ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД УКООПСПІЛКИ
«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ» (ПУЕТ)
Навчально-науковий інститут денної освіти
Кафедра комп'ютерних наук та інформаційних технологій

ПРОГРАМУВАННЯ ІІ

НАВЧАЛЬНО-МЕТОДИЧНИЙ ПОСІБНИК
для самостійного вивчення навчальної дисципліни
студентами спеціальності 122 Комп'ютерні науки,
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Полтава
ПУЕТ
2023

Автори: ***О. П. Кошова,*** к. пед. н., доцен, доцент кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі»;
О. В. Ольховська, к. ф.-м. н., завідувач кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі»;
Д. М. Ольховський, к. ф.-м. н., доцент, доцент кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі»;
О. Г. Оріхівська, ст. викладач кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі».

Рецензенти: ***Т. О. Парфьонова,*** к. ф.-м. н., доцент, доцент кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі»;
Т. В. Чілікіна, к. ф.-м. н., доцент, доцент кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі».

Рекомендовано до видання, розміщення в електронній бібліотеці та використання в освітньому процесі на засіданні кафедри комп'ютерних наук та інформаційних технологій ПУЕТ
15 вересня 2022 р., *протокол №* ***1***

Програмування ІІ : навчально-методичний посібник для самостійного вивчення навчальної дисципліни студентами спеціальності 122 Комп'ютерні науки, освітня програма «Комп'ютерні науки» ступеня бакалавра / О. П. Кошова, О. В. Ольховська, Д. М. Ольховський, О. Г. Оріхівська. – Полтава : ПУЕТ, 2023. – 313 с. – 1 електрон. опт. диск (CVD-ROM).

Відповідальні за зміст навчально-методичного видання автори, рецензенти та завідувач кафедри комп'ютерних наук та інформаційних технологій ***О. В. Ольховська***

ЗМІСТ

ВСТУП

Основною метою вивчення дисципліни «Програмування ІІ» є формування у студентів системного мислення і навичок алгоритмічного програмування й об'єктно-орієнтованого програмування з використанням засобів мови програмування високого рівня С++.

Цей посібник може бути використаний, як для самостійного опанування студентами цією дисципліною, так і під час виконання завдань лабораторних робіт, модульних контрольних чи написання проектів із С++ під час вивчення дисципліни «Проектне навчання з програмування ІІ», яка є логічним продовженням курсу «Програмування ІІ».

У посібнику містяться основі теоретичні відомості з усіх тем дисципліни «Програмування ІІ» із використанням мови програмування С++.

Крім того, наведено достатньо велику кількість прикладів реалізації програмних кодів із відповідних тем. Усі коди є універсальними і можуть бути використаними для реалізації як в онлайн компіляторі так і у середовищах розробки, наприклад Visual Studio чи CLion, тобто студенти не обмежені у виборі компіляторів для їх реалізації. Усі коди протестовано в онлайн компіляторі https://www.onlinegdb.com/online_c++_compiler.

Компетентності, яких студенти набудуть під час вивчення цієї дисципліни, і за допомогою посібника у тому числі, допоможуть у вивченні багатьох дисциплін, таких як «Алгоритми і структури даних», «Програмування і підтримка веб-застосувань», «Виробнича практика», «Переддипломна практика», Дипломне проектування. Крім того, опанування такою широко затребуваною мовою програмування, як С++ дозволить студентам значно легше вивчати інші мови програмування.

Під час створення навчально-методичного посібника використано дані з відкритих джерел і підручників. Повний список використаних ресурсів наведено в списку використаних джерел.

МОДУЛЬ 1. ВВЕДЕННЯ ДО МОВИ ПРОГРАМУВАННЯ С++. КЕРІВНІ СТРУКТУРИ. ФУНКЦІЇ

Тема 1. Введення до мови програмування С++. Керівні структури

С (укр. *Сі*) – універсальна, процедурна мова програмування загального призначення, розроблена у 1972 році Деннісом Рітчі у Bell Telephone Laboratories із метою написання нею операційної системи UNIX.

Хоча С і було розроблено для написання системного програмного забезпечення, наразі вона досить часто використовується для написання прикладного програмного забезпечення.

Мова С імовірно є найпопулярнішою у світі мовою програмування за кількістю вже написаного нею програмного забезпечення, доступного під вільними ліцензіями коду та кількості програмістів, котрі її знають. Версії компіляторів для мови С існують для багатьох операційних систем та апаратних архітектур. С здійснила великий вплив на інші мови програмування, особливо на С++, яка спочатку проектувалася, як розширення для С, а також на Java та C#, які запозичили у С синтаксис.

Перша програма мовою С++

```
#include <iostream>
int main()
{
   std::cout<<"Hello, world!\n";
   std::cin.get();
   return 0;
}
```

Налаштування виводу кирилиці

```
#include <iostream>
using namespace std;
int main ()
{
   setlocale(LC_ALL, "Ukr");
   double x= 2.5;
   cout << "Значення змінної/об'єкту = " << x << endl;
```

```
    cout << "Адреса змінної/об'єкту   = " << &x << endl;
    cin.get();
    return 0;
 }
```

Застосування iostream.h

Сучасні компілятори не підтримують застарілу бібліотеку C++ <iostream.h>

Щоб запустити приклад (наприклад, з деякого підручника), який застосовує цю бібліотеку, необхідно:

− замінити <iostream.h> на нову бібліотеку <iostream>;

− добавити підключення «Простору імен» std такою конструкцією: using namespace std.

Наприклад:

```
#include <iostream>
using namespace std;
int main ()
{
   cout<<"Help me "<<endl;
   cin.get();
   return 0;
}
```

Найчастіше в **C++** використовують потокове введення даних, операції якого включено до складу класів **istream** чи **iostream.** Воно може здійснюватися з визначеним у цих класах вхідним потоком **std::cin** чи вхідним потоком, який визначив користувач. Для читання символів із цього потоку вказується операція витягу з потоку, що позначається за допомогою символів **“>>“**. Це перевантажена операція, визначена для всіх простих типів і покажчика на **char.**

Формат запису оператора **cin** має вигляд:

std::cin >> values;

Наприклад, для введення значень змінним x і y можна написати:

std::cin >> x >> y;

Виведення даних може бути неформатованим і форматованим. Найчастіше для виведення застосовується визначена операція <<, що записується разом з ім’ям вихідного потоку

std::cout. Отже, запис: std::cout << x; означає вивести значення змінної x (чи записати в потік). Цей оператор вибирає необхідну функцію перетворення даних у потік байтів.

Формат запису cout відрізняється від форми запису команд C++:

std::cout << data [<< data << "\n"];

де **data** – це змінні, константи чи вирази; **"\n"** – керуючий символ перенесення каретки. Запис **std::** означає, що потоки cin і cout об'явлені в просторі імен std. Для того, щоб спростити форму запису, можна підключити простір імен у глобальну видимість, застосувавши оператор using:

using namespace std;

Для додаткового керування вихідними даними використовуються маніпулятори **setw(n)** і **setprecision(k)**. Маніпулятор **setw(n)** служить для вказівки довжини поля, що приділяється для виведення таких даних (тут **n** – кількість позицій у рядку), **setprecision(k)** призначений для вказівки кількості позицій у дробовій частині дійсних чисел. Для використання цих маніпуляторів слід підключати заголовний файл <iomanip.h>.

Для форматованого введення даних використовується функція **scanf** з заголовного файлу <c**stdio**>. Функція **scanf** має змінне число параметрів, але як фактичні параметри вона використовує адреси змінних, а не їхні значення. При цьому перед відповідним параметром ставиться знак **&** – символ взяття адреси змінної. Наприклад, **&x** означає "адреса змінної **x**", а не значення, що ця змінна має в цей момент. Рядок форматів функції **scanf** указує, які дані очікуються на вході. Якщо функція зустрічає у форматному рядку знак **%,** за яким стоїть знак перетворення, то вона буде пропускати на вході символи доти, поки не зустріне який-небудь непорожній символ. Форма запису цієї функції має вигляд:

scanf ("рядок форматних кодів", список імен змінних **);**.

Для форматованого виведення даних використовується функція з того ж заголовного файлу **<cstdio>.** Для звертання до функції використовують параметри, які розташовуються у круглих дужках. Найчастіше функція printf використовується для виведення значень змінних. Першим аргументом у звертанні до функції ставиться рядок форматів (знаходиться в подвійних лапках), а наступними, якщо вони є, виведені об'єкти.

Рядок форматів може містити звичайні символи, що копіюються при виведенні, і специфікації перетворення, що починаються зі знака **%**: за специфікаціями випливає символ перетворення. Кожна специфікація перетворення відповідає одному з аргументів, що випливають за форматним рядком, і між ними установлюється взаємно однозначна відповідність, наприклад:

printf ("Значення a, b, c рівні: %d %d %d \n", a, b, c);

Тут буква **d** у специфікації перетворення вказує, що значення аргументу повинне бути надруковане як десяткове ціле число.

При виведення використовуються ті ж специфікації, що і при введенні:

% c – для виведення окремого символу;
% s – для виведення символьного рядка;
% x – для виведення шістнадцяткової літери;
% 0 – для виведення вісімкових чисел;
% f – для виведення чисел із крапкою, що плаває.

Використання математичних функцій у мові С++ потребує підключення заголовного файлу #include <cmath>

Обчислити значення площі круга за значенням радіуса, який вводиться з клавіатури

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
int main()
{
    const double Pi = 3.1415;
    float Rad; double Square;
    cout << "Input radius, radius =";
    cin >> Rad;
    Square = Pi*Rad*Rad;     // Square = Pi*pow(Rad,2);
    cout << "Square = "<<Square << endl;
    system("pause");
    return 0;
}
```

Перетворення типів

Класифікації перетворень типів

Часто у виразах фігурують змінні різних типів. У таких випадках компілятор виконує так зване приведення або перетворення одного типу в інший. У C++ існують такі класифікації перетворень типів:

- класифікація за діапазоном значень;
- класифікація за способом здійснення перетворення.

Класифікація за діапазоном значень при перетворенні типів

При класифікації за діапазоном значень розрізняють:

- звужуюче перетворення типів. У цьому випадку більший тип даних звужується до меншого (звужується діапазон значень). Тут може виникнути втрата даних;
- розширювальне перетворення типів. При такому перетворенні менший тип даних розширюється до більшого типу даних (розширюється діапазон значень).

Нижченаведені приклади демонструють ці два види перетворень.

Приклад звужуючого перетворення даних

Звужуюче перетворення даних, що приводить до втрати даних

```
// При ініціалізації змінної
int d = 5.85; // тип double звужується до типу int, d = 5
cout << "d = " << d << endl; // d = 5 – відбувається втрата даних

// При присвоєнні у виразі
int k;
k = 2.3 + 0.5 + 0.1; // тип double звужується до типу int
cout << "k = " << k << endl; // k = 2 – відбувається втрата даних

Звужуюче перетворення без втрати даних
short f = 2000; // тип int звужується до типу short
cout << "f = " << f << endl; // f = 2000
```

Як видно з прикладу, при звужуючому перетворенні типів можлива втрата даних.

Приклад розширювального перетворення типів

```
// Розширювальне перетворення даних
unsigned int t = 4000000000; // int розширюється до unsigned int без помилок
cout << "t = " << t << endl;
```

У прикладі значення 4000000000 має тип int. Це значення коректно розширюється до типу unsigned int.

Якщо тип змінної t вказати як int

```
int t = 4000000000; // число 4000000000 не поміститься в тип int, t = -294967296 - втрата даних
```

то значення **t** буде рівне -294967296. Це означає, що число 4000000000 не поміститься в тип int.

Класифікація за способом здійснення перетворення. Неявне і явне перетворення типів

Залежно від напрямку перетворення, це перетворення може бути реалізоване одним з двох способів:

− **неявне перетворення**. Це перетворення ще називають автоматичним. Неявне перетворення компілятор виконує самостійно без втручання програміста;

− **явне перетворення** або приведення типів. У цьому випадку в дужках вписується тип перетворення.

Перетворення типів

```
// 1. Неявне перетворення типів
int a = 25;
double x;
x = a; // неявне перетворення int => double

float y = 78.3; // неявне перетворення double => float

// 2. Явне перетворення типів
int b;
b = (int)8.6; // b = 8, явне перетворення double => int

float z = (float)12.55; // явне перетворення double => float
```

Правила перетворення типів у виразах

Якщо у виразі зустрічаються два операнди різних типів, то діють такі правила:

− усі операнди перетворюються до типу самого найбільшого операнду. Процес такого перетворення називають **розширенням типу** (integral promotion);

− усі типи char та short int перетворюються до типу int. Процес такого перетворення називають цілочисельним розширенням (integer promotion);

− якщо один з операндів має тип double тоді будь-який інший операнд приводиться до типу double. Навіть, у випадку з типом char, відбувається приведення до типу double;

− після перетворення обидва операнди мають однаковий тип, який є типом результату операції.

Приклади автоматичного перетворення типів

```
Перетворення між типами char та int:
char c;
int d;

c = 'A';
d = c; // d = 65

d = 67;
c = d; // c = 'C'
Перетворення між типами int та float:
int d = 28;
float x;

x = d;       // x = 28.0 – тип float
d = 5.0 + 5; // d = 10 – тип int
Перетворення між типами float і double
float f;
double d;
int size;

f = 2.54f;
d = f; // d = 2.54 – типу double
d = 2.0f + 8.5; // результат типу double
```

Якщо вираз містить цілочисельний тип, то значення типу bool автоматично перетворюються в цілі числа 0 та 1. Значенню 0 відповідає значення false. Значенню 1 або ненульовому значенню відповідає значення true.

Фрагмент коду, що демонструє перетворення для типу bool

```
bool b;
int a;

a = 0;
b = a; // b = False

a = 1;
b = a; // b = True

a = 50;
b = a; // b = True
```

Якщо вираз містить цілочисельний тип, то значення типу bool автоматично перетворюються в цілі числа 0 та 1. Значенню 0 відповідає значення false. Значенню 1 або ненульовому значенню відповідає значення true.

Приклад. Фрагмент коду, що демонструє перетворення для типу bool

```
bool b;
int a;

a = 0;
b = a; // b = False

a = 1;
b = a; // b = True

a = 50;
b = a; // b = True
```

Арифметичні операції і привласнення. Інкремент і декремент

Арифметичні операції проводяться над числами. Значення, які беруть участь в операції, називають операндами. У мові програмування C++ арифметичні операції бінарними (проводяться над двома операндами) і унарними (виконуються над одним операндом). До бінарних операцій належать такі:

Арифметична операція +

Операція додавання повертає суму двох чисел:

```
int a = 10;
int b = 7 ;
int c = a + b ; // 17
int d = 4 + b ; // 11
```

Арифметична операція –

Операція віднімання повертає різницю двох чисел:

```
int a = 10;
int b = 7 ;
int c = a – b ; // 3
int d = 41 – b ; // 34
```

Арифметична операція *

Операція множення повертає добуток двох чисел:

```
int a = 10;
int b = 7 ;
int c = a ∗ b ; // 70
int d = b ∗ 5 ; // 35
```

Арифметична операція /

Операція ділення повертає частку двох чисел:

```
int a = 2 0;
int b = 5 ;
int c = a / b ; // 4
int d = 2 2. 5 / 4 . 5 ; // 5
```

При діленні варто бути уважним, оскільки якщо в операції беруть участь два цілі числа, то результат ділення округлятиметься до цілого числа, навіть якщо результат привласнюється змінній float або double:

```
double k = 10 / 4 ; // 2
cout << k ;
```

Щоб результат представляв число з плаваючою крапкою, один з операндів також повинен представляти число з плаваючою крапкою:

```
double = 10.0 / 4 ; // 2. 5
cout << k ;
```

Арифметична операція %

Операція отримання залишку від цілочисельного ділення:

```
int a = 33;
int b = 5 ;
int c = a % b ; // 3
int d = 22 % 4 ; // 2 (22 – 4∗5 = 2)
```

Також є дві унарні арифметичні операції, які проводяться над одним числом:

++ (інкремент) і – – (декремент). Кожна з операцій має два різновиди: префіксна і постфіксна:

Префіксний інкремент

Збільшує значення змінної на одиницю і одержаний результат використовується як значення виразу ++x

```
int a = 8 ;
int b = ++a ;
cout << a << "\n " ; // 9
cout << b << "\n " ; // 9
```

Постфіксний інкремент

Збільшує значення змінної на одиницю, але значенням виразу x++ буде те, яке було до збільшення на одиницю

```
int a = 8 ;
int b = a++;
cout << a << "\n " ; // 9
cout << b << "\n " ; // 8
```

Префіксний декремент

Зменшує значення змінної на одиницю, і набуте значення використовується як значення виразу – –x

```
int a = 8 ;
int b = – –a ;
cout << a << "\n " ; // 7
cout << b << "\n " ; // 7
```

Постфіксний декремент

Зменшує значення змінної на одиницю, але значенням виразу x– – буде те, яке було до зменшення на одиницю

```
int a = 8 ;
int b = a– –;
cout << a << "\n " ; // 7
cout << b << "\n " ; // 8
```

Арифметичні операції обчислюються зліва направо. Одні операції мають більший пріоритет ніж інші і тому виконуються спочатку. Операції в порядку зменшення пріоритету:

++ (інкремент)

– – (декремент)

∗ (множення)

/ (ділення)

% (залишок від ділення)

+ (додавання)
– (віднімання)

Пріоритет операцій слід враховувати під час виконання набору арифметичних виразів:

```
int a = 8 ;
int b = 7 ;
int c = a + 5 ∗ ++b ; // 48
cout << c ;
```

Хоча операції виконуються зліва направо, але спочатку виконуватиметься операція інкремента ++b, яка збільшить значення змінної b і поверне його як результат, оскільки ця операція має більший пріоритет. Потім виконується множення 5 * ++b, і лише в останню чергу виконується додавання a + 5 * ++b. Дужки дозволяють перевизначити порядок обчислень. Наприклад:

```
int a = 8 ;
int b = 7 ;
int c = ( a + 5 ) ∗ ++b ; // 104
cout << c ;
```

Не дивлячись на те, що операція додавання має менший пріоритет, але спочатку виконуватиметься саме додавання, а не множення, оскільки операція додавання вкладена в дужки.

Операції додавання (+) та віднімання (–) можуть бути як бінарними, так і унарними.

Бінарні операції + та – використовуються у виразах при проведенні обчислень.

Унарні операції + та – використовуються для позначення знаку числа (додатне число або від'ємне число).

Наприклад.

```
int a, b;

a = -8; // унарна операція '-', позначає знак числа
b = +9; // унарна операція '+', b = 9

a = b - 5; // бінарна операція '-', використовується у виразі для обчислення
```

Операція % використовується над цілими операндами. Операція % дозволяє отримати остачу від ділення цілих операндів.

Наприклад

```
// Операція % – знаходження остачі від ділення
int a, b;
int c;

a = 3;
b = 5;
c = a % b; // c = 3

a = 8;
b = 4;
c = a % b; // c = 0
c = 12 % 35; // c = 12
c = 35 % 12; // c = 11
c = -5 % -3; // c = -2
```

Операції привласнення (присвоювання)

У мові C++ можна використовувати складені оператори присвоювання. Ці оператори є зручними, коли у програмі використовуються довгі імена змінних. У цьому випадку відпадає необхідність зайвий раз вводити довге ім'я змінної.

Загальний вигляд складеного оператора присвоювання такий:

ім'я_змінної ***operation***= вираз;

де

- ім'я_змінної – безпосередньо ім'я змінної, якій присвоюється значення;
- **operation** – одна з операцій **+**, **–**, *****, **/**, **%**, **&**, |, ^, <<, >>.

Мова C++ підтримує такі складені оператори присвоювання:
+=, **-=**, ***=**, **/=**, **%=**, **&=**, **|=**, **^=**, **<<=**, **>>=**

Операції привласнення дозволяють привласнити деяке значення. Ці операції виконуються над двома операндами, причому лівий операнд може представляти іменований вираз, що тільки модифікується, наприклад, змінну.

Базова операція привласнення = дозволяє привласнити значення правого операнда лівому операнду:

```
int x ;
x = 2
```

Тобто в цьому випадку змінна x (лівий операнд) матиме значення 2 (правий операнд).

Варто відзначити, що тип значення правого операнда не завжди може співпадати з типом лівого операнда. У цьому випадку компілятор намагається перетворити значення правого операнда до типу лівого операнда.

При цьому операції привласнення мають правосторонній порядок, тобто виконуються справа наліво. І, таким чином, можна виконувати множинне привласнення:

```
int a , b , c ;
a = b = c = 34;
```

Тут спочатку обчислюється значення виразу c = 34. Значення правого операнда – 34 привласнюється лівому операнду. Далі обчислюється вираз b = c: значення правого операнда c (34) привласнюється лівому операнду b. І в кінці обчислюється вираз a = b: значення правого операнда b (34) привласнюється лівому операнду a.

Крім того, слід зазначити, що операції привласнення мають якнайменший пріоритет у порівнянні з іншими типами операцій, тому виконуються в останню чергу:

```
int x ;
x = 3 + 5 ;
```

Відповідно до пріоритету операцій спочатку виконується вираз 3 + 5, і тільки потім його значення привласнюється змінній x.

Уся решта операцій привласнення є поєднанням простої операції привласнення з іншими операціями:

+=: привласнення після додавання. Привласнює лівому операнду суму лівого і правого операндів: A += B еквівалентно A = A + B

-=: привласнення після віднімання. Привласнює лівому операнду різницю лівого і правого операндів: A -= B еквівалентно A = A - B

*=: привласнення після множення. Привласнює лівому операнду добуток лівого і правого операндів: A *= B еквівалентно A = A * B

/=: привласнення після розподілу. Привласнює лівому операнду частку лівого і правого операндів: A /= B еквівалентно A = A / B

%=: привласнення після частки від ділення по модулю. Привласнює лівому операнду залишок від цілочисельного розділення лівого операнда на правий:

A

«=: привласнення після зсуву розрядів вліво. Привласнює лівому операнду результат зсуву його бітового представлення вліво на певну кількість розрядів, рівну значенню правого операнда: A «= B еквівалентно A = A

« B

»=: привласнення після зсуву розрядів управо. Привласнює лівому операнду результат зсуву його бітового уявлення управо на певну кількість розрядів, рівну значенню правого операнда: A »= B еквівалентно A = A

» B

&=: привласнення після порозрядної кон'юнкції. Привласнює лівому операнду результат порозрядної кон'юнкції його бітового представлення з бітовим представленням правого операнда: A &= B еквівалентно A = A

& B

|=: привласнення після порозрядної диз'юнкції. Привласнює лівому операнду результат порозрядної диз'юнкції його бітового представлення з бітовим представленням правого операнда: A |= B еквівалентно A = A | B

=: ˆ привласнення після операції виключає АБО. Привласнює лівому операнду результат операції виключає АБО його бітового представлення з бітовим представленням правого операнда: A = B еквівалентно A = A ˆ Bˆ

Нариклад:

```
int a = 5 ;
a += 10; // 15
a –= 3 ; // 12
a ∗= 2 ; // 24
a /= 6 ; // 4
a <<= 4 ; // 64
a >>= 2 ; // 16
```

Приклад використання складених операторів присвоювання
// складені оператори присвоювання float x, y;

```
int a, b;

// +=, -=
a = 8;
b = 5;
a += b; // a = a + b = 13
b -= 4; // b = b - 4 = 1

// *=, /=
x = 4;
y = 5;
x *= y; // x = x * y = 4 * 5 = 20
y /= 2.5; // y = y / 2.5 = 2.0

// &=, |=
a = 8;
b = 3;
a &= b + 5; // a = a & b + 5 = a & (b+5) = 8
a |= b; // a = a | b = 11

// >>=, <<=
a = 34;
a >>= 1; // a = a >> 1 = 17
a = 6;
a <<= 3; // a = a << 3 = 48

// %=
b = 15;
b %= 6; // b = b % 6 = 3
```

Базові складові мови C/C++

- Алфавіт (абетка)
- Лексеми
- Вирази
- Оператори

У природній мові спілкування виділяють чотири основні елементи: символ, слово, словосполучення і речення. Подібні елементи існують і в алгоритмічній мові, тільки слова мають назву лексеми, словосполучення – вирази, а речення – опера-

тори. Лексеми створюються із символів, вирази – із лексем і символів, оператори – із символів, виразів і лексем.

Алфавіт

Алфавіт мови C++ містить:

- великі (A-Z) і малі (a–z) літери латинського алфавіту та символ підкреслення (_);
- арабські цифри від 0 до 9;
- знаки арифметичних дій +, -, *, /, %, ++, --;
- знаки побітових операцій <<, >>, &, |, ~, ^;
- знаки відношень <, <=, ==, !=, >, >=;
- знаки логічних операцій &&, ||, !;
- розділові знаки , ; : пропуск;
- спеціальні знаки ., =, ->, ?, \, $, #, ', ";
- символи дужок (,), [,], {, }.

Лексеми й ідентифікатори

- Лексеми, тобто базові елементи мови з певним самостійним значенням, складаються із символів алфавіту. До них відносять ідентифікатори, ключові слова, знаки операцій, константи, роздільники (дужки, крапка, кома, символи пропуску). Межі лексем визначаються іншими лексемами-роздільниками або знаками операцій.
- Ідентифікатором, тобто ім’ям програмного об’єкта, називається будь-яка послідовність літер латинського алфавіту, цифр і символу підкреслення за умови, що першою стоїть літера або символ підкреслення, а не цифра.

Існує два різновиди ідентифікаторів:

- стандартні, наприклад, імена всіх вбудованих у мову функцій;
- користувальницькі (створені користувачем-програмістом).

Примітка: в C/C++ при створені ідентифікаторів розрізняються великі та малі символи (на відміну від інших мов, наприклад Pascal, SQL,…), тобто наступні ідентифікатори абсолютно різні і мають право на існування в одному блоку: min, mIn, Min, MIN. Часто використовується так званий camel стиль.

Ключові слова (зарезервовані ідентифікатори)

Ключовими (службовими) словами називають ряд зарезервованих ідентифікаторів, що вживаються для побудови конструкцій мови і мають фіксоване значення. За смисловим навантаженням службові слова поділяються на такі основні групи:

- специфікатори типів – **char, int, long, typedef, short, float, double, enum, struct, union, signed, unsigned, void;**
- кваліфікатори типів – **const і volatile**;
- класи пам'яті – **auto, extern, register, static**;
- для побудови операторів – **for, while, do, if, else, switch, case, continue, goto, break, return, default, sizeof.**

Основні ключові слова C++

asm	delete	goto	register	throw
auto	do	if	return	try
break	double	inline	short	typedef
case	else	int	signed	typename
catch	enum	long	sizeof	union
char	explicit	new	static	unsigned
class	extern	operator	struct	virtual
const	float	private	switch	void
continue	for	protected	template	volatile
default	friend	public	this	while

Структура простої програми мовою C/C++

- Підключення бібліотек (#include <ім'я файлу>)
- Головна функція main. Наявність функції main обов'язкове, причому в одному екземплярі – це «місце входу» виконання програми.
- Тіло (блок {}) головної функції main.

```
#include <ім'я файлу> // Підключення бібліотек
int main() // головна функція main
{ //початок тіла main
      оператори
      return <ціле значення>; // головна функція по стандарту повинна вертати ціле значення
} //кінець тіла main
```

Примітка: за стандартом існує тільки два вигляди головної функції main:

- int main()
- int main(int argc, char *argv[])

Варіант **void main()**, який достатньо часто зустрічається в літературі, **не за стандартом** і **не підтримується** усіма компіляторами.

Коментарі

Коментарі необхідні для пояснень призначення тих чи інших частин програми і їх текст завжди ігнорується компілятором. Мова C/C++ використовує два різновиди коментарів:

- // текст – **однорядковий коментар**, який починається з двох символів «/» («коса риска») і закінчується символом переходу на новий рядок;
- /* текст */ – **багаторядковий коментар**, що розташовується між символами-дужками «/*» і «*/».

Багаторядкові коментарі не можуть бути вкладеними один в один, а однорядкові коментарі можна вкладати в багаторядкові коментарі.

Створення змінної

Змінна – це іменована область пам'яті, у якій зберігаються дані визначеного типу. Змінна має ім'я, розмір та інші атрибути, такі як видимість, час існування тощо. Ім'я змінної служить для звертання до області пам'яті, у якій зберігається її значення.

Перед використанням будь-яка змінна повинна бути описана, при цьому для неї резервується деяка область пам'яті, розмір якої залежить від конкретного типу змінної. Під час виконання програми змінна може приймати різні значення.

Загальний вигляд опису змінних:
[клас пам'яті] [const] тип ім'я [ініціалізація];
static int a=5;//створення змінної/об'єкту 'a'.
Спрощений синтаксис створення змінної:
тип ім'я;

Класи пам'яті

Клас пам'яті визначає час існування та область видимості програмного об'єкта, тобто змінної. Якщо клас пам'яті не зазна-

чений явно, то він визначається компілятором (зазвичай як auto), виходячи, з контексту оголошення.

- auto – автоматична змінна, для якої пам'ять виділяється у стеку і за необхідності ініціюється кожного разу при виконанні оператора, що містить її визначення. Звільнення пам'яті відбувається при виході з блока, де описана змінна. Час її існування – з моменту опису до кінця виконання блока. Для глобальних змінних цей специфікатор не використовується, а для локальних він приймається за замовчуванням, тому задавати його явно великого сенсу немає;
- (розпочинаючи зі стандарту C++11 значення auto змінено) extern означає, що змінна визначена в іншому місці програми (в іншому файлі або далі по тексту) і використовується для створення змінних, доступних в усіх модулях програми, де вони оголошені. При ініціюванні змінної у тому ж операторі, специфікатор extern ігнорується;
- static – статична змінна, що має постійний час існування. Вона ініціюється один раз при першому виконанні оператору, що містить визначення змінної. Залежно від розташування оператора, описані статичні змінні можуть бути глобальними і локальними. Глобальні статичні змінні видимі тільки у тому модулі, у якому вони описані;
- register – аналогічний до специфікатора auto, але пам'ять виділяється за можливості в регістрах процесора і за відсутності такої можливості у компілятора змінні обробляються як auto.

Створення константи

- модифікатор const указує, що змінна не може змінювати своє значення, у цьому випадку її називають типізованою (іменованою) константою або просто константою;
- ініціалізація для констант обов'язкова.

Наприклад:
```
const int a = 5;
int const b = 7;
const double pi = 3.14;
```

Глобальні і локальні змінні

Локальна змінна визначена всередині блока. Область її дії обмежена початком опису змінної та кінцем блока, включаючи

усі вкладені блоки. Час «життя» з моменту опису до кінця блоку, у якому вона об'явлена.

```
{
        int a; //локальна змінна
}
```

Блок – це частина коду розташована у фігурних дужках {}.

Глобальна змінна – це змінна, визначена поза будь-яким блоком. Областю її дії вважається файл, у якому вона визначена від початку опису до його кінця. Час «життя» з моменту опису до кінця існування програми.

```
int a; //глобальна змінна
int main()
{……}
```

Приклади створення змінних

```
int d;                  //глобальна змінна d
int main()
{
        int b;          //локальна змінна b
        extern int y;   //змінна, визначена у іншому місці (файлі) програми
        static int s;   //локальна статична змінна s
        d = 1;          //присвоювання значення глобальній змінній
        int d;          //створення локальної змінної d
        d = 10;         //присвоювання значення локальній змінній
        d = 3;          //присвоювання значення глобальній змінній
        return 0;
}
int y = 4;              //визначення і ініціалізація змінної y
```

Представлення змінної/об'єкта в пам'яті

Представлення змінної в пам'яті

```
#include <iostream>
using namespace std;
int main ()
```

```
{
   setlocale(LC_ALL, "Ukr");
   double x= 2.5;
   cout << "Значення змінної/об'єкту = " << x << endl;
   cout << "Адреса змінної/об'єкту = " << &x << endl;
   system("pause");
   return 0;
}
```

Результат роботи:

```
Значення змінної/об'єкту = 2.5
Адреса змінної/об'єкту   = 0x7ffe80a5d250
```

Операція розширення області видимості :: . Доступ до глобальних змінних всередині функцій

Під час розробки власних програм, що містять глобальні змінні може виникнути ситуація коли всередині функції використовується локальна змінна з таким самим іменем. У цьому випадку ім'я локальної змінної перекриває ім'я глобальної змінної. При звертанні до імені всередині функції береться до уваги локальна змінна (глобальна змінна ігнорується). Якщо виникла така ситуація, то для доступу до глобальної змінної можна використати операцію розширення області видимості, яка позначається **::** .

За допомогою операції розширення області видимості **::** можна отримати доступ не тільки до змінних, але й до констант і функцій, які ви докладно будете вчити у темі «Функції».

Приклад:
використання локальних і глобальних змінних;
використання прототипу функції;
використання операції розширення області видимості :: для доступу до глобальної змінної

```
#include <iostream>
using namespace std;

// Глобальні та локальні змінні. Операція ::

// Оголошення глобальної змінної d.
// Ця змінна є доступна в обох функціях: MultGlobal2() та main()
int d = 15;
```

```
// Прототип функції, яка множить глобальну змінну d на 2
int MultGlobal2(void);

int main()
{
  // Оголошення локальної змінної d у функції main()
  int d;

  // Виведення значення локальної змінної d
  d = 25;

  cout << "Local: d = " << d << endl; // d = 25 – локальна змінна перевизначає глобальну змінну

  // Виведення значення глобальної змінної d,
  // використовується операція :: - розширення області видимості
  cout << "Global: d = " << ::d << endl;

  // Вивід подвоєного значення глобальної змінної d
  cout << "Global: d*2 = " << MultGlobal2() << endl; // d*2 = 30

  // Запис в локальну змінну d нового значення 27
  d = 27;
  cout << "Local: d = " << d << endl;

  // Запис в глобальну змінну d нового значення 68
  ::d = 68;
  cout << "Global: d = " << ::d << endl; // глобальна змінна d = 68

  // Виведення подвоєного значення глобальної змінної ::d
  cout << "Global d * 2 = " << MultGlobal2() << endl; // буде виведено 136
}

// Функція, яка використовує значення глобальної змінної d.
// До локальної змінної d у функції main() ця функція доступу не має
int MultGlobal2(void)
```

```
{
  // тут доступна глобальна змінна d
  return d * 2;
}
```
Результат роботи:
```
Local: d = 25
Global: d = 15
Global: d*2 = 30
Local: d = 27
Global: d = 68
Global d * 2 = 136
```

Типи даних мови С++

- **int** (цілий);
- **char** (символьний);
- **bool** (логічний);
- **float** (дійсний);
- **double** (дійсний із подвійною точністю);
- **void** (порожній, не має значення).

Типи **int, char, bool** називають **цілими**, а типи **float** та **double** – ***дійсними з плаваючою крапкою***. Код, що формує компілятор для обробки цілих величин, відрізняється від коду для величин із плаваючою крапкою.

Для уточнення внутрішнього подання і діапазону значень стандартних типів **мова С++ використовує чотири специфікатори типу:**

- **short** (короткий);
- **long** (довгий);
- **signed** (знаковий);
- **unsigned** (беззнаковий).

*З*ауваження: символьний (char) та логічний (bool) тип також належать до цілого.

Зауваження: Нові версії компіляторів мають більш розширені типи, наприклад, long long, wchar_t.

Цілі змінні (типу **int, long, short**) необхідні для збереження цілих значень і можуть бути знаковими і беззнаковими. Знакові змінні застосовують для подання як додатних, так і від'ємних

чисел, при цьому один біт (найстарший) виділяється під знак. Для оголошення беззнакової змінної, тобто змінної, що приймає тільки додатні значення, необхідно використовувати ключове слово **unsigned**. За замовчуванням будь-який цілий тип уважається знаковим, і тому немає потреби у використанні ключового слова **signed**.

Символьний тип даних **char** застосовується у випадку, коли змінна містить інформацію про код ASCII або для побудови таких більш складних конструкцій, як рядки, символьні масиви тощо. Дані типу **char** також можуть бути знаковими і беззнаковими.

Змінна типу bool займає 1 байт і використовується, насамперед, у логічних операціях, тому що може приймати значення **0** (**false** – «неправда») або відмінне від нуля (**true** – «істина»). У випадку перетворення до цілого типу **true** має значення 1.

Стандарт C++ визначає три типи даних для збереження **дійсних значень змінних**: **float, double** та **long double** (типи **з плаваючою крапкою**). Тип **float**, зазвичай, використовують для збереження не дуже великих дробових чисел.

Змінна типу void не має значення, оскільки множина значень цього типу порожня. Такі змінні необхідні для узгодження синтаксису. Тип **void** використовується для визначення функцій, що не повертають значення, для вказівки порожнього списку аргументів функції, а також як базовий тип для покажчиків і в операції приведення типів. Наприклад, якщо немає потреби у використанні поверненого значення функції, перед ім'ям функції ставиться тип **void**.

Машинне слово – машинно-залежна і платформозалежна величина, яка вимірюється в бітах або байтах (тритій або трайтен), рівна розрядності регістрів процесора і/або розрядності шини даних (зазвичай деяка ступінь двійки).

Байт (англ. Byte) – одиниця зберігання і обробки цифрової інформації; сукупність бітів, що обробляється комп'ютером одномоментно. У сучасних обчислювальних системах байт складається з восьми бітів і, відповідно, може приймати одне з 256 (28) різних значень (станів, кодів). Однак в історії комп'ютерної техніки існували рішення з іншими розмірами байту (наприклад, 6, 32 або 36 бітів), тому іноді в комп'ютерних стандартах і офіційних документах для однозначного позначення групи з 8 бітів використовується термін «октет» (лат. octet). У більшості обчис-

лювальних архітектур байт – це мінімальний незалежно адресований набір даних.

Біт – одиниця виміру кількості інформації. Може приймати тільки два взаємовиключних значення: «так» або «ні», «1» або «0», «включено» або «вимкнено».

Базові типи даних для ПК на базі платформи Intel

Тип	Розмір, байт	Значення
bool	1	true (1) або false(0)
unsigned short int	2	від 0 до 65 535
short int	2	від –32 768 до 32 767
unsigned long int	4	від 0 до 4 294 967 295
long int	4	від –2 147 483 648 до 2 147 483 647
int (16 розрядів)	2	від –32 768 до 32 767
int (32 розряда)	4	від –2 147 483 648 до 2 147 483 647
unsigned int (16 розрядів)	2	від 0 до 65 535
unsigned int (32 розряда)	4	від 0 до 4 294 967 295
char	1	від 0 до 256
float	4	від 1.2e-38 до 3.4e38
double	8	від 2.2e-308 до 1.8e308
long double	змінний, не менше 8 байт	від 2.2e-308
void	2 або 4	-

Наприклад:

bool dd = true;	логічна змінна dd=1;
bool dd1 = 1;	логічна змінна dd1=true;
int a = 1, b = 0;	цілі змінні a = 1, b = 0;
char sim =’A’;	символьна змінна sim =’A’;
float Age = 18.5;	десяткова змінна Age = 18.5 з крапкою, що плаває;
void MyFunction();	функція не вертає ніякого значення, тощо.

Константи відрізняються від змінних тим, що значення, що присвоєне константі з початку, не може бути змінено протягом виконання всієї програми. Оголошення констант виконується за такою схемою:

const [тип] ідентифікатор = значення;

Приклади оголошення констант.

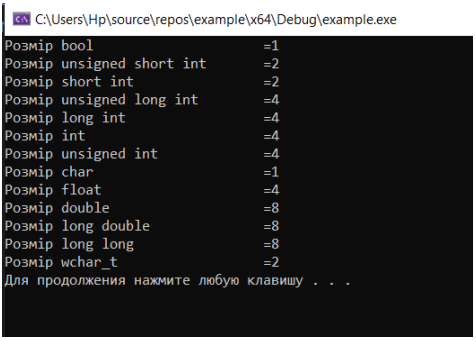
```
const float Pi = 3.1415; // Число Пі
const int Max = 1000;    // константа з іменем Max
int const Min = 100;     // константа з іменем Min
const char New_Line = '\n';
const bool YES = true;
```

Програма яка друкує розмір типу в байтах

Розмір типу в байтах

```
#include <iostream>
using namespace std;
int main()
{
      setlocale(LC_ALL, "Ukr");
      cout << "Розмір bool \t\t\t=" << sizeof( bool ) << endl;
      cout << "Розмір unsigned short int \t=" <<sizeof(unsigned
short int) << endl;
      cout << "Розмір short int \t\t="<<sizeof(short int)<<endl;
      cout << "Розмір unsigned long int \t=" << sizeof(unsigned
long int) <<endl;
      cout << "Розмір long int \t\t=" << sizeof(long int) << endl;
      cout << "Розмір int \t\t\t="<<sizeof(int ) << endl;
      cout << "Розмір unsigned int \t\t=" << sizeof(unsigned int)
<< endl;
      cout << "Розмір char \t\t\t=" << sizeof(char) << endl;
      cout << "Розмір float \t\t\t=" << sizeof(float) << endl;
      cout << "Розмір double \t\t\t=" << sizeof(double) << endl;
      cout << "Розмір long double \t\t=" << sizeof(long double)
<< endl;
      cout << "Розмір long long \t\t=" << sizeof(long long) << endl;
      cout << "Розмір wchar_t \t\t\t=" << sizeof(wchar_t) << endl;
      system("pause");
      return 0;
}
```

Результат роботи програми



Логічні оператори

Умовні оператори

- **if**
- **switch**
- **?:**

Логічні операції

Операції відношення порівнюють значення лівого операнда зі значенням правого операнда:

< – менше;

<= – менше або дорівнює (не перевищує);

== – дорівнює;

> – більше;

>= – більше або дорівнює (не менше);

!= – не дорівнює.

У мові C++ «істина» – це ненульова величина, «неправда» – це нуль **(0)**. Усі не нульові значення це істина.

Операції відношення повертають ціле значення 1, якщо умова правильна, або 0, якщо умова помилкова.

Логічні операції оперують з цілими розмірами або з розмірами, які можна перетворити на цілі. Обчислення зупиняється, як тільки визначиться, чи є вираз правдивим («істина»), або помилковим («неправда»). При цьому, як і для операцій відношення, значенням «істина» відповідає 1, а значенням «неправда» – 0.

&& – логічне **«AND»** (кон'юнкція);

|| – логічне **«OR»** (диз'юнкція);

! – логічне **«NOT»** (заперечення).

Результат операції «&&» є «істина» **(1)**, якщо обидва її операнди правдиві (не рівні 0). Результат операції «||» – «істина»**(1)**, якщо хоча б один з її операндів є «істина». Логічне заперечення «!=» перетворює свій операнд на «істину» **(1)**, якщо він дорівнює **0**, і на «неправду» **(0)**, якщо він не дорівнює **0**.

Операції обробки окремих бітів застосовують для обробки даних як послідовностей бітів (розрядів), кожний з яких набуває значення **0** або **1**.

& – операція бітового множення (кон'юнкція);

| – операція бітового додавання (диз'юнкція);

^ – додавання за модулем 2;

~ – інвертування;

>> – зсув праворуч;

<< – зсув ліворуч.

Оператор if

Загальний запис умовного оператора if має такий синтаксис

if (вираз) оператор 1;

else оператор 2;

Якщо значення виразу «істина» (не нуль), то виконується **оператор1**, якщо ж воно хибне, то виконується **оператор2.**

Гілка else не обов'язкова і може бути відсутня.

Рекомендований синтаксис:

```
if (вираз)
{
Оператори
}
else
{
Оператори
}
```

Використання умовних операторів

Обчислити значення виразу

$$y = \begin{cases} e^{x^a} \\ x^{e^a} \end{cases} \text{ де } x > 0, a > 0$$

```
#include <iostream>
#include <cmath>
```

```
#include <cstdlib>
using namespace std;
int main()
{
	double a, x;
	cout<<"Please input a =";
	cin >> a;
	cout<<"Please input x =";
	cin >> x;
	double y;
	if (x > 0 && a>0)
	{
		y = exp(pow(x,a));
	}
	else
	{
		y = pow(x,exp(a));
	}
	cout<< "Result: y ="<<y<<endl;
	system("pause");
	return 0;
}
```

Складне розгалуження if

Складне розгалуження безпосередньо мовою С не підтримується. Воно реалізовується як комбінація простих розгалужень. Рекомендований синтаксис складного розгалуження

```
if (вираз)
{
Оператори
}
else if (вираз)
{
Оператори
}
else if (вираз)
{
Оператори
}
```

……
else
{
Оператори
}

Використання умовних операторів

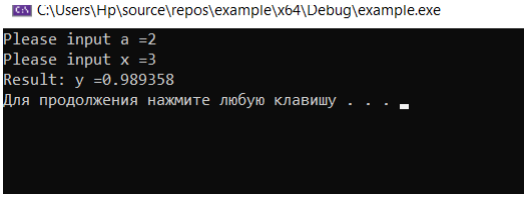
Обчислити значення виразу для введених з клавіатури значень для a і x.

$$y = \begin{cases} sin\left|a^x\right|, & a > 0 \text{ і } x > 0, \\ cos\left|e^x\right|, & a > 0 \text{ і } x < 0 \\ sin(x) * cos(a) \end{cases}$$

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
int main()
{
	double a, x;
	cout << "Please input a =";
	cin >> a;
	cout << "Please input x =";
	cin >> x;
	double y;
	if (a > 0 && x>0)
	{
		y = sin(fabs(pow(a,x)));
	}
	else if (a > 0 && x<0)
	{
		y = cos(fabs(exp(x)));
	}
	else
	{
		y = sin(x)*cos(a);
	}
	cout << "Result: y =" << y << endl;
```

```
        system("pause");
        return 0;
    }
```

Результат роботи програми



Оператори continue і break

Іноді виникає необхідність вийти з циклу до його завершення. У цьому випадку можна скористатися оператором break.

Використання умовних операторів і оператора break

Надрукувати квадрати чисел від 1 до 9.

Тут коли значення змінної і досягне 10, здійснюється вихід із циклу за допомогою оператора break.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
int i = 1 ;
for ( ; ; )
{
cout << i << " * " << i << " = " << i * i <<  endl ;
i ++;
if (i > 9) break ;
}
        system("pause");
        return 0;
}
```

На відміну від оператора break, оператор continue проводить перехід до наступної ітерації.

Використання умовних операторів і оператора continue
Порахувати суму непраних чисел з певного діапазону.

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
int result = 0;
for ( int i =0; i < 10; i++)
{
if ( i % 2 == 0 ) continue; // перевірка на парність
result +=i ;
}
cout << " result = " << result << endl; // 25
      system("pause");
      return 0;
}
```

Щоб взнати, чи парне число, ми одержуємо залишок від цілочисельного ділення на 2, і якщо він рівний 0, то за допомогою оператора continue переходимо до наступної ітерації циклу. А якщо число непарне, то додаємо до решти непарних чисел.

Оператор вибору switch загальний синтаксис

Загальний синтаксис умовного оператора вибору **switch** має такий вигляд

```
switch (вираз)
{
      case константне_значення_1: оператори
      case константне_значення_2: оператори
……
      константне_значення_N: оператори
      default: оператори
}
```

Оператор switch порівнює значення виразу з набором константних значень. Якщо відповідність знайдена то будуть виконуватися усі оператори з відповідного case до кінця оператора switch або поки не «зустріне» оператор break. Якщо відповідність не знайдена, то буде виконуватися гілка default, якщо вона присутня, оскільки вона не обов’язкова.

Оператор вибору switch рекомендований синтаксис

Якщо необхідно реалізувати для кожного case виконання тільки одного набору операторів, то необхідно в кінці кожного case поставити оператор break. Рекомендований синтаксис виглядає так (кожна гілка оформлена додатковим блоком):

```
switch (вираз)
{
    case константне_значення_1:
{
деякий код
break;
}
    case константне_значення_2:
{
деякий код
break;
}
……
    константне_значення_N:
{
деякий код
break;
}
    default:
{
деякий код
}
}
```

Використання умовних операторів (switch)

Для введених цілих чисел від 1 до 5 надрукувати повідомлення «val in [1;5]», для чисел 7 і 8 надрукувати повідомлення «val is 7 or 8». У інших випадках надрукувати «Another number».

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
int main()
```

```
{
        int val;
        cout << "Please input val = ";
        cin >> val;
        switch (val)
        {
                case 1:
                case 2:
                case 3:
                case 4:
                case 5:
                        cout<< "val in [1;5]" << endl;
                        break;
                case 7:
                case 8:
                        cout<< "val is 7 or 8" << endl;
                        break;
                default : cout<< "Another number" << endl;
        }
        system("pause");
        return 0;
}
```

Тернарна операція ?:

Загальний синтаксис тернарної операції має такий синтаксис **(вираз 1)? Вираз2: вираз3**

Вираз 1 задає умову, якщо вираз1 істинний, то буде підставлений вираз 2, у іншому випадку буде підставлений вираз 3.

Наприклад, якщо необхідно знайти мінімальне значення серед змінних x та y, то це можна реалізувати так:

```
min = (x<y)? X : y;
```

у результаті min отримає значення x якщо воно менше y, або y якщо умова y дужка[не виконається.

Примітка: конструкція тернарного оператора повністю повторює конструкцію умовного оператора if else (**?:**).

Аналогічне вирішення завдання находження мінімума через умовний оператор if має такий вигляд:

```
if (x<y)
                min = x;
        else
                min = y;
```

Використання умовних операторів

Обчислити значення виразу, через тернарний оператор

$$y = \begin{cases} e^{x^a} \\ x^{e^a} \end{cases}, \text{ де } x > 0, a > 0$$

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
int main()
{
        double a, x;
        cout<<»Please input a =»;
        cin >> a;
        cout << «Please input x =»;
        cin >> x;
        double y  =(x > 0 && a>0)? Exp(pow(x,a)) :
pow(x,exp(a));
        cout << «Result: y =» << y << endl;
        system(«pause»);
        return 0;
}
```

Оператори циклу

У мові С++ існують три оператори циклу: **while, for, do.**

Цикл із передумовою while

Оператор циклу з передумовою:

while (A) оператор;

– будь-який простий, складений чи порожній оператор, тут **A** – будь-який припустимий вираз. Виконується цей оператор так: якщо результат виразу А не дорівнює нулю («істина»), то виконується цикл, а якщо дорівнює нулю (хибний), то цикл не виконується і керування передається наступному за **while** оператору.

Рекомендований синтаксис

```
while (вираз-умова)
{
оператори
}
```

Цикл for

Загальний синтаксис оператора циклу for
for ([блок1] ; [блок2-умова]; [блок3-дія])
{оператор;}

де **блок 1** – це створення чи оновлення змінних (зазвичай так званих лічильників) циклу, що зазвичай використовується для встановлення початкового значення (необов'язковий параметр);

блок 2 – це вираз умови, що визначає, за якої умови цикл буде повторюватися (необов'язковий параметр);

блок 3 – це вираз, який зазвичай (хоча не обов'язково) задає зміну лічильника циклу (необов'язковий параметр).

Алгоритм виконання оператора **for:**

Одноразово виконується блок 1, у якому створюються або оновлюються початковим значенням змінні-лічильники циклу. Тобто цей блок виконується тільки один раз перед першою ітерацією циклу.

Виконується блок 2. Тобто перевіряється умова (чи є вираз істинним). Умова істинна, якщо результат виразу не дорівнює нулю.

Якщо блок 2 істинний, то виконується оператор (тіло циклу) і блок 3. Після чого знову виконується блок 2. Це повторюється до тих пір, поки вираз блоку 2 буде істинний.

Якщо блок 2 прийме значення «неправда» (вираз-умова дорівнює нулю), тоді буде реалізований вихід із циклу.

Оскільки перевірка умови виконується перед циклом, то цикл може не виконатися жодного разу, якщо умова відразу буде помилковою.

Оператор for рекомендований синтаксис:

```
for ( [ініціалізація] ; [вираз-умова];  [вираз-дія] )
{
деякий код/оператори;
}
```

Оскільки перевірка умови виконується перед циклом, то цикл може не виконатися жодного разу, якщо умова відразу буде помилковою.

Цикл do while

Оператор циклу **do** зазвичай використовується в тих випадках, коли тіло циклу повинне виконуватися хоча б один раз, і має таку структуру запису:

do оператор
while (вираз умова);

Виконується оператор **do** так: спочатку здійснюється вхід у тіло циклу і виконується оператор (він може бути простий чи складний), після цього перевіряється умова і, якщо вона виконується, тобто «істина» (не дорівнює нулю), то цикл повторюється, а якщо «неправда» – здійснюється вихід з циклу.

Рекомендований синтаксис

```
do
{
 оператори
}
while (вираз умова);
```

Виконується оператор **do** таким чином:
спочатку здійснюється вхід у тіло циклу і виконуються оператори (їх може буде декілька, після цього перевіряється умова і, якщо вона виконується, тобто «істина» true (не дорівнює нулю), то цикл повторюється, а якщо «неправда» false – здійснюється вихід з циклу.

Використання циклу for
Вивести на екран усі цілі числа від –10 до 10

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
   for (int i = -10; i <= 10; ++i)
   {
      cout << i <<" " ;
   }
   cout << endl;
   system("pause");
   return 0;
}
```

Використання циклу for

Вивести на екран усі числа від –2 до 2 з кроком 0.5

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{

   for (double x = -2; x <= 2; x += 0.5)
   {
      cout << x << "\n";
   }
   cout << endl;
   system("pause");
   return 0;
}
```

Використання циклу do while

Обчислити вираз $y = sin^x(x)$, для деякого x введеного з клавіатури.

Забезпечити повторне обчислення виразу для різних x не виходячи з програми

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;
int main()
{
   double x,y;
   char ch;
   do
   {
      cout<<"Input x = ";
      cin >> x;
      y=pow(sin(x), x);
      cout << "Result y= = " << y << endl;
      cout << "Continue (y/n)? ";
      cin >> ch;
   }while (ch == 'y' || ch == 'Y');
```

```
        system("pause");
      return 0;
}
```

Використання циклу for

Знайти суму всіх парних чисел від парного a до парного *b* (*a* < *b*)

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
int sum=0, a, b, i;
cout << "Input a, a=";
cin >> a;
cout << "\nInput b, b=";
cin >> b;
for (i=a; i<b; i+=2)
  sum +=i;
cout << "Sum is equal, Sum = " << sum;
cout << "\n";
system("pause");
return 0;
}
```

Використання циклу for

Написати програму для обчислення значення функції при різних значеннях аргументів, заданих інтервалом зміни і величиною кроку

$$z=\begin{cases}\dfrac{x^2}{(x-5)^3}, & x>y;\\ \dfrac{(x-2)^3}{y(x-5)^4}, & x\le y;\end{cases}$$

$$x\in[1,10];\ h_x=2;\ y\in[-4,3];\ h_y=1;$$

```
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

int main()
{
  long double z;
   int x, y;
   for (x = 1; x <= 10; x += 2)
{
      for (y = -4;  y <=3;  y++)
       {
       if  (x > y)
   { z= x*x / pow(x-5, 3);
     cout << "z= " << z << endl;
   }
    else
          { z = pow(x-2, 3) / (y * pow(x-5, 4));
 cout << "z= " << z << endl;}
    }
      }
   return 0;
}
```

Використання циклу for

Задані два цілих числа a і b, а менше b. Знайти суму остач від ділення на задане число k всіх цілих чисел на проміжку від a до b включно

```
#include <iostream>
using namespace std;
int main()
{
   int a = 10;
   int b = 25;
   int k;
   double n;
   double s=0;
```

```
    cin >> k;
    for (int i=a; i <= b; i++)
    {
       n = i % k;
       s += n;
    }
       cout << s;
    return 0;
}
```

Далі представлено різні варіанти (1–6) використання циклів та умовних операторів для однієї задачі.

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$. Верхню межу суми (n) користувач вводить з клавіатури

1. Використання циклу while

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$

Верхню межу суми (n) користувач вводить з клавіатури

```
#include <iostream>
using namespace std;
int main()
{
    int n = 10, S = 0, k = 1;
        while (k <= n) {
           S = S + k * k;
           k++;
           cout << "Sum of squared from 1 to " << n << " : " <<
S <<  endl;
           return 0;
        }
}
```

2. Використання циклу while

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$

Верхню межу суми (n) користувач вводить з клавіатури

```
#include <iostream>
using namespace std;
```

```
int main()
{
int n, S = 0;
      cout << " Input last number n : ";
      cin >> n;
      while (n) {
         cout << " Sum of squared" << S << endl;
         S += n * n;
         n--;
      }
            return 0;
         }
```

3. Використання циклу while та умовних операторів if else

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$

Верхню межу суми (n) користувач вводить з клавіатури

```
#include <iostream>
using namespace std;
int main()
{
   int n, S = 0;
   cout << " Input last number n : ";
   cin >> n;
   if (n > 0) {
      while (n) {
         cout << " Sum of squared : " << S << endl;
         S += n * n;
         n--;
      }
   } else
{
      cout << " false " << endl;
   }
   return 0;
}
```

4. Використання циклу for 1 спосіб

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$

Верхню межу суми (n) користувач вводить з клавіатури

```
#include <iostream>
using namespace std;
int main()
{
   int n, S, k;
   cout << " Input last number n : ";
   cin >> n;
   for (k = 1, S = 0; k <= n; S += k * k, k++);
         cout << "Sum of squared : " << S << endl;
   return 0;
}
```

5. Використання циклу for 2 спосіб

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$

Верхню межу суми (n) користувач вводить з клавіатури

```
#include <iostream>
using namespace std;
int main()
{
   int n, S = 0, k = 1;
   cout << " Input last number n : ";
   cin >> n;
   for ( ; k <= n ; )
   {
   S += k * k;
   k++;
}
    cout << "Sum of squared : " << S << endl;
   return 0;
}
```

6. Використання циклу for 3 спосіб

Написати програму для обчислення суми квадратів послідовних цілих чисел $1^2+2^2+3^2+\ldots n^2$

Верхню межу суми (n) користувач вводить з клавіатури

```
#include <iostream>
using namespace std;
int main()
{
   int n, S = 0, k = 1;
   cout << " Input last number n : ";
   cin >> n;
   for ( ;  ; )
   {
   S += k * k;
   k++;
   if (k > n) {
       break; }
   }
   cout << " Sum of squared " << S << endl;
   return 0;
}
```

Тема 2. Функції

Функція – це об'єднання деякого коду в один об'єкт.

Функції дозволяють розбити програму на логічні частини (функціональна декомпозиція), що забезпечує легше супроводження програми та її розуміння.

Функції доцільно створювати коли деякий фрагмент коду:

- має логічну завершену думку;
- можна дати ім'я (наприклад, «Вивід масиву»);
- дублюється.

Математичні бібліотечні функції

Математичні функції заголовного файлу <cmath>

Прототип функції	Ім'я	Зміст
double acos (double _x);	**acos (x)**	Арккосинус
double asin (double _x);	**asin (x)**	Арксинус
double atan (double _x);	**atan (x)**	Арктангенс

double atan2 (double _y, double _x);	**atan2 (y, x)**	Арктангенс від y/x
double ceil (double _x);	**ceil (x)**	Округлення в більшу сторону
double cos (double _x);	**cos (x)**	Косинус x, в радіанах
double cosh (double _x);	**cosh (x)**	Косинус, гіперболічний
double exp (double _x);	**exp (x)**	$e^{x,}$ e у ступені x
double fabs (double _x);	**fabs (x)**	Абсолютне значення |x| типу double
double floor (double _x);	**floor (x)**	Повертає найближче ціле, не більше x
double fmod (double _x, double_y);	**fmod (x)**	Залишок від ділення x на y
double log (double _x);	**log (x)**	Натуральний логарифм
double log10 (double _x);	**log10 (x)**	Десятковий логарифм
double pow (double _x, double_y);	**pow (x, y)**	x^y, x у степені y
double sin (double _x);	**sin (x)**	Синус x, в радіанах
double sinh (double _x);	**sinh (x)**	Синус, гіперболічний
double sqrt (double _x);	**sqrt (x)**	Корінь з x, x>0
double tan (double _x);	**tan (x)**	Тангенс x, x у радіанах
double tanh (double _x);	**tanh (x)**	Тангенс, гіперболічний
int abs (int _x);	**abs (x)**	Модуль x типу int
double atof (const char*_s);	**atof (s)**	Перетворює рядок символів у число з плаваючою комою
double hypot (double_x, double_y);	**hypot (x, y)**	Корінь із (x^2+y^2)
long labs (long _x);	**labs (x)**	Абсолютна величина типу long |x|
double pow10 (int _p)	**pow10 (p)**	Повертає 10^p

Генерація випадкових чисел

Генератор псевдорандомних чисел – це програма, яка приймає стартове/початкове значення і виконує з ним певні математичні операції, щоб конвертувати його в інше число, яке зовсім не пов'язане зі стартовим. Потім програма використовує

нове згенероване значення і виконує з ним ті ж математичні операції, що і з початковим числом, щоб конвертувати його ще в одне нове число – уже третє, яке не пов'язане ні з першим, ні з другим числом. Застосовуючи цей алгоритм до останнього згенерованого значення, програма може генерувати цілий ряд нових чисел, які здаватимуться випадковими (за умови, що алгоритм буде досить складним).

Функції srand() і rand()

Мови Сі та C++ мають свої власні вбудовані генератори випадкових чисел. Вони реалізовані в двох окремих функціях, які знаходяться в **файлі заголовка** cstdlib:

Функція srand() встановлює передане користувачем значення як стартове. srand() слід викликати тільки один раз – на початку програми (зазвичай у верхній частині функції main()).

Функція rand() генерує наступне випадкове число в послідовності. Воно знаходитиметься в діапазоні від 0 до RAND_MAX (константа в cstdlib, значенням якої є 32767).

Щоб згенерувати рандомне число рішенням є використання системного годинника. Кожен раз, при запусканні програми, час буде інший. Якщо ми будемо використовувати значення часу як стартове число, то наша програма завжди генеруватиме різну послідовність чисел при кожному новому запуску!

У мові Сі є **функція time()**, яка повертає в якості часу загальну кількість секунд від півночі 1 січня 1970 року і до сьогоднішнього часу. Щоб скористатися цією функцією, нам просто потрібно підключити заголовок ctime, а потім ініціалізувати функцію srand() викликом функції srand(time(NULL));.

Створення функції

Об'явлення (оголошення, сигнатура, прототип) функції – це заголовок функцій без тіла.

Описання (визначення) функції – це реалізація функції, тобто опис дій, що виконуються функцією – вихідний код

Виклик функції – це місце коду де функція застосовується з фактичними параметрами.

До виклику функції вона повина бути об'явлена або визначена. Причому функція може бути об'явлена безліч разів, а описана тільки один.

Спрощений синтаксис

Синтаксис оголошення:

тип_результату ім'я_функції (список формальних аргументів);

Список формальних аргументів можна називати параметрами функції

Синтаксис описання функції:

тип_результату ім'я_функції (параметри функції)

{

// тіло функції

return [вираз_відповідний_до_типу_результату];

}

тип_результату – це довільний тип (int, double, char, bool…), який може повернути функція. Також існує спеціальний тип для функцій void. Він означає що функція нічого не вертає. У цьому випадку оператор return може бути відсутній, або присутній без параметра: return; наприклад, для дострокового виходу із функції за потреби. Якщо параметри функції відсутні, то залишають пусті дужки (), або з ключовим словом void: (void).

Приклад функції типу void

Створити функцію, яка друкує просте текстове повідомлення

```
#include <iostream>
using namespace std;
void hi()
{
      cout << "Hello world!\n";
}
 int main()
{
      hi();//виклик функції
      hi();//повторний виклик функції
      system("pause");
      return 0;
}
```

Результат роботи:

Hello world!

Hello world!

У цьому прикладі функція нічого не вертає, а також відсутні параметрі. Функцію викликаємо двічі, тому повідомлення на екрані повторюється.

Приклад функції яка повертає значення.
Варіант 1. Функція оголошена перед функцією main, а описана після.
Створити функцію, яка обчислює різницю для двох цілих чисел.

```
#include <iostream>
using namespace std;
int sub(int a, int b);//оголошення функції
int main()
{
    cout<<sub(7,5)<<endl;/*виклик функції з фактичними параметрами 7 та 5 */
    int x = 8, y = 9;
    cout << sub(x, y) << endl;/*виклик функції з фактичними параметрами x та y*/
    system("pause");
    return 0;
}
int sub(int a, int b) //опис функції з формальними параметрами a та b
{
    return a-b;
}
```

Результат роботи

```
2
-1
```

Приклад функції яка повертає значення.
Варіант 2. Функція оголошена й описана перед функцією main.
Створити функцію, яка обчислює різницю для двох цілих чисел.

```
#include <iostream>
using namespace std;
```

```
int sub(int a, int b);//оголошення функції
int sub(int a, int b) // опис функції з формальними параметрами a та b
{
    return a-b;
}

int main()
{
    cout<<sub(7,5)<<endl;/*виклик функції з фактичними параметрами 7 та 5 */
    int x = 8, y = 9;
    cout << sub(x, y) << endl;/*виклик функції з фактичними параметрами x та y*/
    system("pause");
    return 0;
}
```

Результат роботи

```
2
-1
```

Приклади найпростіших функцій

```
// Сума двох цілих чисел
#include <iostream>
#include <cmath>
using namespace std;
int Sum(int a, int b) {
int result;
result = a + b;
return result; }
int main() {
Sum(2, 5);
cout << Sum(2, 5) << endl;
}
```

Результат роботи

```
7
```

```
// Функція типу void
#include <iostream>
#include <cmath>
using namespace std;
void foo(int a) {
a++; }
int main() {
   int a = 1;
   foo(a);
   cout << a << endl;
}
```

Результат роботи

```
1
```

```
// Функція типу void
#include <iostream>
#include <cmath>
using namespace std;
void foo(int a)
{ a++; }
int main() {
int value = 1;
   foo(value);
   cout << value << endl;
}
```

Результат роботи

```
1
// Функція типу int

#include <iostream>
#include <cmath>
using namespace std;
int foo(int a)
{ return a++; }
int main() {
int value = 1;
   value = foo(value);
   cout << value << endl;
}
```

Результат роботи

```
1
```

Приклад функції, яка повертає значення типу double.
Створити функцію, яка обчислює відсоток за вкладами (за депозитом) залежно від кількості років

```
#include <iostream>
#include <cstdlib>
using namespace std;
double getMoney(double m, double r, int y) {
if (y == 0)
{ return m; }
else { return(1 + r/100) * getMoney(m, r, y-1);} }
int main() {
    double money = 1000;
   double rate = 5;
cout << "initial Sum: " << money << endl;
   cout << "interest Sum: " << rate << "%\n";
   cout << "money for 1 year: " << getMoney(money, rate, 1)
<< endl;
   cout << "money for 7 year: " << getMoney(money, rate, 7)
<< endl;
   cout << "money for 10 year: " << getMoney(money, rate, 10)
<< endl;
   return 0;
}
```

Результат роботи

```
money for 1 year: 1050
money for 7 year: 1407.1
money for 10 year: 1628.89
```

Способи передавання параметрів у функцію

Синтаксично існує три спрособи передачі параметрів у функцію:

- за значенням (by value): void f1(int a);
- за вказівником (by pointer): void f2(int* a);
- за посиланням (by reference): void f3(int& a);

При передаванні параметрів за значенням, в стек заноситься копія об'єкта і функція працює з копією, що ніяк не позначається на оригінальному об'єкті. При передаванні параметра за покажчиком чи посиланням в стек заноситься адреса об'єкта і функція через цю адресу напряму працює з оригінальним об'єктом. Будь-які зміни об'єкта у функції позначаються на об'єкті, який передавався.

Приклад передавання параметрів у функцію

```
#include <iostream>
using namespace std;
void f1(int a) //за значенням
{
   ++a;
   cout << "New value a in f1 = " << a << endl;
}
void f2(int* a) //за покажчиком (вказівником)
{
   ++(*a);
   cout << "New value a in f2 = " << *a << endl;
}
void f3(int& a) //за посиланням
{
   ++a;
   cout << "New value a in f3 = " << a << endl;
}
int main()
{
   int x = 5;
   f1(x);
   cout << "x = " << x << endl;
   x = 5;
   f2(&x);
   cout << "x = " << x << endl;
   x = 5;
   f3(x);
   cout << "x = " << x << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
New value a in f1 = 6
x = 5
New value a in f2 = 6
x = 6
New value a in f3 = 6
x = 6
```

Приклад передавання параметрів у функцію (swap) за посиланням (&)

```
#include <iostream>
#include <cstdlib>
using namespace std;
void swap(char &a, char &b) {
   cout << " виклик функції swap() " << endl;
   cout << " first argument " << a << endl;
   cout << " second argument " << b << endl;
   char t = b;
   b = a;
   a = t;
   for (int i = 1; i <= 20; i++) {
      cout << " - ";
   }
   cout << endl;
   cout << " first argument " << a << endl;
   cout << " second argument " << b << endl;

   cout << " function swap ends" << endl; }
   int main() {
      char x = 'A', y = 'B';
      cout << " first change " << x << endl;
      cout << " second change " << y << endl;
      swap(x, y);
      cout << " first change " << x << endl;
      cout << " second change " << y << endl;
      return 0;
   }
```

Результат роботи

```
first change A
 second change B
 виклик функції swap()
first argument A
 second argument B
 - - - - - - - - - - - - - - - - - - - -
first argument B
 second argument A
function swap ends
first change B
 second change A
```

Приклад передачі параметрів у функцію (swap) за вказівником*

```
#include <iostream>
#include <cstdlib>
using namespace std;
void swap(char *a, char *b) {
   cout << " виклик функції swap() " << endl;
   cout << "first argument " << *a << endl;
   cout << " second argument " << *b << endl;
   char t = *b;
   *b = *a;
   *a = t;
   for (int i = 1; i <= 20; i++) {
      cout << " - ";
   }
   cout << endl;
   cout << "first argument " << *a << endl;
   cout << " second argument " << *b << endl;
   cout << "function swap ends" << endl; }
int main() {
   char x = 'A', y = 'B';
   cout << "first change " << x << endl;
   cout << " second change " << y << endl;
   swap(x, y);
   cout << "first change " << x << endl;
   cout << " second change " << y << endl;
   return 0;
}
```

Результат роботи

```
first change A
 second change B
first change B
 second change A
```

Приклад передавання тексту у функцію

```
#include <iostream>
#include <cstdlib>
#include <math.h>
```

```
using namespace std;
int getLength(const char* str) {
   int s = 0;
   for( int i = 0; str[i]; i++) {
      s++; }
   return s; }
   int getSpace(const char* str) {
   int s = 0;
   for(int i = 0; str[i]; i++) {
      if(str[i] == ' ') {
         s++;
      }
   }
      return s; }
      void show(const char* str)
      {
   cout << "Sentence: " << str << endl;
   cout << "Symbols: " << getLength(str) << endl;
   cout << "Spaces: " << getSpace(str) << endl;
   for(int k = 0; k <= 32; k++) {
            cout << "-"; }
         cout << endl;
      }
int main () {
char txt[100] = "studing programming language in c++" ;
show(txt);
show("in c++ we have a classes and objects") ;
  return 0; }
```

Результат роботи

```
Sentence: studing programming language in c++
Symbols: 35
Spaces: 4
--------------------------------
Sentence: in c++ we have a classes and objects
Symbols: 36
Spaces: 7
--------------------------------
```

Рекурсивні функції

Функція, яка викликає сама себе або декілька функцій, які викликають одна одну, називають рекурсивними.

Першу називають прямою рекурсією, а другу – непрямою.

Люба рекурсивна функція повинна мати не менше однієї нерекурсивної гілки, у противному випадку функція ніколи не вийде із рекурсії.

До недоліків рекурсивних функцій у порівнянні з ітераційними належать:

− можливість переповнення стека:

− повільніша робота, через час, який витрачається на повторні виклики.

Зауваження: будь-яку рекурсію можна замінити ітерацією(циклом).

Приклад рекурсивної функції
Рекурсивна функція факторіала.
Варіант 1 – одне значення

```
#include <iostream>
using namespace std;
long long factorial(int x)//рекурсивна функція
{
   if (x <= 1) return 1;
   return (x * factorial(x - 1));
}
int main()
{
   long long res = factorial(5);
   cout << res << endl;//120
   system("pause");
   return 0;
}
```

Результат роботи
120

Приклад рекурсивної функції
Рекурсивна функція факторіала.
Варіант 2 – значення факторіалу від 1 до 20 включно

```
#include <iostream>
using namespace std;
```

```
long long factorial(int x)//рекурсивна функція
{
   if (x <= 1) return 1;
   return (x * factorial(x - 1));
}
int main()
{
   for (int x = 1; x <= 20; ++x)
   cout << "x=" << x << "\t" << x << "! = " << factorial(x) << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
x=1     1! = 1
x=2     2! = 2
x=3     3! = 6
x=4     4! = 24
x=5     5! = 120
x=6     6! = 720
x=7     7! = 5040
x=8     8! = 40320
x=9     9! = 362880
x=10    10! = 3628800
x=11    11! = 39916800
x=12    12! = 479001600
x=13    13! = 6227020800
x=14    14! = 87178291200
x=15    15! = 1307674368000
x=16    16! = 20922789888000
x=17    17! = 355687428096000
x=18    18! = 6402373705728000
x=19    19! = 121645100408832000
x=20    20! = 2432902008176640000
```

Ітераційна функція факторіала від 1 до 10

```
#include <iostream>
using namespace std;
long long factorial(int x)//рекурсивна функція
```

```
{
   if (x <= 1) return 1;
   long long res = 1;
   for (int i = 2; i <= x; ++i) res *= i;
   return res;
}
int main()
{
   for (int x = 1; x <= 10; ++x)
   cout << "x=" << x << "\t" << x << "! = " << factorial(x) << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
x=1    1! = 1
x=2    2! = 2
x=3    3! = 6
x=4    4! = 24
x=5    5! = 120
x=6    6! = 720
x=7    7! = 5040
x=8    8! = 40320
x=9    9! = 362880
x=10   10! = 3628800
```

Приклад рекурсивної функції

Дано дійсне число $X\left(|X|<1\right)$ і ціле число $N\left(>0\right)$. Знайти значення виразу

$$X - X^2/2 + X^3/3 - \ldots + (-1)^{N-1} \cdot X^N / N.$$

Отримане число є наближеним значенням функції ln в точці $1+X$.

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
```

```
{
   double x, t;
   int n;
   long double sum;
   cout << "Input X, N: ";
   cin >> x >> n;

   if (n <= 0 || abs(x) >= 1) {
      cout << "Error." << endl;
      return 0;
   }
   {
      sum = 0;
      t = x;
      for (int i = 1; i <= n; i++)
       {
         sum += t;
         t *= (-1 * i *x) / (i+1);

       }
      cout << "sum = " << sum << endl;
}
   return 0;
}
```

Результат роботи

```
Input X, N: 0.5 6
sum = 0.404687
```

Аргументи за замовчуванням

Функція може приймати аргументи за замовчанням, тобто деякі значення, які функція використовує, якщо при виклику для параметрів явним чином не передається значення:

Приклад функції, яка повертає значення. Параметри за замовчуванням

Створити функцію, яка обчислює добуток для двох цілих чисел

```
#include <iostream>
using namespace std;
int multiply (int n , int m = 3)
```

```
{
int result = n * m;
cout << "n * m = " << result << endl;
return n*m;
  }
int main ( )
{
multiply (4, 5 ) ;
multiply (4 ) ;
return 0 ;
}
```

Результат роботи:

```
n ∗ m = 20
n ∗ m = 12
```

Для встановлення значення за замовчуванням параметру привласнюється це значення int m = 3. І, якщо для другого параметра не буде передано значення, то він використовуватиме значення за замовчуванням.

При оголошенні прототипу подібної функції він теж може містити значення за замовчуванням для параметра. І в цьому випадку ми можемо не визначати у функції значення за замовчуванням для параметра – воно братиметься з прототипу:

Приклад функції яка повертає значення. Параметри за замовчуванням

Створити функцію, яка обчислює добуток для двох цілих чисел

```
#include <iostream>
using namespace std;
int multiply (int n, int m = 3);

int main ( )
{
multiply (4, 5);
multiply (3  );
return 0 ;
}

int multiply (int n , int m)
```

```
{
int result = n * m;
cout << "n * m = " << result << endl ;
return n*m;
  }
```

Результат роботи:

n ∗ m = 20

n ∗ m = 9

Якщо при виклику функції передаються усі параметри, значення за замовчуванням не застосовуються.

Приклад функції, яка повертає значення. Параметри за замовчуванням

Створити функцію, яка обчислює суму для трьох цілих чисел

```
#include <iostream>
using namespace std;
int sum(int a, int b = 6, int c = -1)
{ return a + b+c; }
int main()
{
      cout << sum(1, 2, 3) << endl;
      cout << sum(1, 2) << endl;
      cout << sum(1) << endl;
      system("pause");
      return 0;
}
```

Результати роботи:

6

2

6

Перевантаження функцій

У мові С++ допускається перевантаження функцій, це коли функції мають однакове ім’я проте мають різні параметри за кількістю або за типом.

Приклад перевантаження функції, яка повертає значення типу double.

Створити функцію, яка обчислює відсоток за вкладами (за депозитом) залежно від кількості років

```
#include <iostream>
#include <cstdlib>
using namespace std;
double getmoney(double m, double r)
{ return m * (1 + (r/100)); }
double getmoney(double m, double r, int y) {
double s = m;
   for (int k = 0; k <= y; k++)
      s *= (1 + r/100);
   return s;}
double getmoney(double m, double r, int y, int n) {
   return getmoney(m, r/n, y * n) ;
}
int main() {
   double money = 1000;
   double rate = 5;

   cout << "initial money " << money << endl;
   cout << "interest rate per year " << rate <<  "%/" << endl;
   cout << "deposit for 1 year " << getmoney(money, rate) << endl;
   cout << "deposit for 7 year (3 times per year) " <<
getmoney(money, rate, 7, 3) << endl;

   return 0;
}
```

Результат роботи

```
initial money 1000
interest rate per year 5%/
deposit for 1 year 1050
deposit for 7 year (3 times per year) 1438.56
```

Приклад перевантаження функції

```
#include <iostream>
using namespace std;
```

```
int sum(int a, int b) { cout << "Called sum(int, int)\n"; return a + b; }
int sum(int a, int b, int c)
{ cout << "Called sum(int, int, int)\n"; return a + b + c; }
double sum(double a, double b) { cout << "Called sum(double, double)\n"; return a + b; }
int sum(int* mas, int n)
{
    cout << "Called sum(int*, int)\n";
    int res = 0;
    for (int i = 0; i < n; ++i) res += mas[i];
    return res;
}

int main()
{
    cout << sum(3, 4) << endl;
    cout << sum(1, 2, 3) << endl;
    cout << sum(2.5, 2.6) << endl;
    int mas[4] = { 1,2,5,8 };
    cout << sum(mas, 4) << endl;
    system("pause");
    return 0;
}
```

Результат роботи

```
7
Called sum(int,int,int)
6
Called sum(double, double)
5.1
Called sum(int*,int)
16
```

Шаблонні функції

Шаблонні функції – це функції, які не прив’язані до типу.

Спрощений синтаксис шаблонної функції такий:

template <typename/class T1, typename/class T2,... typename/class TN >
тип_результату ім’я_функції (список формальних аргументів)
{
// тіло функції
};

Де застосовується ключове слово typename або class для створення власних формальних типів T1, T2, …. TN.

Шаблоні функції викликаються як звичайні, якщо по параметрам функції однозначно можна встановити тип формального параметра, або викликаються за такою синтаксичною конструкцією:

ім'я_функції<фактичний_тип> (список фактичних аргументів);

Якщо для деякого типу існує більш оптимальний алгоритм, то шаблонну фукнцію можна спеціалізувати за таким синтаксисом:

template <>
тип_результату ім'я_функції (список формальних аргументів)
{
// тіло спеціалізованої функції
};

Приклад шаблонної функції для обміну двох елементів місцями. Спеціалізувати функцію для типу int.

```
#include <iostream>
using namespace std;
template <typename myT>
void mySwap(myT& a, myT& b)
{
   cout << "Called mySwap(myT&a, myT&b)\n";
   myT c = a; a = b; b = c;
}
template <>
void mySwap(int& a, int& b)
{//спеціалізація
   cout << "Called mySwap(int&a, int&b)\n";
   a = a + b; b = a - b; a = a - b;
}
int main()
{
   double x = 2.5, y = 3.7;
   mySwap(x, y);
   cout << "x=" << x << " y=" << y << endl;
```

```
    char a = 'A', b = 'B';
    mySwap<char>(a, b);
    cout << "a=" << a << " b=" << b << endl;
    int n = 5, m = 7;
    mySwap(n, m);
    cout << "n=" << n << " m=" << m << endl;
    system("pause");
    return 0;
}
```

Результат роботи

```
Called mySwap(myT&a, myT&b)
x=3.7 y=2.5
Called mySwap(myT&a, myT&b)
a=B b=A
Called mySwap(int&a, int&b)
n=7 m=5
```

inline функції

inline функції – це функції, які дають рекомендацію копілятору «встроїти» їх у місце виклику, як це робиться для макросів. inline функції доречно створювати для невеличких функцій, коли час виклику у порівняні з часом роботи самої функції схожий. Для створення inline функції, необхідно вказати ключове слово inline перед заголовком функції. Наприклад:

```
inline int sum(int a, int b) { return a + b; }
```

Функції зі змінною кількістю параметрів

Існують функції зі зміною кількістю параметрів. У таких функціях після обов'язкових параметрів (а такий хоча б один має бути) вказується параметр з трьома крапками «…». Загальний синтаксис:

тип_результату ім'я_функції (список обов'язкових параметрів, …)

Для забезпечення зручного способу доступу до аргументів функції зі змінним числом параметрів існує три макроозначення (макроси) va_start, va_arg, va_end, що знаходяться в заголовочному файлі cstdarg.

Макрос va_start призначений для установки аргументу arg_ptr на початок списку необов'язкових параметрів і має вигляд функції з двома параметрами:

void va_start(arg_ptr, prav_param);

Параметр prav_param повинен бути останнім обов'язковим параметром функції, що викликається, а покажчик arg_prt повинен бути створений від типу va_list:

va_list arg_ptr;

Макрос va_start має бути застосований до макросу va_arg.

Макрокоманда va_arg забезпечує доступ до наступного необов'язкового параметру

type_arg va_arg(arg_ptr, type);

Ця макрокоманда витягує значення типу type за адресою, заданому покажчиком arg_ptr, збільшує значення покажчика arg_ptr на розмір використаного параметра (розмір type) і таким чином параметр arg_ptr буде вказувати на наступний параметр, що викликається.

Макрокоманда va_arg використовується стільки разів, скільки необхідно для отримання усіх параметрів функції.

Макрос va_end необхідно застосувати перед виходом із функції, щоб відновити початкові стекові змінні.

Головна функція main

Головна функція main за стандартом має тільки дві форми:
без параметрів

int main() {тіло}

із двома параметрами

int main(int argc, char * argv[]) {тіло}

де argc – кількість переданих параметрів;
argv – масив покажчиків на передані аргументі.

При цьому функція має не менше одного параметра.
argv[0] – це повна назва програми, яка запускається.

Завдання для самостійної роботи

1. Написати програму, яка буде обчислювати середнє арифметичне та середнє геометричне трьох чисел, що вводяться з клавіатури.
2. Написати програму, що визначає належність числа p, яке вводиться з клавіатури, до діапазону між min та max. Значення трьох чисел вводяться користувачем з клавіатури.
3. Написати програму для обчислення min{a,b,c}.
4. Написати програму для обчислення max{a,b,c}.
5. Обчислити висоту трикутника, якщо відомі його площа та різниця між основою і висотою.
6. Дано три сторони трикутника a, b, c. Визначити його площу та перевірити, чи є він прямокутним.
7. Скласти програму, яка визначає, чи можна побудувати трикутник за заданими довжинами сторін a,b,c; якщо так, визначити, яким він є – гострокутним, прямокутним, різностороннім, рівнобедреним, рівностороннім.
8. З n чисел, що вводяться з клавіатури, подайте до друку окремо парні та непарні.
9. Написати програму, що знаходить корені звичайного квадратного рівняння за теоремою Вієта.
10. Написати програму повного дослідження сукупності коренів біквадратного рівняння. (Якщо коренів не існує, повинно бути виведене відповідне повідомлення, інакше – два або чотири корені.)
11. Знайти найближче ціле до дійсного числа, яке вводиться користувачем з клавіатури.
12. Одержати роздруківку всіх парних чисел від 1 до 1 000.
13. Одержати роздруківку всіх непарних чисел від 1 до 1 000.
14. Перевірте, чи є введене з клавіатури число простим (просте число ділиться тільки на себе і на одиницю).
15. Знайти в першій тисячі натуральних чисел тільки ті числа, що є простими. Вивести їх на екран по одному в кожному рядку.
16. Обрахуйте факторіал числа, що вводиться з клавіатури, коректно передбачивши введення від'ємних чисел.
17. Напишіть програму, що знаходить суму чисел, які передують першому від'ємному числу у введеній послідовності.
18. Користувач вводить числа, закінчуючи введення нулем. Вивести на екран найменше та найбільше число з набору.

19. Користувач вводить числа, закінчуючи введення нулем. Визначити найменше серед додатних і найбільше серед від'ємних.

20. Користувач вводить числа з клавіатури, закінчуючи введення нулем. Визначити наявність у цьому наборі від'ємних і додатних чисел. Вивести окремо кількість додатних і від'ємних чисел.

21. Користувач вводить будь-які дійсні числа з клавіатури, закінчуючи введення числом 100. Вивести на екран інформацію про підрахунок у цьому наборі як цілих чисел, так і з десятковою комою.

22. Знайти в першій тисячі натуральних чисел тільки ті числа, що без залишку діляться на число, введене користувачем з клавіатури. Вивести їх на екран по три числа в одному рядку.

23. Відшукайте мінімальне та максимальне з десяти чисел, що вводяться з клавіатури (у задачі використайте мінімальну кількість простих змінних).

24. Написати програму, що друкує у напрямку спадання усі дільники введеного числа.

25. Написати програму, що друкує у напрямку зростання усі дільники введеного числа.

26. Одержати роздруківку всіх чисел, що закінчуються на цифру 5, з проміжку від 1 до 1 000.

27. Знайти найменше спільне кратне одночасно не рівних нулю цілих чисел a та b, таких що a>b>0.

28. Знайти найбільший спільний дільник одночасно не рівних нулю цілих чисел a та b, таких що a>b>0 (використайте алгоритм Євкліда).

29. Одержати роздруківку всіх чисел, що закінчуються на цифру 2, з проміжку від 1 до N.

30. Одержати роздруківку всіх чисел, що закінчуються на цифру 3, з проміжку від 1 до N.

31. Напишіть програму, що підраховує пробіли, символи табуляції і нові рядки у вхідній послідовності символів, що вводяться з клавіатури.

32. Написати програму, що видаляє символ, який визначається користувачем, із вхідного потоку символів, що вводяться. Визначений символ для видалення вводиться з клавіатури на початку роботи програми.

33. Написати програму, що перетворює літери, які вводяться з клавіатури, із заголовних у прописні.

34. Написати програму, що підраховує кількість символів пунктуації у рядку символів, що вводиться з клавіатури.

35. Користувач вводить два числа. Визначити, чи ділиться одне число на друге без залишку. Якщо ні, запропонувати найближче ціле, що задовольняє цій умові.

36. Написати програму, що виводить на екран рядкову константу та число, що складається з цифр вашого дня народження, задане у вигляді десяткової, восьмирічної і шістнадцятирічної константи.

37. Перевірте, чи існує чотиризначне натуральне число, куб суми цифр якого дорівнює йому самому.

38. Написати програму, що проводить обмін між значеннями двох змінних, не використовуючи при цьому третьої змінної. Запропонувати декілька варіантів розв'язання такої задачі.

39. Напишіть програму, що підраховує кількість цифр у рядку символів, що вводиться з клавіатури та закінчується точкою.

МОДУЛЬ 2. МАСИВИ. ВКАЗІВНИКИ ТА РЯДКИ

Тема 3. Масиви

Масив (array) – це сукупність однотипних елементів, розташованих у пам'яті одним цілим фрагментом (шматком). Тобто кожний наступний елемент, розташований відразу після попереднього.

Масиви бувають звичайні і динамічні. Для звичайних масивів необхідно на момент створення вказати кількість елементів. Розмір динамічних масивів можна вказати в момент роботи програми.

Спрощений синтаксис створення одновимірного масиву.

тип ім'я_масиву[розмір][= {значення1, значення2,……, значення N}];

де тип – довільний тип;

розмір – ціла, додатна константа;

= {значення1, значення2,……, значення N} – ініціалізація;

[] – те що взяте в ці дужки означає, що не обов'язково вказувати розмір, компілятор може визначити кількість елементів за результатами введених даних.

Приклади створення одновимірного масиву

int arr1[5]; // масив із п’яти цілих елементів

double arr2[4]={2.5, 3.3, 5.3, 2.3};//масив із чотирьох дійсних елементів

int arr2[5]={10,2}; // масив із п’яти цілих елементів перші елементи приймуть значення відповідно 10 та 2, всі інші елементи будуть 0.

int arr3[]={10, 2, 5, 4, 3, 7}; // розмір масиву компілятор визначить як 6, тобто по списку ініціалізації.

Доступ до елемента масиву

Після створення масиву, є можливість «звернутися» до елемента масиву через індексацію (номер елемента у масиві). Індекс, це ціле число, яке визначає розташування елемента у масиві.

Індексація масиву розпочинається з нуля. Тому, якщо в масиві 5 елементів, початковий елемент з індексом нуль, останній, з індексом 4.

Синтаксис доступу до елемента масиву:

ім’я_масиву[індекс]

Доступ до елементів масиву виконується у двох напрямках:

- у режимі читання,
- у режимі запису (зміна елемента масиву).

Синтаксис «Режим читання»

змінна = ім’я_масиву[індекс];

Синтаксис «Режим запису»

ім’я_масиву[індекс] = змінна;

Приклад доступу до елементів масиву типу int

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
int arr[5];
arr[0] = 5;
arr[1] = -6;
```

```
arr[2] = 4;
arr[3] = 3;
arr[4] = 1;
cout << arr[0] << endl;
cout << arr[1] << endl;
cout << arr[2] << endl;
cout << arr[3] << endl;
cout << arr[4] << endl;
system("pause");
return 0;
}
```

Результат роботи

```
5
-6
4
3
1
```

Приклад доступу до елементів масиву типу int

```
#include<iostream>
#include<stdlib.h>
using namespace std;
int main()
{
int arr[5];
arr[0] = 5;
arr[1] = -6;
arr[2] = 4;
arr[3] = 3;
arr[4] = 1;
cout << arr[0] << endl;
cout << arr[1] << endl;
cout << arr[2] << endl;
cout << arr[3] << endl;
cout << arr[4] << endl;
system("pause");
return 0;
}
```

Результат роботи

```
5
-6
4
3
1
```

Сортування масивів і пошук у масивах. Обхід усіх елементів масиву

Зазвичай обхід усіх елементів масиву виконується у циклі. Для цього необхідно створити цикл із деяким лічильником **i**, який прийме значення усіх можливих індексів елементів, тобто, якщо в масиві 5 елементів, то **i** повине прийняти послідовно значення 0,1,2,3,4.

Якщо створити масив наступним чином, то

```
int arr[5];
```

тоді

```
int arr[i] – поточний елемент масиву
```

```
//заповнення деякого масиву розміру n у циклі
const int n = 5;
int arr[n];
for (int i = 0; i < n; ++i)
{
   arr[i] = деяке_значення;
}
```

Приклад створення масиву (введення користувачем) і друк через цикл

```
#include <iostream>
using namespace std;
int main() {
   int a[3];
   for (int i = 0; i < 3; i++)
      cin >> a[i];
   for (int i = 0; i < 3; i++)
      cout << a[i] << " ";
   cout << endl;
   return 0;
}
```

Результат роботи

```
1 3 -2
1 3 -2
```

Приклад створення масиву цілих чисел і заповнення його псевдорандомними числами.

Отриманий масив вивести на екран у рядок через пробіл

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
   const int n = 5;
   int arr[n];
   //заповнення масиву випадковими числами від -10 до +10
   for (int i = 0; i < n; ++i)
   {
      arr[i] = rand() % 21 - 10;
   }
   cout << endl;
   //виведення масиву на екран
   for (int i = 0; i < n; ++i)
   {
     cout << arr[i] << " ";
   }
   cout << endl << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
-9 -6 -1 9 -2
```

Приклад знаходження суми елементів одновимірного масиву

```
#include<iostream>
#include<cstdlib>
```

```
using namespace std;
int main()
{
   const int n = 5;
   int arr[n] = {5, 3, 7, 2, 4};
   int s = 0;
   for (int i = 0; i < n; ++i)
   {
     s+=arr[i];
   }
   cout <<"\nSum = "<<s<< endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
Sum = 21
```

Знаходження екстремумів у масиві (min/max)

Головна ідея знаходження мінімального/максимального елемента масиву:

На першому кроці вважати, що початковий елемент мінімальний/максимальний.

Порівняти цей елемент з усіма іншими елементами масиву, якщо зустрінеться елемент менший/більший, чим зберігається, тоді перевизначити значення поточного мінімального/максимального.

Алгоритм знаходження значення мінімального (максимального) елемента масиву.

1. Створити змінну min/max, в яку записати значення початкового елементу масиву min = arr[0];/max = arr[0];

2. У циклі порівняти поточний елемент масиву arr[i] з min/max.

Якщо зустрівся елемент меньший/більший, тоді перевизначити min/max: min = arr[i]; /max = arr[i].

Приклад знаходження максимального і мінімального елементів одновимірного масиву

```
#include<iostream>
#include<cstdlib>
```

```
using namespace std;
int main()
{
   const int n = 5;
   int arr[n] = {5, 3, 7, 2, 4};
   int min = arr[0];
   int max = arr[0];
   for (int i = 1; i < n; ++i)
   {
     if (arr[i] < min) min = arr[i];
     if (arr[i] > max) max = arr[i];
   }
   cout << "\nmin = " << min << endl;
   cout << "\nmax = " << max << endl << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
min = 2

max = 7
```

Знаходження індексу мінімального/максимального елемента

У попередньому прикладі було знайдено значення мінімального та максимального елементів, проте попередній алгоритм не надає інформації про те, де вони розташовані. Наприклад, якщо поставити задачу поміняти місцями екстремуми, нам необхідно місце розташування (індекси) цих елементів.

Модифікуємо попередній алгоритм: будемо зберігати не значення мінімального/максимального, а їх індекси.

Якщо створити змінні imin/imax для зберігання індексів мінімального/максимального елемента, тоді

arr[imin] – поточний мінімальний

arr[imax] – поточний максимальний

Алгоритм:

- Створити змінну imin/imax, у яку записати значення початкового індексу 0 (або неіснуюче значення, тобто –1, наприклад): imin = 0 /imax = 0.

- У циклі порівняти поточний елемент масиву arr[i] з arr[imin]/ arr[imax].
- Якщо зустрівся елемент менший/більший, тоді перевизначити збережений індекс imin/imax: imin = i / imax = i.

Приклад знаходження індексу максимального і мінімального елементів одновимірного масиву

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
const int n = 5;
   int arr[n] = {5, 3, 7, 2, 4};
   int imin = 0;
   int imax = 0;
   for (int i = 0; i < n; ++i)
   {
     if (arr[i] < arr[imin]) imin = i;
     if (arr[i] > arr[imax]) imax = i;
   }
    cout << "\nmin = " << arr[imin]
         << " index min = " << imin<<endl;
    cout << "\nmax = " << arr[imax]
         << " index max = " << imax << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
min = 2 index min = 3

max = 7 index max = 2
```

Знаходження адреси мінімального/максимального елемента

Розглянемо третій варіант знаходження мінімального/максимального елемента. На відміну від попереднього, будемо зберігати не індекс, а адресу поточного мінімального/максимального.

Модифікуємо попередній алгоритм: будемо зберігати не значення мінімального/максимального, а їх адреси.

Якщо створити покажчики ptrmin/ptrmax для зберігання адреси мінімального/максимального елемента, тоді

*ptrarr – поточний мінімальний

*ptrarr – поточний максимальний

Алгоритм:

1. Створити покажчики ptrmin/ptrmax, у які записати значення початкового індексу 0: ptrmin = &arr[0] / ptrmax = &arr[0]

2. У циклі порівняти поточний елемент масиву arr[i] з *ptrmin/ *ptrarr.

Якщо зустрівся елемент менший/більший, тоді необхідно зберегти нові адреси ptrmin/ptrmax: ptrmin = &arr[i] / ptrmax = &arr[i].

Приклад знаходження індексу та адреси максимального і мінімального елементів одновимірного масиву

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
   const int n = 5;
   int arr[n] = { 5, 3, 7, 2, 4 };
   int* ptrmin = &arr[0]; //int* pmin = mas;
   int* ptrmax = &arr[0];
   for (int i = 0; i < n; ++i)
   {
     if (arr[i] < *ptrmin) ptrmin = &arr[i];// = mas+i;
     if (arr[i] > *ptrmax) ptrmax = &arr[i];// = mas+i;
   }
   cout << "\nmin = " << ptrmin
     << " index min = " << ptrmin-arr << endl;
   cout << "\nmax = " << ptrmax
     << " index max = " << ptrmax-arr << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
min = 0x7ffc267abffc index min = 3

max = 0x7ffc267abff8 index max = 2
```

Сортування одновимірного масиву

Критерій відсортованості масиву: масив уважається відсортованим, якщо для будь-яких його сусідів виконується вибраний критерій.

Наприклад, якщо кожний наступний елемент буде більший за попередній, то масив буде відсортований за зростання, наприклад, –9 –5 3 5 7 8 9.

Усі алгоритми сортування включають в себе дві загальні процедури:

1. Порівняння деяких двох елементів із масиву.
2. Якщо для вибраних елементів не виконується критерій сортування, необхідно їх переставити місцями.

Порівняння і перестановка елементів виконуються до тих пір, поки для усіх елементів масиву не виконається «Критерій відсортованості масиву».

Кількість порівнянь, та які елементі необхідно порівнювати залежить від вибраного алгоритму сортування.

Розглянемо найпростіший алгоритм: метод бульбашки.

Головна ідея алгоритму сортування методом бульбашки містить у собі дві процедури:

1. «Порівняння усіх сусідів».
2. Повторення процедури «Порівняння усіх сусідів» n-1 раз, де n – кількість елементів у масиві.

Тепер розглянемо з чого складається «Порівняння усіх сусідів».

Алгоритм «Порівняння усіх сусідів», на прикладі сортування за зростанням (Критерій відсортованості: Наступний елемент повинен бути менший від поточного).

Попарно порівняємо всі елементи у масиві, якщо поточний елемент arr[i], тоді наступний елемент arr[i+1].

1. Порівнюємо arr[i+1] <arr[i].
2. Якщо нерівність 1) не виконується міняємо ці елементи місцями.

Приклад сортування одновимірного масиву методом бульбашки

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
```

```
{
const int n = 5;
int arr[n] = { 5, 3, 7, 2, 4 };
for (int k = 0; k < n-1; ++k) //повторення порівняння усіх сусідів
for (int i = 0; i < n-1-k; ++i)//процедура порівняння усіх сусідів
{
if (arr[i + 1] < arr[i])
//перестановка елементів
{
int c = arr[i + 1];
arr[i + 1] = arr[i];
arr[i] = c;}
}
//виведення результату на екран
for (int i = 0; i < n; ++i)
{
cout << arr[i] << " ";
}
cout << endl;
system("pause");
return 0;
}
```

Результат роботи

2 3 4 5 7

Одновимірний масив покажчиків тип *arr[n]; Визначення масиву покажчиків

Масив, який зберігає не самі об'єкти а їх адреси

тип* ім'я_масиву[розмір][= {значення1, значення2,……, значення N}];

де тип – довільний тип;

розмір – ціла, додатня константа;

= {значення1, значення2,……, значення N} – ініціалізація;

Приклади застосування одновимірного масиву покажчиків (вказівників)

Створити масив покажчиків, який зберігає адреси елементів масиву дійсних чисел у впорядкованому порядку, наприклад:

00D3FD98	00D3FD94	00D3FD9C	00D3FDA0	00D3FDA4	00D3FDA8

тип* ptrarr

3	2	5	10	4	7

тип ptrarr

Тобто, елементи в масиві об’єктів ми не чіпаємо. Масив може бути, наприклад константним.

Алгоритм

1. Створити масив покажчиків **ptrarr** і зберегти в ньому адреси послідовно

ptrarr[i] = &arr[i] або ptrarr[i] = arr+i

2. Застосувати алгоритм сортування методом бульбашки для масиву покажчиків, де необхідно порівнювати елементи на які вказують сусідні покажчики, но переставляти необхідно не об’єкти, а самі покажчики в масиві покажчиків:

```
if (*ptrarr[i + 1] < *ptrarr[i])
{
   ТИП* c = ptrarr[i + 1];
   ptrarr[i + 1] = ptrarr[i];
   ptrarr[i] = c;
}
```

Приклад сортування одновимірного масиву методом бульбашки із використанням вказівників

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
   const int n = 6;
   int arr[n] = { 3,2,5,10,4,7 };
   int *ptrarr[n];
   for (int i = 0; i < n; ++i)
      ptrarr[i] = &arr[i];

   for (int k = 0; k < n-1; ++k)
      for (int i = 0; i < n - 1 - k; ++i)
```

```
            if (*ptrarr[i + 1] < *ptrarr[i])
            {
               int* c = ptrarr[i + 1];
               ptrarr[i + 1] = ptrarr[i];
               ptrarr[i] = c;
            }
      cout <<"Result:"<< endl;
      for (int i = 0; i < n; ++i)
         cout << *ptrarr[i] << " ";
      cout << endl << "Origin array: " << endl;
      for (int i = 0; i < n; ++i)
         cout << arr[i] << " ";
      cout << endl;
      system("pause");
      return 0;
   }
```

Результат роботи

```
Result:
2 3 4 5 7 10
Origin array:
3 2 5 10 4 7
```

У одновимірному масиві, що складається з цілих чисел, обчислити: 1) кількість елементів рівних 0; 2) кількість елементів масиву, розташованих після мінімального елемента.

Відсортувати елементи масиву за зменшенням модулів його елементів

```
#include<iostream>
#include<cstdlib>
#include<ctime>
using namespace std;
int main() {
   srand(time(NULL));

   int size = 10;
   int arr[size];
   for(int i = 0; i < size; i++)
   { arr[i] = rand () % 10 - 5 ;}
```

```
    for(int i = 0; i < size; i++)
    { cout << arr[i]<< "\t"; }
    int c = 0;
    for(int i = 0; i < size; i++)
       if(arr[i] == 0) ++c;
    cout << " \nZero: " << c << " \n ";
    int imin = 0;
    for (int i = 0; i < size; ++i)
    {
       if (arr[i] < arr[imin]) imin = i;
    }
    cout << "min = " << arr[imin]
        << " index min = " << imin<<endl;
    int quantity;
    quantity = size - 1 -imin;
    cout << " quantity after min " << quantity << endl;


    for (int i = 0; i < size - 1; i++)
       for (int j = 0; j < size - 1 - i; j++)
          if (abs(arr[j]) < abs(arr[j + 1]))
          {
                swap(arr[j], arr[j +1]);
          }
    for (int i = 0; i < size; i++){
    cout << arr[i] << "\t";}
    return 0;
 }
```

Результат роботи

```
-1      3      -4      4      -4      1      -1      3      -5      -3
 Zero: 0
  min = -5 index min = 8
  quantity after min 1
-5      -4      4      -4      3      3      -3      -1      1      -1
```

Багатовимірний масив (матриця) matr[n][m];
Визначення масиву покажчиків

Масив, який зберігає не самі об’єкти а їх адреси

тип ім’я_масиву[розмір1] [розмір2]……[розмірN]
[= {значення1, значення2,……, значення N}];

де тип – довільний тип;
розмір1 – цілі, додатні константи;
= {значення1, значення2,……, значення N} – ініціалізація.

Приклади створення двовимірного масиву

int matr[3][4]; // матриця цілих елементів розміром 3 рядки і 4 стовпчики

	0	1	2	3
0				
1				5
2				

matr[i][j] – поточний елемент

де і номер рядка, приймає значення від 0 по 2 включно;
j номер стовпчика, приймає значення від 0 по 3 включно.
matr[1][3] = 5.

Приклади створення двовимірного масиву

int matr1[3][4]; // матриця цілих елементів розміром 3 рядки і 4 стовпчики
int matr2[3][4]={4,3,6,5,3,7,8,9,3,5,45,7};
int matr3[3][4]={{4,3,6,5},{3,7,8,9},{3,5,45,7}};

	0	1	2	3
0	**4**	**3**	**6**	**5**
1	**3**	**7**	**8**	**9**
2	**3**	**5**	**45**	**7**

Приклад створення двовимірного статичного масиву, друк елементів у рядок

```
#include <iostream>
using namespace std;
int main() {
   int array [3][2] = {{2,6}, {8,0}, {5,7}};
   for ( int i = 0; i < 3; i++)
      for (int j = 0; j < 2; j++) {
         cout << "  " << array[i][j];
      }
```

```
    return 0;
}
```
Результат роботи
```
2 6 8 0 5 7
```

Приклад створення двовимірного статичного масиву, друк елементів у вигляді матриці. Варіант 1

```
#include <iostream>
using namespace std;
int main() {
    int array [3][2] = {{2,6}, {8,0}, {5,7}};
    for ( int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++)
  { cout << "  " << array[i][j]; }
  cout << endl;
    }
    return 0;
}
```
Результат роботи
```
2  6
8  0
5  7
```

Приклад створення двовимірного статичного масиву, друк елементів у вигляді матриці. Варіант 2

```
#include <iostream>
#include <iostream>
using namespace std;
int main() {
    int array [3][2] = {{3,-1}, {5,0}, {4,-2}};
    for ( int i = 0; i < 3; i++, cout << endl) {
        for (int j = 0; j < 2; j++)
  { cout << "  " << array[i][j]; }
     }
    return 0;
}
```
Результат роботи
```
3  -1
5  0
4  -2
```

Приклад створення двовимірного статичного масиву, друк елементів у вигляді матриці. Дані вводить користувач з клавіатури

```
#include <iostream>
#include<cstdlib>
using namespace std;

int main() {
   int array[3][3];
   for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++)
      {
      cin >> array[i][j];
            }
      }
      cout<< endl;
      for (int i = 0; i < 3; i++, cout << endl) {

         for (int j = 0; j < 3; j++)
         { cout << " \t " << array[i][j];  }
      }
      return 0;
}
```

Результат роботи

```
5 -7 8 0 4 12 -9 32 47

 5        -7        8
 0         4       12
-9        32       47
```

Приклад створення двовимірного масиву (псевдорандомні числа в інтервалі від –10 до 10) і підрахунок від'ємних елементів

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
```

```
    const int n = 4;
    const int m = 5;
    int matr[n][m]; // матриця цілих елементів розміром n рядки і m стовпчиків
    // заповнення двовимірного масиву випадковими числами в діапазоні від -10 до 10
    for (int i = 0; i<n;++i)
       for (int j = 0; j < m; ++j)
          matr[i][j] = rand() % 21 - 10;
    // друкування двовимірного масиву у вигляді матриці
    for (int i = 0; i < n; ++i, cout<<endl)
       for (int j = 0; j < m; ++j)
          cout<<matr[i][j] <<"\t";
    // Підрахунок кількості від'ємних елементів у матриці
    int c = 0;
    for (int i = 0; i < n; ++i)
       for (int j = 0; j < m; ++j)
          if (matr[i][j] < 0) ++c;
    cout << "\nNumber negativ: "<< c<<"\n";
    system("pause");
    return 0;
}
```

Результат роботи

```
-9    -6    -1    9    -2
0     0     -1    5    0
-8    9     10    -6   10
-3    -7    5     6    6

Number negativ: 9
```

Приклад створення двовимірного масиву (псевдо-рандомні числа в інтервалі від –10 до 10).

Знайти в кожному рядку мінімальний елемент, а серед них – максимальний

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
```

```
    {
        const int n = 4;
        const int m = 5;
        int matr[n][m]; // матриця цілих елементів розміром n рядки і m стовпчиків
        // заповнення двовимірного масиву випадковими числами в діапазоні від -10 до 10
        for (int i = 0; i<n;++i)
            for (int j = 0; j < m; ++j)
                matr[i][j] = rand() % 21 - 10;
        // друкування двовимірного масиву у вигляді матриці
        for (int i = 0; i < n; ++i, cout<<endl)
            for (int j = 0; j < m; ++j)
                cout<<matr[i][j] <<"\t";
        // Знаходження мінімальних елементів у рядках
        int mins[n] ;
        for (int i = 0; i < n; ++i)
        {
            int numberMin = matr[i][0];
            for (int j = 0; j < m; ++j)
            {
                if (matr[i][j] < numberMin) numberMin = matr[i][j];
            }
            mins[i] = numberMin;
        }
        cout << "\nMins: ";
        for (int i = 0; i < n; ++i)
            cout << mins[i] << " ";
        // Знаходження максимального елементу серед мінімальних
        int max = mins[0];
        for (int i = 0; i < n; ++i)
            if (mins[i] > max) max = mins[i];
        cout << "\nResult. max = " << max << endl;
        system("pause");
        return 0;
    }
```

Результат роботи

```
-9      -6      -1      9       -2
0       0       -1      5       0
-8      9       10      -6      10
-3      -7      5       6       6
```

```
Mins: -9 -1 -8 -7
Result. max = -1
```

Приклад створення двовимірного масиву (псевдорандомні числа в інтервалі від –10 до 10).
Знайти в кожному рядку мінімальний елемент, а серед них – максимальний

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
   const int n = 4;
   const int m = 5;
   int matr[n][m]; // матриця цілих елементів розміром n рядки і m стовпчиків
   // заповнення двовимірного масиву випадковими числами в діапазоні від -10 до 10
   for (int i = 0; i<n;++i)
      for (int j = 0; j < m; ++j)
         matr[i][j] = rand() % 21 - 10;
   // друкування двовимірного масиву у вигляді матриці
   for (int i = 0; i < n; ++i, cout<<endl)
      for (int j = 0; j < m; ++j)
         cout<<matr[i][j] <<"\t";
   // Знаходження мінімальних парних елементів у рядках
   int masEven[m] ;
   for (int j = 0; j < m; ++j)
   {
      int numberEven = 0;
      for (int i = 0; i < n; ++i)
      {
         if (matr[i][j]%2==0) ++numberEven;
      }
      masEven[j] = numberEven;
   }
   cout << "\nPar: ";
   for (int i = 0; i < m; ++i)
```

```
        cout << masEven[i] << " ";
    // Знаходження максимального елементу серед мінімальних
    int imax = 0;
    for (int i = 0; i < m; ++i)
        if (masEven[i] > masEven[imax]) imax = i;
    cout << "\nResult. index max = " << imax << endl;
    cout << "\nNumber max el = " << masEven[imax] << endl;
    system("pause");
    return 0;
}
```

Результат роботи

```
-9    -6    -1    9    -2
0     0     -1    5    0
-8    9     10    -6   10
-3    -7    5     6    6

Par: 2 2 1 2 4
Result. index max = 4

Number max el = 4
Result. max = -1
```

Приклад створення двовимірного масиву (квадратна матриця) рандомних чисел

```
#include <iostream>
#include <ctime>
using namespace std;
int main() {
    const int MAX_N = 10;
    const int MIN_VALUE = 1;
    const int MAX_VALUE = 20;
    int A[MAX_N][MAX_N];
    int n;
    cout << "n = ";
    cin >> n;
    srand(time(NULL));
    for (int i = 0; i < n; i++)
```

```
          for (int j = 0; j < n; j++)
          { A[i][j] = MIN_VALUE + rand() % (MAX_VALUE -
MIN_VALUE + 1); }
       cout << endl << "Array A:" << endl;
       for (int i = 0; i < n; i++)
       {
          for (int j = 0; j < n; j++)
             cout << A[i][j] << '\t';
          cout << endl;
       }
          return 0;
   }
```

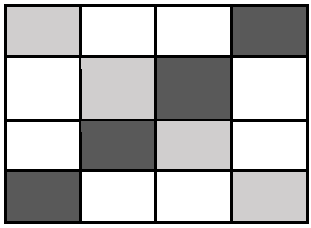
Результат роботи

```
n = 5
Array A:
 10    18    5    15    8
 17    12    7    9    19
 13    7     8    3    1
 8     18    5    2    19
 10    18    2    10   3
```

Квадратна матриця – тип matr[n][n];

Головна діагональ
matr[i][i]
Допоміжна діагональ
matr[i][n-1-i]

Елементи вище головної діагоналі
matr[i][j],
де i Є [0, n-1)
   j Є [i+1, n)
Елементи нижче головної діагоналі
matr[i][j],
де i Є [0, n)
   j Є [0, i)
Елементи вище головної і побічної діагоналей
matr[i][j],
де i Є [0, (n-1)/2)
   j Є [1+i, n-1-i)



Приклад створення двовимірного масиву (квадратна матриця) рандомних чисел

```
#include <iostream>
#include <ctime>
using namespace std;
int main() {
   const int N=10;
   int A[N][N];
   int n;
   cout << "Input number of rows and columns, <=10: "<<" n = " ;
   cin >> n;
   srand(time(NULL));
   for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
      { A[i][j] = rand() % 100; }
   cout << endl << "Array A:" << endl;
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
         cout << A[i][j] << '\t';
      cout << endl;
   }
      return 0;
}
```

Результат роботи

```
Input number of rows and columns, <=10:  n = 3

Array A:
41    71    93
86    32    47
23     2    40
```

Обміняти елементи головної і допоміжної діагоналей двовимірного масиву (квадратної матриці)

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
const int n = 5;
int matr[n][n]; // матриця цілих елементів розміром n рядки і m стовпчиків
for (int i = 0; i < n; ++i, cout << endl)
for (int j = 0; j < n; ++j)
{
matr[i][j] = rand() % 21 - 10;
cout << matr[i][j] << "\t";
}
// Обмін діагоналей місцями
for (int i = 0; i < n; ++i)
{
int c = matr[i][i];
matr[i][i] = matr[i][n - 1 - i];
matr[i][n - 1 - i]=c;
}
//Друкування результату
cout << endl;
for (int i = 0; i < n; ++i, cout << endl)
for (int j = 0; j < n; ++j)
cout << matr[i][j] << "\t";
system("pause");
return 0;
}
```

Результат роботи

```
 -9   -6   -1    9   -2
  0    0   -1    5    0
 -8    9   10   -6   10
 -3   -7    5    6    6
  7    4    2   -1   -8
```

```
 -2   -6   -1   9   -9
  0    5   -1   0    0
 -8    9   10  -6   10
 -3    6    5  -7    6
 -8    4    2  -1    7
```

У квадратній матриці замість головної діагоналі записати 0. Записати 1 вище допоміжної діагоналі. Записати 2 нижче допоміжної діагоналі

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
  const int n = 5;
  int matr[n][n]; // матриця цілих елементів
//розміром n рядки і m стовпчиків
// Заповнення діагоналі
  for (int i = 0; i < n; ++i)
     matr[i][n-1-i] = 0;
// Заповнення вище побічної діагоналі
  for (int i = 0; i < n-1; ++i)
    for (int j = 0; j < n-1-i; ++j)
      matr[i][j] = 1;
// Заповнення нижче побічної діагоналі
  for (int i = 1; i < n ; ++i)
    for (int j = n-i; j < n; ++j)
      matr[i][j] = 2;
//Друкування результату
  for (int i = 0; i < n; ++i, cout << endl)
    for (int j = 0; j < n; ++j)
      cout << matr[i][j] << "\t";
  cout << endl;
  system("pause");
  return 0;
}
```

Результат роботи

```
1   1   1   1   0
1   1   1   0   2
1   1   0   2   2
1   0   2   2   2
0   2   2   2   2
```

Тема 4. Вказівники та рядки

У Visual C++ (CLR – Common Language Runtime) підтримуються три типи вказівників:

- керовані покажчики (managed pointers);
- некеровані покажчики (unmanaged pointers);
- некеровані вказівники на функції (unmanaged function pointers).

Відмінність між керованими та некерованими вказівниками така: на відміну від керованого покажчика некерованому покажчику можна надати будь-яку адресу пам'яті, навіть ту, яка знаходиться за межами виконавчої середовища.

У цьому випадку пам'ять є нерегульованою.

У разі регульованої пам'яті керованому покажчику присвоїти адресу за межами виконавчої середовища не вдасться.

Керовані показники – це показники типу посилання. Ці вказівники передаються для аргументів, методів, які передаються за посиланням. Ці показники є сумісні зі специфікацією Common Language Runtime (CLR). Керовані вказівники є посиланнями на об'єкти. Ці об'єкти розміщуються у загальній керованій пам'яті, яка виділяється для програми у момент її виконання.

У таких вказівниках замість символу * застосовується символ ^. Для виділення пам'яті під керований покажчик використовується утиліта gcnew.

Некеровані вказівники – це традиційні вказівники C/C++. Вони є покажчиками на об'єкти в нерегульованому об'ємі пам'яті, що виділяється для виконання програми. Некеровані покажчики не є сумісні зі специфікацією CLR.

Некеровані вказівники на функції – це вказівники на функції, які можна обробляти так само як і некеровані покажчики на об'єкти (дані).

Загальна форма оголошення некерованого вказівника (unmanaged pointer):

тип_змінної * імя_вказівника;

де

тип_змінної – це тип тієї змінної, на яку вказує вказівник. Його ще називають базовий тип покажчика;

імя_вказівника – імя змінної-покажчика.

Опис вказівників, які вказують на різні типи змінних.

```
int * pi; // вказівник з іменем pi на тип int
float * pf; // вказівник з іменем pf на тип float
```

Для отримання адреси змінної у пам'яті використовується операція взяття адреси &. Ця операція є унарною. Вона повертає адресу свого операнда в пам'яті.

Приклад операції взяття адреси та використання вказівника для доступу до змінної.

```
int * pi; // вказівник з іменем pi на тип int
float * pf; // вказівник з іменем pf на тип float
float f;
int i;

i = 8;
pi = &i; // pi вказує на i
*pi = 10; // i = 10

f = 2.93f;
pf = &f; // pf вказує на f
*pf = 8.85f; // f = 8.85
```

Приклади роботи із вказівниками

```
// Вказівник

#include <iostream>
using namespace std;
int main() {
   int value = 18;
         int *ptrvalue = &value;
   cout << *ptrvalue << endl;
   return 0;
}
```

Результат роботи:

```
18
```

```
// Зміна значення вказівника

#include <iostream>
using namespace std;
int main() {
   int value = 18;
         int *ptrvalue = &value;
         cin >> *ptrvalue;
   cout << value << endl;
   return 0;
}
```

Результат роботи:

```
1
1
```

```
// Вказівник на вказівник

#include <iostream>
using namespace std;
int main() {
   int value = 18;
         int *ptrvalue = &value;
         int **ptr_ptrvalue = &ptrvalue ;
         cin >> **ptr_ptrvalue;
   cout << value << endl;
   return 0;
}
```

Результат роботи:

```
5
5
```

```
// Посилання (адреса) на комірку

#include <iostream>
using namespace std;
int main() {
   int value = 18;
   int &ref = value;
   cin >> ref;
   cout << value << " " << &ref << endl;
   return 0;
}
```

Результат роботи:
6
6 0x7fffa237fcec

Застосування вказівника (покажчика) на функцію

Як і будь-який об'єкт, функція розташована у пам'яті і має адресу. Для зберігання цією адреси необхідно застосовувати вказівник на функцію.

Синтаксис створення вказівника на функцію.

тип_результату (*вказівник) (список формальних аргументів);

При виклику функції через вказівник допускається її виклик як через розіменування так і без.

Приклад виклику функції

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
   double(*pf)(double);
   pf = &sin;
   cout << "sin(0.5) = "<<pf(0.5) << endl;
   cout << "sin(0.5) = " << (*pf)(0.5) << endl;
   pf = &cos;
   cout << "cos(0.5) = " << pf(0.5) << endl;
   system("pause");
   return 0;
}
```

Результат роботи

```
sin(0.5) = 0.479426
sin(0.5) = 0.479426
cos(0.5) = 0.877583
```

Вирази й арифметичні операції із вказівниками (покажчиками). Зміна фізичної адреси покажчика

Над покажчиками можна виконувати такі арифметичні операції:

- операції інкременту ++ та декременту —;
- операції додавання + та віднімання –.

Покажчики можуть брати участь у виразах.

При зміні значення покажчика за допомогою арифметичних операцій фізична адреса покажчика змінюється за формулою:

N * sizeof(тип)

де

− N – константа, що вказує зміну (збільшення/зменшення) значення покажчика;

− тип – тип даних, на який вказує покажчик (базовий тип покажчика);

− sizeof(тип) – операція, що визначає розмір типу даних.

Наприклад: Зміна значення покажчика.

```
// збільшення/зменшення значення покажчика
int a = 5;     // звичайна змінна
int * pi1 = &a; // ініціалізація покажчика адресою змінної a
int * pi2;     // покажчик на int
pi2 = pi1;     // вказують на одну адресу пам'яті
pi2++;         // фізична адреса pi2 на 4 байти більша ніж адреса pi1
```

У вищенаведеному прикладі фізична адреса покажчика pi2 більша на 4 байти від фізичної адреси покажчика pi1. Це пов'язано з тим, що тип int (базовий тип покажчиків pi1 та pi2) має розмір 4 байти (середовище Win32).

Наприклад: Зміна значення покажчика на тип double.

```
// збільшення/зменшення значення покажчика
double x = 3.55;   // звичайна змінна типу double
double * p1 = &x; // ініціалізація покажчика адресою змінної a
double * p2;     // покажчик на double

p2 = p1 - 4; // фізична адреса p2 на 32 байти менша ніж адреса p1
p2 = p1 + 2; // фізична адреса p2 на 16 байт більша ніж адреса p1
```

Покажчики можна порівнювати. Для порівняння покажчиків використовуються операції ==, >, <.

При порівнянні покажчиків важливо, щоб ці покажчики мали деякий зв'язок між собою.

Наприклад, якщо покажчики вказують на різні нічим не зв'язані змінні, то операція порівняння не має сенсу.

У випадку, коли покажчики вказують на змінні, між якими є деякий зв'язок, то результат порівняння має зміст. Це може бути випадок, коли покажчики вказують на елементи одного й того ж масиву.

На операцію взяття адреси & накладаються такі обмеження:

1. Неможливо визначити адресу константи. Наприклад вираз

pi = &0xffe800;

призведе до помилки.

2. Неможливо визначити адресу значення, яке отримується при обчисленні виразу.

Наприклад фрагмент коду призведе до помилки:

```
int *pi;
int d;
d = 8;
pi = &(d + 5); // помилка, неможна взяти адресу виразу
```

Динамічна пам'ять (heap)

Стекова пам'ять
Статична/глобальна пам'ять

```
{
  int   z=9;
}
int a=2;
{
  static int   z=9;
}
```

Динамічна пам'ять
C

```
int* p1 = (int*)malloc(sizeof(int));
int* p2 = (int*)calloc(3, sizeof(int));
int* p3 = (int*)realloc(p2, sizeof(int));
```

C++

```
int* p4 = new int(5);
int* p5 = new int[5];
```

Фукції С виділення динамічної пам'яті

void* malloc(size_t size); – виділення динамічної пам'яті
де size_t це unsigned int
size – кількість пам'яті в байтах

void* calloc(size_t num, size_t size); – виділення динамічної пам'яті

num – кільість блоків
size – розмір блоку в байтах

void* realloc(void* ptr, size_t size); – перевиділення динамічної пам’яті
ptr – покажчик на попередню виділену пам’ять
size – розмір блоку в байтах

Функції вертають адресу розташування виділеної пам’яті.
Якщо пам’ять виділити не вдалося, функції вертають NULL

Фукція С звільнення динамічної пам’яті

void free(void* ptr);
ptr покажчик, який зберігає адресу виділеної динамічної пам’яті

Оператор С++ виділення динамічної пам’яті

ТИП* prt1 = new ТИП;
ТИП* prt2 = new ТИП(значення);
ТИП* mas = new ТИП[кількість елементів];

Якщо оператор new не зміг виділити пам’ять, генерується exception bad_alloc

Оператор С++ звільнення динамічної пам’яті

delete ptr1;
delete []mas;

Головна різниця функцій виділення пам’яті мови С і операторів С++

Функції сімейства malloc шукають вільну пам’ять, помічають її занятою і вертають її адресу
Функція сімейства free помічають пам’ять за вказаної адресою як вільну.
Оператор new не тільки виділяє динамічну пам’ять, а ще й викликає конструктор.
Оператор delete спочатку викликає деструктор, а потім звільняє виділену память.

Створення динамічної змінної

C ТИП* ІМ'Я = (ТИП*)malloc(sizeof(ТИП)); free(ІМ'Я);	C++ ТИП* ІМ'Я = new ТИП; ТИП* ІМ'Я = new ТИП(значення); delete ІМ'Я;

#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
double* a =
(double*)malloc(sizeof(double)
);
*a = 2.5;
cout << *a << endl;
free(a);
return 0;
}
	Варіант 1
	#include <iostream>
	#include <cstdlib>
	using namespace std;
	int main()
	{
	double* a = new double;
	*a = 2.5;
	cout << *a << endl;
	delete a;
	return 0;
	}
	
	Варіант 2
	#include <iostream>
	#include <cstdlib>
	using namespace std;
	int main()
	{
	double* a = new double(2.5);
	cout << *a << endl;
	delete a;
	return 0;
	}

Створення динамічного масиву розміром n

C ТИП* ІМ'Я = (ТИП*)malloc(sizeof(ТИП)*n); free(ІМ'Я);	C++ ТИП* ІМ'Я = new ТИП[n]; delete [] ІМ'Я;
#include <iostream> #include <stdlib.h>	#include <iostream> #include <stdlib.h>

```
using namespace std;
int main()
{
   int n = 5;
   int* mas =
(int*)malloc(sizeof(int)*n);
   for (int i = 0; i < n; ++i)
   {
      mas[i] = rand() % 100;
      cout << mas[i] << " ";
   }
   free(mas);
   return 0;
}
```

```
using namespace std;
int main()
{
   int n = 5;
   int* mas = new int [n];
   for (int i = 0; i < n; ++i)
   {
      mas[i] = rand() % 100;
      cout << mas[i] << " ";
   }
   delete[] mas;
   return 0;
}
```

Створення двовимірного динамічного масиву

Баготовимірні масиви – це комбінація одновимірних.

Алгоритм створення масиву n x m по 1-й схемі

1. Створення динамічного масиву покажчиків довжиною n:
   ТИП** matr = new ТИП* [n];
2. У циклі n раз створюємо динамічній масив довжиною m для даних, адресу яких записуємо в масив покажчиків, тобто в matr[i]:

```
for (int i = 0; i < n; ++i)
   {
      matr[i] = new ТИП[m];
   }
matr[i][j]
```

Алгоритм звільнення масиву створеного по 1-й схемі

Звільнення виконується в зворотному порядку.

1) В циклі n раз звільняються динамічні масиви, створені на 2-му кроці:

```
for (int i = 0; i < n; ++i)
   delete [] matr[i];
```

2) Звільнити масив створений на першому кроці:

```
delete [] matr;
```

1 крок
2 крок

Створення двовимірного динамічного масиву по 1 схемі

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int n = 5;
  int m = 6;
//створення масиву
  int** matr = new int* [n];
  for (int i = 0; i < n; ++i)
    matr[i] = new int[m];
//заповнення масиву
  for (int i = 0; i < n; ++i, cout<<endl)
    for (int j = 0; j < m; ++j)
    {
      matr[i][j] = rand() % 100;
      cout << matr[i][j] << "\t";
    }
//звільнення масиву
  for (int i = 0; i < n; ++i)
    delete[] matr[i];
  delete[] matr;
  system("pause");
  return 0;
}
```

Результат роботи

```
 83     86     77     15     93     35
 86     92     49     21     62     27
 90     59     63     26     40     26
 72     36     11     68     67     29
 82     30     62     23     67     35
```

У одновимірному масиві (динамічному), що складається з цілих рандомних чисел (від –5 до 5), обчислити: 1) кількість елементів рівних 0; 2) кількість елементів масиву, розташованих після мінімального елемента.

Відсортувати елементи масиву за зменшенням модулів його елементів

```
#include<iostream>
#include<cstdlib>
```

```
#include<ctime>
using namespace std;
int main() {
   srand(time(NULL));
   int n = 10;
   int* arr = new int[n];
   for(int i = 0; i < n; i++)
   {
      arr[i] = rand () % 11 - 5 ;}
   for(int i = 0; i < n; i++)
   { cout << arr[i] << "  \t  ";}
   int c = 0;
   for(int i = 0; i < n; i++)
      if(arr[i] == 0) ++c;
   cout << " \nZero: " << c << " \n ";
   int imin = 0;
   for (int i = 0; i < n; ++i)
   {
      if (arr[i] < arr[imin]) imin = i;
   }
   cout << "min = " << arr[imin]
        << " index min = " << imin<<endl;
   int quantity;
   quantity = n - 1 -imin;
   cout << " quantity after min " << quantity << endl;
   for (int i = 0; i < n - 1; i++)
      for (int j = 0; j < n - 1 - i; j++)
         if (abs(arr[j]) < abs(arr[j + 1]))
         {
               swap(arr[j], arr[j +1]);
         }
   cout << "Sorted array: " << endl;
   for (int i = 0; i < n; i++){
      cout << arr[i]<< "  \t  ";}
   delete[]arr;
   return 0;
}
```

Результат роботи

-2 5 -5 -5 -1 4 1 -1 5 -5

```
Zero: 0
 min = -5 index min = 2
 quantity after min 7
Sorted array:
  5     -5     -5     5     -5     4     -2     -1     1     -1
```

Алгоритм створення масиву n x m по 2-й схемі

1. Створення динамічного масиву покажчиків довжиною n:
   ТИП** matr = new ТИП* [n].

2. Створення динамічного масиву для даних довжиною n*m, адресу якого записуємо в елемент matr[0]
   matr[0] = new ТИП [n*m].

3. У циклі n-1 раз обчислюємо адресу початку кожного рядка, розпочинаючи з індексом 1 і записуємо їх у масив покажчиків matr[i]

```
for (int i = 1; i < n; ++i)
    matr[i] = matr[0]+i*m;
```

Алгоритм звільнення масиву створеного по 2-й схемі

Звільнення виконується в зворотному порядку.

1. Звільнити масив створений на другому кроці:
   delete [] matr[0].

2. Звільнити масив створений на першому кроці:
   delete [] matr.

Створення двовимірного динамічного масиву по 2-й схемі

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
  int n = 5;
  int m = 6;
//створення масиву
  int** matr = new int* [n];//1 крок
  matr[0] = new int[n*m]; //2 крок
```

```
  for (int i = 1; i < n; ++i) //3 крок
    matr[i] = matr[0]+i*m;
//заповнення масиву
  for (int i = 0; i < n; ++i, cout<<endl)
    for (int j = 0; j < m; ++j)
    {
      matr[i][j] = rand() % 100;
      cout << matr[i][j] << "\t";
    }
//звільнення масиву
  delete[] matr[0];
  delete[] matr;
  system("pause");
  return 0;
}
```

Результат роботи

```
 83    86    77    15    93    35
 86    92    49    21    62    27
 90    59    63    26    40    26
 72    36    11    68    67    29
 82    30    62    23    67    35
```

Переставити рядки матриці відповідно до зростання елементів другого стовпця. Варіант 1

```
#include <iostream>
#include<cstdlib>
using namespace std;

int main()
{
   int m, n, i, j;
   int* c;
   int** arr;
   cout << "Input numbers of rows (m) and columns (n)" << endl;
   cin >> m >> n;
   // створюємо динамічний масив
   arr = new int*[m]; // створюємо масив покажчиків
   for (i = 0; i < m; i++) {
      arr[i] = new int[n]; // створюємо динамічні рядки
   }
```

```
        cout << "Input " << m << "x" << n << " elements of array"
<< endl;

        // заповнюємо масив з клавіатури
        for (i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                cin >> arr[i][j];
            }
        }

        // Виведення на экран вихідного масиву
        for (i = 0; i < m; i++, cout << endl) {
            for (int j = 0; j < n; j++) {
                cout << arr[i][j] << "\t";
            }
        }

        // порівняння
        for (i = 1; i < m; i++) {
            if (arr[i][1] < arr[i-1][1]) {

                c = arr[i-1];
                arr[i-1] = arr[i];
                arr[i] = c;
            }
        }

        // Виведення на екран відсортованого масиву
        cout << "Array after sorting:\n";
        for (i = 0; i < m; i++, cout << endl) {
            for (int j = 0; j < n; j++) {
                cout << arr[i][j] << "\t";
            }
        }

        for (int i = 0; i < m; ++i)
            delete [] arr[i];
        delete [] arr;

        cin.get(); // затримка екрану
        return 0;
    }
```

Результат роботи

```
Input numbers of rows (m) and columns (n)
2 3
Input 2x3 elements of array
  1       2       8
  5      -1      -8
  1       2       8
  5      -1      -8
Array after sorting:
  5      -1      -8
  1       2       8
```

Переставити рядки матриці відповідно до зростання елементів другого стовпця. Варіант 2

```
#include <iostream>
#include <iomanip>
using namespace std;

int main () {
   int n;
   int m;
   int i;
   cin >> n;
   cin >> m;
   int **matr = new int *[n];
   for (int i = 0; i < n; ++i)
      matr[i] = new int[m];

   for (int i = 0; i < n; ++i)
      for (int j = 0; j < m; ++j)
         matr[i][j] = (rand() % 21) + 2 - 20;

   for (int i = 0; i < n; ++i, cout << endl)
      for (int j = 0; j < m; ++j)
         cout << setw(4) << matr[i][j];
   int numStob = 2;
   for (int k = 0; k < n - 1; ++k)
      for (int i = 0; i < n - 1 - k; ++i) {
```

```
            if (matr[i + 1][numStob] < matr[i][numStob])
                for (int d = 0; d < m; ++d) {

                    int c = matr[i + 1][d];
                    matr[i + 1][d] = matr[i][d];
                    matr[i][d] = c;
                }
        }

 cout << " Result " << endl;
for (int i = 0; i < n; ++i, cout << endl)
for (int j = 0; j < m; ++j)
   cout << setw(4) << matr[i][j];
   delete []matr[i];
delete []matr;
return 0;
}
```

Результат роботи

```
3
4
-17   -14    -9     1
-10    -8    -8    -9
 -3    -8   -16     1
Result
 -3    -8   -16     1
-17   -14    -9     1
-10    -8    -8    -9
```

У одновимірному масиві (динамічному), що складається з цілих псевдорандомних чисел (від –5 до 5), знайти: 1) номер максимального по модулю елемента масиву; 2) добуток елементів масиву, розташованих між першим і останнім нульовим елементом; 3) переставити максимальний по модулю елемент з першим.

```
#include <iostream>
#include <ctime>
using namespace std;
int main() {
```

```
        srand(time(NULL));
        int n = 10, prod = 1;
        int i;
        int* arr = new int[n];
        for (int i = 0; i < n; i++)
    arr[i] = rand() % 10 -5;
        cout << "Unsorted array:" << endl;
        for (int i = 0; i < n; i++)
            cout << arr[i] << "\t";
        cout << endl;
        int imax = 0;
        for (int i = 0; i < n; ++i)
        {
            if (abs(arr[i]) > (abs(arr[imax]))) imax = i;
        }
        cout << "max = " << abs(arr[imax])
             << " index max = " << imax << endl;
        int zero_1 = 0;
        for (int i = 0; i < n; i++)
            if (arr[i] == 0)
            {
                zero_1 = i;
                break;}
        int zero_2 = n - 1;
        for (int i = n-1; i > -1; i--)
            if (arr[i] == 0)
            {
                zero_2 = i;
                break;
            }
        if (zero_1 == zero_2);
        else
            for (int i = zero_1 + 1; i < zero_2; i++){
                prod *= arr[i];}
    cout << "Product of elements between first zero and last zero: "
<< prod << endl << "if product = 1 means that it`s only 1 or 0 zero
in array" << endl;

            swap((arr[imax]), arr[0]);
```

```
    cout << "Sorted array: " << endl;
    for (int i = 0; i < n; i++){
       cout <<  arr[i] << "\t";}

    delete []arr;
          return 0;
}
```

Результат роботи

```
Unsorted array:
 -5     3     1     1     -5     0     -2     -3     4     4
max = 5 index max = 0
Product of elements between first zero and last zero: 1
if product = 1 means that it`s only 1 or 0 zero in array
Sorted array:
 -5     3     1     1     -5     0     -2     -3     4     4
```

Передавання одновимірних масивів у функцію

Класичним способом передавання масиву у функцію виконується двома параметрами:

− покажчик на початковий елемент (зазвичай це ім'я масиву);

− кількість елементів у масиві.

Рекомендований синтаксис

тип_результату ім'я_функції (тип* arr, int n);

де тип – тип об'єктів, які зберігаються у масиві;

n – кількість елементів у масиві;

Зауваження: динамічні та звичайні одновимірні масиви передаються за однаковою схемою.

Приклад роботи з масивами

Створити такі функції;

− заповнення одновимірного масиву випадковими числам;

− друк масиву на екран;

− сортування одновимірного масиву за збільшенням;

− отримання суми всіх елементів масиву.

```
#include <iostream>
using namespace std;
void randArr(int* arr, int n)
```

```
{
    for (int i = 0; i < n; ++i)
        arr[i] = rand() % 21 - 10;
}
void outputArr(int* arr, int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;
}
void sortArr(int* arr, int n)
{
    for (int k = 0; k < n - 1; ++k)
        for (int i = 0; i < n - 1 - k; ++i)
        {
            if (arr[i + 1] < arr[i])
            {
                int c = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = c;
            }
        }
}
int sumArr(int* arr, int n)
{
    int res = 0;
    for (int i = 0; i < n; ++i) res += arr[i];
    return res;
}
int main()
{
    const int size = 10;
    int arrsive[size];
    randArr(arrsive, size);
    outputArr(arrsive, size);
    sortArr(arrsive, size);
    outputArr(arrsive, size);
    int s = sumArr(arrsive, size);
    cout << "Sum = " << s << endl;
```

```
    system("pause");
    return 0;
}
```

Результат роботи

```
-9  -6  -1   9  -2   0   0  -1   5   0
-9  -6  -2  -1  -1   0   0   0   5   9

Sum = -5
```

Передавання двовимірного динамічного масиву у функцію

На відміну від одновимірних масивів передавання звичайних і динамічних масивів у функцію дещо відрізняється.

Класичною схемою передавання **динамічного масиву у функцію** – це три параметри:

- двовимірний покажчик (ім'я масиву);
- кількість рядків;
- кількість стовпчиків.

Рекомендований синтаксис

тип_результату ім'я_фукнції (тип matr, int n, int m);**

де тип – тип об'єктів які зберігаються у масиві;
n – кількість рядків;
m – кількість стовпчиків.

Елемент матриці у функції застосовується звичайним чином matr[i][j], де i – номер рядка, j – номер стовпчика

Передавання двовимірного звичайного масиву у функцію

Класичною схемою передавання **звичайного масиву у функцію** – це три параметри:

- одновимірний покажчик (адреса початкового елементу масиву);
- кількість рядків;
- кількість стовпчиків.

Рекомендований синтаксис:

тип_результату ім'я_функції (тип* matr, int n, int m);

де тип – тип об'єктів які зберігаються у масиві;
n – кількість рядків;
m – кількість стовпчиків.

Зауваження: матриця у функції розглядається як звичайний одновимірний масив.

Поточний елемент у функції застосовується як **matr[i*m+j]**, де i – номер рядка;

j – номер стовпчика;

m – кількість стовпчиків у матриці.

Приклади створення функцій

```
//Функції створення і видалення динамічного масиву;
int** createMatr(int n, int m)
{
   if (n < 1 || m < 1) return NULL;
   int** matr = new int* [n];
   for (int i = 0; i < n; ++i)
      matr[i] = new int[m];
   return matr;
}
void deleteMatr(int**& matr, int n)
/*покажчик matr передається за посиланням, для того щоб можна було його змінити його значення на NULL */
{
   for (int i = 0; i < n; ++i)
      delete[] matr[i];
   delete[] matr;
   matr = NULL;
}

//Функції заповнення і виведення динамічного масиву;
void randMatr(int** matr, int n, int m)
{
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < m; ++j)
         matr[i][j] = rand() % 21 - 10;
}

void outputMatr(int** matr, int n, int m)
{
   for (int i = 0; i < n; ++i, cout << endl)
      for (int j = 0; j < m; ++j)
```

```
            cout << matr[i][j] << "\t";
        cout << endl;
    }

    //Передавання параметра функції через масив
    //Макс суму елементу рядку

    #include <iostream>
    #include <cstdlib>
    using namespace std;
    int sum_Max(int arr[3][3]){
        int max_result = 0;
         for(int i = 0; i < 3; i++) {
            int temp = 0;
            for(int j = 0; j < 3; j++)
                temp += arr[i][j];
            if (temp > max_result) max_result = temp;
         } return max_result;
    }
    int main() {

        int b[3][3] = {1, 21,3,2,11,10,4,0, -1};
        int currentSum = 0;
        for(int j = 0; j < 3; j++) currentSum += b[0][j];
            int maxStr = currentSum;
        int maxStrNum = 0;
        for(int i = 0; i < 3; i++) {
            currentSum = 0;
            for(int j = 0; j < 3; j++)currentSum += b[i][j];
            if(currentSum > maxStr) {
                maxStr = currentSum;
                maxStrNum = i;
            }
        }
    cout << " Sum " << maxStrNum + 1 << "raw ans equel is" <<
maxStr << endl;
        return 0;
    }
```

```
// Функції отримання суми всіх елементів масиву

int sumMatr(int** matr, int n, int m)
{
   int res = 0;
   for (int i = 0; i < n; ++i)
      for (int j = 0; j < m; ++j)
         res += matr[i][j];
   return res;
}

//Приклад застосування функцій

int main()
{
   int n = 5, m = 6;
   int** arr2d = createMatr(n, m);
   randMatr(arr2d, n, m);
   outputMatr(arr2d, n, m);
   int s = sumMatr(arr2d, n, m);
   cout << "Summa = " << s << endl;
   deleteMatr(arr2d, n);
   system("pause");
   return 0;
}
```

Приклади функцій для роботи зі звичайним двовимірним масивом:

- **заповнення двовимірного масиву випадковими числам;**
- **друк двовимірного масиву на екран;**
- **отримання суми всіх елементів двовимірного масиву.**

```
#include <iostream>
using namespace std;
void outputMatr(int* matr, int n, int m)
{
   for (int i = 0; i < n; ++i, cout << endl)
```

```
        for (int j = 0; j < m; ++j)
            cout << matr[i * m + j] << "\t";
    cout << endl;
}
void randMatr(int* matr, int n, int m)
{
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            matr[i * m + j] = rand() % 21 - 10;
}

int sumMatr(int* matr, int n, int m)
{
    int res = 0;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < m; ++j)
            res += matr[i * m + j];
    return res;
}
int main()
{
    const int n = 5, m = 6;
    int arr2d[n][m];
    randMatr(&arr2d[0][0], n, m);
    outputMatr(&arr2d[0][0], n, m);
    int s = sumMatr(&arr2d[0][0], n, m);
    cout << "Sum = " << s << endl;
    system("pause");
    return 0;
}
```

Результат роботи

```
 -9    -6    -1     9    -2     0
  0    -1     5     0    -8     9
 10    -6    10    -3    -7     5
  6     6     7     4     2    -1
 -8    -5    -5     3    -9     9

Sum = 14
```

Оброблення символів і рядків. Рядки

Рядок являє собою масив символів (тип char), який закінчується термінальним нуль-символом.

Нуль-символ – це символ, який має код, 0, і запис у вигляді керуючої послідовності '\0'. За розташуванням нуль-символа визначається фактична довжина рядка.

Кількість елементів символьного масиву складається з кількості символів у рядку плюс 1, тому що нуль-символ також є елементом масиву.

Для опису рядка використовуються звичайні засоби опису масивів, наприклад:

char str[10] = "Help";

Довжина рядка

0	1	2	3	4	5	6	7	8	9
'H'	'e'	'l'	'p'	'\0'					

Зарезервовано пам’яті для рядка.

Індексування такого масиву, як і будь-якого іншого, починається з нуля.

Символьні літерали беруться в одинарні лапки: char ch = 'A';

Рядкові літерали беруться в подвійні лапки: "Help".

Тип даних char

Змінна типу char займає 1 байт. Однак замість конвертації значення типу char в ціле число, воно інтерпретується як ASCII-символ.

ASCII (скор. Від «American Standard Code for Information Interchange») – це американський стандартний код для обміну інформацією, який визначає спосіб представлення символів англійської мови (+ кілька інших) у вигляді чисел від 0 до 127. Наприклад: код букви 'a' – 97, код букви 'b' – 98.

Символи від 0 до 31 використовуються переважно для форматування виводу.

Символи від 32 до 127 використовуються для виведення. Це літери, цифри, знаки пунктуації, які більшість комп’ютерів використовує для відображення тексту (англійською мовою).

Друкування повної таблиці ANSII

```
#include <cstdio>
#include <cstdlib>
using namespace std;
int main()
{
   for ( int c = 0; c <= 255; ++c)
   {
      unsigned char ch = c;
      printf("Code: %d\t Value: %c\n", ch,ch);
   }
   system("pause");
   return 0;
}
```

Створення рядків

```
char s1[] = "Help"; //довжина рядка 4, розмір масиву 5
char s2[10] = "Help";//довжина рядка 4, розмір масиву 10
char s3[] = { 'H','e', 'l' ,'p', '\0' };//довжина рядка 4, розмір масиву 5
const char*s4 = "Help";
```

Основні функції оброблення символьних типів string

У мові **C++** є кілька можливостей роботи із символьними даними. Класична робота зводиться до використання масиву символів. Для того, щоб скористатися стандартними функціями **C++** для роботи з рядками, слід їх файли підключити в директиві **#include <string>**. Основні функції з цього пакету наведено нижче.

- **char*strcat (char*_dest, const char*_src); –** функція реалізує з'єднання рядка dest з рядком src. Функція вертає покажчик на початок отриманого рядка (dest). Проміжний символ '\0' рядка dest видаляється.
- **char*strncat (char*_dest, const char*_src, size_t_maxlen); –** функція приєднує maxlen символів з рядка, на який вказує src, до рядка, на який вказує dest. Рядок dest повинен містити не менше maxlen вільних байтів. Якщо maxlen більше рядка src, виконується проста конкатенація.

− **char*strchr (const char*_, int_c); –** функція вертає покажчик на позицію першого входження символу “c” в рядок, на який вказує s. У рядок s включається і символ ‘\0’.

− **int strcmp (const char*_s1, const char*_s2); –** функція виконує порівняння двох рядків, на початок яких вказують s1 і s2. Функція вертає значення: менше нуля, якщо s1<s2; нуль, якщо s1==s2; більше нуля, якщо s1>s2.

− **int strncmp (const char*_s1, const char*_s2, size_t_maxlen); –** функція, що аналогічна функції strcmp () і відрізняється тим, що виконується порівняння перших maxlen байтів.

− **int stricmp (const char *_s1, const char *_s2); –** функція виконує порівняння двох рядків, на що вказують s1 і s2. Перед порівнянням символи перетворюються на малі. Функція вертає значення: більше нуля, якщо s1>s2; рівне нулю, якщо s1==s2; менш нуля, якщо s1<s2.

− **int strlen (const char*_s); –** функція вертає довжину рядка в байтах, на який вказує s. Нуль-термінатор не враховується.

− **char*stpcpy (char *_dest, const char *_src); –** функція копіює рядок, на який вказує src, в інше місце в пам’яті, на що вказує est. Функція вертає покажчик на кінець рядка, що скопіювався в dest.

− **char*strncpy (char*_dest, const char*_src, size_t_maxlen); –** функція копіює maxlen байт із рядка, на який вказує src, в інше місце в пам'яті, на яке вказує dest. Нуль-термінатор теж копіюється. Якщо maxlen менше довжини рядка src, до рядка src не приєднується символ “\0”. І якщо більше, то рядок src переноситься повністю, а символи, що залишалися, заповнюються символом “\0”. Функція вертає покажчик dest.

− **char *strlwr (char *_s); –** функція перетворює всі символи рядка, на початок якого вказує s, в малі літери. Функція повертає покажчик на початок цього рядка.

− **char *strups (char *_s); –** функція перетворює всі символи рядка, на початок якої вказує s, у великі літери. Функція вертає покажчик на отриманий рядок.

− **char *strset (char *_s, int_ch); –** функція заповнює рядок, на початок якого вказує s, символом ch. Функція вертає покажчик на отриманий рядок.

− **char *strtok (char *_s1, char *_s2); –** функція вертає наступну лексему із s1, яка відділена будь-яким символом з набора s2.

Копіювання рядків

char* strcpy(char* destination, const char* source);
char* strncpy(char* destination, const char* source, size_t num);
strcpy – виконує копіювання символів з рядка source у destination.
strncpy – виконує копіювання не більше num символів з рядка source у destination.

Зауваження: функція не добавляє в кінець рядка **destination** термінальний символ, у випадку коли довжина рядку **source** більша **num**.
Функції не перевіряє розмір масиву-результату.

Об'єднання рядків

char* strcat(char* destination, const char* source);
char* strncat(char* destination, const char* source, size_t num);
strcat – добавляє символи рядка source в кінець рядка destination. Результат в destination
strncpy – добавляє не більше num символів з рядка source в кінець рядка destination. Результат в destination
Зауваження: функція не добавляє в кінець рядка destination термінальний символ, у випадку коли довжина рядку source більша num.
Функції не перевіряє розмір масиву-результату.
Функції вертають адресу результату, тобто це адреса рядка destination

Порівняння рядків

int strcmp(const char* str1, const char* str2);
int strncmp(const char* str1, const char* str2, size_t num);
Функція strcmp порівнюють рядки str1 і str2 з точки зору абетки, та повертає цілу величину, що дорівнює:
< 0 – якщо str1 < str2;
= 0 – якщо str1 == str2;
> 0 – якщо str1 > str2; ,
Функція strncmp робить те саме, за винятком того, що порівнює не більше num символів з рядків.

Функції пошуку

const char* strchr(const char* str, int character);
const char* strrchr(const char* str, int character);

const char* strstr(const char* str1, const char* str2);

Функція strchr/strrchr виконує пошук першого/останнього входження символу character у рядку str.

Функція strstr виконує пошук першого входження рядка str2 у рядку str1.

Усі функції вертають адресу знайденого символу/рядка. Якщо такий відсутній, функції вертають значення NULL.

Розбивання на лексеми

char* strtok(char* str, const char* delimiters);

strtok – розбиття рядка str на лексеми(слова), обмежені символами, що входять до складу рядка delimiters.

delimiters може містити будь-яку кількість різних обмежників – ознак границь лексем,

функція strtok вставляє в кінець лексеми термінальний символ '\0'.

функція strtok вертає адресу знайденої лексеми, або NULL якщо лексему не знайдено.

Алгоритм застосування функції:

- на першому кроці викликати функцію, вказавши першим параметром рядок, у якому необхідно виконувати пошук, другим параметром вказати рядок із розділовими знаками
- char* curWord = strtok(str, " ,.!&?");
- для пошуку наступної лексеми необхідно викликати цю функцію з першим параметром NULL:

curWord = strtok(NULL, " ,.!&?");

це буде вказівкою функції, що необхідно продовжити пошук у рядку після термінального символ '\0', який вона вставила при попередньому виклику. Вона його запам’ятовує у вигляді статичної змінної.

Алгоритм роботи функції

- виконує пошук символу, який не входить у набір delimiters. Якщо такий знайдений, то запам’ятовує його адресу. Якщо ні – то функція вертає NULL і припиняє роботу.
- виконує пошук символу, який входить у набір delimiters. Якщо такий знайдений то вставляє замість нього символ '\0'.
- повертає адресу знайдену на першому кроці.

Функції перетворення рядків у числа та чисел у рядки (cstdlib)

atoi – перетворює рядок у число типу int
atol – перетворює рядок у число типу long
atof – перетворює рядок у число типу double
strtod – перетворює рядок у число типу double

```
int atoi(const char* str);
long int atol(const char* str);
double atof(const char* str);
double strtod(const char* str, char** endptr);
```

str – рядок з числовим представленням
endptr – вказує на елемент де зупинилось перетворення.

Якщо перетворення було невдале, функції повертають значення 0.

Функції перевірки символів ctype.h/ cctype

int isalnum(int c);	– 1 якщо буква чи Буква
int isalpha(int c);	– 1 якщо Буква
int isblank(int c);	– 1 якщо пробіл або табуляція (з C++11)
int iscntrl(int c);	– 1 якщо керуючий символ
int isdigit(int c);	– 1 якщо цифра
int isxdigit(int c);	– 1 якщо цифра в 16річному представленню
int isprint(int c);	– 1 якщо символ друкується
int isgraph (int c);	– 1 якщо символ має графічне представлення (isgraph без пробілу і табуляції)
int islower(int c);	– 1 якщо від ‘a’ до ‘z’
int isupper(int c);	– 1 якщо від ‘A’ до ‘Z’
int ispunct(int c);	– 1 якщо символ відноситься до пунктуації
int isspace(int c);	– 1 якщо символ пробіл, табуляція чи перехід на новий рядок

Порахувати довжину введеної фрази і кількість голосних у фразі (без використання функції бібліотеки string

```
#include <iostream>
#include <cstdlib>
using namespace std;
```

```
int strlength (const char *strArr)
{ int counter = 0;
   while (strArr[counter] != '\0')
   {counter ++;}
   return counter; }

int main() {
   int  i, j;
   int c=0;

 char strArr[256] = " ";
          cin.getline(strArr, 256);
          cout << strArr << endl;
 char strArr1[] = {'A', 'U', 'E', 'I', 'O', 'Y', 'a', 'e', 'o', 'i', 'u', 'y', '\0'};
  cout << "numbers of letters: " << strlength(strArr) << endl; //
довжина строчки
        for ( int i = 0; i < 13; i++) {
        for ( int j = 0; j < strlength(strArr); j++)
    {
       if (strArr[j] == strArr1[i]  !='\0')
      c++; // кількість голосних у фразі
      }
 }
 cout << "numbers of vowel " << c << endl;
   return 0;
}
```

Результат роботи

```
hello, i am a student.
hello, i am a student.
numbers of letters: 22
numbers of vowel 7
```

Сформувати масив, елементи якого дорівнюють кількості голосних літер у словах речення

```
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;
```

```
bool is_char_in_char_array(const char ch, char arr[], int size)
{
    for (int i = 0; i < size; i++){
        if (ch == arr[i]){
            return true;
        }
    }
    return false;
}


int main() {
    int  i, j;
    int curNum;
    int textLeng = 256;
    char text[textLeng] = " ";
    cin.getline(text, textLeng);
    char vowels[] = "euioayEUIOAY";
    char delimiters[]  = " ;,!\t\n";
    int* vowelsInWordCount = new int[textLeng];
    j = 0;
    for(i = 0; i<textLeng; i++){
     if (is_char_in_char_array(text[i], vowels, 12))
     {
        vowelsInWordCount[j] ++;
         continue;
        }
        if (is_char_in_char_array(text[i], delimiters, 6)){
           j++;
          continue;
        }
    }
    j ++;
    for(i = 0; i < j; i++){
        cout << vowelsInWordCount[i] << "\t";
    }
    system("pause");
    return 0;
}
```

Результат роботи

```
Hello, i am a student majoring in Computer Sciences
sh: 1: pause: not found
 2    0    1    1    1    2    3    1    3    3
```

Типи даних створені користувачем

typedef – перейменування типу
enum – іменовані константи/перерахування
struct – структура
union – об'єднання

Перейменування типу

typedef – дає змогу створити новий тип на базі існуючого (псевдонім для типу)

Синтаксис створення простого типу
typedef існуючий_тип новий_тип;
Синтаксис створення тип-масив
typedef існуючий_тип новий_тип[розмір];
Синтаксис створення типу від структури
typedef struct{поля} новий_тип;
Синтаксис створення покажчика на функцію
typedef тип_що_вертає_функція (*новий_тип) (параметри функції);
typedef існуючий_тип новий_тип;

Приклади створення нових типів

```
#include <iostream>
#include <stdlib.h>
using namespace std;
typedef unsigned int uint;
typedef uint myint;
int main()
{
   unsigned int a=5;
   uint b = 6;
   myint c = 7;
   //об'єкти a, b, c однакового типу
   cout << a + b + c << endl;
```

```
    system("pause");
    return 0;
}

// typedef існуючий_тип новий_тип[розмір];

#include <iostream>
#include <stdlib.h>
using namespace std;
typedef char Msg[20];
int main()
{
    char str1[20] = "help";
    Msg  str2 = "help";
    //об'єкти str1, str2 однакового типу
    cout << str1 << endl;
    cout << str2 << endl;
    system("PAUSE");
    return 0;
}

// typedef struct{поля} новий_тип;

#include <iostream>
#include <stdlib.h>
using namespace std;
typedef struct
{
    int x, y;
} Tpoint;
int main()
{
    Tpoint p = {10,20};
    cout << p.x << "\t" << p.y << endl;;
    system("PAUSE");
    return 0;
}
```

Переваги застосування typedef

- задання «довгому» імені типу більш короткий псевдонім;

typedef unsigned int Tcounter; //для створення лічильників
typedef vector<vector<vector<unsigned long long >>>::const_reverse_iterator Tit; //тип для ітератора

- маштабування коду, наприклад щоб змінити розмір лічильника typedef unsigned int Tcounter; достатня змінити його тип typedef unsigned long long Tcounter
- надавання імені типу сенсового навантаження
- typedef unsigned int Tcounter; //для створення лічильників
- typedef double Ttime; //для зберігання дати
- імплементація шаблонних типів до фактичного типу (C++)
- typedef vector<double> TDvector;
- TDvector – вектор який зберігає дійсні числа

Іменовані константи/перерахування enum

Дозволяє визначити кілька іменованих констант, для яких потрібно, щоб всі вони мали різні значення (при цьому конкретні значення можуть бути не важливі). Усі можливі значення якого задаються списком цілочисельних констант

Синтаксис

enum новий_тип { список_констант };

Новий синтаксис C++

enum class новий_тип { список_констант };

Зауваження: вказувати новий_тип не обов'язково.

Завдання для самостійної роботи 1

Написати програми мовою програмування C++ для задач, які наведено нижче.

Варіант 1

1. В одновимірному масиві, що складається з N дійсних елементів, обчислити:

− суму від'ємних елементів масиву;

− добуток елементів масиву, що розташовані між максимальним і мінімальним елементами.

Упорядкувати елементи масиву за зростанням.

2. Дана прямокутна цілочисельна матриця. Визначити:

− кількість рядків, які не містять жодного нульового елемента;

− максимальне із чисел, що зустрічається в заданій матриці більше одного разу.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість слів, які мають непарну довжину; виводить на екран частоту входження кожної літери; видаляє текст, що розміщено в круглих дужках.

Варіант 2

1. В одновимірному масиві, що складається з N дійсних елементів, обчислити:

− суму додатних елементів масиву;

− добуток елементів масиву, що розташовані між максимальним за модулем і мінімальним за модулем елементами.

Упорядкувати елементи масиву за спаданням.

2. Дана прямокутна цілочисельна матриця. Визначити кількість стовпців, які не містять жодного нульового елемента.

Характеристикою рядка цілочисельної матриці назвемо суму її додатних парних елементів. Переставляючи рядки заданої матриці, розташувати їх відповідно до зростання характеристик.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка перевіряє, чи співпадає кількість відкритих і закритих дужок у введеному рядку (перевірити для круглих і квадратних дужок); виводить на екран найдовше слово; видаляє всі слова, що складаються тільки з латинських літер.

Варіант 3

1. В одновимірному масиві, що складається з N цілих елементів, обчислити:

− добуток елементів масиву з парними номерами;

− суму елементів масиву, які розташовані між першим і останнім нульовими елементами.

Упорядкувати масив так, щоб спочатку розташовувались всі додатні елементи, а потім – всі від'ємні (елементи, рівні 0 вважати додатними).

2. Дана прямокутна цілочисельна матриця. Визначити:

− кількість стовпців, які містять хоча б один нульовий елемент;

− номер рядка, в якому знаходиться найдовша серія однакових елементів.

3. Із клавіатури вводиться текстовий рядок. Написати програму, яка підраховує кількість різних слів, що входять до заданого тексту; виводить на екран кількість використаних символів; видаляє всі слова, що мають подвоєні літери.

Варіант 4

1. В одномірному масиві, що складається з N дійсних елементів, обчислити:

− суму елементів масиву з непарними елементами;

− суму елементів масиву, які розташовані між першим і останнім від'ємними елементами.

Переставити перші M елементів у кінець масиву (M вводиться з клавіатури, $M<N$).

2. Дана прямокутна цілочисельна матриця. Визначити:

− добуток елементів у тих рядках, які не містять від'ємних елементів;

− максимум серед сум елементів діагоналей, паралельних головній діагоналі матриці.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість слів у тексті; виводить на екран слово, що містить найбільшу кількість голосних літер; видаляє з тексту всі непотрібні пробіли.

Варіант 5

1. В одновимірному масиві, що складається з N дійсних елементів, обчислити:

− максимальний елемент масиву;

− суму елементів масиву, що розташовані до останнього додатного елемента.

Видалити з масиву всі елементи, модуль яких знаходиться в інтервалі $[a,b]$. Елементи, які звільняться в кінці масиву заповнити нулями.

2. Дана прямокутна цілочисельна матриця. Визначити:

− суму елементів у тих стовпцях, які не містять від'ємних елементів;

− мінімум серед сум модулів елементів діагоналей, паралельних побічній діагоналі матриці.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість розділових знаків у тексті; виводить

всі слова, що мають парну кількість літер; міняє місцями першу і останню літери кожного слова.

Варіант 6

1. В одновимірному масиві, що складається з N дійсних елементів, обчислити:

− мінімальний елемент масиву;

− суму елементів масиву, що розташовані між першим і останнім додатними елементами.

Перетворити масив так, щоб спочатку розташовувались всі елементи, рівні нулю, а потім – решта.

2. Дана прямокутна цілочисельна матриця. Визначити:

− суму елементів у тих стовпцях, які містять хоча б один від'ємний елемент;

− номера рядків і стовпців всіх сідлових точок матриці. Матриця А має сідловий елемент, якщо Аij – мінімальний елемент в і-му рядку і максимальний в j-му стовпці.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість великих літер у тексті; виводить на екран слова, що мають найменшу кількість літер; видаляє всі слова, що починаються з малої літери.

Варіант 7

1. В одновимірному масиві, що складається з N цілих елементів, обчислити:

− номер максимального елемента масиву;

− добуток елементів масиву, що розташовані між першим і другим нульовими елементами.

Перетворити масив так, щоб у його першій половині розташовувались елементи, що стоять в непарних позиціях, а в другій половині – елементи, що стоять у парних позиціях.

2. Для заданої матриці розміру знайти таке k, що k-й рядок матриці співпадає з k-м стовпцем. Знайти суму елементів у тих рядках, які містять хоча б один від'ємний елемент.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість чисел у тексті (не цифр, а саме чисел); виводить на екран всі слова, що складаються тільки з латинських літер; видаляє кожне друге слово.

Варіант 8

1. В одновимірному масиві, що складається з N дійсних елементів, обчислити:

− номер мінімального елемента масиву;

− суму елементів масиву, що розташовані між першим і другим від'ємними елементами.

Перетворити масив так, щоб спочатку розташовувались всі елементи, модуль яких не перевищує 10, а потім – решта.

2. Характеристикою стовпця цілочисельної матриці назвемо суму модулів його від'ємних непарних елементів. Переставляючи стовпці заданої матриці, розташувати їх згідно із ростом характеристик.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість цифр у тексті; виводить на екран слова, що починаються з приголосних літер; знищує всі слова, які починаються і закінчуються на одну й ту ж літеру.

Варіант 9

1. В одновимірному масиві, що складається з N дійсних елементів, обчислити:

− максимальний за модулем елемент масиву;

− суму елементів масиву, що розташовані між першим і другим додатними елементами.

Перетворити масив так, щоб всі елементи, рівні нулю та одиниці, розташовувались після всіх інших.

2. Коефіцієнти системи лінійних рівнянь задані як прямокутні матриці. За допомогою допустимих перетворень звести матрицю до трикутного вигляду. Знайти кількість рядків, середнє арифметичне елементів яких менше заданої величини.

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість слів у тексті, які закінчуються на голосну літеру; виводить на екран усі слова, довжина яких менша п'яти символів; видаляє всі слова, які містять хоча б одну латинську літеру.

Варіант 10

1. В одновимірному масиві, що складається з N цілих елементів, обчислити:

− мінімальний за модулем елемент масиву;

− суму модулів елементів масиву, розташованих після першого елемента, рівного нулю.

Перетворити масив так, щоб в першій його половині розташовувались елементи, що стоять на парних позиціях, а в другій половині – елементи, що стоять в непарних позиціях.

2. Здійснити циклічний зсув елементів прямокутної матриці на п елементів вправо або вниз (залежно від введеного режиму).

3. Із клавіатури вводиться текстовий рядок. Скласти програму, яка підраховує кількість слів у тексті, які починаються з голосної літери; виводить на екран усі слова, які містять непарну кількість приголосних літер; видаляє всі числа з тексту.

Завдання для самостійної роботи

За допомогою класу string стандартної бібліотеки шаблонів виконати такі завдання

Завдання 1.

1. Даний текст. Підрахувати загальну кількість входжень у нього символів «+» і «-».

2. Задано назву футбольного клубу. Написати його назву «стовпчиком».

3. Дано речення. Надрукувати всі його букви «а».

4. Дано речення. Вивести «стовпчиком» його 3, 6, 9 і т. д. символи.

5. Дано речення. Надрукувати всі його букви «п» та «н».

6. Задано речення. Чи правильно, що в ньому присутні п’ять однакових символів, що розташовані поряд?

Завдання 2.

1. Дане речення. Визначити кількість букв *н*, що передують перший комі речення. Розглянути два випадки:

1) відомо, що коми в реченні є;

2) ком у реченні може не бути.

2. Дане речення, у якому є одна буква *к* і одна буква *і*. Визначити, яка з них зустрічається раніше (при перегляді зліва на право).

3. Дано речення. Визначити, скільки символів «пробіл» містить дане речення.

4. Дане речення. Визначити порядкові номери першої трійки однакових сусідніх символів. Якщо таких символів немає, то потрібно надрукувати відповідне повідомлення.

5. Даний текст. Визначити кількість букв *т* у першому реченні. Розглянути два випадки:

1) відомо, що буква *т* у цьому реченні є;

2) букв *т* у тексті може не бути.

6. Дане речення. Визначити, чи є в ньому буквосполучення *ні* або *так*. У випадку позитивної відповіді знайти також його порядковий номер.

МОДУЛЬ 3. КЛАСИ. ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

Тема 5. Класи й абстрагування даних

Структура. Робота зі структурою

Структура – це упорядкована сукупність довільних типів даних, що об'єднані в одній області пам'яті. Тип структури вводиться описом такого вигляду:

struct [ім'я_структури] {тип_1 ім'я_поля_1;
тип_2 ім'я_поля_2;...;
тип_n ім'я_поля_n;};

де **ім'я_структури** – ім'я структури шаблону, що задовольняє правилам задання ідентифікаторів мови **C++; тип_1, тип_2, ..., тип_n** – будь-які призначені типи; **ім'я_поля_1, ..., ім'я_поля_n** – ідентифікатори полів, що задовольняють правилам завдання ідентифікаторів.

Опис структури являє собою задання нового типу **«ім'я_структури»** і не призводить до виділення пам'яті, а лише дає інформацію компілятору про типи і кількість полів. Ця інформація використовується компілятором під час опису структурованих змінних для резервування необхідного місця в пам'яті й організації доступу до необхідних полів структурної змінної. Усі поля структури за замовчування типу public.

Доступ до полів структурних змінних можна забезпечити двома засобами: використовуючи оператор розіменування:

ім'я _структурної_змінної. ім'я_поля;

або використовуючи оператор покажчика на структуру:

покажчик_на_структуру-> ім'я_поля;

На відміну від масивів чи множин, усі елементи яких однотипні, структура може містити елементи різних типів.

Елементи структури називають полями структури і можуть мати довільний тип, крім типу цієї ж структури, але можуть бути покажчиками на неї. Якщо при описі структури відсутній

тип структури, обов'язково повинен бути вказаний список змінних, покажчиків або масивів визначеної структури.

Звернення до окремих полів структури замінюються на їхні адреси ще на етапі компіляції.

Самою найважливішою операцією для структури є операція доступу до вибраного поля структури – операція кваліфікації.

Над вибраним полем структури можливі будь-які операції, які допустимі для типів цього поля.

Більшість мов програмування підтримує деякі операції, які працюють із структурою, як з єдиним цілим, а не з окремими її полями. Це операція присвоєння значення одного запису іншому однотипному запису, при цьому відбувається поелементне копіювання.

У мові C++ визначення структури складається з двох кроків:

оголошення структури (завдання нового типу даних заданого користувачем), структура складається з полів;

```
struct Point
{
double x; double y; //визначені поля структури
}
struct RGBColor
{
char red; char green; char blue; //визначені поля структури
}
```

визначення змінних типу структура;

```
struct Circle
{
Point center; //визначено поле структури як структуру
double radius; //визначено звичайне поле структури
int thickness; //визначено звичайне поле структури
RGBColor color; //визначено поле структури як структуру
}
```

Структури можна вкладати в інші структури.

Для звернення до полів структури слід указати ім'я змінної і через крапку ім'я поля:

Circle crcl; //створенно змінну crcl з типом даних Circle

crcl.radius = 10.0; //полю radius, структури Circle, присвоєно значення crcl.center.x = -1.5;

crcl.center.y = 0.7; //так, як поле center структури Circle оголошене структурою Point, то воно має два відповідних поля, так ми звертаємось через структуру Circle до структури Point.

Задано n комплексних чисел, знайти число найбільшого модуля

```
#include <iostream>
#include <cmath>
using namespace std;
int main ()
{
struct complex
{
float x; // дійсна частина
float y; // уявна частина
};
// Оголошення масиву комплексних чисел
complex p[100];
int i, n, nmax;
float max;
cout<<"n = ";
cin>>n;
for (i = 0; i<n; i ++)
{
cout<< "Input complex number \n";
//введення дійсної частини і-го комплексного числа
cin>> p[i].x;
//введення уявної частини і-го комплексного числа
cin>>p[i].y;
cout << p[i].x << "+" << p[i].y << "i" <<endl;
}
max = pow(p[0].x*p[0].x + p[0].y*p[0].y, 0.5); nmax = 0;
for (i = 1; i<n; i ++)
if (pow(p[i].x*p[i].x + p[i].y*p[i].y, 0.5)>max)
{
max = pow(p[i].x*p[i].x + p[i].y*p[i].y, 0.5); nmax = i;
}
cout<<"Number of maximum module "<<nmax<<"\nValues of
module "
<<max<< endl;
return 0;
}
```

Результат роботи:

```
n = 3
```

```
Input complex number
2
5
2+5i
Input complex number
3
-8
3+-8i
Input complex number
7
-7
7+-7i
Number of maximum module 2
Values of module 9.8995
```

Приклади створення структур

```
//Пуста структура
    struct {};

// Tpoint – створений тип
                p1, p2 – створені об’єкти
           p2 = {-5,4} – об’єкт з ініціалізацією
    struct Tpoint
    {
       int x;
       int y;
    }p1, p2 = {-5,4};

//Створення структури без вказання типу, проте з об’єктом Ivan
    struct
    {
       char pib[100];
       int  bal;
    }ivan= { "Ivanov", 84 };
```

Доступ до полів структури

Доступ до полів структури виконується за допомогою двох операцій:

– операція вибору (.) (точка, крапка)
– операції ->

Операцій вибору. (точка, крапка) застосовується при зверненні до поля через ім'я об'єкту структури.

Операція -> застосовується при зверненні до поля через адресу об'єкту, наприклад, який зберігається у покажчику.

Для структур визначена операція = (присвоїти), яка працює як побітове копіювання.

У мові С++ операцію -> можна перевизначити, а операцію – ні.

Бітові поля

Бітові поля – це особливий вид полів структури. Вони використовуються для щільної упаковки даних, наприклад, прапорців типу «так/ні». Мінімальна комірка пам'яті – 1 байт, а для зберігання прапорця досить одного біта. При описі бітового поля після імені через двокрапку вказується довжина поля в бітах (ціла позитивна константа):

Бітові поля можуть бути будь-якого цілого типу.

Доступ до полю здійснюється звичайним способом – за іменем. Адресу поля отримати не можна, однак в іншому бітові поля можна використовувати точно так же, як звичайні поля структури.

Слід враховувати, що операції з окремими бітами реалізуються набагато менш ефективно, ніж із байтами і словами, так як компілятор повинен генерувати спеціальні коди, і економія пам'яті під змінні обертається збільшенням обсягу коду програми. Розміщення бітових полів у пам'яті залежить від компілятора й апаратури.

Структури дозволяють зручно групувати дані (змінні) за деякими критеріями. У пам'яті такі дані, зазвичай, розміщуються один за одним згідно з оголошенням.

У мові С++ існує засіб стиснення розміру пам'яті, що виділяється під оголошені структурі змінні. Це бітові поля у структурах.

Бітові поля використовуються для зберігання цих типів. Під час використання бітових полів у розмірі типу для них виділяється певна кількість біт.

Бувають випадки, коли для зберігання даних розмір пам'яті, виділений для них, занадто великий. Наприклад, щоб зберігати дані про день тижня, розміру 1 байта чи 8 біт занадто багато.

Існує 7 днів тижня, тому достатньо 3 біти для представлення цих даних (2 у ступені 3 = 8 значень).

Загальний синтаксис оголошення бітового поля у структурі виглядає так

unsigned int field : value;

де field – ім'я поля (змінної) у структурі;

value – кількість біт, виділених на уявлення безлічі значень бітового поля.

У свою чергу оголошення структури, що містить бітові поля виглядає приблизно так:

```
struct StructName
{
  unsigned int field_1 : value_1;
  unsigned int field_2 : value_2;
  ...
  unsigned int field_N : value_N
};
```

де StructName – ім'я структури;

field_1, field_2, field_N – назви змінних, які відповідають бітовим полям структури;

value_1, value_2, value_N – цілочисленні значення, що становлять кількість виділених біт під розміри відповідно до полів field_1, field_2, field_N.

Як бітові поля рекомендується використовувати тип unsigned int. Це не призведе до спотворення результату у випадку, коли перший біт буде встановлений у 1, що означатиме негативне число. Однак, при використанні інших цілих типів, компілятор не буде видавати помилки.

Якщо в структурі оголосити лише одне бітове поле, це не дасть жодного ефекту. Дані в пам'яті все одно зберігатимуться у вигляді, кратному 8 біт або 1 байту (вирівнювання). Користь від бітових полів збільшується, так як у структурі зберігається багато цілих даних. При великій кількості бітових полів можна заощадити значну кількість байт під той самий набір даних без їх втрати. У свою чергу, використання масивів структур із бітовими полями дозволяє суттєво зменшити розмір даних, що зберігаються, і прискорити виконання команд обробки цих даних.

Приклад використання бітових полів у структурі Date, що описує дату дня та номер дня тижня

У задачі оголошується структура Day, яка описує такі дані про день:

номер дня на тижні від 1 до 7. Тут 1 це понеділок, 2 – вівторок тощо, 7 – неділя;

число за місяць – 1..31;

номер місяця – 1.12;

рік. Діапазон від 0 до 3000 року.

Для представлення дня тижня достатньо трьох біт, оскільки 3 біти дозволяють отримати 8 значень (2 ^ 3 = 8). Тобто 8 значень покривають 7 значень, які потрібні для збереження номера дня тижня.

Виходячи з наведених вище міркувань про день тижня, можна сформувати кількість біт для інших полів структури Day:

число за місяць досить 5 біт (2^5 = 32, 32>31);

номер місяця достатньо 4 біт (2 ^ 4 = 16, 16> 12);

задля збереження року досить 12 біт (2^12 = 4096, 4096>3000).

Приклад використання бітових полів у структурі Date, що описує дату дня та номер дня тижня

```
#include <iostream>
using namespace std;

struct Day
{
  unsigned int weekDay : 3; // День тижня 1..7
  unsigned int number : 5; // Число 1..31
  unsigned int month : 4; // Номер місяця 1..12
  unsigned int year : 12; // Рік 0..3000
};

int main()
{
  // Поля бітів у структурах
  // Оголосити структуру
  Day d;
```

```
    // Заповнити структуру даними 26.12.2022, понеділок
    d.weekDay = 1;
    d.number = 26;
    d.month = 12;
    d.year = 2022;

    // Вивести дані
    cout << "d.weekDay = " << d.weekDay << endl;
    cout << "d.number = " << d.number << endl;
    cout << "d.month = " << d.month << endl;
    cout << "d.year = " << d.year << endl;

    // Вивести розмір структури
    cout << "sizeof(Day) = " << sizeof(d) << endl; // 4 байти
}
```

Результат роботи:

```
d.weekDay = 1
d.number = 26
d.month = 12
d.year = 2022
sizeof(Day) = 4
```

Приклад порівняння структури з бітовими полями із структурою, яка не використовує бітові поля. Структура Time

Нехай потрібно реалізувати структуру Time, що описує часовий інтервал у вигляді:

hours: minutes: seconds: milliseconds

де hours – кількість годин в одній добі 0..23;
minutes – кількість хвилин за годину 0..59;
seconds – кількість секунд за хвилину 0..59;
milliseconds – кількість мілісекунд 0..999.

Найкраще для опису подібних даних у структурі використовувати поля бітів. Виходячи з максимально допустимих значень, для різних часових інтервалів можна виділити різну кількість біт у їх поданні у структурах:

поле hours – 5 біт (2^5 = 32 > 23). Із 5 бітів можна сформувати до 2^5=32 різних значень, що цілком достатньо для представлення 24 значень цього поля;

minutes – 6 біт (2 ^ 6 = 64> 59);

seconds – 6 біт (2^6 = 64> 59);

milliseconds – 10 біт (2^10 = 1024 > 999).

У програмі з метою демонстрації створюються дві структури Time і Time2, в яких реалізовані необхідні поля для представлення часу. Проте структура Time використовує бітові поля, а структура Time2 використовує звичайні значення типу unsigned int.

Приклад порівняння структури з бітовими полями із структурою, яка не використовує бітові поля.

Структура Time

```
#include <iostream>
using namespace std;

// Структура Time використовує бітові поля
struct Time
{
  unsigned int hours : 5;
  unsigned int minutes : 6;
  unsigned int seconds : 6;
  unsigned int milliseconds : 10;
};

// Структура Time2 не використовує бітові поля
struct Time2
{
  unsigned int hours;
  unsigned int minutes;
  unsigned int seconds;
  unsigned int milliseconds;
};

int main()
{
  // Порівняння структур із бітовими полями та без них
  // Оголосити змінні
```

```
  Time tm;
  Time2 tm2;

  // Використання структур у програмі
  // Структура типу Time
  tm.hours = 12;
  tm.seconds = 36;
  tm.milliseconds = 777;
  tm.minutes = 3;

  // Структура типу Time2
  tm2.hours = 20;
  tm2.seconds = 39;
  tm2.milliseconds = 555;
  tm2.minutes = 55;

  // Визначити та вивести розмір структур Time, Time2
  //     з допомогою оператора sizeof()
  size_t sizeTime = sizeof(Time);     // 4
  size_t sizeTime2 = sizeof(Time2);   // 16
  cout << "sizeTime = " << sizeTime << endl;
  cout << "sizeTime2 = " << sizeTime2 << endl;
}
```

Результат роботи:

```
sizeTime = 4
sizeTime2 = 16
```

Як видно з результату, відмінностей між використанням звичайних структур і бітових структур немає (доступ до полів, заповнення даними тощо).

Однак, розмір бітової структури (змінна tm) у пам'яті займає всього-навсього 4 байти порівняно з розміром звичайної структури в 16 байт.

У структурі Time відбувається додавання бітів у полях. Сумарна кількість бітів становить:

$5 + 6 + 6 + 10 = 27$.

Якщо розділити 27 на 8 (кількість біт у байті), то вийде 3 з надлишком. Тобто, для представлення 27 біт потрібно 4 байти щоб коректно відобразити дані.

Розмір структури

Розмір структури визначається як сума розмірів усіх полів із можливим вирівнювання до розміру, який кратний машинному слову (кратне 4, 8).

Розмір структури

```
#include <iostream>
#include <cstdlib>
using namespace std;
struct T1 { char ch; };
struct T2 { int x,y; };
struct T3 { int x; double y; };
struct T4 { char mas[6]; };
struct T5 { int x; char mas[5]; };
struct T6 { double x; char mas; };
struct T7 { int x; char c; };

int main()
{
   cout << sizeof(T1) << endl;    // 1
   cout << sizeof(T2) << endl;    // 8
   cout << sizeof(T3) << endl;    // 16
   cout << sizeof(T4) << endl;    // 6
   cout << sizeof(T5) << endl;    // 12
   cout << sizeof(T6) << endl;    // 16
   cout << sizeof(T7) << endl;    // 8
   system("pause");
   return 0;
}
```

Результат роботи:

```
1
8
16
6
12
16
8
```

Об’єднання union

Об’єднання (union) являє собою окремий випадок структури, всі поля якої розташовуються за однією адресою.

Формат опису такий же, як у структури, тільки замість ключового слова struct використовується слово union.

union ім’я_новий_тип { поля}список_обєктів;

Довжина об’єднання визначається найбільшою довжиною з його полів. У кожен момент часу в змінній типу об’єднання зберігається тільки одне значення, і відповідальність за його правильне використання лежить на програмістові.

Об’єднання застосовують для економії пам’яті в тих випадках, коли відомо, що більше одного поля одночасно не потрібно.

У порівнянні зі структурами на об’єднання накладаються деякі обмеження:

- об’єднання може ініціалізуватися тільки значенням його першого елемента;
- об’єднання не може містити бітові поля;
- об’єднання не може містити віртуальні методи, конструктори, деструктори і операцію присвоювання (C++);
- об’єднання не може входити в ієрархію класів (C++).

Приклад об’єднання структур

```
#include <iostream>
#include <cstdlib>
using namespace std;
union T
{
   int a;
   double b;
};
int main()
{
   cout << sizeof(T) << endl; //8
   T t;
   t.a = 5;
   cout << t.a << endl; //5
   t.b = 2.5; // в цей момент значення поля a затерте полем b
   cout << t.a << endl; //0 логічна помилка
   cout << t.b << endl; //2.5
```

```
    system("pause");
    return 0;
}Результат роботи:
8
5
0
2.5
```

Приклад оголошення і використання покажчика на об'єднання

Робота об'єднань з некерованими (*) покажчиками є така сама, як і робота структур із некерованими покажчиками.

Приклад використання некерованого покажчика (*) для об'єднання типу Ints

```
// покажчик на об'єднання
Ints *pI; // некерований покажчик
// виділити пам'ять для об'єднання
pI = new Ints;
// доступ до полів з допомогою покажчика
pI->a = 200;
pI->b = 3400;
```

Приклади оголошення вкладених об'єднань (структур, класів) у шаблоні об'єднання

Шаблон об'єднання може містити поля, що є структурами, об'єднаннями та класами.

У прикладі нижче оголошується шаблон об'єднання з іменем Types, що містить два вкладені об'єднання Floats та Ints, структуру ArrayOfChars і клас MyPoint.

Приклади оголошення вкладених об'єднань (структур, класів) у шаблоні об'єднання

Оголошення структури й об'єднань

```
// об'єднання цілочисельних типів
union Ints
{
   unsigned short int a;
   unsigned int b;
   unsigned long int c;
};
```

```
// структура, що містить 2 рядки
struct ArrayOfChars
{
   char A[10];
   char B[8];
};

// оголошення типу "об'єднання Floats"
union Floats
{
   float f; // розглядається 4 байти
   double d; // розглядається 8 байт
};
```

Оголошення шаблону класу

```
// клас оголошується в окремому модулі, наприклад MyPoint.h
#pragma once
class MyPoint
{
   public:
      int x;
      int y;

      // методи класу
      int GetX(void) { return x; }
      int GetY(void) { return y; }
      void SetXY(int nx, int ny) { x = nx; y = ny; }
};
```

Оголошення типу об'єднання Types із вкладеними складними типами Ints, Floats, ArrayOfChars, MyPoint.

```
// підключити модуль з оголошеним класом MyPoint
#include "MyPoint.h"
// оголошення типу "об'єднання Types"
union Types
{
   Floats Fl; // об'єднання
   Ints I;   // об'єднання
   ArrayOfChars A; // структура
   MyPoint MP;   // клас
};
```

Приклад використання об’єднання Types у деякому програмному коді:

```
// оголосити змінну типу "об’єднання Types"
Types T;
// змінити значення полів змінної T
T.Fl.f = (float)20.35; // об’єднання Floats
T.I.b = 230; // об'єднання Ints
T.A.A[2] = 'A'; // структура ArrayOfChars
T.MP.SetXY(3,8); // клас MyPoint
int d;
d = T.MP.GetX(); // d = 3
```

Масив об’єднань

Приклад оголошення і використання масиву об’єднань.

```
Floats F[5]; // оголошується масив з 5 об’єднань типу Floats
// заповнення значень полів
for (int i=0; i<5; i++)
{
   F[i].d = i*0.2 + i*i;
}
```

Вкладені структури

Приклад опису вкладених структур

```
// Варіант 1
#include <iostream>
#include <stdlib.h>
using namespace std;
struct T1
{
   int a;
   struct
   {
      int b;
   };
   char c;
};
int main()
```

```
{
   T1 t;
   t.a = 1;
   t.b = 2;
   t.c = 3;
   cout << sizeof(t) << endl;//12
   system("PAUSE");
   return 0;
}
```

// Варіант 2

```
#include <iostream>
#include <stdlib.h>
using namespace std;
struct T2
{
   int a;
   struct
   {
      int b;
   }d;
   char c;
};
int main()
{
   T2 t;
   t.a = 1;
   t.d.b = 2;
   t.c = 3;
   cout << sizeof(t) << endl;//12
   system("PAUSE");
   return 0;
}
```

// Варіант3

```
#include <iostream>
#include <stdlib.h>
```

```
using namespace std;
struct T3
{
   int a;
   struct T
   {
      int b;
   };
   char c;
};
int main()
{
   T3 t;
   t.a = 1;
   t.c = 3;
   cout << sizeof(t) << endl;//8
   T3::T t2;
   t2.b = 2;
   cout << sizeof(t2) << endl;//4
   system("PAUSE");
   return 0;
}

// Варіант 4
#include <iostream>
#include <stdlib.h>
using namespace std;
struct T4
{
   int a;
   struct T
   {
      int b;
   }d;
   char c;
};
int main()
{
   T4 t;
```

```
    t.a = 1;
    t.d.b = 2;
    t.c = 3;
    cout << sizeof(t) << endl;//12
    T4::T t2;
    t2.b = 2;
    cout << sizeof(t2) << endl;//4
    system("PAUSE");
    return 0;
}
```

Приклад створення структури, що містить дані про підприємства, їхній дохід і виплати по зарплатні

```
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;

// structure
 struct MyFactors
{
    string name;
    double money;
    double salary;
};
//  function for work with instances of structure

void show (MyFactors str)
{
    cout << "Firm: " << str.name << " \t" << "Money: " << str.money << " \t" << "Salary: " << " \t" <<str. salary << endl;
}

int main()
{
    MyFactors Roshen={"Roshen", 1000.25, 500.5};
    MyFactors ATB={"ATB", 2000.25, 1500.5};
    MyFactors Silpo={"Silpo", 3000.25, 2500.5};
```

```
    MyFactors Prostor={"Prostor", 900.25, 850.5};
    MyFactors Ekvator={"Ekvator", 700.25, 750.3};

    show(Roshen);
    show(ATB);
    show(Silpo);
    show(Prostor);
    show(Ekvator);

    return 0;
}
```

Результат роботи:

```
Firm: Roshen    Money: 1000.25   Salary:   500.5
Firm: ATB       Money: 2000.25   Salary:   1500.5
Firm: Silpo     Money: 3000.25   Salary:   2500.5
Firm: Prostor   Money: 900.25    Salary:   850.5
Firm: Ekvator   Money: 700.25    Salary:   750.3
```

Приклад програми для введення в комп'ютер інформації про автомобілі з такими даними: марка, рік випуску, потужність. Надрукувати марки тих автомобілів, рік випуску та потужність яких співпадає із введеними в рядку запиту

```
#include <iostream>
#include <cstdlib>
#include <string.h>
using namespace std;
struct Car
{
char name[100];
int year;
int power;
};
int main()
{
int n;
cout << "\nInput count auto: n = ";
cin>>n;
```

```
    Car * car = new Car[n];
    int i,yy,pp;
    bool isFound=false;
    // ввід інформації
    for(i=0; i<n; i++)
    {
    cout << "\nInput name auto:";
    cin >> car[i].name;
    cout<< "\nInput year auto:";
    cin >> car[i].year;
    cout << "\nInput power auto':";
    cin >> car[i].power;
    }
    //Рядок запиту: рік випуску і потужність
    cout << "\nInput year for searching:";
    cin >> yy;
    cout << "\nInput pofer for searching':";
    cin >> pp;
    for ( i=0; i < n; ++i)
    if((car[i].year== yy) && (car[i].power == pp))
    {
    cout <<"The name of car is\n"<< car[i].name <<"\n";
    isFound=true; /* наявність автомобіля із заданими
параметрами*/
    }
    if(isFound==false)
    cout<<"Auto not founs\n ";
    delete [] car;

    return 0;
    }
```

У звіті зберігається інформація: найменування підприємства, прибуток підприємства за місяць, нарахування на зарплатню. Відсортувати цей звіт у порядку зменшення прибутку і вивести на екран інформацію про три найбільш прибуткові підприємства

```
    #include <iostream>
    #include <algorithm>
```

```
#include <string.h>
using namespace std;
struct Company
{
char Name[40];
double Income;
 double Spend;
};

bool compare( Company a, Company b){
  if((a.Income - a.Spend) > (b.Income - b.Spend))
       return 1;
  else
        return 0;
        }

        int main()
        {
  int n;
         char tmpstr[1000];
  do
        {
    cout << "\nInput count of companies: n = ";
   cin.getline(tmpstr,999);
    n = atoi(tmpstr);
  } while (n<=0);

 Company * companies = new Company[n];

  for (int i = 0; i < n; i++)
  {
    cout << "Input company name for "<<i+1<<":"<<endl;
   cin.getline(companies[i].Name, 39);

  cout << "Input company income for "<<i+1<<":"<<endl;
  cin.getline(tmpstr, 999);
   companies[i].Income = atof(tmpstr);

   cout << "Input company spend for "<<i+1<<":"<<endl;
```

```
       cin.getline(tmpstr, 999);
        companies[i].Spend = atof(tmpstr);
      }

     sort(companies, companies+n, compare);

      int topCompanyCount;
     if(n>3)
           topCompanyCount = 3;
     else
         topCompanyCount = n;

      for (int i = 0; i < topCompanyCount; i++)
     {
     cout<<"Top company number: "<<i+1<< endl;
       cout<<"Company name: "<<companies[i].Name << endl;
       cout<<"Company income: "<<companies[i].Income << endl;
       cout<<"Company spend: "<<companies[i].Spend << endl;
      cout<<"Company Profit: "<<companies[i].Income -
companies[i].Spend<<endl<<endl;
      }

      return 0;
          }
```

Результат роботи:

```
Input count of companies: n = 5
Input company name for 1:
ATB
Input company income for 1:
125000
Input company spend for 1:
124500
Input company name for 2:
SILPO
Input company income for 2:
245000.25
Input company spend for 2:
137000.98
Input company name for 3:
```

```
ASTERA
Input company income for 3:
568.65
Input company spend for 3:
125.78
Input company name for 4:
PRIVATBANK
Input company income for 4:
987654.65
Input company spend for 4:
654789.25
Input company name for 5:
ELDORADO
Input company income for 5:
369.58
Input company spend for 5:
125.45
Top company number: 1
Company name: PRIVATBANK
Company income: 987655
Company spend: 654789
Company Profit: 332865

Top company number: 2
Company name: SILPO
Company income: 245000
Company spend: 137001
Company Profit: 107999

Top company number: 3
Company name: ATB
Company income: 125000
Company spend: 124500
Company Profit: 500
```

Класи

Клас є загальною схемою чи шаблоном, на основі якого потім створюються об’єкти. Клас містить оголошення полів і методів, які називаються членами класу. Поле класу – аналогіч-

не змінній, а метод – аналог функції. Опис класу починають з ключового слова **class**, після чого вказують ім'я класу, а потім у фігурних дужках **{}** описують тіло класу. У кінці ставиться крапка з комою (**;**).

У тілі класу описують поля і методи класу. Перед початком опису ставлять ключове слово public, після якого ставлять двокрапку (**:**). Це ключове слово ідентифікує блок відкритих членів класу – таких, доступ до яких є не тільки в межах класу, але і для зовнішнього коду.

```
class DateClass
{
public:
   int m_day;
   int m_month;
   int m_year;
};
```

У мові C++ класи дуже схожі на структури, за винятком того, що вони забезпечують набагато більшу потужність і гнучкість. Фактично, подана нижче структура і клас ідентичні за функціоналом:

```
struct DateStruct
{
   int day;
   int month;
   int year;
};

class DateClass
{
public:
   int m_day;
   int m_month;
   int m_year;
};
```

Єдиною істотною відмінністю тут є public – ключове слово в класі (всі класи за замовчуванням типу private).

Так само, як і оголошення структури, оголошення класу не призводить до виділення будь-якої пам'яті. Для використання класу потрібно оголосити змінну цього типу класу:

DateClass today { 26, 12, 2022 }; // ініціалізуємо змінну класу DateClass

Точно так же, як визначення змінної фундаментального типу даних (наприклад, int x) призводить до виділення пам'яті для цієї змінної, так само і створення об'єкта класу (наприклад, DateClass today) призводить до виділення пам'яті для цього об'єкта. Об'єкти класу є аналогами змінних, свого роду новий тип змінної, створеної користувачем.

Методи класів

Функції, визначені всередині класу, називаються **методами.** Методи можуть бути визначені, як всередині, так і поза класом. Поки що ми будемо визначати їх всередині класу (для простоти), як визначити їх поза класом – розглянемо трохи пізніше.

Клас Date з методом виводу дати:

```
class DateClass
{
public:
    int m_day;
    int m_month;
    int m_year;

    void print() // визначаємо метод
    {
        cout << m_day << "/" << m_month << "/" << m_year;
    }
};
```

До членів (змінних і функцій) класу доступ здійснюється через оператор вибору членів (**.**):

Приклад створення класу

```
#include <iostream>
#include<cstdlib>
using namespace std;
class DateClass
```

```
{
public:
   int m_day;
   int m_month;
   int m_year;

   void print()
   {
      cout << m_day << "/" << m_month << "/" << m_year;
   }
};

int main()
{
   DateClass today { 26, 12, 2022 };
    today.m_day = 26; // використовуємо оператор вибору членів для вибору змінної-члена m_day об’єкта today класу DateClass
   today.print(); // використовуємо оператор вибору членів для виклику метода print() об'єкта today класу DateClass

   return 0;
}
```

Результат роботи:

```
26/12/2022
```

У цьому прикладі є **відкриті члени** класу (**public**) – це члени класу, до яких можна отримати доступ ззовні цієї ж структури або класу. У вищенаведеній програмі функція main() знаходиться поза структурою, але вона може безпосередньо звертатися до членів day, month і year, так як вони є відкритими.

Приклад створення класу. Неробоча програма

```
#include <iostream>
#include<cstdlib>
using namespace std;
class DateClass // члени класу є закритими за замовчуванням
```

```
    {
        int m_day; // закрито за замовчуванням, доступ мають тільки інші члени класу
        int m_month; // закрито за замовчуванням, доступ мають тільки інші члени класу
        int m_year; // закрито за замовчуванням, доступ мають тільки інші члени класу
        void print()
        {
            cout << m_day << "/" << m_month << "/" << m_year;
        }
    };

    int main()
    {
        DateClass date;
        date.m_day = 26; // помилка
        date.m_month = 12; // помилка
        date.m_year = 2022; // помилка
     date.print();
        return 0;
    }
```

Результат роботи:
помилка

Вам би не вдалося скомпілювати цю програму, тому що всі члени класу є закритими за замовчуванням. Закриті члени (private) – це члени класу, доступ до яких мають лише інші члени цього ж класу. Оскільки функція main() не є членом DateClass, то вона і не має доступу до закритих членів об'єкта date.

Хоча члени класу є закритими за замовчуванням, ми можемо зробити їх відкритими, використовуючи **ключове слово public**:

Приклад створення класу. Робоча програма

```
#include <iostream>
#include<cstdlib>
using namespace std;
```

```
class DateClass
{
public: // зверніть увагу на ключове слово public і двокрапку
    int m_day; // відкрито, доступ має будь-який об’єкт
    int m_month; // відкрито, доступ має будь-який об’єкт
    int m_year; // відкрито, доступ має будь-який об’єкт
    void print()
    {
        cout << m_day << "/" << m_month << "/" << m_year;
    }
};

int main()
{
    DateClass date;
    date.m_day = 26; // ок, так як m_day має специфікатор доступу public
    date.m_month = 12; // ок, так як m_month має специфікатор доступу public
    date.m_year = 2022; // ок, так як m_year має специфікатор доступу public
    date.print();
    return 0;
}
```

Результат роботи:

```
26/12/2022a
```

Оскільки тепер члени класу DateClass є відкритими, то до них можна отримати доступ безпосередньо з функції main().

Специфікатори доступу

Ключове слово public разом із двокрапкою називається специфікатором доступу. **Специфікатор доступу** визначає, хто має доступ до членів цього специфікатора. Кожен із членів «набуває» рівень доступу відповідно до специфікатора доступу (або, якщо він не вказаний, відповідно до специфікатора доступу за замовчуванням).

У мові C++ є 3 рівні доступу:

специфікатор public робить члени відкритими;

специфікатор private робить члени закритими;
специфікатор protected відкриває доступ до членів тільки для дружніх і дочірніх класів.

Використання специфікаторів доступу

Класи можуть використовувати (і активно використовують) відразу кілька специфікаторів доступу для встановлення рівнів доступу для кожного зі своїх членів. Зазвичай змінні-члени є закритими, а методи – відкритими.

Функції доступу: get (геттери) і set (сеттери)

Залежно від класу, може бути доречно (в контексті того, що робить клас) мати можливість отримувати/встановлювати значення закритих змінних-членів класу.

Функція доступу – це коротка відкрита функція, завданням якої є отримання або зміна значення закритої змінної-члена класу.

Наприклад:

```
class MyString
{
private:
    char *m_string; // динамічно виділяємо рядок
    int m_length; // використовуємо змінну для відстеження довжини рядка

public:
    int getLength() { return m_length; } // функція доступу для отримання значення m_length
};
```

Тут getLength() є функцією доступу, яка просто повертає значення m_length.

Функції доступу зазвичай бувають двох типів:

геттери – це функції, які повертають значення закритих змінних-членів класу;

сеттери – це функції, які дозволяють присвоювати значення закритим змінним-членам класу.

Приклад класу, який використовує геттери і сеттери для всіх своїх закритих членів:

```
class Date
```

```
{
private:
   int m_day;
   int m_month;
   int m_year;

public:
   int getDay() { return m_day; } // геттер для day
   void setDay(int day) { m_day = day; } // сеттер для day

   int getMonth() { return m_month; } // геттер для month
   void setMonth(int month) { m_month = month; } // сеттер для month
    int getYear() { return m_year; } // геттер для year
   void setYear(int year) { m_year = year; } // сеттер для year
};
```

У цьому класі немає ніяких проблем з тим, щоб користувач міг напряму отримувати або присвоювати значення закритим змінним – членам цього класу, так як є повний набір геттерів і сеттерів. У прикладі з класом MyString для змінної m_length не було надано сеттера, так як не було необхідності в тому, щоб користувач міг напряму встановлювати довжину.

Приклад створення класу. Відкриті і закриті члени класу

```
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;
class MyMoney {
 private:
 string name;
double money;
double rate;
int time;
   double getMoney () {
      double S = money;
      for(int k = 1; k <= time; k++){
         S *= (1 + rate/100);
      }
```

```
        return S;
        }
    public:
    void ShowAll() {
        cout << "Name" << name << endl;
        cout << "Money " << money << endl;
        cout << "int. rate % " << rate << endl;
        cout << "Period (years) " << time << endl;
        cout << "Final sum" << getMoney() << endl;
    }
    void SetAll (string n, double m, double r, int t) {
        name = n;
        money = m;
        rate = r;
        time = t;
    }
    };
    int main () {
   MyMoney objA, objB;
   objA.SetAll ( "John Snow", 1200, 8, 5);
        objB.SetAll  ( "Harry Potter", 1200, 6, 9);
        objA.ShowAll ();
        cout << endl;
        objB.ShowAll ();
        cout << endl;
        return 0; }
```

Результат роботи:

```
NameJohn Snow
Money 1200
int. rate % 8
Period (years) 5
Final sum1763.19

NameHarry Potter
Money 1200
int. rate % 6
Period (years) 9
Final sum2027.37
```

Конструктори та деструктори

Конструктор – це особливий тип методу класу, який автоматично викликається при створенні об'єкта цього ж класу. Конструктори зазвичай використовуються для ініціалізації змінних-членів класу значеннями, які надані за замовчуванням/користувачем, або для виконання будь-яких кроків налаштування, необхідних для використовуваного класу (наприклад, відкрити певний файл або базу даних).

На відміну від звичайних методів, конструктори мають певні правила щодо своїх імен:

конструктори завжди повинні мати те ж ім'я, що і клас (враховуючи верхній і нижній регістри);

конструктори не мають типу повернення (навіть void).

Конструктори призначені тільки для виконання ініціалізації. Не слід намагатися викликати конструктор для повторної ініціалізації існуючого об'єкта.

Конструктори за замовчуванням

Конструктор, який не має параметрів (або містить тільки параметри, із значенням за замовчуванням), називається **конструктором за замовчуванням.** Він викликається, якщо користувачем не вказані значення для ініціалізації.

Конструктори з параметрами

Хоча конструктор за замовчуванням відмінно підходить для забезпечення ініціалізації класів значеннями за замовчуванням, часто може бути потрібно, щоб екземпляри класу мали певні значення, які ми надамо пізніше. Конструктори також можуть бути оголошені з параметрами. Зверніть увагу, тепер у нас є два конструктори: конструктор за замовчуванням, який викликатиметься, якщо ми не надамо значення, і конструктор із параметрами, який викликатиметься, якщо ми надамо значення. Ці два конструктора можуть мирно співіснувати в одному класі завдяки перевантаженню функцій. Можна визначити будь-яку кількість конструкторів до тих пір, поки у них будуть унікальні параметри (враховується їх кількість і тип).

Якщо клас не має конструкторів, то мова C++ автоматично згенерує для класу відкритий конструктор за замовчуванням. Його іноді називають **неявним конструктором.**

Делегуючі конструктори

Починаючи з C++11, конструкторам дозволено викликати інші конструктори. Цей процес називається **делегуванням конструкторів** (або **«ланцюжком конструкторів»**). Щоб один конструктор викликав інший, потрібно просто зробити виклик цього конструктора у списку ініціалізації. Наприклад:

```
class Boo
{
private:

public:
   Boo()
   {
      // Частина коду X
   }
    Boo(int value): Boo() // використовуємо конструктор за замовчуванням Boo() для виконання частини коду X
   {
      // Частина коду Y
   }
};
```

Деструктори

Деструктор – це спеціальний тип методу класу, який виконується під час видалення об'єкта класу. Конструктори призначені для ініціалізації класу, деструктори призначені для очищення пам'яті після нього.

Коли об'єкт автоматично виходить з області видимості або динамічно виділений об'єкт явно видаляється за допомогою ключового слова delete, викликається деструктор класу (якщо він існує) для виконання необхідного очищення до того, як об'єкт буде видалений із пам'яті. Для простих класів (тих, які тільки ініціалізують значення звичайних змінних-членів) деструктор не потрібен, так як C++ автоматично виконає очищення самостійно.

Якщо об'єкт класу містить будь-які ресурси (наприклад, динамічно виділену пам'ять або файл/базу даних), або, якщо вам необхідно виконати будь-які дії до того, як об'єкт буде знищений, деструктор є ідеальним рішенням, оскільки він виконує останні дії з об'єктом перед його остаточним знищенням.

Імена деструкторів

Так само, як і конструктори, деструктори мають свої правила, які стосуються їхніх імен:

деструктор повинен мати те ж ім'я, що і клас, зі знаком тильда (~) на самому початку;

деструктор не може приймати аргументи;

деструктор не має типу повернення.

З другого правила випливає ще одне правило: для кожного класу може існувати тільки один деструктор, так як немає можливості перевантажити деструктори, як, наприклад, функції.

Приклад використання деструктора

```
#include <iostream>
#include <cassert>
 using namespace std;
class Massiv
{
private:
	int *m_array;
	int m_length;

public:
	Massiv(int length) // конструктор
	{
		assert(length > 0);

		m_array = new int[length];
		m_length = length;
	}

	~Massiv() // деструктор
	{
		// Динамічно видаляємо масив, який виділили раніше
		delete[] m_array ;
	}
	void setValue(int index, int value) { m_array[index] = value; }
	int getValue(int index) { return m_array[index]; }
```

```
        int getLength() { return m_length; }
    };

    int main()
    {
        Massiv arr(15); // виділяємо 15 цілочисельних значень
        for (int count=0; count < 15; ++count)
            arr.setValue(count, count+1);

        cout << "The value of element 7 is " << arr.getValue(7);

        return 0;
    } // об'єкт arr видаляється тут, тому деструктор ~Massiv() викликається також тут
```

Результат роботи:

```
The value of element 7 is 8
```

Приклад використання конструкторів і деструкторів. З використанням динамічного розподілу пам'яті new і delete

```
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;
class MyMoney {
 private:
 string name;
double money;
double rate;
int time;
   double getMoney () {
      double S = money;
      for(int k = 1; k <= time; k++){
         S *= (1 + rate/100);
      }
      return S;
      }
   public:
   // конструктор без аргументів
```

```
    MyMoney () {
      name = "John Snow";
       money = 100;
       rate = 5;
       time = 1;
       cout<< "Created new object: \n";
       ShowAll ();
    }
   // конструктор з 4 аргументами
    MyMoney (string n, double m, double r, int t) {
       SetAll (n, m, r, t);
       cout<< "Created new object: \n";
       ShowAll ();
       }
// деструктор
~MyMoney () {
   cout << " Object for name " << name << " destroyed \n";
   for (int k=1; k<=32; k++)
   { cout << "*";}
  cout << endl;
}
   // методи класу
    void ShowAll() {
       cout << "Name" << name << endl;
       cout << "Money " << money << endl;
       cout << "int. rate % " << rate << endl;
       cout << "Period (years) " << time << endl;
       cout << "Final sum" << getMoney() << endl;
   for (int k=1; k<=32; k++)
   { cout << "-";}
  cout << endl;
}
    void SetAll (string n, double m, double r, int t) {
       name = n;
       money = m;
       rate = r;
       time = t;
    }
    };
```

```
    //функція
    void foxfellow (){
       MyMoney objD("Fox", 200, 3, 2);
    }

    int main () {
   MyMoney objA;
    MyMoney objB ( "Boris Johnson", 1200, 8, 5);
   foxfellow();
   // створення динамічного обєкту
   MyMoney* objC = new MyMoney ( " Harry Potter", 1200, 6, 9);
   cout << " All objects are created \n";
   delete objC;
   cout << "program has finished \n";
   cout<< endl;

       return 0; }
```

Результат роботи:

```
Created new object:
NameJohn Snow
Money 100
int. rate % 5
Period (years) 1
Final sum105
--------------------------------
Created new object:
NameBoris Johnson
Money 1200
int. rate % 8
Period (years) 5
Final sum1763.19
--------------------------------
Created new object:
NameFox
Money 200
int. rate % 3
Period (years) 2
Final sum212.18
--------------------------------
```

```
Object for name Fox destroyed
*******************************
Created new object:
Name Harry Potter
Money 1200
int. rate % 6
Period (years) 9
Final sum2027.37
------------------------------
All objects are created
Object for name  Harry Potter destroyed
*******************************
program has finished

Object for name Boris Johnson destroyed
*******************************
Object for name John Snow destroyed
*******************************
```

Тема 6. Класи II

Ключові слова default, delete

Ключове слово default введене в C++ 11. Його використання вказує компілятору самостійно генерувати (використовувати) відповідну функцію класу, якщо така не оголошена у класі.

Як відомо, компілятор автоматично генерує ряд конструкторів класу та деструктора. Указання ключового слова default для цих функцій дає такі взаємозв'язані переваги:

− програміст отримує зрозумілу інформацію про відсутність реалізації окремих функцій у класі. Використовуються компіляторні версії цих функцій;

− полегшується сприйняття інформації про особливості використання тих чи інших функцій класу. Замість того, щоб згадувати правила, програміст указує те, що хоче отримати.

Загальний синтаксис використання ключового слова default для конструкторів класу

```
class ClassName
{
   // ...
```

```
    ClassName(parameters) = default;

    // ...
};
```

де:

ClassName – ім'я класу, в якому використано ключове слово default;

ClassName(parameters) – один із конструкторів класу, які автоматично генеруються компілятором;

parameters – список параметрів, що приймає спеціальна функція класу, що генерується автоматично. Список параметрів може бути відсутній (конструктор за замовчуванням).

Загальний синтаксис використання ключового слова default для деструктора класу

```
class ClassName
{
    // ...

    ~ClassName() = default;

    // ...
};
```

Ключове слово default може застосовуватись лише до спеціальних функцій класу, які при оголошенні класу генеруються компілятором автоматично. До таких функцій належать:

- конструктор за замовчуванням (default constructor);
- конструктор копіювання (copy constructor);
- конструктор переміщення (move constructor);
- оператор присвоєння копіюванням (copy assignment operator);
- оператор присвоєння перенесенням (move assignment operator);
- деструктор (destructor).

Приклад використання ключового слова default для спеціальних функцій класу, які генеруються компілятором автоматично
Оголошується клас Float, що реалізує дійсне число.

У класі оголошуються:
внутрішня (private) змінна x;
default-конструктор за замовчуванням;
default-конструктор копіювання;
default-конструктор переміщення;
default-оператор присвоєння, що виконує копіювання;
default-оператор присвоєння, що виконує переміщення;
default-деструктор;
методи доступу Get (), Set ().

```
#include <iostream>
#include <cstdlib>
using namespace std;

class Float
{
private:
  float x;

public:
  // Конструктор за замовчуванням
  // ключове слово default вказує компілятору самостійно формувати
  // відповідний конструктор, якщо такий не оголошений в класі
  Float() = default;

  // конструктор з 1 параметром - не може бути застосоване default
  Float(float _x) :x(_x)
  { }

  // конструктор копіювання - передача прав компілятору на формування цього конструктора
  Float(const Float&) = default;

  // конструктор переміщення
  Float(Float&&) = default;

  // оператор присвоєння копіюванням
```

```
  Float& operator=(const Float&) = default;

  // оператор присвоєння переміщенням
  Float& operator=(Float&&) = default;

  // деструктор
  ~Float() = default;

  // методи доступу
  float Get() { return x; }
  void Set(float _x) { x = _x; }
};

int main()
{
  // default - конструктор
  Float f1(7);
  Float f2 = f1;

  cout << "f2.x = " << f2.Get() << endl;

  Float f3;
  f3.Set(18);
  f3 = f2;
  cout << "f3.x = " << f3.Get() << endl;
}
```

Результат роботи:

```
f2.x = 7
f3.x = 7
```

Ключове слове delete

Ключове слово delete використовується у випадках, якщо потрібно заборонити автоматичне приведення типів у конструкторах і методах класу.

Загальний синтаксис оголошення конструктора класу з ключовим словом delete

```
class ClassName
{
  // ...
```

```
  // конструктори, які заборонено використовувати
  ClassName(parameters1) = delete;
  ClassName(parameters2) = delete;

  // ...

  ClassName(parametersN) = delete;

  // ...
};
```

де

parametes1, parameters2, …, parametersN – список параметрів відповідного конструктора класу.

Параметри можуть бути відсутні. Якщо параметри конструктора відсутні, то заборонено використовувати конструктор за замовчуванням. У цьому випадку на оголошення

```
ClassName obj1;
```

компілятор видаватиме помилку:

```
The default constructor of "ClassName" cannot be referenced – it is a deleted function
```

Ключове слово delete може бути заборонене також у методах. У цьому випадку, для методу під назвою Func() загальний синтаксис оголошення буде таким

```
class ClassName
{
  // ...

  return_type Func(parameters) = delete;

  // ...
};
```

де:

Func() – ім’я функції, яку потрібно заборонити використовувати в класі;

parameters – параметри функції;

return_type – тип, який повертається функцією.

Приклад використання ключового слова delete у класі

Задано клас Point, який реалізує точку на площині. У класі оголошуються такі елементи:

− внутрішні сховані (private) змінні з іменами x, y – координати точки;

− конструктор за замовчуванням Point(), який позначений ключовим словом default. Ключове слово default указує, що повинен виконуватися неявний конструктор за замовчуванням, що генерується компілятором;

− конструктор Point(int) з одним параметром типу int;

− конструктор із двома параметрами Point(int, int);

− методи доступу GetX(), GetY(), SetX(), SetY();

− операторна функція operator+(), яка перевантажує бінарний оператор +;

− операторна функція operator-(), яка перевантажує унарний оператор –.

У класі забороняється використовувати конструктор з одним параметром Point(double), який приймає параметр типу double. Цей конструктор позначено ключовим словом delete Однак, конструктор із параметром типу int можна використовувати в класі.

Також забороняється використовувати деякі функції:

функцію SetX(double) із параметром типу double. Однак, SetX(int) з параметром типу int можна використовувати;

функцію SetXY(int, int) із двома параметрами типу int.

Приклад використання ключового слова delete у класі. Приклади перевантаження унарного і бінарного операторів

```
#include <iostream>
using namespace std;

// Клас, що реалізує точку на координатній площині
// У класі досліджуються ключові слова default та delete
class Point
{
private:
  int x = 0; // координати точки
```

```
  int y = 0;

public:
  // default-конструктор, довірити реалізацію компілятору
  Point() = default;

  // конструктор з одним параметром типу double – заборонено використовувати
  Point(double value) = delete;

  // конструктор з одним параметром типу int – дозволено використовувати
  Point(int value)
  {
    x = y = value;
  }

  // конструктор з 2 параметрами
  Point(int nx, int ny)
  {
    x = nx;
    y = ny;
  }

  // методи доступу до членів класу
  int GetX(void) { return x; }
  int GetY(void) { return y; }

  // функції SetX(), SetY() з параметром типу int
  void SetX(int nx) { x = nx; }
  void SetY(int ny) { y = ny; }

  // заборонити виклик функції SetX(), SetY() з параметром типу double
  void SetX(double) = delete;

  // інформація для компілятора, що заборонено використовувати
  // метод SetXY() з двома параметрами int
  void SetXY(int, int) = delete;
```

```
  // сам метод SetXY – непотрібен, бо компілятор видає помилку
  /*
  void SetXY(int nx, int ny)
  {
    x = nx;
    y = ny;
  }
  */

  // перевантажений бінарний оператор '+'
  Point operator+(Point& pt)
  {
    // p - тимчасовий об’єкт, який створюється з допомогою конструктора без параметрів
    Point p;
    p.x = x + pt.x;
    p.y = y + pt.y;
    return p;
  }

  // перевантажений унарний оператор '-'
  Point operator-(void)
  {
    Point p;
    p.x = -x;
    p.y = -y;
    return p;
  }

  void Show(const char* objName)
  {
    cout << objName << ":" << endl;
    cout << "x=" << x << "; y=" << y << endl;
    cout << endl;
  }
};

int main()
{
  // Оголошення об’єкту з використанням конструктора з 1 параметром
```

```
  // Помилка компілятора
  //Point p1(5.5); //конструктор з 1 параметром типу double заборонений

  // Дозволено використовувати конструктор з 1 параметром типу int
  Point p1(5);
  p1.Show("p1");

  // Заборонено використовувати метод SetX() з параметром типу double
  // p1.SetX(5.5);
  p1.SetY(3.8); // а метод SetY() можна використовувати
  p1.Show("p1");

  // Оголошення об’єкту без параметрів – викликається default-конструктор
  Point p3; // викликається default-конструктор компілятора
  p3.Show("p3");

  // Оголошення об’єкту з допомогою конструктораз 2 параметрами
  Point p4(7, 8); // дозволено, працює
  p4.Show("p4");
}
```

Результат роботи:

```
p1:
x=5; y=5

p1:
x=5; y=3

p3:
x=0; y=0

p4:
x=7; y=8
```

Статичні елементи класу

Статичні члени (елементи) класу

Щоб оголосити покажчик на статичний член даних класу потрібно просто описати покажчик на тип даних, який має

статичний член даних. Потім, у деякому програмному коді, цьому покажчику потрібно присвоїти адресу статичного члена даних об’єкта цього класу.

Послідовність кроків така:

- оголосити клас зі статичним членом деякого типу;
- описати статичний член даних класу за межами класу та ініціалізувати його за необхідності;
- у деякому методі або за межами усіх методів (глобальний покажчик) описати покажчик на тип, що має статичний член даних класу;
- описати об’єкт класу зі статичним членом класу;
- присвоїти покажчику адресу статичного члена даних об’єкта класу.

Приклад оголошення класу CPi зі статичним членом даних Pi. Також оголошується декілька статичних членів, які не є членами класу. У функції main() демонструється використання покажчиків на оголошені статичні члени даних.

```
#include "stdafx.h"
#include <iostream>
using namespace std;

// оголошення класу, що містить статичний член даних
class CPi
{
   public:
   static double Pi; // статичний член даних – константа
};

// оголошення статичної змінної за межами класу з одночасною ініціалізацією
static double Exp = 2.7182818284; // експонента

// ініціалізація статичного члена класу CPi
double CPi::Pi = 3.141592653589793;

// неініціалізовані статичні змінні, значення яких = 0 (за замовчуванням)
static double ZeroD;
```

```
    static int ZeroI;
    static char ZeroC;

    int * pZ; // зовнішній покажчик на int

    int _tmain(int argc, _TCHAR* argv[])
    {
       double d;
       CPi obj; // об'єкт класу
       double * pPi; // покажчик на double
       double * pExp; // покажчик на double

       // покажчик вказує на статичний член даних об'єкту obj
класу CPi
       pPi = &obj.Pi;
       pExp = &::Exp; // вказує на статичний член даних Exp -
інший вид доступу через ::

       // перевірка
       d = obj.Pi; // d = 3.14159
       d = Exp; // d = 2.71828
       d = ::Exp; // d = 2.71828 - так теж можна
       d = *pPi; // d = 3.14159

       // зовнішній покажчик на int
       pZ = &ZeroI;
       d = *pZ; // d = 0

       *pZ = 5;
       d = ZeroI; // d = 5

       return 0;
    }
```

Приклад використання масиву, елементами якого є статичні члени даних класу

Задано 3 класи з іменами CCounter1, CCounter2, CCounter3. У кожному класі реалізовано по одному статичному члену даних відповідно з іменами counter1, counter2, counter3.

Оскільки, в класах є статичні члени даних, то їх відповідним чином можна ініціалізувати. Також у прикладі оголошується глобальний масив з іменем arCount1[], який ініціалізований статичними членами даних класів CCounter1, CCounter2, CCounter3.

Лістинг оголошення класів та ініціалізації статичних членів такий:

```
// оголошується 3 класи зі статичними змінними
// клас, що містить статичну змінну count
class CCount1
{
   public:
   static int count1; // статичний член даних, ключове слово static
};
class CCount2
{
   public:
   static int count2; // статичний член даних, ключове слово static
};
class CCount3
{
   public:
   static int count3; // статичний член даних, ключове слово static
};
// оголошення ініціалізація статичних членів даних класів
// тут ключового слова static не потрібно
int CCount1::count1; // за замовчуванням значення статичного члена = 0
int CCount2::count2 = 5;
int CCount3::count3 = 100;
// оголошення масиву, що ініціалізований статичними членами даних
int arCount1[] =
{
   CCount1::count1,
   CCount2::count2,
   CCount3::count3
};
```

Якщо статичний член не ініціалізований, то встановлюється значення за замовчуванням:

− для числових типів (int, float, double …) значення неініціалізованого статичного члена рівне 0

− для типу char значення неініціалізованого статичного члена рівне символу з кодом 0.

− для типу bool значення рівне false, що також відповідає значенню 0.

Види функцій-елементів (членів) класу. Звичайні функції-елементи. Статичні (static) функції-члени

У класах можна оголошувати такі види функцій:

− «звичайні» функції-члени. Це функції, в оголошенні яких не використовуються додаткові ключові слова (const, volatile, static);

− статичні функції. Це функції, які оголошені з ключовим словом static;

− константні функції. Це функції, які оголошені з ключовим словом const;

− функції-члени, які повертають непостійний об’єкт. Це функції, які оголошені з ключовим словом volatile;

− функції-члени з константним вказівником this;

− функції-члени з непостійним вказівником this;

− вбудовані функції – члени (inline).

У цьому прикладі оголошено клас CLine, що описує лінію на координатній площині. Лінія складається з координат точок (x1; y1), (x2; y2). Клас містить прості функції-члени, які оголошені в розділі public (загальнодоступні). За допомогою цих функцій-членів здійснюється читання/модифікація внутрішніх змінних класу.

Приклад оголошення класу CLine, що описує лінію на координатній площині

```
// клас, що описує лінію на координатній площині
class CLine
{
   int x1, y1;
   int x2, y2;

   public:
```

```
    CLine(void);
    ~CLine(void);

    // методи класу
    void SetLine(int nx1, int ny1, int nx2, int ny2);
    int GetX1(void) { return x1; }
    int GetY1(void) { return y1; }
    int GetX2(void) { return x2; }
    int GetY2(void) { return y2; }
};

// Реалізація методів класу
// конструктор за замовчуванням
CLine::CLine(void)
{
    x1 = y1 = 0;
    x2 = y2 = 1;
}

// деструктор
CLine::~CLine(void)
{
}

// реалізація простої функції-члена класу
void CLine::SetLine(int nx1, int ny1, int nx2, int ny2)
{
    x1 = nx1;
    y1 = ny1;
    x2 = nx2;
    y2 = ny2;
}
Використання класу в іншому програмному коді
// оголошення об’єкту класу CLine
CLine CL;
int t;

// виклик функцій-членів класу
CL.SetLine(5, 6, 10, 20);
t = CL.GetX2(); // t = 10
t = CL.GetY1(); // t = 6
```

Приклад оголошення класу, що містить статичні (static) функції-члени

У цьому прикладі оголошено клас CCircle, що описує коло на координатній площині. Клас реалізує такі внутрішні члени даних і методи:

− внутрішні змінні (члени) класу x, y, r (координати центру та радіус кола);

− конструктор класу CCircle(), який визначає одиничне коло (r=1) з центром на початку координат;

− статичну функцію-член класу SetData1(), яка встановлює значення глобальних змінних xx, yy, rr оголошених за межами класу;

− звичайну (нестатичну) функцію-член класу SetData2(), яка змінює значення внутрішніх змінних x, y, r класу;

− звичайні (нестатичні) методи класу, що читають значення внутрішніх змінних класу x, y, r та глобальних змінних xx, yy, rr.

Приклад оголошення класу CCircle, що описує коло на координатній площині

```
// клас, що описує коло на координатній площині
class CCircle
{
   // внутрішні змінні класу
   int x, y; // координати центру кола
   int r; // радіус кола

   public:
   // конструктор класу
   CCircle(void);

   // статичний метод класу – тільки для глобальних змінних
   static void SetData1(int nx, int ny, int nr);

   // звичайні (нестатичні) методи класу
   // встановити нові значення внутрішніх змінних класу
   void SetData2(int nx, int ny, int nr);

   // зчитати значення внутрішніх змінних класу
   int GetX(void) { return x; }
   int GetY(void) { return y; }
   int GetR(void) { return r; }
```

```
    // зчитати значення глобальних змінних класу
    int GetXX(void);
    int GetYY(void);
    int GetRR(void);
};

// глобальні змінні
int xx, yy, rr; // ці змінні може змінювати статична функція-член класу

// конструктор класу
CCircle::CCircle(void)
{
    // заповнення значень внутрішніх (локальних) змінних
    x = 0; y = 0; r = 1;
}

// статична функція-член класу
void CCircle::SetData1(int nx, int ny, int nr)
{
    // присвоювання значень глобальним змінним,
    // що оголошені поза межами класу
    xx = nx;
    yy = ny;
    rr = nr;

    // працювати з внутрішніми змінними класу у статичній функції заборонено
    // x = nx; // Заборонено! Помилка: "... illegal reference to non-static member 'CCircle::x'
    // y = ny; // також заборонено
}

// встановити нові значення звичайних (нестатичних) методів класу
void CCircle::SetData2(int nx, int ny, int nr)
{
    x = nx;
    y = ny;
    r = nr;
}
```

```
// зчитати значення глобальних змінних класу
int CCircle::GetXX(void)
{
   return xx;
}

int CCircle::GetYY(void)
{
   return yy;
}

int CCircle::GetRR(void)
{
   return rr;
}
```

Змінювати значення внутрішніх змінних класу зі статичної функції-члена класу заборонено. Якщо у функції SetData1() спробувати змінити значення внутрішніх змінних класу x, y

```
void CCircle::SetData1(int nx, int ny, int nr)
{
   ...
   x = nx;
   y = ny;
   ...
}
```

то компілятор видасть помилку

... illegal reference to non-static member 'CCircle::x'

Якщо потрібно змінити значення внутрішніх змінних класу зі статичної функції-члена, тоді цій функції потрібно передати покажчик **this** класу.

Приклад використання статичної функції-члена для класу Circle

```
// використання класу CCircle
CCircle C1;
int tx, ty, tr; // додаткові змінні

// виклик статичної функції-члена
C1.SetData1(2, 3, 4); // змінюються тільки глобальні змінні xx, yy, rr
```

```
    // прочитати значення внутрішніх змінних класу
    tx = C1.GetX(); // tx = 0 - внутрішні змінні x,y,r ініціалізовані конструктором класу
    tr = C1.GetR(); // tr = 1

    // прочитати значення глобальних змінних, що оголошені за межами класу
    tx = C1.GetXX(); // tx = 2
    ty = C1.GetYY(); // ty = 3

    // виклик звичайної (нестатичної) функції-члена класу
    C1.SetData2(2, 3, 4); // змінюються тільки внутрішні змінні x, y, r
    tx = C1.GetX(); // tx = 2
    tr = C1.GetR(); // tr = 4

    // інший спосіб виклику статичної функції-члена
    CCircle::SetData1(7, 8, 9);
    tx = C1.GetXX(); // tx = 7
    ty = C1.GetYY(); // ty = 8
```

Основні відмінності статичних функцій-членів від звичайних:

− викликати статичну функцію-член можна без посилання на об'єкт класу, наприклад CMyClass::MyStaticFuntion(…). У цьому випадку використовується оператор розширення області видимості **'::'**;

− статичні функції-члени не отримують покажчика this класу. Це означає, що вони не можуть отримати доступ до членів – даних класу (за винятком членів-даних класу, які оголошені як статичні – з ключовим словом static);

− у C++ статична функція-член класу є глобальною. Тому її доцільно використовувати для зміни значень глобальних змінних, які оголошені за межами класу, але мають застосування в класі.

Статичні функції-члени класу доцільно застосовувати у випадках, коли в класі потрібно використовувати глобальні змінні, що є спільними для різних об'єктів (екземплярів) цього класу.

Костантні об’єкти та функції-елементи класу. Константні функції

Функції, які повертають константний об’єкт називають константними функціями. Якщо такі функції оголошені в класі, то ці функції називають константними функціями-членами класу.

Щоб оголосити функцію, яка повертає константний об’єкт, потрібно перед оголошенням функції розмістити ключове слово ‘const’

```
const returned_type FunName(parameters)
{
   // function's body
   // ...
}
```

де

returned_type – тип, що повертається функцією;

parameters – параметри функції;

FunName – ім’я функції, що повертає константний об’єкт (константної функції).

Якщо функція оголошена в класі, то загальний синтаксис буде таким:

```
class CMyClass
{
   private:
   // приватні члени та методи класу
   // ...
   public:
   // загальнодоступні члени та методи класу
   // ...
   // оголошення константної функції в класі
   const returned_type FunName(parameters);
}
// реалізація функції FunName
const returned_type CMyClass::FunName(parameters)
{
   // тіло функції FunName()
   // ...
}
```

Приклад оголошення класу, що містить константні (const) функції-члени

Оголошується клас CMyClass, що містить константну функцію GetX(), яка повертає константне значення типу double.

```
// клас, що містить оголошення константної функції
class CMyClass
{
   double x; // внутрішня змінна

   public:
   CMyClass(void); // конструктор

   void SetX(double nx);
   const double GetX(void); // константна функція-член класу
};
```

Використання константної функції в деякому програмному коді:

```
// використання константної функції
CMyClass MC;

MC.SetX(25.08); // встановлення значення x

// використання константної функції GetX() класу CMyClass
double t1 = MC.GetX(); // t1 = 25.08 – ініціалізація змінної
const double t2 = MC.GetX(); // t2 = 25.08 – ініціалізація константи
```

Об’єкт (змінна), що оголошений зі специфікатором volatile може бути змінений у програмі неявно, без застосування явно заданих команд. Ключове слово volatile інформує компілятор, що значення змінної у програмі може бути змінене неявно (програмою обробки переривань, фоновим процесом, тощо). Знаючи про неявну мінливість змінної у програмі (ключове слово volatile), компілятор сформує код, що буде опитувати значення змінної перед кожним її використанням у програмі. Таким чином, буде сформовано реальне значення змінної (а не те, що було до зміни).

Наприклад. Нехай у деякій програмі P реалізована глобальна змінна X, що містить адресу, яка може бути змінена програ-

мою-обробником переривання. Програма-обробник переривання викликається незалежно від виконання даної програми P і відповідним чином змінює значення цієї глобальної змінної X. Якщо у програмі P не вказати ключового слова volatile перед X, то, при зчитуванні значення X у програмі P може відображатись старе значення X. Якщо вказати слово volatile, то компілятор буде оновлювати значення змінної X при кожному звертанні до неї, і, спотворення результату не буде.

Функції-члени, які оголошені зі специфікатором volatile просто повертають непостійний (volatile) об'єкт. Такий об'єкт може бути присвоєний змінній, що оголошена із специфікатором volatile.

Загальний вигляд класу, що містить функцію, яка повертає volatile-значення

```
class CMyClass
{
   private:
   // приватні члени та методи класу
   // ...

   public:
   // загальнодоступні члени та методи класу
   // ...

   // оголошення константної функції в класі
   volatile returned_type FunName(parameters);
}

// реалізація функції FunName
volatile returned_type CMyClass::FunName(parameters)
{
   // тіло функції FunName()
   // ...
}
```

де

- returned_type – тип, що повертається функцією;
- parameters – параметри функції;
- FunName – ім'я функції, що повертає volatile-об'єкт.

Приклад оголошення класу, що містить функції-члени volatile

У класі CMyVolatileClass оголошено функцію Get(), що повертає volatile-значення типу int.

```
// клас, що містить volatile-функцію
class CMyVolatileClass
{
   int d;
   public:
   CMyVolatileClass(void);
   // функції-члени класу
   // звичайна функція-член класу
   void Set(int nd) { d = nd; }
   // функція-член класу, що повертає volatile-значення
   volatile int Get(void);
};
```

Демонстрація використання класу в деякому методі чи програмному коді:

```
// клас, що містить volatile-функцію
CMyVolatileClass VC;
VC.Set(33); // виклик звичайної функції
// виклик volatile-функції для звичайної змінної
int t1 = VC.Get(); // t1 = 33
volatile int t2;
// виклик volatile-функції для volatile-змінної
t2 = VC.Get(); // t2 = 33
```

Приклад оголошення класу, що містить вбудовані функції (inline)

У прикладі оголошується клас CMyPoint. Клас містить внутрішні члени даних а також inline-функцію GetY(). Дві інші функції класу є звичайними, тому що реалізація цих функцій винесена за межі класу.

```
// клас, що містить inline-функції
class CMyPoint
{
   int x, y;
```

```
    public:
    // конструктор класу
    CMyPoint(void);

    int GetX(void); // звичайна (не inline) функція класу
    int GetY(void) // inline-функція
    {
       return y; // реалізація у тілі класу
    }

    // звичайна (не inline) функція
    void SetXY(int nx, int ny);
};

// конструктор класу
CMyPoint::CMyPoint(void)
{
   x = y = 0;
}

// реалізація не inline-функції GetX()
int CMyPoint::GetX(void)
{
   return x;
}

// реалізація не inline функції SetXY()
void CMyPoint::SetXY(int nx, int ny)
{
   x = nx;
   y = ny;
}
```

Додавати модифікатор inline в оголошення функції необхідно у випадках, коли виконуються дві основні умови:

− виклик функції відбувається настільки часто, що негативно впливає на швидкість виконання програми або просто сповільнює виконання програми. Наприклад, функція може викликатись багаторазово в операторі циклу;

− обсяг коду функції є малим. Функції, що мають великі обсяги програмного коду, суттєво збільшують розмір самої

програми, що, іноді, теж небажано. Тому як вбудовані, рекомендовано використовувати тільки дуже малі функції.

Оголошення inline-функції є запитом, а не командою. Тому компілятор може не виконати запит на генерування коду для оголошеної inline-функції. У цьому випадку функція може бути використана як звичайна (не inline).

Наприклад:

у більшості випадків рекурсивні функції не можуть бути використані як inline-функції;

не можна згенерувати inline функцію, що містить статичні члени даних.

Причини, які можуть спричинити ігнорування ключового слова inline.

У деяких випадках компілятор не може визначити функцію як вбудовану та просто ігнорує ключове слово inline. Нижче перераховано можливі причини такої поведінки компілятора:

- дуже великий розмір функції;
- функція є рекурсивною;
- у тому самому виразі функція може повторюватися кілька разів;
- у функції містяться управляючі оператори switch, if або оператори циклу.

Дружні класи і дружні функції. Ключове слово friend

Бувають випадки, коли для заданого класу потрібно оголосити інший клас або функцію, які повинні мати необмежений доступ до внутрішніх змінних і методів класу. Така необхідність виникає із суті завдання, що розв'язується.

Якщо клас А оголошується «дружнім» до класу В, то об'єкти класу А мають доступ до всіх членів даних і методів класу В. Якщо функція оголошується «дружньою» до деякого класу, то ця функція також має необмежений доступ до членів даних і методів цього класу.

Щоб оголосити «дружній» клас до цього класу, використовується ключове слово friend.

Загальний синтаксис оголошення «дружнього» класу виглядає:

```
class CClass
{
    // ...

    friend class CFriendClass;

    // ...
};

class CFriendClass
{
    // ...
};
```

де CClass – клас, у якому оголошується «дружній» клас CFriendClass. Усі змінні (навіть і private) та методи цього класу є доступними для об'єктів класу CFriendClass;

CFriendClass – клас, який є «дружнім» до класу CClass. Оголошення «дружнього» класу CFriendClass до класу CClass може бути у будь-якому місці тіла класу – в межах оголошення класу (між фігурними дужками { }).

Оголошення «дружньої» функції до класу починається із ключового слова friend. Загальна форма оголошення «дружньої» функції до класу має вигляд:

```
friend type FunName(parameters);
```

де FunName – ім'я «дружньої» функції;
type – тип, що повертається функцією FunName();
parameters – параметри «дружньої» функції.

Щоб отримати об'єкт потрібного класу у функції FunName(), цецільно передати у цю функцію посилання (або указівник) на об'єкт цього класу.

Якщо потрібно оголосити «дружню» функцію в деякому класі, то загальний синтаксис такого оголошення такий:

```
class CClass
{
    // ...

    friend type FunName(parameters);

    // ...
};
```

де FunName – ім’я «дружньої» функції;
type – тип, що повертається функцією FunName();
parameters – параметри «дружньої» функції.
Оголошувати «дружній» клас або функцію до заданого класу можна у будь-якому місці чи розділі класу в межах його оголошення (між фігурними дужками **{ }**).

Щоб отримати об’єкт потрібного класу у функції, необхідно передати в цю функцію посилання (або вказівник) на об’єкт цього класу.
Наприклад.
Припустимо, є клас з ім’ям CMyClass. Потрібно оголосити «дружню» функцію до цього класу, яка має ім’я FriendFun(). Функція повертає параметр типу int.

Оголошення класу CMyClass та «дружньої» функції у класі має вигляд:

```
// оголошення класу CMyClass, у якому є «дружня» функція FriendFun()
class CMyClass
{
    // тіло класу
    // ...

    friend int FriendFun(CMyClass &);

    // ...
};
```

Реалізація “дружньої” функції FriendFun():

```
int FriendFun(CMyClass & mc)
{
    // використання об’єкту класу mc для доступу до членів класу CMyClass
    // ...
};
```

Нехай задано клас Number, що містить цілочисельну величину. Також задано клас RangeNum, який містить величину Number але в межах заданого діапазону.

Щоб з класу RangeNum можна було мати доступ до приватної змінної num класу Number, клас RangeNum оголошується «дружнім» до класу Number.

Приклад оголошення класу, що є дружнім до іншого класу

```
// клас, що реалізує ціле число
class Number
{
  // оголошення дружнього класу RangeNum до класу Number
    friend class RangeNum;
    int num;

    public:
    // конструктори
    Number() { num = 0; }
    Number(int num) { this->num = num; }
};

// оголошення класу RangeNum, який тримає число Number в заданих межах
class RangeNum
{
    Number num; // об'єкт класу Number – просте ціле число
    int min; // нижня межа числа num
    int max; // верхня межа числа num

    public:
    // конструктор класу
    RangeNum()
    {
      // доступ до private-члена класу Number, тому що RangeNum є дружнім до Number
        num.num = 0;

        // встановлення діапазону 0..99 за замовчуванням
        min = 0;
        max = 99;
    }
```

```
    // методи доступу
    int GetNum(void)
    {
      return num.num; // доступ до приватного члена з "дружнього" класу Range
    }

    void SetNum(int nnum)
    {
      num.num = nnum; // доступ до приватного члена з "дружнього" класу Range
      if (num.num>max) num.num = max-1;
      if (num.num<min) num.num = min;
    }

    // встановлення діапазону для num в заданих межах
    void SetRange(int min, int max)
    {
      this->min = min;
      this->max = max;
      if (num.num>max) num.num = max-1; // знову доступ через дружній клас
      if (num.num<min) num.num = min;
    }
  };
```

Якщо у класі Number в оголошенні «дружнього» класу RangeNum friend class RangeNum, забрати ключове слово friend, то у конструкторі та усіх методах класу при доступі до num.num компілятор видасть помилку:

Number::num: cannot access private member declared in class 'Number'

Приклад оголошення функції, що є дружньою до іншого класу

У прикладі оголошується клас Radius, що містить величину радіуса деякої геометричної фігури. У класі оголошуються:

- одна прихована (private) змінна radius;
- методи Get() та Set() для доступу до змінної radius;

– дві зовнішні «дружні» функції GetLength() та GetArea();
– один «дружній» клас Volume. У класі Volume оголошується зовнішній (public) метод GetVolume(), який повертає об'єм кулі заданого радіуса.

Приклад оголошення функції, що є дружньою до іншого класу

```
#include "cstdlib"
#include <iostream>

using namespace std;

// клас, що реалізує величину радіуса геометричної фігури
class Radius
{
   private:
   double radius; // прихована змінна radius

   // оголошення "дружніх" функцій – в будь-якому розділі класу Radius
   friend double GetLength(Radius &);
   friend double GetArea(Radius &);

   // оголошення "дружнього" класу
   friend class Volume;

   public:
   // методи доступу до radius
   double Get(void) { return radius; }
   void Set(double nradius) { radius = nradius; }
};

// "дружні" функції до класу Radius
// довжина кола
double GetLength(Radius & r)
{
   // доступ до private-члена radius класу з "дружньої" функції GetLength()
   return (double)(2 * r.radius * 3.1415);
}
```

```
// площа круга
double GetArea(Radius & r)
{
   // доступ до private-члена radius класу з "дружньої" функції
GetArea()
   return (double)(r.radius * r.radius * 3.1415);
}

// оголошення "дружнього" класу
class Volume
{
   public:
   // клас містить тільки одну функцію Volume
   double GetVolume(Radius * r) // функція отримує покажчик
на Radius
   {
      // доступ до private-члена класу Radius з "дружнього"
класу Volume
      return (double)(4.0 / 3.0 * 3.1415 * r->radius * r->radius
* r->radius);
   }
};

int main ()
{
   // об’єкт класу Radius
   Radius r;
   Volume v;
   double len, area, vol;

   r.Set(3);

   // виклик зовнішньої "дружньої" функції GetLength()
   len = GetLength(r); // передача об'єкту класу Radius за
посиланням

   // виклик зовнішньої "дружньої" функції GetArea()
   area = ::GetArea(r); // передача за посиланням
   // виклик функції "дружнього" класу v
   vol = v.GetVolume(&r); // передача за покажчиком
```

```
    cout << "Length = " << len << endl; // Length = 9.4245
    cout << "Area = " << area << endl; // Area = 28.2735
    cout << "Volume = " << vol << endl; // Volume = 113.094

    return 0;
}
```

Результат роботи:

```
Length = 18.849
Area = 28.2735
Volume = 113.094
```

Особливості дружніх функцій:

− «дружня» функція не отримує покажчик this класу, в якому вона реалізована;

− об’єкти класів передаються в «дружню» функцію явно через параметри;

− «дружні» функції не викликаються через об’єкт класу, друзями якого вони є;

− на «дружні» функції не діють специфікатори доступу private, protected, public;

− «дружня» функція не є компонентою того класу, до якого вона «дружня»;

− «дружня» функція може бути «дружньою» відносно декількох класів.

Оператори new і delete []

Виділення пам’яті для структурних змінних, об’єктів класів, масивів. Ініціалізація виділеної пам’яті. Приклад перерозподілу раніше виділеної пам’яті.

Приклад динамічного виділення пам’яті для структурної змінної

Виділення і звільнення пам’яті для структурної змінної.

Нехай задано структуру Date, яка має такий опис:

```
// структура, що описує день року
struct Date
{
   int day;
   int month;
   int year;
};
```

Приклад виділення і використання пам’яті для змінної типу struct Date

```
// структура, що описує день року
#include "cstdlib"
#include <iostream>
using namespace std;

// структура, що описує день року
struct Date
{
   int day;
   int month;
   int year;
};

int main()
{
   // виділення пам’яті для структурної змінної оператором new
   struct Date * pd; // покажчик на struct Date

   try
   {
      pd = new struct Date; // спроба виділити пам’ять
   }
   catch (bad_alloc ba)
   {
      cout << "Пам\’ять не виділено" << endl;
      return -1;
   }

   // якщо пам’ять виділено, то
   // використання pd у програмі
   // встановити дату 01.01.2023
   pd->day = 1;
   pd->month = 1;
   pd->year = 2023;

   cout << pd->month << endl;
```

```
    // звільнити пам’ять використану під змінну pd
    delete pd;

    return 0;
}
```
Результат роботи:
1

Приклад динамічного виділення пам’яті для покажчика на об’єкт класу CDayWeek

```
#include "cstdlib"
#include <iostream>
using namespace std;

// клас, що описує день тижня
class CDayWeek
{
    int d;
    public:
    // конструктор
    CDayWeek() { d = 1; }
    // методи доступу
    int Get(void) { return d; }
    void Set(int nd)
    {
        if ((1<=nd)&&(nd<=7))
            d = nd;
    }
};

int main()
{
    // виділення пам’яті для об’єкту класу оператором new
    CDayWeek *p;
    try
    {
        // спроба виділити пам’ять для покажчика p
        p = new CDayWeek(); // викликається конструктор класу
CDayWeek
    }
```

```
    catch (bad_alloc ba)
    {
       cout << "Пам\'ять не виділено" << endl;
       cout << ba.what() << endl;
       return -1;
    }
    // якщо пам'ять виділено, то використання класу
    p->Set(3);
    cout << p->Get() << endl;
    // звільнити пам'ять, на яку вказує покажчик p
    delete p;
    return 0;
}
```

Результат роботи:

3

Оператор new може бути використаний для виділення пам'яті для масиву. Загальна форма оператора new у разі виділення пам'яті для масиву покажчиків:

ptrArray = новий тип [size];
де

ptrArray – ім'я масиву, для якого виділяється пам'ять;
type – тип елементів масиву.
Тип елементів може бути базовий (int, float, …) або інша структура, клас тощо);
size – розмір масиву (кількість елементів).

Для звільнення пам'яті, виділеної під масив, оператор delete має таку форму використання:

delete[] ptrArray;
де ptrArray – ім'я масиву, для якого виділяється пам'ять.

Приклад динамічного виділення і звільнення пам'яті для масиву покажчиків на базовий тип

У прикладі виділяється пам'ять для масиву покажчиків на тип float. Потім елементи масиву заповнюються довільними значеннями. Після цього, виділена пам'ять звільняється оператором delete[].

```
// оголосити масив покажчиків на float
float * ptrArray;

ptrArray = new float[10]; // виділити пам’ять для 10 елементів типу float

// Використання масиву
int d;
for (int i = 0; i < 10; i++)
    ptrArray[i] = i * 2 + 1;

d = ptrArray [3]; // d = 7

delete[] ptrArray; // знищити пам’ять, виділену під масив
```

Примітка: Про ці оператори ми вже докладно говорили в темі масиви.

Приклад виділення і звільнення пам’яті для масиву з 3-х структур типу TStudent та способів доступу до полів заданого елемента в масиві структур

```
#include "cstdlib"
#include <iostream>
using namespace std;

// структура, що описує інформацію про студента
struct TStudent
{
   string name; // П.І.Б. студента
   string numBook; // номер залікової книжки
   int course; // курс навчання
   float rank; // рейтинг студента
};
int main()
{
   // виділення пам’яті для масиву структур
   int n_students = 3; // к-сть студентів
   TStudent * pS; // покажчик, масив структур

   try
```

```
    {
        // спроба виділити пам’ять для масиву структур
        pS = new struct TStudent[n_students];
    }
    catch (bad_alloc ba)
    {
        cout << "Пам’ять не виділено" << endl;
        cout << ba.what() << endl;
        return –1;
    }

    // якщо пам’ять виділено, то використання масиву структур
    // доступ до елементу масиву з індексом 0
    pS->name = "Johnson B.";
    pS->numBook = "12389239";
    pS->rank = 3.93;
    pS->course = 4;

    // доступ до елементу масиву з індексом 1 – теж добре
    (pS + 1)->name = "Sullivan";
    (pS + 1)->numBook = "20930032";
    (pS + 1)->rank = 4.98;
    pS[1].course = 3;

    // доступ до елементу масиву з індексом 2 ще один спосіб
    pS[2].name = "Abram F.D.";
    pS[2].numBook = "l2l28983";
    pS[2].rank = 4.32;
    pS[2].course = 5;

    // вивід деяких значень
    cout << pS->name.c_str() << endl; // "Johnson J."
    cout << pS[1].rank << endl;       // 4.98
    cout << (pS + 2)->numBook.c_str() << endl;       // "12128983"
    // звільнити пам’ять, на яку вказує покажчик p
    delete[] pS;

    return 0;
}
```

Результат роботи:
Johnson B.
4.98
l2l28983

Приклад виділення і звільнення пам'яті для масиву об'єктів

```
#include "cstdlib"
#include <iostream>
using namespace std;

// клас, що описує місяць року
class CMonth
{
   private:
   int month;

   public:
   // конструктор без параметрів, якщо використовується масив об'єктів,
   // то цей конструктор є обов'язковим
   CMonth() { month = 1; }

   // конструктор з 1 параметром, для створення масиву об'єктів не підходить,
   // підходить тільки для одиночних об'єктів
   CMonth(int nmonth)
   {
      if ((1 <= nmonth) && (nmonth <= 12))
         month = nmonth;
      else
         month = 1;
   }
   // методи доступу
   int Get(void) { return month; }
   void Set(int nmonth)
   {
      if ((1 <= nmonth) && (nmonth <= 12))
         month = nmonth;
   }
};
```

```
int main()
{
   // виділення пам’яті для масиву об’єктів класу
   int nMonths = 12;
   CMonth * pM;

   try
   {
      // спроба виділити пам’ять для масиву з 12 об’єктів
      pM = new CMonth[12]; // для кожного елементу викликається конструктор без параметрів
   }
   catch (bad_alloc ba)
   {
      cout << "Пам\’ять не виділено" << endl;
      cout << ba.what() << endl;
      return -1;
   }

   // якщо пам’ять виділено, то заповнення масиву значеннями
   for (int i = 1; i <= 12; i++)
      pM[i - 1].Set(i);

   // вивід деяких значень
   cout << pM[3].Get() << endl; // 4
   cout << pM[5].Get() << endl; // 6

   // звільнити пам’ять, на яку вказує покажчик p
   delete[] pM;

   return 0;
}
```

Результат роботи:

```
4
6
```

У наведеному вище коді, внутрішня змінна в масиві об’єктів ініціалізується значенням 1, тому що таке значення задано в конструкторі без параметрів CMonth()

// конструктор без параметрів є обов’язковим, тому що він виступає як ініціалізатор масиву

CMonth() {month = 1; }

Цей конструктор виступає ініціалізатором масиву. Однак, у класі реалізовано ще один конструктор – конструктор з 1 параметром або параметризованим конструктором. Згідно із синтаксисом C++, масив не може бути ініціалізований параметризованим конструктором. Тому, в класі CMonth обов'язково має бути реалізований конструктор без параметрів.

Якщо конструктор без параметрів CMonth() забрати з коду класу, то неможливо буде виділити пам'ять для масиву об'єктів. Можна буде виділяти пам'ять для одиночних об'єктів, але не для масиву.

Тобто, якщо потрібно виділити пам'ять для масиву об'єктів деякого класу, то цей клас обов'язково має мати реалізацію конструктора без параметрів.

Як перерозподілити пам'ять, якщо потрібно динамічно збільшити розмір масиву? Перерозподіл пам'яті для структур, ініціалізація структур

У прикладі демонструється процес перерозподілу пам'яті для типу структури DayWeek. Виділення і перерозподіл пам'яті динамічно є основною перевагою цього способу порівняно зі статичним виділенням пам'яті. Пам'ять у програмі можна виділяти коли потрібно та скільки потрібно.

У структурі DayWeek реалізовано конструктор без параметрів (за замовчуванням), який ініціалізує масив структур зі значенням за замовчуванням.

Приклад перерозподілу пам'яті для структур, ініціалізація структур

```
#include "cstdlib"
#include <iostream>
using namespace std;
// структура, що описує день тижня
struct DayWeek
{
   int day;

   // структура як і клас також може мати конструктор
   DayWeek() { day = 1; }
};
```

```
int main()
{
   // виділення пам’яті для масиву структур та ініціалізація
   DayWeek * p;

   try
   {
      // спроба виділити пам’ять для масиву з 5 структур типу
DayWeek
      p = new DayWeek[5];
   }
   catch (bad_alloc ba)
   {
      cout << "Пам\’ять не виділено" << endl;
      cout << ba.what() << endl;
      return -1;
   }

   // якщо пам’ять виділено, то заповнення масиву значеннями
   for (int i = 0; i < 5; i++)
      p[i].day = i + 1;

   // використання масиву з 5 структур
   int d;
   d = (p + 1)->day; // d = 2

   // Демонстрація збільшення (перерозподілу) пам’яті,
   // потрібно збільшити до 7 елементів в масиві (7 днів на
тиждень)
   // Оголосити покажчик на новий масив
   DayWeek * p2;
   try
   {
      // Спробувати виділити пам’ять для нового масиву (для
7 елементів)
      p2 = new DayWeek[7];
   }
   catch (bad_alloc ba)
   {
      // якщо помилка, то завершення з кодом -1
```

```
        cout << "Пам’ять не виділено" << endl;
        cout << ba.what() << endl;
        return -1;
    }

    // Зберегти перші 5 елементів старого масиву (скопіювати p
=> p2)
    for (int i = 0; i < 5; i++)
        p2[i] = p[i];

    // Звільнити пам’ять, що була виділена для старого масиву p
    delete[] p;

    // Перенаправити покажчик з p на p2.
    p = p2; // Обидва покажчики вказують на одну й ту ж
область пам’яті

    // Використати масив з 7 елементів, наприклад, дозапов-
нювати 2 останні
    for (int i = 5; i < 7; i++)
        p[i].day = i + 1;

    d = p[5].day; // d = 6
    d = (p + 6)->day; // d = 7

    // Після завершення роботи, звільнити пам’ять для покаж-
чика p
    delete[] p; // можна було delete[] p2; оскільки p і p2 вказують
на ту саму область пам’яті

    return 0;
}
```

У функції main() спочатку виділяється пам’ять для масиву з 5 структур. Потім ця пам’ять перерозподіляється для масиву з 7 структур. Для цього використовується додатковий показник p2.

При перерозподілі спочатку пам’ять виділяється для p2 (7 елементів). Потім копіюються дані з p у p2. Після цього звільняється пам’ять, що була виділена для покажчика p (5 елементів).

Наступним кроком значення p встановлюється рівним значенню p2. Таким чином, обидва покажчики вказують на одну і ту ж ділянку пам'яті.

По завершенню роботи із вказівниками, пам'ять для масиву p звільняється.

Завдання для самостійної роботи

Завдання 1. Структури

Написати програми мовою програмування C++ для задач, які наведено нижче (згідно зі своїм варіантом).

Варіант 1

Описати структуру з ім'ям STUDENT, яка містить такі поля:

- NAME – прізвище та ініціали;
- GROUP – номер групи;
- SES – оцінки з п'яти предметів (масив з п'яти елементів).

Написати програму, що реалізує такі дії окремими функціями:

- уведення з клавіатури даних у масив STUD, що складається з N змінних типу STUDENT;
- упорядкування записів за зростанням значень поля GROUP;
- виведення на екран прізвищ і номерів груп для всіх студентів, середній бал яких більший за 4.0; якщо таких студентів немає, то вивести відповідне повідомлення.

Варіант 2

Описати структуру з ім'ям ABITURIENT, яка містить такі поля:

- NAME – прізвище, ініціали:
- GENDER – стать:
- SPEC – назва спеціальності:
- EXAM – результати вступних іспитів із трьох предметів (масив із трьох елементів).

Написати програму, що окремими функціями реалізує такі дії:

- уведення з клавіатури даних у масив ABITUR, що складається з N змінних типу ABITURIENT;
- упорядкування записів за зростанням середнього балу;

− виведення на екран прізвищ і назв спеціальностей для всіх абітурієнтів, що мають бал нижче, ніж прохідний, який визначається користувачем програми; якщо таких студентів немає, то вивести відповідне повідомлення.

Варіант 3

Описати структуру з ім'ям SCHOOL, яка містить такі поля:

− NAME – прізвище та ім'я учня;

− GROUP – номер групи:

− SUBJECT – успішність з п'яти предметів (масив із п'яти елементів).

Написати програму, що окремими функціями виконує такі дії:

− введення з клавіатури даних у масив LEARNER, що складається з N змінних типу SCHOOL;

− впорядкування записів за алфавітом;

− виведення на екран прізвищ і номерів груп для всіх студентів що мають хоча б одну оцінку 2: якщо таких студентів немає то вивести відповідне повідомлення.

Варіант 4

Описати структуру з ім'ям AEROFLOT, яка містить такі поля:

− CITY – назва населеного пункту призначення:

− NUM – номер рейсу;

− TYPE – тип літака.

Написати програму, що окремими функціями реалізує такі дії:

− уведення з клавіатури даних в масив AIR. що складається з N змінних типу AVIAEXPRESS;

− упорядкування записів за зростанням номеру рейсу;

− виведення на екран номерів рейсів і типів літаків, що вилетіли в пункт призначення, назва якого співпала з назвою, введеною з клавіатури; якщо таких рейсів немає, то вивести відповідне повідомлення.

Варіант 5

Описати структуру з ім'ям SKLAD. яка містить такі поля:

− NAME – назва товару;

− TYPE – одиниця виміру товару;

− QUANTITY – кількість одиниць товару;

− COST – ціна одиниці товару.

Написати програму, що окремими функціями виконує такі дії:

− уведення з клавіатури даних у масив SHOP, що складається з N змінних типу SKLAD:

− упорядкування записів за назвами товару;

− виведення на екран інформації про товар, його кількість, ціну одиниці та обчислену загальну суму на складі, назва якого вводиться з клавіатури; якщо такого немає, то вивести відповідне повідомлення.

Варіант 6

Описати структуру з ім'ям WORKER, яка містить такі поля:

− NAME – прізвище та ініціали працівника;

− POS – назва посади;

− YEAR – рік прийняття на роботу;

− MONTH – місяць прийняття на роботу.

Написати програму, що окремими функціями виконує такі дії:

− уведення з клавіатури даних у масив TABL. що складається з N змінних типу WORKER:

− упорядкування записів в алфавітному порядку:

− виведення на екран прізвищ працівників, стаж роботи яких перевищує значення, введене з клавіатури; якщо таких працівників немає, то вивести відповідне повідомлення.

Варіант 7

Описати структуру з ім'ям TRAIN, яка містить такі поля:

− NAZV – назва пункту призначення;

− NUMR – номер потягу;

− DATE – дата відправлення;

− TIME – час відправлення.

Написати програму, що окремими функціями виконує такі дії:

− уведення з клавіатури даних у масив RASP, що складається з N змінних типу TRAIN;

− упорядкування записів за алфавітом за назвами пунктів призначення;

− виведення на екран інформації про поїзди, що відправляються після введеного з клавіатури дня та часу; якщо таких поїздів немає, то вивести відповідне повідомлення.

Варіант 8

Описати структуру з ім'ям TIMETABLE, яка містить такі поля:

- NAZV – назва пункту призначення;
- NUMR – номер поїзда;
- DATE – дата відправлення;
- TIME – час відправлення.

Написати програму, що окремими функціями виконує такі дії:

- уведення з клавіатури даних у масив TRAIN, що складається з N змінних типу TIMETABLE;
- упорядкування записів за датою та часом відправлення поїзда;
- виведення на екран інформації про поїзди, що направляються в пункт призначення, назва якого введена з клавіатури; якщо таких поїздів немає, то вивести відповідне повідомлення.

Варіант 9

Описати структуру з ім'ям TIMETABLE, яка містить такі поля:

- NAZV – назва пункту призначення;
- NUMR – номер поїзда;
- DATE – дата відправлення;
- TIME – час відправлення.

Написати програму, що окремими функціями виконує такі дії:

- уведення з клавіатури даних у масив TRAIN, що складається з N структур типу TIMETABLE;
- упорядкування записів за номерами поїздів;
- виведення на екран інформацію про поїзди, дата відправлення яких введена з клавіатури; якщо таких поїздів немає, то вивести відповідне повідомлення.

Варіант 10

Описати структуру з ім'ям ITINERARY, яка містить такі поля:

- FIRST – назва початкового пункту маршруту;
- FINAL – назва кінцевого пункту маршруту;
- NUM – номер маршруту;
- DISTANCE – відстань у кілометрах.

Написати програму, що окремими функціями виконує такі дії:

− уведення з клавіатури даних у масив ROUT, що складається з N змінних типу ITINERARY;

− упорядкування записів за спаданням відстані у кілометрах;

− виведення на екран інформації про маршрут, номер якого введений з клавіатури: якщо таких маршрутів немає, то вивести відповідне повідомлення.

Завдання 2. Робота з класами

Створити клас для обробки записів бази даних відповідно до наданих варіантів. Розмістити інтерфейс класу у заголовочному файлі, а визначення функцій і головну функцію програми – у двох окремих файлах. Передбачити можливість роботи з довільною кількістю записів, а також реалізувати окремими функціями класу:

− конструктори (без параметрів та з параметрами);
− додавання даних;
− знищення даних;
− виведення інформації на екран;
− пошук потрібної інформації за конкретною ознакою;
− редагування записів;
− сортування за різними полями.

Використайте захищення даних для ізоляції елементів-даних класу від підпрограм, у яких цей клас використовується. Програма повинна містити меню для перевірки всіх методів класу.

№	Предметна область БД	Поля бази даних
1.	«Бібліотека»	Інвентарний номер, автор, назва, кількість сторінок, рік видання.
2.	«Телефонний довідник»	Прізвище, ім’я, по батькові, домашня адреса, телефон.
3.	«Розклад руху літаків»	Номер рейсу, тип літака, напрямок руху, періодичність вильоту.
4.	«Колекція CD-дисків»	Інвентарний номер, назва, об’єм диску, тип, дата запису.
5.	«Записна книжка»	Прізвище, ім’я, по батькові, домашня адреса, телефон, електронна пошта.
6.	«Склад товарів»	Інвентарний номер, назва товару, вага, ціна, кількість.

7.	«Користувачі локальної мережі»	Прізвище, ім’я, по батькові, група, обліковий запис, тип облікового запису.
8.	«Предметний покажчик»	Слово, номера сторінок, де це слово зустрічається.
9.	«Банківські рахунки»	Прізвище, ім’я, дата останньої операції, сума останньої операції, сума вкладу.
10.	«Успішність студентів»	Прізвище, ім’я, номер групи, оцінки з трьох предметів.

Тема 7. Перевантаження операцій

Унарні та бінарні оператори

Розрізняють три основні види операторів: унарні, бінарні та n-арні.

Унарні оператори – це оператори, які для обчислення вимаюгають одного операнда, який може розміщуватися праворуч або ліворуч від самого оператора.

Приклади унарних операторів:

```
i++
--a
-8
```

Бінарні оператори – це оператори, які для обчислення вимагають двох операндів. Наприклад, нижче наведено фрагменти виразів із бінарними операторами +, –, %, *

```
a+b
f1-f2
c%d
x1*x2
```

n-арні оператори для обчислень потребують більше двох операндів. У мові C++ є тернарна операція ?:, яка для своєї роботи потребує трьох операндів.

Мова C++ має широкі можливості для перевантаження більшості операторів. Перевантаження оператора означає використання оператора для оперування об’єктами класів. Перевантаження оператора – спосіб оголошення і реалізації оператора таким чином, що він обробляє об’єкти конкретних класів або виконує деякі інші дії. При перевантаженні оператора в класі

викликається відповідна операторна функція (operator function), яка виконує дії, що стосуються цього класу.

Якщо оператор «перевантажено», то його можна використовувати в інших методах у звичайному для нього вигляді. Наприклад, команди поелементного сумування двох масивів a1 та a2

```
a1.add(a2);
a3 = add(a1, a2);
```

краще викликати більш природним способом:

```
a1 = a1 + a2;
a3 = a1 + a2;
```

У цьому прикладі оператор '+' вважається перевантаженим.

Для заданого класу операторну функцію у класі можна реалізувати:

− всередині класу. У цьому випадку, операторна функція є методом класу;

− за межами класу. У цьому випадку операторна функція оголошується за межами класу як «дружня» (з ключовим словом friend).

Загальний синтаксис операторної функції, яка реалізована у класі, має такий вигляд:

```
return_type ClassName::operator#(arguments_list)
{
    // деякі операції
    // ...
}
```

де

return_type – тип значення, що повертається операторною функцією;

ClassName – ім’я класу, в якому реалізовано операторну функцію;

operator# – ключове слово, що визначає операторну функцію у класі. Символ # замінюється оператором мови C++, який перевантажується. Наприклад, якщо перевантажується оператор +, потрібно вказати operator+;

argument_list – список параметрів, які отримує операторна функція. Якщо перевантажується бінарний оператор, argument_list містить один аргумент. Якщо перевантажується унарний оператор, то перелік аргументів порожній.

Приклад перевантаження унарних і бінарних операторів для класу, який містить одиночні дані

Оголошується клас Point, що реалізує точку на координатній площині. У класі реалізовано:
дві внутрішні змінні x, y, що є координатами точки;
два конструктори класу;
методи доступу до внутрішніх змінних класів GetX(), GetY(), SetX(), SetY();
дві операторні функції operator+() та operator-(). Операторна функція operator+() перевантажує бінарний оператор '+'. Операторна функція operator-() перевантажує унарний оператор '-'.

```
// Клас, що реалізує точку на координатній площині
// клас містить дві операторні функції
class Point
{
private:
    int x, y; // координати точки

public:
    // конструктори класу
    Point()
    {
       x = y = 0;
    }

    Point(int nx, int ny)
    {
       x = nx;
       y = ny;
    }

    // методи доступу до членів класу
    int GetX(void) { return x; }
    int GetY(void) { return y; }
    void SetX(int nx) {x = nx; }
    setY(int ny) { y = ny; }

    // перевантажень бінарний оператор '+'
```

```
    Point operator+(Point pt)
    {
      // p – часовий об’єкт, який створюється за допомогою конструктора без параметрів
       Point p;
       p.x = x + pt.x;
       p.y = y + pt.y;
       return p;
    }

    // перевантажень унарний оператор '-'
    Point operator-(void)
    {
       Point p;
       p.x = -x;
       p.y = -y;
       return p;
    }
};
```

Як видно з наведеного вище коду, операторна функція operator+() отримує один параметр. Це означає, що ця функція реалізує бінарний оператор '+'. Цей параметр відповідає операнду, що розміщується у правій частині бінарного оператора +. Операнд, що розміщується в лівій частині оператора '+' передається операторній функції неявно за допомогою покажчика цього класу.

Виклик операторної функції здійснює об’єкт, який розміщується у лівій частині оператора присвоювання.

Демонстрація використання перевантажених операторів класу Point в іншому методі:

Демонстрація використання перевантажених операторів класу Point

```
// оголошення змінних – об’єктів класу CPoint
Point P1(3,4);
Point P2(5,7);
Point P3;
int x, y; // додаткові змінні

// Використання перевантаженого бінарного оператора '+'
P3 = P1 + P2; // Об’єкт P1 викликає операторну функцію
```

```
// перевірка
x = P3.GetX(); // x = 8
y = P3.GetY(); // y = 11

// Використання перевантаженого унарного оператора '-'
P3 = -P2;
x = P3.GetX(); // x = -5
y = P3.GetY(); // y = -7
```

У наведеному вище коді в операції додавання '+' об'єкт P1 викликає операторну функцію. Тобто фрагмент рядка

```
P1 + P2
```

замінюється викликом

```
P1.operator+(P2)
```

Реалізувати операторну функцію operator+() у класі можна й по іншому

```
Point operator+(Point pt)
{
    // Виклик конструктора з двома параметрами
    return Point(x+pt.x, y+pt.y); // створюється тимчасовий об'єкт, який потім копіюється
}
```

У вищенаведеній функції в операторі return створюється часовий об'єкт шляхом виклику конструктора із двома параметрами, який реалізований у класі. Якщо (у цьому випадку) з тіла класу забрати конструктор із двома параметрами

```
// конструктор із двома параметрами
Point(int nx, int ny)
{
   // ...
}
```

то вищенаведений варіант функції operator+() працювати не буде, тому що для створення об'єкта типу Point ця функція використовує конструктор із двома параметрами. У цьому випадку компілятор видасть повідомлення

```
Point::Point : не overloaded function 2 arguments
```

що означає: немає методу (конструктора) Point::Point(), який приймає 2 аргументи.

Приклад перевантаження оператора * що обробляє клас, який містить масив дійсних чисел. Операторна функція реалізована всередині класу

```
// масив дійсних чисел
class ArrayFloat
{
private:
    float A[10]; // масив дійсних чисел, фіксований розмір масиву
    int size;

public:
    // конструктори
    ArrayFloat()
    {
       size = 0;
    }

    ArrayFloat(int nsize, float nA[])
    {
       size=nsize;
       for (int i=0; i<nsize; i++)
          A[i] = nA[i];
    }

    // методи доступу
    float GetAi(int i)
    {
       if ((i>=0) && (i<=size-1))
          return A[i];
       else
          return 0;
    }
    void SetAi(int i, float value)
    {
       if ((i>=0) && (i<=size-1))
          A [i] = value;
    }

    // Перевантажень оператор '*'
    ArrayFloat operator*(ArrayFloat AF)
```

```
    {
        ArrayFloat tmp;
        int n;

        if (size<AF.size)
            n = AF.size;
        else
            n = size;

        for (int i=0; i<n; i++)
            tmp.A[i] = A[i] * AF.A[i];
        tmp.size = n;
        return tmp;
    }
};
```

Приклад додавання двох масивів. Операторна функція operator+() розміщується всередині класу

Задано клас ArrayFloat, що реалізує динамічний масив чисел типу float. У класі реалізовано:

внутрішні змінні size, A, які визначають розмір масиву та сам масив;
два конструктори, які ініціалізують початковими значеннями елементи масиву;
конструктор копіювання;
методи доступу GetSize(), SetSize(), GetAi(), SetAi(), які реалізують доступ до внутрішніх змінних масиву з відповідними операціями (виділення пам'яті, перевірка на допустимі межі);
операторну функцію operator=(), яка реалізує копіювання об'єктів;
операторну функцію operator+(), яка реалізує перевантаження оператора '+' для масивів типу ArrayFloat. Операторна функція додає поелементно значення масивів, які є операндами операції "+";
деструктор.

```
// клас – масив типу float
class ArrayFloat
{
private:
```

```
    int size; // розмір масиву
    float * A; // динамічний розмір масиву

public:
    // конструктори класу
    // конструктор без параметрів
    ArrayFloat()
    {
       size = 0;
       A = NULL;
    }

    // конструктор із двома параметрами
    ArrayFloat(int nsize, float * nA)
    {
       if (size>0)
          delete A;

       size=nsize;
       A = new float [size];

       for (int i=0; i<nsize; i++)
          A[i] = nA[i];
    }

    // конструктор копіювання
    ArrayFloat(const ArrayFloat&_A)
    {
       size = _A.size;
       A = new float [size];
       for (int i = 0; i < size; i++)
          A[i] = _A.A[i];
    }
    // методи доступу
    int GetSize(void) { return size; }
    void SetSize(int nsize)
    {
       if (size>0)
       {
          delete A;
```

```
        size = 0;
    }

    size=nsize;
    A = new float [size]; // виділити новий фрагмент пам'яті

    // заповнити масив нулями
    for (int i=0; i<size; i++)
        A [i] = 0.0f;
}

float GetAi(int index)
{
    if ((index>=0) && (index<size))
        return A[index];
    else
        return 0;
}

void SetAi(int index, float value)
{
    if ((index>=0) && (index<size))
        A[index] = value;
}

// оператор копіювання operator=()
ArrayFloat operator=(const ArrayFloat& _A)
{
    if (size > 0)
        delete [] A;
    size = _A.size;
    A = new float [size]; // виділити пам'ять

    for (int i = 0; i < size; i++)
        A[i] = _A.A[i];

    return *this;
}
```

```
    // Перевантаження оператора '+',
    // Операційна функція
    ArrayFloat operator+(ArrayFloat AF2)
    {
       int n;
       // взяти мінімальний розмір з двох масивів
       if (size<AF2.size) n = size;
       else n = AF2.size;

       ArrayFloat tmpA; // Об'єкт класу
       tmpA.SetSize(n); // новий розмір масиву

       // поелементне додавання
       for (int i=0; i<n; i++)
          tmpA.A[i] = A[i] + AF2.A[i];
       return tmpA;
    }

    // деструктор
    ~ArrayFloat()
    {
       if (size > 0)
          delete [] A;
    }
};
```

Нижче показано використання класу ArrayFloat та операторної функції operator+() цього класу.

```
// додаткові змінні, масиви
float x, y;
float F1[] = {3.8f, 2.9f, 1.5f};
float F2[] = {4.3f, 1.5f, 7.0f, 3.3f};

// оголошення об'єктів класу ArrayFloat
ArrayFloat AF1(3, F1);
ArrayFloat AF2(4, F2);
// перевірка
x = AF1. GetAi (0); // x = 3.8
y = AF2.GetAi(3); // y = 3.3

// оголошення додаткового об'єкта – результату
```

```
ArrayFloat AF3;
AF3 = AF1 + AF2; // Виклик операторної функції operator+

// перевірка
x = AF3. GetAi (0); // x = 3.8 + 4.3 = 8.1
y = AF3.GetAi(1); // y = 2.9 + 1.5 = 4.4

//ще один визов
AF3 = AF1 + AF1 + AF2;

x = AF3. GetAi (1); // x = 2.9 + 2.9 + 1.5 = 7.3
y = AF3.GetAi(2); // y = 1.5 + 1.5 + 7.0 = 10.0
```

На використання перевантажених операторів накладаються такі обмеження:

− при перевантаженні оператору не можна змінити пріоритет цього оператора;

− не можна змінити кількість операндів оператора. Однак, у коді операторної функції можна один із параметрів (операндів) не використовувати;

− не можна перевантажувати оператори **:: . *?:;**

− не можна викликати операторну функцію з аргументами за замовчуванням. Виняток – операторна функція виклику функції operator()().

Не можна перевантажувати такі оператори:

:: – розширення області видимості;
. (крапка) – вибір члена класу чи структури;
***** – доступ за вказівником;
?: – тернарний оператор.

Особливості перевантаження інкрементних і декрементних операторів ++, - -

Інкрементні та декрементні оператори в мові С++ можуть бути перевантажені. Особливість перевантаження цих операторів полягає в тому, що потрібно перевантажувати як префіксну, так і постфіксну форму цих операторів.

Як відомо, у мові С++ операції ++ та – мають префіксну і постфіксну форми, як показано у прикладі:

```
++x; // префіксна форма оператора інкремента
x++; // Постфіксна форма оператора
```

--x; // префіксна форма оператора декрементуа
x--; // Постфіксна форма оператора декременту

Відповідно до синтаксису C++ оператори ++ та – потребують одного операнда.

Згідно з синтаксисом C++ оператори декременту та інкременту потребують одного операнда. Виникає питання: як реалізувати префіксну та постфіксну перевантажену операторну функцію так, щоб вони відрізнялися (не було співпадіння)?

Для того щоб при реалізації в класі, відрізнити префіксну та постфіксну форми реалізації операторної функції ++ або —, потрібно дотримуватися таких правил:

− якщо перевантажується префіксна форма оператора++, то у класі потрібно реалізувати операторну функцію operator++() без параметрів;

− якщо перевантажується префіксна форма оператора – –, то у класі потрібно реалізувати операторну функцію operator- -() без параметрів;

− якщо перевантажується постфіксна форма оператора++, то в класі потрібно реалізувати операторну функцію operator++(int d) з одним цілочисельним параметром. У цьому випадку параметр d не використовується у функції. Він указується тільки для того, щоб указати, що це саме постфіксна реалізація оператора ++. Ім'я d може бути замінене іншим ім'ям;

− якщо перевантажується постфіксна форма оператора – –, то у класі потрібно реалізувати операторну функцію operator–(int d) з одним цілочисельним параметром. Параметр d необхідний для зазначення--.

Наприклад

```
class SomeClass
{
    // ...
    // операторна функція, яка переванажує префіксний оператор ++ (++X)
    type operator++()
    {
        // тіло операторної функції
        // ...
    }
```

```
        // операторна функція, яка переванажує постфіксний
оператор ++ (X++)
        type operator++(int d)
        {
           // тіло операторної функції
           // ...
        }

        // операторна функція, яка переванажує префіксний
оператор -- (--X)
        type operator--()
        {
           // тіло операторної функції
           // ...
        }

        // операторна функція, яка переванажує постфіксний
оператор -- (X--)
        type operator--(int d)
        {
           // тіло операторної функції
           // ...
        }
    };
```

Приклад перевантаження операторних функцій ++ та --у класі для одиночних об'єктів

Демонструється реалізація операторних функцій для класу Integer, який реалізує цілочислове значення.

Оголошується клас Integer, який містить:

- внутрішню змінну number цілого типу. Ця змінна зберігає цілочисельне значення;
- два конструктори, які ініціалізують змінну number;
- методи доступу до змінної number Get(), Set();
- дві операторні функції, які перевантажують оператор ++;
- дві операторні функції, що перевантажують оператор --.

Приклад перевантаження операторів для класу Integer

```
// перевантаження інкрементних операторів для класу Integer
class Integer
```

```
{
private:
    int number; // Член даних

public:
    // конструктори класу
    // конструктор без параметрів
    Integer() { number = 0; }

    // конструктор з 1 параметром
    Integer(int _number) { number = _number; }

    // методи доступу
    void Set(int _number) {number = _number; }
    int Get(void) { return number; }

    // операторна функція, яка перевантажує
    // префіксну форму оператора ++ для класу Integer
    Integer operator++(void)
    {
       number++;
       return *this; // повернути об'єкт даного класу
    }

    // операторна функція, яка перевантажує
    // Постфіксну форму оператора ++ для класу Integer
    Integer operator++(int d) // параметр d не використовується
    {
       number++;
       return *this;
    }
    // перевантаження префіксного -
    Integer operator-(void)
    {
       number--;
       return *this;
    }

    // перевантаження постфіксного -
```

```
    Integer operator--(int d) // параметр d не використовується
    {
       number--;
       return *this;
    }
};

Використання класу
// демонстрація використання перевантажених інкрементних
// операторних функцій
Integer d1, d2(5); // оголошення об'єктів класу Integer
Integer d3;
int t1, t2;

// перевірка
t1 = d1.Get(); // t1 = 0
t2 = d2.Get(); // t2 = 5

// виклик перевантажених операторних функцій
++d1; // виклик префіксної операторної функції operator++()
t1 = d1.Get(); // t1 = 1

d2++; // виклик постфіксної операторної функції
operator++(int)
t2 = d2.Get(); // t2 = 6

--d1; // префіксний --
d1--; // постфіксний --
t1 = d1.Get(); // t1 = -1
```

Якщо оператори ++ та — перевантажуються за допомого дружніх функцій до класу, то потрібно враховувати таке:

- дружня до класу функція не отримує покажчика this класу;
- оскільки дружня функція не отримує покажчика this, то параметр у цю функцію має передаватись за посиланням а не за значенням. Це означає, що перед іменем параметру вказується символ посилання &.

Відмінності у перевантаженні префіксних і постфіксних операторів інкременту (++) та декременту (- -) з допомогою «дружніх» функцій

Для того, щоб при реалізації дружньої функції до класу, відрізнити префіксну та постфіксну форми реалізації операторної функції ++ або --, потрібно дотримуватись таких правил:

− якщо перевантажується префіксна форма оператора ++ (++x), то «дружня» операторна функція повинна отримувати один параметр. Цей параметр є посиланням на клас, для якого ця функція реалізована;

− якщо перевантажується префіксна форма оператора -- (--x), то «дружня» операторна функція повинна отримувати один параметр, який є посиланням на клас, у якому ця функція реалізована;

− якщо перевантажується постфіксна форма оператора ++ (x++), то «дружня» операторна функція отримує два параметри. Перший параметр є посиланням на дружній клас. Другий параметр є формальним параметром цілого типу, який є індикатором того, що перевантажується саме постфіксна, а не префіксна форма оператора ++. Другий параметр у функції не використовується;

− якщо перевантажується постфіксна форма оператора -- (x--), то дружня операторна функція отримує два параметри. Перший параметр є посиланням на дружній клас. Другий параметр не використовується, а служить для визначення того, що саме постфіксна форма оператора -- реалізована.

Наприклад

```
// оголошення деякого класу
class SomeClass
{
   // тіло класу
   // ...
   // оголошення дружніх функцій для класу SomeClass
   friend SomeClass operator++(SomeClass & ref); // ++x
   friend SomeClass operator--(SomeClass & ref); // --x
   friend SomeClass operator++(SomeClass & ref, int d); // x++
   friend SomeClass operator--(SomeClass & ref, int d); // x--
};
```

```
// оголошення операторної функції,
// яка перевантажує префіксний оператор ++ (++x)
SomeClass operator++(SomeClass & ref)
{
   // ...
}

// оголошення операторної функції,
// яка перевантажує префіксний оператор -- (--x)
SomeClass operator++(SomeClass & ref)
{
   // ...
}

// оголошення операторної функції,
// яка перевантажує постфіксний оператор ++ (x++)
SomeClass operator++(SomeClass & ref, int d)
{
   // ...
}

// оголошення операторної функції,
// яка перевантажує постфіксний оператор -- (x--)
SomeClass operator++(SomeClass & ref, int d)
{
   // ...
}
```

Як видно з прикладу, «дружня» операторна функція отримує посилання на «дружній» клас.

Перевантажуються оператори ++ та -- для класу Integer за допомогою «дружніх» операторних функцій. «Дружні» операторні функції реалізуються за межами класу

```
// клас Integer, що реалізує ціле число
class Integer
{
private:
    int number;

public:
    // конструктори класу
```

```
    Integer() { number = 0; }
    Integer(int _number) { number = _number; }

    // методи доступу
    int Get(void) { return number; }
    void Set(int _number) {number = _number; }

    // оголошення "дружніх" операторних функцій
    friend Integer operator++(Integer & ref);
    friend Integer operator-(Integer & ref);
    friend Integer operator++(Integer & ref, int d);
    friend Integer operator-(Integer & ref, int d);
};

// Реалізація "дружніх" операторних функцій
Integer operator++(Integer & ref)
{
    ref.number++; // доступ за посиланням
    return ref;
}

Integer operator--(Integer & ref)
{
    ref.number--;
    return ref;
}

// Постфіксна форма - x++
Integer operator++(Integer & ref, int d)
{
    ref.number++;
    return ref;
}
// Постфіксна форма - x--
Integer operator--(Integer & ref, int d)
{
    ref.number--;
    return ref;
}
```

Приклад використання класу Integer в іншому методі

```
// "дружні" операторні функції, перевантаження операторів ++, --
Integer d1, d2(8);
int t;

// перевірка
t = d1. Get (); // t = 0
t = d2.Get(); // t = 8

// Виклик "дружніх" операторних функцій
++d1; // "дружня" префіксна операторна функція operator++()
d2--; // "дружня" постфіксна операторна функція operator--()

t = d1. Get (); // t = 1
t = d2.Get(); // t = 7
```

Перевантаження скорочених операторів присвоювання +=, –=, *=, /=, %=

Скорочені оператори присвоювання +=, -=, *=, /=, %= є бінарними, тобто вони вимагають двох операндів.

Якщо операторна функція реалізована в класі, то вона отримує один операнд і має такий загальний синтаксис:

```
class ClassName
{
   // ...

   // операторна функція, яка перевантажує один із скорочених операторів присвоювання
   ClassName operator##(ClassName obj)
   {
      // ...
   }
}
```

де

ClassName – ім’я класу, для якого реалізується операторна функція operator##();

operator##() – деяка операторна функція. Символі ## замінюються символами +=, -=, *=, /=, %=, тобто операторна функція матиме відповідно такі імена: operator+=(), operator-=(), operator*=(), operator/= (), Operator% = ().

obj – примірник (об’єкт) класу ClassName.

Якщо операторна функція реалізована як «дружня», вона отримує два операнди. У цьому випадку загальна форма використання операторної функції має вигляд:

```
class ClassName
{
    // ...

    // "Дружня" операторна функція,
    // яка перевантажує один із скорочених операторів присвоювання
    friend ClassName operator##(ClassName& obj1, ClassName& obj2)
    {
       // ...
    }
}

// Реалізація "дружньої" до класу ClassName операторної функції
ClassName operator##(ClassName& obj1, ClassName& obj2)
{
    // ...
}
```

тут obj1, obj2 – посилання на примірники класу ClassName, які передаються у «дружню» операторну функцію operator##().

Приклад перевантаження оператора += для класу Integer, який реалізує ціле число. Операторна функція оголошується всередині класу

Задано клас Integer, який реалізує операції над цілочисельним значенням. У класі оголошуються:
внутрішня змінна number цілого типу, яка зберігає значення;
два конструктори;
методи доступу до цілочисельної змінної number;
операторна функція operator+=(), яка перевантажує скорочений оператора присвоювання +=.
Реалізація класу Integer

```
// клас Integer - реалізує ціле число
class Integer
```

```
{
private:
    int number;

public:
    // конструктори класу
    Integer() { number = 0; }
    Integer(int _number) { number = _number; }

    // методи доступу
    int Get(void) { return number; }
    void Set(int _number) {number = _number; }

    // перевантаження оператора +=, операторна функція всередині класу
    Integer operator+=(Integer obj)
    {
       number = number + obj.number; // доступ за посиланням
       return *this;
    }
};
```

Приклад використання об'єкта класу Integer в іншому програмному коді

```
// перевантаження скорочених операторів присвоєння +=, -= і т. д.
Integer d1(10), d2(18); // Примірники класу Integer
int t;

// Виклик операторної функції operator+=()
d1+=d2; // d1.number = d1.number + d2.number

t = d1. Get (); // t = 28
```

За цим зразком можна перевантажувати інші скорочені оператори присвоювання -=, *=, /=, %=.

Приклад перевантаження оператора *= для класу Complex, який реалізує комплексне число. Операторна функція реалізована всередині класу

```
class Complex
{
private:
```

```
    double real; // дійсна частина
    double imag; // уявна частина

public:
    // конструктор класу
    Complex(double _real, double _imag)
    {
        real = _real;
        imag = _imag;
    }

    // методи доступу
    double GetReal(void) { return real; }
    double GetImag(void) { return imag; }

void Set(double _real, double _imag)
{
    real = _real;
    imag = _imag;
}

// операторна функція, яка перевантажує оператор *=
// функція реалізує множення комплексних чисел
Complex operator*=(Complex obj)
{
    double r;
    double i;
    r = real*obj.real - imag*obj.imag;
    i = real*obj.imag + obj.real*imag;
    real = r;
    imag = i;
    return *this;
}
};
Демонстрація використання операторної функції
// перевантаження скороченого оператора *=
Complex c1(2,3); // 2+3j
Complex c2(-1,1); // -1+j
double t;

// виклик операторної функції *=
c1 *= c2; // c1 = -5-1j
```

```
t = c1.GetReal(); // t = -5
t = c1.GetImag(); // t = -1
```

Приклад перевантаження оператора /= для масиву чисел типу double. Масив має фіксований розмір. Операторна функція реалізована всередині класу

У класі ArrayDouble реалізується масив чисел типу double. Операторна функція operator /= реалізує поелементне ділення двох масивів

```
// фіксований масив типу double
class ArrayDouble
{
private:
   double D[10]; // масив

public:
   // конструктор
   ArrayDouble()
   {
      for (int i=0; i<10; i++)
         D[i] = 0;
   }

   // методи доступу
   double Get(int index)
   {
      if ((index>=0)&&(index<10))
         return D[index];
      return 0.0;
   }
   void Set(int index, double value)
   {
      if ((index>=0)&&(index<10))
         D[index] = value;
   }

   // операторна функція, поелементно ділить два масиви
   ArrayDouble operator/=(ArrayDouble AD)
```

```
    {
        // поелементне ділення поточного масиву на масив AD
        for (int i=0; i<10; i++)
            D[i] = D[i] / AD.Get(i);
        return *this;
    }
};
```

Використання класу ArrayDoble може бути таким

```
// перевантаження скороченого оператора *=
ArrayDouble A1;
ArrayDouble A2;
double t;

// формування масивів
for (int i=0; i<10; i++)
   A1.Set(i,10); // A1 = {10,10,10,10,10,10,10,10,10,10}

for (int i=0; i<10; i++) // A2 = {1,2,3,4,5,6,7,8,9,10}
   A2.Set(i, i+1);

// виклик операторної функції
A1 /= A2;

// перевірка
t = A1.Get(0); // t = 10/1 = 10.0
t = A1.Get(1); // t = 10/2 = 5.0
t = A1.Get(2); // t = 10/3 = 3.33333333
t = A1.Get(7); // t = 10/8 = 1.25
```

Особливості перевантаження оператора посилання на член об'єкта ->. Загальна форма. Операторна функція operator->()

У C++ оператор доступу до члена об'єкта -> можна перевантажувати. Якщо оператор -> перевантажений, то виклик елемента класу має такий загальний синтаксис:

obj->item

де

obj – об'єкт класу;

item – деякий елемент класу (внутрішня змінна, метод). Цей елемент повинен бути членом класу, який є доступним усередині об'єкта.

При перевантаженні слід враховувати такі особливості:

− оператор -> вважається унарним;

− операторна функція operator->() повинна повертати покажчик на об’єкт класу, для якого він визначений;

− операторна функція operator ->() повинна бути членом класу, для якого вона реалізується.

Загальна форма класу, в якому перевантажений оператор ->, така

```
class ClassName
{
  // ...

  // операторна функція
  ClassName* operator->()
  {
    // тіло операторної функції
    // ...

    return this; // повернути покажчик на об’єкт класу
  }
};
```

Приклад перевантаження оператору -> в класі, що містить одиночну внутрішню змінну типу double

Оголошується клас Double, що містить внутрішню змінну d типу double. У класі реалізована перевантажена функція operator->(), яка перевантажує оператор ->.

```
#include <iostream>
using namespace std;

class Double
{
public:
  double d; // внутрішня змінна

  // операторна функція, що перевантажує
  // оператор ->
  Double* operator->()
```

```
    {
        return this; // повернути покажчик на клас
    }
};

void main()
{
    // перевантаження оператора доступу за покажчиком ->
    Double D; // екземпляр класу D
    double x;

    D.d = 3.85;

    // виклик операторної функції operator->()
    x = D->d;

    // інший спосіб доступу
    x = D.d; // x = 3.85

    cout << "x = " << x;
}
```

Результат виконання

```
x = 3.85
```

Особливості перевантаження оператора (,) (кома). Операторна функція operator,()

У мові C++ оператор **(,)** може бути перевантажений. При перевантаженні оператора ‘ , ‘ у класі має бути оголошена операторна функція operator,(). У тіло операторної функції можна помістити будь-який код. Тобто, оператор ‘ , ‘ за бажанням може виконувати будь-які нестандартні операції над об’єктами класу.

У стандартному випадку оператор ‘ , ‘ використовується в операції присвоєння за зразком

obj1 = (obj2, obj3, ..., objN);

де obj1, …, objN – екземпляри деякого класу.

У випадку стандартного використання оператора ‘ , ‘ потрібно врахувати такі особливості:

− оператор (,) вважається бінарним. Тому операторна функція operator,() отримує один параметр;

– при використанні перевантаженого оператора (,) в операції присвоєння береться до уваги останній аргумент (цей аргумент є результатом оператора). Усі інші аргументи ігноруються.

У загальному вигляді при перевантаженні оператора (,) клас має такий вигляд

```
class ClassName
{

   // ...

   // операторна функція, яка перевантажує оператор (,)
   ClassName operator,(ClassName obj)
   {
      // ...
   }
};
```

де

ClassName – ім'я класу, в якому перевантажується оператор ' , ';

obj – ім'я екземпляру класу, що передається як параметр в операторну функцію operator,().

Перевантаження операторів через методи класів

Для класів користувача можна перевизначати (задавати) правила обчислення виразів з основними операторами. Для перевантаження певного оператора потрібно описати спеціальну функцію – **операторну функцію** або **операторний метод**. Операторна функція або метод описуються, як і звичайна функція чи метод. Так, назву операторної функції або методу отримують об'єднанням ключового слова **operator** і символу оператора. Тип результату операторної функції бо методу – тип значення, яке повертається під час застосування оператора до об'єкта класу. Аргументи операторної функції визначаються операндами виразу, для якого задається дія оператора. Якщо йдеться про операторний метод, то об'єкт виклику методу ототожнюється з першим операндом виразу.

У прикладі наведено клас MyMoney. Для нього визначено декілька операторних функцій і операторних методів.

Дії з об'єктами класу MyMoney:

Віднімання об’єктів: якщо від одного об’єкту класу MyMoney відняти інший об’єкт класу MyMoney, то у результаті отримаємо різницю кінцевих сум за відповідними вкладами.

Префіксна форма оператора декремента: під час застосування оператора декремента до об’єкта значення поля money об’єкта зменшується на величину 1000 (але залишається зі значенням не меншим за нуль).

Постфіксна форма оператора декремента: під час застосування оператора декремента до об’єкта значення поля time об’єкта зменшується на 1 (але не може бути меншим за нуль).

За допомогою операторних методів задаємо такі дії з об’єктами класу MyMoney:

Під час додавання двох об’єктів класу MyMoney отримуємо новий об’єкт того ж класу, у якого значення полів обчислюється на основі значень полів вихідних об’єктів (тих, що додаються).

Префіксна форма оператора інкремента: під час застосування оператора інкремента до об’єкта значення поля money об’єкта збільшується на величину 1000.

Постфіксна форма оператора інкремента: під час застосування оператора інкремента до об’єкта значення поля time об’єкт збільшується на 1.

Приклад перевантаження операторів через методи класів

На прикладі класу MyMoney

```
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;
class MyMoney {
  public:
 string name;
double money;
double rate;
int time;
   MyMoney ()
   {
      name = " ";
      money = 0;
      rate = 0;
```

```
    time = 0;
}
MyMoney (string n, double m, double r, int t) {
    SetAll (n, m, r, t); }

double getMoney () {
    double S = money;
    for(int k = 1; k <= time; k++){
        S *= (1 + rate/100);
    }
    return S;
    }
void ShowAll() {
    cout << "Name" << name << endl;
    cout << "Money " << money << endl;
    cout << "int. rate % " << rate << endl;
    cout << "Period (years) " << time << endl;
    cout << "Final sum" << getMoney() << endl;
}
void SetAll (string n, double m, double r, int t) {
    name = n;
    money = m;
    rate = r;
    time = t;
}
// префіксна форма оператора інкремента
MyMoney operator++() {
    money=money+1000;
          return *this;
     }
//Постфіксна форма оператора інкремента
MyMoney operator++(int) {
    time++;
    return *this;
     }
// операторний метод для додавання двох обєктів
MyMoney operator+(MyMoney obj){
    MyMoney tmp;
    tmp.name="Harry Potter";
    tmp.money=money+obj.money;
```

```
        tmp.rate=(rate>obj.rate)?rate:obj.rate;
        tmp.time=time+obj.time / 2;
        return tmp;
    }
};

// операторна функція для віднімання обєктів
double operator-(MyMoney objX, MyMoney objY)
{return objX.getMoney() - objY.getMoney();}
//префіксна форма оператора декремента
MyMoney operator--(MyMoney &obj){
    if (obj.money > 1000)
    {obj.money -= 1000;}
    else {obj.money=0;}
    return obj;
}
//постфіксна форма оператора декремента
MyMoney operator--(MyMoney &obj, int){
    if (obj.time > 0)
    {obj.time--;}
    else {obj.time=0;}
    return obj;
}

    int main () {
    MyMoney objA ("Boris Johnson", 1200, 7, 1);
    objA.ShowAll ();
    //зменшення значення поля time
    objA--;
    objA.ShowAll ();
    //зменшення значення поля time
    objA--;
    objA.ShowAll ();
    //збільшення значення поля time
    objA++;
    objA.ShowAll ();
    // зменшення значення поля money
    --objA;
    objA.ShowAll ();
    // зменшення значення поля money
```

```
    --objA;
    objA.ShowAll ();
    // збільшення значення поля money
    ++objA;
    objA.ShowAll ();
    // створення обєкта
MyMoney objB ( "John Snow", 1100, 8, 5);
objB.ShowAll ();
        MyMoney  objC;
        // обчислення суми обєктів
        objC=objA+objB;
        objC.ShowAll ();
        //обчислення різниці обєктів
        cout << objC-objB << endl;

        return 0; }
```

Результат роботи:

```
Period (years) 1
Final sum1284
NameBoris Johnson
Money 1200
int. rate % 7
Period (years) 0
Final sum1200
NameBoris Johnson
Money 1200
int. rate % 7
Period (years) 0
Final sum1200
NameBoris Johnson
Money 1200
int. rate % 7
Period (years) 1
Final sum1284
NameBoris Johnson
Money 200
int. rate % 7
Period (years) 1
Final sum214
NameBoris Johnson
```

```
Money 0
int. rate % 7
Period (years) 1
Final sum0
NameBoris Johnson
Money 1000
int. rate % 7
Period (years) 1
Final sum1070
NameJohn Snow
Money 1100
int. rate % 8
Period (years) 5
Final sum1616.26
NameHarry Potter
Money 2100
int. rate % 8
Period (years) 3
Final sum2645.4
1029.13
```

Принципи перевантаження операцій

У більшості випадків мова C++ дозволяє вибирати самостійно спосіб перевантаження операторів.

Але при роботі з бінарними операторами, які не змінюють лівий операнд (наприклад, operator+()), зазвичай використовується перевантаження через звичайну або дружню функцію, оскільки таке перевантаження працює для всіх типів даних параметрів (навіть якщо лівий операнд не є об'єктом класу або є об'єктом класу, який змінити не можна). Перевантаження через звичайну/дружню функцію має додаткову перевагу «симетрії», так як усі операнди стають явними параметрами (а не як у перевантаженні через метод класу, коли лівий операнд стає неявним об'єктом, на який вказує вказівник *this).

При роботі з бінарними операторами, які змінюють лівий операнд (наприклад, operator+=()), зазвичай використовується перевантаження через методи класу. У цих випадках лівим операндом завжди є об'єкт класу, на який вказує прихований вказівник *this.

Унарні оператори зазвичай теж перевантажуються через методи класу, так як у такому випадку параметри не використовуються взагалі.

Тому:

Для операторів присвоювання (=), індексу ([]), виклику функції (()) або вибору члена (->) використовуйте перевантаження через методи класу.

Для унарних операторів використовуйте перевантаження через методи класу.

Для перевантаження бінарних операторів, які змінюють лівий операнд (наприклад, operator+=()) використовуйте перевантаження через методи класу, якщо це можливо.

Для перевантаження бінарних операторів, які не змінюють лівий операнд (наприклад, operator+()) використовуйте перевантаження через звичайні/дружні функції.

МОДУЛЬ 4. НАСЛІДУВАННЯ. ВІРТУАЛЬНІ ФУНКЦІЇ І ПОЛІМОРФІЗМ

Тема 8. Наслідування

Базові класи та похідні класи. Специфікатор доступу protected

У мові C++ є третій специфікатор доступу, про який ми ще не говорили, тому що він корисний тільки в контексті спадкування. **Специфікатор доступу protected відкриває доступ до членів класу дружнім і дочірнім класам**. Доступ до protected-члену поза тілом класу закритий.

```
class Parent
{
public:
   int m_public; // доступ до цього члена відкритий для всіх об’єктів
private:
   int m_private; // доступ до цього члена відкритий тільки для інших членів класу Parent і для дружніх класів/функцій (але не для дочірніх класів)
protected:
   int m_protected; // доступ до цього члена відкритий для інших членів класу Parent, дружніх класів/функцій, дочірніх класів
};
```

```
class Child: public Parent
{
public:
   Child()
   {
      m_public = 1; // дозволено: доступ до відкритих членів батьківського класу з дочірнього класу
      m_private = 2; // заборонено: доступ до закритих членів батьківського класу з дочірнього класу
      m_protected = 3; // дозволено: доступ до захищених членів батьківського класу з дочірнього класу
   }
};

int main()
{
   Parent parent;
   parent.m_public = 1; // дозволено: доступ до відкритих членів класу ззовні
   parent.m_private = 2; // заборонено: доступ до закритих членів класу ззовні
   parent.m_protected = 3; // заборонено: доступ до захищених членів класу ззовні
}
```

У прикладі, наведеному вище, ви можете бачити, що член m_protected класу Parent напряму доступний дочірньому класу Child, але доступ до нього для членів ззовні — закритий.

Коли слід використовувати специфікатор доступу protected?

До protected-членів батьківського класу доступ відкритий для членів дочірнього класу, а це означає, що якщо ви пізніше зміните що-небудь в protected-члені (тип даних, значення), то вам доведеться внести зміни як до батьківського, так і в усі дочірні класи. Тому використання специфікатора доступу protected найбільш корисне, коли ви наслідуєте тільки свої класи і кількість дочірніх класів невелика. Отже, якщо ви внесете зміни в реалізацію батьківського класу, і вам знадобиться оновити всі дочірні класи, ви зможете зробити ці оновлення самі і це не займе багато часу (тому що дочірніх класів буде небагато).

Створення private-членів надає кращу інкапсуляцію і ізолює батьківські класи від змін, викликаних дочірніми класами. Але ціна цьому – додаткове створення відкритого або захищеного інтерфейсу (способу взаємодії інших об'єктів із класами і їх членами, тобто геттери і сеттери. Це додаткова робота, яка не варта того, якщо ви самі працюєте зі своїми ж класами (чужі класи не звертаються до вашого класу) і кількість дочірніх класів невелика.

Типи спадкувань. Доступ до членів

Існує три типи спадкування класів:
public;
private;
protected.

Для визначення типу спадкування потрібно просто вказати потрібне ключове слово біля успадкованого класу:

```
// Відкрите спадкування
class Pub: public Parent
{
};

// Закрите спадкування
class Pri: private Parent
{
};

// Захищене спадкування
class Pro: protected Parent
{
};

class Def: Parent // за замовчуванням мова C++ встановлює закрите спадкування
{
};
```

Якщо ви самі не визначили тип спадкування, то в мові C++ за замовчуванням буде обрано тип спадкування private (аналогічно і для членів класу, які за замовчуванням є private, якщо не вказано інше).

Це дає нам 9 комбінацій: 3 специфікатори доступу (public, private і protected) і 3 типи спадкування (public, private і protected).

Так в чому ж різниця між ними? Якщо коротко, то при спадкуванні специфікатор доступу члена батьківського класу може бути змінений в дочірньому класі (залежно від типу спадкування). Іншими словами, члени, які були public або protected в батьківському класі, можуть стати private в дочірньому класі.

Це може здатися трохи заплутаним, але все не так уже й погано. Ми зараз з усім цим розберемося, але перед цим згадаємо **наступні правила**:

- клас завжди має доступ до своїх (не успадкованих) членів;
- доступ до члену класу базується на його специфікаторі доступу;
- дочірній клас має доступ до успадкованих членів батьківського класу на основі специфікатора доступу цих членів в батьківському класі.

Спадкування типу public

Відкрите спадкування є одним з найбільш використовуваних типів спадкування. Дуже рідко ви побачите або будете використовувати інші типи. На щастя, відкрите спадкування є найлегшим і найпростішим з усіх типів. Коли ви відкрито наслідуєте батьківський клас, то успадковані public-члени залишаються public, успадковані protected-члени залишаються protected, а успадковані private-члени залишаються недоступними для дочірнього класу. Нічого не змінюється.

Правило: використовуйте відкрите спадкування, якщо у вас немає вагомих причин робити інакше.

Спадкування типу private

При закритому спадкуванні всі члени батьківського класу наслідуються як закриті. Це означає, що private-члени залишаються недоступними, а protected- і public-члени стають private у дочірньому класі.

Зверніть увагу, це не впливає на те, як дочірній клас отримує доступ до членів батьківського класу. Це впливає тільки на те, як іншими об’єктами здійснюється доступ до цих членів через дочірній клас:

```
class Parent
```

```
{
public:
    int m_public;
private:
    int m_private;
protected:
    int m_protected;
};

class Priv: private Parent // закрите спадкування
{
    // Закрите спадкування означає, що:
    // - public-члени стають private (m_public тепер private) в дочірньому класі;
    // - protected-члени стають private (m_protected тепер private) в дочірньому класі;
    // - private-члени залишаються недоступними (m_private недоступний) в дочірньому класі.
public:
    Priv()
    {
        m_public = 1; // дозволено: m_public тепер private в Priv
        m_private = 2; // заборонено: дочірні класи не мають доступу до закритих членів батьківського класу
        m_protected = 3; // дозволено: m_protected тепер private в Priv
    }
};

int main()
{
    Parent parent;
    parent.m_public = 1; // дозволено: m_public доступний ззовні через батьківський клас
    parent.m_private = 2; // заборонено: m_private недоступний ззовні через батьківський клас
    parent.m_protected = 3; // заборонено: m_protected недоступний ззовні через батьківський клас

    Priv priv;
```

```
    priv.m_public = 1; // заборонено: m_public недоступний ззовні через дочірній клас
    priv.m_private = 2; // заборонено: m_private недоступний ззовні через дочірній клас
    priv.m_protected = 3; // заборонено: m_protected недоступний ззовні через дочірній клас
}
```

Закрите спадкування може бути корисним, коли дочірній клас не має очевидного зв'язку з батьківським класом, але використовує його в своїй реалізації. У такому випадку ми не хочемо, щоб відкритий інтерфейс батьківського класу був доступний через об'єкти дочірнього класу (як це було, коли ми використовували відкритий тип спадкування).

На практиці спадкування типу private використовується рідко.

Спадкування типу protected

Цей тип спадкування майже ніколи не використовується, за винятком особливих випадків. Із захищеним спадкуванням, public- і protected-члени стають protected, а private-члени залишаються недоступними.

Оскільки цей тип спадкування дуже рідко використовується, то ми пропустимо приклад на практиці.

Приклад

```
class Parent
{
public:
    int m_public;
private:
    int m_private;
protected:
    int m_protected;
};
```

Клас Parent може звертатися до своїх членів безперешкодно. Доступ до m_public відкритий для всіх. Дочірні класи можуть звертатися як до m_public, так і до m_protected:

```
class D2 : private Parent  // закрите спадкування
{
    // Закрите спадкування означає, що:
    // - public-члени стають private в дочірньому класі;
```

```
        // - protected-члени стають private в дочірньому класі;
        // - private-члени недоступні для дочірнього класу.
    public:
        int m_public2;
    private:
        int m_private2;
    protected:
        int m_protected2;
    };
```

Клас D2 може безперешкодно звертатися до своїх членів. D2 має доступ до членів m_public і m_protected класу Parent, але не до m_private. Оскільки D2 наслідує клас Parent закрито, то m_public і m_protected тепер стають закритими при доступі через D2. Це означає, що інші об'єкти не зможуть отримати доступ до цих членів через використання об'єкта D2, а також будь-які інші класи, які будуть дочірніми класу D2, не матимуть доступу до цих членів:

```
    class D3 : public D2
    {
        // Відкрите спадкування означає, що:
        // - успадковані public-члени залишаються public в дочірньому класі;
        // - успадковані protected-члени залишаються protected в дочірньому класі;
        // - успадковані private-члени залишаються недоступними в дочірньому класі.
    public:
        int m_public3;
    private:
        int m_private3;
    protected:
        int m_protected3;
    };
```

Клас D3 може безперешкодно звертатися до своїх членів. D3 має доступ до членів m_public2 і m_protected2 класу D2, але не до m_private2. Оскільки D3 наслідує D2 відкрито, то m_public2 і m_protected2 зберігають свої специфікатори доступу і залишаються public і protected при доступі через D3. D3 не має доступу до m_private класу Parent. Він також не має доступу до m_protected або m_public класу Parent, обидва з яких стали закритими, коли D2 успадкував їх.

Приклад створення класу. Успадкування

```
#include <iostream>
#include <cstdlib>
#include <string>
using namespace std;
class MyMoney {
  public:
 string name;
double money;
double rate;
int time;
   double getMoney () {
      double S = money;
      for(int k = 1; k <= time; k++){
         S *= (1 + rate/100);
      }
      return S;
      }
   void ShowAll() {
      cout << "Name" << name << endl;
      cout << "Money " << money << endl;
      cout << "int. rate % " << rate << endl;
      cout << "Period (years) " << time << endl;
      cout << "Final sum" << getMoney() << endl;
   }
   void SetAll (string n, double m, double r, int t) {
      name = n;
      money = m;
      rate = r;
      time = t;
   }
   MyMoney (string n, double m, double r, int t) {
      SetAll (n, m, r, t); }
   MyMoney () {
      SetAll (" ", 0, 0, 0);
         }
   };
class BigMoney: public MyMoney {
public:
```

```
    int periods;
    double getMoney () {
       double S = money;
       for (int k = 1; k <= time * periods; k++) {
          S *= (1 + rate/100/periods);
       }
       return S;
    }
    void ShowAll () {
       cout << "Name" << name << endl;
       cout << " Money " << money << endl;
       cout << " int. rate % " << rate << endl;
       cout << "Period (years) " << time << endl;
       cout << "Money per year " << periods << endl;
       cout << "Final sum" << getMoney() << endl;}

       void SetAll (string n, double m, double r, int t, int p) {
          MyMoney::SetAll (n, m, r, t);
          periods = p; }

       BigMoney (string n, double m, double r, int t, int p = 1) :
       MyMoney(n, m, r, t)
       { periods = p; }
       BigMoney () : MyMoney () {
    periods = 1;
} };

    int main () {
    MyMoney objA ("Em.Nastia", 200, 3, 2);
BigMoney objB ( "John Snow", 1200, 8, 5);
       BigMoney  objC ( "Harry Potter", 1200, 6, 9);
       BigMoney objD;
       objD.SetAll ( "Kip Polok", 100, 8, 9, 4);
             objA.ShowAll ();
       cout << endl;
       objB.ShowAll ();
       cout << endl;
       objC.ShowAll ();
       cout << endl;
```

```
        objD.ShowAll ();
        cout << endl;
        return 0; }
```

Результат роботи:

```
NameEm.Nastia
Money 200
int. rate % 3
Period (years) 2
Final sum212.18

NameJohn Snow
 Money 1200
 int. rate % 8
Period (years) 5
Money per year 1
Final sum1763.19

NameHarry Potter
 Money 1200
 int. rate % 6
Period (years) 9
Money per year 1
Final sum2027.37

NameKip Polok
 Money 100
 int. rate % 8
Period (years) 9
Money per year 4
Final sum203.989
```

C++ починається тоді, коли ми переходимо до класів (class), а значить переходимо до об'єктно-орієнтованого програмування (ООП), а відповідно до основних його постулатів:

- Інкапсуляція
- Наслідування
- Поліморфізм

Клас є просто представленням типу об'єкта; його можна представити як план (креслення), що описує об'єкт.

Подібно до того, як один план (креслення) може бути використаний для побудови декількох будівель, окремий клас може бути використаний для створення необхідної кількості об'єктів.

Інкапсуляція

Інкапсуляція (encapsulation) – це механізм, який об'єднує дані з кодом, що обробляє ці дані, а також захищає і те, і інше від зовнішнього втручання або неправильного використання. В об'єктно-орієнтованому програмуванні код і дані можуть бути об'єднані разом; у цьому випадку говорять, що створюється так званий «чорний ящик». Коли коди і дані об'єднуються таким способом, створюється об'єкт (object). Іншими словами, об'єкт – це те, що підтримує інкапсуляцію.

Засіб реалізації інкапсуляції в C++ це class.

Одне з визначень класу: клас – це механізм, який об'єднує дані з кодом, який обробляє ці дані. Захист даних виконується за допомогою специфікаторів доступу public, protected, private.

Приклад класу, який зберігає дані цілого типу

```
#include <iostream>
usind namespace std;
class Demo
{
int data;
public:
int getData() const
{return data;}
void setData( int d)
{data=d;}
};
int main()
{
Demo d;
d.setData(45);
cout<<d.getData()<<endl;
cin.get();
return 0;
```

Наслідування

Наслідування (inheritance) – це процес, за допомогою якого один об'єкт може набувати властивостей іншого з можливістю розширити або перевизначити властивості базового об'єкта.

Поліморфізм

Поліморфізм (від грецького polymorphos) – 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.

У більш загальному сенсі, концепцією поліморфізму є ідея «один інтерфейс, безліч реалізацій методів». Це означає, що можна створити загальний інтерфейс для групи близьких за змістом дій.

Наприклад, навчившись керувати одним легковим автомобілем (тобто Ви освоїли інтерфейс автомобіля), Ви можете керувати також і іншими легковими автомобілями. І Вас «не цікавить», що, наприклад, коробка передач чи гальма у різних автомобілях технічно реалізовано по-різному, Ви просто їх використовуєте.

Приклад наслідування. Поліморфізму

```
#include <iostream>
using namespace std;
class Point
{
int x,y;
public:
int getX() const {return x;}
int getY() const {return y;}
void setXY( int _x, int _y) {x=_x; y=_y;}
virtual void print() const {std::cout<< "x="<<x<<" y="<<y;}
};
class ColorPoint : public Point
{
int color;
```

```
    public:
    int getC() const {return color;}
    void setC(int c)  {color =c;}
    virtual void print() const {Point::print(); std::cout<<" color="<<color;} // тут поліморфізм
    };
    void printPoint (Point& point) { point.print(); std::cout<<std::endl;}

    int main()
    {
    Point p1;
    p1.setXY(5,6);
    p1.print();
    cout<<std::endl;
    ColorPoint p2;
    p2.setXY(5,6);
    p2.setC(256);
    p2.print();
    cout<<std::endl;
    cout<<"\nCall function printPoint:\n"<<endl;
    printPoint(p1);
    printPoint(p2);
    return 0;
    }
```

Результат роботи:

```
x=5 y=6
x=5 y=6 color=256

Call function printPoint:

x=5 y=6
x=5 y=6 color=256
```

Шаблонний клас StackList, реалізує стек як однозв'язковий список. За бажанням до тексту класу можна додати інші функції, які розширять його можливості.

У класі оголошуються такі поля і методи:

- покажчик pTop – вершина стека;
- конструктор за замовчуванням;

- конструктор копіювання StackList (const StackList &);
- деструктор;
- метод Push(), який розміщує елемент у стек;
- метод Pop() – добувальний елемент зі стека;
- метод Count() – повертає кількість елементів у стеку;
- метод Empty() – видаляє всі елементи зі стека;
- оператор копіювання operator=(const StackList&) – необхідний реалізації коректного привласнення об’єктів класу StackList;
- метод Print(), який виводить вміст стека на екран.

Повний текст класу StackList та структури NodeStack

```
#include <iostream>
using namespace std;

// Структура, що описує вузол
template <typename T>
struct NodeStack
{
  T item;
  NodeStack<T>* next;
};

// Шаблонний клас Стек з урахуванням однозв'язкового списку
template <typename T>
class StackList
{
private:
  NodeStack<T>* pTop; // покажчик на вершину стека

public:
  // конструктор за замовчуванням
  StackList() { pTop = nullptr; }

  // конструктор копіювання
  StackList (const StackList & SL)
  {
    NodeStack<T>* p; // Додаткові покажчики
    NodeStack<T>* p2;
    NodeStack<T>* p3;
```

```
        // Ініціалізувати pTop
        pTop = nullptr;
        p3 = nullptr;

        p = SL.pTop; // покажчик p рухається за списком SL.pTop->...
        while (p != nullptr)
        {
          // 1. Сформувати вузол p2
          try {
            // Спроба виділення пам'яті
            p2 = new NodeStack<T>;
          }
          catch (bad_alloc e)
          {
            // якщо пам'ять не виділено, то вихід
            cout << e.what() << endl;
            return;
          }
          p2->item = p->item;
          p2->next = nullptr;

          // 2. pTop = pTop + p2
          if (pTop == nullptr) // створити чергу
          {
            pTop = p2;
            p3 = p2;
          }
          else
          {
            p3->next = p2;
            p3 = p3->next;
          }

          // 3. Перейти на наступний елемент
          p = p->next;
        }
      }
```

```
// Розмістити елемент у стек
// елемент міститься початку списку
void Push(T item)
{
  NodeStack<T>* p;

  // 1. Сформувати елемент
  try {
    p = new NodeStack<T>; // Спроба виділити пам'ять
  }
  catch(bad_alloc e)
  {
    // якщо пам'ять не виділилася, вихід
    cout << e.what() << endl;
    return;
  }
  p-> item = item;
  p->next = pTop; // p вказує на 1-й елемент

  // 2. Перенаправити pTop на p
  pTop = p;
}

// Кількість елементів у стеку
int Count()
{
  if (pTop == nullptr)
    return 0;
  else
  {
    NodeStack<T>* p = pTop;
    int count = 0;

    while (p != nullptr)
    {
      count++;
      p = p->next;
    }
  }
}
```

```
    // Очищає стек – видаляє всі елементи зі стеку
    void Empty()
    {
      NodeStack<T>* p; // Додатковий покажчик
      NodeStack<T>* p2;

      p = pTop;

      while (p != nullptr)
      {
        p2 = p; // зробити копію з p
        p = p->next; // Перейти на наступний елемент стека
        delete p2; // видалити пам’ять, виділену попереднього
елемента
      }
      pTop = nullptr; // виправити вершину стека
    }

    // оператор копіювання
    StackList<T>& operator=(const StackList<T>& LS)
    {
      // Чи є елементи у стеку?
      if (pTop != nullptr) Empty ();

      NodeStack<T>* p; // Додатковий покажчик
      NodeStack<T>* p2;
      NodeStack<T>* p3;

      // Ініціалізувати pTop
      pTop = nullptr;
      p3 = nullptr;

      p = LS.pTop; // покажчик p рухається за списком
SL.pTop->...
      while (p != nullptr)
      {
        // 1. Сформувати вузол p2
        try {
          // Спроба виділити пам’ять
```

```
        p2 = new NodeStack<T>;
      }
      catch (bad_alloc e)
      {
        // якщо пам'ять не виділено, то вихід
        cout << e.what() << endl;
        return *this;
      }
      p2->item = p->item;
      p2->next = nullptr;

      // 2. pTop = pTop + p2
      if (pTop == nullptr) // Створити стек
      {
        pTop = p2;
        p3 = p2;
      }
      else
      {
        p3->next = p2;
        p3 = p3->next;
      }

      // 3. Перейти на наступний елемент
      p = p->next;
    }
    return *this;
  }

  // виведення стека
  void Print(const char* objName)
  {
    cout << "Object:" << objName << endl;
    if (pTop == nullptr)
      cout << "stack is empty." << endl;
    else
    {
      NodeStack<T>* p; // Додатковий покажчик
      p = pTop;
      while (p != nullptr)
```

```
        {
          cout << p-> item << "\t";
          p = p->next;
        }
        cout << endl;
      }
    }

    // деструктор
    ~StackList()
    {
      Empty();
    }

    // метод, що витягує елемент зі стека
    T Pop()
    {
      // Перевірка, чи порожній стек?
      if (pTop == nullptr)
        return 0;

      NodeStack<T>* p2; // Додатковий покажчик
      T item;
      item = pTop-> item;

      // перенаправити покажчики pTop, p2
      p2 = pTop;
      pTop = pTop->next;

      // Звільнити пам'ять, виділену під 1-й елемент
      delete p2;

      // повернути item
      return item;
    }
};

int main()
{
  StackList<int> SL;
  SL.Print("SL");
```

```
    SL.Push(8);
    SL.Push(5);
    SL.Push(10);
    SL.Push(7);
    SL.Print("SL");

    // Перевірка конструктора копіювання
    StackList<int> SL2 = SL;
    SL2.Print("SL2");

    SL.Empty();
    SL.Print("SL");

    SL = SL2;
    SL.Print("SL = SL2");

    StackList<int> SL3;
    SL3.Push(1);
    SL3.Push(2);
    SL3.Push(3);
    SL3.Print("SL3");

    SL = SL2 = SL3;
    SL.Print("SL");
    SL2.Print("SL2");

    int d = SL2. Pop ();

    cout << "d = " << d << endl;
    SL2.Print("SL2-1");

    d = SL2. Pop ();
    SL2.Print("SL2-2");

    d = SL2. Pop ();
    SL2.Print("SL2-3");

    d = SL2. Pop ();
    cout << "d = " << d << endl;
    SL2.Print("SL2----");
  }
```

Результат роботи:

```
Object: SL
stack is empty.
Object: SL
7       10      5       8
Object: SL2
7       10      5       8
Object: SL
stack is empty.
Object: SL = SL2
7       10      5       8
Object: SL3
3       2       1
Object: SL
3       2       1
Object: SL2
3       2       1
d = 3
Object: SL2-1
2       1
Object: SL2-2
1
Object: SL2-3
stack is empty.
d = 0
Object: SL2----
stack is empty.
```

Виділення пам’яті для елемента стека

Виділення пам’яті для вузла виконується за допомогою оператора new. Однак можлива ситуація, що пам’ять не буде виділена. І тут буде згенеровано виняток типу bad_alloc. Тип bad_alloc є класом. У програмі обробки цієї ситуації використовується наступний блок try…catch.

```
try {
  p = New NodeStack<T>; // Спроба виділити пам’ять
}
catch (bad_alloc e)
```

```
{
  // якщо пам'ять не виділилася, вихід
  cout << e.what() << endl;
  return;
}
```

Очищення стека. Метод Empty()

Очищення стека реалізована у методі Empty(). Для очищення стека потрібно в циклі звільнити пам'ять кожного елемента стека як показано нижче

```
void Empty()
{
  NodeStack<T>* p; // Додаткові покажчики
  NodeStack<T>* p2;

  p = pTop;
  while (p! = nullptr)
  {
    p2 = p; // зробити копію з p
    p = p->next; // Перейти на наступний елемент стека
    delete p2; // Видалити пам'ять, виділену під попередній елемент
  }
  pTop = nullptr; // виправити вершину стека
}
```

Конструктор копіювання та оператор копіювання

Оскільки пам'ять у класі StackList виділяється динамічно, то в цьому класі обов'язково потрібно реалізовувати власний конструктор копіювання та оператор копіювання, щоб уникнути недоліків побітового копіювання. Оператор копіювання необхідний для правильного присвоєння об'єктів класу StackList.

Приклад створення шаблонного класу MATRIX. Клас реалізує матрицю розміром m*n. Пам'ять для матриці виділяється динамічно

За цим прикладом можна створювати і використовувати власні класи, які містять структури даних, у яких пам'ять виділяється динамічно.

У класі реалізовано:

− прихована (private) внутрішня змінна M типу «покажчик на покажчик». Ця змінна визначає матрицю. Пам'ять для матриці виділятиметься динамічно;

− дві цілочисленні внутрішні private змінні m, n. Ці змінні визначають розмірність матриці M;

− конструктор за замовчуванням (без параметрів);

− конструктор із двома параметрами MATRIX(int, int). Цей конструктор створює матрицю розміром m*n. У конструкторі виділяється пам'ять для стовпців і рядків матриці. Значення кожного елемента матриці встановлюється 0;

− конструктор копіювання MATRIX (MATRIX &). Цей конструктор необхідний для створення копії об'єкта-матриці з іншого об'єкта-матриці;

− методи читання/запису елементів матриці GetMij(), SetMij();

− метод Print(), який виводить матрицю;

− оператор копіювання operator=(MATRIX&). Цей оператор перевантажує оператор привласнення = і призначений для коректного копіювання об'єктів типу obj2=obj1;

− деструктор.

Приклади створення шаблонного класу Матриця із динамічним виділенням пам'яті

```
#include <iostream>
using namespace std;

// шаблонний клас Матриця
template <typename T>
class MATRIX
{
private:
  T** M; // матриця
  int m; // кількість рядків
  int n; // кількість стовпців

public:
  // конструктори
  MATRIX()
```

```
    {
      n = m = 0;
      M = nullptr; // не обов'язково
    }

    // конструктор із двома параметрами
    MATRIX(int _m, int _n)
    {
      m = _m;
      n = _n;

      // Виділити пам'ять для матриці
      // Виділити пам'ять для масиву покажчиків
      M = (T**) new T * [m]; // кількість рядків, кількість
покажчиків

      // Виділити пам'ять кожному покажчику
      for (int i = 0; i < m; i++)
        M [i] = (T*) new T[n];

      // Заповнити масив M нулями
      for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
          M[i][j] = 0;
    }

    // Конструктор копіювання – обов'язковий
    MATRIX(const MATRIX& _M)
    {
      // Створюється новий об'єкт якого бачиться пам'ять
      // Копіювання даних *this <= _M
      m = _M.m;
      n = _M.n;

      // Виділити пам'ять для M
      M = (T**) new T * [m]; // кількість рядків, кількість
покажчиків

      for (int i = 0; i < m; i++)
        M[i] = (T*) new T[n];
```

```
  // Заповнити значеннями
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      M[i][j] = _M.M[i][j];
}

// методи доступу
T GetMij(int i, int j)
{
  if ((m > 0) && (n > 0))
    return M[i][j];
  else
    return 0;
}

void SetMij(int i, int j, T value)
{
  if ((i < 0) || (i >= m))
    return;
  if ((j < 0) || (j >= n))
    return;
  M[i][j] = value;
}

// метод, що виводить матрицю
void Print(const char* ObjName)
{
  cout << "Object:" << ObjName << endl;
  for (int i = 0; i < m; i++)
  {
    for (int j = 0; j < n; j++)
      cout << M[i][j] << "\t";
    cout << endl;
  }
  cout << "----------------------" << endl << endl;
}

// Оператор копіювання – обов'язковий
MATRIX operator=(const MATRIX& _M)
```

```
    {
      if (n > 0)
      {
        // звільнити пам’ять, виділену раніше для об’єкта *this
        for (int i = 0; i < m; i++)
          delete[] M[i];
      }

      if (m > 0)
      {
        delete [] M;
      }

      // Копіювання даних M <= _M
      m = _M.m;
      n = _M.n;

      // Виділити пам’ять для M знову
      M = (T**) new T * [m]; // кількість рядків, кількість покажчиків
      for (int i = 0; i < m; i++)
        M[i] = (T*) new T[n];

      // Заповнити значеннями
      for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
          M[i][j] = _M.M[i][j];
      return *this;
    }

    // Деструктор – звільняє пам’ять, виділену для матриці
    ~MATRIX()
    {
      if (n > 0)
      {
        // звільнити виділену пам’ять кожному за рядки
        for (int i = 0; i < m; i++)
          delete[] M[i];
      }
```

```
    if (m > 0)
      delete [] M;
  }
};

int main()
{
  // Тест для класу MATRIX
  MATRIX<int> M(2, 3);
  M.Print("M");

  // Заповнити матрицю значеннями за формулою
  int i, j;
  for (i = 0; i < 2; i++)
    for (j = 0; j < 3; j++)
      M.SetMij(i, j, i + j);

  M.Print("M");

  MATRIX<int> M2 = M; // виклик конструктора копіювання
  M2.Print("M2");

  MATRIX<int> M3; // Виклик оператора копіювання – перевірка
  M3 = M;
  M3.Print("M3");

  MATRIX<int> M4;
  M4 = M3 = M2 = M; // Виклик оператора копіювання у вигляді "ланцюжка"
  M4.Print("M4");
}
```

Результат роботи:

```
Object:M
0    1    2
1    2    3
```

```
--------------------

Object:M2
0      1      2
1      2      3
--------------------

Object:M3
0      1      2
1      2      3
--------------------

Object:M4
0      1      2
1      2      3
--------------------
```

Завдання для самостійної роботи

1. Створити клас для роботи з бази даних відповідно до наданих варіантів із вказаними полями. Утворити похідний клас, залучивши до нього як мінімум два додаткових поля так, щоб клас набув більшої спеціалізованості. Для другого класу використати конструктор, аби він містив усі аргументи, необхідні для ініціалізації об'єкта похідного класу. Створити необхідні функції, що дозволяють виводити інформацію на екран і можливість додавати та знищувати записи.

№	Предметна область БД	Поля бази даних
1.	«Бібліотека»	Інвентарний номер, автор, назва, кількість сторінок, рік видання.
2.	«Телефонний довідник»	Прізвище, ім'я, по батькові, домашня адреса, телефон.
3.	«Розклад руху літаків»	Номер рейсу, тип літака, напрямок руху, періодичність вильоту.
4.	«Колекція CD-дисків»	Інвентарний номер, назва, об'єм диску, тип, дата запису.

5.	«Записна книжка»	Прізвище, ім’я, по батькові, домашня адреса, телефон, електронна пошта.
6.	«Склад товарів»	Інвентарний номер, назва товару, вага, ціна, кількість.
7.	«Користувачі локальної мережі»	Прізвище, ім’я, по батькові, група, обліковий запис, тип облікового запису.
8.	«Предметний покажчик»	Слово; номера сторінок, де це слово зустрічається.
9.	«Банківські рахунки»	Прізвище, ім’я, дата останньої операції, сума останньої операції, сума вкладу.
10.	«Успішність студентів»	Прізвище, ім’я, номер групи, оцінки з трьох предметів.

2. Спроектувати ієрархію класів для представлення графічних об’єктів. Головним базовим класом для всіх об’єктів є клас Point – точка на площині (у просторі) з її координатами. Опис класів слід розмістити у заголовочному файлі, а визначення функцій і головну функцію програми – у двох окремих файлах. Передбачити методи для створення об’єкта, його переміщення на екрані, зміни розмірів і кольору. Використати захищення даних для ізоляції елементів-даних класу від підпрограм, у яких цей клас використовується, а також поліморфізм для визначення дії певних функцій у класовій ієрархії. Напишіть головну функцію, що демонструє роботу з цим класом. Програма повинна містити меню, що дозволяє здійснити перевірку всіх методів класу.

№	**Варіант**
1.	Трапеція
2.	Рівнобічний трикутник
3.	Ромб
4.	Прямокутник
5.	Ламана лінія
6.	Прямокутний трикутник
7.	Шестикутник
8.	Коло
9.	Відрізок на площині
10.	Квадрат

Тема 9. Віртуальні функції і поліморфізм

Віртуальна функція

Перед вивченням поняття абстрактного класу слід ознайомитись із поняттям «суто віртуальна функція».

Як відомо, використання віртуальних функцій має сенс у тому разі, якщо класи утворюють ієрархію успадкування. Якщо немає ієрархії класів, механізм віртуальних функцій не працює. Ієрархії класів будуються за принципом від загального до конкретного. Це означає, що у верхніх рівнях ієрархій оголошуються класи, що визначають щось спільне. Із просуванням до нижніх рівнів ієрархії це загальне деталізується (спеціалізується). Класи нижніх рівнів містять конкретні реалізації.

Отже, часто програмний код віртуальних функцій у вершині ієрархії класів не має ніякого значення, оскільки він деталізується в класах нижніх рівнів. Має значення лише організація зв'язку між класами нижніх рівнів. І тут віртуальні функції, оголошені у вершині ієрархій класів, грають лише сполучну роль. Отже, виникає потреба у оголошенні «порожніх» функцій, які містять код.

Якщо розглянути ієрархію класів, то багато з них містять віртуальні функції, що не мають коду або не мають тіла функції. Іноді класи верхніх рівнів ієрархій містять лише порожні віртуальні функції.

У мові C++ віртуальна функція без коду називається суто віртуальна функція (pure virtual function).

У синтаксисі мови C++ можна оголосити суто віртуальну функцію. Суто віртуальна функція – це функція, яку:

оголошено у класі зі специфікатором virtual;

оголошено з використанням спеціального синтаксису C++, який визначає функцію як таку, що не має блоку фігурних дужок { }, що містить програмний код.

У найбільш загальному випадку оголошення суто віртуальної функції має вигляд:

```
class ClassName
{
  virtual return_type FuncNamePure(list_of_parameters) = 0;
};
```

де
FuncNamePure() – ім’я суто віртуальної функції;
return_type – тип, що повертається функцією;
list_of_parameters – перелік параметрів функції.

Важливо розуміти, що наведене вище оголошення суто віртуальної функції не слід плутати з оголошенням віртуальної функції, що містить порожній блок коду

```
// Це не є суто віртуальна функція
virtual return_type FuncName(list_of_parameters)
{}
```

Щоб зробити функцію віртуальною, потрібно просто вказати **ключове слово virtual** перед оголошенням функції. Наприклад:

Приклад створення віртуальної функції

```
#include <iostream>
#include <cstdlib>
using namespace std;

class A
{
public:
   virtual const char* getName() { return "A"; }
};

class B: public A
{
public:
   virtual const char* getName() { return "B"; }
};

class C: public B
{
public:
   virtual const char* getName() { return "C"; }
};

class D: public C
```

```
{
public:
   virtual const char* getName() { return "D"; }
};

int main()
{
   C c;
   A &rParent = c;
   std::cout << "rParent is a " << rParent.getName() << '\n';

   return 0;
}
```

Результат роботи:

```
rParent is a C
```

Тут спочатку створюється об’єкт із класу C.

rParent – це посилання класу A, якому вказуємо посилатися на частину A об’єкта C.

Потім викликається метод rParent.getName().

Виклик rParent.GetName() приводить до виклику A::getName(). Однак, оскільки A::getName() є віртуальною функцією, то компілятор шукає «найдочірніший» метод між A і C. У цьому випадку – це C::getName().

Зверніть увагу, компілятор не викликатиме D::getName(), оскільки наш вихідний об’єкт був класу C, а не класу D, тому розглядаються методи тільки між класами A і C.

Типи повернення віртуальних функцій

Типи повернення віртуальної функції і її перевизначень повинні збігатися. Розглянемо приклад:

```
class Parent
{
public:
   virtual int getValue() { return 7; }
};

class Child: public Parent
{
public:
   virtual double getValue() { return 9.68; }
};
```

В цьому випадку Child::getValue() не вважається підходящим перевизначенням для Parent::getValue(), тому що типи повернень різні (метод Child::getValue() вважається повністю окремою функцією).

Увага! Ніколи не викликайте віртуальні функції в тілі конструкторів або деструкторів.

Під час створення об'єкта класу Child спочатку створюється батьківська частина цього об'єкта, а потім вже дочірня Якщо ви викликатимете віртуальну функцію з конструктора класу Parent при тому, що дочірня частина створюваного об'єкта ще не була створена, то викликати дочірній метод замість батьківського буде неможливо, тому що об'єкт child для роботи з методом класу Child ще не буде створений. У таких випадках у мові C++ викликатиметься батьківська версія методу.

Аналогічна проблема існує і з деструкторами. Якщо ви викликаєте віртуальну функцію в тілі деструктора класу Parent, то завжди викликатиметься метод класу Parent, тому що дочірня частина об'єкта вже буде знищена.

Недолік віртуальних функцій

Обробка і виконання виклику віртуального методу займає більше часу, ніж обробка і виконання виклику звичайного методу. Крім того, компілятор також повинен виділяти один додатковий вказівник для кожного об'єкта класу, який має одну або кілька віртуальних функцій.

Абстрактний клас. Особливості оголошення і використання

Якщо класи утворюють ієрархію, класи у верхній частині ієрархії зазвичай містять одну або кілька віртуальних функцій. Ці функції використовуються з метою узгодження взаємодії інших класів, розміщених на нижніх рівнях ієрархії. Найчастіше класи верхніх рівнів ієрархії містять оголошення всіх загальних чи суто віртуальних функцій, даючи можливість використовувати поліморфізм C++ визначення функцій нижчих класів ієрархії визначаючих конкретні реалізації.

Бувають випадки, коли класи верхніх рівнів ієрархії містять лише суто віртуальні функції і більше ніяких. Таким чином, клас, що містить загальні або суто віртуальні функції, нічого не робить, він є лише сполучною ланкою для забезпечення полі-

морфізму. Отже, немає сенсу створювати екземпляр такого класу, оскільки об'єкт цього класу не потрібен. Цей об'єкт займає місце в пам'яті, але не виконує жодної роботи. У C++ є можливість обмежити використання об'єктів «порожніх» класів шляхом оголошення їх абстрактними.

Абстрактний клас – це клас, що містить хоча б одну суто віртуальну функцію. Компілятор цей клас визначає по особливому. Створити об'єкт абстрактного класу не вдасться. Спроба створення такого об'єкта буде розпізнана компілятором як помилка. Це логічно, навіщо виділяти пам'ять під елемент, який не визначає код, який надає послуги.

Загальний синтаксис оголошення абстрактного класу, що містить лише одну суто віртуальну функцію, має вигляд.

```
class ClassName
{
  virtual return_type Func(list_of_parameters) = 0;

  // Інші елементи класу
  // ...
}
```

де

ClassName – ім'я класу, що визначає компілятор як абстрактний на основі наявності в ньому суто віртуальної функції з ім'ям Func();

Func – ім'я суто віртуальної функції;

return_type – тип, який повертає суто віртуальна функція;

list_of_parameters – перелік параметрів.

Спроба оголосити екземпляр класу ClassName призведе до помилки на етапі компіляції

ClassName obj; // помилка компіляції, заборонено

Проте допускається оголошувати покажчик чи посилання на абстрактний клас

ClassName* p; // оголошення вказівника на абстрактний клас ClassName

ClassName&ref=obj; // оголошення посилання на абстрактний клас ClassName

У разі оголошення посилання на абстрактний клас ClassName, це посилання має бути ініціалізоване одразу зна-

ченням адреси екземпляра будь-якого неабстрактного класу, успадкованого від класу ClassName.

У прикладі оголошуються три класи, що утворюють ієрархію:

− базовий абстрактний клас A. Цей клас містить суто віртуальну функцію Func(). Примірник цього класу створити неможливо;

− не абстрактний клас B, успадкований від класу A. У класі B оголошується віртуальна функція Func(), яка перевизначає однойменну функцію базового класу A;

− не абстрактний клас C, успадкований від класу B. У класі C визначено віртуальну функцію під назвою Func(). Ця функція визначає функцію Func() класу B.

Приклади створення абстрактних класів

```
#include <iostream>
using namespace std;

// Це абстрактний клас, екземпляр (об'єкт) цього класу створити не вдасться
class A
{
public:
  // суто віртуальна функція
  virtual void Func() = 0;
};

// Клас, успадкований від класу A
class B: public A
{
public:
  // Перевизначити суто віртуальну функцію – обов'язково (вимоги компілятора)
  void Func() override
  {
    cout << "B::Func()" << endl;
  }
};

// Клас, успадкований від класу B
```

```
class C: public B
{
public:
  // Перевизначити суто віртуальну функцію
  void Func() override
  {
    cout << "C::Func()" << endl;
  }
};

int main()
{
  // 1. Спроба створення екземпляра абстрактного класу A
  // A objA; – заборонено, помилка компіляції

  //2. Спроба створення екземплярів похідних класів B, C
  B objB; // можна, оскільки клас B не абстрактний
  C objC; // можна, оскільки клас C не абстрактний

  // 3. Оголосити покажчик абстрактний клас A
  A* pA; // дозволено

  // 3.1. Перемістити покажчик pA на екземпляр похідного класу C
  pA = & objC;
  pA->Func(); // C::Func() – пізнє зв'язування, поліморфізм

  // 3.2. Перемістити вказівник pA на екземпляр похідного класу B
  pA = &objB;
  pA->Func(); // B::Func() – поліморфізм у дії

  // 4. Оголосити посилання refA на абстрактний клас A та ініціалізувати його
  // значенням екземпляра похідного класу B
  A&refA = objB;
  refA.Func(); // B::Func() – поліморфізм у дії
}
```

Результат роботи:

```
C::Func()
B::Func()
B::Func()
```

Приклад демонстрації використання абстрактного класу під час реалізації успадкування: Figures=>Circle=>CircleColor

У прикладі показано фрагмент ієрархії побудови геометричних фігур. У вершині ієрархії лежить абстрактний клас Figures. Цей клас використовується для забезпечення правильної роботи механізму віртуальних функцій. Клас Figures визначає геометричну фігуру загалом. Більш конкретна фігура (коло, прямокутник, трикутник тощо) може бути визначена в успадкованих класах. Відповідно, клас Figures містить оголошення віртуальної функції Area(), яка призначена для отримання площі фігури і повинна бути перевизначена в успадкованих класах. Оскільки на цьому рівні ієрархії (у цьому класі) ще невідомо площу саме якої фігури потрібно обчислювати (площу кола, площу прямокутника тощо), то функція Area() має загальний характер і визначена як суто віртуальна функція. На основі визначення функції Area() як суто віртуальної, клас Figure буде розпізнавати компілятор як абстрактний клас.

У майбутньому з класу Figures можна успадковувати класи інших геометричних фігур (Triangle – трикутник, Rectangle – прямокутник тощо). У цих класах метод Area() буде також віртуальним, але він матиме конкретний код обчислення площі, на відміну від Area() класу Figures.

Приклад демонстрації використання абстрактного класу під час реалізації успадкування: Figures=>Circle=>CircleColor

```
#include <iostream>
using namespace std;

// C++. Анотація класу Figure
class Figure
{
  // 1. Внутрішні поля класу
private:
  string name; // Назва фігури

public:
  // 2. Конструктор
  Figure(string _name) :name(_name)
  {
  }
```

```
  // 3. Методи доступу (гетери, сетери)
  string GetName()
  {
    return name;
  }

  void SetName(string name)
  {
    this->name = name;
  }

  // 4. Чисто віртуальна функція Area() – площа постаті
  virtual double Area() = 0;
};

// Клас, що реалізує коло.
// Похідний від класу Figure – розширює клас Figure
class Circle: public Figure
{
  // 1. Внутрішні приховані поля класу
private:
  double x, y; // Координати центру кола
  double r; // Радіус кола

public:
  // 2. Конструктор класу
  Circle(double x, double y, double r) :Figure("Circle")
  {
    this->x = x;
    this->y = y;
    if (r > 0) this-> r = r;
      else
    this->r = 1.0;
  }

  // 3. Методи доступу
  void GetXYR(double& x, double& y, double& r)
  {
    x = this->x;
    y = this->y;
    r = this->r;
  }
```

```
  void SetXYR(double x, double y, double r)
  {
    this->x = x;
    this->y = y;

    // відкоригувати радіус якщо потрібно
    if (r > 0)
      this->r = r;
    else
      this->r = 1.0;
  }

  // 4. Віртуальна функція Area() - площа постаті
  double Area() override
  {
    const double Pi = 3.141592;
    return Pi*r*r;
  }
};

// Клас, що розширює можливості класу Circle – додає колір кола.
class CircleColor : public Circle
{
  // 1. Внутрішні змінні класу
private:
  unsigned int color = 0; // колір кола

public:
  // 2. Конструктор
  CircleColor(double x, double y, double r, unsigned int color) :Circle(x, y, r)
  {
    this->color = color;
  }

  // 3. Методи доступу
  unsigned int GetColor()
  {
    return color;
  }
```

```
  void SetColor(unsigned int color)
  {
    this->color = color;
  }
};

int main()
{
  // 1. Клас Circle
  // 1.1. Створити екземпляр класу Circle
  Circle cr(1, 3, 2);

  // 1.2. Викликати метод Area() класу Circle
  double area = cr.Area();
  cout << "The instance of class Color:" << endl;
  cout << "area = " << area << endl;

  // 1.3. Взяти значення полів екземпляра cr
  double x, y, r;
  cr.GetXYR(x, y, r);

  // 1.4. Вивести значення полів на екран
  cout << "x = " << x << ", y = " << y << ", radius = " << r << endl;

  // 2. Примірник класу CircleColor
  // 2.1. Створити екземпляр класу CircleColor
  CircleColor crCol(2, 4, 6, 1);

  // 2.2. Отримати значення полів екземпляра crCol
  // 2.2.1. // взяти значення кольору з методу класу CircleColor
  unsigned int col = crCol.GetColor();

  // 2.2.2. Взяти значення x, y, r із методу базового класу
  crCol.GetXYR(x, y, r);

  // 2.3. Вивести отримані значення на екран
  cout << "The instance of class ColorCircle: " << endl;
  cout << "color = " << col << endl;
  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "r = " << r << endl;
```

```
  // 2.4. Викликати метод Area() базового класу
  cout << "area = " << crCol.Area() << endl;
}
```
Результат роботи:
```
The instance of class Color:
area = 12.5664
x = 1, y = 3, radius = 2
The instance of class ColorCircle:
color = 1
x = 2
y = 4
r = 6
area = 113.097
```

Спадкування абстрактних класів

Абстрактні класи, що знаходяться на вершині ієрархії, використовуються переважно для потреб похідних класів. Будь-яка ієрархія класів може містити від однієї до кількох абстрактних класів. Клас, похідний від абстрактного може бути абстрактним. У такому разі такий клас просто перевизначає функцію базового абстрактного класу за прикладом нижче

```
// базовий абстрактний клас
class BaseClass
{
  // Чисто віртуальна функція
  virtual return_type Func(list_of_parameters) = 0;
}

// Похідний абстрактний клас
class DerivedClass : BaseClass
{
 // Перевизначення суто віртуальної функції
  virtual return_type Func(list_of_parameters) override = 0;
}
```
де:

BaseClass – базовий абстрактний клас, що визначає суто віртуальну функцію Func();

DerivedClass – похідний абстрактний клас, який перевизначає суто віртуальну функцію Func();

Func() – суто віртуальна функція;
return_type – тип, що повертається суто віртуальною функцією Func();
list_of_parameters – перелік параметрів функції Func().

Крім того, при наслідуванні абстрактного класу будь-яка суто віртуальна функція цього класу має бути перевизначена у похідному класі. В іншому випадку компілятор видасть помилку.

Наприклад:

Якщо заданий абстрактний клас із суто віртуальною функцією

```
class BaseClass
{
  virtual void Func() = 0;
}
```

то в успадкованому (похідному) класі ця функція обов'язково має бути перевизначена. Розглядаються два можливі випадки.

Випадок 1. Віртуальна функція містить певний код (тіло функції). Тоді зразковий код успадкованого класу буде таким

```
class DerivedClass : BaseClass
{
  ...

  void Func() override
  {
    // обов'язково потрібно визначати override-функцію,
    // яка містить блок коду
    // ...
  }
}
```

Випадок 2. Віртуальну функцію реалізовано як суто віртуальну функцію (без тіла функції). У цьому випадку ця суто віртуальна функція має бути перевизначена у похідному класі в ієрархії.

```
class DerivedClass : BaseClass
{
  ...

  // Продовження ланцюжка суто віртуальних функцій
  virtual void Func() override = 0;
}
```

У наведеному вище коді можна опустити ключове слово virtual.

Клас vector

Клас vector – це динамічний масив, розмір якого у програмі може змінюватись за необхідності. Цей клас є одним із найбільш універсальних і поширених у використанні при написанні програм на C++. Доступ до елементів класу здійснюється як до звичайного масиву з допомогою квадратних дужок **[ ]**.

Щоб використовувати методи та константи масиву типу vector потрібно підключити заголовок <vector> та простір імен std

```
#include <vector>
using namespace std;
```

Шаблонна специфікація класу vector має такий вигляд:

template <class T, class Allocator = allocator<T> > class vector

де

T – тип елементів масиву. Це може бути будь-який базовий тип (int, float, char тощо) або тип, розроблений користувачем;

Allocator – назва класу, який забезпечує розподіл пам’яті.

Методи класу vector можна умовно розбити на 3 групи:

1. Методи, що визначають та змінюють загальні характеристики масиву.

1.1. Метод size(). Визначити розмір вектора.

1.2. Метод max_size(). Максимально допустимий розмір масиву.

1.3. Метод capacity(). Визначити розмір масиву з врахуванням зарезервованої пам’яті.

1.4. Метод empty(). Визначити, чи порожній вектор.

1.5. Метод shrink_to_fit(). Вирівняти розмір масиву з врахуванням зарезервованої пам’яті за фактичним розміром масиву.

1.6. Метод reserve(). Зарезервувати пам’ять для додаткових елементів масиву.

1.7. Метод resize(). Змінити розмір масиву.

2. Методи, що модифікують (змінюють) дані у масиві.

2.1. Метод push_back(). Додати елемент до кінця вектора.

2.2. Метод pop_back(). Видалити останній елемент вектора.

2.3. Метод clear(). Видаляє з масиву всі елементи.

2.4. Спосіб swap(). Обмін місцями двох векторів.

2.5. Присвоєння масивів. Перевантажений оператор =.

2.6. Метод erase(). Видалили елемент або ряд елементів заданого діапазону.

2.7. Метод insert(). Вставляє елемент або групу елементів у вектор.

2.7.1. Вставка списку ініціалізації у вектор.

2.7.2. Вставка елемента задану кількість разів у задану позицію.

2.7.3. Вставка одиночного елемента у задану позицію.

2.7.4. Вставка декількох елементів із заданого діапазону.

2.8. Метод assign(). Утворити масив з існуючих даних.

3. Методи, що забезпечують доступ до елементів масиву

3.1. Метод at(). Отримати елемент вектора за його позицією (індексом).

3.2. Метод front(). Повертає посилання на перший елемент вектора.

3.3. Метод back(). Повертає посилання на останній елемент вектора.

3.4. Метод data(). Отримати вказівник на вектор.

3.5. Метод begin(). Повернути ітератор, що вказує на перший елемент вектора.

3.6. Метод end(). Повернути ітератор, що вказує на останній елемент масиву.

3.7. Методи cbegin(), cend(). Отримати константний ітератор на початок і кінець масиву.

3.8. Методи rbegin(), rend(). Доступ до елементів масиву за допомогою реверсного ітератора.

3.9. Методи crbegin(), crend(). Встановити на початок і кінець масиву константний реверсний ітератор.

Для створення масиву в класі vector оголошується низка конструкторів. Нижче наведено найпоширеніші з них.

Щоб створити порожній масив, використовується конструктор.

explicit vector(const Allocator &a = Allocator());

Щоб створити масив, який містить num елементів зі значенням val, використовується конструктор

explicit vector(size_t num, const T & val = T(),
const Allocator &a = Allocator());

Щоб створити масив на основі іншого масиву ob, потрібно використати конструктор

vector(const vector<T, Allocator> &ob);

Щоб створити масив на основі діапазону, на який вказують ітератори start та end, використовується конструктор

template <class InIter> vector<InIter start, InIter end,
const Allocator &a = Allocator());

Приклад створення різних масивів типу vector

```
#include <vector>
using namespace std;

...

// Створити вектор з 10 чисел типу int
vector<int> A1(10);

// Створити пустий вектор, елементи якого мають тип char
vector<char> A2;

//Створити вектор з 5 чисел типу double,
// записати у вектор значення 1.0
vector<double> A3(5, 1.0);

// На основі вектору A3 створити новий вектор A4
vector<double> A4(A3);

// Створити вектор на основі ітераторів, які вказують на
// елементи вектору A4 з індексами від 0 до 1.
vector<double>::iterator start, end; // оголосити ітератори
start = A4.begin();   // встановити перший ітератор на позицію 0
```

```
end = A4.begin() + 2; // другий ітератор в позиції 2 (позиція після 1)
vector<double> A5(start, end); // створити вектор
```

Способи доступу до елементів вектору. Індексування [ ]

Після створення вектора, можна отримувати доступ до його елементів одним зі способів:

- за допомогою операції індексування **[ ]**, як у випадку зі звичайним масивом;
- за допомогою ітератора;
- за допомогою методу at().

Приклади доступу до елементів vector

```
#include <vector>
using namespace std;

// 1. Доступ до вектора з допомогою індексатора []
// 1.1. Створити вектор з 5 чисел типу int
vector<int> A1(5);

// 1.2. Записати у вектор значення [ 1, 2, 3, 4, 5 ]
for (int i = 0; i < A1.size(); i++)
  A1[i] = i + 1;

// 1.3. Вивести вектор на екран
for (int i = 0; i < A1.size(); i++)
  cout << A1[i] << " ";
cout << endl;

// 2. Доступ до елементів вектору з допомогою ітераторів
// 2.1. Створити вектор з 5 чисел типу float
vector<float> A2(5);

// 2.2. Оголосити ітератор на вектор типу float
vector<float>::iterator p;

// 2.3. Записати у вектор числа [ 1.1, 2.2, 3.3, 4.4, 5.5 ]
//      з допомогою ітератора
p = A2.begin();
int i = 0;
while (p != A2.end())
```

```
{
  *p = (float)((i + 1) + (i + 1) * 0.1);
  p++;
  i++;
}

// 2.4. Вивести вектор з допомогою ітератора
p = A2.begin();
while (p != A2.end())
{
  cout << *p << " ";
  p++;
}
cout << endl;

// 3. Доступ до елементів вектору з допомогою методу at()
//    Метод at() – отримати елемент вектору за його позицією
vector<double> A3(5); // Створити масив з 5 елементів типу double

// Заповнити довільними значеннями
A3.at(0) = 2.8; // A3[0] = 2.8;
A3.at(1) = 3.3; // A3[1] = 3.3;
```

Завдання для виконання

Завдання 1. З використанням контейнерного класу vector стандартної бібліотеки шаблонів виконати такі задачі.

1. Задано масив. Визначити:
   а) кількість максимальних елементів у масиві;
   б) кількість мінімальних елементів у масиві.
2. Знайти середнє арифметичне додатних і від'ємних елементів масиву.
3. Визначити, чи містяться в одновимірному масиві однакові елементи.
4. В одновимірному масиві містяться тільки два однакових елементи. Знайдіть їх.
5. Задано масив із 20 натуральних чисел. Знайти послідовність з 5-ти елементів, сума яких максимальна.
6. Визначити, чи містяться в масиві однакові елементи. Вивести їхні номери на екран.

Завдання 2.

1. У двовимірному масиві зберігається інформація про бали, отримані спортсменами-п'ятиборцями в кожному з п'яти видів спорту (у першому рядку – інформація про бали першого спортсмена, у другий – другого і т. д.). Загальна кількість спортсменів – 20.

Визначити:

а) скільки балів набрав спортсмен-переможець змагань;

б) скільки балів набрав спортсмен, що зайняв останнє місце.

2. Даний двовимірний масив із двох рядків і двадцяти стовпців, максимальну суму елементів у двох сусідніх стовпцях.

3. Даний двовимірний масив з двадцяти двох рядків і двох стовпців Знайти номери двох сусідніх рядків, сума елементів у яких максимальна.

4. Визначити:

а) який елемент двовимірного масиву більше: розташований у верхньому лівому або у верхньому правому куті;

б) який елемент двовимірного масиву менше: розташований у нижньому правом або у верхньому лівому куті.

Завдання для виконання

Варіант 1

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям STUDENT, що містить такі поля:

прізвище та ініціали – номер групи – успішність (масив із п'яти елементів).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на дисплей прізвищ і номерів груп для всіх студентів, включених до масиву, якщо середній бал студента більше заданого, якщо таких студентів немає, вивести відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з десяти структур типу STUDENT;

− виведення на дисплей прізвищ і номерів груп для всіх студентів, включених до масиву, якщо середній бал студента більше 4.0;

Варіант 2

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям STUDENT, що містить такі поля;

прізвище та ініціали – номер групи – успішність (масив із п'яти елементів).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на дисплей прізвищ і номерів груп для всіх студентів, включених до масиву, які мають оцінки з предметів не менше заданої кількості, якщо таких студентів немає, вивести відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з десяти структур типу STUDENT;

− виведення на дисплей прізвищ і номерів груп для всіх студентів, які мають оцінки 4 та 5, якщо таких студентів немає, вивести відповідне повідомлення.

Варіант 3

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям STUDENT, що містить такі поля:

прізвище та ініціали – номер групи – успішність (масив із п'яти елементів).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на дисплей прізвищ і номерів груп для всіх студентів, включених до масиву, які мають хоча б одну оцінку із задних предметів, якщо таких студентів немає, вивести відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з десяти структур типу STUDENT; записи мають бути впорядковані за абеткою;

− виведення на дисплей прізвищ і номерів груп для всіх студентів, які мають хоча б одну оцінку 2, якщо таких студентів немає, вивести відповідне повідомлення.

Варіант 4

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям AVIAEXPESS, що містить такі поля:

назву пункту призначення рейсу – номер рейсу – тип літака.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран номерів рейсів і типів літаків, що вилітають до пункту призначення, назва якого збіглася з назвою, переданою як параметр, якщо таких рейсів немає вивести відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з семи елементів типу AVIAEXPESS;

− виведення на екран номерів рейсів та типів літаків, що вилітають до пункту призначення, назва якого збіглася з назвою, введеною з клавіатури;

Варіант 5

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям AVIAEXPESS, що містить такі поля:

назва пункту призначення рейсу – номер рейсу – тип літака.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран номерів рейсів, які обслуговуються заданим типом літака, переданим як параметр, якщо таких рейсів немає вивести відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з семи елементів типу AVIAEXPESS;

− виведення на екран пунктів призначення і номерів рейсів, які обслуговує літак, тип якого введений з клавіатури;

− якщо таких рейсів немає, видати відповідне повідомлення на дисплей.

Варіант 6

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям WORKER, що містить такі поля:

прізвище та ініціали працівника – назва займаної посади – з якого року працює.

Перевизначити функції введення та виведення.

Реалізувати у вигляді методів такі дії: висновок на екран прізвищ працівників, чий стаж роботи в організації перевищує заданого значення (значення задається як параметра методу), якщо таких працівників немає, вивести на екран відповідне повідомлення.

Написати програму, яка виконує такі дії: введення з клавіатури даних у масив, що складається з десяти структур типу WORKER. Висновок на екран прізвищ працівників, чий стаж роботи в організації перевищує значення, введене з клавіатури.

Варіант 7

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням,

конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям TRAIN, що містить такі поля:

назва пункту призначення – номер поїзда – час відправлення.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про поїзди, що відправляються після заданого часу (час передається в метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу TRAIN;

− виведення на екран інформації про поїзди, що відправляються після введеного з клавіатури часу.

Варіант 8

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям TRAIN, що містить такі поля:

назва пункту призначення – номер поїзда – час відправлення.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про поїзди, що прямують у заданий пункт (пункт передається у метод вигляді параметра), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з шести елементів типу TRAIN;

− виведення на екран інформації про поїзди, що прямують до пункту, назва якого введена з клавіатури.

Варіант 9

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням,

конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям TRAIN, що містить такі поля:

назва пункту призначення – номер поїзда – час відправлення.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про поїзди за вказаним номером (номер передається у метод як параметр).

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу TRAIN;

− виведення на екран інформації про поїзд, номер якого введено з клавіатури.

Варіант 10

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям MARSH, що містить такі поля:

назву початкового пункту маршруту – назва кінцевого пункту маршруту – номер маршруту.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про маршрут за вказаним номером (номер передається у метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу MARSH;

− виведення на екран інформації про маршрут, номер якого введено з клавіатури;

Варіант 11

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням,

конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям MARSH, що містить такі поля:

назву початкового пункту маршруту – назва кінцевого пункту маршруту – номер маршруту.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про маршрути, які починаються або закінчуються в пункті (назва пункту передається у метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних до масиву, що складається з восьми елементів типу MARSH;

− виведення на екран інформації про маршрути, які починаються або закінчуються в пункті, назва якого введена з клавіатури;

− якщо таких маршрутів немає, видати відповідне повідомлення на дисплей.

Варіант 12

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям NOTE, що містить такі поля:

прізвище ім'я – номер телефону – дата народження (масив із трьох чисел).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про маршрути, що починаються або закінчуються в пункті (назва пункту передається у метод як параметр).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про номери абонентів, що починаються або закінчуються певним прізвищем (прізвище передається у метод як параметр).

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу NOTE;

− виведення на екран інформації про людину, номер якого введений з клавіатури.

Варіант 13

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення та виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям NOTE, що містить такі поля:

прізвище ім'я – номер телефону – дата народження (масив із трьох чисел).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про людей, чиї дні народження припадають на місяць (місяць передається у метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу NOTE;

− виведення на екран інформації про людей, чиї дні народження припадають на місяць, значення якого введено з клавіатури.

Варіант 14

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям NOTE, що містить такі поля:

прізвище ім'я – номер телефону – дата народження (масив із трьох чисел).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про людину (ім'я людини передається у метод як параметр), якщо таких немає, видати відповідне повідомлення на дисплей.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу NOTE;

− виведення на екран інформації про людину, чиє прізвище введене з клавіатури.

Варіант 15

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, в якому зазначено, що масив є масивом структур з ім'ям ZNAK, що містить такі поля:

прізвище ім'я – знак зодіаку – дата народження (масив із трьох чисел).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про людину (ім'я людини передається у метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу ZNAK;

− виведення на екран інформації про людину, чиє прізвище введене з клавіатури.

Варіант 16

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям ZNAK, що містить такі поля:

прізвище ім'я – знак зодіаку – дата народження (масив із трьох чисел).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про людей, що народилися під знаком (назва знака передається у метод як параметр), якщо таких немає, видати відповідне повідомлення на дисплей.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу ZNAK;

− виведення на екран інформації про людей, що народилися під знаком, назва якого введена з клавіатури!

Варіант 17

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення та виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям ZNAK, що містить такі поля:

прізвище, ім'я – знак зодіаку – дата народження (масив із трьох чисел).

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про людей, що народилися протягом місяця (назва місяця передається у метод як параметр), якщо таких немає, видати відповідне повідомлення на дисплей.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу ZNAK;

− виведення на екран інформації про людей, що народилися протягом місяця, значення якого введено з клавіатури.

Варіант 18

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям PRICE, що містить такі поля:

назва товару – назва магазину, в якому продається товар – вартість товару в грн.

Перевизначити функції введення і виведення. Реалізувати у вигляді методів такі дії: висновок на екран інформації про товар (назва товару передається у метод як параметр), якщо такого товару немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу PRICE;

− виведення на екран інформації про товар, назву якого введено з клавіатури.

Варіант 19

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям PRICE, що містить такі поля:

назва товару – назву магазину, в якому продається товар – вартість товару в грн.

Перевизначити функції введення і виведення.

Реалізувати у вигляді методів такі дії: виведення на екран інформації про товари, що продаються в магазині (назва магазину передається у метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу PRICE;

− виведення на екран інформації про товари, що продаються в магазині, назви яких уведені з клавіатури.

Варіант 20

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор із параметрами та конструктор копіювання. Деструктор має бути віртуальним. У базовому класі визначити кількість віртуальних методів введення і виведення.

Створити клас спадкоємець, у якому зазначено, що масив є масивом структур з ім'ям ORDER, що містить такі поля:

розрахунковий рахунок платника – розрахунковий рахунок одержувача – Перерахована сума в руб.

Перевизначити функції введення та виведення. Реалізувати у вигляді методів такі дії: виведення на екран інформації про суму (сума передається у метод як параметр), якщо таких немає, видати на дисплей відповідне повідомлення.

Написати програму, яка виконує такі дії:

− введення з клавіатури даних у масив, що складається з восьми елементів типу ORDER;

− виведення на екран інформації про суму, зняту з розрахункового рахунку платника, запровадженого з клавіатури;

СПИСОК РЕКОМЕНДОВАНИХ ІНФОРМАЦІЙНИХДЖЕРЕЛ

1. Васильєв О. Програмування на С++ в прикладах і задачах : навч. посіб. / О. Васильєв. – Київ : Ліра-К, 2020. – 382 с.
2. Стратієнко Н. К. Алгоритми і структури даних: практикум : навч. посіб. / Н. К. Стратієнко, М. Д. Годлевський, І. О. Бородіна. – Харків : НТУ «ХПІ», 2017. – 224 с.
3. Програмування. Теорія і практика. С++ [Електронний ресурс]. – Режим доступу: https://www.bestprog.net/ru/sitemap_ru/c/ – Назва з екрана.
4. Основи алгоритмізації та програмування : курс лекцій. Частина І. Основи алгоритмізації / укладач М. М. Чепілко. – КПІ ім. Ігоря Сікорського, 2022. Електронний ресурс. – Режим доступу: https://ela.kpi.ua/bitstream/123456789/48938/1/Osnovy_alhorytmizatsii_kurs_lektsii.pdf. – Назва з екрана.
5. Методичні вказівки до лабораторних та практичних робіт з дисципліни «Програмування» за спеціальністю 125 «Кібербезпека», освітня програма «Кібербезпека» для студентів освітньо-кваліфікаційного рівня «бакалавр» [Електронний ресурс] / упоряд.: В. А. Любченко, О. В. Яковлева, Д. О. Руденко – Харків : ХНУРЕ, 2022. – Режим доступу: https://studfile.net/preview/3024298/ – Назва з екрана.
6. Соболь М. О. Основи програмування на С/С++ в прикладах. Частина 2 : навч.-метод. посіб. / Соболь М. О., Любченко Н. Ю, Івашко А. В., Паржин Ю. В., Пугачов Р. В. – Харків : НТУ «ХПІ», 2022. – 200 с.
7. Соболь М. О. Основи програмування на С/С++ в прикладах. : навч.-метод. посіб. / Соболь М. О., Любченко Н. Ю, Паржин Ю. В., Пугачов Р. В. – Харків : НТУ «ХПІ», 2021. – Ч. 1. – 113 с.
8. Уроки програмування на С++ [Електронний ресурс]. – Режим доступу: https://acode.com.ua/uroki-po-cpp/ – Назва з екрана.
9. Ольховський Д. М. Програмування : метод. рек. щодо виконання курсового проекту / Д. М. Ольховський. – Полтава : ПУЕТ, 2013. – 19 с.
10. Owen Hughes. C++ programming language: How it became the invisible foundation for everything, and what's next. – 2020. Acces: https://www.techrepublic.com/article/c-programming-

language-how-it-became-the-invisible-foundation-for-everything-and-whats-next/
11. Алгоритми та структури даних: конспект лекцій. Частина 1 : Структури даних / укладачі: О. Д. Воробйова, Л. В. Глазунова. – Одеса : ОНАЗ ім. О. С. Попова, 2017. – 48 с.
12. Алгоритми та структури даних: конспект лекцій. Частина 2 : Алгоритми пошуку, стиснення даних, внутрішнього та зовнішнього сортування, алгоритми на графах / укладачі: О. Д. Воробйова, Л. В. Глазунова – Одеса : ОНАЗ ім. О. С. Попова, 2017. – 52 с.
13. Довідник з мови C++ [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/ru-ru/cpp/cpp/cpp-language-reference?view=msvc-170. – Назва з екрана.
14. LearnCpp [Електронний ресурс]. – Режим доступу: https://www.learncpp.com/ – Назва з екрана.
15. Microsoft C++, C, and Assembler documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/cpp/?view=msvc-170. – Назва з екрана.

Навчально-методичне видання

КОШОВА Оксана Петрівна
ОЛЬХОВСЬКА Олена Володимирівна
ОЛЬХОВСЬКИЙ Дмитро Миколайович
ОРІХІВСЬКА Оксана Григорівна

ПРОГРАМУВАННЯ ІІ

НАВЧАЛЬНО-МЕТОДИЧНИЙ ПОСІБНИК

Редагування *Л. М. Діденко*
Комп'ютерне верстання *О. С. Корніліч*

Формат 60х84/16. Ум. друк. арк. 18,2.
Зам. № 265/2044.

Видавець і виготовлювач
Вищий навчальний заклад Укоопспілки
«Полтавський університет економіки і торгівлі»,
к. 115, вул. Коваля, 3, м. Полтава, 36014; ☎(0532) 50-24-81

Свідоцтво про внесення до Державного реєстру видавців, виготівників і розповсюджувачів видавничої продукції ДК № 3827 від 08.07.2010 р.