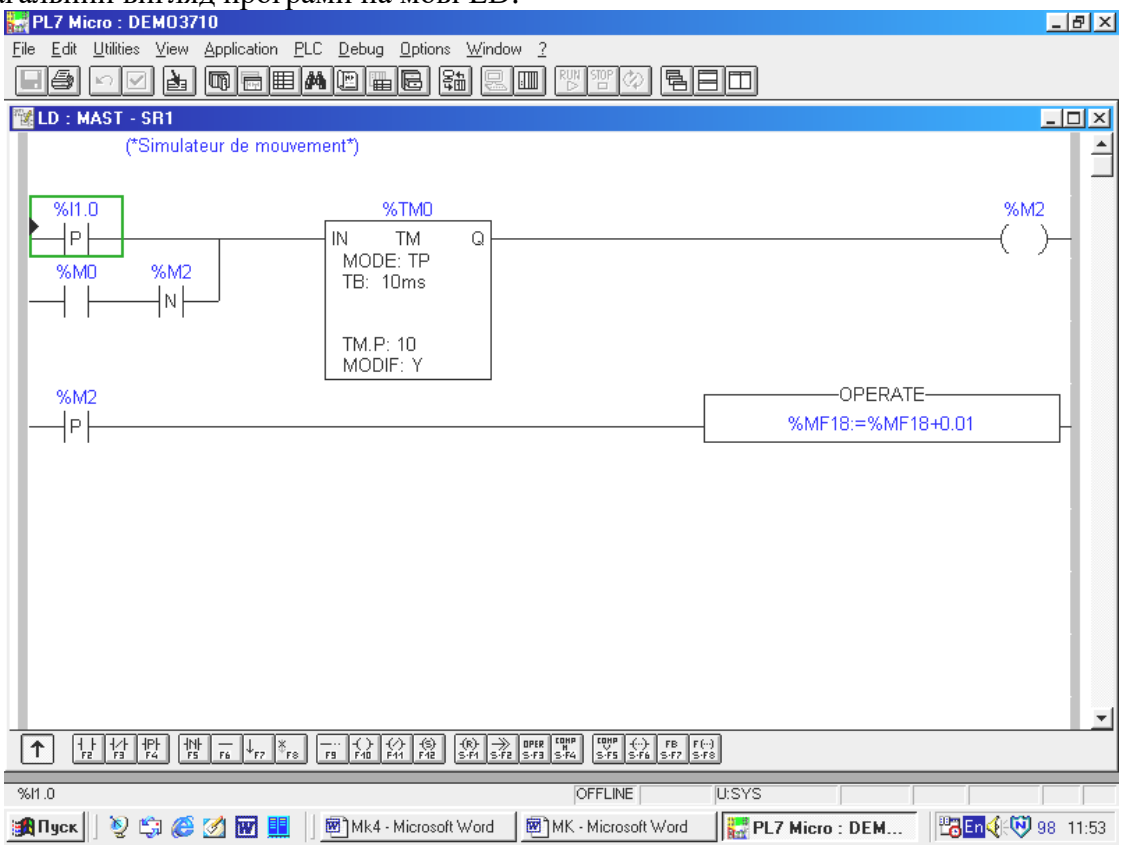
Графічні елементи взаємозв’язані горизонтальними і вертикальними зв'язками.

Кожний блок складається з максимум 7 рядків і 11 стовпців, що розділені на 2 зони:

– Зона тестування (Test Zone), що містить умови, необхідні для виконання дій.
– Зона дії (Action Zone), що містить дії, які будуть виконані відповідно до результатів виконання зони тестування.

Кожен блок може бути ідентифікований міткою і озаглавлений коментарями.

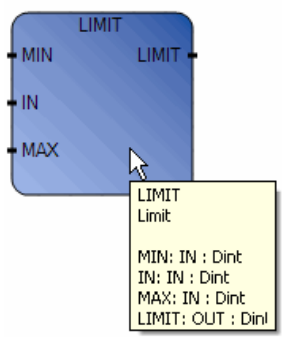
Загальний вигляд програми на мові LD:



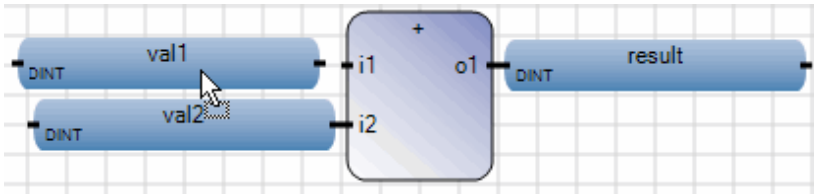
30.3 Графічні елементи

Функціональний блок

Функція

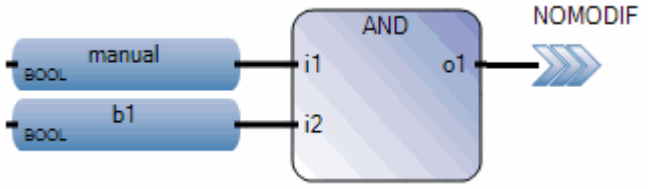


Змінні



Блоки

Переходи

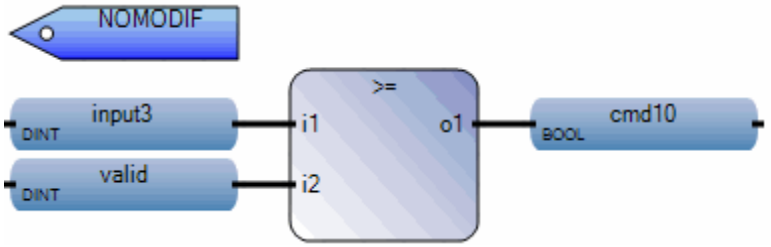


Коментарі

Коментарі інтегровані у блок і містять до 222 буквено-цифрових знаків, обмежених з обох кінців знаками (* і *).

Коментарі зберігаються в ПЛК, і користувач може завжди одержати до них доступ. Тому вони використовують пам'ять програми.

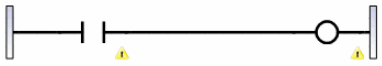
Мітки



Мітки використовують для ідентифікації блока в межах об'єкта програми (основна програма, підпрограма, і т.д), але вони не обов'язкові.

Кожна мітка може бути призначена тільки на один блок у межах того ж самого об'єкта програми.

Ланцюги



Обмотки

Пряма обмотка

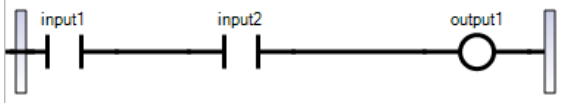

```
(* ST Equivalence: *)
output1 := input1;
output2 := input1;
```

Інверсна обмотка


```
(* ST Equivalence: *)
output1 := NOT (input1);
output2 := input1;
```

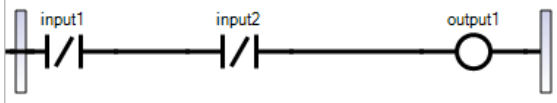
Контакти

Прямий контакт




```
(* ST Equivalence: *)
output1 := input1 AND input2;
```

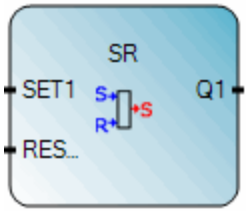
Інверсний контакт




```
(* ST Equivalence: *)
output1 := NOT (input1) AND NOT (input2);
```

Булеві операції

SR-тригер



ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: СТАНДАРТ IEC 61131-3

Стандарт МЕК 61131-3 встановлює п'ять мов програмування ПЛК, три графічних і два текстових. Спочатку стандарт називався IEC 1131-3 і був опублікований в 1993 р але в 1997 р МЕК (IEC) перейшов на нову систему позначень і в назві стандарту додалася цифра "6". Просуванням стандарту займається організація PLCopen (http://www.plcopen.org).

Основною метою стандарту було підвищення швидкості і якості розробки програм для ПЛК, а також створення мов програмування, орієнтованих на технологів, забезпечення відповідності ПЛК ідеології відкритих систем, виключення етапу додаткового навчання при зміні типу ПЛК.

Системи програмування, засновані на МЕК 61131-3, характеризуються такими показниками:

– надійністю створюваного програмного забезпечення. Надійність забезпечується тим, що програми для ПЛК створюються за допомогою спеціально призначеної для цього середовища розробки, яка містить всі необхідні засоби для написання, тестування і налагодження програм за допомогою емуляторів і реальних ПЛК, а також безліч готових фрагментів програмного коду;

– можливістю простої модифікації програми і збільшення її функціональності;

– перехід проекту з одного ПЛК на інший;

– можливістю повторного використання відпрацьованих фрагментів програми;

– простотою мови і обмеженням кількості його елементів.

– Мови МЕК 61131-3 з'явилися не як теоретична розробка, а як результат аналізу безлічі мов, які вже використовуються на практиці і запропонованих ринку виробниками ПЛК. Стандарт встановлює п'ять мов програмування з наступними назвами:

– структурований текст (ST - Structured Text);

– послідовні функціональні схеми (SFC - "Sequential Function Chart");

– діаграми функціональних блоків (FBD - Function Block Diagram);

– релейно-контактні схеми, або релейні діаграми (LD - Ladder Diagram);

– список інструкцій (IL - Instruction List).

Графічними мовами є SFC, FBD, LD. Мови IL і ST є текстовими.

У стандарт було введено кілька мов (а не одна) для того, щоб кожен користувач міг застосувати найбільш зрозумілу йому мову. Програмісти частіше вибирають мову IL (схожу на асемблер) або ST, схожу на мову високого рівня Паскаль; фахівці, що мають досвід роботи з релейного логікою, вибирають мову LD, фахівці з систем автоматичного управління (САУ) та схемотехніки вибирають звичну для них мову FBD.

Вибір однієї з п'яти мов визначаться не тільки власними уподобаннями, а й сенсом розв'язуваної задачі. Якщо вихідна задача формулюється в термінах послідовної обробки і передачі сигналів, то для неї простіше і наочніше використовувати мову FBD. Якщо завдання описується як послідовність спрацьовувань деяких ключів і реле, то для неї найбільш наочна буде мова LD. Для завдань, які спочатку формулюються у вигляді складного розгалуженого алгоритму, зручніша буде мова ST.

Мови МЕК 61131-3 базуються на наступних принципах:

1. вся програма розбивається на безліч функціональних елементів - Program Organization Units (POU), кожен з яких може складатися з функцій, функціональних блоків і програм. Будь-який елемент МЕК-програми може бути сконструйований ієрархічно з простіших елементів;

2. стандарт вимагає суворої типізації даних. Вказівка типів даних дозволяє легко виявляти більшість помилок в програмі до її виконання;

3. є засоби для виконання різних фрагментів програми в різний час, з різною швидкістю, а також паралельно. Наприклад, один фрагмент програми може сканувати кінцевий датчик з частотою 100 разів на секунду, в той час як другий фрагмент буде сканувати датчик температури з частотою один раз в 10 сек;

4. для виконання операцій в певній послідовності, яка задається моментами часу або подіями, використовується спеціальна мова послідовних функціональних схем (SFC);

5. стандарт підтримує структури для опису різнорідних даних. Наприклад, температуру підшипників насоса, тиск і стан "включено-виключено" можна описати за допомогою єдиної структури "Pomp" і передавати її всередині програми як єдиний елемент даних;

6. стандарт забезпечує спільне використання всіх п'яти мов, тому для кожного фрагмента завдання може бути обрана будь-яка, найбільш зручна мова;

7. програма, написана для одного контролера, може бути перенесена на будь-який контролер, сумісний зі стандартом МЕК 61131-3.

Будь-який ПЛК працює в циклічному режимі. Цикл починається зі збору даних з модулів введення, потім виповнюється програма ПЛК і закінчується цикл висновком даних в пристрої виведення. Тому величина контролерних циклів залежить від часу виконання програми та швидкодії процесорного модуля.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: АЛГОРИТМИ АНАЛОГОВОГО РЕГУЛЮВАННЯ

Аналоговий сигнал являє собою фізичну величину, яка може приймати будь-яке значення в заданому діапазоні від електричного сигналу до аналогового значення.

Для обробки системою фізичних величин необхідно виконати кілька кроків:

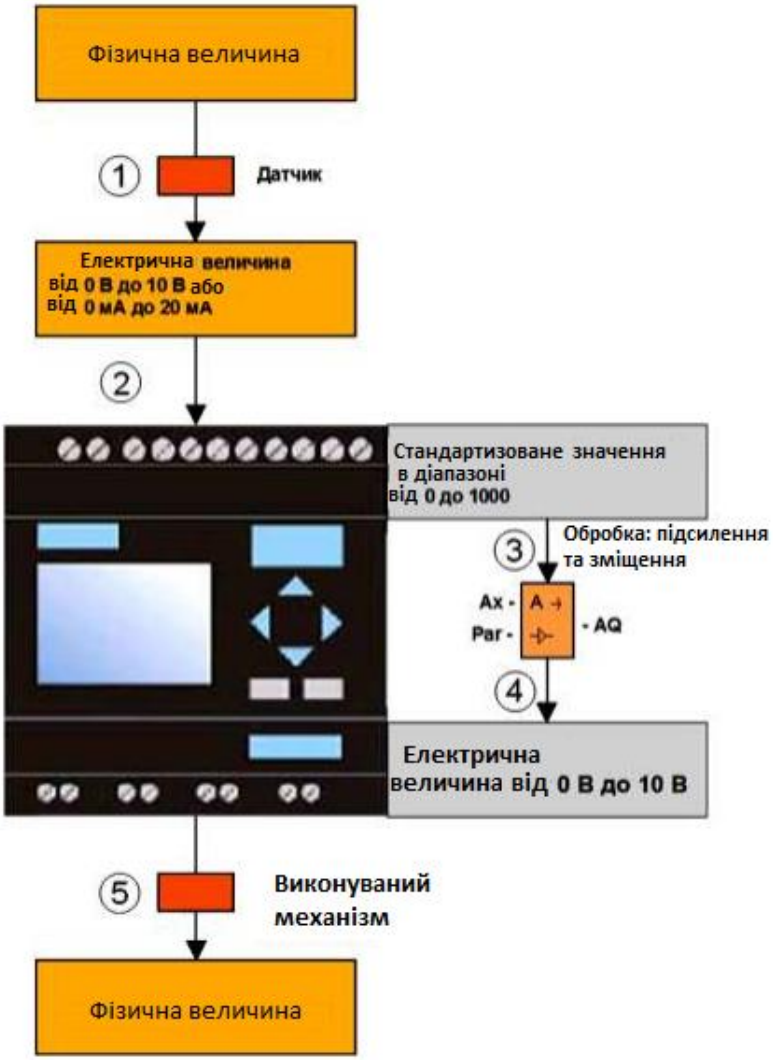
1. Модулі можуть зчитувати електричні напруги від 0 до 10 В або струм і від 0 до 20 мА на одному аналоговому вході. Тому фізична величина (наприклад, температура, тиск, частота обертання і т.п.) повинна бути перетворена в необхідний вид електричного сигналу в обраному діапазоні і мати відповідну величину. Це перетворення виконується зовнішнім датчиком.

2. Пристрій зчитує електричне значення і, в ході подальшого перетворення воно перетворюється в стандартизоване значення в діапазоні від 0 до 1000. Потім це значення використовується в комутаційної програмі на вході аналогової спеціальної функції.

3. Для того щоб пристосувати стандартизоване значення до конкретної області застосування, пристрої використовують спеціальну аналогову функцію, що враховує посилення і зміщення, для обчислення аналогового значення.

Аналогове значення після цього оцінюється спеціальною функцією (наприклад, аналоговим підсилювачем). Якщо у спеціальній аналогової функції є аналоговий вихід, то аналогове значення також подається на вихід спеціальної функції.

4. В ПЛК можна перетворювати аналогові значення назад в електричний сигнал. У цьому випадку напруга може приймати значення в межах від 0 до 10 В.



5. За допомогою цієї напруги пристрій може керувати зовнішнім виконавчим механізмом, що перетворює напругу, а також аналогове значення назад в фізичну величину.

Схема на рисунку ілюструє послідовність операцій перетворення «фізична величина» (1), «електрична величина»(2), «стандартизоване значення», «обчислення» (3), «електрична величина» (4), перетворення сигналу виконавчим механізмом (5) на фізичну величину.

Стандартизоване значення множиться на параметр. Застосування цього параметра дозволяє в більшій чи меншій мірі змінити електричну величину. Відповідно цей параметр називається «Підсиленням». Застосування цього параметра дозволяє в більшій чи меншій мірі перемістити нульову точку електричної величини. Відповідно, цей параметр називається «зміщенням нульової точки». Аналогове значення обчислюється таким чином: Аналогове значення = (стандартизоване значення х посилення) + Зсув.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: АЛГОРИТМИ СТАТИСТИЧНОГО ТА ДИНАМІЧНОГО ПЕРЕТВОРЕННЯ

У статичному стані збурюючі дії, що управляють, на систему постійні. Якщо при цьому значення регульованого параметра дорівнює заданій величині, то говорять про сталий режим роботи системи управління. Залежність між вихідними У і вхідними Х величинами в сталих режимах роботи називається статичною характеристикою системи. Статичні характеристики дають можливість оцінити характер і ступінь зв'язку між вхідними і вихідними величинами.

На практиці статичні режими вельми рідкісні, оскільки численні збурення постійно виводять систему із стану рівноваги. Режим, відмінний від статичного, називають динамічним, а перехід в часі від початкового сталого стану до нового називається перехідним процесом.

Види перехідних процесів в системах управління визначаються характером зміни вихідної величини при спрямуванні тієї або іншої дії на систему. Вони можуть бути такими, що коливаються або неперіодичними, такими, що сходяться або розходяться.

Перехідний процес в системі управління може початися або під впливом збурюючих дій, або внаслідок зміни задаючої дії (тобто при налаштування системи на нове задане значення вихідної величини). Для порівняння різних систем або оцінки їх придатності для вирішення конкретних завдань управління розглядають їх поведінку в динаміці. Вид перехідного процесу залежить не тільки від властивостей власне системи, але і від характеру зміни прикладених дій. Тому в розгляд вводять типові дії, які є найбільш несприятливими або найбільш характерними серед всіх можливих.

1) Найчастіше як типову дію використовують стрибкоподібні функції, наприклад, ступінчаста дія – одиничний стрибок.

Такий сигнал є характерним для систем автоматичної стабілізації. Наприклад, при регулюванні швидкості і натягнення прокатної на стані смуги стрибкоподібні дії можуть виникати при раптовому включенні (відключенні) системи управління.

2) Іншою типовою дією є імпульсна. Ця дія виникає в системах різкою і значною зміною навантаження за час, значно менше часу перехідного процесу. Як приклад можна вказати на стежачу систему, призначену для управління летючими ножицями в прокатному стані при розрізанні гуркоту на смуги.

3) Для систем, що працюють в умовах періодичних збурень, використовують гармонійні типові дії. Отримувані при цьому частотні характеристики дозволяють якнайповніше оцінити динамічні властивості системи.

Практика використання систем управління пред'являє до них найрізноманітніші вимоги. Так, у багатьох випадках необхідно, щоб за строго певний час система переходила з одного стійкого стану в інше (швидкодія) або щоб система достатньо точно відтворювала задаючі дії (точність).

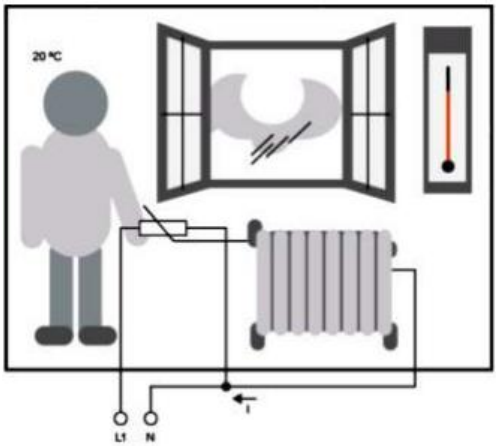
До деяких САУ пред'являють вимоги економічності процесу управління, плавності зміни вихідних величин і тому подібне. Комплекс вимог, що визначають поведінку системи в сталому і перехідному режимах при заданій дії, об'єднують в поняття якості процесу управління. Зрозуміло, що якість регулювання залежить від прийнятого алгоритму функціонування регулятора – закону регулювання.

Для оцінки якості управління використовується ряд числових показників. У статичному стані про якість управління судять по величині статичної помилки. У динамічних режимах якість систем оцінюється по характеру перехідного процесу. Показники якості, визначувані безпосередньо по кривій перехідного процесу, називають прямими оцінками якості. Найчастіше прямі оцінки отримують по кривій перехідної характеристики h(t), тобто по кривій перехідного процесу, викликаного одиничним ступінчастим сигналом за нульових початкових умов. Перехідна характеристика може бути отримана як для вихідної величини y(t), так і для її відхилення ε(t) від заданого значення.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: АЛГОРИТМИ ДИСКРЕТНОГО КЕРУВАННЯ ТА АНАЛОГО-ДИСКРЕТНОГО ПЕРЕТВОРЕННЯ

В інженерній практиці кількісні значення допускають управління і регулювання. При управлінні здійснюється зміна кількості без можливості компенсації зовнішніх впливів.

При виконанні регулювання кількість підтримується рівною заданому значенню з метою компенсації зовнішніх впливів. У наведеному прикладі «управління» означає, що людина може встановити фіксований значення теплової потужності. Нагрівач не може компенсувати падіння температури в приміщенні в разі відкриття вікна.



У наведеному прикладі струм електронагрівача є регульованою змінною. Змінний резистор є виконавчим механізмом. Рука, що управляє виконавчим механізмом (ІМ), називається органом управління. Фактична температура в приміщенні є керованою змінною або значенням технологічного процесу. Необхідна температура в приміщенні є заданим значенням змінної або уставкою.

Електрична система опалення називається керуючим процесом, термометр називається датчиком, падіння температури при відкритті вікна називається змінною збурення.

Таким чином, людина вимірюючи значення технологічного процесу (температуру в приміщенні) датчиком (термометром), порівнює значення технологічного процесу (температуру в приміщенні) з заданим значенням змінної (необхідної температурою в приміщенні) і використовує виконавчий механізм (змінний резистор) для ручного регулювання регульованої змінної (струму опалення) з метою компенсації змінної обурення (падіння температури при відкритті вікна). Людина в цій системі грає роль регулятора.

Пристрій управління складається з виконавчого механізму і органу управління. Орган управління спільно з регулятором утворюють регулюючий пристрій. Елемент порівняння використовує датчик для порівняння заданого значення змінної зі значенням технологічного процесу. При виявленні відхилення заданого значення змінної від значення технологічного процесу в контурі формується позитивне або від'ємне значення помилки, яке в свою чергу змінює значення технологічного процесу.

Значення технологічного процесу x впливає на регульовану змінну M за допомогою пристрою регулювання. При цьому утворюється замкнутий контур, відомий під назвою контуру регулювання.

Якщо в розглянутому вище прикладі відкривається вікно, то температура в приміщенні зменшується. Людина повинна збільшити теплову потужність обігрівача. У разі надмірного збільшення теплової потужності температура стає занадто високою. В цьому випадку людині необхідно зменшити теплову потужність.

У разі занадто швидкого збільшення або зменшення теплової потужності в контурі регулювання починаються коливання. Температура в приміщенні починає коливатися: стає то занадто тепло, то занадто холодно. Для запобігання цьому оператор повинен обережно і повільно зменшувати або збільшувати теплову потужність.

ТЕМА ДЛЯ САМОСТІЙНОГО ВИВЧЕННЯ: АЛГОРИТМИ ЛОГІЧНИХ ОПЕРАЦІЙ

Будь-який технологічний процес повинен виконуватися відповідно до певних правил, послідовностей операцій, тобто відповідно до його "Алгоритму функціонування" й "Алгоритму управління". Складні процеси можуть бути поділені на елементарні операції, взаємодія яких здійснюється за елементарною логікою.

При автоматизації технологічних процесів виникає необхідність у керуванні процесами без безпосереднього втручання людини, в цьому випадку спеціальні технічні засоби автоматизації повинні забезпечити ведення процесу відповідно до його алгоритму керування. Схема, що пояснює, яким чином взаємодіють технічні засоби автоматичного керування, називається структурною схемою керування.

Найбільш широке застосування при автоматизації процесів знаходять технічні засоби, що використовують електричну енергію. У цьому випадку схема керування буде називатися принциповою електричною схемою регулювання або керування, сигналізації тощо.

Для виконання елементарної логіки широко використовуються релейноконтактні пристрої (електромагнітні реле, кнопки, перемикачі, пускачі, крокові пускачі тощо).

До основних елементарних логічних операцій відносяться:

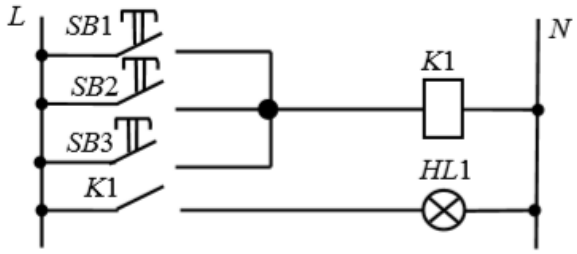
а) операції "АБО";

б) операції "І";

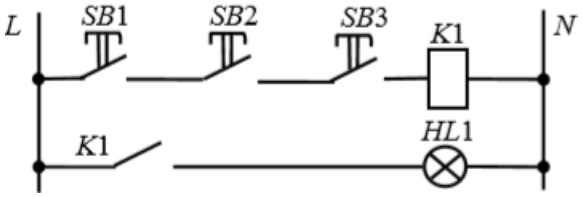
в) операції "НІ";

г) операції "Пам'ять" тощо.

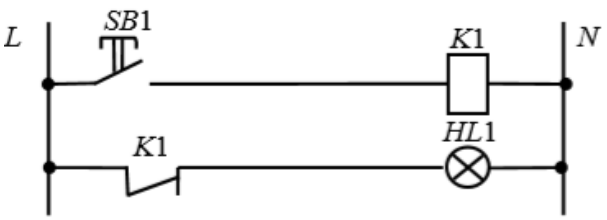
Складні схеми керування складаються з набору типових схем, що реалізують елементарні логічні операції: 1. Схема, що реалізує операцію "АБО". У цій схемі проміжне електромагнітне реле К1 включається при замиканні кожної із кнопок (або SB1, або SB2, або SB3) і замикає нормально розімкнутий контакт К1 реле, який включає сигнальну лампу HL1. Замість кнопок можуть використовуватися технологічні контактні датчики, кінцеві вимикачі, реле тощо.



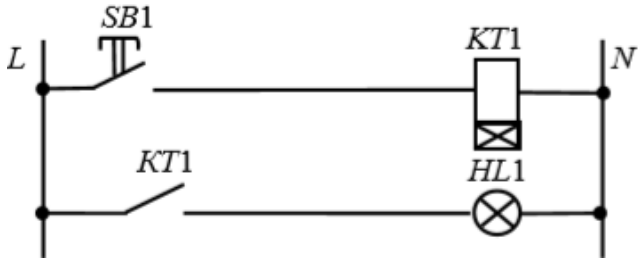
Схема, що реалізує операцію "І". У цій схемі реле К1 включається тільки при замиканні всіх кнопок SB1, SB2, SB3.



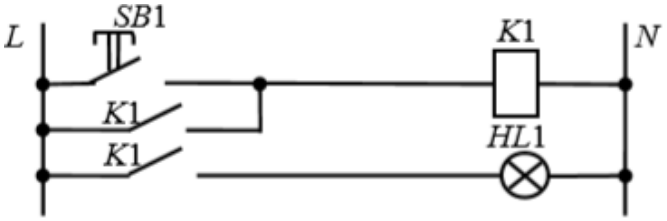
Схема, що реалізує операцію "НІ" ("заперечення", "інверсія"). У цій схемі при замиканні кнопки подається живлення на обмотку реле К1, яке включається, та своїм розімкнутим контактом К1 включає лампу HL1 і, навпаки, при розімкненні SB1, лампа HL1 виключається.



Схема, що реалізує операцію "затримка". У цій схемі при замиканні кнопки SB1 подається живлення на обмотку реле часу KT1, однак лампа HL1 включається тільки після замикання контактів KT1 з витримкою τ до замикання.



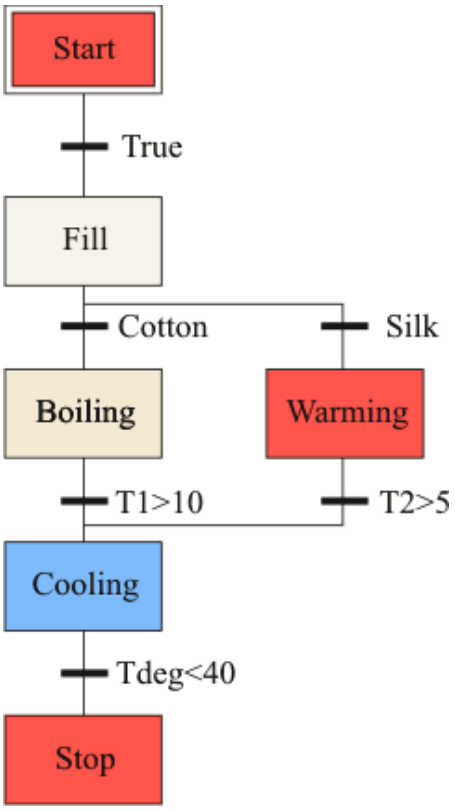
Операція "Пам'ять". Схема використовується за необхідності перетворення командного імпульсу малої діяльності в довгостроково діючий вплив. При короткочасному замиканні кнопки SB1 реле K1 включається. При цьому замикається його контакт K1, що шунтує кнопку SB1, тому при розмиканні SB1 реле K1 залишається включеним необмежено довго (поки є живлення на шинах схеми).



ЛЕКЦІЯ 31. МОВА ПОСЛІДОВНИХ ФУНКЦІОНАЛЬНИХ СХЕМ SFC

SFC називають мовою програмування, хоча по суті це не мова, а допоміжний засіб для структурування програм. Він призначений спеціально для програмування послідовності виконання дій системою управління, коли ці дії повинні бути виконані в задані моменти часу або при настанні деяких подій. В його основі лежить уявлення системи управління за допомогою понять станів і переходів між ними.

Мова SFC призначена для опису системи управління на самому верхньому рівні абстракції, наприклад, в термінах "Старт", "Наповнення автоклава", "Виконання етапу №1", "Виконання етапу №2", "Вивантаження з автоклава". Мова SFC може бути використана також для програмування окремих функціональних блоків, якщо алгоритм їх роботи природним чином описується за допомогою понять станів і переходів. Наприклад, алгоритм автоматичного з'єднання модему з комутованою лінією описується станами "Включення", "Виявлення тони", "Набір номер", "Ідентифікація сигналу" і переходами "Якщо довгий - то чекати 20 сек", "Якщо короткий - перейти в стан" Набір номери "" і т.д.



На рисунку показаний фрагмент програми на мові SFC. Програма складається з кроків і умов переходів. Кроки показуються на схемі прямокутниками, умови переходів - жирною перекресленою лінією. Програма виконується зверху вниз. Початковий крок на схемі показується у вигляді подвійного прямокутника. Умови переходів записуються поруч з їх позначеннями. Кожен крок програми може являти собою реалізацію складного алгоритму, написаного на одному з МЕК-мов.

Мова діаграмного типу SFC (Sequential Function Chart) дозволяє представити програму POU або функціональний блок ПЛК за допомогою графічної і текстової системи позначень. SFC по суті є допоміжним засобом для структурування програм за допомогою розбиття основної керуючої гілки застосунку на менші компоненти і контролю їх виконання.

Графічне представлення дозволяє чітко позначити потік виконання програми, а також робить можливим проектування послідовних і паралельних процесів застосунку.

Програмування на SFC зазвичай розділяється на 2 різних рівня: рівень 1 – показує графічно блок-схеми, номери посилань на кроки, переходи і коментарі, приєднані до них; опис кроків і переходів дається всередині прямокутників, приєднаних до символів кроку і переходу, у вигляді вільного коментаря (який не є частиною мови); рівень 2 – програмування дій всередині кроку або умов, приєднаних до переходу, на мові ST або IL; підпрограми, написані на інших мовах (FDB, ST, LD або IL) можуть звертатися до цих дій або переходах.

Початкова ситуація описується початковим кроком; після запуску програми автоматично активізуються (виділяються) все початкові кроки.

Перший рівень структурування на SFC - мережа, яка складається з елементів під назвою кроки і стани. Крок може бути активним або неактивним. Коли він активний, необхідні команди виконуються до тих пір, поки крок не стане неактивним. Заміну статусу кроку визначає стан переходу, який є логічним виразом. Якщо умова переходу стає «Істина», то наступний крок стає активним, а попередній крок дезактивується.

Зі зміною переходу, властивість «активний» переходить від активного кроку до його наступника або наступників; таке пересування утворює мережу. Ця властивість може бути розділена в разі виконання паралельних гілок, а потім відновлена при закінченні виконання таких гілок. Дії в кроках мають спеціальні класифікатори, що визначають спосіб їх виконання всередині кроку: циклічне виконання (N), одноразове виконання (P) і т.д. Всього таких класифікаторів дев'ять, причому серед них є класифікатори зі збереженими (S), відкладеними (D) і обмеженими за часом (L) діями.

Список інформаційних джерел

1. Іванов В. Г. Основи інформатики та обчислювальної техніки: підручник /В. Г. Іванов, В. В. Карасюк, М. В. Гвозденко; за заг. ред. В. Г. Іванова. – Х.: Право, 2012. – 158 с.
2. Інформатика та комп'ютерна техніка: Навчальний посібник / За ред. М.Є. Рогози. - К: Видавничий центр «Академія», 2006. – 368 с.
3. Клименко О. Ф., Головко Н. Р., Шарапов О.Д. Інформатика та комп'ютерна техніка: Навч.-метод. посібник / За заг. ред. О. Д. Шарапова. – К.: КНЕУ, 2002. – 534 с.
4. Макарова М.В., Карнаухова Г.В., Запара С.В. Інформатика та комп'ютерна техніка: Навчальний посібник / За заг. ред. д.е.н., проф. М.В. Макарової. – 3-тє видання, перероб. і доп. Суми: ВТД «Університетська книга», 2008. – 665 с.
5. Таненбаум Е., Остин Т. Архитектура компьютера. – Питер, 2013. – 816 с.
6. Леонтьев В.П. Windows 10: новейший самоучитель, 2-е издание. – М.: Издатиельство «Э», 2016 – 576 с.
7. Ратбон.Э. Windows 10 для чайников. – М.: Диалектика, 2016. – 480с.
8. Завадський І. О., Забарна А.П. Microsoft Excel у профільному навчанні. – К.: Вид. група BHV, 2011. – 272с.: іл.
9. Нелюбов В. О., Куруца О. С. Основи інформатики. Microsoft Excel 2016: навчальний посібник. Ужгород: ДВНЗ «УжНУ», 2018. – 58 с.: іл.
10. Серогодский В.В. Microsoft Office 2016 / Office 365. Полное руководство. – Издательство: Наука и техника, 2017. – 448 с.: ил.
11. Абрамов В.О., Чегренець В.М. Основи баз даних та робота в СУБД Access: навчальний посібник для спеціальності «Інформатика». – К.: Київ. ун-т ім. Б. Грінченка, 2013. –100 с.
12. Сорочак А.П. Основи автоматизації проектування в будівництві: конспект лекцій – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2018. – 120 с.
13. Креневич, А.П., Обвінцев О.В. С у задачах і прикладах: навчальний посібник із дисципліни "Інформатика та програмування". – К.: Видавничо-поліграфічний центр "Київський університет", 2011. – 208 с.
14. Евстифеев А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL – М.: Додэка-XXI, 2008. – 560 с.
15. Трамперт В. AVR-RISC микроконтроллеры. – К.: МКПресс, 2006. – 464 с.
16. Петров И.В Программируемые контроллеры. Стандартные языки и приемы прикладного программирования / Под ред. проф. В.П. Дьяконова. – М: ООО «СОЛОН-Пресс», 2004. – 256 с.
17. Павельчак А.Г., Самотий В.В., Яцук Ю.В. Програмування мікроконтролерів систем автоматики: конспект лекцій. – Львів: Львівська політехніка. – 2012. – 143 с.
18. Антонов О. С. Обчислювальна техніка та мікропроцесори: підручник / О. С. Антонов, І. В. Хіхловська. – Одеса: ОНАЗ, 2011. – 440 с.
19. Абель П. Язык Ассемблера для IBM PC и программирования. – М.: Высшая школа, 2002. – 324 с.
20. Інженерна комп’ютерна графіка: підручник / Р. А. Шмиг, В. М. Боярчук, І. М. Добрянський, В. М. Барабаш; за заг. ред. Р. А. Шмига. – Львів: Український бестселер, 2012. – 600 с.
21. Грабовский Р. Иллюстрированный справочник по AutoCAD 2004: пер. с англ. – М.: Изд. дом Вильямс, 2005. – 880 с.: ил.
22. http://www.3s-software.com/
23. http://www.codesys.ru/
24. www.autodesk.com.