Лабораторна робота № 1

з курсу «Об’єктно-орієнтоване програмування»

Тема: «Структура-пара»

Структурою-парою називається структура з двома полями, які зазвичай мають імена first і second. Потрібно реалізувати тип даних за допомогою такої структури. У всіх завданнях обов'язково повинні бути присутніми:

- метод ініціалізації Init; метод повинен контролювати значення аргументів на коректність;
- ввід з клавіатури Read;
- вивід на екран Display.

Реалізувати зовнішню функцію з ім'ям make_mun(), де mun - тип реалізованої структури. Функція повинна отримувати в якості аргументів значення для полів структури і повертати структуру необхідного типу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу. Варіанти наступні.

1. Поле first – дробове число; поле second - ціле число, показник степеня. Реалізувати метод power() - піднесення числа first в степінь second. Метод повинен правильно працювати при будь-яких допустимих значеннях first і second.

2. Поле first - дробове число; поле second - дробове число, показник степеня. Реалізувати метод power() - піднесення числа first в степінь second. Метод повинен правильно працювати при будь-яких допустимих значеннях first і second.

3. Поле first - ціле позитивне число, чисельник; поле second - ціле позитивне число, знаменник. Реалізувати метод ipart() - виділення цілої частини дробу first/second. Метод повинен перевіряти нерівність знаменника нулю.

4. Поле first - ціле позитивне число, номінал купюри; номінал може приймати значення 1, 2, 5, 10, 50, 100, 500. Поле second - ціле позитивне число, кількість купюр даної вартості. Реалізувати метод summa() - обчислення грошової суми.

5. Поле first - дробове позитивне число, ціна товару; поле second - ціле позитивне число, кількість одиниць товару. Реалізувати метод cost() - обчислення вартості товару.

6. Поле first - ціле позитивне число, калорійність 100 г продукту; поле second - дробове позитивне число, маса продукту в кілограмах. Реалізувати метод power() - обчислення загальної калорійності продукту.

7. Поле first - дробове число, ліва межа діапазону; поле second - дробове число, права межа діапазону. Реалізувати метод rangecheck() - перевірку заданого числа на належність до діапазону.

8. Поле first - ціле число, ліва межа діапазону, включається в діапазон; поле second - ціле число, права межа діапазону, не включається до діапазону. Пара чисел являє напіввідкритий інтервал [first, second). Реалізувати метод rangecheck() - перевірку заданого цілого числа на приналежність діапазону.

9. Поле first - ціле позитивне число, годинник; поле second - ціле позитивне число, хвилини. Реалізувати метод minutes() - приведення часу в хвилини.

10. Лінійне рівняння y= Ax + B. Поле first - дробове число, коефіцієнт A; поле second - дробове число, коефіцієнт B. Реалізувати метод function() - обчислення для заданого x значення функції y.

11. Лінійне рівняння y = Ax + B. Поле first - дробове число, коефіцієнт A; поле second - дробове число, коефіцієнт B. Реалізувати метод root() - обчислення кореня лінійного рівняння. Метод повинен перевіряти нерівність коефіцієнта B нулю.

12. Поле first - дробове число, координата x точки на площині; поле second - дробове число, координата y точки на площині. Реалізувати метод distance() - відстань точки від початку координат.

13. Поле first - дробове позитивне число, катет a прямокутного трикутника; поле second - дробове позитивне число, катет b прямокутного трикутника. Реалізувати метод hypotenuse() - обчислення гіпотенузи.

14. Поле first - дробове позитивне число, оклад; поле second - ціле число, кількість відпрацьованих днів у місяці. Реалізувати метод summa() - обчислення нарахованої суми за дану кількість днів для заданого місяця: оклад/дні_місяця * відпрацьовані_дні.

15. Поле first - ціле позитивне число, тривалість телефонної розмови в хвилинах; поле second - дробове позитивне число, вартість однієї хвилини у гривнях. Реалізувати метод cost() - обчислення загальної вартості розмови.

16. Поле first - дробове число, ціла частина числа; поле second - позитивне дробове число, дробова частина числа. Реалізувати метод multiply() - множення на довільне дробове число типу double. Метод повинен правильно працювати при будь-яких допустимих значеннях first і second.

17. Поле first - ціле позитивне число, координата курсора/покажчика по горизонталі; поле second - ціле позитивне число, координата курсора по вертикалі. Реалізувати метод changex() - зміна горизонтальної координати курсору; реалізувати метод changey() - зміна вертикальної координати курсору. Методи повинні перевіряти вихід за межі екрану.

18. Поле first - ціле число, ціла частина числа; поле second - позитивне ціле число, дробова частина числа. Реалізувати метод multiply() - множення на довільне ціле число типу int. Метод повинен правильно працювати при будь-яких допустимих значеннях first і second.

19. Число сполучень по k об'єктів з n об'єктів (k<n) обчислюється за формулою

$$C(n,k) = n!/((n-k)! \times k!)$$

Поле first - ціле позитивне число, k; поле second - позитивне ціле число, n. Реалізувати метод combination() - обчислення C(n,k).

20. Елемент a_j геометричної прогресії обчислюється за формулою:

$$a_j = a_0 r^j,\ j = 0, 1, 2, ...$$

Поле first - дробове число, перший елемент прогресії a_0; поле second - стале відношення r. Визначити метод elementj() для обчислення заданого елемента прогресії.

Лабораторна робота №2. Оголошення та структура класу.

Мета роботи: вивчення засобів введення, редагування та компіляції програм, стандартних потоків введення/виведення, вбудованих типів даних, оголошення класу та рівнів захисту.

Теоретичний матеріал лекції, [1, розділ 2], [2, розділ 12], [3, розділ 10].

1. Короткі теоретичні відомості

Організація однофайлової програми С+:

```
# директиви препроцесора включення файлів
Задання простору імен
Оголошення прототипів функцій;
Оголошення глобальних типів, констант та змінних;
/* допоміжні функції */
Тип_функці Імя_функції(Список формальних параметрів)
{
    оголошення локальних типів, констант, змінних;
   інструкції;
   return вираз;
}
…
/* основна функція */
int main()
{
    Оголошення локальних типів, констант та змінних;
    інструкції;
    return 0;
}
```

Включення файлів у програму:

```
#include <iostream>   // потокове введення-виведення
#include <cstdlib>    // стандартна бібліотека
#include <cmath>      // математичні функції
#include <cstring>    // робота із символьними стрічками
```

Задання простору імен `std` або користувача:

```
using namespace std;
using namespace MySpace;
```

Компіляція однофайлової програми С++ з використанням компілятора (ОС Linux) `g++` у пакетному режимі:

1. Запуск програми на мові С++ на компіляцію і компонування виконуваного файлу:

```
g++ prog_1.c -o prog_1
```

2. Запуск на виконання виконуваного файлу:

```
./prog_1 > out
```

3. Прогляд результату виконання:

```
kate out  або  vim out або kwrite out
```

Рекомендована нотація імен для ідентифікаторів програми. Ідентифікатор бажано будувати з декількох об’єднаних слів, кожне з яких починається з великої літери, наприклад `CountEventElements`. Ідентифікатор повинен містити інформацію про призначення змінної і її тип. Інформація про тип змінної записується як префікс.

Таблиця 1. Префікси ідентифікаторів

Префікс	Тип даних
a	масив
ch	символ
i	індекс
l	довге ціле
lp	далекий вказівник
n	ціле
np	ближній вказівник
sz	стрічка, яка закінчується нульовим символом (ASCIIZ-рядок)
w	ціле без знаку довжиною слово
dw	довге ціле без знаку
m	елемент класу

Для організації потокового введення-виведення необхідно виконати включення: `#include <iostream>`. C++ використовує стандартні потокові об'єкти `cin, cout, cerr`.

Для введення з потоку використовується перевантажена операція >> , яка застосовується до потокового об'єкту `cin`:

```
int x;
cin >> x;
```

Для виведення у потік використовується перевантажена операція << , яка прцює з потоковим об'єктом cout:

```
cout << "Виведення рядка" << endl;
```

Для виведення повідомлень про помилки використовується перевантажена операція << , яка працює з потоковим об'єктом `cerr`:

```
cerr << "Помилки виконання" << endl;
```

Інструкція for each, in

Нова інструкція `for each, in` використовується для роботи з колекціями. Колекція - це набір об'єктів одного типу. Колекціями є масиви і рядки символів.

```
int vector[]={-2, 9, 5, -1, 8};
for each ( int x in vector ) cout << x << endl;
```

У межах дії інструкції `for each, in` можна змінити елементи колекції, але не дозволяється доповнення колекції або вилучення її членів.

Типи даних

Прості типи даних: символ `(char)`, цілі `(int, long, long long)`, bool (`true, false`). У булевих виразах відмінне від нуля значення перетворюється на `true`, а нульове - на `false`. У числових виразах, навпаки, `true` перетворюється на 1, а `false` - на 0). Типи з плаваючою крапкою (`float, double`).

Структуровані типи даних: `struct, enum, union, class`.

Посилання - це тип даних, який має властивості розіменованого вказівника:

```
Тип & ідентифікатор1 = ідентифікатор2;
```

Клас - це структурований тип даних, який інкапсулює (обмежує область досяжності) оголошення полів даних та функцій для їх використання. Клас використовується для позначення множини об'єктів, які мають однакову структуру. Структура (зміст) класу називається його *протоколом*. Сукупність методів класу, через які відбувається доступ до елементів класу, називається *інтерфейсом*.

Елементи класу (дані та методи) захищені оголошенням класу:

```
class назва_класу {
public:
   // дані та методи
protected:
```

```
   // дані та методи
private:
   // дані та методи
};
```

Клас C++ визначає три рівні захисту своїх елементів:

- public (відкриті);
- private (закриті) - діє за замовчуванням;
- protected (захищені).

Відкриті елементи досяжні у класі та поза класом у межах дії встановленого простору імен. Закриті елементи можуть використовуватися тільки у межах класу. Захищені елементи доступні у класі та похідних від нього класах.

Клас надає зовнішнім функціях та класам можливість доступу своїх елементів за допомогою методів, розміщених у відкритій частині.

Для ініціалізації даних класу використовується *конструктор* – метод ім'я якого співпадає з іменем класу. Конструктор не повертає значення. Конструктори автоматично викликаються при створенні об'єктів у сегментах коду, стека або у динамічній пам'яті.

У класі за замовчуванням завжди існує конструктор без параметрів (void-конструктор), конструктор копіювання, деструктор та операторна функція присвоєння об'єктів.

Ініціалізація даних конструктором може здійснюватися у списку ініціалізації після символу двокрапка, або в тілі конструктора. Константні поля даних, дані з типом посилання, успадковані дані, дані з типом іншого класу (якщо перекритий конструктор за замовчуванням) ініціалізуються конструктором після символу ':'.

Якщо об'єкт виходить із області досяжності він автоматично знищується методом спеціального призначення – деструктором. Назва деструктора складається з символу "~" та назви класу.

Крім членів класу, оголошення класу може містити прототипи дружніх (friend) функцій або класів. Методи класу та дружні до класу функції мають можливість доступу до елементів усіх частин класу. Рівні захисту діють тільки на членів класу і не діють на друзів класу. Визначати методи та дружні функції можна як у класі, так і поза класом.

Елементи класу (крім друзів, конструкторів, деструктора та операції присвоєння допускають успадкування, яке полягає у використанні оголошень даних та методів базового класу у похідному класі.

У протоколі класу дозволяється оголошувати типи даних за допомогою специфікатора typedef:

```
class Base {
public:
typedef void * PVOID;
Base(int a=0, int b=0, int c=0) {x=a; y=b; z=c;}
~Base(){}
private:
int x,y,x;
};
void main() {
Base::PVOID q;
…
}
```

Запитання.

1. Дати коротке визначення інкапсуляції, поліморфізму і успадкування.
2. Дати характеристику простим типам даних мови C++.
3. Дати характеристику структурованим типам даних мови C++.
4. Для чого використовується анонімне об'єднання.
5. Для чого використовується перелічимий тип.
6. Як працює інструкція for each, in.

7. В чому відмінність між структурою і класом.

8. Для чого використовуються простори імен.

9. Оголошення класу і його синтаксис. Відкриті, закриті і захищені члени класу.

10. Як можна ініціалізувати дані конструктором.

11. Де можна розміщувати функції-члени і як до них звертатися.

12. Як використовуються дружні функції і класи.

Завдання.

1. Написати програму, яка запитує в циклі у користувача ціле число N=10, а потім виводить вказане число зірочками (перший рядок - одна зірочка, другий рядок - дві зірочки і т.д.). При введенні числа більшого від N, програма завершує роботу.

2. Написати програму, яка використовує безіменне об'єднання для переставлення байтів числа `short int` (16 бітів).

3. Написати програму для обчислення виразу $y = a \cdot x^2 + b \cdot x + c$ для комплексних коефіцієнтів *a, b, c* у точці *x*.

4. Припустимо, що є наступне оголошення структури:

```
struct app {
   char name[32];
   int  state[2];
};
```

а) написати функцію, яка приймає структуру `app` як аргумент і відображає її вміст;

б) написати функцію, яка приймає адрес структури `app` як аргумент і відображає її вміст;

в) написати функцію, яка приймає поилання на структуру `app` як аргумент і відображає її вміст.

5. Припустимо, що є наступне оголошення структури:

```
struct app {
char name[32] ;
int  state[2];
};
```

Написати програму, яка розміщує масив з двох структур у динамічній пам'яті, ініціалізує елементи масиву структур значеннями і відображає їх вміст.

6. Припусимо, що функції `f1()` и `f2()` використовують структуру `app` і мають наступні прототипи:

```
void f1(app * a);
const char * f2(const app * a1, const app * a2);
```

Оголосити `p1` як вказівник на функцию `f1`, а `p2` - як вказівник на функцію `f2`. Оголосити `ap` як масив з двох вказівників того ж типу, що і `p1`, і оголосити `pa` як вказівник на масив з трьох вказівників того ж типу, що і `p2`. Використати інструкцію `typedef`. Написати програму, в якій викликаються функції з використанням цих вказівників, а у фунціях відображаються значення отриманих аргументів.

7. Написати програму, яка відображає значення, яке повертає функція `q`

```
double q = calculate (2.5, 10.4, add);
```

яка викликає функцію `add`

```
double add(double x, double y)
{
    return x + y;
}
```

8. Створити клас `box`, конструктору якого передається три значення типу `double`, які є довжинами сторін паралепіпеда. Класс `box` має підраховувати його об'єм і зберігати результат у вигляді значення `double`. Включити в клас метод `vol()`, який буде виводити на екран об'єм любого об'єкта типу `box`.

9. Використовуючи клас

```
#include <ctime>
```

```
class timer {
       clock_t start;
public:
        timer();
       ~timer();
};
```

і стандартну бібліотечну функцію `clock()`, яка повертає число часових тактів з моменту запуску програми створити клас `secmeter` для імітації секундоміра. Якщо поділити число часових тактів на число `CLOCKS_PER_SEC`, то можна отримати значення в секундах. Використати конструктор для початкового встановлення секундоміра в 0. Утворити дві функції-члени `start()` і `stop()` відповідно для запуску і зупинки секундоміру. Включити в класс функцію-член `show()` для виводу на екран величини проміжків часу, які пройшли. Використати деструктор для автоматичного виводу на екран часу, який пройшов з моменту створення об’єкту класу `secmeter`, до його вилучення.

9. Написати програму з використанняс функції, яка знаходить суму своїх аргументів та повертає результат через перший аргумент. Реалізувати варіанти функції, яка дозволяє звернення з різною кількістю аргументів:

- перевантаження функцій;
- функція з параметрами за замовчуванням;
- функція зі змінною кількістю параметрів `f(int &,…)`.

Результати функції повернути через вказівник та через посилання.

10. Написати програму для переписування менших від середнього арифметичного елементів одновимірного масиву дійсних чисел в інший масив. Виділити динамічну пам'ять для збереження масивів. Вхідні дані ввести з клавіатури, результат вивести на екран.

11. В області динамічної пам’яті визначити масив структур з даними про вироби підприємства: дата, код виробу, назва виробу, кількість виробів, ціна одного виробу. Знайти вартість продукції заданого коду, яка обчислюється як сума добутків кількості виробів на ціну одного виробу за всіма входженнями даного коду.

12. Визначити масив рядків символів в області динамічної пам’яті. Кожен рядок символів складається з полів фіксованої довжини: прізвище студента, номер залікової книжки, рейтинг за 100-бальною шкалою. Відсортувати рядки за спаданням рейтингу. Впорядкований масив вивести на екран.

2. Завдання для самостійної робота № 1

```
/* Розд. 12 1. listing 1  - введення-виведення даних типу int у C++*/
#include <iostream>
using namespace std;

int main()
{
  int i;
  cout << "Це вивід.\n";  // коментарій C++
  /* коментарій в стилі C */
  // введення числа за допомогою операції >>
  cout << "Введіть число: ";
  cin >> i;
  // тепер, вивести число за допомогою оператора <<
  cout << i << " в квадраті буде " << i*i << "\n";
  return 0;
}
>Це вивід.
Введіть число: 12
12 в квадраті буде 144

/* 2. listing 10 – введення-виведення даних типу float, double, char*/
#include <iostream>
using namespace std;

int main()
{
  float f;
  char str[80];
  double d;
  cout << "Введіть два числа з плаваючою крапкою: ";
  cin >> f >> d;
  cout << "Введіть символьний рядок: ";
  cin >> str;
  cout << f << " " << d << " " << str;
  return 0;
}
>Введіть два числа з плаваючою крапкою: 123.45 345.123
Введіть символьний рядок: abcdef
123.45 345.123 abcdef

/* 3. listing 13 - оголошенняя str перед використанням */
#include <iostream>
using namespace std;

int main()
{
  float f;
  double d;
  cout << "Введіть два числа з плаваючою крапкою: ";
  cin >> f >> d;
  cout << "Введіть символьний рядок: ";
  char str[80];  // str оголошений перед використанням
  cin >> str;
  cout << f << " " << d << " " << str;
  return 0;
}
>Введіть два числа з плаваючою крапкою: 123.45 345.123
Введіть символьний рядок: abcd efghi
123.45 345.123 abcd

/* 4. listing 27 – клас стек з методами init(), push(), pop() */
```

```
#include <iostream>
using namespace std;
#define SIZE 100

// Створення класу стек
class stack {
  int stck[SIZE];
  int tos;
public:
  void init();
  void push(int i);
  int pop();
};

void stack::init()
{
  tos = 0;
}

void stack::push(int i)
{
  if(tos==SIZE) {
    cout << "Стек повний.\n";
    return;
  }
  stck[tos] = i;
  tos++;
}

int stack::pop()
{
  if(tos==0) {
    cout << "Стек порожній.\n";
    return 0;
  }
  tos--;
  return stck[tos];
}
int main()
{
  stack stack1, stack2;  // створюємо два об’єкти типу stack

  stack1.init();
  stack2.init();

  stack1.push(1);
  stack2.push(55);

  stack1.push(2);
  stack2.push(66);

  cout << stack1.pop() << " ";
  cout << stack1.pop() << " ";
  cout << stack2.pop() << " ";
  cout << stack2.pop() << "\n";

  return 0;
}
```

>**2 1 66 55**

```
/* 5. listing 29 – перевантаження функції аргументами різних типів */
#include <iostream>
using namespace std;
// Функція abs перевантажена три рази
```

```
int abs(int i);
double abs(double d);
long abs(long l);

int main()
{
  cout << abs(-10) << "\n";
  cout << abs(-11.0) << "\n";
  cout << abs(-9L) << "\n";
  return 0;
}
int abs(int i)
{
  cout << "Функція abs() з аргументом типу int\n";
  return i<0 ? -i : i;
}
double abs(double d)
{
  cout << " Функція abs() з аргументом типу double\n";
  return d<0.0 ? -d : d;
}
long abs(long l)
{
  cout << " Функція abs() з аргументом типу long\n";
  return l<0 ? -l : l;
}
>10
Using double abs()
11
Using long abs()
9

/* 6. listing 30 - перевантаження функції аргументами різних типів */
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

void stradd(char *s1, char *s2);
void stradd(char *s1, int i);

int main()
{
  char str[80];
  strcpy(str, "Hello ");
  stradd(str, "there");
  cout << str << "\n";
  stradd(str, 100);
  cout << str << "\n";
  return 0;
}

// зчеплення двох символьних рядків
void stradd(char *s1, char *s2)
{
  strcat(s1, s2);
}
// зчеплення символьного рядка з "stringized" integer
void stradd(char *s1, int i)
{
  char temp[80];
  sprintf(temp, "%d", i);
  strcat(s1, temp);
}
```

```
>Hello there
Hello there100

/* 7. listing 33 – механізм успадкування. Створення класів house і school похідних від класу building */
#include <iostream>
using namespace std;

class building {
  int rooms;
  int floors;
  int area;
public:
  void set_rooms(int num);
  int get_rooms();
  void set_floors(int num);
  int get_floors();
  void set_area(int num);
  int get_area();
};

// клас house є похідним від класу building
class house : public building {
  int bedrooms;
  int baths;
public:
  void set_bedrooms(int num);
  int get_bedrooms();
  void set_baths(int num);
  int get_baths();
};

// клас school є похідним від класу building
class school : public building {
  int classrooms;
  int offices;
public:
  void set_classrooms(int num);
  int get_classrooms();
  void set_offices(int num);
  int get_offices();
};

void building::set_rooms(int num) {
  rooms = num;
}

void building::set_floors(int num) {
  floors = num;
}

void building::set_area(int num) {
  area = num;
}

int building::get_rooms() {
  return rooms;
}

int building::get_floors() {
  return floors;
}

int building::get_area() {
```

```
  return area;
}

void house::set_bedrooms(int num) {
  bedrooms = num;
}

void house::set_baths(int num) {
  baths = num;
}

int house::get_bedrooms() {
  return bedrooms;
}

int house::get_baths() {
  return baths;
}

void school::set_classrooms(int num) {
  classrooms = num;
}

void school::set_offices(int num) {
  offices = num;
}

int school::get_classrooms() {
  return classrooms;
}

int school::get_offices() {
  return offices;
}

int main()
{
  house h;
  school s;

  h.set_rooms(12);
  h.set_floors(3);
  h.set_area(4500);
  h.set_bedrooms(5);
  h.set_baths(3);

  cout << "В будинку " << h.get_bedrooms();
  cout << " спальних кімнат\n";

  s.set_rooms(200);
  s.set_classrooms(180);
  s.set_offices(5);
  s.set_area(25000);

  cout << "В школі " << s.get_classrooms();
  cout << " класних кімнат\n";
  cout << "Її площа дорівнює " << s.get_area();

  return 0;
}
```

>В будинку 5 спальних кімнат
В школі 180 класних кімнат
Її площа дорівнює 25000

```
/* 8. Клас з конструктором і деструктором */
#include <iostream>

using namespace std;
class myclass {
private:
    int a;
public:
      myclass();   // конструтор
      ~myclass();  // деструктор
      void show();
};

myclass:: myclass() {
    cout << ”Вміст конструктора\n”;
    a=10;
}

myclass::~myclass() {
    cout << ”Деструктор\n”;
}

void myclass::show() {
   cout << ”a=” << a << ”\n”;
}

int main() {
    myclass obj;
    ob.show();
    return 0;
}

/* 9. Конструктор і деструтор з таймером */
#incluse <iostream>
#include <ctime>
using namespace std;

class timer {
privet:
    clock_t start;
public:
    timer();
    ~timer();
};

timer::timer()
{
    start=clock();
}

timer::~timer()
{
    clock_t end;
    end=clock();
    cout << ”Затрачений час: ” << (end-start)/CLOCKS_PER_SEC << ”\n”;
}

int main() {
    timer obj;
    char c;
    cout << ”Натисність любу клавішу, а потім Enter: “;
    cin >> c;
    return 0;
}
```

/* 10. Використання об'єднання для побайтного виведення значення типу double у двійковому поданні */

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

union bits {
     bits(double n);
     void show_bits();
     double d;
     unsigned char c[sizeof(double)];
};

bits::bits(double n); {
    d=n;
}

void bits::show_bits() {
    int i, j;
    for(j=sizeof(double)-1; j>=0; j--); {
        cout << "Двійкове подання байту" << j << ":";
        for(i=128;i;i>=1)
            if(i & c)[j]) cout << "1";
            else cout << "0";
        cout << "\n";
    }
}

int main() {
    bits ob(1991.829);
    ob.show();
    return 0;
}
```

/* 11 – анонімне об'єднання */

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

int main() {
    union {
       unsigned char bytes[8];
       double value;
    };

   int I;
   value=859345.324;
   // побайтне виведення тиру double
   for(i=0;i<8;i++) cout << (int) bytes[i] << " ";

   return 0;
}
```

/* 12. listing 37 – клас стек з використання конструкторів і деструкторів */

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

#define SIZE 100

// Створюється клас стек
class stack {
  int stck[SIZE];
  int tos;
public:
  stack();  // constructor
  ~stack(); // destructor
```

```
  void push(int i);
  int pop();
};

// конструктор
stack::stack() {
  tos = 0;
  cout << "Стек ініціалізований\n";
}

// деструктор
stack::~stack() {
  cout << "Стек знищений\n";
}

void stack::push(int i)
{
  if(tos==SIZE) {
    cout << "Стек повний\n";
    return;
  }
  stck[tos] = i;
  tos++;
}

int stack::pop() {
  if(tos==0) {
    cout << "Стек порожній\n";
    return 0;
  }
  tos--;
  return stck[tos];
}

int main()
{
  stack a, b;  // створюємо два об'єкти stack

  a.push(1);
  b.push(55);
  a.push(2);
  b.push(66);

  cout << a.pop() << " ";
  cout << a.pop() << " ";
  cout << b.pop() << " ";
  cout << b.pop() << "\n";

  return 0;
}
```

```
>2 1 66 55
Стек ініціалізований
Стек ініціалізований
2 1 66 55
Стек знищений
Стек знищений
```

Лабораторна робота № 3. Класи, члени класу, робота з класами

Мета роботи: – вивчення класів, конструкторів і деструкторів, дружніх функцій і дружніх класів, вбудованих функцій, статичних і константних членів класу.

Теоретичний матеріал лекції, [1, розділ 2, 3], [2, розділ 12], [3, розділ . 10, 11, 12].

1. Короткі теоретичні відомості

Дані класу можуть бути:

- звичайними, без кваліфікаторів (`int x`);
- статичними (`static int x`);
- константними (`const int x`).

Статичні поля оголошуються за допомогою слова `static` і мають бути визначені поза класом на глобальному рівні оголошень (незалежно від їх розміщення у класі). Статичні дані зберігаються разом з глобальними, окремо від об'єкта класу.

Статичні дані є спільними для всіх об'єктів цього класу та успадкованих від нього об'єктів. За допомогою статичних полів можна передавати інформацію всім об'єктам цього класу. Статичні поля існують навіть тоді, коли не оголошений жоден об'єкт цього класу.

```
class A {
public:
static int x;     // оголошення статичного елемента даних
          int y;
};
int A::x=5;       // визначення статичного елемента даних
```

Статичні поля можуть бути ініціалізовані також конструктором у його тілі (всередині блоку {}), але не у списку ініціалізації (після символу ':'). Особливістю такого способу ініціалізації є те, що при кожному створенні нового об'єкта змінюватимуться статичні поля у всіх об'єктів класу.

Константні дані оголошуються за допомогою слова `const`. Ініціалізуються константні дані після двокрапки у списку ініціалізації конструктора. Константні дані не можна змінити у програмі. У класі також можна визначити символьні константи перелічимого типу:

```
class A {
enum (DAY=30, MOHTH=3, YEAR=2015};
…
};
```

1.1. Методи класу

Методи (функції-члени), оголошені у протоколі класу, можуть бути:

- членами класу;
- друзями класу.

Методи та друзі мають доступ до усіх частин класу. Методи допускають перевантаження. Перевантажені методи мають однакові імена, але повинні відрізнятися кількістю параметрів або їх типами. Методи викликаються у межах класу безпосередньо, за їх іменами. Поза класом вони викликаються за допомогою об'єкта класу, посилання або вказівника на клас.

Якщо метод повертає об'єкт, посилання або вказівник на об'єкт, то можливий ланцюжковий виклик методів.

Кожний нестатичний метод класу має доступ до об'єкта, з якого він викликаний, через вказівник `this`. За допомогою цього вказівника можна отримати доступ до всіх елементів класу(даних та методів):

```
class Base {
int x;
public:
```

```
void print () { cout << x << ' ' << (*this).x << ' ' << this->x << endl; }
};
```

Розіменований вказівник `this` використовується для повернення методом класу посилання на об'єкт свого класу:

```
class Base {
int x;
public:
Base & Copy(const Base& a) { x=a.x; return *this; }
};
```

1.2. Статичні та константні методи

Статичний метод оголошується за допомогою слова `static`, яке записується перед заголовком функції. Статичний метод може бути викликаний незалежно від існування об'єкта класу. У цьому випадку він викликається з іменем класу, наприклад `Base::f1()` з врахуванням рівнів захисту елементів класу.

Статичному методу не передається вказівник `this`, тому він не може звертатися до нестатичних елементів класу безпосередньо, за їх іменами. Однак, статичний метод може звертатися до нестатичних елементів класу через посередник - об'єкт, посилання або вказівник на об'єкт класу:

```
class A {
   int x;
public:
   A(int x=0) { this->x=x; }
   static void print() {
   A a(7);
     cout << a.x << endl;
   }
};
void main() {
   A::print();
}
```

Константний метод оголошується за допомогою слова `const`, яке записується після його заголовка. Якщо константний метод визначається за межами класу, то слово `const` повинно зберігатися в кінці заголовка функції. Константний метод не може змінювати дані класу. Статичний метод не може бути константним та навпаки.

```
// Програма не компілюється!
#include <iostream>
using namespace std;

class Demo {
  int i;
public:
  int geti() const {
    return i; // ok
  }

  void seti(int x) const {
    i = x; // помилка
  }
};

int main()
{
  Demo ob;

  ob.seti(1900);
  cout << ob.geti();
```

```
  return 0;
}
```

Іноді виникає необхідність модифікувати той або інший член класу за допомогою константної функції-члена, зберігаючи у недоторканості іншу частину. Для цього призначене ключове слово `mutable`. Воно відміняє атрибут `const`. Тобто член, оголошений з атрибутом `mutable`, може модифікуватися константною функцією-членом. У прикладі змінна і оголошена з атрибутом `mutable`, тому функція `seti()` може її модифікувати.

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

class Demo {
  mutable int i;
  int j;
public:
  int geti() const {
    return i; // все вірно
  }

  void seti(int x) const {
    i = x; // тепер все вірно
  }

// Ця функція не компілюється, так як j не mytable
//  void setj(int x) const {
//    j = x; // помилка
//  }
};

int main() {
  Demo ob;
  ob.seti(1900);
  cout << ob.geti();
  return 0;
}
```

1900

1.3. Вказівник на дані класу

У програмі може оголошуватися *вказівник на дані класу*:

```
тип ім'я_класу::* ім'я_вказівника=&імя_класу::ім'я_поля;
```

Наприклад:

```
class Base {
   public:
      int x;
      const int y;
      static int z;
      Base(int x1=0, int y1=0) : y(y1) { x=x1; }
};
int Base::z=3;
int Base::*p1 = &Base::x;
```

Для використання вказівника використовуються операції `.*, ->*`, наприклад

```
Base a;
a.*p1=5;   // a.x=5
Base *q=new Base;
q->*p1=5;
```

1.4. Вказівник на метод класу

Вказівник на метод класу можна використовувати для безпосереднього виклику методу або як аргумент чи тип результату іншої функції.

Оголошення та ініціалізація вказівника на нестатичний метод класу:

```
тип_функції (ім'я_класу::*ім'я_вказівника) (типи параметрів) =
&ім'я_класу::ім'я_методу;
```

Вказівник використовується за допомогою операції `.*` або `->*`, наприклад

```
class Base {
   int x;
public:
   Base(int x) {x=y;}
   int GetX() { return x; }
};
int (Base::*p) () = &Base::GetX;
void main() {
   Base a(5);
   Cout << (a.*p)() << endl;  // або a.GetX();
   Base *q=&a;
   Cout << (q->*p)() << endl;
}
```

1.5. Конструктори і деструктори

Конструктори розрізняють за призначенням:

- ініціалізації;
- копіювання;
- перетворення типів.

1.6. Конструктор ініціалізації

Перед використання обєктів їх необхідно проініціалізувати. У С++ автоматична ініціалізація обєктів здійснюється конструктором ініціалізації. *Конструктор* – це особлива функція, член класу, ім'я якої співпадає з іменем класу. Конструктори не повертають значення.

Конструктори ініціалізації викликаються:

- при створенні нового об'єкту у сегменті даних, стеку, динамічній пам'яті та ініціалізації його полів;

- для створення локального об'єкта у виразі:

```
тип_класу(список параметрів конструктора);
```

Приклади створення об'єктів, коли викликається конструктор ініціалізації:

```
A a(1,2);
A *p=new A(1,2);
A& q=*new A(1,2);
```

Якщо конструктор має тільки один аргкмент, то для ініціалізації об'єкту `obj` можна застосувати вираз `obj(x)` або `obj=x`. При використання виразу `obj=x` неявно створюється функція перетворення `obj(x)`. Іноді таке перетворення є небажаним. У таких випадках використовується ключове слово `explicit`, яке допускає тільки один спосіб виклику конструктора `obj(x)`.

```
// явний (explicit) конструктор
#include <iostream>
using namespace std;

class myclass {
  int a;
public:
// myclass(int x) { a = x; }
  explicit myclass (int x) { a=x; }  // явний конструктор без перетворень
  int geta() { return a; }
};
int main() {
//  myclass ob = 4; // автоматично перетворюється у вираз myclass(4)
```

```
  myclass ob(4);  // конструктор без перетворення
  cout << ob.geta();
  return 0;
}
```
4

Звичайно члени класу ініціалізуються всередині конструкторів.

// ініціалізація членів класу всередині конструктора
```
#include <iostream>
using namespace std;

class MyClass {
  int numA;
  int numB;
public:
// ініціалізація зі звичайним синтаксисом
  MyClass(int x, int y) {
    numA = x;
    numB = y;
  }
  int getNumA() { return numA; }
  int getNumB() { return numB; }
};

int main() {
  MyClass ob1(7, 9), ob2(5, 2);
  cout << "Значення в об'єкті ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в об'єкті ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
  return 0;
}
```
Значення в об'єкті ob1 є 9 і 7
Значення в об'єкті ob2 є 2 і 5

Але такий не можна застосувати коли дані-члени оголошені константними, коли необхідно проініціалізувати посилання, і якщо у класі не передбачений конструктор за замовчуванням. У цих випадках використовується альтернативний спосіб ініціалізації членів класу при створенні об’єкту:

```
    Конструктор(список_аргументів) : член1(ініціалізатор),
                                     . . .
                                     членN(ініціалізатор)
    { //тіло конструктора }
```

// ініціалізація константних членів
```
#include <iostream>
using namespace std;

class MyClass {
  const int numA; // константний член
  const int numB; // константний член
public:
  // ініціалізація  numA і numB використовуючи список ініціалізації
  MyClass(int x, int y) : numA(x), numB(y) { }
  int getNumA() { return numA; }
  int getNumB() { return numB; }
};

int main() {
  MyClass ob1(7, 9), ob2(5, 2);
  cout << "Значення в ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
```

```
  return 0;
}
```
Значення в ob1 є 9 і 7
Значення в ob2 є 2 і 5

Конструктори із списком ініціалізації використовуються, якщо клас має тип, для якого не передбачений конструктор за замовчуванням.

```
// клас, який не має конструктора за замовчуванням
// Програма не компілюється!
#include <iostream>
using namespace std;

class IntPair {
public:
  int a;
  int b;
  IntPair(int i, int j) : a(i), b(j) { }
};

class MyClass {
  IntPair nums; // Клас IntPair не має конструктора без параметрів!
public:
   MyClass(int x, int y) {
    nums.a = x;
    nums.b = y;
  }
  int getNumA() { return nums.a; }
  int getNumB() { return nums.b; }
};

int main() {
  MyClass ob1(7, 9), ob2(5, 2);
  cout << "Значення в ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
  return 0;
}
```

```
// конструктор зі списком ініціалізації
#include <iostream>
using namespace std;

class IntPair {
public:
  int a;
  int b;
  IntPair(int i, int j) : a(i), b(j) { }
};

class MyClass {
  IntPair nums; // now OK
public:
  // Застосування списка ініціалізації
  // відкриває доступ до членів класу (також при відсутності source коду класу)
  MyClass(int x, int y) : nums(x,y) { }
  int getNumA() { return nums.a; }
  int getNumB() { return nums.b; }
};

int main()
{
  MyClass ob1(7, 9), ob2(5, 2);
```

```
  cout << "Значення в ob1 є " << ob1.getNumB() <<
          " і " << ob1.getNumA() << endl;
  cout << "Значення в ob2 є " << ob2.getNumB() <<
          " і " << ob2.getNumA() << endl;
  return 0;
}
```

Значення в ob1 є 9 і 7
Значення в ob2 є 2 і 5

1.7. Конструктор копіювання

Конструктор копіювання:

- має один параметр з посиланням на тип класу: `A&` або `const A&`. Може мати більше одного параметра, які задаються за замовчуванням;
- якщо не визначений у класі, то конструктор копіювання генерується автоматично і виконує порозрядне копіювання об'єктів;
- виконує копіювання усіх видів полів, зокрема константних та з типом посилання;
- якщо поля класу мають тип вказівника або посилання на будь-який тип, то конструктор копіювання повинен бути визначений користувачем.

Конструктор копіювання викликається при:

- створенні нового об'єкту та його ініціалізації вже існуючим об'єктом цього ж класу (новий об'єкт може бути створений у сегменті даних, стеку, динамічній пам'яті, може бути глобальним або локальним);
- передачі об'єктів через список параметрів функції "за значенням" (не через вказівник або посилання);
- поверненні функціям локальних об'єктів класу "за значенням " (не через вказівник або посилання).

Приклад оголошення та визначення конструктора копіювання:

```
clas A {
     const int x;
     int& y;
     int z;
// визначення конструктора ініціалізації
    A(int x1=0, int y1=0, int z1=0) : x(x1), y(y1) { z=z1; }
// оголошення конструктора копіювання
    A(const A&);
};
// визначення конструктора копіювання
A::A(const A& a) : x(a.x), y(a.y)  { z=a.z; }
```

Приклади створення об'єктів, коли викликається конструктор копіювання:

```
A a;
A b(a);          // A b=a;
A *p=new A(a);
A& q=*new A(a);
```

Якщо у протоколі класу є поля з типом вказівника або посилання на будь-який тип, то для такого класу необхідно визначити власний конструктор, який здійснює глибоке копіювання:

```
class A {
   int *p;
   public:
A(int x) { p=new int(x); }          // звичайний конструктор
A(const A& a) { p=new int(*a.p); } // конструктор копіювання
~A(){ delete p; }
int  Get(void)  { return *p; }
void Set(int x) { *p=x; }
};
void main() {
   Int x=5;
   A a(x);
   A b(a);
```

```
cout << a.Get() << ” ” << b.Get() << endl;
}
```

Конструктор копіювання закріплює за вказівником `p` нову область динамічної пам’яті, тому `a.p != b.p`.

1.8. Конструктор перетворення типів

Конструктор перетворення типів призначений для перетворення значення заданого типу до типу класу. Має один параметр з типом, який вимагає перетворення, або декілька параметрів, значення яких задаються за замовчуванням. Наприклад:

```
clas A {
    int x;
public:
// конструктори перетворення типів
    A(int x) { this->x=x; }
    A(double x) { this->x=int(x); }
};
```

Конструктор перетворення типів викликається у таких випадках:

- для ініціалізації об’єкта класу значенням, тип якого є відмінним від типу цього класу;
- для передавання параметрів у функцію, коли формальний параметр має класовий тип, а фактичний - інший тип;
- для повернення значення функцією, коли функція має класовий тип, а вираз оператора `return` має інший тип.

1.9. Деструктор

Деструктор призначений для звільнення пам’яті, відведеної під поля об’єкту. Деструктор не перевантажується і не успадковується. Деструктор викликається автоматично, якщо об’єкт виходить із області досяжності програми. Пам'ять, відведена у стеку для локального об’єкта з класом пам’яті “автоматичний” (`auto`), буде звільнена під час завершення роботи функції. Пам'ять статичного локального об’єкту буде звільнена перед завершенням роботи функції `main()`. Якщо локальний об’єкт розміщено в області динамічної пам’яті, то при завершенні роботи функції деструктор не викликається. Всю незвільнену динамічну пам'ять буде автоматично звільнено після завершення роботи програми.

Деструктор можна викликати явно:

```
Class A {
    Int *p;
Public:
A(int x) { p=new int(x); }
~A() { delete p; }
…
};
Void main() {
A a(5);
….
b->!A();
a.!A();
}
```

Запитання.

1. Призначення дружньої функції і як вона описується в класі. Чи успадковується дружня функція.
2. Що таке дружній клас і який він має доступ до базового класу.
3. Що таке вбудовувана функція. В чому її недоліки і переваги.
4. Як використовуються статичні члени класу (дані і методи).
4. Що таке вказівники на дані і функції класу, для чого вони використовуються.

5. Константні члени класу (дані і методи). Атрибут `mutable`.

6. Що таке вказівник `this` і коли він використовується.

7. Що таке конструктор і деструктор. Коли вони викликаються.

8. Для чого і як конструкторам передаються параметри. Як ініціалізувати дані за допомогою конструктора.

9. Конструктор ініціалізації. Особливості використання конструкторів ініціалізації.

10. Що таке конструктор копіювання і коли він викликається.

11. Що таке конструктор перетворення типів і коли він викликається.

12. Оголошення, автоматичний та явний виклик деструктора.

Завдання.

1. Задано неповне визначення класу

```
class strtype {
   char *p;
   int len;
public:
   char *getstring() { return p; }
   int getlength() { return len; }
```

Добавити в це визначення два конструктори. В першому не має бути параметрів. Він має виділяти 255 байтів пам'яті оператором *new*, ініціалізувати цю пам'ять нульовою стрічкою і встановити змінну *len* рівною 255. В другому конструкторі має бути два параметри. Перший – це стрічка, яка використовується при ініціалізації, другий – число байтів, які необхідно виділити. Показати в програмі роботу цих конструкторів.

2. Конструктор класу А виділяє в динамічній пам'яті масив int[80]. Добавити в клас конструктор копіювання. Створити об'єкт класу А і зробити його копію В. Проініціалізувати масиви обох об'єктів різними числами і вивести їх значення.

3. Виконати завдання згідно номеру студента у журналі групи.

3.1. Створити клас ДАТИ з полями у закритій частині: день (1-31), місяць (1-12), рік (ціле число). Клас має конструктор, методи встановлення дня, місяця і року, методи отримання значень дня місяця року, а також методи виведення за шаблонами "12 лютого 2015" і 12.02.2015. Методи встановлення полів класу повинні перевіряти коректність параметрів, що задаються.

3.2. Створити клас МАТРИЦІ, який у закритій частині містить вказівник на `int`, кількість рядків, стовпців та змінну стану. Визначити конструктор без параметрів, конструктор з одним параметром, та конструктор з двома параметрами, деструктор. Визначити метод для повернення значення елемента за індексами `[i][j]`. Визначити функцію виведення матриці. Визначити функції додавання, віднімання та множення матриць. Визначити функції множення матриці на число. Перевірити роботу цього класу. У випадку нестачі пам'яті, невідповідності розмірностей, виходу за межі масиву встановлювати код помилки у змінній стану.

3.3. Створити клас ЧАСУ з полями у закритій частині: година (0-23), хвилини (0-59), секунди (0-59). Клас має конструктор, методи встановлення часу, методи отримання годин, хвилин, і секунд, а також два методи виведення за шаблонами "16 годин 18 хвилин 3 секунди" і "4 p.m. 18 хвилин 3 секунди". Методи встановлення полів класу повинні перевіряти коректність параметрів, що задаються.

3.4. Створити клас ПРЯМОКУТНИК. У закритій частині визначити поля - висоту і ширину. Метод класу обчислюють площу, периметр, встановлюють поля даних і повертають їхні значення. У методах встановлення значень полів класу необхідно перевіряти коректність заданих параметрів. Визначити функцію виведення елементів класу.

3.5. Створити клас ОДНОЗВЯЗНИЙ СПИСОК. Методи класу додають елемент до списку, вилучають елементи зі списку, сортують список, відображають елементи списку від початку до кінця. Вилучити заданий елемент списку.

3.6. Створити клас ДВОЗВЯЗНИЙ СПИСОК. Методи класу додають елемент до списку,

вилучають елементи зі списку, сортують список, відображають елементи списку від початку до кінця. Знайти у списку заданий елемент.

3.7. Створити клас КАЛЕНДАР. У закритій частині визначити дані - день, місяць, рік. Визначити необхідні конструктори, деструктори та методи. Методи класу встановлюють та зчитують значення полів даних, визначають назву тижня за заданою датою, виводять результат на екран. Ввести дату та визначити назву дня свого дня народження.

3.8. Створити клас ДАТА з полями у закритій частині: день (1-31), місяць (-12), рік (ціле число). Клас має конструктор, методи встановлення дня, місяця і року, методи читання дня, місяці і року. У методах встановлення полів класу необхідно перевіряти коректність заданих параметрів. Визначити метод, який збільшує значення дати на 1 день.

3.9. Створити клас ЧАС з полями у закритій частині: година(0-23), хвилини (0-59), секунди (0-59). Клас має конструктор, методи встановлення часу, методи читання години, хвилини і секунди. У методах встановлення полів класу необхідно перевіряти коректність параметрів, що задаються. Розробити метод для збільшення значення часу на 1 секунду, 1 хвилину, 1 годину.

3.10. Створити клас КВАДРАТ з полями у закритій частині: координати головної діагоналі. Методи класу обчислюють довжину сторони квадрату, площу, периметр, встановлюють значення полів і повертають їх значення. Методи встановлення полів класу повинні перевіряти коректність параметрів, що задаються.

3.11. Створити клас СТЕК. Розробити методи класу для введення елемента у стек та вилучення елемента зі стека. Розробити функцію, яка визначає кількість елементів у стеку.

3.12. Розробити клас квадратна матриця. У закритій частині визначити дані: порядок матриці та вказівник на її початок в області динамічної пам’яті. Клас має конструктор та деструктор. Розробити методи класу, які виконують такі операції: встановлення та виведення значень елементів матриці, визначення сліду матриці (суми елементів головної діагоналі), суми елементів вище та нижче головної діагоналі.

3.13. Розробити клас МНОЖИНА ЦІЛИХ ЧИСЕЛ. У закритій частині визначити вказівник на цілий тип (на елементи множини). Визначити конструктори, деструктор та методи створення множини, виведення вмісту множини, об’єднання, різниці та перетину множин.

3.14. Розробити клас РЯДОК СИМВОЛІВ. У закритій частині визначити вказівник на символьний тип (на початок рядка). Визначити конструктори, деструктор та методи введення-виведення рядка, конкатенації, порівняння рядків, перевірки входження підрядка у рядок.

3.15. Розробити клас СТУДЕНТ. У закритій частині визначити дані - прізвище, номер залікової книжки, оцінки з предметів. Визначити конструктори, деструктор та методи встановлення і читання значень полів даних, знаходження середнього балу, визначення кількості незадовільних оцінок.

3.16. Розробити клас КНИГА. У закритій частині визначити дані - автор, назва, видавництво, рік видання. Визначити конструктори, деструктор та методи встановлення і читання значень полів даних, визначення відповідності книги пошуковим критеріям.

3.17. Цифровий лічильник - це змінна з обмеженим діапазоном, яка скидається у початкове значення, коли її цілочисельне значення досягає визначного максимуму. Опишіть клас такого ЛІЧИЛЬНИКА. Забезпечте можливість встановлення максимального і мінімального значень, збільшення значень лічильника на 1, повернення поточного значення.

3.18. Створити клас для виконання арифметичних операцій над ЦІЛИМИ ЧИСЛАМИ у двійковій системі числення. Числа задаються як стрічка символів в області динамічної пам’яті. Визначити конструктори, деструктор, методи виконання операцій, введення чисел та виведення результату.

3.19. Створити клас для виконання арифметичних операцій над ЦІЛИМИ ЧИСЛАМИ у шістнадцятковій системі числення. Числа задаються як стрічка символів в області динамічної пам’яті. Визначити конструктори, деструктор, методи виконання операцій, введення чисел та виведення результату.

3.20. Розробити клас, який виконує статистичну обробку ТЕКСТОВОГО ФАЙЛУ - підрахунок кількості символів, слів, речень. Визначити необхідні конструктори, деструктор та методи роботи з файлом.

3.21. Розробити клас для роботи зі СЛОВНИКОМ, який складається з масиву слів та їх перекладу іншою мовою. Визначити конструктори, деструктор та методи для додавання нових слів, пошуку слів, злиття двох словників без повторень елементів.

3.22. Розробити клас для роботи з КАТОТЕКОЮ КНИГ. Клас містить дані про назви книги: автор, назва книги, видавництво, рік видання. Реалізувати методи додавання даних про книгу до картотеки, витирання з картотеки. Пошук книг за прізвищем автора, назвою книги, роком видання.

3.23. Розробити клас для роботи з ПРАЙС АРКУШЕМ комп’ютерної фірми, у якому вказано дані про марку комп’ютера, тип процесора, частоту процесора, об’єм оперативної та дискової пам’яті, характеристики відеокарти, ціну комп’ютера.

3.24. Створити клас для створення ДІЛОВОГО ЗАПИСНИКА з планом роботи на тиждень. Визначити конструктори, деструктор, методи роботи із записником - додавання нових записів, корегування та витирання записів.

3.25. Створити клас для роботи з ЧЕРГОЮ мешканців міста, які потребують покращення житлових умов. У закритій частині класу визначити поля даних - номер черги, прізвище та ініціали, склад сім’ї, дата поставлення у чергу. У відкритій частині класу визначити методи для поставлення та вилучення з черги, пошуку за номером черги, прізвищу, дані.

3.26. Розробити клас для роботи з ЧЕРГОЮ на біржі праці. У закритій частині класу визначити необхідні дані: прізвище, паспортні дані, освіта, професія, дата поставлення на облік, бажаний вид зайнятості, бажана зарплата. Визначити методи для роботи з цими даними - додавання, вилучення, корегування записів, виведення статистики.

2. Завдання для самостійної роботи № 2

```
/* 1. chapter 12, listing 1 – чередування специфікацій доступу public, private в класі */
#include <iostream>
#include <cstring>
using namespace std;

class employee {
  char name[80]; // private – закриті по замовчуванню
public:
  void putname(char *n); // public – відкриті члени
  void getname(char *n);
private:
  double wage; // знов, private
public:
  void putwage(double w); // знов public
  double getwage();
};

void employee::putname(char *n) {
  strcpy(name, n);
}

void employee::getname(char *n) {
  strcpy(n, name);
}

void employee::putwage(double w) {
  wage = w;
}

double employee::getwage() {
  return wage;
}

int main() {
  employee ted;
  char name[80];

  ted.putname("Ted Jones");
  ted.putwage(75000);

  ted.getname(name);
  cout << name << " makes $";
  cout << ted.getwage() << " per year.";

  return 0;
}
>Ted Jones makes $75000 per year.

/* 2. listing 3 – доступ до відкритих членів класу */
#include <iostream>
using namespace std;

class myclass {
public:
  int i, j, k; // доступні з любої точки програми
};

int main() {
  myclass a, b;
```

```
  a.i = 100; // доступ до i, j, k є
  a.j = 4;
  a.k = a.i * a.j;

  b.k = 12; // a.k i b.k належать різним об'єктам
  cout << a.k << " " << b.k;

  return 0;
}
```
>**400 12**

```
/* 3. listing 4 - використання структури як класу */
#include <iostream>
#include <cstring>
using namespace std;

struct mystr {
  void buildstr(char *s);  // відкритий член
  void showstr();
private:
  char str[255];           // закритий член
};

void mystr::buildstr(char *s) {
  if(!*s) *str = '\0'; // ініціалізація рядка
  else strcat(str, s);
}

void mystr::showstr() {
  cout << str << "\n";
}

int main() {
  mystr s;
  s.buildstr(""); // ініціалізація
  s.buildstr("Привіт ");
  s.buildstr("всім!");
  s.showstr();
  return 0;
}
```
>**Привіт всім!**

```
/* listing 6 – union, об'єднання в C++ можуть містити не тільки дані, але і
   функції */
#include <iostream>
using namespace std;

union swap_byte {
  void swap();
  void set_byte(unsigned short i);
  void show_word();

  unsigned short u;
  unsigned char c[2];
};

void swap_byte::swap() {
  unsigned char t;

  t = c[0];
  c[0] = c[1];
  c[1] = t;
}
```

```
void swap_byte::show_word() {
  cout << u;
}

void swap_byte::set_byte(unsigned short i) {
  u = i;
}

int main() {
  swap_byte b;

  b.set_byte(1);
  b.swap();
  b.show_word();

  return 0;
}
```
>**256**

/* 4. listing 7 – union, об'єднання без імен */
```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  // визначення безіменного union
  union {
    long l;
    double d;  char s[10];
  } ;

  // тепер, відкритий прямий доступ до елементів об'єднання без оператора "."
  l=1;
  cout << l << " ";
  d=2.22;
  cout << d << " ";
  strcpy(s,"123456789a");
  cout << s << "\n";
  return 0;
}
```
>**1 2.22 123456789a**

/* 5. listing 8 - за допомогою ключового слова friend можна надати звичайній функції доступ до закритих членів класу */
```
#include <iostream>
using namespace std;

class myclass {
  int a, b;
public:
  friend int sum(myclass x);
  void set_ab(int i, int j);
};

void myclass::set_ab(int i, int j) {
  a = i;
  b = j;
}

// Увага: sum() не є функцією членом класу
int sum(myclass x) {
  /* Так як() є дружньою по відношенню до класу myclass, вона має прямий доступ до
a і b */
```

```
  return x.a + x.b;
}

int main() {
  myclass n;
  n.set_ab(3, 4);
  cout << sum(n);
  return 0;
}
>7
```

```
/* 6. listing 9 – дружня функція по відношенню до двох класів */
#include <iostream>
using namespace std;

const int IDLE = 0;
const int INUSE = 1;

class C2;
// неповне оголошення, так як в класі C1 дружня функція idle має аргумент C2

class C1 {
  int status;
// дорівнює IDLE якщо екран вільний, і INUSE, якщо екран зайнятий
public:
  void set_status(int state);
  friend int idle(C1 a, C2 b);
};

class C2 {
  int status;
// дорівнює IDLE якщо екран зайнятий, і INUSE якщо екран вільний
public:
  void set_status(int state);
  friend int idle(C1 a, C2 b);
};

void C1::set_status(int state) {
  status = state;
}

void C2::set_status(int state) {
  status = state;
}

int idle(C1 a, C2 b) {
  if(a.status || b.status) return 0;
  else return 1;
}

int main() {
  C1 x;
  C2 y;

  x.set_status(IDLE);
  y.set_status(IDLE);

  if(idle(x, y)) cout << "Екран вільний.\n";
  else cout << "Зайнятий.\n";

  x.set_status(INUSE);

  if(idle(x, y)) cout << " Екран вільний.\n";
  else cout << " Зайнятий.\n";
```

```
  return 0;
}
>Екран вільний.
Зайнятий.

/* 7. listing 10 – дружня функція може бути членом іншого класу */
#include <iostream>
using namespace std;

const int IDLE = 0;
const int INUSE = 1;

class C2;  // forward declaration – неповне описання

class C1 {
  int status;
// дорівнює IDLE, якщо екран зайнятий, і INUSE, якщо екран вільний
  // ...
public:
  void set_status(int state);
  int idle(C2 b);  // тепер функція idle – член класу C1
};

class C2 {
  int status;
// дорівнює IDLE, якщо екран зайнятий, і INUSE, якщо відьний
  // ...
public:
  void set_status(int state);
  friend int C1::idle(C2 b);  // дружня функція idle

};

void C1::set_status(int state) {
  status = state;
}

void C2::set_status(int state) {
  status = state;
}

// Функція idle() є членом класу C1, але другом класу C2
int C1::idle(C2 b) {
  if(status || b.status) return 0;
  else return 1;
}

int main() {
  C1 x;
  C2 y;
  x.set_status(IDLE);
  y.set_status(IDLE);
  if(x.idle(y)) cout << x.idle(y) << "Екран вільний.\n";
  else cout << "Зайнятий.\n";
  x.set_status(INUSE);
  if(x.idle(y)) cout << x.idle(y) << " Екран вільний.\n";
  else cout << "Зайнятий.\n";
  return 0;
}
>Екран вільний.
Зайнятий.

/* 8. listing 11 – дружні класи */
```

```
// Один клас може бути дружнім по відношенню до іншого. У цьому випадку
// дружній клас і всі його члени-функції мають доступ до закритих членів,
// визначених у другому класі.
#include <iostream>
using namespace std;

class TwoValues {
  int a;
  int b;
public:
  TwoValues(int i, int j) { a = i; b = j; }
  friend class Min;    // дружній клас
};

class Min {
public:
  int min(TwoValues x);
};

int Min::min(TwoValues x) {
  return x.a < x.b ? x.a : x.b;
}

int main() {
  TwoValues ob(10, 20);
  Min m;
  cout << m.min(ob);
  return 0;
}
```
>**10**

/* 9. listing 12 – функції, які підставляються, а не викликаються. Ключове слово inline */

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

inline int max(int a, int b) {
  return a>b ? a : b;
}
int main() {
  cout << max(10, 20);           // cout << (10>20) ? 10 : 20);
  cout << " " << max(99, 88);    // cout << ” ” << (99>88 ? 99 : 88);
  return 0;
}
```
>**20 99**

/* 10. listing 13 – програма з точки зору компілятора після підставлення */

```
#include <iostream>
using namespace std;
int main() {
  cout << (10>20 ? 10 : 20);
  cout << " " << (99>88 ? 99 : 88);
  return 0;
}
```
>**20 99**

/* 11. listing 14 – підставляємі функції, можуть бути членами класу */

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

class myclass {
  int a, b;
public:
  void init(int i, int j);
```

```
  void show();
};

// Створення inline функції.
inline void myclass::init(int i, int j) {
  a = i;
  b = j;
}
// Створити іншу inline функцію.
inline void myclass::show() {
  cout << a << " " << b << "\n";
}

int main() {
  myclass x;
  x.init(10, 20);
  x.show();
  return 0;
}
>10 20
```

/* 12. listing 15 – короткі функції можна визначити всередині класу і вони перетворюються в inline функції*/

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

class myclass {
  int a, b;
public:
  // функції автоматично inline
  void init(int i, int j) { a=i; b=j; }
  void show() { cout << a << " " << b << "\n"; }
};

int main() {
  myclass x;
  x.init(10, 20);
  x.show();
  return 0;
}
>10 20
```

/* 13. listing 17 – конструктори з параметрами */

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

class myclass {
  int a, b;
public:
  myclass(int i, int j) {a=i; b=j;}
  void show() {cout << a << " " << b;}
};

int main() {
  myclass ob(3, 5);
  ob.show();
  return 0;
}
>3 5
```

/* 14. listing 20 – конструктор з параметрами */

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

```
const int IN = 1;
const int CHECKED_OUT = 0;

class book {
  char author[40];
  char title[40];
  int status;
public:
  book(char *n, char *t, int s);
  int get_status() {return status;}
  void set_status(int s) {status = s;}
  void show();
};

book::book(char *n, char *t, int s) {
  strcpy(author, n);
  strcpy(title, t);
  status = s;
}

void book::show() {
  cout << title << " by " << author;
  cout << " is ";
  if(status==IN) cout << "in.\n";
  else cout << "out.\n";
}

int main() {
  book b1("Twain", "Tom Sawyer", IN);
  book b2("Melville", "Moby Dick", CHECKED_OUT);

  b1.show();
  b2.show();

  return 0;
}
```

>**Tom Sawyer by Twain is in.**
Moby Dick by Melville is out.

/* 15. listing 21 – конструктор з одним параметром можна ініціалізувати оператором *obj* */

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

class X {
  int a;
public:
  X(int j) { a = j; }
  int geta() { return a; }
};

int main() {
  X ob = 99; // передати параметру j значення 99
//  X ob = X(99);
  cout << ob.geta(); // вивести на екран 99
  return 0;
}
```

>**99**

/* 16. listing 23 – статичні змінні-члени класу мають тільки один екземпляр цієї змінної для всіх об'єктів цього класу. Оголошення статичної змінної-члена в класі не означає її визначення (виділення пам'яті). Для виділення пам'яті її потрібно визначити поза класом, тобто глобально. */

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

class shared {
  static int a;  // статична змінна-член
  int b;
public:
  void set(int i, int j) {a=i; b=j;}
  void show();
} ;

int shared::a; // визначення змінної поза класом, глобально

void shared::show() {
  cout << "Це статична змінна a: " << a << " і нестатична b: " << b << "\n";
}

int main() {
  shared x, y;

  x.set(1, 1); // присвоїти a значення 1
  cout << "Obj x. ";
  x.show();

  cout << "Obj y. ";
  y.set(2, 2); // змінити значення a на 2
  y.show();


  cout << "Obj x. ";
  x.set(3,3);
  x.show(); /* Тут одночасно змінюється значення змінних-членів
               об'єктів x і y,
               оскільки змінна a використовується обома об'єктами. */

  cout << "Obj y. ";
  y.show();

  return 0;
}
```

```
>Obj x. Це статична змінна a: 1 і нестатична: 1
Obj y. Це статична змінна a: 2 і нестатична b: 2
Obj x. Це статична змінна a: 3 і нестатична b: 3
Obj y. Це статична змінна a: 3 і нестатична b: 2
```

```
/* 17. listing 24 – статична змінна-член створюється до створення першого об'єкта
класу */
#include <iostream>
using namespace std;

class shared {
public:
  static int a;
} ;

int shared::a; // визначення змінної a

int main() {
  shared::a = 99; // ініціалізація перед створенням об'єкта
  cout << "Початкове значення змінної a: " << shared::a;
  cout << "\n";
  shared x;  // створення об'єкту
  cout << "Значення змінної як члена x.a: " << x.a;
  return 0;
```

```
}
>Початкове значення змінної: 99
Значення змінної як члена x.a: 99

/* 18. listing 25 – статична змінна, як індикатор використання ресурсів усіма об'єктами класу */
#include <iostream>
using namespace std;

class cl {
  static int resource;
public:
  int get_resource();
  void free_resource() {resource = 0;}
};

int cl::resource; // визначення ресурсу

int cl::get_resource() {
  if(resource) return 0; // ресурс зайнятий
  else {
    resource = 1;
    return 1;  // ресурс наданий об'єкту
  }
}

int main() {
  cl ob1, ob2;
  if(ob1.get_resource()) cout << "ob1 володіє ресурсом\n";
  if(!ob2.get_resource()) cout << "ob2 доступ заборонений\n";
  ob1.free_resource();  // Звільнення ресурсу
  if(ob2.get_resource())
    cout << "ob2 може використовувати ресурс\n";

  return 0;
}
>ob1 володіє ресурсом
ob2 доступ заборонеий
ob2 може використовувати ресурс

/* 20. listing 26 – визначення кількості існуючих об'єктів конкретного класу за
   допомогою статичної змінної */
#include <iostream>
using namespace std;

class Counter {
public:
  static int count;
  Counter() { count++; }
  ~Counter() { count--; }
};
int Counter::count;

void f();

int main(void) {
  Counter o1;
  cout << "Існуючі об'єкти: ";
  cout << Counter::count << "\n";

  Counter o2;
  cout << " Існуючі об'єкти: ";
  cout << Counter::count << "\n";
```

```
  f();
  cout << " Існуючі об'єкти: ";
  cout << Counter::count << "\n";

  return 0;
}

void f() {
  Counter temp;
  cout << " Існуючі об'єкти:: ";
  cout << Counter::count << "\n";
  // temp знищується після повернення з функції f()
}
```
>Існуючі об'єкти: 1
Існуючі об'єкти: 2
Існуючі об'єкти: 3
Існуючі об'єкти: 2

/* 21. listing 27 – статичні функції-члени мають доступ тільки до інших статичних членів класу. Приклад надання одного ресурсу різним об'єктам. */

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

class cl {
  static int resource;
public:
  static int get_resource();
  void free_resource() { resource = 0; }
};

int cl::resource; // визначення ресурсу

int cl::get_resource() {
  if(resource) return 0; // ресурс зайнятий
  else {
    resource = 1;
    return 1;  // ресурс наданий іншому об'єкту
  }
}

int main() {
  cl ob1, ob2;

  /* Функція get_resource() є статичною, тому її можна викликати незалежно від
любого обєкту */
  if(cl::get_resource()) cout << "ob1 має ресурс\n";
  if(!cl::get_resource()) cout << "ob2 доступ заборонений\n";
  ob1.free_resource();

// може викликатися використовуючи об'єктний синтаксис
  if(ob2.get_resource())
     cout << "ob2 тепер може використовувати ресурс\n";

  return 0;
}
```
>ob1 має ресурс
ob2 доступ заборонений
ob2 тепер може використовувати ресурс

/* 22. listing 28 – статичні функції члени можуть ініціалізовувати закриті статичні змінні-члени до створення реальних об'єктів */

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

```
class static_type {
  static int i;
public:
  static void init(int x) {i = x;}
  void show() {cout << i;}
};

int static_type::i; // визначення змінної i

int main() {
  // ініціалізація статичної змінної перед створенням об'єкта
  static_type::init(100);

  static_type x;
  x.show(); // displays 100

  return 0;
}
>100
```

/* 23. listing 29 – конструктори локальних об'єктів викликаються послідовно при їх створенні, а деструктори викликаються у зворотному порядку. Конструктори глобальних об'єктів виконуються до функції main(), а деструктори після закінчення роботи main()*/

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

class myclass {
public:
  int who;
  myclass(int id);
  ~myclass();
} glob_ob1(1), glob_ob2(2);

myclass::myclass(int id) {
  cout << "Ініціалізація " << id << "\n";
  who = id;
}

myclass::~myclass() {
  cout << "Знищення " << who << "\n";
}

int main() {
  myclass local_ob1(3);

  cout << "Цей рядок не буде першим.\n";

  myclass local_ob2(4);

  return 0;
}
>Ініціалізація 1
Ініціалізація 2
Ініціалізація 3
Цей рядок не буде першим.
Ініціалізація 4
Знищення 4
Знищення 3
Знищення 2
Знищення 1
```

/* 24. listing 32 – локальні класи оголошуються всередині функцій */

```
#include <iostream>
```

```
using namespace std;

void f();

int main() {
  f();
  // клас myclass з цього місця не видно
  return 0;
}

void f() {
  class myclass {
    int i;
  public:
    void put_i(int n) { i=n; }
    int get_i() { return i; }
  } ob;

  ob.put_i(10);
  cout << ob.get_i();
}
>10

/* 25. listing 33 – передача об'єктів функціям по значенню*/
#include <iostream>
using namespace std;

class myclass {
  int i;
public:
  myclass(int n);
  ~myclass();
  void set_i(int n) { i=n; }
  int get_i() { return i; }
};

myclass::myclass(int n) {
  i = n;
  cout << "Створення " << i << "\n";
}

myclass::~myclass() {
  cout << "Знищення " << i << "\n";
}

void f(myclass ob);

int main() {
  myclass o(1);

  f(o);
  cout << "Змінна i в функції main: ";
  cout << o.get_i() << "\n";

  return 0;
}

void f(myclass ob) {
  ob.set_i(2);

  cout << "Це локальна змінна i: " << ob.get_i();
  cout << "\n";
}
//  конструктор викликався один раз, а деструктор два рази. При створенні копії
```

аргумента звичайний конструктор не викликається. Замість нього викликається конструктор копіювання.

>**Створення 1**
Це локальна змінна i: 2
Знищення 2
Змінна i в функції main: 1
Знищення 1

```
/* 26. listing 34 – повернення функціями об’єктів викликаючому модулю */
// Returning objects from a function.
#include <iostream>
using namespace std;

class myclass {
  int i;
public:
  void set_i(int n) { i=n; }
  int get_i() { return i; }
};

myclass f();  // повернення об’єкта типу myclass

int main() {
  myclass o;
  o = f();
  cout << o.get_i() << "\n";
  return 0;
}

myclass f() {
  myclass x;
  x.set_i(1);
  return x;
}
```

>**1**

```
/* 27. listing 35 – присвоєння об’єктів однакового типу один одному */
// Присвоєння об’єктів.
#include <iostream>
using namespace std;

class myclass {
  int i;
public:
  void set_i(int n) { i=n; }
  int get_i() { return i; }
};

int main() {
  myclass ob1, ob2;
  ob1.set_i(99);
  ob2 = ob1; // присвоєння даних об’єкта ob1 об’єкту ob2
  cout << "Змінна i з об’єкта ob2': " << ob2.get_i();
  return 0;
}
```

>**Змінна i з об'єкта ob2: 99**

```
/* 28. listing 36 (timer.cpp) -  робота з часом */
#include <iostream>
#include <ctime>
using namespace std;
//clock_t clock(void);
```

```
class timer {
  double start, stop;
public:
    timer() {
    start = (double) clock() / CLOCKS_PER_SEC;
    cout << "Конструктор " << start << endl;
  } ; // constructor

  ~timer() {
     stop = (double) clock() / CLOCKS_PER_SEC;
     cout << "Деструктор " << stop << "\n";
  }; // destructor
};

int main() {
  timer ob;
  char c;

  // delay ...
  cout << "Натисніть любу клавішу і ENTER: ";
  cin >> c;

  return 0;
}
```

/* 29. listing 37 (swap.cpp) – родова функція. Переставлення значень змінних незалежно від типу */

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

void swap(void *&a, void *&b) {
  void *temp=a;
  a=b;
  b=temp;
}

int main() {
  int *i, *j;
  i = new int;   *i=5;
  j = new int;   *j=20;

  cout << " *i=" << *i << " *j=" << *j << endl;
  swap(i,j);
  cout << " *i=" << *i << " *j=" << *j << endl;
  delete i;
  delete j;

  double *x, *y;
  x = new double;   *x=4.0;
  y = new double;   *y=10.0;

  cout << " *x=" << *x << " *y=" << *y << endl;
  swap(x,y);
  cout << " *x=" << *x << " *y=" << *y << endl;
  delete x;
  delete y;

  return 0;
}
```

```
/.swap
*i=5 *j=20
*i=20 *i=5
*x=4 *y=10
```

***x=10 *y=4**

```
/* 30. 38 (bits.cpp) – використання об’єднання для виведення бітів в байтах */
#include <iostream>
using namespace std;

union bits {
  bits(double n);       // об’єднання може містити члени-функції
  void show_bits();
  double d;
  unsigned char c[sizeof(double)];
};

bits::bits(double n) {
  d = n;
}

void bits::show_bits() {
  int i, j;

  for(j = sizeof(double)-1; j>=0; j--) {
    cout << "Біти в байті " << j << ": ";
    for(i = 128; i; i >>= 1)   // 128 -> 1000'0000
      if(i & c[j]) cout << "1";
      else cout << "0";
    cout << "\n";
  }
}

int main() {
  double i;
  cout << "Введіть число double xxxx.xxxx : ";
  cin >> i;
  bits ob(i);
  ob.show_bits();
  return 0;
}
```

/.bits
Введіть число xxxx.xxxx double: 2012.1234
Біти в байті 7: 01000000
Біти в байті 6: 10011111
Біти в байті 5: 01110000
Біти в байті 4: 01111110
Біти в байті 3: 01011100
Біти в байті 2: 10010001
Біти в байті 1: 11010001
Біти в байті 0: 01001110

/* 31. 39 (bits_union.cpp)– використання неіменованого об’єднання для виведення байтів */

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

int main() {
  union {
    unsigned char bytes[8];
    double value;      // розмір double 8 байтів
  };
  int i;

  value = 859345.324;  // звернення до елементу неіменованого об’єднання
  // виведення байтів числа типу double
  for(i=0; i<8; i++)
    cout << (int) bytes[i] << " ";
```

```
  return 0;
}
./bits_union
248 83 277 165 162 57 42 65
```

Лабораторна робота №4. Екземпляри класів.

Мета роботи – вивчення об’єктів класів, колекцій об’єктів, виділення під них пам’яті.

Теоретичний матеріал лекції, [1, розділ 3], [2, розділ 12, 13], [3, розділи 10, 11, 12].

1. Короткі теоретичні відомості

1.1. Оголошення об’єктів

Після визначення класу можна створювати його екземпляри (об’єкти), посилання та вказівники з типом класу:

- у сегменті або стеку:

```
Тип_класу ідентифікатор_об’єкту(аргументи конструктора);
Тип_класу ідентифікатор_об’єкту = Тип класу(аргументи конструктора);
```

- динамічній пам’яті:

```
Тип_класу * вказівник = new Тип_класу(аргументи конструктора)
Тип_класу & вказівник = *new Тип_класу(аргументи конструктора)
```

Звільнення динамічної пам’яті:

```
delete вказівник
delete & посилання
```

1.2. Види об’єктів

Залежно від можливості зміни полів даних класу розрізняють:

- звичайні змінні об’єкти;
- константні об’єкти (оголошуються зі словом `const`);
- `volatile`-об’єкти.

Звичайні об’єкти можуть викликати будь-які методи класу. Константні об’єкти можуть викликати тільки константні методи класу. Дані константного об’єкта не можна змінити у програмі. Якщо деяке поле константного об’єкта необхідно змінити, то воно оголошується як `mutable`.

Об’єкти з оголошенням `volatile` можуть бути змінені деяким стороннім процесом (апаратурою, операційною системою, паралельними потоками коду).

Залежно від місця оголошення розрізняють:

- глобальні об’єкти;
- локальні об’єкти.

Глобальні об’єкти оголошуються між функціями, а локальні – всередині функцій.

Об’єкти можуть мати наступні класи пам’яті:

- автоматичний (`auto` – за замовчуванням);
- статичний (`static`);
- зовнішній (`extern`).

Локальні об’єкти можуть мати усі класи пам’яті. Глобальні об’єкти не можуть бути автоматичними.

Автоматичні об’єкти існують тільки всередині блоку, в якому вони визначені. Статичні об’єкти існують від моменту їх створення до завершення роботи програми.

Об’єкт можна оголосити статичним або (та) константним. Якщо глобальний об’єкт оголошений як статичний, то його використання у програмі обмежується тільки тим файлом, у якому оголошено статичний об’єкт.

Локальний статичний об’єкт зберігає проміжні значення своїх полів між повторними викликами функції, у якому оголошено об’єкт.

Тимчасові об’єкти створюються у виразах та для виклику методів класу без попереднього створення об’єктів класу. Для створення тимчасових об’єктів вказується назва конструктора та

його аргументи:

```
Class Base {
//…
     Base(int a=0, int b=0) { x=a; y=b; }
     int Get() { retuen x; }
};
Int x=Base().Get();
Int y=(new Base)->Get();
Int z=(*new Base).Get();
void main() {
      Cout << Base(1).Get() << endl;
      Cout << base(2,3).y << endl;
}
```

1.3. Колекції об'єктів. Об'єкти масивів і масиви об'єктів

Об'єкт класу може містити масиви даних. Такий об'єкт є контейнером для масиву. Наприклад, клас, закрита частина якого містить вказівник на двовимірний масив у динамічній пам'яті `int **p`. Двовимірний масив створюється за допомогою одновимірного масиву вказівників, за кожним елементом якого закріплюється відповідний рядок матриці:

```
p = new int* [row];
for(int i=0; i<row;i++)  p[i] = new int[col];
```

Масиви об'єктів можуть оголошуватися одним із таких способів:

```
Тип_класу ідентифікатор_масиву[кількість_об'єктів];
Тип_класу * вказівник =  new Тип_класу[кількість_об'єктів];
Тип_класу & посилання = *new Тип_класу[кількість_об'єктів];
```

При оголошенні масивів без їх початкової ініціалізації вимагається наявність конструктора без параметрів або конструктора, всі параметри якого набувають значення за замовчуванням.

```
class A{
   int x,y;
public:
   A(int x=0, int y=0{
   this->x=x;
   this->y=y;
}
// …
};
void main() {
     A b[]={A(1,2), A(3,4){;
//…
}
```

Звільнення динамічної пам'яті, виділеної для масиву об'єктів:

```
delete [] вказівник
delete []& посилання
```

1.4. Об'єкт список

Контейнерні об'єкти можуть містити в собі різноманітні набори елементів. Розглянемо організацію класу для роботи з однонаправленим списком. У закритій частині класу `list` оголошена структура `elem`, яка описує елемент списку, та вказівник на початок списку. Структура містить поле даних `int`, вказівник `next` - на сусідній елемент списку та конструктор ініціалізації полів структури.

У відкртій частині класу `list` розміщені конструктор, деструктор та допоміжні методи: `add()` - для добавлення елементу у список, `del()` - для вилучення елементу зі списку, `print()` - для виведення списку на екран.

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

```
class list
{
private:
    struct elem
    {
      int info;
      elem *next;  // попередній елемент списку
      elem(int x, elem *p) {info=x; next=p;}
    };
    elem *head;      // вказівник на початок списку
    // конструктор списку з клавіатури
public:
    list()
    {
      int x;
      head=NULL;
      cout << "Введіть 5 цілих чисел" << endl;
      for(int i=0;i<5;i++) { cin>>x; add(x); }
    }
   // добавлення елемента
    void add(int x) {elem *p=new elem(x,head); head=p; }
  // вилучення початкового елемента списку
    void del() { elem *p=head->next; delete head; head=p; }
    void print()   // друк елемента
    {
      elem *p=head;
      while (p!=NULL)
      {
          cout << p->info << ' ';
          p=p->next;
      }
      cout << endl;
    }
};

int main()
{
    list s1;
    s1.print();
    s1.add(7);
    s1.print();
    s1.del();
    s1.print();
    return 0;
}
```

Введіть 5 цілих чисел
5 4 3 2 1
7 5 4 3 2 1
5 4 3 2 1

1.5. Список об’єктів

Замість списку, поміщеного у клас, можна створити список об’єктів. Тепер клас `list` містить у закритій частині поле даних `info` та вказівник на сусідній у списку об’єкт класу `list`. У відкритій частині знаходиться конструктор для ініціалізації елементів та методи доступу до полів закритої частини класу.

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

class list {
    int info;
    list *next;
```

```
public:
    list(int x, list *p) { info=x; next=p; }
    int get_info()    { return info; }
    list * get_next() { return next; }
};

int main() {
    list *head = NULL, *p;
    int x;
    cout << " Введіть 5 цілих чисел" << endl;
    for(int i=0;i<5;i++) { cin>>x; p=new list(x,head); head=p; }
    cout << "Виведення списку" << endl;
    p=head;
    while( p!=NULL) { cout << p->get_info() << ' '; p=p->get_next(); }
    cout << endl;
     // Витирання списку
    while(head != NULL ) { p=head; head=head->get_next(); delete p; }

    return 0;
}
```

Введіть 5 цілих чисел
5 4 3 2 1
Виведення списку
5 4 3 2 1

1.6. Розміщення та оголошення класів

Всередині класу можна оголошувати дані, що мають тип вказівника на цей самий клас, але не можна оголосити посилання або об’єкт цього самого класу (табл. 1). Методи класу, їх параметри та локальні оголошення можуть мати посилання на цей же клас або тип цього самого класу.

Таблиця 1

Використання імен вищерозміщених класів

Клас		**Можливі оголошення у класі**
A	Дані	A*
	Параметри, типи методів, локальні дані	A&, A*, A
B	Дані	A&, A*, A, B*
	Параметри, типи методів, локальні дані	A&, A*, A, B&, B*, B

Нижче оголошення класу можна оголосити його об’єкт, посилання або вказівник. Такі оголошення можна виконати всередині іншого зовнішнього класу, всередині зовнішньої функції або на глобальному рівні.

Щоб мати можливість оголосити посилання або вказівник на нижче розміщений клас, необхідно виконати випереджувальне оголошення класу (табл. 2).

Таблиця 2

Використання імен нищерозміщених класів

Клас B;		**Можливі оголошення у класі**
A	Дані	A*, B&, B*
	Параметри, типи методів, локальні дані	A&, A*, A, B&, B*
B	Дані	A&, A*, A, B*
	Параметри, типи методів, локальні дані	A&, A*, A, B&, B*, B

Випереджуюче оголошення не дає можливості оголосити об’єкт нижче розміщеного класу у вище розміщеному класі (або на глобальному рівні). Метод вище розміщеного класу не може отримувати або повертати об’єкт нижче розміщеного класу “за значенням”.

У вищерозміщеному класі дозволяється тільки цілісне використання вказівників та посилань на нижчерозміщений клас. За допомогою таких вказівників або посилань не можна отримати доступ до елементів (даних або методів) нижчерозміщеного класу.

У програмі потрібно уникати звернень до нижчерозміщених класів.

Запитання.

1. Як оголошуються об’єкти класів.
2. Які є види об’єктів.
3. Як створити тимчасові об’єкти.
4. Як генеруються і обробляються винятки при відсутності місця під динамічну пам’ять.
5. Як створити двовимірний масив у динамічній пам’яті конструктором класу.
6. Як створити масив об’єктів класу в статичній, стековій і динамічній пам’яті.
7. Як створити масив об’єктів у динамічній пам’яті і їх ініціалізувати.
8. Як створити об’єкт список.
9. Як створити список об’єктів.
10. Як використовуються всередині класу імена вище розміщених класів.
11. Як використовуються всередині класу імена вище нищерозміщених класів.

Завдання.

1. Написати програму, в якій оператор `new` використовується для динамічного виділення пам’яті під змінні типу `float[2][2], long[2][2], char[20]`. Присвоїти масивам значення і вивести на екран. Конструктором звільнити динамічно виділену область пам’яті.

2. Написати функцію `neg()`, яка міняє знак свого цілого параметра. Функцію написати двома способами: з використанням параметра вказівника і з використанням параметра посилання.

3. Використовуючи наступне оголошення класу, створити масив з 10 елементів і ініціалізувати змінну `ch` значеннями від A до J, вивести значення на екран

```
class letters {
    char ch;
public:
    letters(char c) { ch=c; }
    char get_ch() { t=return ch; }
};
```

4. Виконати завдання згідно номеру студента в журналі групи:

4.1. Створити клас Вектор цілих чисел. Розробити клас СТЕК, що містить об'єкт класу ВЕКТОР. Визначити необхідні конструктори, деструктори, методи занесення елемента у стек та читання зі стека. Вивести вміст стека на екран.

4.2. Створити клас СПИСОК цілих чисел. Розробити клас СТЕК, що містить об’єкт класу СПИСОК. Визначити необхідні конструктори, деструктори, методи занесення елемента у стек та читання зі стеку. Вивести вміст стека на екран.

4.3. Розробити клас КОМП’ЮТЕР, який містить об’єкти класів СИСТЕМНИЙ БЛОК, ЕКРАН, КЛАВІАТУРА, МИША. Визначити необхідні елементи даних, конструктори, деструктори, методи роботи з елементами даних. Створити масив об’єктів класу КОМП’ЮТЕР та визначити комп’ютер з мінімальним відхиленням параметрів від заданих.

4.4. Створити клас АВТОМОБІЛЬ, який містить об’єкти класу ДВИГУН. У класі ДВИГУН визначено дані про об’єм двигуна, потужність, заводський номер, Клас АВТОМОБІЛЬ додатково містить марку, колір, та номер державної реєстрації. Визначити конструктори без параметрів і з різним числом параметрів, деструктор, методи для роботи з полями даних. Визначити методи зміни номера і кольору.

4.5. Створити клас ДВИГУН із заданням потужності та об’єму двигуна. Створити клас ВАНТАЖІВКА, що містить об’єкт класу ДВИГУН. Додатково задається марка (вказівник на рядок), вантажопідйомність та номер державної реєстрації. Визначити конструктори,

деструктори, методи встановлення та читання елементів даних.

4.6. Створити клас ФАЙЛ, який містить дані про файл: назву, розмір, дату та час його створення, атрибути. Створити клас КАТАЛОГ, що містить масив об'єктів класу ФАЙЛ. Визначити необхідні конструктори, деструктори, методи роботи з файлами та каталогами. Дані про файли отримати командою `Linux ls`.

4.7. Розробити клас УНІВЕРСИТЕТ, який містить прізвище ректора та масив об'єктів ІНСТИТУТ. Об'єкт класу ІНСТИТУТ містить прізвище директора та масив об'єктів класу КАФЕДРА. Клас КАФЕДРА містить прізвище завідувача кафедри та кількість співробітників. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Обчислити загальну кількість співробітників університету.

4.8. Клас ГРУПА містить масив об'єктів класу СТУДЕНТ. Клас СТУДЕНТ містить прізвище студента, номер залікової книжки та масив оцінок. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Знайти середній бал кожного студента. Вивести на екран прізвище п'яти студентів з найвищим балом.

4.9. Клас ФАЙЛ містить масив об'єктів класу РЯДОК символів. Клас рядок символів містить вказівник на рядок символів в області динамічної пам'яті. Визначити необхідні конструктори, деструктори, методи роботи з файлом рядків символів - запису у файл, читання з файлу, поповнення, копіювання.

4.10. Клас ВІКНО містить об'єкти класів ПРЯМОКУТНИК, КОЛІР_ФОНУ, МЕНЮ, СМУГИ_ПРОКРУТКИ. Клас ПРЯМОКУТНИК містить координати головної діагоналі вікна, класс МЕНЮ - масив рядків з назвами команд меню, клас СМУГИ_ПРОКРУТКИ - поточні координати повзунків. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних.

4.11. Клас ПРОГРАВАЧ містить масив об'єктів класу БІБЛІОТЕКА мультимедіа. Клас БІБЛІОТЕКА містить дані про відеофільми: заголовок, тему, акторів, оцінку, довжину, швидкість відтворення, назву файла, дату. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Здійснити пошук фільмів за акторами, назвами, темами.

4.12. Клас БІБЛІОТЕКА містить масив об'єктів класу КНИГА. Клас КНИГА містить прізвище автора, назву, видавництво, рік видання, кількість сторінок. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Здійснити пошук книг за прізвищем автора, назвою, видавництвом, роком видання.

4.13. Клас РОЗКЛАД руху поїздів містить масив об'єктів класу ІНФОРМАЦІЯ про час відправлення поїздів. У класі ІНФОРМАЦІЯ задано номер поїзда, напрям руху, час відправлення, номер платформи, час прибуття на кінцеву станцію. Визначити необхідні конструктори, деструктори та методи роботи з елементами даних. Ввести поточний час та визначити найближчий час відправлення поїзда за заданим напрямом.

4.14. Клас ЕЛЕКТРОННА таблиця містить масив об'єктів класу ВЕКТОР дійсних чисел. Розробити методи для опрацювання даних за встановленими формулами. Визначити необхідні конструктори, деструктори та методи доступу до елементів даних. Знайти суму елементів у кожному рядку, стовпчику, інші обчислення за формулами.

4.15. Клас МАТРИЦЯ містить масив об'єктів ВЕКТОР дійсних чисел. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Виконати додавання, віднімання матриць, множення матриці на вектор, множення матриці на матрицю.

4.16. Клас ВЕКТОР містить масив об'єктів РЯДОК символів. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Здійснити виділення лексем рядка, пошуку найдовшого і найкоротшого слова, найдовшого і найкоротшого рядка.

4.17. Клас ЧЕРГА містить масив об'єктів типу ОСОБА. Визначити необхідні дані, конструктори, деструктори та методи ставлення у чергу та вилучення з черги. Вивести поточний вміст черги на екран.

4.18. Клас МАГАЗИН містить масив об'єктів класу ТОВАРИ. Клас ТОВАРИ містить назву товару, свідоцтво якості, вартість. Клас ПОКУПЕЦЬ містить перелік потрібних товарів. Визначити необхідні дані, конструктори, деструктори та методи робот з елементами даних.

Здійснити купівлю та визначити вартість товарів згідно з переліком.

4.19. Клас домашній кінотеатр містить об’єкти класів ТЮНЕР, ТЕЛЕВІЗОР, ЗВУКОВІ_КОЛОНКИ, ПУЛЬТ_КЕРУВАННЯ. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Клас ТЮНЕР містить канали програм. Клас ПУЛЬТ керування містить методи перемикання каналів, регулювання гучності, включення/виключення. Використовуючи ПУЛЬТ керування здійснити перемикання і вибирати потрібний канал.

4.20. Клас ЦИФРОВИЙ ГОДИННИК містить об’єкти класів КОНТРОЛЕР_ЧАСУ і КОНТРОЛЕР_ДАТИ, ЕКРАН для відображення часу. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Встановити поточні дату і час. Встановити час будильника. Зімітувати спрацювання будильника.

4.21. Клас вентилятор містить класи ДВИГУН, КОНТРОЛЕР, ПУЛЬТ керування. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. За допомогою пульта керування виставити необхідну швидкість обертів двигуна та час включення ВЕНТИЛЯТОРА.

4.22. Клас ЕКРАННИЙ_РЕДАКТОР тексту містить масив об’єктів класу РЯДОК символів та об’єкт класу КУРСОР. Клас КУРСОР містить координати екранного курсора. Клас РЯДОК символів містить список об’єктів СИМВОЛІВ. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Виконати моделювання набору та редагування тексту: включення символів у рядок, витирання символів з рядка. Відредагований текст зберегти у файл.

4.23. Клас АПТЕКА містить список об’єктів класу ЛІКИ. Клас ЛІКИ містить назву препарату, концентрацію, інструкцію до застосування, вартість. Клас РЕЦЕРТ містить перелік ліків. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Визначити наявність потрібних ліків в аптеці та вартість ліків за рецептом.

4.24. Клас ПРІОРИТЕТНА_ЧЕРГА містить масив об’єктів класу ОСОБА. Визначити необхідні конструктори, деструктори та методи ставлення у чергу і вилучення з черги. Вивести поточний вміст черги на екран.

4.25. Клас СТРІЛЬБА по мішені містить об’єкти класів МІШЕНЬ та СТРІЛЕЦЬ. Клас МІШЕНЬ задається набором очок влучності стрільбою У кожному акті стрільби МІШЕНЬ з’являється із заданою імовірністю. Влучність стрільця задається дискретним розподілом ймовірностей потрапляння у мішень. Визначити суму очок з десяти спроб стрільби по мішені.

4.26. Клас мобільний ТЕЛЕФОН містить об’єкти класів SIM-КАРТА, ТЕЛЕФОННА_КНИГА та РАХУНОК. Визначити необхідні дані, конструктори, деструктори та методи роботи з елементами даних. Виконати поповнення рахунку, редагування телефонної книги, визначення залишку на рахунку.

2. Завдання для самостійної роботи № 3

```
/* Розд. 13. 1. listing 1 — масив об'єктів */
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  void set_i(int j) { i=j; }
  int get_i() { return i; }
};

int main() {
  cl ob[3];
  int i;
  for(i=0; i<3; i++) ob[i].set_i(i+1);
  for(i=0; i<3; i++)
    cout << ob[i].get_i() << " ";
  return 0;
}
```

1 2 3

```
/* 2. listing 2 клас з конструктором  */
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  cl(int j) { i=j; }
  int get_i() { return i; }
};

int main() {
  cl ob[3] = {1, 2, 3}; // скорочена форма ініціалізації,
//  c1 ob[3]={ c1(1), c1(2), c1(3) };  повна форма ініціалізаціїї
  int i;
  for(i=0; i<3; i++)
    cout << ob[i].get_i() << " ";
  return 0;
}
```

1 2 3

```
/* 3. listing 4 — конструктор з декількома параметрами */
#include <iostream>
using namespace std;

class cl {
  int h;
  int i;
public:
  cl(int j, int k) { h=j; i=k; } // конструктор з 2-ма параметрами
  int get_i() {return i;}
  int get_h() {return h;}
};

int main() {
  cl ob[3] = {
    cl(1, 2), // іціалізація
    cl(3, 4),
    cl(5, 6)
```

```
  };
  int i;
  for(i=0; i<3; i++) {
    cout << ob[i].get_h();
    cout << ", ";
    cout << ob[i].get_i() << "\n";
  }
  return 0;
}
```
1, 2
3, 4
5, 6

```
/* 4. listing 8 — вказівники на об'єкти */
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  cl(int j) { i=j; }
  int get_i() { return i; }
};

int main(){
  cl ob(88), *p;
  p = &ob; // тримати адрес ob
  cout << p->get_i(); // для виклику функції get_i() застосовується
                      // вказівник ->
  return 0;
}
```
88

```
/* 5. listing 9 — доступ до елементів масиву через вказівник*/
#include <iostream>
using namespace std;

class cl {
  int i;
public:
  cl() { i=0; }
  cl(int j) { i=j; }
  int get_i() { return i; }
};

int main() {
  cl ob[3] = {1, 2, 3};
  cl *p;
  int i;

  p = ob; // адрес першого елемента масиву p=&ob[0]
  for(i=0; i<3; i++) {
    cout << p->get_i() << " ";
    p++; // вказівник на наступний елемент масиву
  }
  return 0;
}
```
1 2 3

```
/* 6. listing 10 — доступ до члена об'єкта */
#include <iostream>
using namespace std;

class cl {
```

```
public:
  int i;
  cl(int j) { i=j; }
};

int main() {
  cl ob(1);
  int *p;
  p = &ob.i; // отримання адресу члена ob.i
  cout << *p; // звернення до члена  ob.i через вказівник p
  return 0;
}
```
1

/* 7. listing 13 — вказівник this */
```
#include <iostream>
using namespace std;

class pwr {
  double b;
  int e;
  double val;
public:
  pwr(double base, int exp);
  double get_pwr() { return val; }   // return this->val;
};

pwr::pwr(double base, int exp) {  // функції члену неявно передається
                                  // вказівник this
  b = base;   // this->b=base;
  e = exp;    // this->e=exp;
  val = 1;    // this->val=1;
  if(exp==0) return;
  for( ; exp>0; exp--) val = val * b;    // this->val = this->val * this->b;
}

int main() {
  pwr x(4.0, 2), y(2.5, 1), z(5.7, 0);
  cout << x.get_pwr() << " ";
  cout << y.get_pwr() << " ";
  cout << z.get_pwr() << "\n";
  return 0;
}
```
16 2.5 1

/* 8. listing 19 — доступ до об'єктів похідного класу через вказівник на базовий клас.
Вказівник базового класу *B може посилатися на об'єкти похідного класу */
```
#include <iostream>
using namespace std;

class base {
  int i;
public:
  void set_i(int num) { i=num; }
  int get_i() { return i; }
};

class derived: public base {
  int j;
public:
  void set_j(int num) { j=num; }
  int get_j() { return j; }
};
```

```
int main() {
  base *bp;
  derived d;

  bp = &d; // базовий вказівник вказує на об'єкт похідного класу

  // доступ до успадкованих елементів в похідному класі з використанням вказівника
на базовий клас
  bp->set_i(10);
  cout << bp->get_i() << " ";

/* Наступний оператор не працює! На нові елементи похідного класу не можна
вказувати базовим вказівником
   bp->set_j(88); // error
   cout << bp->get_j(); // error
*/
/* для доступу до нових членів похідного класу потрібно перетворити тип вказівника
на похідний клас */
   ((derived * ) bp)->set_j(88);
   cout << ((derived *) bp) ->get_j();
  return 0;
}
10
88

/* 9. listing 21 — адресна арифметика залежить від типу базового вказівника */
#include <iostream>
using namespace std;

class base {
  int i;
public:
  void set_i(int num) { i=num; }
  int get_i() { return i; }
};

class derived: public base {
  int j;
public:
  void set_j(int num) {j=num;}
  int get_j() {return j;}
};

int main() {
  base *bp;   // вказівник на базовий тип !
  derived d[2];

  bp = d;  // bp=&a[0] - базовий вказівник вказує на об'єкт похідного класу

  d[0].set_i(1);
  d[1].set_i(9);

  cout << bp->get_i() << " ";
  bp++;    // інкремент стосується базового, а не похідного типу !
  cout << bp->get_i(); // тому на екран виводиться невірне значення

  return 0;
}
1 0

/* 10. listing 22 — вказівники на члени класу через об'єкт */
#include <iostream>
using namespace std;
```

```
class cl {
public:
  cl(int i) { val=i; }
  int val;
  int double_val() { return val+val; }
};

int main() {
  int cl::*data; // вказівник на член дані класу
  int (cl::*func)(); // вказівник на член функцію класу
  cl ob1(1), ob2(2); // створення об'єктів

  data = &cl::val; // визначення зміщення члена val
  func = &cl::double_val; // визначення зміщення функції double_val()

  cout << "Значення: ";
  cout << ob1.*data << " " << ob2.*data << "\n";

  cout << "Подвоєне значення: ";
  cout << (ob1.*func)() << " ";
  cout << (ob2.*func)() << "\n";

  return 0;
}
```

Значення: 1 2
Подвоєне значення: 2 4

```
/* 11. listing 23 - вказівники на члени класу через вказівник на об'єкт */
#include <iostream>
using namespace std;

class cl {
public:
  cl(int i) { val=i; }
  int val;
  int double_val() { return val+val; }
};

int main() {
  int cl::*data;     // вказівник на член дані класу
  int (cl::*func)(); // вказівник на член функцію класу
  cl ob1(1), ob2(2); // створення об'єктів
  cl *p1, *p2;

  p1 = &ob1; // доступ до об'єктів через вказівники
  p2 = &ob2;

  data = &cl::val; // зміщення члена  даних val
  func = &cl::double_val; // зміщення члена функції double_val()

  cout << "Значення: ";
  cout << p1->*data << " " << p2->*data << "\n";

  cout << "Подвоєне значення: ";
  cout << (p1->*func)() << " ";
  cout << (p2->*func)() << "\n";

  return 0;
}
```

Значення: 1 2
Подвоєне значення: 2 4

```
/* 12. listing 25 — аргумент передається за допомогою явного вказівника  */
```

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

void neg(int *i);
int main() {
  int x;
  x = 10;
  cout << x << " і його негативне значення ";
  neg(&x);
  cout << x << "\n";
  return 0;
}
void neg(int *i) {
  *i = -*i;
}
```
10 і його негативне значення -10

```
/* 13. listing 27 — аргумент передається через посилання (неявний вказівник) */
#include <iostream>
using namespace std;

void neg(int &i); // посилання

int main() {
  int x;

  x = 10;
  cout << x << " і його негативне значення ";
  neg(x); // оператор & більш не потрібний
  cout << x << "\n";

  return 0;
}

void neg(int &i) {
  i = -i; // i є посиланням і оператор * більш не потрібний
}
```
10 і його негативне значення -10

```
/* 14. listing 30 – використання посилань для переставлення елементів масиву */
#include <iostream>
using namespace std;

void swap(int &i, int &j);
int main() {
  int a, b, c, d;
  a = 1;
  b = 2;
  c = 3;
  d = 4;

  cout << "a i b: " << a << " " << b << "\n";
  swap(a, b); //оператор  & не потрібний
  cout << "a i b: " << a << " " << b << "\n";

  cout << "c i d: " << c << " " << d << "\n";
  swap(c, d);
  cout << "c i d: " << c << " " << d << "\n";

  return 0;
}

void swap(int &i, int &j) {
  int t;
```

```
  t = i; // оператор * не потрібний
  i = j;
  j = t;
}
a i b: 1 2
a i b: 2 1
c i d: 3 4
c i d: 4 3

/* 15. listing 31 — передача посилань на об'єкти */
#include <iostream>
using namespace std;

class cl {
  int id;
public:
  int i;
  cl(int i);
  ~cl();
  void neg(cl &o) { o.i = -o.i; } // при передачі посилання копія об’єкта не
                                  // створюється
};
cl::cl(int num) {
  cout << "Створення об'єкта " << num << "\n";
  id = num;
}
cl::~cl() {
  cout << "Знищення об'єкта " << id << "\n";
}

int main() {
  cl o(1);
  o.i = 10;
  o.neg(o);
  cout << o.i << "\n";

  return 0;
}
Створення об'єкта 1
-10
Знищення об'єкта 1

/* 16. listing 32 — функція повертає посилання */
#include <iostream>
using namespace std;

char &replace(int i); // повернення посилання
char s[80] = "Hello world!\n";

int main() {
  replace(5) = '='; // функція в лівій частині повертає посилання на символ
                    // який входить в рядок s, індекс якого заданий змінною i
                    // вставлення символу ”=” після слова Hello
  cout << s;
  return 0;
}

char &replace(int i)
{
  return s[i];
}
Hello=world!
```

```
/* 17. listing 33 — незалежні посилання, псевдоніми об'єктів*/
#include <iostream>
using namespace std;

int main() {
  int a;
  int &ref = a; // незалежне посилання
  a = 10;
  cout << a << " " << ref << "\n";
  ref = 100;
  cout << a << " " << ref << "\n";
  int b = 19;
  ref = b; // b->a
  cout << a << " " << ref << "\n";
  ref--; // a--
  cout << a << " " << ref << "\n";
  return 0;
}
10 10
100 100
19 19
18 18

/* 18. listing 36 — виділення динамічної пам'яті, оператори new, delete */
#include <iostream>
#include <new>      // заголовок для bad_alloc
using namespace std;
int main() {
  int *p;

  try {
    p = new int; // виділити пам'ять для int
  } catch (bad_alloc xa) {    // перехоплення виняткової ситуації
    cout << "Виняткова ситуація\n";
    return 1;
  }
  *p = 100;
  cout << "За адресом  " << p << "  ";
  cout << "записано значення " << *p << "\n";
  delete p;
  return 0;
}
По адресу 0x603010 записано значення 100

/* 19. listing 37 – ініціалізація динамічної пам'яті */
#include <iostream>
#include <new>
using namespace std;

int main() {
  int *p;
  try {
    p = new int (87); // ініціалізація значенням 87
  } catch (bad_alloc xa) {
    cout << "Виняткова сутуація\n";
    return 1;
  }
  cout << "По адресу " << p << "  ";
  cout << "записано число " << *p << "\n";
  delete p;
  return 0;
}
За адресою 0x603010 записано число 87
```

```
/* 20. listing 38 — виділення пам’яті під масив цілих чисел, new, delete [] */
#include <iostream>
#include <new>
using namespace std;
int main()
{
  int *p, i;
  try {
    p = new int [10]; // виділення пам'яті під вектор 10 int
  } catch (bad_alloc xa) {
    cout << "Виняткова ситуація\n";
    return 1;
  }
  for(i=0; i<10; i++ )   // ініціалізація вектора
    p[i] = i;
  for(i=0; i<10; i++)
    cout << p[i] << " ";
  delete [] p; // звільнення пам’яті, яка займав вектор
  return 0;
}
0 1 2 3 4 5 6 7 8 9

/* 21. listing 39 — виділення пам'яті для об'єктів */
#include <iostream>
#include <new>
#include <cstring>
using namespace std;

class balance {
  double cur_bal;
  char name[80];
public:
  void set(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }

  void get_bal(double &n, char *s) {
    n = cur_bal;
    strcpy(s, name);
  }
};

int main() {
  balance *p;
  char s[80];
  double n;

  try
 {
    p = new balance;
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні пам'яті\n";
    return 1;
  }

  p->set(1200.50, "Іванченко І.І."); // доступ до членів через вказівник
  p->get_bal(n, s);
  cout << s << " має на рахунку : " << n << " грн.\n";
  delete p;
  return 0;
}
Іванченко І.І. має на рахунку : 1200.5 грн.
```

```
/* 22. listing 40 — конструктор з параметрами */
#include <iostream>
#include <new>
#include <cstring>
using namespace std;

class balance {
  double cur_bal;
  char name[80];
public:
  balance(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }
  ~balance() {
    cout << "Знищення об'єкту ";
    cout << name << "\n";
  }
  void get_bal(double &n, char *s) {
    n = cur_bal;
    strcpy(s, name);
  }
};

int main() {
  balance *p;
  char s[80];
  double n;

  // конструктор з параметрами
  try {
    p = new balance (12387.87, "Іванченко І.І.");
  } catch (bad_alloc xa) {
    cout << "Виняткова ситуація\n";
    return 1;
  }

  p->get_bal(n, s);
  cout << s << " сума дорівнює: " << n;
  cout << "\n";
  delete p;
  return 0;
}
```

```
Іванченко І.І. сума дорівнює: 12387.9
Знищення об'єкту Іванченко І.І.
```

```
/* 23. listing 41 — масиви розміщені в динамічній пам'яті не можна ініціалізувати
                в операторі new, тому має бути конструктор без параметрів */
#include <iostream>
#include <new>
#include <cstring>
using namespace std;

class balance {
  double cur_bal;
  char name[80];
public:
  balance(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }
  balance() {} // конструктор без параметрів !
  ~balance() {
    cout << "Знищення ";
```

```
    cout << name << "\n";
  }
  void set(double n, char *s) {
    cur_bal = n;
    strcpy(name, s);
  }
  void get_bal(double &n, char *s) {
    n = cur_bal;
    strcpy(s, name);
  }
};

int main() {
  balance *p;
  char s[80];
  double n;
  int i;
  try {
    p = new balance [3]; // виділення пам'яті під масив об'єктів
  } catch (bad_alloc xa) {
    cout << "Виняткова ситуація\n";
    return 1;
  }

  // використовується оператор . , а не -> !
  p[0].set(12387.87, "Іванченко І.І."); // ініціалізація масиву об'єктів
  p[1].set(144.00, "Василенко В.В.");
  p[2].set(-11.23, "Степаненко С.С.");
  for(i=0; i<3; i++) {
    p[i].get_bal(n, s);
    cout << s << " сума = " << n;
    cout << "\n";
  }
  delete [] p;
  return 0;
}
```

Іванченко І.І. сума = 12387.9
Василенко В.В. сума = 144
Степаненко С.С. сума = -11.23
Знищення Степаненко С.С.
Знищення Василенко В.В.
Знищення Іванченко І.І.

/* 24. listing 42 - альтернативний оператор new(nothrow), який при відсутності пам'яті не генерує вийняткову ситуацію, а повертає 0. (для сумісності із старими програмами */

```
#include <iostream>
#include <new>
#define N 5
using namespace std;
int main()
{
  int *p, i;
  p = new(nothrow) int[N]; // застосування опції nothrow
  if(!p) {
    cout << "Помилка виділення пам'яті.\n";
    return 1;
  }
  for(i=0; i<N; i++) p[i] = i;
  for(i=0; i<N; i++) cout << p[i] << " ";
  delete [] p; // звільнення пам'яті
  return 0;
}
```

0 1 2 3 4

Лабораторна робота № 5. Перевантаження функцій і операцій.

Мета роботи: вивчення операторів перевантаження функцій і операцій, конструкторів копіювання, використання аргументів по замовчуванню.

Теоретичний матеріал лекції, [1, розділ 4], [2, розділ 14, 15], [3, розділи 8, 11].

1. Короткі теоретичні відомості

1.1. Перевантаження функцій

Перевантаження функцій – це використання одного імені для декількох функцій, які використовують різні типи або різну кількість параметрів.

```
int func(int i);
int func(int i, int j);
double func(double i);
```

Тип значення, яке повертає функція не дозволяє її перевантажувати

```
int func(int i);
double func(int i);
```

Іноді оголошення двох функцій зовнішньо відрізняється, але фактично співпадає

```
void func(int *p);
void func(int p[]);
```

1.2. Перевантаження операцій

Операції C++ є контекстнозалежними і можуть бути перевантажені у класі. Перевантажені операції використовуються як мікрооперації над об’єктами класів. Допускається перевантаження таких операцій:

- первинних

```
()      []      ->
```

-унарних

```
!       ~       +       -       ++      --      &       *       new     delete
```

- бінарних

```
->*     *       /       %       +       -       <<      >>
<       <=      >       >=      ==      !=      &       ^
|       &&      ||      =       *=      /=      %=      +=
-=      &=      ^=      |=      <<=     >>=     ,
```

Перевантаження операцій здійснюється за допомогою *операторних функцій*:

```
тип operator операція (список формальних аргументів) {…}
```

При перевантаженні не можна змінити пріорітети та асоціативність операцій.

Операторні функції можуть бути оголошені членами або друзями класу. Тільки членами класу оголошуються операції `()`, `[]`, `->`, `new`, `delete`, `=`. Усі інші операції можуть бути оголошені як членами, так і друзями класу. Операторні функції не можуть мати параметрів за замовчуванням.

1.3. Варіанти перевантаження операцій

Якщо операторна функція є членом класу, то її першим (неявним) параметром є вказівник `this`. Кількість параметрів операторної функції визначається її видом та способом оголошення - як члена або як друга класу, як подано в табл.4.1.

Таблиця 4.1

Варіанти перевантаження операцій

Операції	**Члени класу**	**Друзі класу**
Унарні	Без параметрів	Один параметр з типом класу, посиланням або вказівником на об’єкт класу

Бінарні	Один параметр (другий)	Два параметри (перший параметр повинен мати тип класу, посилання або вказівник на об’єкт класу)

1.4. Унарні операції члени і друзі класу

Унарна операція член класу перевантажується за допомогою нестатичної операторної функції без параметрів. Якщо унарна операція перевантажується як друг класу, то вона має один параметр з типом класу, посилання або вказівника на об’єкт класу.

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    A& operator-() const;
};
A& A::operator-() const
{ return A(-x); }

void main()
{
   A a(1), b;
   b=-a;
   //…
}
```

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    friend A operator-(const A&);
};
A operator-(const A& a)
{ return A(-a.x); }

void main()
{
   A a(1), b;
   b=-a;
   //…
}
```

1.5. Бінарні операції члени і друзі класу

Перевантаження бінарної операції як члена класу є нестатичною операторною функцією з одним параметром (другим, перший є вказівник `this`). Бінарна операція перевантажена як друг класу, є операторною функцією з двома параметрами (перший - змінна класу, посилання або вказівник на об’єкт класу).

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    A& operator-() const;
};
A& A::operator-(const A& a) const
{ return A(x-a.x); }

void main()
{
   A a(1), b(2), c;
   c=b-a;
   //…
}
```

```
class A {
    int x;
public:
    A(int y=0) : x(y) {}
    friend  A  operator-(const  A&,
const A&);
};
A operator-(const A& b, const A& a)
{ return A(b.x-a.x); }

void main()
{
   A a(1), b(2), c;
   c=b-a;
   //…
}
```

1.6. Виклики операторних функцій

Можливі два варіанти викликів операторних функцій – явний та неявний. Явний виклик операторних функцій здійснюється так само, як і звичайних дружніх функцій або методів класу:

```
b=a.operator-()          b=-a       унарний мінус - метод класу
b=operator-(a)           b=-a       унарний мінус - друг класу
b=a.operator-(a)         c=b-a      бінарний мінус - метод класу
b=operator-(b,a)         c=b-a      бінарний мінус - друг класу
```

Неявний виклик операторних фінкцій будується на основі застосування відповідних

операцій до об’єктів:

```
b=-a;   // унарний мінус - метод класу
b=-a;   // унарний мінус - друг класу
c=b-a;  // бінарний мінус - метод класу
c=b-a;  // бінарний мінус - друг класу
```

1.7. Перевантаження первинних та унарних операцій

Операція `()` перевантажується як член класу, може повертати один з визначених типів та може мати змінну кількість аргументів.

Операція `[]` перевантажується як бінарна нестатична функція член класу. Якщо повертає посилання, то дозволяє використання з обох сторін операції присвоєння (функції з типом поилання можна присвоїти значення).

Операція `->` перевантажується як унарна постфіксна операція. Повинна бути нестатичною функцією членом і має повертати вказівник на структурований тип. Не повинна мати параметрів. Клас, який перевантажує операцію `operator->`, називають розумним (smart) вказівником.

Операції інкремента та декремента застосовуються у префіксній (`++x, --x`) та постфіксній (`++x, --x`) формах. Перевантажуються як унарні операції, можуть бути членами або друзями класу. Для розрізнення префіксної та постфіксної форм у постфікс ній формі використовують додатковий параметри типу `int`, який явно не задається:

- префіксна форма (метод класу – `A& operator++();` друг класу – `friend A& operator++(A&);`

- постфіксна форма (метод класу – `A& operator++(int);` друг класу – `friend A& operator++(A&, int)`.

Операції new та delete можуть бути перевантажені у контексті класу або без використання класу. У контексті класу є статичними, перевантажуються за допомогою методів, не можуть бути друзями, у них не можна використовувати нестатичні елементи класу, не можуть бути віртуальними. Операції `new` та `new[]` можуть мати більше одного параметра. Тоді вони оголошуються так:

```
void * operator new(size t, додатковий список параметрів);
void * operator new[](size t, додатковий список параметрів);
```

При виклику перевантажений операцій new, new[] додатковий список аргументів задається у круглих дужках після слова new.

```
A *p = new(додатковий список аргументів) A(аргументи конструтора);
A *q = new(додатковий список аргументів) A[кількість об’єктів];
```

Якщо додатковим параметром операції `new` є вказівник на раніше визначену змінну, то за допомогою такої операці можна розмістити об’єкт класу за відомою адресою. У цьому випадку динамічна пам’ять для об’єкта не виділяється, а операція `new` повертає адресу раніше виділеної області пам’яті.

1.8. Інші операції перевантаження

Операція присвоєння використовується для присвоєння одного об’єкту іншому існуючому об’єкту класу. Оголошується як нестатична функція член класу, не може бути константою, не успадковується, розміщується у відкритій частині класу.

Алгоритм перевантаження операції присвоєння подібний до алгоритму конструктора копіювання. Відмінність полягає у тому, що конструтор копіювання створює нову копію об’єкта, а в операції присвоєння використовуються раніше створені об’єкти. Якщо клас містить поля, які є вказівниками або посиланнями на будь-який тип, то операція присвоєння повинна бути перевантажена:

```
A& A::operator=(A& a) {
   if (this!=&a) { delete &r; r=*new int(a.r) }
```

```
    return this;
}
```

Операторна функція перетворення типу призначена для перетворення об'єкта класу у значення іншого типу, зокрема у тип іншого класу:

```
operator інший_тип();
```

Операція допускає перевантаження. У протоколі класу може бути оголошено декілька операторних функцій перетворення типу. Повертає значення, яке має "інший тип". Може бути віртуальною, успадковується.

Потокові операції введення-виведення перевантажуються як друзі класу, оскільки їх першим параметром повинен бути об'єкт потокового типу: `istream` – для введення, `ostream` – для виведення об'єктів класу. Перевантаження операцій потокового введення-виведення дає можливість цілісно виводити об'єкти оголошеного класу.

Запитання.

1. Що таке перевантаження функцій.
2. Як викликати через адресу перевантажену функції і передати їй параметр.
3. Аргументи за замовчуванню як альтернатива перевантаження функцій.
4. Неоднозначності, які виникають при перевантаженні функцій.
5. Що таке перевантаження операцій і який синтаксис операторної функції.
6. Особливості перевантаження унарних і бінарних операторних функцій, як членів класу і як друзів класу.
7. Перевантаження первинних операцій `()`, `[]`, `->`.
8. Перевантаження унарних операцій інкремента `++` та декремента `--`.
9. Перевантаження унарних операцій `new` та `delete`. Розміщувана форма операції `new`.
10. Перевантаження операції присвоєння =.
11. Операція перетворення типу `operator інший тип()`.
12. Перевантаження потокових операцій введення-виведення.

Завдання.

1. Створити класс ДІЙСНИХ ЧИСЕЛ. Перевантажити бінарні операції `+`, `-`, `*`, `/` для виконання арифметичних обчислень над дійсними числами. Операції `+`, `-` перевантажити за допомогою методів класу, а операції `*`, `/`, як дружні функції. Перевантажити потокову операцію >> для введення з клавіатури та операцію << для виведення на екран.

2. Оголосити клас ДОВГЕ ЦІЛЕ у двійковій системі числення. Двійкове число задати як рядок символів. Перевантажити арифметичні та логічні операції роботи з двійковими числами. Визначити операції введення з потоку та виведення у потік.

3. Оголосити клас МАТРИЦЯ. Перевантажити операції `+`, `-`, `*` відповідно для додавання, віднімання та множення матриць. Визначити операції введення матриці з потоку та виведення у потік.

4. Перевантажити операцію `[]` для визначення цілої, а операцію `()` – дробової частини дійсного числа. Визначити операцію введення з потоку та виведення у потік.

5. Перевантажити операцію `()` для виділення підмасиву із заданого масиву цілих чисел. Параметрами операції `()` є значення індексів елементів масиву. Визначити операції введення матриці з потоку та виведення у потік.

6. Перевантажити операції `->` та `[]` для звернення до елементів лінійного списку. Визначити операції введення з потоку та виведення у потік.

7. Перевантажити операції `new` та `delete` так, щоб перед знищенням об'єкта виводився час його існування у програмі. Визначити операції введення з потоку та виведення у потік.

8. Перевантажити операції `+`, `-`, `*` для виконання операцій над множинами цілих чисел. Визначити операції введення з потоку та виведення у потік.

9. Створити клас СТУДЕНТ, що має ім'я (вказівник на рядок), номер залікової книжки, оцінки з предметів. Визначити клас ГРУПА студентів, який містить список студентів. Перевантажити операції + для додавання нового елементу у список, – для вилучення зі списку даних про студента, який має незадовільні оцінки, * для об'єднання даних про студентів в один список. Визначити операції введення з потоку та виведення у потік.

10. Перевантажити операцію + для дописування одного файлу в кінець іншого, операцію – для вилучення з першого файлу елементів, які входять у другий файл, * для створення файлу, який складається зі спільних елементів обох файлів, ^ для встановлення позиції покажчика запису-читання файлу. Визначити операції введення з потоку та виведення у потік.

11. Створити клас КООРДИНАТА, який має дві координати `x` і `y`. Перевантажити операції `+, *` так, щоб можна було використовувати об'єкти типу `coord` для виконання інструкцій `ob+int, int+ob, ob*int, int*ob`.

12. Нехай задано наступне оголошення класу

```
class dynarray {
     int *p;
     int size;
public:
    dynarray(int s);   // передача розміру масива
    int &put(int i);   // повернення посилання на елемент i
    int  get(int i);   // повернення значення змінної i
    // створити функцію operator=()
};
```

Добавити все необхідне для створення типу динамічний масив. Тобто, виділити пам'ять для масиву і зберігти вказівник на цю пам'ять за адресою `p`. Розмір масиву в байтах зберегти в змінній `size`. Створити функцію `put()`, яка повертає посилання на заданий елемент масиву і функцію `get()`, яка повертає значення даного елемента. Забезпечити контроль границь масиву. Перевантажити інструкцію присвоєння так, щоб виділена кожному масиву такого типу пам'ять не була випадково пошкоджена при присвоєнні одного масиву іншому.

2. Завдання для самостійної роботи № 5

2.1. Перевантаження функцій

```
/* 1. розділ 14,listing 1 — перевантаження функцій з різним типом параметрів */
#include <iostream>
using namespace std;

int myfunc(int i); // ці варіанти відрізняються типами параметрів
double myfunc(double i);

int main()
{
  cout << myfunc(10) << " "; // виклик myfunc(int i)
  cout << myfunc(5.4);       // виклик myfunc(double i)
  return 0;
}

double myfunc(double i)
{
  return i;
}

int myfunc(int i)
{
  return i;
}
10 5.4

/* 2. listing 2 — перевантаження функцій з різною кількістю параметрів */
#include <iostream>
using namespace std;

int myfunc(int i); // ці варіанти відрізняються різною кількістю параметрів
int myfunc(int i, int j);

int main()
{
  cout << myfunc(10) << " "; // виклик myfunc(int i)
  cout << myfunc(4, 5);      // виклик myfunc(int i, int j)

  return 0;
}

int myfunc(int i)
{
  return i;
}

int myfunc(int i, int j)
{
  return i*j;
}
10 20


/* 3. listing 13 — визначення адресу перевантаженої функції */
/* Функція має адрес. Якщо цей адрес присвоїти вказівнику, то функцію можна
викликати не по імені, а через вказівник. Якщо функція перевантажена, то як
визначити її адрес? Адрес визначається по типу вказівника */
#include <iostream>
using namespace std;
```

```
int myfunc(int a);
int myfunc(int a, int b);

int main()
{
  int (*fp)(int a); // вказівник на функцію int f(int)
  //int (*fp) (int a, int b); // вказівник на функцію int f(int, int)
  fp = myfunc; // адрес функції на myfunc(int)
  cout << fp(5);
  return 0;
}

int myfunc(int a)
{
  return a;
}
int myfunc(int a, int b)
{
  return a*b;
}
```
5

/* 4. listing 18 — аргументам функції можна присвоїти значення по замовчуванню*/
```
#include <iostream>
using namespace std;
/* функція очистки екрана дисплея */
void clrscr(int size=25);

int main()
{
  register int i;

  for(i=0; i<30; i++ ) cout << i << endl;
  cin.get();
  clrscr(); // стирає 25 рядків

  for(i=0; i<30; i++ ) cout << i << endl;
  cin.get();
  clrscr(10); // стирає 10 рядків

  return 0;
}

void clrscr(int size)
{
   for(; size; size--) cout << endl;
}
```

/* 5. listing 20 — аргументи функцій по замовчуванню. Автоматична вставка перед рядком заданої кількості пропусків від краю екрана */
```
#include <iostream>
using namespace std;

/* За замовчуванням indent=-1 і функція використовує попереднє значення
відступів. */
void iputs(char *str, int indent = -1);

int main()
{
  iputs("Вітання всім", 10);
  iputs("Перед цим рядком вставлено 10 пропусків");
  iputs("Перед цим рядком вставлено 5 пропусків", 5);
  iputs("Перед цим рядком немає пропусків", 0);
```

```
  return 0;
}

void iputs(char *str, int indent)
{
  static int i = 0; // застосовується попереднє значення indent

  if(indent >= 0)
    i = indent;
  else              // використовується старе значення indent
    indent = i;

  for( ; indent; indent--) cout << " ";

  cout << str << "\n";
}
```

Вітання всім
Перед цим рядком вставлено 10 пропусків
Перед цим рядком вставлено 5 пропусків
Перед цим рядком немає пропусків

```
/* 6. listing 26 — аргументи по замовчуванню, як альтернатива перевантаженню */
#include <iostream>
#include <cstring>
using namespace std;

//void mystrcat(char *s1, char *s2);
//void mystrcat(char *s1, char *s2, int len);
/* функція еквівалентна двом попереднім, тобто реалізовує перевантаження */
void mystrcat(char *s1, char *s2, int len = -1);

int main() {
  char str1[80] = "This is a test";
  char str2[80] = "0123456789";

  mystrcat(str1, str2, 5); // конкатенація 5 символів
  cout << str1 << '\n';

  strcpy(str1, "This is a test"); // відновлення str1

  mystrcat(str1, str2); // конкатененація двох рядків
  cout << str1 << '\n';

  return 0;
}

// налаштована версія strcat()
void mystrcat(char *s1, char *s2, int len) {
  // знаходимо кінець рядка s1
  while(*s1) s1++;

  if(len == -1) len = strlen(s2);

  while(*s2 && len) {
    *s1 = *s2; // копіюємо символи
    s1++;
    s2++;
    len--;
  }

  *s1 = '\0'; // признак кінця рядка s1
}
```

```
This is a test01234
This is a test0123456789
```

```
/* 7. listing 28 — неоднозначність виклику перевантажених функцій */
#include <iostream>
using namespace std;

float myfunc(float i);
double myfunc(double i);

int main() {
  cout << myfunc(10.1) << " "; // однозначний виклик myfunc(double)
// неоднозначний виклик, неясно в який тип float чи double перетворити 10
  cout << myfunc(10);
  return 0;
}

float myfunc(float i) {
  return i;
}

double myfunc(double i) {
  return -i;
}
c14_28.cpp: In function ‘int main()’:
c14_28.cpp:11: error: call of overloaded ‘myfunc(int)’ is ambiguous
c14_28.cpp:5: note: candidates are: float myfunc(float)
c14_28.cpp:6: note:                 double myfunc(double)
```

```
/* 8. listing 29 — неоднозначний виклик функцій */
#include <iostream>
using namespace std;

char myfunc(unsigned char ch);
char myfunc(char ch);

int main()
{
  cout << myfunc('c');       // виклик функції myfunc(char)
  cout << myfunc(88) << " "; // неоднозначність => char чи unsigned char
  return 0;
}

char myfunc(unsigned char ch)
{
  return ch-1;
}

char myfunc(char ch)
{
  return ch+1;
}
c14_29.cpp: In function ‘int main()’:
c14_29.cpp:11: error: call of overloaded ‘myfunc(int)’ is ambiguous
c14_29.cpp:5: note: candidates are: char myfunc(unsigned char)
c14_29.cpp:6: note:                 char myfunc(char)
```

```
/* 9. listing 30  - неоднозначність, спричинена параметрами по замовчуванню */
#include <iostream>
using namespace std;

int myfunc(int i);
int myfunc(int i, int j=1);
```

```
int main()
{
  cout << myfunc(4, 5) << " "; // однозначність
  cout << myfunc(10);          // неоднозначність

  return 0;
}

int myfunc(int i)
{
  return i;
}

int myfunc(int i, int j)
{
  return i*j;
}
```

```
c14_30.cpp: In function ‘int main()’:
c14_30.cpp:11: error: call of overloaded ‘myfunc(int)’ is ambiguous
c14_30.cpp:5: note: candidates are: int myfunc(int)
c14_30.cpp:6: note:                 int myfunc(int, int)
```

```
/* 10. listing 31 — неоднозначність спричинена оператом &x */
/* В цій програмі помилка. Функція неможна перезавантажувати, якщо один параметр
передається по посиланню, а другий по значенню. */
#include <iostream>
using namespace std;

void f(int x);
void f(int &x); // error

int main()
{
  int a=10;
  f(a); // error, which f()?
  return 0;
}

void f(int x)
{
  cout << "In f(int)\n";
}

void f(int &x)
{
  cout << "In f(int &)\n";
}
```

```
c14_31.cpp: In function ‘int main()’:
c14_31.cpp:13: error: call of overloaded ‘f(int&)’ is ambiguous
c14_31.cpp:6: note: candidates are: void f(int)
c14_31.cpp:7: note:                 void f(int&)
```

2.2. Перевантаження операцій

```
/* 1. розділ 15, listing 1 — перевантаження оператора "+"  за допомогою операторної функції */
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;   /* клас містить географічні координати */
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";  // географ. довгота
    cout << latitude << "\n";  // географ. широта
  }

  loc operator+(loc op2);
};

// Перевантаження бінарного оператора + стосовно класу loc.
// лівий операнд this, а правий op2.
// При перевантаженні бінарного оператора виклик операторної
// функції генерується лівим операндом this
loc loc::operator+(loc op2)   // операторна функція
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30);

  ob1.show(); // виводить на екран 10 20
  ob2.show(); // виводить на екран 5 30

  ob1 = ob1 + ob2;
  ob1.show(); // виводить на екран 15 50

  return 0;
}
```

```
10 20
5 30
15 50
```

```
/* 2. listing 4 — перевантаження операторів "+", "-", "=", "++"  */
#include <iostream>
using namespace std;

class loc {
  int longitude, latitude;
public:
```

```
  loc() {} // конструктор для створення тимчасових об'єктів
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  loc operator+(loc op2);
  loc operator-(loc op2);
  loc operator=(loc op2);
  loc operator++();
};

// Перевантажений оператор "+" для класу loc.
loc loc::operator+(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}

// Перевантажений оператор "-" для класу loc.
loc loc::operator-(loc op2)
{
  loc temp;

  // Зверніть увагу на порядок елементів.
  // Дані об'єкта op2 мають відніматися від даних об'єкта, на які вказує
  // вказівник this
  temp.longitude = longitude - op2.longitude;
  temp.latitude  = latitude  - op2.latitude;

  return temp;
}

// Перевантажений оператор "=" для класу loc
loc loc::operator=(loc op2)
{
  longitude = op2.longitude;
  latitude = op2.latitude;

  return *this; // повернення об'єкта, який генерує виклик
}
// Перевантажений унарного оператор "++" для класу loc. Операнди функції
// передаються за допомогою вказівника this
loc loc::operator++()
{
  longitude++;
  latitude++;
  return *this;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30), ob3(90, 90);

  ob1.show();
```

```
  ob2.show();

  ++ob1;      // префіксний інкремент
  ob1.show(); // виводить на екран числа 11 21

  ob2 = ++ob1;
  ob1.show(); // виводить на екран числа 12 22
  ob2.show(); // виводить на екран числа 12 22

  ob1 = ob2 = ob3; // множинне присвоєння
  ob1.show(); // виводить на екран числа 90 90
  ob2.show(); // виводить на екран числа 90 90

  return 0;
}
```

10 20
5 30
11 21
12 22
12 22
90 90
90 90

/* 3. listing 8 — перевантаження операторів за допомогою дружніх функцій.
Дружні функції не можуть перезавантажити оператори ”=”, ”()”, ”[]”, ”->”.
Дружні функції не отримують неявного вказівника this, тому при перезавантаженні бінарного оператора функція отримує два параметри, унарного – один.
*/

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

class loc {
  int longitude, latitude;
public:
  loc() {} // конструктор потрібний для створення тимчасових об'єктів
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  friend loc operator+(loc op1, loc op2); //  дружня функція
  loc operator-(loc op2);
  loc operator=(loc op2);
  loc operator++();
};

// Оператор ”+” перевантажений дружньою функцією
loc operator+(loc op1, loc op2)
{
  loc temp;

  temp.longitude = op1.longitude + op2.longitude;
  temp.latitude = op1.latitude + op2.latitude;

  return temp;
}

// Перевантажений оператор ”-” з використанням неявного вказівника this
loc loc::operator-(loc op2)
```

```
{
  loc temp;

  // notice order of operands
  temp.longitude = longitude - op2.longitude;
  temp.latitude = latitude - op2.latitude;

  return temp;
}

// Перевантаження оператора "="
loc loc::operator=(loc op2)
{
  longitude = op2.longitude;
  latitude = op2.latitude;

  return *this; // i.e., return object that generated call
}
// Перевантаження оператора "++"
loc loc::operator++()
{
  longitude++;
  latitude++;
  return *this;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30);

  ob1 = ob1 + ob2;
  ob1.show();
  return 0;
}
15 50
```

/* 4. listing 9 – використання дружніх функцій для перезаванаження операторів "++", "--". Так як дружні функції не отримують вказівника this, їх операнди потрібно передавати за допомогою посилань
*/

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

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  loc operator=(loc op2);
  friend loc operator++(loc &op);
  friend loc operator--(loc &op);
};

// Перевантажений оператор "=" для loc.
loc loc::operator=(loc op2)
```

```
{
  longitude = op2.longitude;
  latitude = op2.latitude;
  return *this; // повертається об’єкт, який генерував виклик
}

// Оператор ++ перевантажений дружньою функцією
// Використовується передача параметра за допомогою посилання
loc operator++(loc &op)
{
  op.longitude++;
  op.latitude++;
  return op;
}

// Оператор -- перевантажений дружньою функцією
// Використовується передача параметра за допомогою посилання
loc operator--(loc &op)
{
  op.longitude--;
  op.latitude--;
  return op;
}

int main()
{
  loc ob1(10, 20), ob2;

  ob1.show();
  ++ob1;
  ob1.show(); // displays 11 21

  ob2 =  ++ob1;
  ob2.show(); // displays 12 22

  --ob2;
  ob2.show(); // displays 11 21

  return 0;
}
10 20
11 21
12 22
11 21
```

Примітка. Якщо потрібно перезавантажити постфіксу форму операторів ”++”, ”--” то потрібно задати другий фіктивний параметр

```
// перезавантаження постфіксного оператора за допомогою дружньої функції
friend loc operator++(loc &op, int x);
```

/* 5. listing 11 — використання дружніх операторних функцій для додавання об'єкт + число (Ob + 100), число + об'єкт (100 + Ob). При використанні вказівника this вираз 100+Ob невизначений. Використання дружніх функцій усовує цей недолік, так як для них неважливий порядок слідування операндів.

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

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
```

```
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  friend loc operator+(loc op1, int op2);
  friend loc operator+(int op1, loc op2);
};

// Оператор + перевантажений для додавання  об'єкт + int
loc operator+(loc op1, int op2)
{
  loc temp;

  temp.longitude = op1.longitude + op2;
  temp.latitude = op1.latitude + op2;

  return temp;
}
// Оператор + перевантажений для додавання int + об'єкт
loc operator+(int op1, loc op2)
{
  loc temp;

  temp.longitude = op1 + op2.longitude;
  temp.latitude = op1 + op2.latitude;

  return temp;
}

int main()
{
  loc ob1(10, 20), ob2( 5, 30), ob3(7, 14);

  ob1.show();
  ob2.show();
  ob3.show();

  ob1 = ob2 + 10; // обидва вирази
  ob3 = 10 + ob2; // вірні

  ob1.show();
  ob3.show();

  return 0;
}
```

10 20
5 30
7 14
15 40
15 40

```
/* 6. listing 13  - перевантаження операторів new, delete для конкретного класу */
#include <iostream>
#include <cstdlib>
#include <new>
using namespace std;

class loc {
  int longitude, latitude;
```

```
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  void *operator new(size_t size);
  void operator delete(void *p);
};

// оператор new перевантажений для класу loc
void *loc::operator new(size_t size)
{
  void *p;

  cout << "Всередені перезавантаженого оператора new\n";
  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
  return p;
}

// оперетор delete перевантажений для класу loc
void loc::operator delete(void *p)
{
  cout << "Всередені перевантаженого оператора delete.\n";
  free(p);
}

int main()
{
  loc *p1, *p2;

  try {
    p1 = new loc (10, 20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні памяті для об'єкта p1.\n";
    return 1;
  }

  try {
    p2 = new loc (-10, -20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні памяті для об'єкта p2.\n";
    return 1;
  }

  p1->show();
  p2->show();

  delete p1;
  delete p2;

  return 0;
}
```

Всередені перезавантаженого оператора new

```
Всередені перезавантаженого оператора new
10 20
-10 -20
Всередені перезавантаженого оператора delete
Всередені перезавантаженого оператора delete
```

```
/* 7. listing 15 — глобальне перевантаження операторів new, delete.
Якщо оператори new, delete перевантажити поза класами, то вони будуть глобальними.
*/
#include <iostream>
#include <cstdlib>
#include <new>
using namespace std;

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }
};

// глобальний оператор new
void *operator new(size_t size)
{
  void *p;

  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
  return p;
}

// глобальний оператор delete
void operator delete(void *p)
{
  free(p);
}

int main()
{
  loc *p1, *p2;
  float *f;

  try {
    p1 = new loc (10, 20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні пам'яті для об'єкта p1.\n";
    return 1;
  }

  try {
    p2 = new loc (-10, -20);
  } catch (bad_alloc xa) {
    cout << "Помилка при виділенні пам'яті для об'єкта p2.\n";
```

```
    return 1;
  }
 try {
    f = new float; // використовується перезавантажена версія оператора new
  } catch (bad_alloc xa) {
    cout << "Помилка виділення пам'яті.\n";
    return 1;
  }

  *f = 10.10F;
  cout << *f << "\n";

  p1->show();
  p2->show();

  delete p1;
  delete p2;
  delete f;

  return 0;
}
```

10.1
10 20
-10 -20

/* 8. listing 17 – перевантаження операторів new, delete для масивів.
Розміщення і звільнення з памяті масиву об’єктів loc */

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

class loc {
  int longitude, latitude;
public:
  loc() {longitude = latitude = 0;}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  void *operator new(size_t size);
  void operator delete(void *p);

  void *operator new[](size_t size);
  void operator delete[](void *p);
};

// перевантажений оператор new для класу loc.
void *loc::operator new(size_t size)
{
void *p;

  cout << "In overloaded new.\n";
  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
```

```
  return p;
}

// перевантажений оператор delete для класу loc.
void loc::operator delete(void *p)
{
  cout << "In overloaded delete.\n";
  free(p);
}

// перевантажений оператор new для масиву об’єктів.
void *loc::operator new[](size_t size)
{
  void *p;

  cout << "Using overload new[].\n";
  p =  malloc(size);
  if(!p) {
    bad_alloc ba;
    throw ba;
  }
  return p;
}

// перевантажений оператор delete для масиву об’єктів loc.
void loc::operator delete[](void *p)
{
  cout << "Freeing array using overloaded delete[]\n";
  free(p);
}

int main()
{
  loc *p1, *p2;
  int i;

  try {
    p1 = new loc (10, 20); // розміщення об’єкту в пам’яті
  } catch (bad_alloc xa) {
    cout << "Allocation error for p1.\n";
    return 1;;
  }

  try {
    p2 = new loc [10]; // розміщення масиву в пам’яті
  } catch (bad_alloc xa) {
    cout << "Allocation error for p2.\n";
    return 1;;
  }

  p1->show();

  for(i=0; i<10; i++)
    p2[i].show();

  delete p1; // знищення об'єкту
  delete [] p2; // знищення масиву

  return 0;
}
```

/* 9. listing 21 — перевантаження оператора []. При перезавантаженні оператор [] вважається бінарним, тому він має мати такий вид: operator[](int i) */

```
#include <iostream>
```

```
using namespace std;

class atype {
  int a[3];
public:
  atype(int i, int j, int k) {
    a[0] = i;
    a[1] = j;
    a[2] = k;
  }
  int operator[](int i) { return a[i]; }
};

int main()
{
  atype ob(1, 2, 3);
  cout << ob[1]; // виводить число 2

return 0;
}
```
2

/* 10. listing 22 перевантаження оператора []. Для того, щоб оператор [] міг бути і в лівій і правій частині функція operator[] має повертати посилання */

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

class atype {
  int a[3];
public:
  atype(int i, int j, int k) {
    a[0] = i;
    a[1] = j;
    a[2] = k;
  }
  int &operator[](int i) { return a[i]; }
};

int main()
{
  atype ob(1, 2, 3);
  cout << ob[1]; // Оператор [] стоїть справа, виводить на екран 2
  cout << " ";
  ob[1] = 25;    // Оператор [] стоїть зліва від ”=”.
  cout << ob[1]; // виводить на екран 25
  return 0;
}
```
2 25

/* 11. listing 23 — використання перевантаженого оператора [] для перевірки діапазону індексів масиву */

```
// Приклад безпечного масиву
#include <iostream>
#include <cstdlib>
using namespace std;

class atype {
  int a[3];
public:
  atype(int i, int j, int k) {
    a[0] = i;
    a[1] = j;
    a[2] = k;
  }
```

```
  int &operator[](int i);
};

// Перевірка діапазону всередині класу atype.
int &atype::operator[](int i)
{
  if(i<0 || i> 2) {
    cout << " Вихід за допустимий діапазон\n";
    exit(1);
  }
  return a[i];
}

int main()
{
  atype ob(1, 2, 3);

  cout << ob[1]; // виводить на екран  2
  cout << " ";

  ob[1] = 25;    // [] оператор зліва
  cout << ob[1]; // виводить на екран  25

  ob[3] = 44;    // Генерується помилка, значення 3 лежить за межами допустимого
інтервалу

  return 0;
}
```

2 25 Вихід за допустимий діапазон

/* 12. listing 28 — перевантаження оператора (). При перезавантаженні оператора ”()” створюється операторна функція, якій можна передавати довільне число параметрів */

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

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }

  loc operator+(loc op2);
  loc operator()(int i, int j);
};

// Перевантажений оператор ”()” для класу loc
loc loc::operator()(int i, int j)
{
  longitude = i;
  latitude = j;

  return *this;
}

// Перевантажений оператор ”+” для класу loc
```

```
loc loc::operator+(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}

int main()
{
  loc ob1(10, 20), ob2(1, 1);
  ob1.show();
  ob1(7, 8);  // Оператор () застосовується самостійно
  ob1.show();
  ob1 = ob2 + ob1(10, 10);  // оператор () використовується у виразі
  ob1.show();
  return 0;
}
```

10 20
7 8
11 11

/* 13. listing 29 — оператор ”->” при перевантаженні вважається унарним */

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

class myclass {
public:
 int i;
 myclass *operator->() {return this;} //операторна функція operator->() має бути членом класу
};

int main()
{
  myclass ob;
  ob->i = 10; // еквівалентно виразу ob.i
  cout << ob.i << " " << ob->i;
  return 0;
}
```

10 10

/* 14. listing 30 — перевантаження оператора ”,”. Оператор вважається бінарним. Мета перезавантаження може бути різна. В даному випадку будуть ігноруватися всі аргументи, записані через кому, крім крайнього правого */

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

class loc {
  int longitude, latitude;
public:
  loc() {}
  loc(int lg, int lt) {
    longitude = lg;
    latitude = lt;
  }

  void show() {
    cout << longitude << " ";
    cout << latitude << "\n";
  }
```

```
  loc operator+(loc op2);
  loc operator,(loc op2);
};

// Перевантаження оператора ”,” для класу loc
loc loc::operator,(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude;
  temp.latitude = op2.latitude;
  cout << op2.longitude << " " << op2.latitude << "\n";

  return temp;
}

// Перезавантаження оператора ”+” для класу loc
loc loc::operator+(loc op2)
{
  loc temp;

  temp.longitude = op2.longitude + longitude;
  temp.latitude = op2.latitude + latitude;

  return temp;
}
int main()
{
  loc ob1(10, 20), ob2( 5, 30), ob3(1, 1);
  ob1.show();
  ob2.show();
  ob3.show();
  cout << "\n";
  ob1 = (ob1, ob2+ob2, ob3);
  ob1.show(); // виводиться на екран значення 1 1, тобто значення ob3
  return 0;
}
```

```
10 20
5 30
1 1

10 60
1 1
1 1
```

Лабораторна робота № 6. Одинарне успадкування

Мета роботи: вивчення способів та механізмів одинарного успадкування.
Теоретичний матеріал лекції, [1, розділ 6], [2, розділ 16]; [3, розділ 13].

1. Короткі теоретичні відомості

1.1. Суть успадкування класів

Успадкування класів і структур є одним із способів повторного використання коду. Успадкування полягає у використанні *оголошень* елементів базового класу у структурі іншого класу, який називається похідним від базового. У результаті успадкування об'єкт похідного класу набуває властивостей базового класу.

Похідний клас може користуватися полями даних та методами, успадкованими від базового класу без їх повторного оголошення. Крім того, похідний клас може оголосити власні поля даних та методи. Оголошення елементів похідного класу перекривають однойменні оголошення елементів базового класу.

Поля даних базового класу не копіюються у похідний клас. На машинному рівні для обєкта похідного класу буде виділено окрему від базового класу область пам'яті під успадковані поля.

Конструктори, деструктори, операторна функція присвоєння та друзі класу не успадковуються.

Між обєктами представниками базового та похідного класу діє відношення "є": похідний клас є нащадком базового класу, а базовий клас є його предком. В інформаційному плані нащадок є багатшим від предка.

1.2. Оголошення успадкування

Для успадкування базового класу необхідно після імені базового класу через символ ':' вказати режим успадкування та ім'я базового класу:

```
class Base { . . . };
class Derived: режим_успадкування Base { . . . };
```

Режим успадкування може мати наступні значення: `private, protected, public`. Режим успадкування визначає, в які частини похідного класу потрапляють елементи базового класу.

При `private`-успадкуванні елементи всіх частин базового класу потрапляють у `private`-частину похідного класу.

При `protected`-успадкуванні `private`-частина базового класу потрапляє у `private`-частину похідного класу, а `protected`- та `public`-частини базового класу потрапляють у `protected`-частину похідного класу.

При `public`-успадкуванні `private, protected` та `public`-частини базового класу потрапляють в однойменні частини похідного класу.

Для будь-якого варіанта успадковуються всі елементи базового класу у похідний клас, але `private`-частина базового класу безпосередньо недоступна у похідному класі.

Схеми успадкування класів показано в табл. 1.

Для ініціалізації успадкованих полів викликається конструктор базового класу у списку ініціалізації конструктора похідного класу (після символу ':').

У похідному класі існує можливість зміни застосованого режиму успадкування елементів базового класу. Успадковані елементи можна перевести тільки на той самий рівень, на якому вони були розміщені в базовому класі. Для цього в області дії цього рівня у похідному класі необхідно виконати оголошення:

```
ім'я_базового_класу::імя_елемента;
```

Таблиця 1. **Дія режимів успадкування класів**

Режим успадкування	Рівні захисту класу Base	Рівні захисту класу Derived
private	private	private*
	protected	private
	public	private
protected	private	private*
	protected	protected
	public	protected
public	private	private*
	protected	protected
	public	public

У табл. 1.1 символ ‘*’ позначає `private`-частину базового класу, безпосередньо не доступну у похідному класі.

1.4. Перекриття елементів класу

Елементи похідного класу перекривають дію однойменних елементів базового класу. Якщо у будь-якій частині похідного класу містяться елементи, імена яких збігаються з успадкованими елементами базового класу, то відбувається перекриття таких успадкованих елементів. Поля даних похідного класу перекривають однойменні дані базового класу. Для доступу до перекритих полів базового класу у похідному класі використовується конструкція:
`ім’я_базового_класу::ім’я_поля;`

Методи похідного класу перекривають однойменні методи базового класу. Перекриття методів реалізується через їх перевизначення (`override`) або перевантаження (`overload`). Перевизначені функції мають однакове ім’я, однакову кількість та типи параметрів, однаковий тип результату. Перевизначення допускається між успадкованими класами, але не допускається у межах одного класу.

Для доступу до перекритих методів використовується ім’я класу, де визначений метод, та операція дозволу доступу:

```
ім’я_базового_класу::ім’я_методу(фактичні_аргументи);
```

Перевантажені функції мають однакове ім’я, можуть мати однаковий або різний тип результату, але обов’язково відрізняються кількістю або типами своїх параметрів. Перевантаження допускається у межах одного класу та між успадкованими класами. Вибір конкретного перевантаженого методу визначається списком фактичних аргументів.

1.5. Особливості успадкування закритої частини базового класу

Хоча `private`-частина базового класу успадковується похідним класом, але її елементи безпосередньо не доступні для використання у похідному класі. За необхідності такий доступ може бути реалізований одним із способів:

- через методи базового класу, розміщені у `protected` або `public`-частинах базового класу;
- оголошенням похідного класу другом до базового класу. Якщо клас нащадок є дружнім до базового класу, то він має доступ до елементів усіх частин класу за їх іменами, а не за допомогою об’єкта, як це забезпечується для дружніх зовнішніх класів та функцій.

1.6. Порядок виклику конструкторів та деструкторів при успадкуванні класів

Основна задача конструктора полягає в ініціалізації полів-даних об’єктів класу. За необхідності у конструкторі можна виділити динамічну пам’ять для даних об’єкта. Якщо клас містить оголошення віртуальних функцій, то конструктор додатково виконує ініціалізацію вказівника на таблицю віртуальних методів.

При успадкуванні діє такий порядок виклику конструкторів:

- викликається конструктор базового класу; якщо явний виклик конструктора базового класу не здійснюється у списку ініціалізації конструктора похідного класу (після двокрапки), то викликається конструктор базового класу за замовчуванням (без параметрів);

- перед викликом конструктора контейнерного класу (базового або похідного) викликаються конструктори його полів-об'єктів в порядку їх запису у протоколі відповідного класу; якщо ініціалізація об'єктів не здійснюється у списку ініціалізації конструктора контейнерного класу, то викликаються конструктори за замовчуванням.

- викликається конструктор похідного класу.

Деструктори викликаються при виході об'єктів із області досяжності програми або при знищенні об'єктів у динамічній пам'яті. Деструктори успадкованих класів викликаються у зворотному порядку - від похідного до базового класу.

1.7. Успадкування статичних даних і методів

За відсутності перекриття об'єкти класів-нащадків мають спільні статичні поля даних, успадковані від A `B::x==A::x`. Якщо у похідному класі B визначити однойменне статичне поле зі статичним полем класу A, то об'єкт класу B матиме спільне з класом A статичне поле `A::x` та індивідуальне статичне поле `B::x!=A::x`.

Статичні методи призначені для роботи зі статичними даними класу. Успадкування статичних методів відбувається за загальними правилами і визначається режимами успадкування. За відсутності перекриття статичні методи базового класу будуть доступними для виклику у похідному класі. Якщо статичний метод похідного класу перекриває статичний метод базового класу, то у похідному класі буде два різні статичні методи.

1.8. Успадкування константних елементів класу

Константні елементи класу успадковуються як звичайні елементи з тією відмінністю, що ініціалізація константних елементів класу здійснюється у списку ініціалізації конструктора. Допускається перекриття константних полів. Тоді у похідному класі будуть константні поля, успадковані від базового класу, та власні константні поля.

Константні методи не можуть змінювати значення даних членів класу. Їх успадкування підлягає загальним правилам. Якщо похідний клас перекриває константні методи базового класу, то у похідному класі існують методи, успадковані від базового класу, та власні однойменні методи. Перекриті методи викликаються за допомогою імені базового класу.

```
class A {
protected:
   int x;
public:
 A(int) { this->x=x; }
 int Get() const
        {return x;};
};
```

```
class B: public A {
  int y;
public:
  B(int x, int y):A(x)
     {this->y=y;}
  int Get() const
  {return y;}
};
```

```
void main() {
B b(1,2);
cout<<b.A::Get()<<endl;
cout<<b.Get()<<endl;
}
```

1.9. Присвоєння об'єктів при успадкуванні

Об'єкту класу предка можна присвоїти об'єкт одного з класів-нащадків. Це саме справедливе також для вказівників і посилань. Таке присвоєння можливе лише для `public` успадкованих класів.

Зворотне присвоєння об'єкту похідного класу об'єкта базового класу можливе за наявності конструктора за замовчуванням `A::A()` та конструктора перетворення типу `B::B(A)`, наприклад такого: `B:B(A a) {y=a.GetX(); }`

Вказівник (або посилання) на об'єкт похідного класу можна присвоїти вказівнику

(посиланню) з типом базового класу за допомогою відповідного перетворення типів:

```
B* pb=(B*) &a;
B& rb=(B&) a;
```

Таке перетворення не завжди коректне. Для перетворення типів при успадкуванні класів необхідно використовувати операції приведення типів.

Запитання.

1. В чому суть успадкування класів. Як оголошується успадкування класів.
2. Як діють режими успадкування класів.
3. Як змінити режим успадкування.
4. Коли виникає перекриття елементів класу. Як доступитися до перекритих елементів.
5. Як при успадкуванні отримати доступ до закритої частини базового класу.
6. Порядок виклику конструкторів та деструкторів при успадкуванні.
7. Як успадковуються статичні методи та дані.
8. Як успадковуються константні елементи класу.
9. Присвоєння об’єктів при успадкуванні.

Завдання.

1. Розробити клас ВЕКТОР цілих чисел. Успадкувати цей клас для створення СТЕКА. Визначити необхідні дані, конструктори, деструктор та методи занесення елемента у стек та зчитування елемента зі стеку. Вивести вміст створеного стека на екран.

2. Створити клас КІМНАТА з даними: ширина, дрвжзина, висота. Визначити конструктор і метод доступу. Створити клас ОДНОКІМНАТНОЇ КВАРТИРИ, яка містить кімнату і кухню, номер та поверх. Визначити конструктори, методи доступу. Визначити `public`-похідний клас ОДНОКІМНАТНИХ КВАРТИР РІЗНИХ МІСТ (додатковий параметр - назва міста). Визначити конструктори, деструктор і функцію виведенння.

3. Створити ієрархію класів СПОРТИВНА ГРА і ФУТБОЛ. Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

4. Створити клас АВТОМОБІЛЬ, який має марку (вказівник на рядок), колір, об’єм двигуна, потужність, номер реєстрації. Визначити конструктори, деструктор і функцію виведення. Створити public-похідний клас ВАНТАЖІВКА, що має вантажопідйомність кузова. Визначити конструктори без параметрів і з різним числом параметрів, деструктор, методи виведення. Визначити методи перепризначення кольору і номера реєстрації.

5. Створити клас ПРЯМОКУТНИК (довжина, ширина) з методом обчислення його площі. Створити похідний від нього клас ПАРАЛЕЛЕПІПЕД (довжина ширина, висота) з методом обчислення об’єму. Всі дані для створення об’єктів задаються у програмі. Вивести на екран характеристики об’єктів, їх розміри, площу та об’єм.

6. Створититклас КРАПКА, що має координати. Визначити класи ЕЛІПСІВ і КІЛ. Визначити ієрархію типів. Визначити функції виведення, конструктори, деструктори, обчислення площі.

7. Створити ієрархію класів ВЕКТОР та БЕЗПЕЧНИЙ ВЕКТОР з перевіркою виходу індексу за межі одновимірного масиву. Безпечний вектор визначає змінні: нижню і верхню межу (індекси). Перевизначити виведення у потік і введення з потоку, конструктор копіювання, операцію присвоювання через відповідні функції базового класу.

8. Створити класи транспортних засобів: АВТОМОБІЛЬ, ВАНТАЖІВКА, ПАРОПЛАВ і ЛІТАК. Створити з них ієрархію. В основу ієрархії покласти клас ТРАНСПОРТНИЙ ЗАСІБ зі спільними для усіх цих класів елементами. Визначити функції виведення, конструктори і деструктори.

9. Створити клас РІДИНА, що має назву (вказівник на рядок), густину. Визначити конструктори, деструктор і функцію виведення даних. Створити `public`-похідний клас – БЕЗАЛКОГОЛЬНИЙ НАПІЙ, що має колір та ознаку смаку. Визначити конструктори за

замовчуванням і зрізним числом параметрів, деструктори, функцію виведення. Визначити функції зміни густини, кольору та ознаки смаку.

10. Використовуючи ієрархію та успадкування, створити скласи ВІКНА, ВІКНА З ЗАГОЛОВКОМ і ВІКНА З КНОПКОЮ. Визначити необхідні конструктори, деструктори та метод зміни назви заголовку вікна. Зімітувати натиснення кнопки для закривання вікна.

11. Створити ієрархію класів ОСОБА, СТУДЕНТ і СТУДЕНТ-ДИПЛОМНИК. Перевизначити виведення у потік і введення з потоку, визначити конструктор копіювання, операцію присвоєння через відповідні функції базового класу.

12. Створити клас ОСОБА, що має ім’я (вказівник на рядок), вік, вагу. Визначити конструктори, деструктор і функцію виведення. Створити `public`-похідний клас ШКОЛЯР, який має рік навчання. Визначити конструктори за замовчуванням та з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення віку і класу.

13. Створити базовий клас – ЧЕРГА з двома кінцями. Елементи можуть вилучатися і додаватися з будь-якого кінця. Створити похідний клас ЗВИЧАЙНА_ЧЕРГА. Класи повинні мати конструктори, зокрема конструктор копіювання, деструктори, перевантажені функції виведення у потік і введення з потоку.

2. Самостійна робота

/* 1. rozdil 16, listing 1 — рівень доступу до членів базового класу public. Всі відкриті і захищені члени базового класу стають відкритими і захищеними членами похідного класу */

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

class base {
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};
class derived : public base {
  int k;
public:
  derived(int x) { k=x; }
  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob(3);

  ob.set(1, 2); // access member of base
  ob.show(); // access member of base

  ob.showk(); // uses member of derived class
  return 0;
}
```

1 2
3

/* 2. listing 2 - рівень доступу до членів базового класу private. Всі відкриті і захищені члени базового класу стають закритими членами базового класу */

```
// This program won't compile.
#include <iostream>
using namespace std;

class base {
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n";}
};

// Public elements of base are private in derived.
class derived : private base {
  int k;
public:
  derived(int x) { k=x; }
  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob(3);

  ob.set(1, 2); // error, can't access set()
  ob.show(); // error, can't access show()
```

```
  return 0;
}
```
c16_2.cpp:9: error: ‘void base::set(int, int)’ is inaccessible
c16_2.cpp:25: error: within this context
c16_2.cpp:25: error: ‘base’ is not an accessible base of ‘derived’
c16_2.cpp:10: error: ‘void base::show()’ is inaccessible
c16_2.cpp:26: error: within this context

/* 3. listing 3 - рівень доступу до членів базового класу public. Захищені члени базового класу стають захищеними членами похідного класу */

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

class base {
protected:
  int i, j; // private to base, but accessible by derived
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

class derived : public base {
  int k;
public:
  // derived may access base's i and j
  void setk() { k=i*j; }

  void showk() { cout << k << "\n"; }
};

int main()
{
  derived ob;

  ob.set(2, 3); // OK, known to derived
  ob.show(); // OK, known to derived

  ob.setk();
  ob.showk();

  return 0;
}
```
2 3
6

/* 4. listing 4 — відкрите успадкування захищених членів базового класу похідними класами*/

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

class base {
protected:
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

// i,j успадковуються, як захищені члени.
class derived1 : public base {
  int k;
public:
  void setk() { k = i*j; } // legal
  void showk() { cout << k << "\n"; }
```

```
};

// i, j успадковуються, як захищені члени через успадкований клас derived1.
class derived2 : public derived1 {
  int m;
public:
  void setm() { m = i-j; } // legal
  void showm() { cout << m << "\n"; }
};

int main()
{
  derived1 ob1;
  derived2 ob2;

  ob1.set(2, 3);
  ob1.show();
  ob1.setk();
  ob1.showk();

  ob2.set(3, 4);
  ob2.show();
  ob2.setk();
  ob2.setm();
  ob2.showk();
  ob2.showm();

  return 0;
}
2 3
6
3 4
12
-1

/* 5. listing 5 -  закрите успадкування базового класу */
// Програма не компілюється
#include <iostream>
using namespace std;

class base {
protected:
  int i, j;
public:
  void set(int a, int b) { i=a; j=b; }
  void show() { cout << i << " " << j << "\n"; }
};

// Всі елементи базового класу є закритими членами класу derived1.
class derived1 : private base {
  int k;
public:
  // це можна робити, оскільки i, j є закритими членами класу derived1
  void setk() { k = i*j; } // OK
  void showk() { cout << k << "\n"; }
};

// доступ до членів i, j, set(), and show() не успадковуэться
class derived2 : public derived1 {
  int m;
public:
  // нема доступу до i,j, оскільки i, j є  закритими членами derived1
  void setm() { m = i-j; } // Помилка
  void showm() { cout << m << "\n"; }
```

```
};

int main()
{
  derived1 ob1;
  derived2 ob2;
  ob1.set(1, 2); // помилка, не можна викликати функцію set()
  ob1.show(); // помилка, не можна викликати функцію show()
  ob2.set(3, 4); // помилка, не можна викликати функцію set()
  ob2.show(); // помилка, не можна викликати функцію show()
  return 0;
}
```

```
c16_5.cpp:8: error: ‘int base::i’ is protected
c16_5.cpp:28: error: within this context
c16_5.cpp:8: error: ‘int base::j’ is protected
c16_5.cpp:28: error: within this context
c16_5.cpp: In function ‘int main()’:
c16_5.cpp:10: error: ‘void base::set(int, int)’ is inaccessible
c16_5.cpp:37: error: within this context
c16_5.cpp:37: error: ‘base’ is not an accessible base of ‘derived1’
c16_5.cpp:11: error: ‘void base::show()’ is inaccessible
c16_5.cpp:38: error: within this context
```

```
/* 6. listing 6 — захищене успадкування */
#include <iostream>
using namespace std;

class base {
protected:
  int i, j; // закриті члени класу base, доступні класу derived
public:
  void setij(int a, int b) { i=a; j=b; }
  void showij() { cout << i << " " << j << "\n"; }
};

// Клас, отриманий за допомогою захищеного успадкування
class derived : protected base{
  int k;
public:
  // клас derived має доступ до членів i, j і setij() з класу base.
  void setk() { setij(10, 12); k = i*j; }

  // звідси можна викликати функцію showij()
  void showall() { cout << k << " "; showij(); }
};

int main()
{
  derived ob;

//  ob.setij(2, 3); // невірно, setij() є захищений член класу derived

  ob.setk(); // вірно, викликається відкритий член класу derived
  ob.showall(); // вірно, викликається відкритий член класу derived

// ob.showij(); // невірно, функція showij() є захищени членом класу derived
  return 0;
}
```

```
120 10 12
```

```
/* 7. listing 8 — послідовність виклику конструкторів і деструкторів */
#include <iostream>
using namespace std;
```

```
class base {
public:
  base() { cout << "Створюється об'єкт класу base\n"; }
  ~base() { cout << "Знищується об'єкт класу base\n"; }
};

class derived: public base {
public:
  derived() { cout << "Створюється об'єкт класу derived\n"; }
  ~derived() { cout << "Знищується об'єкт класу  derived\n"; }
};

int main()
{
  derived ob;

  // тільки створюється і знищується об'єкт

  return 0;
}
```

Створюється об'єкт класу base
Створюється об'єкт класу derived
Знищується об'єкт класу derived
Знищується об'єкт класу base

```
/* 8. listing 10 — виклик конструкторів при ієрархічному успадкуванні */
#include <iostream>
using namespace std;

class base {
public:
  base() { cout << "Створення базового класу base\n"; }
  ~base() { cout << "Знищення базового класу base\n"; }
};

class derived1 : public base {
public:
  derived1() { cout << "Створення базового класу derived1\n"; }
  ~derived1() { cout << "Знищення базового класу derived1\n"; }
};

class derived2: public derived1 {
public:
  derived2() { cout << "Створення базового класу derived2\n"; }
  ~derived2() { cout << "Знищення базового класу derived2\n"; }
};

int main()
{
  derived2 ob;

  // construct and destruct ob

  return 0;
}
```

Створення базового класу base
Створення базового класу derived1
Створення базового класу derived2
Знищення базового класу derived2
Знищення базового класу derived1
Знищення базового класу base

```
/* 9. listing 12 -  виклик конструкторів при множинному успадкуванні */
```

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

class base1 {
public:
  base1() { cout << "Створення об'єкта класу  base1\n"; }
  ~base1() { cout << "Знищення об'єкта класу  base1\n"; }
};

class base2 {
public:
  base2() { cout << "Створення об'єкта класу base2\n"; }
  ~base2() { cout << "Знищення об'єкта класу base2\n"; }
};

class derived: public base1, public base2 {
public:
  derived() { cout << "Створення о'бєкта класу derived\n"; }
  ~derived() { cout << "Знищення об'єкта класу derived\n"; }
};

int main()
{
  derived ob;
  // створення і знищенння об'єкта ob
  return 0;
}
```

Створення об'єкта класу base1
Створення об'єкта класу base2
Створення о'бєкта класу derived
Знищення об'єкта класу derived
Знищення об'єкта класу base2
Знищення об'єкта класу base1

```
/* 10. listing 16 — передача параметрів конструктору базового класу */
#include <iostream>
using namespace std;

class base {
protected:
  int i;
public:
  base(int x) { i=x; cout << "Створення об'єкта класу base\n"; }
  ~base() { cout << "Знищення об'єкта класу base\n"; }
};

class derived: public base {
  int j;
public:
  // клас derived використовує x; а y передається класу base
  derived(int x, int y): base(y)
    { j=x; cout << "Створення об'єкта класу derived\n"; }

  ~derived() { cout << "Знищення об'єкта класу derived\n"; }
  void show() { cout << i << " " << j << "\n"; }
};

int main()
{
  derived ob(3, 4);
  ob.show(); // висвічує 4 3
  return 0;
}
```

Створення об'єкта класу base

```
Створення об'єкта класу derived
4 3
Знищення об'єкта класу derived
Знищення об'єкта класу base
```

```
/* 11. listing 17 - передача параметрів конструкторам базових класів при множинному успадкуванні */
#include <iostream>
using namespace std;

class base1 {
protected:
  int i;
public:
  base1(int x) { i=x; cout << "Створення об'єкту base1\n"; }
  ~base1() { cout << "Знищення об'єкту base1\n"; }
};

class base2 {
protected:
  int k;
public:
  base2(int x) { k=x; cout << "Створення об'єкту base2\n"; }
  ~base2() { cout << "Знищення об'єкту base1\n"; }
};

class derived: public base1, public base2 {
  int j;
public:
  derived(int x, int y, int z): base1(y), base2(z)
    { j=x; cout << "Створення об'єкту derived\n"; }

  ~derived() { cout << "Знищення об'єкту derived\n"; }
  void show() { cout << i << " " << j << " " << k << "\n"; }
};

int main()
{
  derived ob(3, 4, 5);

  ob.show(); // висвічує 4 3 5

  return 0;
}
```

```
Створення об'єкту base1
Створення об'єкту base2
Створення об'єкту derived
4 3 5
Знищення об'єкту derived
Знищення об'єкту base1
Знищення об'єкту base1
```

```
/* 12. listing 18 — конструктор похідного класу не має власних параметрів */
#include <iostream>
using namespace std;

class base1 {
protected:
  int i;
public:
  base1(int x) { i=x; cout << "Створення об'єкту base1\n"; }
  ~base1() { cout << "Знищення об'єкту base1\n"; }
};
```

```
class base2 {
protected:
  int k;
public:
  base2(int x) { k=x; cout << "Створення об'єкту base2\n"; }
  ~base2() { cout << "Знищення об'єкту base2\n"; }
};

class derived: public base1, public base2 {
public:
  /* Конструктор класу Derived не має параметрів */

  derived(int x, int y): base1(x), base2(y)
    { cout << "Створення об'єкту derived\n"; }

  ~derived() { cout << "Знищення об'єкту derived\n"; }
  void show() { cout << i << " " << k << "\n"; }
};

int main()
{
  derived ob(3, 4);

  ob.show(); // висвічує 3 4

  return 0;
}
```

Створення об'єкту base1
Створення об'єкту base2
Створення об'єкту derived
3 4
Знищення об'єкту derived
Знищення об'єкту base2
Знищення об'єкту base1

```
/* 13. listing 22 — відновлення закритого статусу закритих членів */
#include <iostream>
using namespace std;

class base {
  int i; // закритий член класу base
public:
  int j, k;
  void seti(int x) { i = x; }
  int geti() { return i; }
};

// закрите успадкування класу  base
class derived: private base {
public:
  /* Відновлення відкритого статусу членів
     j,  seti(), geti() */
  base::j;    // змінна j відкрита, а k  - ні
  base::seti; // функція seti() відкрита
  base::geti; // функція geti() відкрита

// base::i; // невірно, рівень доступу піднімати не можна

  int a; // відкритий член
};

int main()
{
  derived ob;
```

```
//ob.i = 10; // невірно, так як змінна i є закритим членом класу derived

  ob.j = 20; // невірно, так як змінна j відкрита в класі derived
//ob.k = 30; // невірно, так як змінна k є закритим членом в класі  derived

  ob.a = 40; // вірно, так як a є відкритим членом класа derived
  ob.seti(10);

  cout << ob.geti() << " " << ob.j << " " << ob.a;

  return 0;
}
```

10 20 40

/* 14. listing 25 — застосування оператора розкриття області видимості для усунення неоднозначності для доступу до членів базового класу */

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

class base {
public:
  int i;
};

// клас derived1 успадковує клас  base
class derived1 :  public base {
public:
  int j;
};

// клас derived2 успадковує клас  base
class derived2 : public base {
public:
  int k;
};

/* клас derived3 успадковує класи  derived1 і derived2 і має дві копії класу base
*/
class derived3 : public derived1, public derived2 {
public:
  int sum;
};

int main()
{
  derived3 ob;

  ob.derived1::i = 10; // для усунення неоднозначності використовується змінна i з
класу derived1
  ob.j = 20;
  ob.k = 30;

  // неоднозначність усунена
  ob.sum = ob.derived1::i + ob.j + ob.k;

  // неоднозначність усунена
  cout << ob.derived1::i << " ";

  cout << ob.j << " " << ob.k << " ";
  cout << ob.sum;

  return 0;
}
```

10 20 30 60

```
/* string 26  використання віртуального базового класу  */
#include <iostream>
using namespace std;

class base {
public:
  int i;
};

// клас derived1 успадковує віртуальний клас base
class derived1 : virtual public base {
public:
  int j;
};

// клас derived2 успадковує віртуальний клас base
class derived2 : virtual public base {
public:
  int k;
};

/* клас derived3 успадковує класи  derived1 і derived2.
   Але в цьому випадку він має одну копію базового класу */
class derived3 : public derived1, public derived2 {
public:
  int sum;
};

int main()
{
  derived3 ob;

  ob.i = 10; // неоднозначність відсутня
  ob.j = 20;
  ob.k = 30;

  // неоднозначність відсутня
  ob.sum = ob.i + ob.j + ob.k;
  // неоднозначність відсутня
  cout << ob.i << " ";
  cout << ob.j << " " << ob.k << " ";
  cout << ob.sum;

  return 0;
}
```

10 20 30 60

Лабораторна робота №7. Множинне успадкування класів

Мета роботи: вивчення особливостей множинного успадкування класів.
Теоретичний матеріал лекції, [1, розділ 8], [2, розділ 16, 17], [3, розділи 8, 13].

1. Короткі теоретичні відомості

1.1. Оголошення множинного успадкування

Один клас C++ може успадковувати елементи декількох інших класів. Це забезпечує поєднання властивостей декількох класів в рдному класі.

Оголошення множинного успадкування:

```
class Derived: режим успадкування_1 Base1,
               /* … */
               режим успадкування_1 BaseN
{ … };
```

Семантично множинне успадкування ґрунтується на відношенні “є”: похідний клас у деякому сенсі є екземпляром усіх базових класів.

1.2. Порядок виклику конструкторів і деструкторів

При множинному успадкуванні конструктор похідного класу викликає конструктори базових класів для ініціалізації успадкованих елементів даних. Порядок виклику конструкторів:

- викликаються конструктори базових класів за послідовністю їх множинного успадкування; віртуальні базові класи ініціалізуються перед невіртуальними базовими класами;
- якщо базовий клас є контейнерним, то спочатку будуть викликані конструктори для інкапсульованих у нього об’єктів за послідовністю їх розміщення у протоколі класу, а потім – відповідний конструктор базового класу;
- для тих базових класів, які не вказані у списку ініціалізації конструктора похідного класу, викликаються конструтори за замовчуванням (без параметрів);
- викликаються конструтори об’єктів, інкапсульованих у похідний клас;
- на завершення викликається конструктор похідного класу.

1.3. Доступ до перекритих елементів класі

Оголошуючи власні елементи даних та методи, похідний клас може перекривати однойменні елементи, успадковані від базових класів. Тоді доступ до перекритих методів відбувається через імена базових класів то операції дозволу доступу ‘::’.

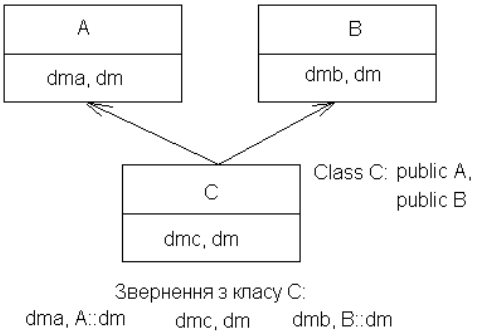

Рис.1. Діаграма множинного успадкування

1.4. Неоднозначність параметричного перевантаження методів

У разі множинного успадкування методів, що мають однакове ім'я, але відрізняються типами параметрів, може виникати неоднозначність їх виклику.

```
class A {
public:
   void output(int i) {cout<<"A::output()="<<i<<endl;}
};
class B {
public:
   void output(double d) {cout<<"B::output()="<<d<<endl;}
};
class C: public A, public B { . . . };
int main() {
  C c;
  c.output(5);
  c.output(3.14);
  return 0;
}
```

При множинному успадкуванні виникає неоднозначність виклику методів `output(int)` та `output(double)` з класу `C` у зв'язку з неявним перетворенням типів їх аргументів. Значення цілого типу перетворюється до дійсного, а дійсне значення – до цілого, тому компілятор не може визначити, який з двох методів необхідно викликати.

Для уникнення неоднозначностей необхідно у класі C перевизначити успадковані методи:

```
class C: public A, public B {
public:
    void output(int i)    {A::output(i);}
    void output(double d) {B::output(d);}
};
```

В результаті виклики методів відбуваються правильно.

1.5. Множинне успадкування класів із загальною базою

На практиці можливі випадки, коли декілька класів успадковують один і той самий клас. Такий варіант називають успадкуванням із загальною базою. При цьому у похідний клас потрапить декілька екземплярів елементів загального базового класу. Тоді для звернення до множинно успадкованих елементів базового класу можна використати назву одного із похідних класів та операцію дозволу доступу.

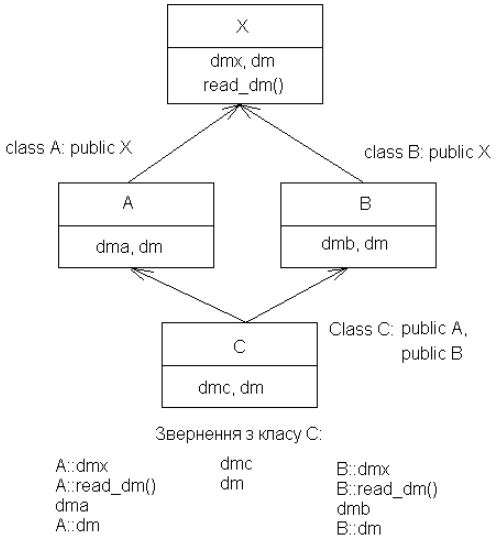

Рис.2. Діаграма множинного успадкування класів із загальною базою

Ситуація ускладнюється, коли перекриваються однойменні елементи даних або однойменні методи класів (таким полем є `dm`). Тому для забезпечення доступу з похідного класу `C` до поля `dm` класу `X` необхідна наявність методу `X::read_dm()`, який не перекривається у

класах A та B.

Клас C не може виконати пряме успадкування класу X. Посиланню на клас X не можна присвоїти об'єкт класу C, а вказівнику на клас X – адресу об'єкта класу C. Однак, для public-успадкування дозволяється ланцюжок присвоєнь від похідного класу C множинного успадкування через проміжний клас A або B до загального базового класу X:

```
X x3=a=c;     //  або  X x3=b=c;
```

Елементи-дані і методи класу X попадають у клас C у двох екземплярах.

1.6. Віртуальне успадкування класів

Щоб існував тільки один екземпляр елементів базового класу при множинному успадкуванні із загальною базою, використовують механізм віртуального успадкування. Для цього при прямому успадкуванні базового класу X похідними класами A та B разом з режимом успадкування вказується режим `virtual`. У цьому випадку для доступу до елементів віртуального базового класу не потрібно використовувати операцію доступу через проміжний клас, оскільки вони існують тільки в одному екземплярі.

При віртуальному успадкуванні об'єкт класу X можна проініціалізувати об'єктом класу C.

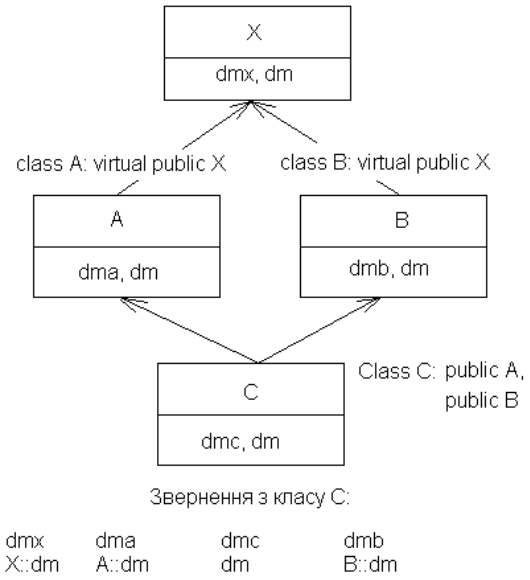

Рис. 3. Діаграма множинного успадкування з віртуальним базовим класом

1.7. Віртуальні методи множинного успадкування

1.7.1. Безпосереднє множинне успадкування

Множинне успадкування підтримує механізм пізнього зв'язування за допомогою віртуальних методів, вказівників або посилань на клас. Нехай у базових класах A та B визначено віртуальні методи `output()`. Клас C множинно успадковує класи A та B і визначає власний віртуальний метод `C::output()`. Для підтримування пізнього зв'язування використовується тільки `public`-успадкування базових класів.

У функції `main()` посилання на базові класи A та B ініціалізуються назвою об'єкта класу C і використовуються для виклику віртуального методу `C::output()`. Завдяки пізньому зв'язуванню ці посилання вказують на похідний клас C, і за їх допомогою викликається віртуальний метод `C::output()`.

```
#include <iostram>
#include <typeinfo.h>
using namespace std;
```

```
class A {
protected:
   int dma;
public:
   A(int x):dma(x) {cout<<"A::A(int)"<<endl;}
   virtual void output() {cout<<dma<<endl;}
};
class B {
protected:
   int dmb;
public:
   B(int x):dmb(x) {cout<<"B::B(int)"<<endl;}
   virtual void output() {cout<<dmb<<endl;}
};
class C: public A, public B {
private:
   int dmc;
public:
   C(int x1, int x2, int x3): A(x1),B(x2),dmc(x3)
    {cout<<"C::C(int,int,int)"<<endl;}
   virtual void output() {cout<<dma<<' '<<dmb<<' '<<dmc<<endl;}
};
int main() {
   C c(1,2,3);
   A& a=c;
   B& b=c;
   cout<<typeid(a).name()<<endl;  // class C
   cout<<typeid(b).name()<<endl;  // class C
   a.output();       // 1 2 3
   b.output();       // 1 2 3
   c.A::output();    // 1
   c.B::output();    // 2
   c.output();       // 1 2 3
}
```

1.7.2. Множинне успадкування зі спільним базовим класом

Нехай задано схему множинного успадкування класом `C` класів `A` та `B`, які в свою чергу успадковують спільний базовий клас `X`. Кожен клас визначає віртуальний метод `output()` для виведення на екран його елементів даних. Посилання на класи `A` та `B`, проініціалізовані іменем об'єкту класу `C`, викликають віртуальний метод `C::output()` з похідного класу `C`.

Посилання на клас `X` не може бути проініціалізоване об'єктом класу `C` у зв'язку з неоднозначністю перетворення типу, оскільки успадкування класу `X` відбулося двома шляхами – через клас `A` і через клас `B`.

Однак є можливим ланцюжок присвоєнь: посилання на базовий клас `X` можна проініціалізувати посиланням на клас `A` або `B`, які, в свою чергу, ініціалізуються об'єктом класу `C`. Так проініціалізоване посилання на клас `X` вказуватиме на клас `C` і за його допомогою буде викликаний віртуальний метод `C::output()` класу множинного успадкування `C`.

```
#include <iostream>
#include <typeinfo.h>
using namespace std;
class X {
protected:
   int dmx;
public:
   X(int x=0):dmx(x){cout<<"X::X()"<<endl;
   virtual void output() {cout<<dmx<<endl;}
```

```
};
class A: public X {
protected:
   int dma;
public:
   A(int x1, int x2):X(x1), dma(x2){cout<<”A::A()”<<endl;}
   virtual void output {cout<<dmx<<’ ’<<dma<<endl;
};
class B: public X {
protected:
   int dmb;
public:
   B(int x1, int x2):X(x1), dmb(x2){cout<<”B::B()”<<endl;}
   virtual void output {cout<<dmx<<’ ’<<dmb<<endl;
};
class C:public A, public B {
private:
   int dmc;
public:
   C(int x1, int x2, int x3, int x4, int x5):
     A(x1,x2),B(x3,x4),dmc(x5) {cout<<”C::C()”<<endl;}
     virtual void output()
      {cout<<A::dmx<<’ ’dma<<’ ’<<B::dmx<<’ ’<<dmb<<’ ’<< dmc<<endl;{
};
int main() {
   C c(1,2,3,4,5);
   A& a=c;
   B& b=c;
   a.output();     // 1 2 3 4 5
   b.output();     // 1 2 3 4 5
   c.A::output();  // 1 2
   c.B::output();  // 3 4
   c.output();     // 1 2 3 4 5
   X& x2=a;        // посилання на клас C
   X2.output();    // 1 2 3 4 5
   X& x3=b;        // посилання на клас C
   x3.output();    // 1 2 3 4 5
```

1.7.3. Множинне успадкування з віртуальним базовим класом

За множинного успадкування з віртуальним базовим класом посилання на клас X може бути проініціалізоване об’єктом класу C. Неоднозначності перетворення типів не виникає, оскільки успадкування класу X відбулося тільки один раз.

Так проініціалізоване посилання на клас X можна використати для виклику віртуального методу з класу C. Замість посилання можна використати вказівник на клас X, проініціалізований адресою об’єкта класу C.

```
#include <iostream>
using namespace std;
class X {
protected:
   int dmx;
```

```
public:
   X(int x=0):dmx(x){cout<<"X::X()"<<endl;}
   virtual void output() {cout<<dmx<<endl;}
};
class A: virtual public X {
protected:
   int dma;
public:
   A(int x1, int x2):X(x1), dma(x2){cout<<"A::A()"<<endl;}
   virtual void output() {cout<<dmx<<' '<<dma<<endl;}
};
class B: virtual public X {
protected:
   int dmb;
public:
   B(int x1, int x2):X(x1), dmb(x2){cout<<"B::B()"<<endl;}
   virtual void output() {cout<<dmx<<' '<<dmb<<endl;}
};
class C:public A, public B {
private:
   int dmc;
public:
   C(int x1, int x2, int x3, int x4, int x5):
     A(x1,x2),B(x3,x4),dmc(x5) {cout<<"C::C()"<<endl;}
     virtual void output()
      {cout<<A::dmx<<' '<<dma<<' '<<B::dmx<<' '<<dmb<<' '<< dmc<<endl;}
};
int main() {
   C c(1,2,3,4,5);
   A& a=c;
   B& b=c;
   a.output();      // 0 2 0 4 5
   b.output();      // 0 2 0 4 5
   c.output();      // 0 2 0 4 5
   X& x1=c;         // посилання на клас C
   x1.output();     // 0 2 0 4 5
   x1.X::output(); // 0
return 0;
}
X::X()
A::A()
B::B()
C::C()
```

Запитання.

1. Як оголосити множинне успадкування.
2. Порядок виклику конструкторів і деструкторів при множинному успадкуванні.
3. Доступ до перекритих елементів класу.
4. Неоднозначність параметричного пере визначення методів.
5. Множинне успадкування класів із загальною базою.

6. Віртуальне успадкування класів.
7. Безпосереднє множинне успадкування.
8. Множинне успадкування зі спільним базовим класом.
9. Множинне успадкування з віртуальним базовим класом.

Завдання.

Сформувати варіант успадкування класів, вилучивши із зображеного на рис. 4 орієнтованого графу вершини, номер `k` якої визначається як порядковий номер `n` студента в алфавітному списку групи за модулем 12 (`k=n mod 12`) та вершину з номером 12-`k`. Нумерація вершин здійснюється послідовно, спочатку зліва направо, а потім зверху вниз. Разом з вершиною вилучаються усі її вхідні та вихідні ребра. Біля стрілок вказано режими успадкування класів відповідно до варіанта завдання.

У кожному класі визначити дані та методи роботи з ними. Визначити дані або методи, які перекриваються у похідних класах. Кожен клас повинен містити конструктор ініціалізації, копіювання, деструктор, функцію для встановлення та читання значень даних, перевантажені операції для введення і виведення даних. Функція `main()` повинна ілюструвати доступ до даних та методів у кожному класі ієрархії успадкування.

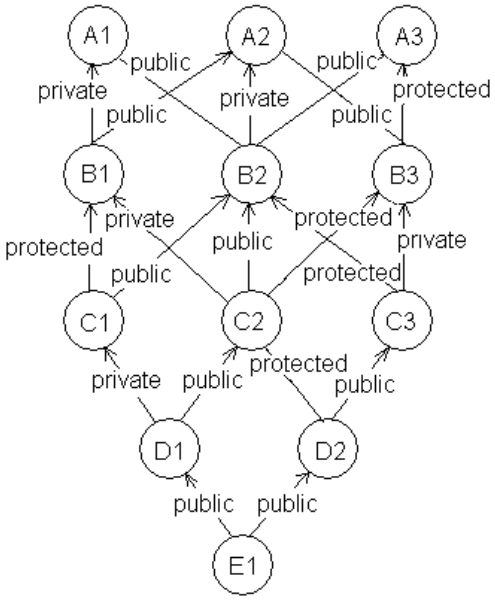

Рис. 7.4. Варіанти множинного успадкування

2. Самостійна робота

1. Приклад програми.

Створити два класи: Рамка (координати головної діагоналі) та Меню (масив рядків символів з назвами команд меню та індекс активної команди меню).

Використовуючи множинне успадкування цих класів, створити новий клас – Вікно з рамкою та меню. Додатково задати назву заголовків вікна та колір фону (рядки символів). Визначити необхідні дані, методи для роботи з даними, конструктори та деструктори, операторні функції введення-виведення даних.

```
// 7_1.cpp
// g++ 7_1.cpp –o 7_1  - компіляції і компонування програми
#include <iostream>
#include <cstring>
using namespace std;

class Frame {
protected:
   int x1,y1,x2,y2;  // координати рамки
public:
  Frame(int x1,int y1,int x2,int y2)
  {
    this->x1=x1; this->y1=y1; this->x2=x2; this->y2=y2;
  }
  // конструктор копіювання
  Frame(Frame& f)
  {
    x1=f.x1; y1=f.y1; x2=f.x2; y2=f.y2;
  }
// операція введення з потоку
  friend istream& operator>>(istream& is, Frame& f)
  {
    is>>f.x1>>f.y1>>f.x2>>f.y2;
    return is;
  }
// операція виведення у потік
  friend ostream& operator<<(ostream& os, Frame& f)
  {
    os<<f.x1<<' '<<f.y1<<' '<<f.x2<<' '<<f.y2<<endl;
    return os;
  }
};
// клас меню
class Menu {
protected:
  int count;       // кількість команд
  int active;      // номер активної команди
  char ** command; // назва команди
public:
// конструктор
  Menu(int count, int active, char **command) {
    this->count=count;
    this->active=active;

    this->command=new char*[count];
    for(int i=0; i<count; i++)
    {
      this->command[i]=new char[20];
      strcpy(this->command[i], command[i]);
    }
  }
// конструктор копіювання
  Menu(Menu& m) {
    count=m.count;
```

```
    active=m.active;

    command=new char*[m.count];
    for(int i=0; i<count; i++)
    {
      command[i]=new char[20];
      strcpy(command[i],m.command[i]);
    }
  }
//деструктор
  ~Menu()
  {
    for(int i=0; i<count; i++) delete []command[i];
  }
// операція введення з потоку
  friend istream& operator>>(istream& is, Menu& m)
  {
    for(int i=0;i<m.count;i++) is>>m.command[i];
    return is;
  }
// операція виведення у потік
  friend ostream& operator<<(ostream& os, Menu& m)
  {
    for(int i=0;i<m.count;i++) os<<m.command[i]<<endl;
    return os;
 }
};

// клас вікна
class Window: public Frame, public Menu
{
protected:
  char *title;    // заголовок вікна
  int bkcolor;    // колір фону
public:
// конструктор
Window(Frame f,Menu m,char *title,int bkcolor):Frame(f),Menu(m)
{
  this->title=new char[100];
  strcpy(this->title,title);
  this->bkcolor=bkcolor;
}
// деструктор
~Window()
{
  delete []title;
}
// операція введення з потоку
friend istream& operator>>(istream& is, Window& w)
{
  is>>w.x1>>w.y1>>w.x2>>w.y2;
  for(int i=0; i<w.count; i++) is>>w.command[i];
  is>>w.title;
  is>>w.bkcolor;
  return is;
}

// операція виведення у потік
friend ostream& operator<<(ostream& os, Window& w)
{
  Frame f(w.x1,w.y1,w.x2,w.y2),fr=f;
  os<<f;
  Menu m(w.count,w.active,w.command),mr=m;
  os<<m;
```

```
  os<<w.title<<endl;
  os<<w.bkcolor<<endl;
  return os;
}

};

// головна функція
int main()
{
  char
s0[8]="File",s1[8]="Edit",s2[8]="View",s3[8]="Project",s4[8]="Build",s5[8]="Help";
  //char * command[6]={"File","Edit","View","Project","Build","Help"};
  char * command[6]={s0,s1,s2,s3,s4,s5};
  enum colors {RED, GREEN, BLUE, WHITE, BLACK};
  Frame f(1,1,80,25),&fr=f;
  cout<<"Склад об'єкта Frame\n"<<fr;
  Menu m(6,0,command),&mr=m;
  cout<<"Склад об'єкта Menu\n"<<mr;
  char s6[10]="MyWindow";
  Window w(fr,mr,s6,WHITE),&wr=w;
  cout<<"Склад об'єкта Window\n"<<wr;
}
>./7_1
Склад об'єкта Frame
1 1 80 25
Склад об'єкта Menu
File
Edit
View
Project
Build
Help
Склад об'єкта Window
1 1 80 25
File
Edit
View
Project
Build
Help
MyWindow
3

/* 2. rozdil 16, listing 7 — множинне успадкування */
#include <iostream>
using namespace std;

class base1 {
protected:
  int x;
public:
  void showx() { cout << x << "\n"; }
};

class base2 {
protected:
  int y;
public:
  void showy() {cout << y << "\n";}
};

// множинне успадкування базових класів
class derived: public base1, public base2 {
```

```
public:
  void set(int i, int j) { x=i; y=j; }
};

int main()
{
  derived ob;

  ob.set(10, 20); // функція належить класу  derived
  ob.showx(); // функція належить класу base1
  ob.showy(); // функція належить класу base2

  return 0;
}
10
20
```

Лабораторна робота № 8. Віртуальні функції і поліморфізм

Мета роботи: вивчення віртуальних функцій і тирів поліморфізму.
Теоретичний матеріал лекції, [1, розділ 7], [2, розділ 17], [3, розділ 13].

1. Короткі теоретичні відомості

1.1. Види поліморфізму

Поліморфізм – це використання під одним іменем різних функцій, призначених для опрацювання даних різних типів. У C++ є такі типи поліморфізму:

• *статичний* – перевантаження і перевизначення функцій та операцій, шаблони функцій та класів;

• *динамічний* – перевизначення віртуальних функцій.

Статичний поліморфізм забезпечується під час компіляції програми, а динамічний – під час її виконання.

Поліморфізм *перевантаження функцій* (overload) полягає у можливості вибору компілятором різних реалізацій однієї і тієї ж самої функції, залежно від кількості або типів її аргументів. Наприклад:

```
#include <iostream>
using namespace std;
class A {
public:
  void f() { cout<<"A::f()"<<endl;}
  void f(int) { cout<<"A::f(int)"<<endl;}
  void f(double) { cout<<"A::f(double)"<<endl;}
};
int main() {
  A a;
  a.f();      // A::f()
  a.f(1);     // A::f(int)
  a.f(3.14);  // A::f(double)
  return 0;
}
```

```
A::f()
A::f(int)
A::f(double)
```

У програмі викликаються перевантажені методи `A::f()`.

Перевантаження операцій основується на операторних функціях `operator`. Приклад програми де перевантажується бінарна операція + і операція <<.

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

class A {
protected:
  int x;
public:
  A(int x=0) {this->x=x;}
  A operator+(A& a) {return A(x+a.x);}
  friend ostream& operator<<(ostream& os, A& a) {os<<a.x<<endl; return os;}
};
class B: public A {
protected:
  int y;
public:
  B(int x=0, int y=0) : A(x) {this->y=y;}
```

```
  B operator+(B& b) {return B(x+b.x, y+b.y);}
  friend ostream& operator<<(ostream& os, B& b) {os<<b.x<<' '<<b.y<<endl; return
os;}
};

int main() {
  A a1(1), a2(2), a3;
  a3=a1+a2;
  cout<<a3;    // 3

  B b1(3,4), b2(5,6), b3;
  b3=b1+b2;
  cout<<b3;    // 8 10
  return 0;
}
```

```
3
8 10
```

Функція демонструє поліморфне застосування перевантаження операцій. Залежно від типів операндів одні і ті самі операції + та << будуть викликані з класу A або з класу B.

У контексті успадкування класів поліморфізм полягає у перевизначенні (`override`) нащадком методів базового класу. Тоді для роботи з об'єктами похідних класів використовується інтерфейс їх базового класу.

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

class A {
public:
  void f() const {cout<<"A::f()"<<endl;}
};
class B: public A {
public:
  void f() const {cout<<"B::f()"<<endl;}
};
class C: public A {
public:
  void f() const {cout<<"C::f()"<<endl;}
};

int main() {
  C *p1 = new C;
  p1->f();       // C::f()
  A *p2 = p1;
  p2->f();       // A::f()

  B *p3 = new B;
  p3->f();       //  B::f()
  A *p4 = p3;
  p4->f();       // A::f()
  return 0;
}
```

```
C::f()
A::f()
B::f()
A::f()
```

У базовому класі визначено метод `f()`, який перевизначається у похідних класах `B` та `C`. У функції `main()` вказівник `p2` на клас `A` проініціалізовано вказівником `p1` на клас `C`. Така ініціалізація допускається без явного перетворення типу, оскільки клас `C` є похідним від класу `A`. Проініціалізовані вказівники використовуються для виклику метода `f()`. Незважаючи на те, що вказівники `p1` та `p2` набувають однакових значень, за допомогою `p1` буде викликано метод `C::f()`, а за допомогою `p2` – метод `A::f()`. Аналогічні дії відбуваються з вказівниками `p3` та `p4` на класи `B` та `A`. Те, який метод буде викликано за допомогою того чи іншого вказівника,

визначається під час компіляції (раннє зв'язування методів і об'єктів). Це означає, що за допомогою об'єкта класу, посилання або вказівника на клас можна викликати тільки методи цього самого класу. Якщо об'єкт, посилання або вказівник на базовий клас проініціалізовані об'єктом (або його адресою – для вказівника) похідного класу, то за їх допомогою не можна викликати методи похідного класу.

Розглянуті вище поліморфізму є *статичними*. При *динамічному* поліморфізмі вибір того або іншого методу здійснюється під час виконання програми (пізнє зв'язування реалізується за допомогою віртуальних методів). *Віртуальний метод* – це сукупність перевизначених функцій-членів, оголошених зі словом `virtual`.

Приклад поліморфізму віртуальних методів. Нехай клас `B` успадковує клас `A` і перевизначає метод `f()`. Якщо віртуальний метод викликається з об'єкта класу, то діє раннє зв'язування. Якщо для виклику віртуального методу використовується вказівник (або посилання), проініціалізований адресою (або іменем – для посилання) об'єкта похідного класу, то діє пізнє зв'язування і в результаті викликається метод з похідного класу.

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

class A {
public:
  virtual void f() const {cout<<"A::f()"<<endl;}
};

class B: public A {
public:
  virtual void f() const {cout<<"B::f()"<<endl;}
};

int main() {
  A a;
  a.f();  // раннє зв'язування, A::f()
  B b;
  b.f();  // раннє зв'язування, B::f()
  a=b;
  a.f();  // раннє зв'язування, A::f()

  A *p1 = new B;
  p1->f();       // пізнє зв'язування, B::f()
  A& p2 = *new B;
  p2.f();        // пізнє зв'язування, B::f()

   return 0;
}
```

1.2. Особливості віртуальних методів

Сукупність класів, у яких оголошується, визначається та перевизначається метод, називається *поліморфічним кластером*. Поліморфічний кластер реалізується тільки для *public-успадкувань* класів. У межах поліморфного кластера віртуальні методи мають однакову адресу. В одній ієрархії успадкування класфі може бути визначено декілька поліморфічних кластерів.

Особливості віртуальних методів:

- віртуальними можуть бути тільки функції-члени класу, друзі не можуть бути віртуальними;
- віртуальний метод може бути оголошений дружнім (`friend`) до іншого класу;
- віртуальні методи не можуть бути статичними (`static`);
- слово `virtual` достатньо записати тільки у першому оголошенні віртуальних методів;
- віртуальний метод повинен бути визначений у класі або оголошений чистим;

- для забезпечення пізнього зв'язування віртуальні методи повинні викликатися за допомогою вказівників або посилань. Вказівник на клас-предок ініціалізується адресою об'єкта-нащадка, а посилання на клас-предок ініціалізується об'єктом класу-нащадка.

- у межаж поліморфічного кластера віртуальний метод не повинен змінювати свій тип, кількість або типи параметрів. Якщо віртуальний метод не перевизначається у похідному класі, то діє віртуальний метод попереднього в ієрархії успадкування класу.

1.3. Виключення поліморфізму віртуальних методів

Для виключення дії механізму пізнього зв'язування при виклику віртуального методу необхідно вказати назву класу, до якого він належить:

```
вказівник->назва класу::назва_методу(аргументи);
посилання.назва_класу::назва_мнтоду(аргументи);
```

Наприклад:

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

class A {
public:
  virtual void f() {cout<<"A::f()"<<endl;}
};
class B: public A {
public:
  virtual void f() {cout<<"B::f()"<<endl;}
};

int main() {
  A *p =new B;
  p->A::f();  // A::f()
  p->f();     // B::f()
return 0;
}
```

В результаті викликається метод `A::f()` базового класу, а не метод `B::f()` похідного класу.

1.4. Коваріантні віртуальні методи

Віртуальний метод не повинен змінювати назву, кількість та типи параметрів і тип результату у межах свого поліморфного кластера.

Зміна типу результату можлива лише для коваріантних віртуальних методів. Методи є *коваріантними*, якщо їх типи є узгодженими з типами класів в іншій ієрархії успадкувань.

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

class B1 {
public:
  virtual void f1() {cout<<"B1::f1()"<<endl;}
};
class D1:public B1 {
public:
  virtual void f1() {cout<<"D1::f1()"<<endl;}
};

class B2 {
public:
  virtual B1* f2(){cout<<"B2::f2()"<<endl; return new B1;}
};

class D2:public B2 {
```

```
public:
  virtual D1* f2(){cout<<"D2::f2()"<<endl; return new D1;}
};

int main() {
  B2* p=new B2;
  p->f2()->f1();  // B2::f2()
                 // B1::f1()
  B2* q=new D2;
  q->f2()->f1();  // D2::f2()
                  // D1::f1()
  delete p;
  delete q;
  return 0;
}
```

У програмі оголошено дві ієрархії успадкувань класів: `B1<-D1` та `B2<-D2`. Віртуальний метод базового класу `B2` повертає вказівник на базовий клас `B1`, а віртуальний метод похідного класу `D2` – на похідний клас `D1`. Ці вказівники використано для викликів віртуальних методів.

Віптуальні методи є також коваріантними, якщо їхні типи є вказівниками або посиланнями на класи у межах дії поліморфічного кластера, наприклад:

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

class A {
protected:
  int x;
public:
  A(int x=0) {this->x=x;}
  virtual A& inc(){++x;return *this;}
  virtual void output() {cout<<x<<endl;}
};
class B:public A {
  int y;
public:
  B(int x=0, int y=0):A(x){this->y=y;}
  virtual B& inc() { ++x;++y; return *this;}
  virtual void output() {cout<<x<<' '<<y<<endl;}
};

int main() {
  A& a = *new B(2,7);
  a.inc();
  a.output();   // 3 8
  return 0;
}
```

Віртуальний метод `inc()` відрізняється типом результату в ієрархії успадкування класів. У класі `A` він має тип посилання на `A`, а у класі `B` – тип посилання на `B`. Ці типи є узгодженими в ієрархії успадкування класів (у межах поліморфічного кластера). Незважаючи на це, механізм дії пізнього зв'язування не порушується.

1.5. Віртуальні операторні методи

Операторні методи класів можуть бути віртуальними. Для ілюстрації цього модифікується попередня програма, в якій замінюється віртуальний метод `inc()` операторним методом префіксного інкременту.

```
#include <iostream>
using namespace std;
class A {
protected:
  int x;
```

```
public:
  A(int x=0) {this->x=x;}
  virtual A& operator++(){++x;return *this;}
  virtual void output() {cout<<x<<endl;}
};
class B:public A {
  int y;
public:
  B(int x=0, int y=0):A(x){this->y=y;}
  virtual B& operator++(){ ++x;++y; return *this;}
  virtual void output() {cout<<x<<' '<<y<<endl;}
};

int main() {
  A& a = *new B(2,7);
  ++a;
  a.output();   // 3 8
  return 0;
}
```

У функції `main()` посилання на базовий клас `A` проініціалізовано об'єктом похідного класу `B`. Операція префіксного інкременту, застосована до такого посилання, призведе до виклику віртуального операторного методу `B::operator++()` з похідного класу.

У цьому прикладі віртуальні методи мають коваріантні типи результатів. Оскільки посилання на базовий клас може бути проініціалізоване об'єктом похідного класу, то результат роботи не зміниться, якщо у класі `B` операторний метод матиме тип результату, тотожний типу результтату операторного методу з класу `A`.

1.6. Вказівники на віртуальні методи

Поліморфізм може бути реалізований також за допомогою вказівників на віртуальні методи. У наступній програмі віртуальний метод `Get()` визначає поліморфічний кластер у межах успадкування класів `A<-B`. Вказівник на базовий клас `A` проініціалізовано адресою об'єкта похідного класу `B` і використано для виклику віртуального методу `Get()` за допомогою вказівника `pf` на цей метод. Звернення до віртуального методу `Get()` за допомогою вказівника `pf` еквівалентне зверненню `p->Get()`.

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

class A {
protected:
  int x;
public:
  A(int x=0) {this->x=x;}
  virtual void Get() {cout<<x<<endl;}
};
class B:public A {
  int y;
public:
  B(int x=0, int y=0):A(x){this->y=y;}
  virtual void Get() {cout<<x<<' '<<y<<endl;}
};

void (A::*pf) ()=&A::Get; // PIPєP°P·C–PIPSPёPє PSP° PIC–CЂC,CґP°P»CЊPSPёPЅ
PjPµC,PsPґ

int main() {
  A *p = new B(1,2);
  //p->Get();
  (p->*pf) ();  //  1 2
  delete p;
```

```
  return 0;
}
```

1.7. Динамічні віртуальні методи

Динамічні віртуальні методи – це підклас віртуальних методів, який відрізняється способом виклику на етапі виконання. Оголошуються за допомогою індексу динамічного методу: `virtual void f(void)=[100];`

Індекс задається у квадратних дужках константою цілого типу. Він повинен бути унікальним серед індексів інших динамічних методів, але однаковим в ієрархії успадкування конкретного методу.

Якщо клас оголошує або успадковує віртуальні методи, то для його об'єктів автоматично під час компіляції створюється таблиця віртуальних методів (ТВМ). У ТВМ записуються адреси усіх віртуальних методів. Для виклику віртуальних методів спочатку з об'єкта читається адреса ТВМ, а потім з неї вибирається адреса потрібного віртуального методу та здійснюється її виклик.

Для роботи з динамічними віртуальними методами створюється додаткова таблиця динамічних віртуальних методів (ТДВМ). Адреса ТДВМ записується на початку ТВМ. У ТДВМ записуються тільки адреси тих динамічних віртуальних методів, які визначаються або перевизначаються на даному рівні успадкування. У результаті ТДВМ займає менший обсяг оперативної памяті, ніж ТВМ. Однак для пошуку адреси динамічного віртуального методу витрачається більше часу, ніж у ТВМ.

Функція `main()` демонструє виклик динамічного віртуального методу за допомогою посилання та вказівника на клас `A`, проініціалізованих відповідно іменем та адресою об'єкта класу `B`. В обох випадках викликається віртуальний метод з похідного класу `B`.

Примітка. Підтримка динамічних віртуальних методів у версії g++ 4.8.1 не реалізована.

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

class A {
public:
  virtual void f(void)=[100];
};
void A::f(void) { cout<<"A::f"<<endl;}

class B:public A {
public:
  virtual void f(void)=[100];
};
void B::f(void) { cout<<"B::f"<<endl;}

int main() {
 B b;
 A*p=&b;
 p->f();    // B::f()
 A &a=b;
 a.f();     // B::f()
  return 0;
}
```

1.8. Віртуальний деструктор

При роботі з вказівниками та посиланнями на тип базового класу, які ініціалізуються об'єктами похідних класів, у базовому класі бажано використовувати віртуальний деструктор. Віртуальний деструктор започатковує поліморфічний кластер. Якщо базовий клас містить віртуальний деструктор, то деструктор похідного від нього класу теж буде віртуальним.

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

```
using namespace std;

class A {
protected:
  int *p;
public:
  A(int x=0) {cout<<"A(int)"<<endl; p=new int(x); }

  virtual
 ~A() {cout<<"~A()"<<endl; delete p;}
  virtual void print(){cout<<*p<<endl;}
};

class B:public A {
  int *q;
public:
  B(int x=0,int y=0):A(x) {cout<<"B(int,int)"<<endl; q=new int(y);}
 ~B() {cout<<"~B()"<<endl; delete q;}
  virtual void print() {cout<<*p<<' '<<*q<<endl;}
};

int main() {
  //cout<<hex;
 // cout<<coreleft()<<endl;
  A *p=new B(1,2);
  p->print();
 // cout<<coreleft()<<endl;
  delete p;
  //cout <<coreleft()<<endl;
  return 0;
}

// A(int)
// B(int,int)
// 1 2
// ~B() СЦРєС‰Ps virtual ~A()
// ~A()
```

Якщо деструктор `~A()` не віртуальний, то при звільненні динамічної памяті, закріпленої за вказівником `p`, викликається тільки деструктор `~A()`, що може привести до втрати ресурсів похідного класу `B`.

Якщо деструктор `~A()` віртуальний, то викликаються деструктор `~B()`, а потім деструктор `~A()`, і об'єкт класу `B` повністю знищується. Зміну розміру динамічної памяті можна відстежити за допомогою функції `coreleft()` (в g++ відсутня).

1.9. Чисті віртуальні методи та абстрактні класи

Чисто віртуальна функція – це функція-член (метод) класу, для якої оголошено лише інтерфейс, а реалізація знаходиться в одному з похідних класів. Чистий віртуальний метод оголошується за допомогою специфікатор `= 0`.

Чистий віртуальний метод не може бути викликаний явно або неявно з конструктора.

Абстрактний клас містить один або декілька чистих віртуальних методів. Чистий абстрактний клас складається тільки з чистих віртуальних методів. Основне призначення абстрактного класу – це оголошення інтерфейсу похідних від нього класів.

Абстрактний клас може використовуватися тільки як базовий для інших класів. Неможливо оголосити об'єкт абстрактного класу, але можна оголосити вказівник на нього. Можна оголосити посилання на абстрактний клас, якщо воно ініціалізується об'єктом похідного класу.

Якщо похідний від абстрактного клас не визначає всіх чистих віртуальних методів, то він

теж є абстрактним.

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

class A {     // абстрактний клас
protected:
  int x;
public:
  A(int x1):x(x1){}
  virtual void output() const=0; // чистий ВМ
};

class B:public A {  // абстрактний клас
public:
  B(int x):A(x){}
  virtual void output() const=0;
};

class C:public B {
public:
  C(int x):B(x){}
  virtual void output() const {cout<<x<<endl;}
};

int main() {
  C c(3);
  c.output();    // 3
  A *p=&c;
  B *q=&c;
  p->output();   // 3
  q->output();   // 3
}
```

Параметри та результат функції не можуть мати тип абстрактного класу. Однак вони можуть бути вказівниками або посиланнями на абстрактний клас:

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

class B{
public:
  virtual void f()=0;
};
class D:public B {
public:
  virtual void f() {cout<<"D::f()"<<endl;}
};
B& ref_B(B* p, B& r) {
  //...
  return r;
}
int main() {
  B *p = new D;
  B &r = *new D;
  ref_B(p,r).f();
  delete p;
  delete &r;
  return 0;
}
// D::f()
```

1.10. Приклад програми застосування абстрактних класів

Оголосити абстрактний клас з віртуальною функцією Площа. Оголосити похідні класи – Трикутник, Прямокутник та Круг, у яких визначити функції обчислення площі:

- трикутника – $S=[p(p-a)(p-b)(p-c)]^{0.5}$, де $p=(a+b+c)/2$, a,b,c – сторони трикутника;
- прямокутника – $S=xy$, де x,y – сторони прямокутника;
- круга – $S=\pi r^2$, де r – радіус.

Реалізувати механізм пізнього зв'язування за допомогою масиву вказівників на абстрактний клас, елементам якого присвоєно адреси об'єктів похідних неабстрактних класів. Використати вказівники для виклику віртуального методу.

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
#include <iomanip>
using namespace std;

class Figure {
public:
  virtual float Area()=0;
};

class Triangle:public Figure {
  float a,b,c;
public:
  Triangle(float a1=0, float b1=0, float c1=0) {
    a=a1; b=b1;c=c1;
  }
  virtual float Area() {
    cout<<"Площа трикутника: ";
    float p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
  }
};

class Rectangle:public Figure {
  float x,y;
public:
  Rectangle(float x1=0, float y1=0) {x=x1;y=y1;}
  virtual float Area() {
    cout <<"Площа прямокутника: ";
    return x*y;
  }
};

class Circle:public Figure {
  float r;
public:
  Circle(float r1=0) {r=r1;}
  virtual float Area() {
    cout<<"Площа круга: ";
    return M_PI*r*r;
  }
};

float GetArea(Figure* f) { return f->Area();}

int main() {
  Figure* p[3]={
    new Triangle(5,8,7),
    new Rectangle(3,6),
    new Circle(4)
  };
  cout.setf(ios::fixed);
  cout.precision(2);
```

```
  for(int i=0; i<3; ++i)
  cout << GetArea(p[i]) << endl;
  return 0;
}
// Площа трикутника: 17.32
// Площа прямокутника: 18.00
// Площа круга: 50.27
```

Запитання.

1. Що таке поліморфізм і які є його типи.
2. Що таке раннє і пізнє звязування.
3. Який метод називається віртуальним і які його особливості.
4. Як виключити поліморфізм віртуальних методів.
5. Які віртуальні методи є коваріантними.
6. Що таке выртуальны операторні методи.
7. Як реалізувати вказівники на віртуальні методи
8. Динамічні віртуальні методи.
9. Особливості застосування віртуальних деструкторів.
10. Що таке чисті віртуальні методи і для чого вони використовуються.
11. Що таке абстрактний і чистий абстрактний класи.
12. Пояснити приклад програми застосування абстрактних класів

Завдання.

1. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – площа. Створити похідні класи: ПРЯМОКУТНИК, КОЛО, ПРЯМОКУТНИЙ ТРИКУТНИК, ТРАПЕЦІЯ зі своїми функціями площі. Для перевірки викликів віртуальних функцій визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів.

2. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – норма. Створити похідні класи: КОМПЛЕКСНІ ЧИСЛА, ВЕКТОР, МАТРИЦЯ. Визначити функцію норми: для комплексних чисел – модуль комплексного числа, для вектора – корінь квадратний із суми квадратів елементів, для матриці – максимальне значення за модулем. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

3. Створити абстрактний клас (КРИВІ) обчислення залежності y від x. Створити похідні класи: ПРЯМА, ЕЛІПС, ГІПЕРБОЛА зі своїми функціями обчислення $y(x)$ залежно від вхідного параметра x. Ріняння прямої: $y=ax+b$; еліпса: $x^2/a^2+y^2/b^2=1$, де a – велика піввісь, b – мала піввісь; гіперболи: $x^2/a^2-y^2/b^2=1$, де a – дійсна піввісь, b – уявна піввісь. Для перевірки визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

4. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – сума прогресії. Створити похідні класи: АРИФМЕТИЧНА ПРОГРЕСІЯ і ГЕОМЕТРИЧНА ПРОГРЕСІЯ. Кожен клас маєдва поля типу double. Перше – перший елемент прогресії, друге – постійна різниця для арифметичної і постійне відношення для геометричної прогресії. Визначити функцію обчислення суми, де параметром є кількість елементів прогресії. Арифметична прогресія $a_j=a_0+jd,\ j=0,1,2,\ldots$.Сума арифметичної прогресії $s_n=(n+1)(a_0+a_n)/2$. Геометрична прогресія: $a_j=a_0r^j,\ j=0,1,2,\ldots$. Сума геометричної прогресії: $s_n=(a_0-a_nr)/(1-r)$. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

5. Створити базовий клас – СПИСОК. Реалізувати на базі списку СТЕК і ЧЕРГУ з віртуальними функціями включення і вилучення елементів. Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуальних функцій.

6. Створити базовий клас – геометрична ФІГУРА, і похідні класи – КОЛО,

ПРЯМОКУТНИК, ТРАПЕЦІЯ. Визначити віртуальні функції визначення площі, периметра і виведення на екран. Площа круга $S=\pi r^2$ (r – радіус); площа прямокутника $S=ab$ (a, b – сторони); площа трапеції $S=(a+b)h/2$ (a, b – основа трапеції, h – висота). Довжина кола $L=2\pi r$; периметр прямокутника $L=2(a+b)$; периметр трапеції $L=a+b+c+d$ (a, b, c, d – сторони). Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуальних функцій.

7. Створити базовий клас – ПРАЦІВНИК і похідні класи – СЛУЖБОВЕЦЬ З ПОГОДИННОЮ ОПЛАТОЮ, СЛУЖБОВЕЦЬ З ОКЛАДОМ. Визначити віртуальну функцію нарахування зарплати. Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуальних функцій.

8. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – площа поверхні. Створити похідні класи: прямокутний ПАРАЛЕЛЕПІПЕД, ТЕТРАЕДР, КУЛЯ зі своїми функціями площі поверхні. Площа поверхні паралелепіпеда: $S=2(ab+bc+ca)$, де a, b, c – ребра. Площа поверхні кулі: $S=4\pi r^2$, де r – радіус. Площа поверхні тетраедра: $S=a^2s^{0.5}$, де a – довжина ребра. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

9. Створити абстрактний клас – ССАВЦІ. Визначити похідні класи – ТВАРИНИ і ЛЮДИ. У тварин визначити похідні класи КОНЕЙ і КОРІВ. Визначити віртуальні функції опису людини, коня і корови. Для перевірки пізнього зв'язування визначити масив вказівників на абстактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

10. Створити базовий клас – БАТЬКО, у якого є ім'я. Визначити віртуальну функцію виведення імені на екран. Створити похідний клас ДИТИНА, у якої є імя та успадковане поле по батькові. Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуального методу.

12. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – корені рівняння. Створити похідні класи: клас ЛІНІЙНИХ РІВНЯНЬ і клас КВАДРАТНИХ РІВНЯНЬ. Визначити функцію обчислення коренів рівнянь. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

13. Створити абстрактний клас для роботи з геометричними ФІГУРАМИ на екрані. У захищеній частині класу знаходяться такі дані: координати центра фігури; кут повороту (у градусах); масштабний фактор. У відкритій частині розміщено функції-методи: зобразити фігуру на екрані; зробити фігуру невидимою; повернути фігуру на заданий кут (задається у градусах); перемістити фігуру на заданий вектор. Застосовуючи успадкування та наведений вище абстрактний клас, створити похідні класи для роботи з фігурою: ТРИКУТНИК, ЧОТИРИКУТНИК, БАГАТОКУТНИК. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюється адреси об'єктів похідних класів. Використати вказівники для виклику віртуальної функції.

2. Самостійна робота

```
/* listing 1 — виклик віртуальної функції за допомогою вказівника на об'єкти */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з базового класу.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

class derived2 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived2.\n";
  }
};

int main()
{
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник на базовий клас
  p = &b;
  p->vfunc(); // виклик функції vfunc() з класу base()
  // вказівник на об'єкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції vfunc() з класу derived1

  // вказівник на об'єкт класу derived2
  p = &d2;
  p->vfunc(); // виклик функції vfunc() з класу derived2

  return 0;
}
Функція vfunc() з базового класу.
Функція vfunc() з класу derived1.
Функція vfunc() з класу derived2.

/* listing 3  - викликати віртуальну функцію можна за допомогою посилання на об'єкт базового класу  */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};
```

```
class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1\n";
  }
};

class derived2 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived2\n";
  }
};

// Використовується параметр, який є посиланням на об’єкт базового класу
void f(base &r) {
  r.vfunc();
}

int main()
{
  base b;
  derived1 d1;
  derived2 d2;

  f(b);  // функції f() передається об'єкт класу base
  f(d1); // функції f() передається об'єкт класу derived1
  f(d2); // функції f() передається об'єкт класу derived2

  return 0;
}
```

```
Функція vfunc() з класу base.
Функція vfunc() з класу derived1
Функція vfunc() з класу derived2
```

```
/* listing 4 – успадкування атрибуту virtual */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

/* Клас derived2 успадковує віртуальну функцію vfunc() від класу derived1. */
class derived2 : public derived1 {
public:
  // функція vfunc() залишається віртуальною
  void vfunc() {
    cout << "Функція vfunc() з класу derived2.\n";
  }
};

int main()
{
```

```
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник на обєкт класу  base
  p = &b;
  p->vfunc(); // виклик функції vfunc() з класу base

  // вказівник на обєкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції vfunc() з класу derived1

  // вказівник на клас  derived2
  p = &d2;
  p->vfunc(); // виклик функції vfunc() з класу derived2

  return 0;
}
```

Функція vfunc() з класу base.
Функція vfunc() з класу derived1.
Функція vfunc() з класу derived2.

/* listing 5 - віртуальні функції є ієрархічними. Віртуальну функцію не обов'язково заміщати. */

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

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

class derived2 : public base {
public:
// функція не заміщується в класі derived2, а використовується версія з класу base
};

int main()
{
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник тна об'єкт класу base
  p = &b;
  p->vfunc(); // виклик функції  vfunc() з класу base

  // вказівник на об'єкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції vfunc() з класу derived1

  // вказівник на об'єкт класу derived2
  p = &d2;
  p->vfunc(); // виклик функції vfunc() з класу base
```

```
  return 0;
}
```
Функція vfunc() з класу base.
Функція vfunc() з класу derived1.
Функція vfunc() з класу base.

```
/* listing 6 - віртуальні функції є ієрархічними. Якщо віртуальна функція не
заміщається то викликається її попередня версія  */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

class derived2 : public derived1 {
public:
/* Функція vfunc() не заміщається в класі derived2.
   Так як клас derived2 є наслідником класу derived1,
   то викликається функція vfunc() з класу derived2 */
};

int main()
{
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник на об'єкт класу base
  p = &b;
  p->vfunc(); // виклик функції з класу base

  // вказівник на об'єкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції з класу derived1

  // вказівник на об'єкт derived2
  p = &d2;
  p->vfunc(); // функція vfunc() з класу derived1

  return 0;
}
```
Функція vfunc() з класу base.
Функція vfunc() з класу derived1.
Функція vfunc() з класу derived1.

```
/* listing 7 — чисто віртуальна функція не має визначення у базовому класі */
#include <iostream>
using namespace std;

class number {
protected:
  int val;
public:
```

```
  void setval(int i) { val = i; }

  // функція show() є чисто віртуальною, признак  show() = 0;
  virtual void show() = 0;
};

class hextype : public number {
public:
  void show() {
    cout << hex << val << " ";
  }
};

class dectype : public number {
public:
  void show() {
    cout << val << " ";
  }
};

class octtype : public number {
public:
  void show() {
    cout << oct << val << "\n";
  }
};

int main()
{
  dectype d;
  hextype h;
  octtype o;

  d.setval(20);
  d.show(); // виводить десяткове число 20

  h.setval(20);
  h.show(); // виводить шістнадцяткове число 14

  o.setval(20);
  o.show();  // виводить вісімкове число 24

  return 0;
}
```

20 14 24

/* listing 8 - застосування віртуальних функцій у ієрархії класів. Клас, який містить хоча б одну чисто віртуальну функцію називається абстрактним. Так як віртуальні функції не мають визначення, то створити об'єкт абстрактного класу неможливо. Абстрактні класи можуть бути тільки основою для похідних класів. */

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

class convert {     // абстрактний клас
protected:
  double val1;  // початкові значення
  double val2;  // перетворене значення
public:
  convert(double i) {
    val1 = i;
  }
  double getconv() { return val2; }
  double getinit() { return val1; }
```

```
  virtual void compute() = 0;    // чисто віртуальна функція
};

// перетворення літрів в галони
class l_to_g : public convert {
public:
  l_to_g(double i) : convert(i) { }
  void compute() {
    val2 = val1 / 3.7854;
  }
};

// перетворення шкали Фарангейта в шкалу Цельсія
class f_to_c : public convert {
public:
  f_to_c(double i) : convert(i) { }
  void compute() {
    val2 = (val1-32) / 1.8;
  }
};

int main()
{
  convert *p;  // вказівник на базовий клас

  l_to_g lgob(4);
  f_to_c fcob(70);

  // застосування віртуальної функції для конвертації
  p = &lgob;
  cout << p->getinit() << " літри дорівнює ";
  p->compute();
  cout << p->getconv() << " галонів\n";  // l_to_g

  p = &fcob;
  cout << p->getinit() << " по Фарангейту дорівнює ";
  p->compute();
  cout << p->getconv() << " по Цельсію\n";  // f_to_c

  return 0;
}
```

4 літри дорівнює 1.05669 галонів
70 по Фарангейту дорівнює 21.1111 по Цельсію

Лабораторна робота № 9. Бібліотека стандартних шаблонів

Мета роботи: вивчення і практичне використання бібліотеки стандартних шаблонів.
Теоретичний матеріал лекції, [1, розділ 11], [2, розділи 33-35], [3, розділ 16].

1. Короткі теоретичні відомості

Бібліотека стандартних шаблонів (Standard Template Library, STL) є бібліотекою контейнерних класів C++. До складу бібліотеки входять такі елементи:

• *Контейнер* – об'єкт, який містить інші об'єкти, організовані у вигляді послідовностей (колекції об'єктів). Контейнери поділяють на базові (динамічний вектор, список, черга) та асоціативні (відображення, множини).

• *Ітератор* – забезпечує, доступ до вмісту контейнера. Над літераторами можна виконувати операції, призначені для роботи з вказівниками (інкремент, декремент, розадресування). Розрізняють такі види ітераторів: однонапрямлений, двонапрямлений, введення, виведення, довільного доступу, зворотний.

• *Алгоритм* – це параметризовані зовнішні функція, що забезпечують обчислювальні процедури над елементами контейнерів, наприклад ініціалізація, пошук, сортування, заміна елементів.

• *Алокатор* – забезпечує керування динамічною пам'яттю, виділеною для STL-об'єктів.

• *Функтор* – об'єкт з перевантаженою операцією operator(). Використовується замість вказівника на функцію за потреби передавання функції в іншу функцію через список параметрів.

• *Предикат* – функція булевого типу, яка перевіряє визначені програмістом властивості або відношення між об'єктами. Предикат може бути унарним або бінарним. Типи параметрів предикату відповідають типам об'єктів, інкапсульованих контейнером.

• *Адаптор* – пристосовує елемент бібліотеки для забезпечення різних інтерфейсів. Наприклад, на основі вектора, списку або черги можна створити стек, а на основі списку – чергу.

Для використання алгоритмів та функторів у програмі необхідно під'єднати заголовкові файли:

```
#include <algorithm>
#include <functional>
```

Таблиця 1. Класи бібліотек стандартних шаблонів

Класи	**Файли включення**	**Призначення**
vector	<vector>	динамічний вектор
valarray	<valarray>	масив для математичних обчислень
complex	<complex>	комплексне число
list	<list>	лінійний список
stack	<stack>	стек
queue	<queue>	черга
priority_queue	<queue>	пріоритетна черга
deque	<deque>	черга з двома кінцями
map	<map>	відображення (словник) без повторень елементів
multimap	<map>	відображення з повтореннями елементів
set	<set>	множина без повторень елементів
multiset	<set>	множина з повтореннями елементів
basic_string	<string>	базовий клас для роботи з рядками
string	<string>	спеціалізація basic_string<char>

1.1. Ітератори шаблонних класів

Ітератор використовуються для звернення до елементів контейнера. Оголошуються за допомогою слова `iterator`. Більшість класів STL мають методи `begin()` та `end()`, які повертають значення ітератора відповідно на початок та кінець послідовності елементів контейнера. Ітератори мають властивості вказівників. Для них визначені операції розадресації (*), інкременту (++), декременту(--), цілочисельне збільшення значення (+), різниці ітераторів (-) та операцій порівняння (!=, ==, <, <=,>,>=).

Приклад виведення усіх елементів контейнера за допомогою ітератора і циклу `for`:

```
vector<int> C;
 vector <int>::iterator p;
 for(p=C.begin();p!=C.end;p++) cout<<" "<<*p; cout<<endl;
```

1.2. Шаблонний клас string

Шаблонний клас (контейнер) string призначений для роботи з рядками символів. У контейнері визначено перевантажені конструктори для оголошення об'єкту без ініціалізації, з ініціалізацією об'єкту значенням рядка або іншого об'єкту класу string, наприклад:

```
string s, str("ABCD"), t(str);
```

Створений об'єкт міститиме рядок, розмір якого може динамічно змінюватися. Звертатися до символів рядка можна за допомогою ітератора або індексу. Методи для роботи з символьними рядками:

`str.insert(ofs,s)` – вставити у позицію `ofs` значення `s`, яке може бути об'єктом `string` або рядком символів типу `char *`.

`str.erase(ofs,n)` – вилучити `n` символів з позиції `ofs`.

`str.substr(ofs,s)` – виділити (скопіювати) `n` символів з позиції `ofs`.

`str.copy(x,ofs,n)` – скопіювати `n` символів з позиції `ofs` у рядок `x`.

`str.c_str()` – перетворити рядок символів типу `string` у рядок типу `char`, який закінчується нуль-символом (у стилі С).

`str.find(s,ofs)` – знайти позицію підрядка `s` починаючи із зміщення `ofs`.

`str.push_back(c)` – помістити символ `c` у кінець рядка.

`str.append(s)` – зчепити рядки символів `str` і `s`.

`str1.compare(ofs,n,str2)` – порівняння `n` символів із зміщення `ofs` рядка `str1` з рядком `str2`.

`str.clear()` – витирання усіх символів рядка `str`.

`str.erase(ofs,n)` – витирання `n` символів із зміщенням `ofs` в рядку `str`.

`str.length()`, `str.size()` – визначення довжини рядка `str`.

`str.resize(n, ch)` – змінити довжину рядка, `ch` – символ заповнювач у випадку розширення рядка.

1.3. Шаблонний клас вектор

Шаблонний клас вектор забезпечує роботу з динамічними масивами різних типів. Оголошення об'єкта-вектор:

```
vector<int> v, a(5), b(5,0), c(b);
```

Методи для роботи з об'єктом-вектор:

`push_back()` – розширення базового вектора шляхом занесення елементів у його кінець.

`resize(n, x)` – зміна кількості елементів вектора, `n` – нова кількість елементів, `x` – символ заповнювач.

`size()` - поточна кількість елементів вектора.

`begin()`, `end()` – повертають ітератор на початок, кінець вектора.

`insert(iter, x)` – занесення значення `x` у позицію, яка визначається літератором `iter`.

`erase(iter), erase(iter_from,iter_to` – вилучення елементів з позиції, яка визначається ітератором.

`clear()` – вилучення усіх елементів вектора.

`front(), back()` – повернення посилання на початковий, кінцевий елемент.

Виводити елементи вектора (та інших колекцій) можна поелементно у циклі або застосувати алгоритм `copy`:

```
copy(v.begin(), v.end(), ostream_iterator<float>(cout," "));
```

Фрагмент програми роботи з вектором:

```
vector <float> v;
vector <float>::iterator p,g;
// вектор з випадкових чисел
for(int i=0;i<n;i++) {
    x=(float) rand()/RAND_MAX
    v.push_back(x);
}
// сортування вектора
for(p=v.begin();p!=v.end-1;p++)
   for(q=p+1;q!=v.end();q++)
         if(*p>*q) {x=*p; *p=*q;}
// виведення вектора
for(p=v.begin();p!=v.end;p++) cout<<" "<<*p;
```

На основі вектора можна побудувати різноманітні адаптори, наприклад двовимірні та багатовимірні масиви.

Наприклад, можна організувати матрицю, оголосивши її як вектор, елементами якого є теж вектори:

```
vector <float> v;
vector<float>::iterator p;
float x; int n=3,m=4;
vector <vector<float>>::iterator r;
vector<vector<float>> matrix;
for(int i-0;i<n;i++) {
   for(int j=0;i<m;j++) {
      x=(float) rand/RAND_MAX;
      v.push_back(v); // допоміжний вектор для занесення у матрицю
   }
      matrix.push_back(v);
      v.clear();
}
for(r=matrix.begin();r!=matrix.end();r++) {
   for(p=r->begin();p!=r->end();p++ cout<<" "<<*p;
```

1.4. Алгоритми

Алгоритми використовуються для роботи з елементами контейнерів різних типів. Для використання алгоритмів необхідно включити у текст програми заголовковий файл `#include <algorithm>`.

Алгоритми:

`min_element()` – пошук мінімального елементу, `imin=min_element(v.begin(),v.end());`

`max_element()` – пошук максимального елементу, `imax=max_element(v.begin(),v.end());`

`find()` – пошук заданого елементу, `i=find(v.begin(),v.end(),*imin);`

`remove()` – вилучення елементу, `i=remove(v.begin(),v.end(),*imax);`

`sort()` – сортування вектора, `sort(v.begin(),v.end(),greater(<int>());`

`copy()` – копіювання елементів вектора у інший вектор,
`copy(v.begin(),v.end(),t.begin());`

Бібліотека STL містить і інші корисні алгоритми для роботи з колекціями.

1.5. Масиви значень

Клас valarray призначений для роботи з масивами числових значень. Для роботи з класом valarray необхідно під'єднати заголовковий файл: #include <valarray>.

При створенні масиву вказується кількість елементів та значення для їх ініціалізації, наприклад:

```
int a[5]={-3, 5, 2, -4, 8};
valarrary<int> x(10), y(7,-1), (z(a,5);
```

Визначені у класі valarray методи та операції виконуються над кожним елементом масиву.

Методи призначені для обчислення математичних функцій: abs(), cos() , sin(), tan(), acos() , asin() , atan() , cosh(), sinh(), tanh(), sqrt(), exp(), log(), log10().

У класі valarray перевантажено ряд операцій, які виконуються над кожним елементом масиву або над парами елементів двох масивів: арифметичні (+,-,*,/,%), відношення(==,!=,>,>=,<,<=), порозрядні логічні (&,|,^), зсуву (>>,<<), логічні (&&,||), виділення елемента ([]).

Операції застосовуються до цілісних масивів. Бінарні операції виконуються над масивами з однаковою кількістю елементів. Результат виконання операції повертається як масив значень такої самої розмірності. Операції Відношення повертають масив значень логічного типу, кожен елемент якого є результатом відношення між відповідними парами елементів. Перевантажена операція [] забезпечує звернення до елементів масиву за допомогою індексу.

Приклад обчислення виразу y=(a*b-c)/(a+b+c) над масивами дійсних чисел a,b,c:

```
valarray<double> a(-2,5),b(4,5),c(-7,5),y(5);
y=(a*b-c)/(a+b+c);
for(int i=0;i<y.size();i++) cout<<y[i]<<" ";
```

Інші методи класу valarray:

```
size() – кількість елементів масиву.
resize(n), resize(n,x) – зміна розміру масиву.
min(), max() – мінімальний, максимальний елемент масиву.
sum() – сума елементів масиву.
shift(ofs) – зсув на ofs позицій елементів масиву(ofs>0 – вправо,ofs<0 – вліво).
slice(beg, coutn, end) – виділення підмножини елементів (зрізи).
```

1.6. Стеки

Стек – це облвсть пам'яті з дисципліною доступу LIFO (Last Input – First Output, останній записаний елемент буде прочитаний першим). Стек реалізовано на основі контейнеру дек. За необхідності можна визначити інший базовий контейнер стеку:

Створення стеку:

```
stack<int> s;
stack <int, vector<int> > s2;
stack <int, list<int> > i3;
```

Методи стеку:

push() – занести елемент у стек.

pop() – вилучити елемент зі стеку.

top() – читання елементу з верхівки стеку.

size() – кількість елементів стеку.

empty() – перевірка наявності елементів у стеку.

Приклад роботи із стеком:

```
stack <int> s;
stack <int>::size_type size;
int i=10;
s.push(i);
Ccut<<s.top()<<" ";
```

```
s.pop();
```

1.7. Списки

Шаблонний клас `list` реалізує двонаправлений список елементів. Створення списків:

```
list<int> s, s1(), s2(7, 0), s3(s2);
```

Методи списку:

`push_front()` – занесення елементів у початок списку.
`push_end()` – занесення елементів у кінець списку
`pop_front()` – читання списку з початку.
`pop_back()` – читання списку з кінця.
`begin(), end()` – повертають ітератор на початковий, кінцевий елемент.
`insert(iter,x)` – вставлення елемента у позицію визначену значенням ітератора.
`erase(iter)` – вилучення елемента за значенням ітератора.
`remove(x)` – вилучення елемента `x`.
`clear()` – вилучення всіх елементів списку.
`s1.merge(s2)` – злиття списків `s1, s2`.
`sort()` – сортування списку.
`resize(n),` – зміна розміру списку
`resize(n,x)` – збільшення розміру списку із заповнення розширення символом `x`.

Приклад створення списку і виведення елементів списку:

```
list<int> s;
list<int>::iterator p;
for(int i=0;i<n;i++) s.push_back(i);
for(p=s.begin();p!=s.end();p++) cout << *p <<" ";
```

1.8. Черги і деки

Черги – це організована послідовність елементів з дисципліною доступу FIFO (First Input – First Output). Елементи добавляються в чергу з її кінця, а вилучаються – з початку. За замовчуванням черга будується на основі базового контейнера `deque`. За необхідності можна змінити базовий контейнер черги, наприклад на список:

```
queue<int> q;
queue<char, list<char> > q;
```

Методи черги:

`push(x)` – занесення елемента `x` в кінець черги.
`pop()` – вилучення елемента з початку черги.
`back()` – повернення значення останнього елемента черги.
`front()` – повернення значення першого елемента черги.
`empty()` – перевірка наявності елементів у черзі.
`size()` – поточне число елементів черги.

Приклад створення і роботи з чергою:

```
queue <int> q;
queue <int>::size_type size
q.push(5);
cout<<q.size();
q.pop();
```

Дек – це черга, в якій елементи добавляються і вилучаються з обох кінців. Дек можна порівняти з динамічним вектором, який може зростати і скоручуватися з обох кінців. Оголошення об’єкта шаблонного класу `deque`:

```
deque <int> d, d1(5), d2(5,-1), d3(d2);
```

Методи класу `deque` забезпечують двосторонній доступ до деку.

`push_front(x)` – занесення елемента `x` на початок деку.
`push_back(x)` – занесення елемента `x` у кінець деку.
`pop_front()` - читання елемента з початку деку.
`pop_back()` – читання елемента з кінця деку.
`begin(), end()` – повертають ітератор на початковий, кінцевий елемент деку.
`front(), back()` – читання елементів з початку, кінця деку.
`insert(iter,x)` – добавлення елемента `x` у визначену значенням ітератора позицію.
`erase(iter)` – вилучення елемента за значенням ітератора.
`clear()` –вилучення всіх елементів деку.
`size()` – поточна кількість елементів деку.
`resize(n)` – зміна розміра деку.
`resize(n,x)` – збільшення розміра деку із заповненням розширених елементів значенням `x`.
`empty()` – перевірка наявності елементів у деку.

Приклад створення і роботи з деком:

```
deque <int> d;
dequeue <int>::size_type size
d.push_front(5);
d.push_back(6);
cout<<d.size();
d.pop_back();
d.pop_front();
d.empty();
```

1.9. Асоціативні відображення

Асоціативні контейнери містять елементи, які складаються з двох частин: ключа та елемента даних. Ключ та елемент даних перебувають в асоціативній залежності. За значенням ключа можна відшукати відповідний елемент даних.

Об’єкти відображення (словники) елементів створюються за допомогою шаблонного класу `map`:

```
map<string, int> m;
```

Колекції асоціативних контейнерів зберігаються у пам’яті як бінарне дерево. Конструктори усіх асоціативних контейнерів містять необовязковий параметр, який задає критерій сортування колекції його елементів:

```
map<string, int, less<string> > m1;
map<string, int, greater<string> > m2;
```

Кожний елемент відображення складається з пари значень: ключа та відповідного йому поля даних. Для доступу до пар значень використовується об’єкт шаблонного класу `pair`:

```
pair<string, float> x, *p=new pair<string, float>;
```

Поля пари значень мають назви `first` та `second`. Звертатися до них можна використовуючи назву об’єкта або вказівник: `x.first, x.second, p->first, p->second`.

Пару елементів можна створити за допомогою конструктора класу `pair`:

```
x=pair<string, float> (”Вага”, 15.5);
```

або за допомогою функції `make_pair`:

```
x=make_pair(”Об’єм”, 120.5);
```

Методи відображень:

`begin(), end()` – методи повертають ітератор на початок, кінець колекції елементів
`insert(iter, x)` – вставлення елемента `x` за значенням ітератора
`find(key)` – пошук елемента колекції за ключем.
`lower_bound(key)` – повертає ітератор на перший елемент, який є менший або дорівнює заданому ключу.
`upper_bound(key)` – повертає ітератор на перший елемент, який є більшим за заданий ключ.

`count(key)` – повертає кількість елементів, ключі яких дорівнюють ключу, заданому у списку аргументів.

`erase(iter)` – вилучення елемента за заданим ітератором.

`erase(iter1, iter2)` – вилучення послідовності елементів заданих двома ітераторами.

`clear()` – вилучення усіх елементів.

`empty()` – нявність елементів у колекції.

Приклад створення і роботи з відображенням:

```
map<string,int> m;
map<string,int>::const_iterator p;
typedef pair<string,int> Student;
m.isert(Student(”Курс”, 2));
m.insert(Student(”Курс”,3));
for(p=m.begin();p!=m.end();p++) cout<<p-first<<” ”<<p->second<<endl;
```

1.10. Множини

Множини є різновидністю відображення, у якому значення елемента дорівнює його ключу. Множина задається контейнером шаблонного класу `set`:

```
set<int> s1;
set<int, less<int> > s2;
set<int, greater<int> > s3;
```

У памяті множини зберігаються як бінарне дерево. Для роботи з множинами використовують ті самі методи та операції, що і для класу `map`.

Приклад створення і роботи з множиною:

```
set <int> s1;
set <int>::iterator p,q;
for(int i=0;i<n;i++) s1.insert(i);
for(p=s1.begin();p!=s1.ens();p++) cout<<*p<<” ”;
for each (x in s1) cout <<x<<” ”;
```

1.11. Множини бітів

Множина бітів є об’єктом шаблонного класу `bitset`, але не має ітератора і не вважається контейнером STL-бібліотеки. При оголошенні об’єкта класу `bitset` аргумент шаблону задається константою цілого типу, яка визначає кількість бітів множини:

```
#include <bitset>
bitset<> b0;
bitset<8> b1(5);
string str(”1110011110001110”);
bitset<16> b2(str);
```

Методи для роботи множиною бітів:

`test(n)` – перевірка значення біта у позиції `n`.

`any()` – повертає `true`, якщо хоча б один біт множини встановлено в 1.

`none()` – повертає `true`, якщо у множині немає бітів зі значенням 1.

`set(n)` – встановлює біт з номером `n`.

`set()` – встановлює всі біти множини в 1.

`reset()` – встановлює всі біти множини в 0.

`flip(n)` – інвертує біт з номером `n`.

`flip()` – інвертує всі біти.

`count()` – повертає число одиниць у бітовій множині.

`size()` – повертає загальне число бітів у множині.

`to_string()` – перетворює бітову множину в об’єкт класу `string`.

`to_ulong()` – перетворює бітову множину в ціле число.

Крім методів у класі `bitset` перевантажено операції для роботи з бітовими множинами: порівняння(`==, !=`), інверсії бітів (`~`), комбіновані порозрядні (`&=, |=, ^=`), зсуву (`<<, <<=, >>, >>=`), потокового введення та виведення (`>>,<<`), квадратні дужки. Операція `[]` повертає значення біта у заданій позиції.

Приклад створення і роботи з множиною бітів:

```
#include <bitset>
bitset<8> b1(45),b2(9),b3; cout<<b1<<" "<<b2<<endl;
b3=b1 | b2; cout<<b3<<endl;
b3=b1 & (b1^b2); cout<<b3<<endl;
```

Приклад програми. Розробити телефонну книгу на основі класу Особа (`Person`) та контейнерного класу Множина з повторенням елементів (`multiset`). Ввести прізвище Особи та визначити номер її телефону. Визначити кількість осіб, що мають однакове прізвище та ім'я, і вивести на екран номери їхніх телефонів.

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

// клас особа
class Person {
  string LastName;   // Прізвище
  string FirstName;  // Ім'я
  long PhoneNumber;  // Номер телефону
public:

// конструктор
Person(string LN="", string FN="", long PN=0) :
  LastName(LN), FirstName(FN), PhoneNumber(PN) {}

// перевантажена операція new
void * operator new(size_t, void *ptr) {return ptr;}

// перевантажена операція ==
friend bool operator==(const Person& P1, const Person& P2)
   {return (P1.LastName==P2.LastName) && (P1.FirstName==P2.FirstName)?true:false;}

// перевантажена операція <
friend bool operator<(const Person& P1, const Person& P2)
   {if(P1.LastName==P2.LastName) return (P1.FirstName<P2.FirstName)?true:false;
    return (P1.LastName<P2.LastName)?true:false;}

//перевантажена операція потокового введення
friend istream& operator>>(istream&, Person&);

// перевантажена операція потокового виведення
friend ostream& operator<<(ostream&, Person&);

};

istream& operator>>(istream& is, Person& P)
  {is>>P.LastName>>P.FirstName>>P.PhoneNumber; return is;}

ostream& operator<<(ostream& os, Person& P)
   //{os<<P.LastName<<'\t'<<P.FirstName<<'\t'<<P.PhoneNumber<<endl;return os;}
   {os<<P.LastName<<endl;return os;}

// Головна функція
int main() {
// Множина з повторенням елементів
multiset<Person> S;
```

```
multiset<Person, less<Person> >::iterator i;

// об'єкт класу Person;
Person P;

cout<<"Введіть дані про власників телефонів"<<endl;
while(cin>>P) S.insert(P);   // введення до натиснення Ctrl+Z
cin.clear();                 //очищення потоку введення від ознаки EOF (Ctrl+Z)

cout<<"Число записів="<<S.size()<<endl;

cout<<"Склад множини"<<endl;

//for(i=S.begin(); i!=S.end(); i++) {  cout<<reinterpret_cast<char>(*i)<<endl;}
i=S.begin(); cout <<*i<<endl;


cout<<"Пошук номера телефону"<<endl;
string LN, FN;
cout<<"Введіть прізвище особи"<<endl;
cin>>LN;
cout<<"Введіть ім'я"<<endl;
cin>>FN;

// повторна ініціалізація об'єкта
new(&P) Person(LN,FN,0);
cout<<"Кількість осіб з таким прізвищем та іменем"<<endl;
cout<<S.count(P)<<endl;

cout<<"Всі записи, що відповідають запиту"<<endl;
//for(i=S.lower_bound(P);i!=S.upper_bound(P);i++) cout<<*i<<endl;

return 0;
}
```

Запитання.

1. Які елементи входять до складу бібліотеки стандартних шаблонів?
2. Дати характеристику об’єктам контейнер, ітератор, алгоритм. Типи ітераторів.
3. Дати характеристику об’єктам алокатор, функтор, предикат, адаптер.
4. Шаблонний клас `string` і його методи для роботи з рядками символів.
5. В чому перевага і недоліки використання об’єктів типу `string`.
6. Шаблонний клас `vector` і його методи для роботи з динамічними векторами.
7. Призначення алгоритмів для роботи з колекціями.
8. Масив значень і його методи.
9. Стек і його методи.
10. Список і його методи.
11. Черги і деки та їх методи.
12. Асоціативні відображення.
13. Множини і множини бітів.

Завдання.

1. Написати шаблонну функцію створення, заповнення, сортування і виведення значень динамічного вектора.

2. Написати шаблонну функцію, яка читає текстовий файл і вставляє елементи файла у список.

3. Написати програму яка записує пари номер місяця – назва місяця в асоціативний масив і потім за введеним номером друкує назву місяця.

4. Написати шаблонну функцію послідовного пошуку значення в масиві за заданим

ключем. Функція повертає перший елемент масиву, який відповідає заданому ключу, або виводить повідомлення, що елемент не знайдено.

5. Написати шаблонну функцію сортування одновимірного масиву за зростанням елементів методом бульбашки. Суть методу полягає в порівнянні та перествлянні сусідніх елементів.

6. Написати функцію шаблон, яка перевіряє впорядкованість елементів масиву за зростанням або спаданням.

7. Написати шаблонну функцію формування файлу числових даних у текстовому форматі.

8. Створити шаблонний клас КВАДРАТНА МАТРИЦЯ. Обчислити контрольну суму елементів матриці як суму усіх елементів за модулем 2.

9. Створити шаблонний клас СТЕК. Визначити методи занесення рядків у стек та їх читання зі стеку. Параметризувати стек рядками символів. Знайти найдовший занесений у стек рядок.

10. Створити шаблонний клас для роботи з однонапрямленим лінійним списком. Впорядкувати список за зростанням значень елементів.

11. Створити шаблонний клас для роботи з лінійним списком з подвійними зв’язками. Занести елемент на початок списку. Вилучити елемент з кінця списку.

12. Створити шаблонний клас МНОЖИНА, призначений для збереження елементів і виконання операцій над множинами. Реалізувати класичні операції над множинами: об’єднання, перетину, різниці.

13. Створити шаблонний клас ЧЕРГА З ПРІОРИТЕТАМИ. Занести елемент у чергу. Вилучити з черги елементи з найвищим та з найнижчим пріоритетами.

2. Приклади для самостійної роботи

```
//listing 2 - основні операції над векторами
#include <iostream>
#include <vector>
#include <cctype>
using namespace std;

int main()
{
  vector<char> v(10); // створення вектора з 10 елементів
  unsigned int i;

  // вивід на екран розміру вектора v
  cout << "Розмір = " << v.size() << endl;

  // присвоєння елементам вектора деякі значення
  for(i=0; i<10; i++) v[i] = i + 'a';

  // вивід на екран вмісту вектора
  cout << "Поточний вміст:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  cout << "Розширений вектор\n";
  // добавлення нових елементів в кінець вектора
  for(i=0; i<10; i++) v.push_back(i + 10 + 'a');

  // виводимо на екран розмір вектора v
  cout << "Новий розмір = " << v.size() << endl;

  // виводимо на екран вміст вектора
  cout << "Поточний вміст:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";
```

```
  // змінюємо вміст вектора
  for(i=0; i<v.size(); i++) v[i] = toupper(v[i]);
  cout << "Модифікований вміст:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  return 0;
}
```

Розмір = 10
Поточний вміст:
a b c d e f g h i j
Розширений вектор
Новий розмір = 20
Поточний вміст:
a b c d e f g h i j k l m n o p q r s t
Модифікований вміст:
A B C D E F G H I J K L M N O P Q R S T

```
// listing 3 - доступ до елементів вектора за допомогою ітератора
#include <iostream>
#include <vector>
#include <cctype>
using namespace std;

int main()
{
  vector<char> v(10); // створення вектора довжини 10
  vector<char>::iterator p; // створення ітератора
  int i;

  // присвоєння елементам вектора значення
  p = v.begin();
  i = 0;
  while(p != v.end()) {
    *p = i + 'a';
    p++;
    i++;
  }

  // вивід на екран вмісту вектора
  cout << "Початковий вміст:\n";
  p = v.begin();
  while(p != v.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  // зміна вмісту вектора
  p = v.begin();
  while(p != v.end()) {
    *p = toupper(*p);
    p++;
  }

  // вивід на екран вмісту вектора
  cout << "Модифікований вміст:\n";
  p = v.begin();
  while(p != v.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;
```

```
  return 0;
}
```
Початковий вміст:
a b c d e f g h i j
Модифікований вміст:
A B C D E F G H I J

//listing 4 - функції вставки і вилучення
```
#include <iostream>
#include <vector>
using namespace std;

int main()
{
  vector<char> v(10);
  vector<char> v2;
  char str[] = "<Vector>";
  unsigned int i;

  // ініціалізуємо вектор v
  for(i=0; i<10; i++) v[i] = i + 'a';

  // копіювання символів із str у v2
  for(i=0; str[i]; i++) v2.push_back(str[i]);

  // вивід на екран вмісту вектора
  cout << "Початковий вміст v:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  vector<char>::iterator p = v.begin();  // ітератор
  p += 2; // встановлення ітератора на 3-й елемент

  // вставка  10-ти символів X у v
  v.insert(p, 10, 'X');

  // вивід на екран вмісту після вставки
  cout << "Розмір після вставки = " << v.size() << endl;
  cout << "Вміст  після вставки:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  // вилучення цих елементів
  p = v.begin();
  p += 2; // встановлення ітератора на 3-й елемент
  v.erase(p, p+10); // вилучення наступних 10 елементів

  // вивід вмісту після вилучення
  cout << "Розмір після вилучення = " << v.size() << endl;
  cout << "Вміст  після вилучення:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << "\n\n";

  // Вставка вектора  v2 у вектор v
  v.insert(p, v2.begin(), v2.end());
  cout << "Розмір після вставки вектора v2 = ";
  cout << v.size() << endl;
  cout << "Вміст після вставка вектора v2:\n";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  return 0;
}
```

```
Початковий вміст v:
a b c d e f g h i j
Розмір після вставки = 20
Вміст  після вставки:
a b X X X X X X X X X c d e f g h i j
Розмір після вилучення = 10
Вміст  після вилучення:
a b c d e f g h i j
Розмір після вставки вектора v2 = 18
Вміст після вставка вектора v2:
a b < V e c t o r > c d e f g h i j
```

```
//listing 5 - вектор, який містить об'єкти класу
#include <iostream>
#include <vector>
#include <cstdlib>
using namespace std;

class DailyTemp {
  int temp;
public:
  DailyTemp() { temp = 0; }
  DailyTemp(int x) { temp = x; }

  DailyTemp &operator=(int x) {
    temp = x; return *this;
  }

  double get_temp() { return temp; }
};

bool operator<(DailyTemp a, DailyTemp b)
{
  return a.get_temp() < b.get_temp();
}

bool operator==(DailyTemp a, DailyTemp b)
{
  return a.get_temp() == b.get_temp();
}

int main()
{
  vector<DailyTemp> v;
  unsigned int i;

  for(i=0; i<7; i++)
    v.push_back(DailyTemp(60 + rand()%30));

  cout << "Температура по Фарангейту:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i].get_temp() << " ";

  cout << endl;

  // Перетворення із шкали Фарангейта у шкалу Цельсія
  for(i=0; i<v.size(); i++)
    v[i] = (int) (v[i].get_temp()-32) * 5/9 ;

  // Перетворення із шкали Фарангейта у шкалу Цельсія
  for(i=0; i<v.size(); i++)
    v[i] = (int) (v[i].get_temp()-32) * 5/9 ;
```

```
  cout << "Температура в градусах цельсія:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i].get_temp() << " ";

  return 0;
}
```

Температура по Фарангейту:
73 76 87 85 83 85 76
Температура в градусах Цельсія:
22 24 30 29 28 29 24

```
//listing 6 - операції над списком
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst; // створення порожнього списку
  int i;

  for(i=0; i<10; i++) lst.push_back(i);

  cout << "Розмір = " << lst.size() << endl;

  cout << "Вміст: ";
  list<int>::iterator p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  // зміна вмісту списку
  p = lst.begin();
  while(p != lst.end()) {
    *p = *p + 100;
    p++;
  }

  cout << "Модифікований вміст: ";
  p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Розмір = 10
Вміст: 0 1 2 3 4 5 6 7 8 9
Модифікований вміст: 100 101 102 103 104 105 106 107 108 109

```
// Listing8 - Функція end().
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst; // створення порожнього списку
  int i;
```

```
  for(i=0; i<10; i++) lst.push_back(i);

  cout << "Вивід списку в прямому порядку:\n";
  list<int>::iterator p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  cout << "Вивід списку в зворотньому порядку:\n";
  p = lst.end();
  while(p != lst.begin()) {
   p--; // зменшення вказівника перед cout, так як останній елемент відповідає
значенню end()-1
    cout << *p << " ";
  }

  return 0;
}
```

Вивід списку в прямому порядку:
0 1 2 3 4 5 6 7 8 9
Вивід списку в зворотньому порядку:
9 8 7 6 5 4 3 2 1 0

//listing 9 - функції push_back() і push_front().

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

int main()
{
  list<int> lst1, lst2;
  int i;

  for(i=0; i<10; i++) lst1.push_back(i);
  for(i=0; i<10; i++) lst2.push_front(i);

  list<int>::iterator p;

  cout << "Вміст списку lst1:\n";
  p = lst1.begin();
  while(p != lst1.end()) {
    cout << *p << " ";
    p++;
  }
  cout << "\n\n";

  cout << "Вміст списку lst2:\n";
  p = lst2.begin();
  while(p != lst2.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Вміст списку lst1:
0 1 2 3 4 5 6 7 8 9
Вміст списку lst2:
9 8 7 6 5 4 3 2 1 0

```
//listing 10 - сортування списку
#include <iostream>
#include <list>
#include <cstdlib>
using namespace std;

int main()
{
  list<int> lst;
  int i;

  // створення списку з випадкових цілих чисел
  for(i=0; i<10; i++)
    lst.push_back(rand()%100);

  cout << "Початковий вміст:\n";
  list<int>::iterator p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl << endl;

  // sort the list
  lst.sort();

  cout << "Відсортований вміст:\n";
  p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст:
83 86 77 15 93 35 86 92 49 21
Відсортований вміст:
15 21 35 49 77 83 86 86 92 93

```
//listing 11 - злиття двох списків
#include <iostream>
#include <list>
using namespace std;

int main()
{
  list<int> lst1, lst2;
  int i;

  for(i=0; i<10; i+=2) lst1.push_back(i);
  for(i=1; i<11; i+=2) lst2.push_back(i);

  cout << "Вміст списку lst1:\n";
  list<int>::iterator p = lst1.begin();
  while(p != lst1.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl << endl;

  cout << "Вміст списку lst2:\n";
  p = lst2.begin();
  while(p != lst2.end()) {
```

```
    cout << *p << " ";
    p++;
  }
  cout << endl << endl;

  // злиття двох файлів
  lst1.merge(lst2);
  if(lst2.empty())
    cout << "lst2 тепер порожній\n";

  cout << "Вміст списку lst1 після злиття:\n";
  p = lst1.begin();
  while(p != lst1.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

```
Вміст списку lst1:
0 2 4 6 8

Вміст списку lst2:
1 3 5 7 9

lst2 тепер порожній
Вміст списку lst1 після злиття:
0 1 2 3 4 5 6 7 8 9
```

```
//listing 12 - список з об'єктами
#include <iostream>
#include <list>
#include <cstring>
using namespace std;

class myclass {
  int a, b;
  int sum;
public:
  myclass() { a = b = 0; }
  myclass(int i, int j) {
    a = i;
    b = j;
    sum = a + b;
  }
  int getsum() { return sum; }

  friend bool operator<(const myclass &o1,
                        const myclass &o2);
  friend bool operator>(const myclass &o1,
                        const myclass &o2);
  friend bool operator==(const myclass &o1,
                        const myclass &o2);
  friend bool operator!=(const myclass &o1,
                        const myclass &o2);
};

bool operator<(const myclass &o1, const myclass &o2)
{
  return o1.sum < o2.sum;
}

bool operator>(const myclass &o1, const myclass &o2)
{
```

```
  return o1.sum > o2.sum;
}

bool operator==(const myclass &o1, const myclass &o2)
{
  return o1.sum == o2.sum;
}

bool operator!=(const myclass &o1, const myclass &o2)
{
  return o1.sum != o2.sum;
}
int main()
{
  int i;

  // створення першого списку
  list<myclass> lst1;
  for(i=0; i<10; i++) lst1.push_back(myclass(i, i));

  cout << "Перший список: ";
  list<myclass>::iterator p = lst1.begin();
  while(p != lst1.end()) {
    cout << p->getsum() << " ";
    p++;
  }
  cout << endl;

  // створення другого списку
  list<myclass> lst2;
  for(i=0; i<10; i++) lst2.push_back(myclass(i*2, i*3));

  cout << "Другий список: ";
  p = lst2.begin();
  while(p != lst2.end()) {
    cout << p->getsum() << " ";
    p++;
  }
  cout << endl;

  // злиття двох списків
  lst1.merge(lst2);
  // вивід на екран списку після злиття
  cout << "Список після злиття: ";
  p = lst1.begin();
  while(p != lst1.end()) {
    cout << p->getsum() << " ";
    p++;
  }
  return 0;
}
```

Перший список: 0 2 4 6 8 10 12 14 16 18
Другий список: 0 5 10 15 20 25 30 35 40 45
Список після злиття: 0 0 2 4 5 6 8 10 10 12 14 15 16 18 20 25 30 35 40 45

//listing 13 - асоціативний масив

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

int main()
{
  map<char, int> m;
  int i;
```

```
  // ввід пари в асоціативний масив
  for(i=0; i<26; i++) {
    m.insert(pair<char, int>('A'+i, 65+i));
  }

  char ch;
  cout << "Введіть ключ (велика баква): ";
  cin >> ch;

  map<char, int>::iterator p;

  // знаходження значення по заданому ключу
  p = m.find(ch);
  if(p != m.end())
    cout << "ASCII-код ключа дорівнює  " << p->second;
  else
    cout << "Ключ не знайдений.\n";

  return 0;
}
```

Введіть ключ (велика баква): F
ASCII-код ключа дорівнює 70

```
//listing 15 - використання асоціативного масиву для створення телефонної книжки
#include <iostream>
#include <map>
#include <cstring>
using namespace std;

class name {
  char str[40];
public:
  name() { strcpy(str, ""); }
  name(char *s) { strcpy(str, s); }
  char *get() { return str; }

};

// визначення оператора < для об'єктів класу name.
bool operator<(name a, name b)
{
   return strcmp(a.get(), b.get()) < 0;
}

class phoneNum {
  char str[80];
public:
  phoneNum() { strcmp(str, ""); }
  phoneNum(char *s) { strcpy(str, s); }
  char *get() { return str; }
};


int main()
{
  map<name, phoneNum> directory;

 // занесення імен і номерів в асоціативний масив
  directory.insert(pair<name, phoneNum>(name("Іван"),
                   phoneNum("555-4533")));
  directory.insert(pair<name, phoneNum>(name("Петро"),
                   phoneNum("555-9678")));
  directory.insert(pair<name, phoneNum>(name("Горпина"),
```

```
            phoneNum("555-8195")));
  directory.insert(pair<name, phoneNum>(name("Параска"),
                   phoneNum("555-0809")));

  // given a name, find number
  char str[80];
  cout << "Введіть ім'я: ";
  cin >> str;

  map<name, phoneNum>::iterator p;

  p = directory.find(name(str));
  if(p != directory.end())
    cout << "Номер телефону: " <<  p->second.get();
  else
    cout << "Таке ім'я відсутнє в книжці.\n";

  return 0;
}
```

Введіть ім'я: Іван
Номер телефону: 555-4533

//listing 16 - алгоритм count()

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

int main()
{
  vector<bool> v;
  unsigned int i;

  for(i=0; i < 10; i++) {
   if(rand() % 2) v.push_back(true);
   else v.push_back(false);
  }

  cout << "Послідовність:\n";
  for(i=0; i<v.size(); i++)
    cout << boolalpha << v[i] << " ";
  cout << endl;

  i = count(v.begin(), v.end(), true);
  cout << "Заданому предикату <true> відповідає наступна кількість елементів: " <<
i << "\n";

  return 0;
}
```

Послідовність:
true false true true true true false false true true
Заданому предикату <true> відповідає наступна кількість елементів: 7

//listing 17 - алгоритм count_if()

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

// унарний предикат, який визначає чи ділиться число на 3
bool dividesBy3(int i)
{
```

```
  if((i%3) == 0) return true;

  return false;
}

int main()
{
  vector<int> v;
  unsigned int i;

  for(i=1; i < 20; i++) v.push_back(i);

  cout << "Послідовність:\n";
  for(i=0; i<v.size(); i++)
    cout << v[i] << " ";
  cout << endl;

  i = count_if(v.begin(), v.end(), dividesBy3);
  cout << "Кількість чисел кратних 3, дорівнює " << i << "\n";

  return 0;
}
```

Послідовність:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Кількість чисел кратних 3, дорівнює 6

//listing 18 - алгоритми remove_copy і replace_copy

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

int main()
{
  char str[] = "Шаблони STL мають великі можливості";
  int i;
  vector<char> v, v2(80);

  for(i=0; str[i]; i++) v.push_back(str[i]);

  //  алгоритм remove_copy
  cout << "Початкова послідовність:\n";
  for(i=0; i<v.size(); i++) cout << v[i];
  cout << endl;

  // вилучення всіх пропусків
  remove_copy(v.begin(), v.end(), v2.begin(), ' ');

  cout << "Результат після вилучення всіх пропусків:\n";
  for(i=0; i<v2.size(); i++) cout << v2[i];
  cout << endl << endl;

  // алгоритм replace_copy
  cout << "Початкова послідовність:\n";
  for(i=0; i<v.size(); i++) cout << v[i];
  cout << endl;

  // заміна пропусків двокрапками
   replace_copy(v.begin(), v.end(), v2.begin(), ' ', ':');

  cout << "Результат після заміни пропусків двокрапками:\n";
  for(i=0; i<v2.size(); i++) cout << v2[i];
  cout << endl << endl;
```

```
  return 0;
}
```
Початкова послідовність:
Шаблони STL мають великі можливості
Результат після заміни пропусків двокрапкаМИ:
Шаблони:STL:мають:великі:можливості

//listing 19 - алгоритм reverse
```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main()
{
  vector<int> v;
  unsigned int i;

  for(i=0; i<10; i++) v.push_back(i);

  cout << "Початкова послідовність: ";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";
  cout << endl;

  reverse(v.begin(), v.end());

  cout << "Зворотня послідовність ";
  for(i=0; i<v.size(); i++) cout << v[i] << " ";

  return 0;
}
```
Початкова послідовність: 0 1 2 3 4 5 6 7 8 9
Зворотня послідовність 9 8 7 6 5 4 3 2 1 0

//listing 20 - алгоритм transform
```
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

// функція трансформації
double reciprocal(double i) {
  return 1.0/i;
}

int main()
{
  list<double> vals;
  int i;

  // запис значень в список
  for(i=1; i<10; i++) vals.push_back((double)i);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  cout << endl;

  // перетворення значень списку  vals
```

```
  p = transform(vals.begin(), vals.end(),
                vals.begin(), reciprocal);

  cout << "Перетворений вміст сриску vals:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
1 2 3 4 5 6 7 8 9
Перетворений вміст сриску vals:
1 0.5 0.333333 0.25 0.2 0.166667 0.142857 0.125 0.111111

//listing 22 - використання унарного функтора

```
#include <iostream>
#include <list>
#include <functional>
#include <algorithm>
using namespace std;

int main()
{
  list<double> vals;
  int i;

  // запис значень в список
  for(i=1; i<10; i++) vals.push_back((double)i);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;

  // перетворення списку з використанням функтора negate
  p = transform(vals.begin(), vals.end(),
                vals.begin(),
                negate<double>());

  cout << "Перетворений вміст списку vals:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
1 2 3 4 5 6 7 8 9
Перетворений вміст списку vals:
-1 -2 -3 -4 -5 -6 -7 -8 -9

//listing 23 - використання бінарного функтора

```
#include <iostream>
#include <list>
```

```
#include <functional>
#include <algorithm>
using namespace std;

int main()
{
  list<double> vals;
  list<double> divisors;
  int i;

  // запис значення в список
  for(i=10; i<100; i+=10) vals.push_back((double)i);
  for(i=1; i<10; i++) divisors.push_back(3.0);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  cout << endl;

  // трансформація списку  vals
  p = transform(vals.begin(), vals.end(),
                divisors.begin(), vals.begin(),
                divides<double>()); // виклик бінарного функтора

  cout << "Вміст списку після ділення:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
10 20 30 40 50 60 70 80 90
Вміст списку після ділення:
3.33333 6.66667 10 13.3333 16.6667 20 23.3333 26.6667 30

//listing 25 - створення функтора reciprocal.

```
#include <iostream>
#include <list>
#include <functional>
#include <algorithm>
using namespace std;

// Простий функтор
class reciprocal: unary_function<double, double> {
public:
  result_type operator()(argument_type i)
  {
    return (result_type) 1.0/i;
  }
};

int main()
{
  list<double> vals;
  int i;

  // запис значення в список
```

```
  for(i=1; i<10; i++) vals.push_back((double)i);

  cout << "Початковий вміст списку vals:\n";
  list<double>::iterator p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;

  // застосування функтора reciprocal
  p = transform(vals.begin(), vals.end(),
                vals.begin(),
                reciprocal()); // виклик функтора

  cout << "Змінений вміст списку vals:\n";
  p = vals.begin();
  while(p != vals.end()) {
    cout << *p << " ";
    p++;
  }

  return 0;
}
```

Початковий вміст списку vals:
1 2 3 4 5 6 7 8 9
Змінений вміст списку vals:
1 0.5 0.333333 0.25 0.2 0.166667 0.142857 0.125 0.111111

//listing 26 - редактор зв'язків bind2nd()

```
#include <iostream>
#include <list>
#include <functional>
#include <algorithm>
using namespace std;

int main()
{
  list<int> lst;
  list<int>::iterator p, endp;

  int i;

  for(i=1; i < 20; i++) lst.push_back(i);

  cout << "Початкова послідовність:\n";
  p = lst.begin();
  while(p != lst.end()) {
    cout << *p << " ";
    p++;
  }
  cout << endl;

  endp = remove_if(lst.begin(), lst.end(),
                   bind2nd(greater<int>(), 8));

  cout << "Результуюча послідовність:\n";
  p = lst.begin();
  while(p != endp) {
    cout << *p << " ";
    p++;
  }

  return 0;
```

```
}
```
Початкова послідовність:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Результуюча послідовність:
1 2 3 4 5 6 7 8

//listing 30 - клас string
```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  string str1("Alpha");
  string str2("Beta");
  string str3("Omega");
  string str4;

  // присвоєння значень
  str4 = str1;
  cout << str1 << "\n" << str3 << "\n";

  // зчеплення двох string-рядків
  str4 = str1 + str2;
  cout << str4 << "\n";

  // зчеплення string-рядка з  C-char-рядком
  str4 = str1 + " to " + str3;
  cout << str4 << "\n";

  // порівняння двох strings-рядків
  if(str3 > str1) cout << "str3 > str1\n";
  if(str3 == str1+str2)
    cout << "str3 == str1+str2\n";

  // об'єкту string присвоюється значення c-char
  str1 = "Цей символьний рядок закінчується нульовим байтом.\n";
  cout << str1;

  // створення string об'єкта з c-char рядка
  string str5(str1);
  cout << str5;

  // input a string
  cout << "Введіть символьний рядок: ";
  cin >> str5;
  cout << str5;

  return 0;
}
```
Alpha
Omega
AlphaBeta
Alpha to Omega
str3 > str1
Цей символьний рядок закінчується нульовим байтом.
Цей символьний рядок закінчується нульовим байтом.
Введіть символьний рядок: STL
STL

//listing 31 - функції insert(), erase(), and replace().
```
#include <iostream>
#include <string>
```

```
using namespace std;

int main()
{
  string str1("1234567 символьних рядків в стилі C++.");
  string str2("#POWER STL#");

  cout << "Початкові символьні рядки:\n";
  cout << "str1: " << str1 << endl;
  cout << "str2: " << str2 << "\n\n";

  // функція insert()
  cout << "Вставити  str2 в str1 з 7-ї позиції:\n";
  str1.insert(7, str2);
  cout << str1 << "\n\n";

  // функція erase()
  cout << "Вилучити 11 символів з позиції 7 в str1:\n";
  str1.erase(7, 11);
  cout << str1 <<"\n\n";

  // demonstrate replace
  cout << "Замінити з 7-ї позиції  11 символів в str1 рядком str2:\n";
  str1.replace(7, 11, str2);
  cout << str1 << endl;

  return 0;
}
```

```
Початкові символьні рядки:
str1: 1234567 символьних рядків в стилі C++.
str2: #POWER STL#

Вставити  str2 в str1 з 7-ї позиції:
1234567#POWER STL# символьних рядків в стилі C++.

Вилучити 11 символів з позиції 7 в str1:
1234567 символьних рядків в стилі C++.

Замінити з 7-ї позиції  11 символів в str1 рядком str2:
1234567#POWER STL#льних рядків в стилі C++.
```

```
//listing 32 - пошук символів
#include <iostream>
#include <string>
using namespace std;

int main()
{
  int i;
  string s1 =
    "Швидкий думкою, Сильний тілом, Гарячий серцем";
  string s2;

  i = s1.find("Швидкий");
  if(i!=string::npos) {
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }
  cout << "\n\n";

  i = s1.find("Сильний");
  if(i!=string::npos) {
```

```
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }
  cout << "\n\n";

  i = s1.find("Гарячий");
  if(i!=string::npos) {
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }
  cout << "\n\n";

  // знаходимо останню кому
  i = s1.rfind(",");
  if(i!=string::npos) {
    cout << "Знайдено співпадіння в позиції " << i << endl;
    cout << "Залишок:\n";
    s2.assign(s1, i, s1.size());
    cout << s2;
  }

  return 0;
}
```

```
Знайдено співпадіння в позиції 0
Залишок:
Швидкий думкою, Сильний тілом, Гарячий серцем

Знайдено співпадіння в позиції 29
Залишок:
Сильний тілом, Гарячий серцем

Знайдено співпадіння в позиції 56
Залишок:
Гарячий серцем

Знайдено співпадіння в позиції 54
Залишок:
, Гарячий серцем
```

```
//listing 33 - рядки Strings як контейнери
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main()
{
  string str1("Обробка Strings-рядків в C++ дуже проста");
  string::iterator p;
  unsigned int i;

  // використання функції  size()
  for(i=0; i<str1.size(); i++)
    cout << str1[i];
  cout << endl;

  // використання ітератора
  p = str1.begin();
  while(p != str1.end())
    cout << *p++;
```

```
  cout << endl;

  // використання алгоритму  count()
  i = count(str1.begin(), str1.end(), 'i');
  cout << "В string-рядку є " << i << " символів <i>\n";

  // використання функції transform() для перетворення малих букв у великі
  transform(str1.begin(), str1.end(), str1.begin(),toupper);
  p = str1.begin();
  while(p != str1.end())
    cout << *p++;
  cout << endl;


  return 0;
}
```

//listing 34 - використання асоціативного масиву символьних рядків
// для імітації телефонної книжки.

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

int main()
{
  map<string, string> directory;

  directory.insert(pair<string, string>("Всеволод", "555-4533"));
  directory.insert(pair<string, string>("Богодар", "555-9678"));
  directory.insert(pair<string, string>("Меланія", "555-8195"));
  directory.insert(pair<string, string>("Ксенія", "555-0809"));

  string s;
  cout << "Введіть ім'я: ";
  cin >> s;

  map<string, string>::iterator p;

  p = directory.find(s);
  if(p != directory.end())
    cout << "Номер телефону: " << p->second;
  else
    cout << "Таке ім'я відсутнє в довідникеу.\n";

  return 0;
}
```

Введіть ім'я: Всеволод
Номер телефону: 555-4533

Лабораторна робота № 10. Шаблони.

Мета роботи: вивчення шаблонів та їх застосування.

Теоретичний матеріал лекції, [1, розділ 10], [2, розділ 18], [3, розділ 16]

1. Короткі теоретичні відомості

Шаблонна функція – узагальнене визначення функції, за яким компілятор автоматично згенерує код конкретного екземпляру функції. Шаблонна функція оголошується за допомогою кваліфікатора `template`. Параметризований тип задається ключовим словом `class` або `typename`.

```
template <class T1,…,class Tn>
тип_функції імя_функції (T1 t1, … Tn tn);
```

Ключове слово `export` перед оголошенням `template` дозволяє використовувати шаблон з іншого файлу, повторюючи його оголошення а не все визначення.

Ключове слово `typename` може:

- замінити ключове слово `class` в оголошенні шаблону, тобто задавати узагальнений тип;
- інформувати компілятор про те, що деяке ім'я використовується в оголошенні шаблонного класу як ім'я типу, а не об'єкту, наприклад

```
typename X::Name someObject;
```

Тут імя `X::Name` використовується як ім'я типу.

Приклад шаблонної функції для обміну значень двох змінних:

```
template <class T>
void swap(T& t1, T& t2) {
   T t=t1;
   t1=t2;
   t2=t;
}
```

Під час виклику шаблонної функції її формальні параметри типів набувають значень відповідних фактичних аргументів. Якщо фактичний аргумент є об'єктом класу, то, як правило, виникає необхідність у перевантаженні деяких операцій.

Шаблонний клас – це узагальнене визначення множини класів, з якого компілятор автоматично згенерує код конкретного класу. Шаблонний клас реалізує один з варіантів поліморфізму – поліморфізм типів. Оголошення шаблонного класу:

```
template <список параметрів шаблону>
class імя_класу
{
   протокол класу, який використовуэ параметри шаблону
}
```

У списку параметрів шаблону можуть бути *параметризовані і спеціалізовані* типи. Якщо метод шаблонного класу визначається поза протоколом класу, то у заголовку метода вказуються оголошення формальних параметрів, наприклад

```
template <class T, int k>
A<T, k>::~A()
{ delete []p; }
```

Параметри типу можуть набувати стандартних значень або бути типами, визначеними користувачем. Можна визначити параметр-посилання на змінну заданого типу:

```
template <int & ref>
class X {
…
};
```

Параметри шаблону можуть набувати значення за замовчуванням. Якщо один з параметрів

набуває значення за замовчуванням, то усі наступні повинні бути параметрами за замовчуванням:

```
template <class T=int, int k=0>
class A {
   …
public:
   A(int n);
   …
}
```

Тоді при оголошенні об'єкта фактичні аргументи шаблону, які набувають значення за замовчуванням, можна не задавати:

```
void main() {
   A<> a(5);
   A<unsigned> b(10);
   A<short, 7> c(20);
}
```

Допускається вкладеність шаблонів при оголошенні об'єктів параметризованих класів:

```
template <class T>
class X {…};
template <int n>
class Y {…};
int main() { X<Y<5>> x; }
```

Методи шаблонного класу можуть бути шаблонними функціями:

```
template <class T1, class T2>
class X {
public:
   template <class U, class V>
      void mf(const U &u, const V &v) {}
};
```

Визначаючи шаблонний метод поза класом, необхідно повторити `template`-оголошення для кожної групи параметрів типу:

```
template <class T1, cass T2>
template <class U, class V>
   void X<T1, T2>::mf(const U &u, const V &v){}
```

Статичні елементи шаблонних класів є спільними для усіх об'єктів з однаковими значеннями аргументів шаблону.

1.1. Друзі шаблонних класів

Друзями шаблонних класів можуть бути зовнішні функції, методи інших класів, інші класи. Якщо дружня функція використовує об'єкт шаблонного класу, то перед її заголовком необхідно розмістити `template`-оголошення:

```
template <class T>
class A {
   friend void f1();
template <class T>
   friend void f2(A<T>&);
template <class U>
   friend void f3(U&);
};
// звичайна функція, друг множини класів, відношення 1:n
void f1();
// функція-друг одного типу T, відношення 1:1
template <class T>
void f2(A<T>& a) {}
// параметризована дружня функція, відношення m:n
template <class U>
void f3(U& u) {}
```

```
void main() {
   f1();
   A<int> a;
   f2(a);
   double x;
   f3(x);
}
```

Друзями шаблонного класу можуть бути методи інших класів:

```
template <class T> class A;
template <class T>
class D {
public:
template <class U>
   void f3(U& u) {}     // шаблонний метод шаблонного класу
};

template <class T>
class C {
public:
   void f2(A<T>& a) {} // звичайний метод шаблонного класу
};

class B {
public:
   void f1() {}        // звичайний метод нешаблонного класу
};
template <class T>
class A {
   friend void B::f1();
   friend void C::f2(A<T>&);
   template <class T>
   template <class U>
      friend void D<U>::f3(U&);
};
void main() {…}
```

1.2. Перевантаження операцій шаблонних класів

Під час роботи з шаблонними класами виникає необхідність у перевантаженні операцій. Приклад перевантаження бінарної операції `operator+` як члена класу:

```
template <class T>
class A {
   T x;
public:
   A(){}
   A(T x){this->x=x;}
   A<T> operator+(A<T>&);
   void print(){cout<<x<<endl;}
};
template <class T> A<T> A<T>::operator+(A<T>& a) {
   return A<T>(x+a.x);
}
void main() {
   A<int> a(2),b(3),c;
   c=a+b;
   c.print();
}
```

1.3. Віртуальні методи шаблонних класів

Шаблонний клас може містити оголошення віртуальних методів. Шаблонні методи (з оголошенням template) не можуть бути віртуальними.

Віртуальні методи забезпечують пізнє зв'язування, якщо вони викликаються за допомогою вказівників (або посилань) на базовий клас, проініціалізованих адресою (або ідентифікатором об'єкта – для посилань) похідного класу з public успадкуванням. Для шаблонних класів вказівник (або посилання) на базовий клас та об'єкт повинні мати однакову параметризацію.

Приклад програми. Створити шаблонний клас `vector` з двома параметрами: перший є параметром типу, а другий – цілочисловим значенням для ініціалізації елементів вектора. Визначити конструктор, деструктор, методи пошуку екстремальних значень (мінімуму та максимуму), впорядкування, обчислення евклідової норми. Перевантажити операції +, =, [], <<, >> відповідно для додавання, присвоєння, контролю діапазону індексу та введення об'єктів.

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

using namespace std;

template <class T, int k>
class vector
{
  //T *v;
  int size;
public:
  T *v;
  vector(int newsize);
  ~vector();

  int getsize() { return size; }

  T extr(char *);
  vector& sort(char *);
  double norma(void);

  vector& operator+(vector&);
  T& operator[] (int index);
  vector& operator=(const vector&);

  template <class Y>
    friend ostream& operator<<(ostream&, vector&);
  template <class Y>
    friend istream& operator>>(istream&, vector&);
};


template <class T, int k>
  vector<T,k>::vector(int newsize)
  {
    v=new T[size=newsize];
    for(int i=0;i<size;i++) v[i]=k;
  }


  template <class T, int k>
  vector<T,k>::~vector() { delete []v; }


  template <class T, int k>
  T vector<T,k>::extr(char * MinOrMax)
  {
    T ExtrElem=v[0];
```

```
for(int i=0;i<size;i++)
{
  if (!strncmp(MinOrMax,"min",3))
  {
  if(v[i]<ExtrElem) ExtrElem=v[i];
  }
  else if(v[i]>ExtrElem) ExtrElem=v[i];

}
  return ExtrElem;
}

template <class T, int k>
vector<T,k>& vector<T,k>::sort(char * UpOrDown)
{
  T x;
  for(int i=0;i<size-1;i++)
  for(int j=i+1;j<size;j++)
    if( !strncmp(UpOrDown,"max",3) )
    {
      if(v[i]>v[j]) {
        x=v[i];
        v[i]=v[j];
        v[j]=x;
      }
    }
    else
      if(v[i]<v[j]) {
        x=v[i];
        v[i]=v[j];
        v[j]=x;
      }
      return *this;
}

template <class T, int k>
 double vector<T,k>::norma(void)
 {
  double s=0;
  for(int i=0;i<size;i++) s+=v[i]*v[i];
  return sqrt(s);
  }


template <class T, int k>
vector<T,k>& vector<T,k>::operator+(vector<T,k>& y)
{
  if(size!=y.size) false;
  for(int i=0;i<size;i++) v[i]+=y.v[i];
  return *this;
}

template <class T, int k>
T& vector<T,k>::operator[] (int index)
{
  if(index<0 || index>=size)
  throw "Індекс за межами діапазону можливих значень";
  return v[index];
}

template <class T, int k>
vector<T,k>& vector<T,k>::operator=(const vector<T,k>& x)
{
  if(this!=&x)
```

```
      {
      delete []v;
      v=new T[size=x.size];
      for(int i=0;i<size;i++) v[i]=x.v[i];
      }
      return *this;
    }

    template <class T>
    istream& operator>>(istream& is, vector<T,0>& x)
    {
      cout<<"Введіть "<<x.getsize()<<" елементів вектора\n";
      for(int i=0;i<x.getsize();i++) is>>x.v[i];  // ???
      return is;
    }

    template <class T>
    ostream& operator<<(ostream& os, vector<T,0>& x)
    {
      for(int i=0;i<x.getsize();i++) os<<x[i]<<' ';
      os<<endl;
      return os;
    }

    int main() {
      int n=5;
      vector<int,0> V(n),U(n);
      vector<int,0> *Z=new vector<int,0>(n);

      char s1[]="min", s2[]="max";

      try
      {
      cin>>V;
      cin>>U;
      *Z=V+U;
      cout<<"Сума " << *Z;
      cout<<"Мінімальний елемент " <<Z->extr(s1)<<endl;
      //cout<<"Максимальний елемент "<<Z->extr(s2)<<endl;
      Z->sort(s2);
      cout<<"Сума відсортована "<<*Z;
      cout<<"Норма "<<Z->norma()<<endl;
      }
      catch(bool)
      {
      cout<<"Різна кількість елементів векторів"<< endl;
      }
      catch(char *s)
      {
      cout<<s<<endl;
      }

    }
```

```
Введіть 5 елементів вектора
3 23 65 3 9
Введіть 5 елементів вектора
12 9 34 66 3
Сума 15 32 99 69 12
Мінімальний елемент 12
Сума відсортована 12 15 32 69 99
Норма 126.313
```

Запитання.

1. Що таке шаблонна функція і як вона оголошується?
2. Що таке шаблонний клас і як він оголошується?
3. Що таке параметризовані і спеціалізовані типи в шаблонах функцій і класів?
4. Які можуть бути друзі у шаблонних методів?
5. Перевантаження операцій, як членів шаблонних класів.
6. Ключові слово `typename` і `export`.

Завдання.

1. Написати шаблонний клас для послідовного пошуку елементів МАСИВУ за заданим ключем. Якщо елемент знайдено, то на екран виводиться уся інформація, що відповідає заданому ключу. Застосувати клас для пошуку елементів різних типів.

2. Створити шаблонний клас для роботи з ФАЙЛАМИ даних різних типів. Перевизначити методи відкривання файлу, запису даних у файл, зчитування даних з файлу. Створити файл рядків та файл дійсних чисел. Знайти найдовший рядок та найбільше дійсне число у створених файлах.

3. Написати шаблонний клас для сортування одновимірного МАСИВУ за зростанням значень елементів. Застосувати цей клас для сортування масивів цілих та дійсних чисел, масивів та рядків символів.

4. Побудувати шаблонний клас ВПОРЯДКОВАНИЙ МАСИВ. Включити елементи у масив так, щоб не порушити впорядкованості масиву. Вилучити елемент з масиву. Вивести масив на екран. Застосувати шаблон для роботи з різними типами елементів масивів.

5.Написати шаблонний клас, який знаходить контрольну суму ЕЛЕМЕНТА довільного типу. Контрольна сума – це кількість одиниць у машинному зображенні заданого елемента.

6. Створити параметризований клас однозв’язного СПИСКУ. Тип елемента списку визначається параметром шаблона. Передбачити функції для виконання таких операцій: створення нового елемента списку на його початку; вилучення першого елемента списку; створення нового елемента списку у його кінці; вилучення останнього елемента списку; визначення кількості елементів списку.

7. Створити параметризований однонаправлений лінійний кільцевий СПИСОК. Тип елемента списку визначається параметром шаблону. Передбачити функції для виконання таких операцій: занесення елемента списку у список; вилучення елемента зі списку; виведення елементів списку на екран; визначення кількості елементів списку.

8. Написати шаблонний клас, який створює копію двовимірного динамічного МАСИВУ довільного типу. Передбачити операції копіювання за рядками, за стовпцями, копіювання вибраного рядка або стовпчика, копіювання головної діагоналі. Розробити функції потокового введення-виведення масивів.

9. Створити клас СТЕК, який ґрунтується на статичному масиві вказівників. Тип елемента стеку визначається параметром шаблону. Передбачити функції для виконання таких операцій: занесення елементів у стек; вилучення значення з верхівки стеку; виведення усіх значень стеку на екран; повернення кількості елементів стеку.

10. Розробити шаблонний клас СТЕК. Побудувати два стеки із впорядкованих елементів даних. Виконати злиття стеків в один стек так, щоб не порушити загальної впорядкованості елементів.

11. Створити клас БЛОКНОТ, параметризований структурованими типами даних. Визначити конструктори, деструктор, методи роботи з даними. На основі класу БЛОКНОТ створити клас ЖУРНАЛ обліку продукції на складі. Вивести список товарів та їх кількість.

12. Розробити шаблонний клас КОНТРОЛЕР файлової системи для керування доступом до файлів і каталогів. Використати КОНТРОЛЕР для роботи з об’єктами класів ФАЙЛ і КАТАЛОГ. Виконати пошук, копіювання, перейменування, витирання, виведення вмісту файла

або каталога на екран. Визначити необхідні дані, конструктори, деструктори, методи та перевантажені операції. Передбачити опрацювання можливих помилок.\

13. Розробити клас ПРИВІД оптичного диску, який може бути параметризований об'єктами класів CD або DVD. Забезпечити ідентифікацію об'єктів CD/DVD та виведення їхнього вмісту на еран у вигляді списку файлів та каталогів. Визначити необхідні дані, конструктори, деструктори, методи та перевантажені операції.

2. Приклади для самостійної роботи

```
/* listing 1 - приклад шаблонної функції, яка міняє місцями дві змінні незалежно
від їх типу. Позначення шаблонної функції:
template <class Тип> return_тип імя_функції(список параметрів узаг. типів)
або
template <typename Тип> return_тип імя_функції(список параметрів узаг. типів)
class або typename – синоніми.
*/
#include <iostream>
using namespace std;

// Шаблонна функція
template <class X> void swapargs(X &a, X &b)
//template <typename X> void swapargs(X &a, X &b)

{
  X temp;

  temp = a;
  a = b;
  b = temp;
}

int main()
{
  int i=10, j=20;
  double x=10.1, y=23.3;
  char a='x', b='z';

  cout << "Початкові значення i, j: " << i << ' ' << j << '\n';
  cout << "Початкові значення x, y: " << x << ' ' << y << '\n';
  cout << "Початкові значення a, b: " << a << ' ' << b << '\n';

  swapargs(i, j); // переставлення цілих    integers
  swapargs(x, y); // переставлення дійсних  floats
  swapargs(a, b); // переставлення символів chars

  cout << "Переставлені значення i, j: " << i << ' ' << j << '\n';
  cout << "Переставлені значення x, y: " << x << ' ' << y << '\n';
  cout << "Переставлені значення a, b: " << a << ' ' << b << '\n';

  return 0;
}
Початкові значення i, j: 10 20
Початкові значення x, y: 10.1 23.3
Початкові значення a, b: x z
Переставлені значення i, j: 20 10
Переставлені значення x, y: 23.3 10.1
Переставлені значення a, b: z x

/* listing 5 – шаблонна функція з двома узагальненими типами */
#include <iostream>
using namespace std;
/* шаблонна функція записана в два рядки */
```

```
template <class type1, typename type2>
void myfunc(type1 x, type2 y)
{
  cout << x << ' ' << y << '\n';
}

int main()
{
  myfunc(10, "Я вчу C++");

  myfunc(98.6, 19L);

  return 0;
}
```

```
10 Я вчу C++
98.6 19
```

```
/* listing 6 – явне перевантаження шаблонної функції */
#include <iostream>
using namespace std;

// Шаблонна функція
template <class X> void swapargs(X &a, X &b)
{
  X temp;

  temp = a;
  a = b;
  b = temp;
  cout << "всередині шаблону swapargs.\n";
}

// Заміщення шаблонної функції swapargs() для цілих чисел
void swapargs(int &a, int &b)
{
  int temp;

  temp = a;
  a = b;
  b = temp;
  cout << "Всередині спеціалізації функції swapargs для цілих\n";
}

/* ---------------------------------------------------------------
нова синтаксична конструкція для явного заміщення (спеціалізації) параметрів
шаблонної функції
template<> void swapargs<int>(int &a, int &b)
{
  int temp;

  temp = a;
  a = b;
  b = temp;
  cout << "Всередині спеціалізації функції swapargs для цілих\n";
}
--------------------------------------------------------------- */

int main()
{
  int i=10, j=20;
  double x=10.1, y=23.3;
  char a='x', b='z';

  cout << "Початкові значення i, j: " << i << ' ' << j << '\n';
```

```
  cout << "Початкові значення x, y: " << x << ' ' << y << '\n';
  cout << "Початкові значення a, b: " << a << ' ' << b << '\n';

  swapargs(i, j); // виклик явно перевантаженої функції swapargs()
  swapargs(x, y); // виклик узагальненої функції swapargs()
  swapargs(a, b); // виклик узагальненої функції swapargs()

  cout << "Переставлені значення i, j: " << i << ' ' << j << '\n';
  cout << "Переставлені значення x, y: " << x << ' ' << y << '\n';
  cout << "Переставлені значення a, b: " << a << ' ' << b << '\n';

  return 0;
}
```

Початкові значення i, j: 10 20
Початкові значення x, y: 10.1 23.3
Початкові значення a, b: x z
Всередині спеціалізації функції swapargs для цілих
всередині шаблону swapargs.
всередині шаблону swapargs.
Переставлені значення i, j: 20 10
Переставлені значення x, y: 23.3 10.1
Переставлені значення a, b: z x

/* listing 8 - перевантаження шаблонної функції створенням шаблонів, які відрізняються списком параметрів */

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

// Перша версія шаблонної функції f()
template <class X> void f(X a)
{
  cout << "Всередені функції f(X a)\n";
}

// Друга версія шаблонної функції  f()
template <class X, typename Y> void f(X a, Y b)
{
  cout << "Всередині функції f(X a, Y b)\n";
}

int main()
{
  f(10);     // виклик функції f(X)
  f(10, 20); // виклик функції f(X, Y)

  return 0;
}
```

Всередені функції f(X a)
Всередині функції f(X a, Y b)

/* listing 9 – змішування узагальнених і стандартних параметрів */

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

const int TABWIDTH = 8;

// Виведення на екран даних з позиції tab
template <class X> void tabOut(X data, int tab)
{
  for(; tab; tab--)
    for(int i=0; i<TABWIDTH; i++) cout << ' ';

    cout << data << "\n";
```

```
}

int main()
{
  tabOut("Перевірка", 0);
  tabOut(100, 1);
  tabOut('X', 2);
  tabOut(10/3, 3);

  return 0;
}
```

Перевірка
100
X
3

```
/* обмеження на узагальнені функції. Узагальнені функції повинні мати однакове тіло для всіх реалізацій.
/* listing 10 – наступні функції не можна замінити на узагальнені, так як вони мають різне тіло і різне призначення */
#include <iostream>
#include <cmath>
using namespace std;

void myfunc(int i)
{
  cout << "Значення дорівнює: " << i << "\n";
}

void myfunc(double d)
{
  double intpart;
  double fracpart;

  fracpart = modf(d, &intpart);
  cout << "Дробова частина: " << fracpart;
  cout << "\n";
  cout << "Ціла частина: " << intpart;
}

int main()
{
  myfunc(1);
  myfunc(12.2);

  return 0;
}
```

Значення дорівнює: 1
Дробова частина: 0.2
Ціла частина: 12

```
//  застосування узагальнених (шаблонних) функцій
/* listing 11 - узагальнене сортування методом бульбашки */
#include <iostream>
using namespace std;

template <class X> void bubble(
  X *items,  // вказівник на впорядковуваний масив
  int count) // число елементів в масиві
{
  register int a, b;
  X t;
```

```
  for(a=1; a<count; a++)
    for(b=count-1; b>=a; b--)
      if(items[b-1] > items[b]) {
        // переставлення елементів
        t = items[b-1];
        items[b-1] = items[b];
        items[b] = t;
      }
}

int main()
{
  int iarray[7] = {7, 5, 4, 3, 9, 8, 6};
  double darray[5] = {4.3, 2.5, -0.9, 100.2, 3.0};

  int i;

  cout << "Невпорядкований масив цілих чисел: ";
  for(i=0;  i<7; i++)
    cout << iarray[i] << ' ';
  cout << endl;

  cout << "Невпорядкований масив дійсних чисел double: ";
  for(i=0;  i<5; i++)
    cout << darray[i] << ' ';
  cout << endl;

  bubble(iarray, 7);
  bubble(darray, 5);

  cout << "Відсортований масив цілих чисел: ";
  for(i=0;  i<7; i++)
    cout << iarray[i] << ' ';
  cout << endl;

  cout << "Відсортований масив дійсних чисел double: ";
  for(i=0;  i<5; i++)
    cout << darray[i] << ' ';
  cout << endl;

  return 0;
}
```

```
Невпорядкований масив цілих чисел: 7 5 4 3 9 8 6
Невпорядкований масив дійсних чисел double: 4.3 2.5 -0.9 100.2 3
Відсортований масив цілих чисел: 3 4 5 6 7 8 9
Відсортований масив дійсних чисел double: -0.9 2.5 3 4.3 100.2
```

```
/* listing 12 - узагальнена функція ущільнення масиву  */
#include <iostream>
#include <string.h>

using namespace std;

template <class X> void compact(
  X *items,  // вказівник на ущільнюваний масив
  int count, // число елементів в масиві
  int start, // індекс першого вилученого елементу
  int end)   // індекс останнього вилученого елементу
{
  register int i;

  for(i=end+1; i<count; i++, start++)
    items[start] = items[i];
```

```
  /* з демонстраційною метою незаповнена частина масиву обнуляється */
  for( ; start<count; start++) items[start] = (X) 0;
}

int main()
{
  int nums[] = {0, 1, 2, 3, 4, 5, 6};
  char str[] = "Common function";

  int i,N1,N2;

  cout << "size(int)=" << sizeof(int) << "\n";
  N1=sizeof(nums)/4;
  cout << "Неущільнений int масив size=" << N1 << " :";
  for(i=0;  i<N1; i++)
    cout << nums[i] << ' ';
  cout << endl;

  cout << "size(char)=" << sizeof(char) << "\n";
  N2=strlen(str);
  cout << "Неущільнений символьний рядок size(string)=" << N2 << " :";
  for(i=0;  i<N2; i++)
    cout << str[i] << ' ';
  cout << endl;

  compact(nums, N1, 2, 4);
  compact(str, N2, 6, 10);

  cout << "Ущільнений integer масив: ";
  for(i=0;  i<7; i++)
    cout << nums[i] << ' ';
  cout << endl;

  cout << "Ущільнений символьний рядок: ";
  for(i=0;  i<N2; i++)
    cout << str[i] << ' ';
  cout << endl;

  return 0;
}
```

```
size(int)=4
Неущільнений int масив size=7 :0 1 2 3 4 5 6
size(char)=1
Неущільнений символьний рядок size(string)=15 :C o m m o n   f u n c t i o n
Ущільнений integer масив: 0 1 5 6 0 0 0
Ущільнений символьний рядок: C o m m o n t i o n
```

```
/* Оголошення узагальненого класу
template <class Тип> class імя_класу
{
…
}
Створення обєкту узагальненого класу
імя_класу <типЮ імя_обєкту;
*/

/* listing 13 - узагальнений стек */
#include <iostream>
using namespace std;

const int SIZE = 10;
```

```
// Створення узагальненого класу стек
template <class StackType> class stack {
  StackType stck[SIZE];     // містить елементи стека
  int tos;                  // індекс верхівки стека

public:
  stack() { tos = 0; }      // ініціалізація стека
  void push(StackType ob);  // заштовхування обєкту в стек
  StackType pop();          // виштовхування об'єкту зі стека
};

// реалізація функції push - заштовхування об'єкта в стек
template <class StackType>
void stack<StackType>::push(StackType ob)
{
  if(tos==SIZE) {
    cout << "Стек повний\n";
    return;
  }
  stck[tos] = ob;
  tos++;
}
// реалізація функції pop - виштовхування зі стеку
template <class StackType>
StackType stack<StackType>::pop()
{
  if(tos==0) {
    cout << "Стек порожній\n";
    return 0; // якщо стек порожній, то повертається null
  }
  tos--;
  return stck[tos];
}

int main() {
 // стек символів
 stack <char> s1, s2;
 int i;
 s1.push('a');
 s2.push('x');
 s1.push('b');
 s2.push('y');
 s1.push('c');
 s2.push('z');

 for(i=0; i<3; i++) cout << "Виштовхування s1: " << s1.pop() << "\n";
 for(i=0; i<3; i++) cout << "Виштовхування s2: " << s2.pop() << "\n";

 // стек дійсних чисел
 stack <double> ds1, ds2;
 ds1.push(1.1);
 ds2.push(2.2);
 ds1.push(3.3);
 ds2.push(4.4);
 ds1.push(5.5);
 ds2.push(6.6);

for(i=0; i<3; i++) cout << "Виштовхування ds1: " << ds1.pop() << "\n";
for(i=0; i<3; i++) cout << "Виштовхування ds2: " << ds2.pop() << "\n"

  return 0;
}
```

Виштовхування s1: c

```
Виштовхування  s1: b
Виштовхування  s1: a
Виштовхування  s2: z
Виштовхування  s2: y
Виштовхування  s2: x
Виштовхування  ds1: 5.5
Виштовхування  ds1: 3.3
Виштовхування  ds1: 1.1
Виштовхування  ds2: 6.6
Виштовхування  ds2: 4.4
Виштовхування  ds2: 2.2
```

```
/* listing 18 – використання класом двох узагальнених типів */
#include <iostream>
using namespace std;

template <class Type1, class Type2> class myclass
{
  Type1 i;
  Type2 j;
public:
  myclass(Type1 a, Type2 b) { i = a; j = b; }
  void show() { cout << i << ' ' << j << '\n'; }
};

int main()
{
  myclass<int, double> ob1(10, 0.23);
  myclass<char, char *> ob2('X', "Шаблони потужний механізм");

  ob1.show(); // show int, double
  ob2.show(); // show char, char *

  return 0;
}
```

```
10 0.23
X Шаблони потужний механізм
```

```
/* listing 19 – комбінування перевантаженого оператора [] з шаблонним класом для створення узагальненого безпечного масиву */
#include <iostream>
#include <cstdlib>
using namespace std;

const int SIZE = 10;

template <class AType> class atype {
  AType a[SIZE];
public:
  atype() {
    register int i;
    for(i=0; i<SIZE; i++) a[i] = i;
  }
  AType &operator[](int i);
};

// Перевірка діапазону масива для об'єкту  Atype.
template <class AType> AType &atype<AType>::operator[](int i)
{
  if(i<0 || i> SIZE-1) {
    cout << "\nЗначення індекса ";
    cout << i << " виходить за допустимі межі.\n";
    exit(1);
  }
```

```
  return a[i];
}

int main()
{
  atype<int> intob; // масив integer
  atype<double> doubleob; // масив double

  int i;

  cout << "Масив integer: ";
  for(i=0; i<SIZE; i++) intob[i] = i;
  for(i=0; i<SIZE; i++) cout << intob[i] << "  ";
  cout << '\n';

  cout << "Масив Double: ";
  for(i=0; i<SIZE; i++) doubleob[i] = (double) i/3;
  for(i=0; i<SIZE; i++) cout << doubleob[i] << "  ";
  cout << '\n';

  intob[12] = 100; // генерація повідомлення про помилку під час виконання

  return 0;
}
```

Масив integer: 0 1 2 3 4 5 6 7 8 9
Масив Double: 0 0.333333 0.666667 1 1.33333 1.66667 2 2.33333 2.66667 3
Значення індекса 12 виходить за допустимі межі.

```
/* listing 20 – використання стандартних типів в узагальнених класах.
В шаблонних параметрах можна використовувати стандартні типи */
#include <iostream>
#include <cstdlib>
using namespace std;

// Аргумент int size є стандартним
template <class AType, int size> class atype {
  AType a[size]; // довжина масиву передається через параметр size
public:
  atype() {
    register int i;
    for(i=0; i<size; i++) a[i] = i;
  }
  AType &operator[](int i);
};

// Перевірка діапазону для об'єкту atype.
template <class AType, int size>
AType &atype<AType, size>::operator[](int i)
{
  if(i<0 || i> size-1) {
    cout << "\nЗначення індекса ";
    cout << i << " виходить за допустимі межі\n";
    exit(1);
  }
  return a[i];
}

int main()
{
  atype<int, 10> intob;       // масив integer розміром 10
  atype<double, 15> doubleob; // масив double розміром  15

  int i;
```

```
  cout << "Масив integer: ";
  for(i=0; i<10; i++) intob[i] = i;
  for(i=0; i<10; i++) cout << intob[i] << "  ";
  cout << '\n';

  cout << "Масив double: ";
  for(i=0; i<15; i++) doubleob[i] = (double) i/3;
  for(i=0; i<15; i++) cout << doubleob[i] << "  ";
  cout << '\n';

  intob[12] = 100; // генерація повідомлення про помилку

  return 0;
}
```

```
Масив integer: 0  1  2  3  4  5  6  7  8  9
Масив double: 0  0.333333  0.666667  1  1.33333  1.66667  2  2.33333  2.66667  3
3.33333  3.66667  4  4.33333  4.66667
Значення індекса 12 виходить за допустимі межі
```

```
/* listing 23 – використання аргументів за замовчуванням у шаблонних класах */
#include <iostream>
#include <cstdlib>
using namespace std;

// Параметр типу AType по замовчуванню дорівнює int, а змінна i дорівнює 10.
template <class AType=int, int size=10> class atype {
  AType a[size]; // розмір масиву передається аргументом size
public:
  atype() {
    register int i;
    for(i=0; i<size; i++) a[i] = i;
  }
  AType &operator[](int i);
};

// Перевірка діапазону для об'єкту atype.
template <class AType, int size>
AType &atype<AType, size>::operator[](int i)
{
  if(i<0 || i> size-1) {
    cout << "\nЗначення індекса ";
    cout << i << " виходить за межі\n";
    exit(1);
  }
  return a[i];
}

int main()
{
  atype<int, 5> intarray;  // integer масив, розміром 100
  atype<double> doublearray; // double  масив, розміром size
  atype<> defarray;      // по замовчуванню оголошено int масив розміром 10

  int i;

  cout << "int масив: ";
  for(i=0; i<5; i++) intarray[i] = i;
  for(i=0; i<5; i++) cout << intarray[i] << "  ";
  cout << '\n';

  cout << "double масив: ";
  for(i=0; i<10; i++) doublearray[i] = (double) i/3;
  for(i=0; i<10; i++) cout << doublearray[i] << "  ";
  cout << '\n';
```

```
  cout << "масив по замовчуванню: ";
  for(i=0; i<10; i++) defarray[i] = i;
  for(i=0; i<10; i++) cout << defarray[i] << "  ";
  cout << '\n';

  return 0;
}
int масив: 0  1  2  3  4
double масив: 0  0.333333  0.666667  1  1.33333  1.66667  2  2.33333  2.66667  3
масив по замовчуванню: 0  1  2  3  4  5  6  7  8  9

/* listing 25 -  явна спеціалізація (заміщення типів) узагальненого класу
задається конструкцією template<> */
#include <iostream>
using namespace std;

template <class T> class myclass {
  T x;
public:
  myclass(T a) {
    cout << "Всередині узагальненого класу\n";
    x = a;
  }
  T getx() { return x; }
};

// Явна спеціалізація типу int для узагальненого класу
template <> class myclass<int> {
  int x;
public:
  myclass(int a) {
    cout << "Всередині спеціалізації  myclass<int>\n";
    x = a * a;
  }
  int getx() { return x; }
};

int main()
{
  myclass<double> d(10.1);
  cout << "double: " << d.getx() << "\n\n";

  myclass<int> i(5);
  cout << "int: " << i.getx() << "\n";

  return 0;
}
Всередині узагальненого класу
double: 10.1
Всередині спеціалізації  myclass<int>
int: 25
```