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

ПРОЕКТНЕ НАВЧАННЯ З 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., доцентка, доцентка кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі»;
О. В. Ольховська, к. ф.-м. н., завідувачка 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. викладачка кафедри комп'ютерних наук та інформаційних технологій Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі».

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

Проектне навчання з програмування II : 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. – 210 с. – 1 електрон. опт. диск (CVD-ROM).

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

Повне чи часткове відтворення, тиражування, передрук і розповсюдження цього видання без дозволу Вищого навчального закладу Укоопспілки «Полтавський університет економіки і торгівлі» **ЗАБОРОНЕНО**

© Вищий навчальний заклад Укоопспілки «Полтавський університет економіки і торгівлі», 2023

ЗМІСТ

Вступ 4

Модуль 1. Введення до мови програмування C++.
Керівні структури. Функції 6

Тема 1. Введення до мови програмування C++.
Керівні структури 6

Тема 2. Функції 14

Завдання для самостійної роботи 24

Модуль 2. Масиви. Вказівники та рядки 26

Тема 3. Масиви 26

Тема 4. Вказівники та рядки 38

Завдання для самостійної роботи 62

Модуль 3. Класи. Перевантаження операцій 64

Тема 5. Класи й абстрагування даних 64

Тема 6. Класи II 85

Завдання для самостійної роботи 116

Тема 7. Перевантаження операцій 118

Модуль 4. Наслідування. Віртуальні функції і поліморфізм ... 146

Тема 8. Наслідування 146

Завдання для самостійної роботи 175

Тема 9. Віртуальні функції і поліморфізм 178

Завдання для самостійної роботи 195

Рекомендовані теми проектів 207

Список рекомендованих інформаційних джерел 209

ВСТУП

Основною метою вивчення дисципліни «Проектне навчання з Програмування ІІ» є формування у студентів системного мислення і навичок алгоритмічного програмування та об'єктно-орієнтованого програмування з використанням засобів мови програмування високого рівня С++. Ця дисципліна є логічним продовження курсу «Програмування ІІ» і призначена для підвищення рівня сформованості загальних і спеціальних компетентностей, передбачених навчальною програмою дисципліни та освітньо-професійною програмою підготовки фахівців із спеціальності 122 Комп'ютерні науки. Крім цього, виконання студентами запропонованих авторами завдань підвищеної складності з дисципліни сприятимуть формуванню саме професійної компетентності майбутніх фахівців із комп'ютерних наук.

Цей посібник може бути використаний, як для самостійного опанування студентами цієї дисципліни, так і під час виконання практичних завдань чи написання проектів із С++. Оскільки студенти вже почали вивчення мови програмування С++ у дисципліні «Програмування ІІ», акцент у посібнику зміщено на практичні приклади реалізації різноманітних задач із тем курсу. Якщо у студентів є необхідність повторення теоретичних відомостей, слід використовувати навчально-методичний посібник із дисципліни «Програмування ІІ» або відповідні дистанційні курси.

У посібнику представлено 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 в 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.

Перелік орієнтовних тем виконання проектів також міститься в цьому посібнику. Оскільки чимала частина студентів уже мають навички практичної роботи із програмування, авторами передбачена можливість вибору і написання проекту

з власної теми (яка цікава саме студенту), узгодженої попередньо з викладачем.

Компетентності, якими студенти опанують під час вивчення цієї дисципліни, і з допомогою посібника у тому числі, допоможуть у вивченні багатьох дисциплін, та яких як «Алгоритми і структури даних», «Програмування і підтримка вебзастосувань», «Виробнича практика», «Переддипломна практика», «Дипломне проектування». Крім того, опанування такою широко затребуваною мовою програмування, як C++ 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 у списку використаних джерел.

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

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

Логічні оператори. if (else), switch, ?: Оператори break та continue

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;
}
```

Використання умовних операторів

Обчислити значення виразу, для введених із 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;
}
```

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 while, for, do (while)

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

Задано два цілих числа a і b, a 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 уводить 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 уводить 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 уводить 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 уводить 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 уводить 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 уводить 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;
}
```

Приклад передачі 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 передачі 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 передачі 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

Функцію, 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 відносять:

- можливість переповнення стеку;
- повільніші, за рахунок часу, який витрачається на повторні виклики.

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

Приклад рекурсивної функції
Рекурсивна функція факторіалу.
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 факторіалу 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 (| X | <1) і ціле число N (> 0). 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
```

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
```

Шаблонні 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 компілятору «встроїти» їх у місце виклику, як це робиться для макросів (тема макросів розглядається у 6-му розділі). 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; }
```

Завдання для самостійної роботи

1. Створити функцію знаходження мінімального елемента квадратної матриці і сортування її головної діагоналі.
2. Розробити функцію, яка знаходить однакові елементи у двох масивах, і використати її для обробки рядків матриці.
3. Розробити функцію, що знаходить оптимуми математичної функції, заданої на деякому інтервалі.
4. Для заданих m та n знайти число сполучень і перестановок, скориставшись функцією обчислення факторіалу числа.
5. Розробити функцію для обчислення середнього арифметичного та середнього геометричного масиву чисел.
6. Створити функцію, яка міняє місцями перший і останній рядки квадратної матриці та знаходить добуток елементів головної діагоналі.

7. Створити функцію, яка вставляє задане число у відсортований масив будь-яку кількість разів.

8. Розробити функцію, яка для довільної матриці знаходить мінімуми в рядках, а серед них – максимум.

9. Перевірити, чи є задана матриця ортогональною, тобто скалярні добутки кожного рядка на інший дорівнюють 0.

10. Написати функцію, яка залежно від заданої ознаки проводить сортування масиву за зростанням або за зменшенням.

11. Створити функцію сортування другого рядка матриці і знаходження його максимального елемента.

12. Написати функцію, що розраховує відсотки за вкладом залежно від виду та розміру вкладу.

13. Створити функцію знаходження суми додатних елементів квадратної матриці і сортування головної діагоналі.

14. Знайти шість найменших елементів масиву, користуючись функцією сортування елементів.

15. Створити функцію знаходження суми діагоналей квадратної матриці і сортування її першого стовпця.

16. Створити функцію знаходження суми елементів, що розташовані над головною діагоналлю квадратної матриці, і в тому випадку, коли ця сума парна, обнулити головну діагональ.

17. Створити функцію, що визначає номери елементів, які зустрічаються у масиві тільки один раз.

18. Створити функцію, яка у масиві обнуляє елементи, що знаходяться між максимумом і мінімумом.

19. Створити функцію, яка міняє місцями перший і останній стовпці квадратної матриці та знаходить суму елементів побічної діагоналі.

20. Провести злиття відсортованих за зростанням масивів, розробивши відповідну функцію.

21. Створити функцію, яка сортує елементи матриці за зростанням для кожного стовпця і знаходить мінімальний елемент.

22. Створити функцію знаходження суми елементів квадратної матриці та сортування її головної діагоналі.

23. Скориставшись функцією сортування масиву, відсортувати рядки матриці.

24. Створити функцію знаходження добутку трикутної матриці, що знаходиться нижче головної діагоналі у квадратній матриці, та сортування її другого стовпця.

25. Створити функцію, яка сортує елементи масиву до максимального елемента.

26. Написати функцію, яка в масиві робить перетворення: спочатку записує додатні, а потім – від'ємні елементи, не змінюючи їх порядку слідування.

27. Розробити функцію, яка сортує тільки додатні елементи масиву на своїх місцях.

28. Створити функцію знаходження мінімального елемента квадратної матриці і сортування її побічної діагоналі.

29. Розробити функцію знаходження суми від'ємних елементів одновимірного масиву та сортування масиву за зростанням.

30. Розробити функцію, що знаходить максимальний і мінімальний елементи, їх номери в одновимірному масиві та міняє ці елементи місцями.

МОДУЛЬ 2. МАСИВИ. ВКАЗІВНИКИ ТА РЯДКИ

Тема 3. Масиви

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
```

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
```

Приклад знаходження індексу й 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
```

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

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];

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 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

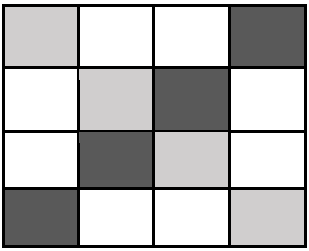
```
-9      -6      -1      9       -2
0       0       -1      5       0
-8      9       10      -6      10
-3      -7      5       6       6

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

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

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)



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

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

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;
	Варіант 1
	#include <iostream>
	#include <cstdlib>
	using namespace std;
	int main()
	{
	double* a = new double;
	*a = 2.5;
	cout << *a << endl;

```
   cout << *a << endl;
   free(a);
   return 0;
}
```

```
   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* ІМЯ = new ТИП[n]; delete [] ІМ'Я;

#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] << " ";
	#include <iostream>
	#include <stdlib.h>
	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] << " ";

```
    }
    free(mas);
    return 0;
}
```

```
    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) В 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
```

В 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) В циклі n-1 раз обчислюємо адресу початку кожного рядка, розпочинаючи з індексом 1, і записуємо їх в 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 у 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 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
```

В 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 із 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
```

Передача одновимірних масивів у функцію

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
```

Передача двовимірного динамічного масиву у функцію

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;
}
```

```
//Передача параметра функції через масив
//Макс суму елемента рядка

#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

Обробка символів і рядків
Рядки.
Основні функції обробки символьних типів string

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

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;
}
```

Завдання для самостійної роботи

Завдання 1. Робота з масивами

1. Відсортувати за зростанням перший рядок і другий стовпець матриці.
2. Визначити, скільки елементів матриці в рядках із парними номерами перевищують відношення максимального елемента до мінімального?
3. Відсортувати за зростанням добутки від'ємних елементів рядків матриці.
4. Відсортувати за зменшенням масив максимальних елементів стовпців матриці.
5. У матриці відсортувати за зростанням елементи головної діагоналі та поміняти місцями максимальний елемент із мінімальним.
6. Переставити рядки матриці відповідно до зростання елементів третього стовпця.

7. Знайти мінімальний елемент матриці, і якщо він парний і парний рядок його розташування — відсортувати третій стовпець матриці.

8. Переставити стовпці матриці відповідно до зростання елементів третього рядка.

9. Вихідну матрицю відсортувати за зростанням кількісті додатних елементів у стовпцях.

10. У квадратній матриці провести сортування тих рядків, у яких на побічній діагоналі стоїть парне число.

11. У стовпцях матриці знайти максимуми та відсортувати стовпці за зменшенням.

12. Переставити стовпці матриці відповідно до зростання елементів другого рядка.

13. Знайти добутки діагоналей квадратної матриці. Відсортувати перший рядок за зростанням.

14. Задано дві матриці розміром 3*3. Перемножити їх за правилом «рядок на стовпець».

15. Знайти скалярні добутки першого рядка матриці на всі інші.

Завдання 2. Робота з рядками

1. У тексті слово «країна» замінити словом «Україна».

2. Виписати із тексту слова, що починаються та закінчуються цифрою.

3. Знайти у тексті найдовше слово, що складається з цифр.

4. Видалити із тексту слова, що закінчуються на «а».

5. Чи є в тексті слова, що починаються і закінчуються однією і тією ж літерою?

6. У якому слові тексту менше всього голосних звуків?

7. Видалити з тексту всі слова, що починаються з літер, які задаються в рядку запиту.

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

9. Уведене речення обробити так, щоб його послідовність слів надрукувати у зворотному порядку.

10. Надрукувати слово тексту з найбільшою кількістю приголосних звуків.

11. Увести два речення. Якщо у них є однакові слова, то замінити їх у другому реченні на відповідну кількість символів «=».

12. Підрахувати у тексті кількість слів, у яких немає цифр.

13. Вибрати з тексту найдовше слово.

14. Сформувати масив, елементи якого налічують кількість літер у словах речення.

15. Видалити з тексту всі зайві (більше одного) пропуски .

МОДУЛЬ 3. КЛАСИ. ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

Тема 5. Класи й абстрагування даних

Структура

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 указання 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'єкта, наприклад, який зберігається у покажчику.

Для структур визначена операція = (присвоїти), яка працює як побітове копіювання.

У мові C++ операцію -> можна перевизначити, а операцію . – ні.

Бітові поля

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.

Приклад використання бітових полів у структурі Date, що описує дату дня і 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

```
#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 з 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
```

Об’єднання union

Об’єднання (union) являє собою окремий випадок структури, всі поля якої розташовуються за однією адресою.

Формат опису такої ж, 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 у 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
```

Приклади оголошення вкладених об'єднань (структур, класів) в 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
```

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 співпадають 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

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

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

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

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 вказує, 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) із 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 – тимчасовєий об’єкт, який створюється за 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’єкта 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

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 указує на статичний член даних об'єкта obj класу CPi
       pPi = &obj.Pi;
       pExp = &::Exp; // указує 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;
    }
```

Види функцій-елементів (членів) класу. Звичайні функції-елементи. Статичні (static) функції-члени

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'єкта 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

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;
}
```

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
```

Костантні об'єкти та функції – елементи класу

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
```

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;
}
```

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).

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 та дружні функції. Ключове слово friend

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 у 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 []

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’єкта 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
```

Приклад динамічного виділення і звільнення пам’яті для масиву покажчиків на базовий тип

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 елемента 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
```

Перерозподіл пам’яті для структур, ініціалізація структур

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, дозаповнювати 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 покажчика p
    delete[] p; // можна було delete[] p2; оскільки p і p2 вказують на ту саму область пам’яті

    return 0;
}
```

Завдання для самостійної роботи

Завдання 1. Структури

1. Враховуючи інформацію про результати сесії групи, надрукувати список студентів, які будуть отримувати стипендію.

2. Сформувати список співробітників відділу: прізвище, ім'я та по батькові (ПІБ), число, місяць народження. Вивести інформацію про співробітників, які народилися в задані місяці.

3. Сформувати документ, у якому зберігається інформація про факультети та їх академічні заборгованості. Надрукувати інформацію про факультети з максимальною і мінімальною кількістю заборгованостей.

4. Враховуючи інформацію про предмети, вилучені на митниці за звітний період: дата вилучення, найменування предмета, кількість одиниць, вартість; визначити назву предмета, якого вилучено максимальну кількість разів.

5. Враховуючи інформацію про фінансову діяльність підприємств: найменування підприємства, місяць, прибуток підприємства; сформувати список підприємств, за зростанням їх прибутку.

6. Ввести в комп'ютер інформацію про співробітника відділу: прізвище, ім'я, вік. Надрукувати імена співробітників, вік яких перевищує 28 роки.

7. У відомості зберігається інформація про пропуски занять студентами: ПІБ, факультет, група, кількість пропусків. Вивести список із 10 студентів із найбільшою кількістю пропусків.

8. У списку міститься інформація про власників гаражного кооперативу: ПІБ, марка автомобіля, номер, рік випуску автомобіля. Сформувати список володарів машин, рік експлуатації яких перевищує 10 років.

9. У списку міститься інформація про власників автотранспорту: ПІБ, марка, номер, потужність автомобіля. Сформування список власників машин, відсортований за зростанням потужності автомобілів.

10. Враховуючи інформацію про підрозділи: найменування підрозділу, кількість співробітників, рік заснування; відсортувати інформацію про підрозділи за зростанням року заснування.

11. У розкладі зберігається інформація про рух поїздів до станції Харків: номер поїзду, маршрут руху, час прибуття, час відправлення. Необхідно надрукувати інформацію про поїзди, що прибувають в заданий час.

12. 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.

13. Ввести в комп'ютер інформацію про студентів групи. На екран вивести повідомлення про те, у який місяць року народилась найбільша кількість студентів групи.

14. Сформувати документ, у якому зберігається інформація про асортимент продовольчих товарів у магазині: назва магазину, код товару, кількість цього товару, ціна за кілограм. Відсортувати магазини для оптової закупки заданого товару згідно з його ціною.

15. Враховуючи інформацію про предмети, вилучені на митниці за звітний період: дата вилучення, код предмету, кількість одиниць; визначити групу предметів, по якій найчастіше відбувалося вилучення.

Завдання 2. Робота з класами

1. Створити клас «Ліфт», що є гранично спрощеною моделлю ліфта. Клас повинен забезпечити:

- встановлення діапазону руху ліфта (нижній і верхній поверх);
- включення/вимикання ліфта;
- повернення поточного стану ліфта (працює/не працює);
- повернення поточного положення ліфта (поверх);
- обробку виклику ліфта (поверх).

Написати програму, яка тестує клас «Ліфт».

2. Написати програму «Вартість шпалер». програма запитує:

- кількість кімнат у квартирі, в яких планується клеїти шпалери;
- параметри кожної кімнати;

− параметри кожного виду шпалер, які планується використовувати.

У результаті розрахунку програма видає: необхідну кількість рулонів кожного виду, загальну вартість закупівлі шпалер.

Розробити та використовувати у програмі класи:

«Квартира», «Кімната», «Рулон» «Шпалери».

Атрибути (поля): список кімнат.

Атрибути кімнати: назва, розміри, клеїти стелю чи ні.

Атрибути рулону: назва, розмір, ціна.

3. Написати програму «Група студентів».

Програма отримує такі дані:

− список студентів групи;
− список предметів для групи;
− таблицю оцінок студентів групи з усіх предметів.

Дані вводяться із відповідних файлів. Імена файлів запитує програма. Програма виводить такі дані:

− таблицю оцінок студентів із предметів;
− середні оцінки студентів;
− середні оцінки з предметів;
− середній бал групи;
− максимальні та мінімальні оцінки з предметів із зазначенням студентів.

Розробити та використовувати у програмі класи:

«Група», «Студент», «Предмет».

Атрибути групи: назва, перелік студентів, перелік предметів.

Атрибути студента: ім’я, перелік оцінок.

Атрибути предмета: назва.

Тема 7. Перевантаження операцій

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;
    }
};
```

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
}
```

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;
    }
};
```

На використання перевантажених операторів накладаються такі обмеження:

− під час перевантаження, 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 ++, - -

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

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
```

Відмінності в перевантаженні префіксних і постфіксних операторів інкременту (++) та декременту (- -) з допомогою «дружніх» функцій

Для того, щоб під час реалізації «дружньої» функції до класу, відрізнити префіксну та постфіксну форми реалізації операторної функції ++ або --, потрібно дотримуватись 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 в 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 служить 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 +=, –=, *=, /=, %=

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 скорочень 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->()

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 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 або 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’єкта класу MyMoney відняти інший об’єкт класу MyMoney, у результаті отримаємо різницю кінцевих сум за відповідними вкладами.

Префіксна форма оператора декременту: 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
```

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-члена поза тілом класу закритий.

```
class Parent
{
public:
   int m_public; // доступ до цього члену 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 змініте 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 та 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.

Спадкування типу public

Відкрите спадкування є одним з найбільш використовуваних типів спадкування. Дуже рідко ви побачите або будете використ-

товувати 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

У разі закритого спадкування всі члени батьківського класу наслідуються як закриті. Це означає, що private-члени залишаються недоступними, а protected- і public-члени стають private в 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 доступ до цих членів:

```
class D3 : public D2
{
    // Відкрите спадкування означає, що:
    // - успадковані public-члени залишаються public у дочірньому класі;
    // - успадковані protected-члени залишаються protected у 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
```

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. Поліморфізм

```
#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'язковиц список. За бажанням до тексту класу можна додати інші функції, які розширять його можливості.

У класі оголошуються такі поля та методи:

- покажчик pTop – вершина стека;
- конструктор за замовчуванням;
- конструктор копіювання StackList (const StackList &);
- деструктор;
- метод Push(), який містить 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 реалізовано 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

Необхідно створити клас «двовимірний масив». Відповідні завдання мають бути реалізовані у вигляді методів класу. Інтерфейс класу та його реалізація мають бути в окремих модулях: файлах *.h *.cpp. Крім того кожен основний клас, що розробляється, повинен реалізовувати:

1. Конструктор за замовчуванням.
2. Деструктор.
3. Конструктори з параметрами.
4. Конструктор копіювання.
5. Перевантажену операцію присвоєння.
6. Перевантажену операцію виведення в потік.
7. Метод завантаження даних із файлу.
8. Метод збереження даних у файл.
9. Заповнення даних випадковим чином.

Варіанти завдань:

Варіант 1

Дана ціла прямокутна матриця. Визначити:

1) кількість рядків, що не містять жодного нульового елемента;

2) максимальне з чисел, що зустрічаються в заданій матриці більше одного разу.

Варіант 2

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

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

Варіант 3

Дана ціла прямокутна матриця. Визначити:

1) кількість стовпців, що містять хоча б один нульовий елемент;

2) номер рядка, в якому знаходиться найдовша серія однакових елементів.

Варіант 4

Дана цілочисельна квадратна матриця. Визначити:

1) добуток елементів у тих рядках, які не містять негативних елементів;

2) максимум серед сум елементів діагоналей, паралельних головній діагоналі матриці.

Варіант 5

Дана цілочисельна квадратна матриця. Визначити:

1) суму елементів у стовпцях, які не містять негативних елементів;

2) мінімум серед сум модулів елементів діагоналей, паралельних до побічної діагоналі матриці.

Варіант 6

Дана ціла прямокутна матриця. Визначити:

1) суму елементів у тих рядках, які містять хоча б один негативний елемент;

2) номери рядків і стовпців усіх сідлових точок матриці.

Варіант 7

Для заданої матриці розміром 8 на 8 знайти такі k, у якому k рядок матриці збігається з k-м стовпцем.

Знайти суму елементів у рядках, які містять хоча б один негативний елемент.

Варіант 8

Характеристикою стовпця цілої матриці назвемо суму модулів його негативних непарних елементів. Переставляючи стовпці заданої матриці, розташувати їх відповідно до зростання характеристик.

Знайти суму елементів у стовпцях, які містять хоча б один негативний елемент.

Варіант 9

Сусідами елемента – у матриці назвемо елементи сі-1 кі+1, j-1 j+1, (к,) (i, j). Операція згладжування матриці дає нову матрицю того ж розміру, кожен елемент якої виходить як середнє арифметичне сусідів відповідного елемента вихідної матриці. Побудувати результат згладжування заданої речової матриці розміром 10 на 10. У згладженій матриці знайти суму модулів елементів, розташованих нижче головної діагоналі.

Варіант 10

Елемент матриці називається локальним мінімумом, якщо він строго менший від усіх наявних у нього сусідів. Підрахувати кількість локальних мінімумів заданої матриці розміром 10 на 10.

Знайти суму модулів елементів, що розташовані вище головної діагоналі.

Варіант 11

Коефіцієнти системи лінійних рівнянь задані як прямокутна матриця. За допомогою допустимих перетворень привести систему до трикутного вигляду.

Знайти кількість рядків, середнє арифметичне елементів яких менше заданої величини.

Варіант 12

Ущільнити задану матрицю, видаляючи з неї рядки та стовпці, заповнені нулями. Знайти номер першого рядка, що містить хоча б один позитивний елемент.

Варіант 13

Здійснити циклічний зсув елементів прямокутної матриці на п елементів вправо або вниз (залежно від уведеного режиму), п може бути більше кількості елементів у рядку або стовпці.

Варіант 14

Здійснити циклічний зсув елементів квадратної матриці розмірності MxN вправо на k елементів таким чином: елементи 1-го рядка зрушуються в останній стовпець зверху вниз, з нього в останній рядок праворуч наліво, з неї в перший стовпець знизу вгору, з нього в першу рядок; для інших елементів – аналогічно.

Варіант 15

Дана ціла прямокутна матриця. Визначити номер першого зі стовпців, які містять хоча б один нульовий елемент.

Характеристикою рядка цілої матриці назвемо суму її негативних парних елементів. Переставляючи рядки заданої матриці, розташувати їх відповідно до зменшення характеристик.

Тема 9. Віртуальні функції і поліморфізм

Віртуальна функція

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
```

Увага! Ніколи не викликайте віртуальні функції в тілі конструкторів або деструкторів

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 для 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 чисто 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 поліморфізму. Отже, немає сенсу створювати екземпляр такого класу, оскільки об'єкт цього 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 значенням 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 і т. д.) 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 чисто 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 чисту 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 є динамічним масивом, 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

тут

- 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 урахуванням зарезервованої пам'яті.

1.4. Метод empty(). Визначити, чи порожній вектор.

1.5. Метод shrink_to_fit(). Вирівняти розмір масиву з урахуванням 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 вектору.

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(). Видалити 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 та кінець масиву.

3.8. Методи rbegin(), rend(). Доступ до елементів масиву за допомогою реверсного ітератора.

3.9. Методи crbegin(), crend(). Установити на початок та 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 наступний 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 вектору, можна отримувати доступ до його елементів одним з наступних 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 за 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 за допомогою ітератора
p = A2.begin();
while (p != A2.end())
{
  cout << *p << " ";
  p++;
}
cout << endl;

// 3. Доступ до елементів вектору за 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 самостійної роботи

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 та 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 даниху 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 оцінки з предметів хоча б одну задану оцінку, 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 AVIAEXPRESS, що містить такі поля:

назву пункту призначення рейсу – номер рейсу – тип літака.

Перевизначити функції введення s 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 AVIAEXPRESS;

− виведення на екран номерів рейсів і 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 AVIAEXPRESS, 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 AVIAEXPRESS;

− 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 і виведення.

Реалізувати у вигляді методів такі дії:

− висновок на екран прізвищ працівників, чий стаж роботи в організації перевищує задане 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 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 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, в 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, назву якого введено 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, в 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 номери телефонів абонентів, що починаються або закінчуються певним прізвищем (прізвище передається у метод як параметр).

Перевизначити функції введення та виведення.

Реалізувати у вигляді методів такі дії:

− виведення на екран інформації про людину за вказаним номером телефону (номер 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 NOTE;

− виведення на екран інформації про людину, номер якої введено 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 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 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 запроваджено 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 (прізвище 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, 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 людину (прізвище людини передається 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 із клавіатури.

Варіант 16

Створити шаблонний базовий клас, що містить одновимірний шаблонний масив. Визначити конструктор за замовчуванням, конструктор з 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, назву якого введено 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 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 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, у 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, у якому продається товар – вартість товару в грн.

Перевизначити функції введення та 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 уведена 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 грн.

Перевизначити функції введення і 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 масив, що складається з восьми елементів типу ORDER;

− виведення на екран інформації про суму, зняту з розрахункового рахунку платника, введено з клавіатури;

РЕКОМЕНДОВАНІ ТЕМИ ПРОЕКТІВ

1. Власна тема (узгоджена з викладачем).
2. Операції з комплексними числами.
3. Реалізація комп'ютерної гри шахи на основі ієрархії класів.
4. Програмна реалізація операцій із комбінаторними множинами.
5. Програмна реалізація роботи з геометричними фігурами.
6. Реалізація операцій із многочленами.
7. Робота з векторами на площині.
8. Програмна реалізація матричних операцій.
9. Реалізація класу вектору в просторі.
10. Програмна реалізація класу повороту тривимірного простору.
11. Програмна реалізація функціонування кімнатного акваріуму.
12. Реалізація багаторозрядної числової арифметики.
13. Арифметика роботи з раціональними дробами.
14. Розробка програми тестування знань.
15. Програмування бази даних із реалізацією пошукового механізму (телефонний довідник, словник іноземних слів із гра-

фічними зображеннями, картотека працівників підприємства, опис основних функцій мови програмування, учбовий процес студентів).

16. Побудова на площині графіків функцій.
17. Програмування методів чисельного обчислення інтегралів.
18. Розробка спеціального калькулятора (статистичний калькулятор, інженерний калькулятор (обчислення моментів, центра ваги та ін.), калькулятор перетворення систем обчислень, математичний калькулятор (обчислення сум рядів, визначених інтегралів, розв'язання рівнянь та ін.), фінансовий калькулятор (обчислення відсотків за вкладами та ін.).
19. Використання алгоритмів асиметричної криптографії для захисту електронних документів.
20. Реалізація арифметики великих чисел.
21. Створення ігрової комп'ютерної програми «Тетріс».
22. Розробка аудіопрогравача.
23. Програмна реалізація клавіатурного тренажера.
24. Програма обліку та формування електронних бланків рахунків за спожиті комунальні послуги.
25. Програмна реалізація генератора лабіринтів, що мають проходи.

СПИСОК РЕКОМЕНДОВАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

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 [Електронний ресурс] : курс лекцій. Частина І. Основи алгоритмізації / укладач М. М. Чепілко. – КПІ ім. Ігоря Сікорського, 2022. 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. Частина 1 : навч.-метод. посіб. / Соболь М. О., Любченко Н. Ю, Паржин Ю. В., Пугачов Р. В. – Харків : НТУ «ХПІ», 2021. – 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. – 48с.
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

ПРОЕКТНЕ НАВЧАННЯ З ПРОГРАМУВАННЯ ІІ

НАВЧАЛЬНО-МЕТОДИЧНИЙ ПОСІБНИК

Редагування *Л. М. Діденко*
Комп'ютерне верстання *О. С. Корніліч*

Формат 60х84/16. Ум. друк. арк. 13,1.
Зам. № 266/2045.

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.