МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет
науки ітехнологій**

Кафедра економічної інформатики

В авторській редакції

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Навчально-методичні рекомендації
з виконання лабораторних робіт

ДНІПРО
2024

УДК 004.652.5(076.5)
О 29

Упорядники:
Л. І. Лозовська, Л. М. Савчук, Т. О. Климкович

Електронний аналог
друкованого видання

Схвалено Групою забезпечення якості освітньої програми
«Комп'ютерні технології в бізнесі»
Протокол № 2 від 18.вересня 2024 р.
«Інформаційні технології та моделювання в економіці»
Протокол № 2 від 19 вересня 2024 р.

О 29 Об'єктно-орієнтоване програмування : навчально-методичні рекомендації з виконання лабораторних робіт / упоряд. Л. І. Лозовська, Л. М. Савчук, Т. О. Климкович ; Укр. держ. ун-т науки і технологій. – Дніпро : УДУНТ, 2024. – 60 с.

Навчально-методичні рекомендації призначені для використання студентами денної та заочної форм навчання спеціальностей 051 – Економіка, 126 – Інформаційні системи та технології (бакалаврський рівень) під час виконання лабораторних робіт з дисципліни «Об'єктно-орієнтоване програмування».

Навчально-методичні рекомендації містять методичні матеріали для вивчення основ об'єктно-орієнтованого програмування мовою високого рівня С++. Одночасне ознайомлення з можливостями системи програмування і конструкцій мови програмування дозволяє студентам з перших занять одержати уявлення про технологію розробки об'єктно-орієнтованих додатків.

ЗМІСТ

ПЕРЕДМОВА

Мета навчальної дисципліни «Об’єктно-орієнтоване програмування» полягає у формуванні теоретичних знань і практичних навичок створення програмних систем з використанням об’єктно-орієнтованої технології.

Навчальна дисципліна забезпечує набуття таких передбачених освітньою програмою компетентностей:

Здатність до проектування, розробки, налагодження та вдосконалення системного, комунікаційного та програмно-апаратного забезпечення інформаційних систем та технологій, Інтернету речей (IoT), комп’ютерно-інтегрованих систем та системної мережної структури, управління ними.

Здатність проектувати, розробляти та використовувати засоби реалізації інформаційних систем, технологій та інфокомунікацій (методичні, інформаційні, алгоритмічні, технічні, програмні та інші).

для ОПП «Комп’ютерні технології в бізнесі» та :

Здатність до абстрактного мислення, аналізу та синтезу.

Здатність бути критичним і самокритичним.

Здатність приймати обґрунтовані рішення.

Здатність самостійно виявляти проблеми економічного характеру при аналізі конкретних ситуацій, пропонувати способи їх вирішення.

Здатність до розробки програмного забезпечення інформаційних систем або їх фрагментів.

для ОПП «Інформаційні технології та моделювання в економіці».

Відповідно до освітньої програми дисципліна спільно з іншими освітніми компонентами має забезпечити досягнення таких програмних результатів навчання:

Використовувати базові знання інформатики й сучасних інформаційних систем та технологій, навички програмування, технології безпечної роботи в комп'ютерних мережах, методи створення баз даних та інтернет-ресурсів, технології розроблення алгоритмів і комп’ютерних програм мовами високого рівня із застосуванням об’єктно-орієнтованого програмування для розв’язання задач проектування і використання інформаційних систем та технологій.

Аргументувати вибір програмних та технічних засобів для створення інформаційних систем та технологій на основі аналізу їх властивостей,

призначення і технічних характеристик з урахуванням вимог до системи і експлуатаційних умов; мати навички налагодження та тестування програмних і технічних засобів інформаційних систем та технологій.

Обґрунтовувати вибір технічної структури та розробляти відповідне програмне забезпечення, що входить до складу інформаційних систем та технологій

для ОПП «Комп’ютерні технології в бізнесі» та :

Демонструвати здатність до розробки програмного забезпечення інформаційних систем або їх фрагментів, розробки автономних додатків

для ОПП «Інформаційні технології та моделювання в економіці».

Навчальна дисципліна є обов'язковою для вивчення студентами, які здобувають освітній ступінь бакалавра за освітніми програмами «Комп’ютерні технології в бізнесі» та «Інформаційні технології та моделювання в економіці».

Відповідно до робочої програми навчальної дисципліни «Об’єктно-орієнтоване програмування» передбачено виконання лабораторних робіт. Дане методичне видання містить основні теоретичні положення для засвоєння матеріалу, інструкції до виконання завдань, запитання до захисту лабораторних робіт.

ЛАБОРАТОРНА РОБОТА № 1

ТЕМА РОБОТИ: Програмування з використанням класів.
МЕТА РОБОТИ: Навчитися виявляти з опису предметної області об’єкти, його атрибути та методи, зображати клас графічно (мовою UML). Набути практичних навичок оголошення класів, опису членів функцій класів, створення об'єктів, виклику членів-функцій класу на мові С++.
ТРИВАЛІСТЬ: 4 години.

МЕТОДИЧНІ ВКАЗІВКИ

Основними поняттями об'єктно-орієнтованого програмування є об'єкт і клас. Класи надають програмісту можливість моделювати об'єкти, які мають атрибути (представлені як дані) і варіанти поведінки або операції (представлені як функції). Функції іноді називають методами, вони викликаються у відповідь на повідомлення, що посилаються об'єкту. Повідомлення відповідає виклику функції.

Визначення або оголошення класу починається з ключового слова **class**. Далі у фігурних дужках ({}) описується тіло класу. Визначення класу закінчується крапкою з комою. Тіло класу включає дані (члени-дані), методи (члени-функції) і специфікатори доступу до членів класу.

Специфікатори доступу до членів класу завжди закінчуються двокрапкою (:) і можуть з'являтися у визначенні класу багато раз і у будь-якому порядку.

Визначення (оголошення) класу записується таким чином:

class <ім’я класу>
{
<тип атрибуту 1> < ім’я атрибуту 1 >;
<тип атрибуту 2> < ім’я атрибуту 2>;
...
специфікатор доступу:
<тип функції 1> < ім’я функції 1>;
<тип функції 2> < ім’я функції 2>;
....}**;**

Після того, як клас визначений, і його функції-члени оголошені, ці функції-члени повинні бути визначені (описані). Кожна функція-член може бути описана прямо в тілі класу або після тіла класу. Коли функція-член описується після відповідного визначення класу, імені функції передує ім'я класу і оператор дозволу області дії (::). Оскільки різні класи можуть мати члени з однаковими іменами, оператор дозволу області дії «прив'язує» ім'я члена до імені класу, щоб однозначно ідентифікувати функції-члени даного класу.

Таким чином, клас - це структурований тип даних, визначений користувачем. Коли клас визначений, ім'я класу може бути використано для оголошення об'єкту цього класу. Методи, оголошені усередині класу можна викликати тільки для об'єктів даного класу, тобто змінних типу даного класу, з використанням оператора «.» або «->».

Отже, між термінами клас і об'єкт існує чітка межа: клас - це опис, об'єкт - те, що створене відповідно до цього опису. Щоб використовувати новий тип в програмі, потрібно, як мінімум, оголосити змінну цього типу. Змінні типу клас називають екземплярами класу або об'єктами.

Визначення класу містить оголошення даних, методів і специфікаторів доступу до них, і завжди закінчується крапкою з комою. На UML - уніфікованій мові моделювання - клас зображується у вигляді прямокутника, розділеного на три частини. У першій міститься ім'я класу, в другій - атрибути, в третій – методи.

Наприклад, об'єкт товар характеризується двома атрибутами: найменуванням і ціною. Необхідно ввести з клавіатури дані про товар і вивести їх на екран. Отже, для об'єкта товар можна оголосити клас, що буде містити два поля даних (найменування товару й ціну) і два методи (уведення даних з клавіатури і вивід даних на екран).

Зобразимо клас мовою UML наступним чином:

Товар
Найменування Ціна
Введення() Виведення()

Рисунок 1.1 – Зображення класу Товар мовою UML

Оголосимо клас CTovar:

```
class CTovar // CTovar- ім'я класу
{
//оголошення членів-даних класу
char name[15];      // найменування товару
float price;        // ціна товару
public:             // специфікатор доступу (відкритий)
//оголошення членів-функцій класу (методів)
void input();       //функція уведення з клавіатури даних про товар
void output();      //функція виводу на екран даних про товар
};
```

Далі необхідно визначити (описати) члени-функції. Визначення функцій виконаємо поза класом, для цього необхідно вказати до якого класу вона належить (у програмі може бути кілька класів). Тобто перед ім'ям функції необхідно вказати ім'я класу. Приклад опису членів-функцій поза класом наведений нижче:

```
void CTovar::input()      //оголошення функції input() класу
{                         // початок функції
cout <<"Уведіть найменування товару ";
cin >>name;
cout <<"Уведіть ціну товару ";
cin >>price;
}               // кінець функції

void CTovar::output()
{
cout <<"Найменування товару: " <<name<<endl;
cout <<"Ціна товару: "<<price<< endl;
}
```

Тому що клас є шаблоном для створення об'єктів (тобто екземплярів класу), на основі одного класу можна створити багато об'єктів. Приклад оголошення об'єкта наведений нижче:

```
CTovar Tovar; //оголошення об'єкта Tovar на основі класу CTovar
```

Виклик члена-функції класу поза функціями цього класу виконується із вказівкою ім'я об'єкта, після якого слідує крапка (.), а потім ім'я функції. Наприклад, виклик функції output() об'єкту Tovar запишеться в такий спосіб:

Tovar.output();

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №1

Використовуючи індивідуальне завдання до лабораторної роботи №1 (див. нижче) необхідно:

1) визначити, виходячи з наведених даних, об’єкт, його атрибути та методи;
2) зобразити клас графічно (мовою UML);
3) оголосити клас мовою C++;
4) визначити функції поза класом;
5) оголосити три об'єкти у функції main();
6) увести дані 3-х об'єктів;
7) вивести дані 3-х об'єктів.

ІНДИВІДУАЛЬНІ ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ № 1

Увести з клавіатури й вивести на екран наступні дані:

1. Прізвище працівника, рік народження, стать, спеціальність.
2. Найменування видання, індекс видання, тираж, ціна.
3. Шифр постачальника, найменування постачальника, розрахунковий рахунок, телефон.
4. Найменування ЕОМ, обсяг пам'яті, вартість, швидкодія.
5. Прізвище, ім'я, по батькові студента, рік народження, рік вступу в ЗВО.
6. Назва області, чисельність населення на території області, площа території, кількість ЗВО.
7. Назва фірми, що випускає ЕОМ, рік випуску, вартість.
8. Шифр кафедри, найменування кафедри, кількість співробітників.
9. Прізвище, ім'я, по батькові працівника, посада; стаж роботи.

10. Прізвище, ім'я, по батькові читача, номер читацького білета, найменування книги.
11. Номер деталі, найменування деталі, кількість деталей.
12. Прізвище, ім'я, по батькові робітника, табельний номер, розряд, заробітна плата.
13. Прізвище, ім'я, по батькові абонента, номер телефону, рік придбання телефону, адреса.
14. Код вкладника, прізвище, ім'я, по батькові вкладника, сума внеску.
15. Табельний номер робітника, прізвище, ім'я, по батькові, планова норма виробітку, фактична норма виробітку.
16. Прізвище, ім'я, по батькові студента, шифр групи, середній бал.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення класу та об’єкту у програмуванні.
2. Що може містити оголошення класу?
3. Як оголосити клас на мові С++?
4. Де може бути визначена (описана) функція класу?
5. Порівняйте виклик члена-функції класу у функціях класу та поза функціями цього класу.

ЛАБОРАТОРНА РОБОТА № 2

ТЕМА РОБОТИ: Програмування з використанням конструкторів з параметрами, деструкторів.
МЕТА РОБОТИ: Ознайомитися з видами конструкторів та їх призначенням. Набути практичних навичок оголошення конструкторів різних видів та деструкторів класу, визначення тіла конструкторів та деструкторів.
ТРИВАЛІСТЬ: 4 години.

МЕТОДИЧНІ ВКАЗІВКИ

Після створення об'єкту його атрибути можуть бути ініціалізовані (тобто атрибутам задані початкові значення) за допомогою конструктора. Конструктор - це спеціальна функція класу з тим же ім'ям, що і клас, який потім викликається автоматично при створенні екземпляра класу (об'єкту). Дані-члени класу не можуть набувати початкового значення у визначенні класу. Вони або повинні набути цього значення в конструкторі класу, або їх значення можна встановити пізніше, після створення об'єкту. Конструктори не можуть повертати які-небудь значення. Конструктори можна перенавантажувати, щоб забезпечити декілька початкових значень об'єктів класу, тобто клас може мати декілька конструкторів.

Хороший стиль програмування свідчить: завжди передбачайте конструктор для упевненості в тому, що об'єкт набув відповідного значення, що має сенс. Якщо для класу не визначено ніякого конструктора, компілятор автоматично вбудовує в програму конструктор без параметрів. Такий конструктор називається конструктором за замовчуванням. Такий конструктор не задає ніяких початкових значень (дані об'єкту мають випадкові значення, що залишилися в оперативній пам'яті від роботи попередніх програм, тобто «сміття» оперативної пам'яті). Якщо ж для нас важливо, якими значеннями ініціалізуються поля об'єкту, то нам слід явно визначити конструктор.

Наприклад, оголосимо клас круг з конструктором без параметрів:

```
class CKrug
{
int radius;
```

```
public:
CKrug (); // оголошення конструктора без параметрів
};
```

Визначення (опис) конструктора.

```
CKrug ::CKrug ()
{
radius=0; // тут можна задати будь-яке значення
}
```

Радіус круга ініціалізується константним значенням, в даному випадку цілим значенням 0. Об'єкти, які будуть створені за допомогою цього конструктора, завжди матимуть однакові значення (в даному випадку 0).

Зручно проводити ініціалізацію об'єкта різними значеннями. Такі значення можна передавати конструктору як аргументи. Отже, можна оголосити конструктор з параметрами. Додаємо в клас CKrug оголошення конструктора з параметром:

```
class CKrug
{
int radius;
public:
CKrug ();              // оголошення конструктора без параметрів
CKrug (int rad);    // оголошення конструктора з параметром
};
```

Визначення (опис) конструктора поза класом:

```
CKrug ::CKrug (int rad)
{
radius=rad;
}
```

При оголошенні об'єкту класу, що містить конструктор з параметрами між ім'ям об'єкту і крапкою з комою можна в дужках вказати список ініціалізації атрибутів (членів-даних). Ці значення передаються в конструктор класу, якими ініціалізують дані об'єкту цього класу. Наприклад:

```
CKrug Krug(25);
```

Після створення об'єкту Krug, радіус матиме значення 25.

Ми розглянули два способи ініціалізації об'єктів. Конструктор без аргументів може ініціалізувати атрибути об'єкту константними значеннями, а конструктор, що має хоч би один аргумент. може ініціалізувати атрибути значеннями, переданими йому як аргументи. Тепер ми розглянемо третій спосіб ініціалізації об'єкту, що використовує значення полів вже існуючого об'єкту. До вашого можливого здивування, для цього не потрібно самим створювати спеціальний конструктор, оскільки такий конструктор надається компілятором для кожного створюваного класу і називається конструктором копіювання за замовчуванням. Конструктор копіювання має єдиний аргумент, що є об'єктом того ж класу, що і конструктор.

Приклад використання конструктора копіювання демонструє наступний фрагмент:

CKrug Krug1(40);

CKrug Krug2(Krug1);

Об'єкт Krug1 ініціалізувався значенням 40 за допомогою конструктора з аргументом (параметром). Потім ми визначили ще один об'єкт з ім'ям Krug2, який ініціалізувався значенням об'єкту Krug1. Дія конструктора копіювання зводиться до копіювання атрибутів об'єкту Krug1 у відповідний атрибут об'єкту Krug2. Таким чином, якщо в класі оголошено декілька конструкторів, то відбувається виклик того конструктора, у якого кількість і типи аргументів відповідають кількості і типам, вказаним при створенні екземпляра класу (об'єкта).

Деструктор - це також спеціальна функція класу, що застосовується для звільнення пам'яті, яку займає об'єкт. Ім'я деструктора співпадає з ім'ям класу, але перед ним ставиться символ тильда (~). Деструктор класу викликається при знищенні об'єкту - наприклад, коли виконувана програма покидає область дії, в якій був створений об'єкт цього класу. Насправді деструктор сам не знищує об'єкт - він виконує підготовку завершення перед тим, як система звільняє область пам'яті, в якій зберігався об'єкт, щоб використовувати її для розміщення нових об'єктів.

Якщо деструктор явним чином не визначений, компілятор автоматично створює порожній деструктор. Описувати в класі деструктор явним чином потрібно у разі, коли об'єкт містить покажчики на пам'ять, що виділяється

динамічно - інакше при знищенні об'єкту пам'ять, на яку посилалися його поляпокажчики, не буде помічена як вільна.

Деструктор не приймає ніяких параметрів і не повертає ніяких значень.

Клас може мати тільки один деструктор - перевантаження деструктора не дозволяється.

Додамо в клас CKrug оголошення деструктора:

```
class CKrug
{
int radius;
public:
CKrug ();            // оголошення конструктора без параметрів
CKrug (int rad);     // оголошення конструктора з параметром
~CKrug ();           // оголошення деструктора
};
```

Визначення (опис) деструктора поза класом:

```
CKrug ::~CKrug ()
{
// звільнити динамічну пам'ять
}
```

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №2

Завдання 1. Обчислити й вивести на екран відстань від точки А и точки В до початку координат.

Порядок виконання лабораторної роботи:

1. Зобразити для точки клас мовою UML.
2. Оголосити клас для точки, що буде містити два поля даних: координату X та координату Y.
3. Оголосити в класі й описати поза класом конструктор без параметрів, що буде ініціалізувати координату X значенням 5, а координату Y - значенням 10.
4. Оголосити в класі й описати поза класом функцію обчислення відстані від точки до початку координат. Функція повинна повертати відстань.
5. У функції main() створити об'єкт - точку А.

6. Вивести на екран відстань від точки A до початку координат.
7. Оголосити в класі й описати поза класом конструктор з параметрами, що буде ініціалізувати координату X и координату Y значеннями, переданими йому як аргументи.
8. Створити об'єкт - точку B с координатами 3 і 4.
9. Вивести на екран відстань від точки B до початку координат.
10. Створити об'єкт - точку C с координатами, як у точки B, використовуючи конструктор копіювання.
11. Вивести на екран відстань від точки C до початку координат.
12. Оголосити в класі й описати поза класом конструктор з параметрам, що буде ініціалізувати тільки координату X значенням, переданим йому як аргумент.
13. Створити точку D і ініціалізувати її координату X значення 10 за допомогою конструктора оголошеного у п.12.
14. Вивести на екран відстань від точки D до початку координат.

Нижче наведений текст програми, у якій реалізовані п.п. 1-8.

```
#include <iostream.h>
#include <math.h>
#include<conio.h>
class CPoint // оголошення класу для точки
{
int x;  // координата x
int y;  // координата y
public:
CPoint(); //конструктор без параметрів
CPoint(int fx, int fy);
double DistanceKoord();
};
CPoint::CPoint()
{
x=5;
y=10;
}
```

```
CPoint::CPoint(int fx, int fy)
{
x=fx;
y=fy;
}
double CPoint::DistanceKoord()
{
return sqrt(pow(x,2)+pow(y,2));
}
void main()
{
clrscr();
CPoint PointA;
CPoint PointB(3,4);
cout<<"Відстань від точки А до початку координат:"
<<PointA.DistanceKoord()<<"\n";
system(“pause”);
}
```

Завдання 2. Розробити констуктори та метод виконання певної дії для класу зі свого варіанту лабораторної роботи №1. Продемонструвати використання розробленого класу та його методів.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення конструктору класу.
2. Для чого призначений конструктор класу?
3. Які види конструкторів ви знаєте?
4. Скільки конструкторів можна оголосити у класі?
5. Як у класі оголосити конструктор з параметрами?
6. Для чого призначений деструктор класу?
7. Скільки деструкторів можна оголосити у класі
8. Як викликати конструктор і деструктор?
9. Якщо клас має декілька конструкторів, то який з конструкторів викликається при створенні об’єкту цього класу?

ЛАБОРАТОРНА РОБОТА № 3

ТЕМА РОБОТИ: Доступ до членів класу.
МЕТА РОБОТИ: Ознайомитися зі специфікаторами доступу до членів класу. Набути практичних навичок використання специфікаторів доступу при оголошенні класу, а також у зміненні та читанні закритих даних з використанням відкритих функцій цього класу.
ТРИВАЛІСТЬ: 2 години.

МЕТОДИЧНІ ВКАЗІВКИ

Як було вказано в лабораторній роботі №1 при описі класу можна використовувати специфікатори доступу до членів класу. Специфікатори доступу - це спеціальні синтаксичні конструкції, що явно задають область видимості кожного члена класу. Специфікатори доступу до членів класу завжди закінчуються двокрапкою (:) і можуть з'являтися у визначенні класу багато раз і у будь-якому порядку. Існують наступні специфікатори доступу:

public - відкритий доступ;

private - закритий доступ;

protected - захищений доступ.

Хороший стиль програмування рекомендує використовувати при визначенні класу кожен специфікатор доступу до елементів тільки один раз, що робить програму яснішою і простій для читання. За замовчанням доступ до членів класу - private. Отже, в класі можуть бути оголошені відкриті (public), захищені (protected) і закриті (private) члени. Ці члени відрізняються способом доступу до ним. Доступ до відкритих членів класу мають будь-які функції в програмі, доступ до закритих членів класу мають тільки члени-функції даного класу і друзі класу.

Дані-члени класу звичайно робляться закритими, а функції-члени - відкритими. Глибокий сенс такого підходу полягає в тому, що дійсне представлення даних усередині класу не торкається клієнтів класу. Клієнти класу використовують клас, не знаючи внутрішніх деталей його реалізації. Якщо реалізація класу змінюється, інтерфейс класу залишається незмінним і

дійсний код клієнта класу не вимагає змін. Це значно спрощує модифікацію систем.

З того, що дані класу закриті, не вивипливає, що клієнти не можуть змінювати ці дані. Дані можуть бути змінені функціями-членами або друзями цього класу. Наприклад, щоб дозволити клієнтам класу прочитати закрите значення даних, клас може мати функцію «одержати» (get). Щоб дати клієнтам можливість змінювати закриті дані, клас може мати функцію «встановити» (set) Таким чином, доступ до закритих членів класу здійснюється через відкриті члени-функції даного класу.

Основне завдання відкритих членів полягає в тому, щоб дати клієнтам класу уявлення про можливості (послуги), які забезпечує клас. Цей набір послуг складає відкритий інтерфейс класу. Клієнтів класу не повинно торкатися, яким чином клас виконує їх завдання. Закриті члени класу і опис відкритих функцій-членів недоступні для клієнтів класу. Ці компоненти складають реалізацію класу.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №3

Завдання 1. Обчислити й вивести на екран відстань між двома точками, а також вивести на екран координати цих точок. Вивід на екран координат виконувати у такій спосіб: ім’я точки (значення X, значення Y). Наприклад: B(3,4).

Порядок виконання лабораторної роботи:

1. Оголосити в класі й визначити (описати) поза класом функцію, яка дає можливість одержати значення координати X наступним чином:

Оголошення функції:

```
int GetX();
```

Визначення функції GetX() поза класом:

```
int CPoint::GetX()
{
return x;
}
```

2. Оголосити в класі й визначити поза класом функцію, яка дає можливість одержати значення координати Y.

3. Вивести на екран координати точки A і точки B.
4. Вивести на екран координати точки C і точки D.
5. Оголосити в класі й описати поза класом функції, які дають можливість встановити інші значення координати X и координати Y.
6. Змінити координату Y точки D на 5.
7. Вивести на екран координати точки D.
8. Змінити координати точки B на будь-які інші.
9. Вивести на екран координати точки B;
10. Вивести на екран відстань від точки B до початку координат.
11. Оголосити в класі й описати поза класом функцію обчислення відстані між двома точками. Функція повинна повертати відстань, а також у функцію передавати 1 параметр (точку до якої треба обчислити відстань).
12. Вивести на екран відстань між точками A і B.
13. Вивести на екран відстань між точками A і C.
14. Вивести на екран відстань між точками B і D.
15. Вивести на екран відстань між точками C і D.

Завдання 2. Для індивідуального варіанту з попередніх лабораторних робіт сформулювати самостійно та виконати аналогічні дії (до Завдання 1).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Назвіть специфікатори доступу до членів класу.
2. Чи може з'являтися у визначенні класу один і той же специфікатор доступу багато разів?
3. У якому порядку можуть з'являтися специфікатори доступу до членів класу?
4. Який рівень доступу за замовчуванням для члена-класу?
5. Які функції в програмі мають доступ до відкритих членів класу?
6. Які функції в програмі мають доступ до закритих членів класу?
7. Як здійснюється доступ до закритих членів класу функціями, які не є членами цього класу?

ЛАБОРАТОРНА РОБОТА № 4

ТЕМА РОБОТИ: Одиночне спадкування. Поліморфізм.
МЕТА РОБОТИ: Ознайомитися з видами спадкування та його перевагами. Набути практичних навичок створювання нових класів шляхом спадкування існуючих класів, використання у програмі мовою C++ одиночного спадкування та поліморфізму.
ТРИВАЛІСТЬ: 4 години.

МЕТОДИЧНІ ВКАЗІВКИ

Спадкування - це спосіб повторного використання програмного забезпечення, при якому нові класи створюються із уже існуючих класів шляхом запозичення їхніх атрибутів і функцій і збагачення цими можливостями нових класів. Повторне використання кодів заощаджує час при розробці програм. Від будь-якого класу можна породити один або кілька підкласів, які називаються похідними класами, у результаті чого сформується ієрархія класів.

Батьківські класи звичайно містять методи більше загального характеру, тоді як вирішення специфічних завдань поручається похідним класам. При простому (одиночному) спадкуванні клас породжується одним базовим класом. При множинному спадкуванні похідний клас породжується декількома базовими класами. Похідні класи успадковують члени-дані й члени-функції базового класу. Спадкування в UML називають узагальнення і на діаграмі класів його зображають у вигляді стрілки. (Діаграма класів описує структуру системи, показуючи її класи, їх атрибути і методи, а також відносини (зв'язки), які існують між ними.)

Похідний клас не може мати доступ до закритих елементів свого базового класу (дозвіл такого доступу з'явилося б порушенням інкапсуляції базового класу), але має доступ до відкритих і захищених елементів свого базового класу.

Похідний клас може мати доступ до закритих елементів свого базового класу тільки за допомогою функцій доступу, передбачених у відкритому інтерфейсі базового класу.

Спадкування може бути відкрите, закрите й захищене.

Наприклад, клас kart оголошується похідним від класу iskus, спадкування - відкрите:

class kart : public iskus

При відкритому спадкуванні відкриті й захищені елементи (члени) базового класу успадковуються як відкриті й захищені елементи (члени) похідного класу відповідно.

Для похідного класу можна визначати методи (функції), що мають такі ж імена, як і у методів базового класу. В цьому випадку має місце перевантаження функцій.

Поліморфізм - можливість для об'єктів різних класів, пов'язаних з допомогою спадкування, реагувати різним образом при звертанні до однієї й тієї ж функції-елементу. Поліморфізм реалізується за допомогою віртуальних функцій. Якщо при використанні віртуальної функції виклик здійснюється за допомогою покажчика базового класу (або посилання), то C++ вибирає правильну перевизначену функцію у відповідному похідному класі, пов'язаному з даним об'єктом.

Якщо функція-елемент визначений у базовому класі не як віртуальна, але перевизначена в похідному класі, і викликається через покажчик базового класу, то використовується функція базового класу. Якщо ж ця функція-елемент викликається через покажчик похідного класу, то використовується функція похідного класу. Це не поліморфне поводження. Функція оголошується віртуальною за допомогою ключового слова virtual, що передує прототипу функції в базовому класі. Наприклад, virtual void ekran(); Якщо функція один раз оголошена віртуальною, то вона залишається віртуальною на кожному більше низькому рівні ієрархічної структури.

Якщо в похідному класі вирішено не описувати віртуальну функцію, то похідний клас безпосередньо успадковує опис віртуальної функції з базового класу.

Приклад виклику функції ekran() через покажчик на базовий клас:

```
iskus *pIskus; // оголошення покажчика pIskus на базовий клас iskus
pIskus =&pict; // покажчику привласнюється адреса об'єкта pict
pIskus->ekran(); // виклик функції ekran() через покажчик на клас
```

Розглянемо приклад програми з використанням одиночного спадкування й поліморфізму.

Завдання для прикладу. Вивести дані про наступні твори мистецтва: картину, музичний твір, музичний твірі для оркестру.

Дані про картину: автор, назва, дата, висота, ширина.

Дані про музичний твір: автор, назва, дата, тональність.

Дані про музичний твір для оркестру: автор, назва, дата, тональність, кількість музикантів.

Діаграма класів наведена на рисунку 4.1. Клас «Твори мистецтва» є базовим класом, клас «Картина» та клас «Музичний твір» є похідними від класу «Твори мистецтва», клас «Музичний твір для оркестру» є похідним від класу «Музичний твір».

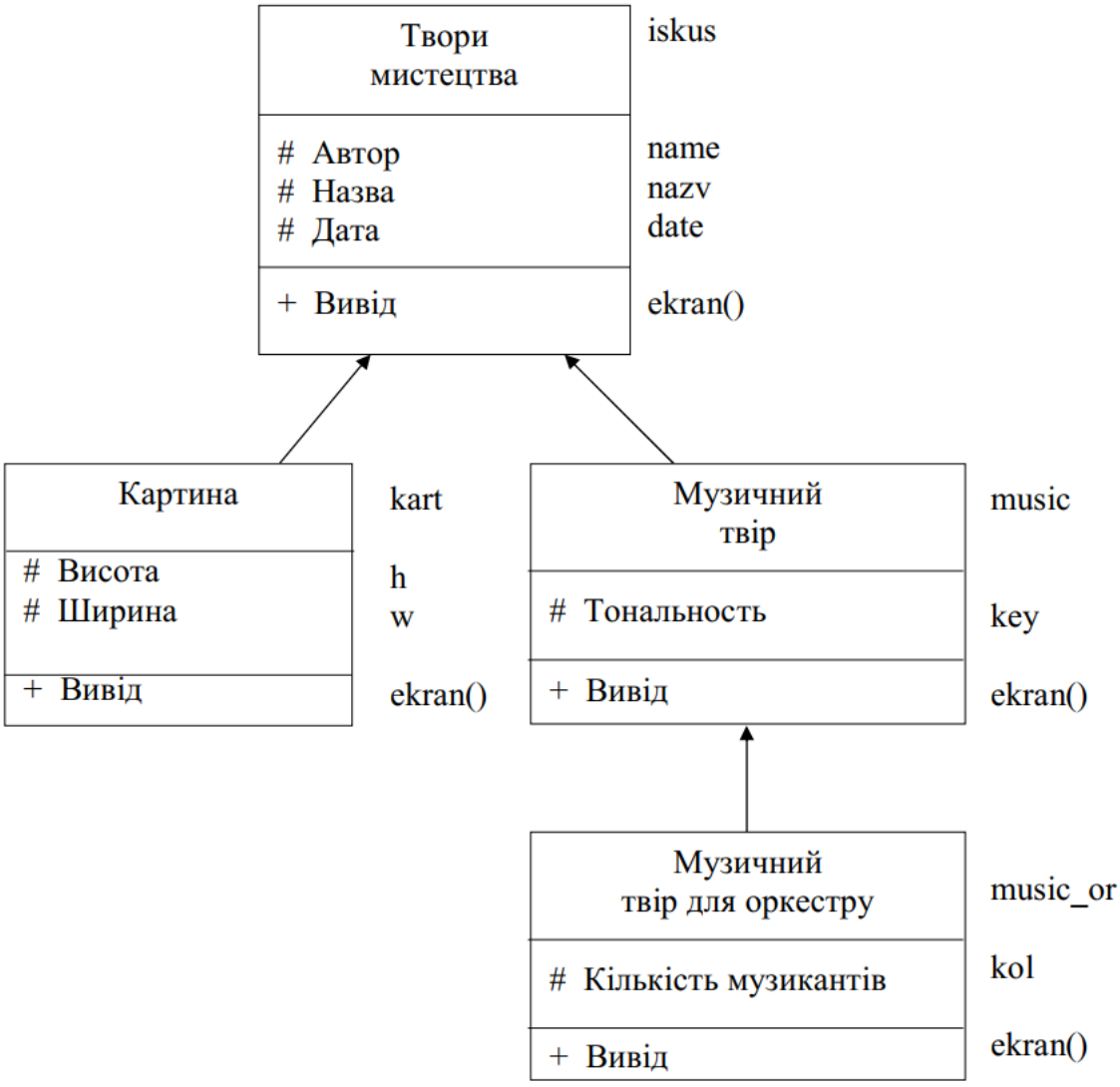


Рисунок 4.1 - Діаграма класів

Приклад програми з використанням спадкування й поліморфізму представлений нижче.

```
#include<stdio.h>
#include<string.h>
#include<iostream.h>
#include<conio.h>
class iskus
{
public:
iskus (char*, char*, char*);
virtual void ekran();
protected:
char name[50], nazv[60], date[50];
};
class kart : public iskus   //похідний клас – картина
{
public:
kart(char*, char*, char*, int, int);
void ekran();
protected:
int w, h;
};
class music : public iskus          //похідний клас - музика
{
public:
music (char*, char*, char*, char*);
void ekran();
protected:
char key[40];
};
class music_or : public music //похідний клас – музика для оркестру
{
public:
music_or (char*, char*, char*, char*, int);
```

```
void ekran();
protected :
int kol;
};
//опис конструктора
iskus :: iskus (char *who, char *what, char *when)
{
strcpy(name,who);
strcpy(nazv,what);
strcpy(date,when);
}
//опис віртуальної функції ekran()
void iskus :: ekran()
{
cout<<"\n\n Автор: "<<name;
cout<<"\n Назва : "<<nazv;
cout<<"\n Дата : "<<date;
};
kart :: kart (char *who, char *what, char *when, int v, int sh):
iskus(who, what, when)
{
w=sh;
h=v;
}
void kart :: ekran()
{
iskus::ekran();
cout<<"\n Розміри - Висота: "<<h <<"Ширина:"<<w;
}
music :: music(char *who, char *what, char *when, char *ky):
iskus(who, what, when)
{
strcpy(key,ky);
}
```

```
void music :: ekran()
{
iskus :: ekran();
cout<<"\n Тональність:"<<key;
}
music_or :: music_or (char *who, char *what, char *when, char *ky, int kl): music(who, what, when, ky)
{
kol=kl;
}
void music_or :: ekran()
{
music :: ekran();
cout<<"\n Кількість музикантів "<<kol;
}
void fn (iskus &isk)
{
isk.ekran();
}
 //Головна функція
int main()
{
clrscr();
kart pict ("Леонардо да Вінчі", "Мона Ліза”, "1503", 24, 36);
music symphon ("Людвіг Ван Бетховен ", "Дев'ята симфонія", "1824", "ля-мінор");
music_or mus_or ("Вольфганг Амадей Моцарт","Хофмейстер", "1786", "до-мажор",4);
fn(pict);
fn(symphon);
fn(mus_or);
getch();
return 0;
}
```

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №4

Є наступні дані по поточному банківському рахунку: номер рахунку, прізвище вкладника, первісний внесок, сума приходу, сума витрат. Обчислити залишок на рахунку й вивести на екран номер рахунку, прізвище вкладника, первісний внесок, сума приходу, сума витрат, залишок на рахунку. Залишок на рахунку = первісний внесок + сума приходу - сума витрат. Є наступні дані по депозитному банківському рахунку: номер рахунку, прізвище вкладника, первісний внесок, процентна ставка, строк внеску (у місяцях). Обчислити суму відсотків і вивести на екран номер рахунку, прізвище вкладника, первісний внесок, процентну ставку, строк внеску (у місяцях) і суму відсотків.

Сума відсотків = первісний внесок * процентна ставка/365 * строк внеску * 30.

Порядок виконання роботи:

1. Вивчіть діаграму класів і текст програми, що представлений вище, і введіть його.
2. Запустіть програму на виконання й попрацюйте з нею.
3. Видаліть ключове слово virtual, що оголошує функцію ekran() віртуальною. Запустіть програму на виконання й поясніть, чому вона працює по-іншому.
4. Розробити діаграму класів для завдання про банківські рахунки.
5. Оголосіть класи й члени-дані класу.
6. У кожному класі оголосити конструктор з параметрами.
7. Написати текст конструкторів класів.
8. Оголосити й визначити функції розрахунку.
9. Оголосити й визначити функції виводу на екран.
10. Створити об'єкти й виконати ініціалізацію їх довільними даними, використовуючи конструктори з параметрами.
11. Передбачити у функції main() вивід меню на екран, що дозволяє вибрати наступні дії: вивід відомостей по поточному рахунку, вивід відомостей по депозитному рахунку й вихід із програми.
12. Оголосити й визначити функції уведення даних із клавіатури по кожному рахунку.

13. Змінити меню таким чином, щоб була можливість вибору уведення даних по поточному й депозитному рахунку.
14. Змінити програму таким чином, щоб вона підтримувала поліморфізм.

Завдання 2. Для індивідуального варіанту з попередніх лабораторних робіт сформулювати самостійно та виконати аналогічні дії до Завдання 1 (кроки 4-14).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. У чому полягає суть спадкування?
2. У чому переваги використання спадкування?
3. Які ви знаєте види спадкування?
4. Як оголосити похідний клас та конструктор похідного класу?
5. Як на діаграмі класів позначається спадкування?
6. До всіх елементів свого базового класу може мати доступ похідний клас?
7. За допомогою чого реалізується поліморфізм?
8. Що необхідно для реалізації поліморфізму в програмі?
9. Які знаки використовують для того, щоб на діаграмі класів показати захищені атрибути і функції?

ЛАБОРАТОРНА РОБОТА № 5

ТЕМА РОБОТИ: Абстрактні класи.

МЕТА РОБОТИ: Ознайомитися з визначенням абстрактних класів. Набути практичних навичок оголошення абстрактних класів, та використання їх у програмі.

ТРИВАЛІСТЬ: 2 години.

МЕТОДИЧНІ ВКАЗІВКИ

Клас є абстрактним, якщо він має хоча б одну чисту віртуальну функцію. Чистою віртуальною функцією є така функція, у якої в її оголошенні тіло визначене як 0. Наприклад:

virtual float Func()=0;

Абстрактні класи призначені для представлення загальних понять, які передбачається конкретизувати в похідних класах. Абстрактний клас може використовуватися тільки як базовий для інших класів - об'єкти абстрактного класу створювати не можна, оскільки прямий або непрямий виклик чисто віртуального методу приводить до помилки при виконанні.

Якщо клас, похідний від абстрактного класу, не визначає всі чисто віртуальні функції, він також є абстрактним.

Створювати функцію, параметром якої є покажчик на абстрактний клас, можна. На місце цього параметра при виконанні програми може передаватися покажчик на об'єкт будь-якого похідного класу. Це дозволяє створювати поліморфні функції, що працюють з об'єктом будь-якого типа в межах однієї ієрархії.

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №5

Обчислити об’єм кулі, циліндра й паралелепіпеда й вивести на екран найменування й об’єм фігури.

Порядок виконання роботи

1. Розробити діаграму класів.
2. Оголосити класи (обов'язково повинен бути абстрактний клас).

3. У кожному класі оголосити конструктор з параметрами.
4. Написати текст конструкторів класів.
5. Оголосити й визначити функції розрахунку об’єм кожної фігури.
6. Оголосити й визначити функції виводу на екран.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення абстрактного класу.
2. Чи можна створювати об’єкти абстрактного класу?
3. Виявіть відмінності між віртуальними функціями і чистими віртуальними функціями.
4. Чи вірно, що всі віртуальні функції в абстрактному базовому класі повинні бути оголошені як чисті віртуальні функції?
5. Чи можна створювати функцію, параметром якої є покажчик на абстрактний клас?

ЛАБОРАТОРНА РОБОТА № 6

ТЕМА РОБОТИ: Реалізація в С++ агрегації й асоціації. Робота з файловими потоками вводу-виводу в С++.
МЕТА РОБОТИ: Ознайомитися з відношеннями між класами: агрегацією та асоціацією, та зображення їх на діаграмах класів. Набути практичних навичок реалізації в С++ агрегації й асоціації.
ТРИВАЛІСТЬ: 6 години.

МЕТОДИЧНІ ВКАЗІВКИ

У лабораторній роботі №4 було вивчено спадкування, яке підтримує між класами відношення узагальнення, тобто «є» (депозитній рахунок є банківським рахунком). Між класами може бути встановлено також відношення «містить», при якому клас може містити інші класи як елементи, таке відношення називається агрегацією, тобто відношення між цілим і його частинами. Графічно агрегація представляється порожнім ромбом біля класу, що представляє ціле, і лінією, що йде від цього ромба до класу, що міститься в ньому й представляє частину. Приклад агрегації: автомобіль и двигун.

Якщо два класи просто зв'язано один з одним і ніякі інші типи зв'язків (узагальнення та агрегація) тут застосувати не можна, то ці класи мають відношення асоціації. Тобто один об'єкт може «*знати»* інший об'єкт, наприклад, постачальник і товар. Для того, щоб один об'єкт «*знав»* інший об'єкт достатньо, щоб об'єкт містив покажчик або посилання на інший об'єкт. Є і інші способи використання послуг класів. Наприклад, яким відношенням зв'язані об'єкти людина і автомобіль? Об'єкт людина не *є* автомобілем і не *містить* автомобіль, але об'єкт людина *використовує* автомобіль. Відношення «використовує» об'єкт здійснюється просто шляхом звернення до функції-елементу цього об'єкту.

Відношення асоціації позначається на діаграмі класів лінією.

Для форматування виводу на екран використайте маніпулятори потоку:

setw(int) – встановлює максимальну ширину поля виводу, наприклад setw(6);

setiosflags(long) – встановлює ознаки стану потоку, наприклад, setiosflags(ios::right) – значення змінних притиснуті до правого краю поля,

setiosflags(ios::left) – значення змінних притиснуті до лівого краю;

setprecision(int) – встановлює кількість цифр після коми для нецілих чисел, наприклад, setprecision(3).

Для використання маніпуляторів потоку додайте наступне:

#include <iomanip.h>

Приклад форматування виводу на екран з використанням маніпуляторів потоку наведено нижче.

```
float price;
char name[10];
cout<< setiosflags(ios::left)<<setw(15)<<name<< setiosflags(ios::right) << setprecision(2)<<setw(6)<< price<<endl;
```

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №6

Визначити економію металу при прокатці його на мінусових допусках. Стан холодної прокатки катає метал у рулонах на необхідну товщину, а ножиці ріжуть рулон на листи, необхідних розмірів. Листовий метал сортується в пачки, зважується, упаковується й перевозиться на склад для відвантаження. На партію металу заповнюється паспорт. У паспорт заносяться наступні дані: дата, номер зміни, номер партії, номер плавки, марка стали, кількість пачок у партії, кількість листів у пачці, розміри листа (ширина, довжина, товщина) і фактична вага (маса) партії, найменування замовника. Про замовника є наступні дані: найменування підприємства, телефон і адреса.

Метал катають на мінусових допусках, прийнятих у стандартах. Така технологія дає економію металу. Визначення економії може здійснюватися шляхом визначення ваги (маси) металу партії, як при фізичному зважуванні, так і визначенні теоретичної ваги (маси) металу. Різниця між теоретичною й фактичною вагою й дає економію від впровадженої технології прокатки. Теоретична вага (маса) металу визначається за формулою:

TVes =m *KL*Kp,

де: KL-Кількість листів у пачці;

KP- Кількість пачок у партії металу;

m - маса листа, що обчислюється за формулою:

m=V*7850,

де: 7850 – щільність стали в кг/м3,

V - обсяг листа в м3, що розраховується за формулою:

V=S*1.0e-3* t,

де: t - товщина листа;

1.0e-3 – коефіцієнт для переводу товщини, що задається в мм, у м;

S - площа листа в м2, що обчислюється за формулою:

S= 1.0e-6 * a* l,

де: 1.0e-6 – коефіцієнт для переводу площі листа із мм^2, у м^2,

a - ширина листа;

l - довжина листа.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Розгляньте діаграму класів (див. рисунок 6.1).
2. Оголосити клас «Лист».

```
class CList
{
public:
CList( double m_a=0,double m_t=0, double m_l=0 );
~CList() {};
double Area(void);
double Volume(void);
double Massa();
double GetA(){ return a; }
double GetL(){ return l; }
double GetT(){ return t; }
void SetA(double al) { a= al;}
void SetT(double tl) { t= tl;}
void SetL(double ll) { l= ll;}
private:
double a;
double t;
```

```
double l;
double p;
};
```

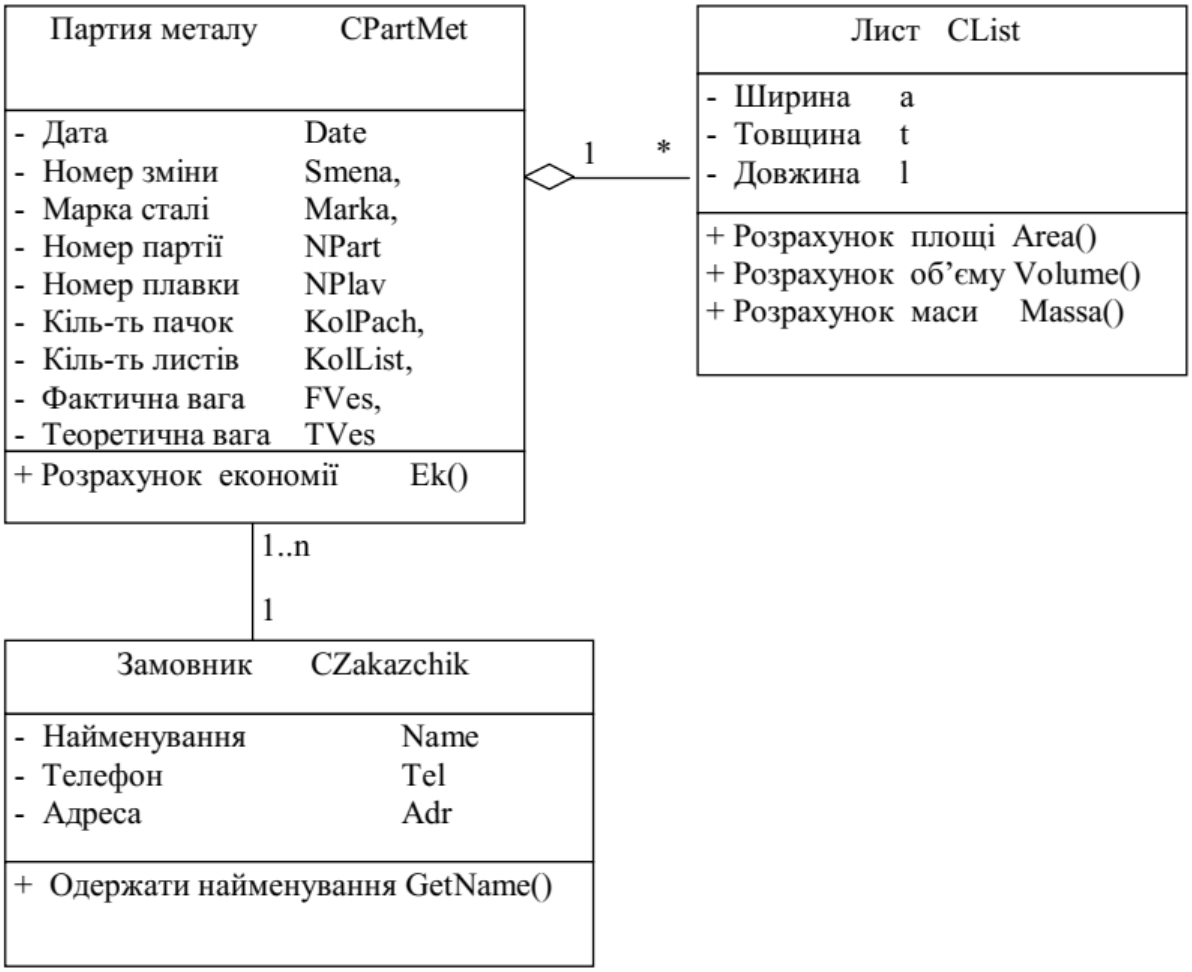


Рисунок 6.1 – Діаграма класів

3. Визначити конструктор класу «Лист».

```
CList::CList(double m_a,double m_t, double m_l)
{
a=m_a;
t=m_t;
l=m_l;
p=7850;
}
```

4. Визначити функції класу «Лист», які розраховують площу, об'єм та масу листа.

```
double CList::Area(void)
```

```
{
return 1.0 e-6*a*l;
}
double CList::Volume(void)
{
return Area()*1.0 e-3*t;
}
double CList::Massa()
{
return p*Volume();
}
```

5. Оголосити клас «Замовник».

```
class CZakazchik
{
char Name[15];
char Tel[10];
char *Adr;
public:
CZakazchik(char *nm="", char *tl="",char *ad="");
char* GetName();
};
```

6. Визначити конструктор класу «Замовник».

```
CZakazchik::CZakazchik(char *nm, char *tl,char *ad)
{
strcpy(Name,nm);
strcpy(Tel,tl);
strcpy(Adr,ad);
}
```

7. Визначити функцію класу «Замовник», яка буде повертати назву замовника.

```
char* CZakazchik::GetName()
```

```
{
return Name;
}
```

8. Оголосити клас «Партія металу».

```
class CpartMet
{
public:
CPartMet(char*Dat,int Sm,int Part,int Plav,char* Mrka, int Klpach, int Kllist,double Ves,double Tvs,CZakazchik &Zk);
~CPartMet() {};
void SetTVes(double) {TVes=List.Massa()*KolList*KolPach;}
double GetTVes(){ return TVes; }
double GetFVes(){ return FVes; }
int GKolList() { return KolList; }
int GKolPach() { return KolPach; }
int GSmena() { return Smena; }
char* GData() { return Data; }
int GNPart() { return NPart; }
int GNPlav() { return NPlav; }
char* GMarka() { return Marka; }
double Ek();
CList List;              // реалізація агрегації
CZakazchik *pZak;    // реалізація асоціації
private:
char Data[9];
int Smena;
int NPart;
int NPlav;
char Marka[6];
int KolPach;
int KolList;
double FVes;
double TVes;
```

```
};
```

9. Визначити конструктор класу «Партія металу».

```
CPartMet::CPartMet(char*Dat,int Sm,int Part,int Plav,char* Mrka, int Klpach,int Kllist,double Ves,double Tvs,CZakazchik &Zk)
{
strcpy(Data,Dat);
Smena=Sm;
NPart=Part;
NPlav=Plav;
strcpy(Marka,Mrka);
KolPach=Klpach;
KolList=Kllist;
FVes=Ves;
TVes=Tvs;
pZak=&Zk;
}
```

10. Визначити функції класу «Партія металу».

```
double CPartMet::Ek()
{
return TVes - FVes;
}
```

11. Визначити функцію main().

```
void main ()
{
}
```

12. Оголосити у функції main() екземпляри класів CZakazchik і CPartMet і виконати ініціалізацію них наступними даними:
Замовник : ООО Орбита, 770-89-22, ін.Кірова,23;

Партія металу: дата - 11.12.06, номер зміни - 1, номер партії -3, номер плавки - 12, марка стали - 10Х45, кількість пачок - 5, кількість листів - 10, фактична вага - 1900, теоретична вага - 0.

13. Задати наступні розміри листа довжина - 1000 мм, ширина - 500 мм, товщина - 10 . Використовувати для цього функції SetL(),SetA(),SetT().
14. Обчислити теоретичну вагу партії.
15. Вивести у функції main() дані про партію металу в наступному вигляді:

Номер партії	Номер плавки	Марка сталі	Кіль-ть пачок	Кіль-ть листів	Розміри, мм			Вага, кг		Економія, кг	Замов-ник
					довж	ширина	товщ	факт	теор		

Рисунок 6.2 – Таблиця виводу результату

Для форматування виводу на екран використайте маніпулятори потоку.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які відношення можуть бути встановлено між класами?
2. Яким чином зображуються асоціації на діаграмі класів?
3. Яким чином на діаграмі класів зображується агрегація?
4. Які відношення між класами «Партія металу» і «Лист» та «Партія металу» і «Замовник»?
5. Як реалізувати в С++ агрегацію й асоціацію?
6. Назвіть маніпулятори потоку, які використовували у даній лабораторній роботі, і їх призначення.

ЛАБОРАТОРНА РОБОТА № 7

ТЕМА РОБОТИ: Робота з файлами.
МЕТА РОБОТИ: Ознайомитися з видами потоків, з класами для створення файлових потоків. Набути практичних навичок створення файлових потоків, читання даних з файлу та запису у файл.
ТРИВАЛІСТЬ: 2 години.

МЕТОДИЧНІ ВКАЗІВКИ

Потік - це абстрактне поняття, що відноситься до будь-якого перенесення даних від джерела до приймача. По виду пристроїв, з якими працює потік, можна розділити потоки на стандартні, файлові і строкові.

Стандартні потоки призначені для передачі даних від клавіатури і на екран дисплея. Файлові потоки - для обміну інформацією з файлами на зовнішніх носіях даних (наприклад, дисках), а строкові потоки - для роботи з масивами символів в оперативній пам'яті.

Стандартна бібліотека містить три класи для роботи з файлами:

ifstream - клас вхідних файлових потоків;

ofstream - - клас вихідних файлових потоків;

fstream - клас двонаправлених файлових потоків.

Використання файлів в програмі припускає наступні операції:

створення потоку;

відкриття потоку і скріплення його з файлом;

обмін (введення/висновок);

знищення потоку;

закриття файлу.

Для використання файлів у C++ повинно включити заголовний файл <fstream>. Файл <fstream> включає визначення класів для створення файлових потоків: ifstream (для читання з файлу), ofstream (для запису у файл) і fstream (для читання і запису у файл).

Ці класи потоків є похідними (тобто успадковують функціональні можливості) відповідно від класів ostream,. iostream, istream. Ієрархія класів вводу/виводу наведена на рисунку 7.1.

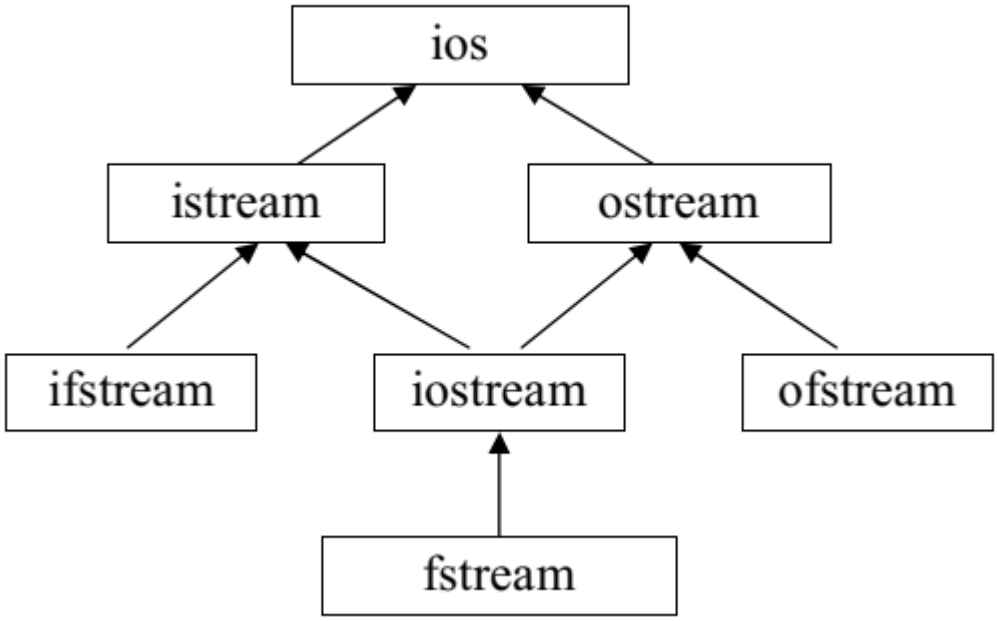


Рисунок 7.1 – Ієрархія класів вводу/виводу

Кожен клас файлових потоків містить конструктори, за допомогою яких можна створювати об'єкти цих класів різними способами.

Конструктори без параметрів створюють об'єкт відповідного класу. Не пов'язуючи його з файлом:

ifstream();

ofstream();

fstream().

Конструктори з параметрами створюють об'єкт відповідного класу, відкривають файл з вказаним ім'ям (*name) і пов'язують файл з вказаним об'єктом:

ifstream(const char *name, int mode = ios::in);

ofstream(const char *name, int mode = ios::out | ios::trunc);

fstream(const char *name, int mode = ios::in | ios::out).

Другим параметром конструктора є режим відкриття файлу. Якщо встановлене за замовчанням значення не влаштовує програміста, можна вказати інше (див. таблицю 7.1).

Таблиця 7.1 - Режими відкриття файлів

Режим	Опис
ios::app	Відкрити файл для додавання даних у кінець файлу
ios::ate	Перемістити покажчик в кінець файлу.
ios::in	Відкрити файл для читання (вводу).
ios::out	Відкрити файл для запису (виводу)
ios::trunc	Якщо файл існує, видалити (це також за замовчуванням робиться для ios::out).
ios::nocreate	Якщо файл не існує, то видати помилку.
ios::noreplace	Якщо файл існує, то видати помилку.
ios::binary	Відкрити файл у двійковому режимі

Відкрити файл в програмі можна з використанням або конструкторів, або методу open(), що має такі ж параметри, як і у відповідному конструкторі. Для закриття потоку визначений метод close(), але оскільки він неявно виконується деструктором. Явний виклик необхідний тільки тоді, коли потрібно закрити потік раніше кінця його області видимості.

Приклади відкриття файлу Tovar.dat для запису:

1) ofstream f(("Tovar.dat"); // оголошення файлового потоку виводу й відкриття файлу Tovar.dat для запису
2) ofstream f; // оголошення файлового потоку виводу
   open ("Tovar.dat"); // відкриття файлу Tovar.dat
3) fstream f ("Tovar.dat", ios::out); // оголошення файлового потоку вводу/виводу й відкриття для запису
4) fstream f; // оголошення файлового потоку вводу/виводу f.open ("Tovar.dat", ios::out); // відкриття файлу для запису
5) fstream f ("Tovar.dat", ios::app); // оголошення файлового потоку вводу/виводу й відкриття файлу Tovar.dat для додавання

   Приклади відкриття файлу Tovar.dat для читання:

   1) ifstream f(("Tovar.dat"); // оголошення файлового потоку вводу й відкриття файлу Tovar.dat для читання
   2) ifstream f; // оголошення файлового потоку вводу
      f.open ("Tovar.dat"); // відкриття файлу Tovar.dat

3) fstream f ("Tovar.dat", ios::in); // оголошення файлового потоку вводу/виводу й відкриття для читання

4) fstream f; // оголошення файлового потоку вводу/виводу

f.open ("Tovar.dat", ios::in); // відкриття файлу для читання

Приклад запису у файл із використанням функції write (використаний клас CTovar із прикладу з лабораторної роботи №1):

```
CTovar Tovar; //Оголошення об'єкта
ofstream fTovOut ("Tovar.dat", ios::app); // оголошення файлового потоку виводу fTovar і відкриття файлу для додавання
Tovar.input();//виклик функції input() об'єкта Tovar
fTovOut.write((char*)&Tovar,sizeof(Tovar)); // запис у файл даних об'єкта Tovar
fTovOut.close();// закриття файлу
```

Приклад запису у файл із використанням операції << (помістити в потік):

```
void CTovar::input()
{
ofstream fTovOut ("Tovar.dat", ios::app);
cout <<"Уведіть найменування товару ";
cin >>name;
cout <<"Уведіть ціну товару ";
cin >>price;
fTovOut <<name <<' '<< price<<endl; // запис у файл Tovar.dat
fTovOut.close();
}
```

Для читання з файлу можна використовувати функцію read() класу ifstream і операцію >> (взяти з потоку).

Приклад читання з файлу всіх даних з використанням функції read():

ifstream fTovIn("Tovar.dat"); // оголошення файлового потоку уведення fTovar і відкриття файлу для додавання

fTovIn.read ((char*)&Tovar,sizeof(Tovar)); // читання з файлу даних в об'єкт Tovar

```
while (fTovIn.Eof()) // поки не кінець файлу
{
Tovar.output();
fTovIn.read ((char*)&Tovar,sizeof(Tovar));
}
```

Приклад читання з файлу всіх даних з використанням операції >> (взяти з потоку) і вивід їх на екран:

```
void CTovar::output()
{
ifstream fTovIn("Tovar.dat");
cout <<"Найменування товару "<<" Ціна товару "<<endl ;
while (fTovIn>>name>>price) //поки є дані читати з файлу
{
cout<<" "<<setiosflags(ios::left)<<setw(15) <<name;
cout.precision(2); // із плаваючою крапкою, 2 знаки після коми
cout<<setiosflags(ios::right)<<setw(10)<<price<<"\n";}
}
```

setiosflags(ios::left) - вирівнювати по лівому краї поля

setiosflags(ios::right) - вирівнювати по правому краї поля

setw(15) - установити ширину поля, рівним 15 символам;

ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ №7

Змінити програму, розроблену в лабораторній роботі №4 таким чином, щоб уведені із клавіатури дані по поточних і депозитних рахунках записувалися у файл. По депозитних рахунках вивести звіт, форма якого представлена на рисунку 7.2, по поточних рахунках вивести звіт, форма якого представлена на рисунку 7.3.

Звіт

по депозитних рахунках

строк _____ місяців

Номер рахунку	Первісний внесок	Сума відсотків
Разом:		∑

Рисунок 7.2 - Форма звіту по депозитних рахунках

Звіт

по поточних рахунках

Номер рахунку	Залишок на рахунку
Разом:	∑

Рисунок 7.3 - Форма звіту по поточних рахунках

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які види потоків ви знаєте?
2. Назвіть класи для створення файлових потоків і їх призначення.
3. Які дії необхідно виконати для використання файлів у програмі?
4. Які існують режими відкриття файлів?
5. Для чого призначений метод open() і які він має параметри?
6. Які способи відкриття та закриття файлу ви знаєте?
7. Як записати дані у файл?
8. Як прочитати дані з файлу?

ЛАБОРАТОРНА РОБОТА № 8

ТЕМА РОБОТИ: Шаблони функцій і класів. Параметризовані контейнерні класи бібліотеки STL.

МЕТА РОБОТИ: Ознайомитися із базовими механізмами використання шаблонів функцій. Навчитись створювати та використовувати шаблонів класів. Засвоїти правила створення та використання параметризованих контейнерних класів. Дослідити основні класи з бібліотеки STL.

ТРИВАЛІСТЬ: 8 години.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Шаблони функцій.

Шаблони функцій – це потужний інструмент у C++, який суттєво спрощує роботу програміста. Наприклад, нам потрібно запрограмувати функцію, яка виводила б на екран елементи масиву. При цьому ми хочемо, щоб функція виводила масиви типу *int, double, float* і *char*. Тобто, нам потрібно запрограмувати 4 функції, які виконують одні й ті самі дії, але для різних типів даних. Скористаємося перевантаженням функцій.

```
void printArray(const int * array, int count)
{   for (int ix = 0; ix < count; ix++)
       cout << array[ix] << "   ";
   cout << endl;
}
void printArray(const double * array, int count)
{   for (int ix = 0; ix < count; ix++)
       cout << array[ix] << "   ";
   cout << endl;
}
void printArray(const float * array, int count)
{   for (int ix = 0; ix < count; ix++)
       cout << array[ix] << "   ";
   cout << endl;
```

```
}
void printArray(const char * array, int count)
{   for (int ix = 0; ix < count; ix++)
       cout << array[ix] << "   ";
   cout << endl;
}
```

Таким чином, ми маємо 4 перевантажені функції, для різних типів даних. Ввони відрізняються тільки заголовком, тіло у них абсолютно однакове.

А що якщо, нам знадобиться запрограмувати алгоритм сортування у вигляді функції. Виходить, для кожного типу даних доведеться свою функцію створювати. Тобто, один і той же код буде в декількох примірниках, що дуже нераціонально. Тому в С ++ придуманий такий механізм – шаблони функцій.

Шаблони функцій – це інструкції, згідно з якими створюються локальні версії функції для певного набору параметрів і типів даних.

Синтаксис:

template <class T>

template <typename T>

template <typename T1, typename T2>

Всі шаблони функцій починаються зі слова *template*, після якого йдуть кутові дужки, в яких перераховується список параметрів. Кожному параметру має передувати зарезервоване слово *class* або *typename*.

Ключове слово *typename* говорить про те, що у шаблоні буде використовуватися вбудований тип даних, такий як: *int, double, float, char* і т. д. А ключове слово *class* повідомляє компілятору, що в шаблоні функції як параметр будуть використовуватися типи даних користувача, тобто класи.

Ми створюємо один шаблон, в якому описуємо всі типи даних. Таким чином код не буде захаращуватися.

Приклад 1. Програма виводить на екран елементи масивів різних типів, використовуючи шаблон функції.

```
#include <iostream>
using namespace std;
   // шаблон функції printArray
template <typename T>
```

```
void printArray(const T * array, int count)
{   for (int ix = 0; ix < count; ix++)
       cout << array[ix] << "   ";
   cout << endl;
}
int main()
{   const int iSize = 10, dSize = 7, fSize = 10, cSize = 5;
       // масиви різних типів даних
   int    iArray[iSize]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   double dArray[dSize]={1.2345,2.234,3.57,4.67876,5.346,6.1545,7.7682};
    float   fArray[fSize]={1.34,2.37,3.23,4.8,5.879,6.345,73.434,8.82,9.33,10.
4};
   char   cArray[cSize] = {"MARS"};
   cout << "\n\t Використання шаблонів функцій:\n";
   cout << "\n\tМасив типу int:\n\t";  printArray(iArray, iSize);
   cout << "\n\tМасив типу double:\n\t"; printArray(dArray, dSize);
   cout << "\n\tМасив типу float:\n\t"; printArray(fArray, fSize);
   cout << "\n\tМасив типу char:\n\t"; printArray(cArray, cSize);
   system("pause");
   return 0;
}
```

Результат роботи програми:

```
 Використання шаблонів функцій:

Масив типу int:
1   2   3   4   5   6   7   8   9   10

Масив типу double:
1.2345   2.234   3.57   4.67876   5.346   6.1545   7.7682

Масив типу float:
1.34   2.37   3.23   4.8   5.879   6.345   73.434   8.82   9.33   10.4


Масив типу char:
M   A   R   S
```

Шаблони класів.

У міру того, як кількість створюваних класів зростає, виявляється, що деякий клас, створений для однієї програми, необхідний в іншій програмі. У

багатьох випадках класи можуть відрізнятися тільки типами. Наприклад, один клас працює з цілочисельними значеннями, в той час як в іншій програмі він повинен працювати зі значеннями типу float.

Щоб збільшити ймовірність повторного використання існуючого коду, C++ дозволяє програмам визначати шаблони класів.

Шаблон класу визначає тіпонезалежний клас, який надалі служить для створення об'єктів необхідних типів.

Якщо компілятор C++ зустрічає оголошення об'єкта, засноване на шаблоні класу, то для побудови класу необхідного типу він буде використовувати типи, зазначені при оголошенні. Дозволяючи швидко створювати класи, що відрізняються тільки типом, шаблони класів скорочують обсяг програмування, що, в свою чергу, заощаджує час.

Пояснимо використання шаблонів класів на простому прикладі. Нехай необхідно створити простий клас "Масив", який буде виконувати прості дії: Додавання і Відображення елементів.

Приклад. Створення класу "Масив" без шаблону.

```
#include <iostream>
using namespace std;
        /*НАШ КЛАС*/
 class Massiv
 {
   int Array[10];    //Масив цілочислених значень з 10 елементів
   int count;        //Лічильник елементів масиву
  public:
   void Add(int );   //Метод для додавання елементів в масив
   void Show();      //Метод для відображення масиву на екрані
 };
 void Massiv::Add(int x)
 {
  static int pos=0;
  Array[pos]=x;
  pos++;
  count=pos;
 }
```

```
 void Massiv::Show()
 {
  for (int i=0;i<count;i++)
      cout<<Array[i]<<"\t";
  cout<<endl;
 }
int main()
{
  Massiv Arr;
  Arr.Add(100.555);
  Arr.Add(200);
  Arr.Add(300);
  Arr.Show();
  system("pause");
  return 0;
}
```

Це приклад створення звичайного класу. Але у програмістів іноді виникає необхідність створення такого ж класу, в якому відрізняється тільки тип даних. Наприклад, може знадобитися створення класу, в якому потрібно створення масиву, який буде зберігати в собі і обробляти не цілочисельні змінні, а рядкові. Як варіант, можна дописати купу класів для кожного з типів змінних, але це не раціонально. Код вийде більшим, громіздким. Чим більше коду, тим простіше в ньому помилятися і тим складніше шукати. Ось тут і приходять на допомогу шаблони класів.

Приклад 2. Створення класу "Масив" з використанням шаблону.

```
#include <iostream>
using namespace std;
int pos=0; //Позиція в масиві
template <class T> //Шаблон с класу з параметром T
 class Massiv
 {
   T Array[10];      //Масив цілочислених значень з 10 елементів
```

```
   int count;        //Лічильник елементів масиву
  public:
   void Add(T ); //Метод для додавання елементів в масив
   void Show();      //Метод для відображення масиву на екрані
};
template <class T> void Massiv<T>::Add(T x)
{ static int pos=0;
  Array[pos]=x;
  pos++;
  count=pos;
}
template <class T> void Massiv<T>::Show()
{  cout<<"\t"<<endl;
   for (int i=0;i<count;i++)
        cout<<"\t"<<Array[i];
   cout<<endl;
}
int main()
{
 setlocale(0,"");
 Massiv<int> Arr;
 Arr.Add(100.555);
 Arr.Add(200);
 Arr.Add(300);
 Arr.Show();
 Massiv<char *> Arr2;
 Arr2.Add("Рядок");
 Arr2.Add("Починаю розуміти");
 Arr2.Add("УРА");
 Arr2.Show();
 cout<<endl;
 system("pause");
 return 0;
}
```

Результат роботи програми:

100 200 300

Рядок Починаю розуміти УРА

Стандартна бібліотека шаблонів STL

Стандартна бібліотека шаблонів надає набір добре сконструйованих узагальнених компонентів C++. Бібліотека містить п'ять основних видів компонентів:

1. алгоритм (*algorithm*) – визначає обчислювальну процедуру;
2. контейнер (*container*) – управляє набором об'єктів в пам'яті;
3. ітератор (*iterator*) – забезпечує для алгоритмів засіб доступу до вмісту контейнера;
4. функціональний об'єкт (*function object*) – інкапсулює функцію в об'єкті для використання іншими компонентами;
5. адаптер (*adaptor*) – адаптує компонент для забезпечення різного інтерфейсу.

До найпопулярніших контейнерів відносять такі:

vector – колекція елементів, збережених в масиві, що збільшується в міру необхідності. Заголовочний файл – <*vector*>.

list – колекція елементів, збережених, як двонаправлений зв'язаний список. Заголовочний файл – <*list*>.

map – це колекція, яка зберігає пари значень pair <*Key,T*> і призначена для швидкого пошуку значення за ключем Key. Як ключ може бути використано, наприклад, рядок або *int*, але при цьому необхідно пам'ятати, що головною особливістю ключа є можливість застосувати до нього операцію порівняння. Важливо: ключ повинен бути унікальним. Заголовочний файл – <*map*>.

set – це колекція унікальних значень *Key* – кожне з яких є також і ключем. Тобто, це відсортована колекція, призначена для швидкого пошуку необхідного значення. До ключа пред'являються ті ж вимоги, що й у випадку ключа для *map*. Природно, використовувати її для цієї мети немає сенсу, якщо зберігати в ній прості типи даних, щонайменше необхідно визначити свій клас, який зберігає пару ключ – значення і визначає операцію порівняння по ключу. Дуже зручно використовувати

дану колекцію, якщо треба уникнути повторного збереження одного і того ж значення. Заголовочний файл – <*set*>.

multimap – це модифікований *map*, в якому відсутня вимоги унікальності ключа Тобто, якщо здійснювати пошук по ключу, то повернеться не одне значення, а набір значень, збережених з даними ключем. Заголовочний файл – <*map*>.

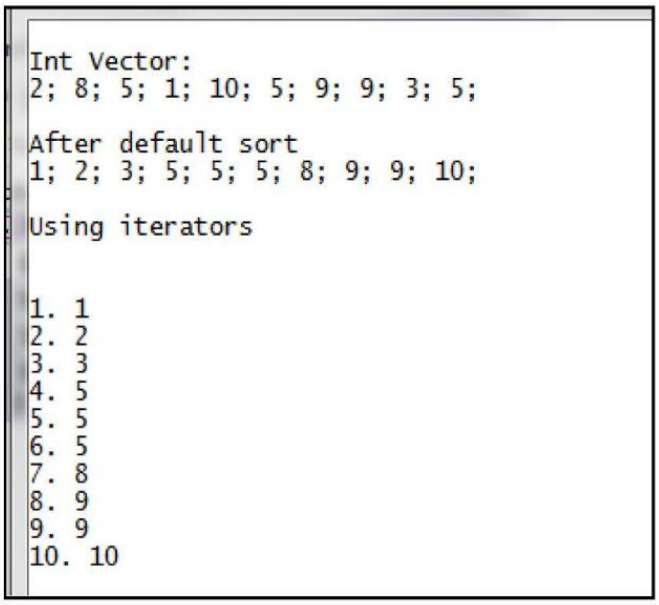
multiset – те ж саме відноситься і до цієї колекції, вимоги унікальності ключа в ній не існує, що призводить до можливості зберігання дублікатів значень. Тим не менш, існує можливість швидкого знаходження значень по ключу у випадку, якщо визначити свій клас. Оскільки всі значення в *map* і *set* зберігаються у відсортованому вигляді, то в цих колекціях можна швидко відшукати необхідне значення за ключем, але при цьому операція вставляння нового елемента буде складнішою, ніж, наприклад, в *vector*. Заголовочний файл – <*set*>.

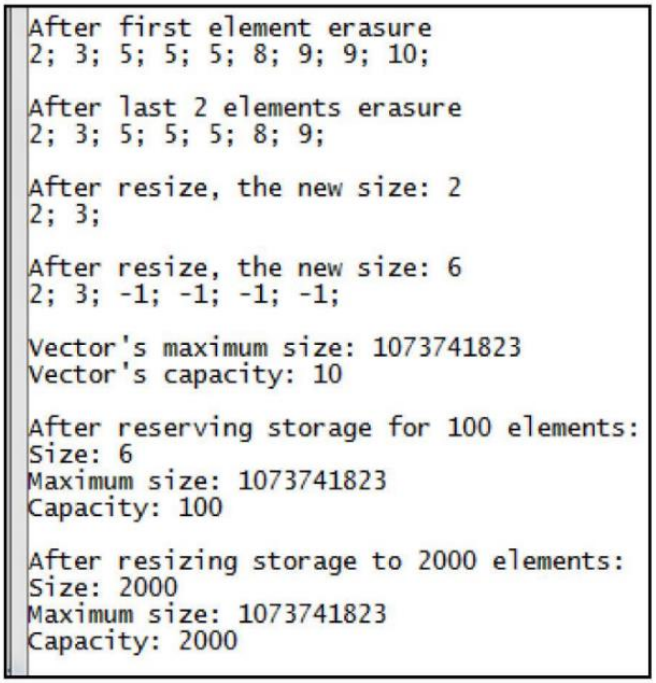
Приклад 3. Використання контейнера *vector* з *STL* для цілих чисел.

```
#include <vector>
#include <algorithm>
#include <iostream>
using namespace std;
template <class T> void pr(T& v)
{           //===== Шаблон функції для виведення за допомогою ітератора
   T::iterator p;    //====== Ітератор для будь-якого контейнера
   for ( p=v.begin(),int i=0; p!=v.end(); p++, i++)
        cout << endl << i + 1 <<". "<< *p;
   cout << '\n';
}
void main ()
{   vector <int> v(10);     //======== Вектор цілих
   cout << "\nInt Vector:\n";
   for (int i=0; i<v.size(); i++)
   {     v[i] = rand()%10 + 1;
```

```
    cout << v[i]<< "; ";
}
sort (v.begin (), v.end());   //======= Сортування за замовчуванням
cout << "\n\nAfter default sort\n";
for (int i=0; i<v.size(); i++) cout << v[i]<< "; ";
cout << "\n\nUsing iterators\n\n";
pr(v);
v.erase(v.begin());     //======== Видалення елементів
cout << "\n\nAfter first element erasure\n";
for (int i=0; i<v.size(); i++) cout << v[i]<< "; ";
v.erase (v.end()-2, v.end());
cout << "\n\nAfter last 2 elements erasure\n";
for (int i=0; i<v.size(); i++) cout << v[i]<< "; ";
int size = 2;           //======== Зміна розмірів
v.resize(size);
cout << "\n\nAfter resize, the new size: " << v.size()<< endl;
for (uint i=0; i<v.size(); i++) cout << v[i]<< "; ";
v.resize(6,-1);
cout << "\n\nAfter resize, the new size: " << v.size()<< endl;
for (uint i=0; i<v.size(); i++) cout << v[i]<< "; ";
cout << "\n\nVector's maximum size: " << v.max_size()
     << "\nVector's capacity: " << v.capacity() << endl;
v.reserve (100);
cout << "\nAfter reserving storage for 100 elements:\n"<< "Size: "
    <<v.size()<<endl<<"Maximum size: "<<v.max_size()<< endl
    << "Capacity: " << v.capacity() << endl;
v.resize(2000);
cout << "\nAfter resizing storage to 2000 elements:\n"<<"Size: "<<v.size()
    <<endl<<"Maximum size: "<<v.max_size() << endl<<"Capacity: "
    <<v.capacity()<<endl<< "\n\n";
_getch();
}
```

Результат роботи програми:





ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

Розробити програми згідно індивідуального завдання

Створити клас, який описує та забезпечує дії над даними параметризованого масиву, розмірність якого визначається під час роботи програми. Усі обчислення і перетворення реалізувати у вигляді функцій-членів класу.

1	В масиві обчислити: – номер елемента масиву, найближчого до середньоарифметичного його значень; – різниця елементів масиву, що розташовані між першим від'ємним та другим додатним елементами. Перетворити масив так, щоб у його першій половині розташовувались елементи, що стоять в парних позиціях, а в другій – елементи, що стоять в непарних позиціях.
2	Дана прямокутна матриця. Визначити: – кількість від'ємних елементів в тих рядках, які містять хоча б один нульовий елемент; – суму модулів елементів, які розташовані після першого додатного елемента. Впорядкувати елементи матриці за спаданням модулів елементів.
3	У довільній матриці обчислити: – кількість елементів масиву, рівних нулю; – суму елементів масиву, які лежать в діапазоні від A до B. Впорядкувати елементи масиву за спаданням модулів елементів.
4	У довільній матриці обчислити: – кількість елементів масиву, рівних нулю; – суму елементів масиву, які лежать в діапазоні від A до B. Впорядкувати елементи масиву за спаданням модулів елементів.
5	В одновимірному масиві елементів, обчислити: – номер максимального за модулем елемента; – суму модулів елементів, які розташовані після першого додатного елемента. Перетворити масив таким чином, щоб спочатку розташовувались всі елементи, ціла частина яких лежить в інтервалі [a,b], а потім – всі інші.
6	В масиві обчислити: – мінімальний за модулем елемент масиву;

	– суму модулів елементів, які розташовані після першого від'ємного елемента. Ущільнити масив, видаливши з нього елементи, величина яких знаходиться на інтервалі [a,b]. Місце, яке звільниться в кінці масиву заповнити символом чи числом з клавіатури.
7	В масиві обчислити: – мінімальний за модулем елемент масиву; – суму модулів елементів масиву, розташованих після першого нульового елемента. Перетворити масив так, щоб в першій його половині розташовувались елементи, що стоять на парних позиціях, а в другій – елементи, що стоять в непарних позиціях.
8	У матриці обчислити: – максимальний за модулем елемент масиву; – суму елементів масиву, що розташовані між першим і другим додатними елементами. Перетворити матрицю так, щоб всі елементи, рівні нулю, розташовувались в кінці.
9	Дана прямокутна матриця. Визначити: – кількість рядків, які не містять жодного нульового елемента; – максимальне із чисел, що зустрічається в заданій матриці більше одного разу. Перетворити матрицю так, щоб всі нульові елементи розташовувались на початку.
10	Дана прямокутна матриця. Визначити: – кількість стовпців, які не містять жодного нульового елемента. – кількість елементів, менших за A , але більших за B. Переставляючи рядки заданої матриці, розташувати їх у відповідності із зростанням суми значень у стовпцях.
11	В одномірному масиві обчислити: – добуток елементів масиву з парними номерами; – суму елементів масиву, які розташовані між першим і останнім нульовими елементами.

	Впорядкувати масив таким чином, щоб спочатку розташовувались всі додатні елементи, а потім – всі від'ємні (елементи, рівні 0 вважати додатними).
12	Дана прямокутна матриця. Визначити : – кількість стовпців, які містять хоча б один нульовий елемент; – номер рядка, в якому знаходиться найдовша серія з однакових елементів. Впорядкувати масив таким чином, щоб спочатку розташовувались всі серії з однакових елементів, а потім – всі решта елементів.
13	В одномірному масиві, що складається з N дійсних елементів, обчислити: – суму елементів масиву з непарними елементами; – суму елементів масиву, які розташовані між першим і останнім від'ємними елементами. Перетворити масив, видаливши з нього всі елементи, модуль яких не перевищує число, що вводиться з клавіатури. Елементи, які звільняться в кінці масиву заповнити нулями.
14	Дана прямокутна матриця. Визначити : – добуток елементів в тих рядках, які не містять від'ємних елементів; – максимум серед сум елементів діагоналей, паралельних головній діагоналі матриці. Перетворити матрицю, видаливши з неї всі елементи, модуль яких не перевищує число, що вводиться з клавіатури. Елементи, які звільняться в кінці масиву, заповнити нулями.
15	В одномірному масиві, що складається з N дійсних елементів, обчислити: – максимальний елемент масиву; – суму елементів масиву, що розташовані до останнього додатного елемента. Перетворити масив, видаливши з нього всі елементи, модуль яких знаходиться в інтервалі [a,b]. Елементи, які звільняються в кінці масиву заповнити нулями.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яка різниця між шаблоном та макросом?

2. Чим відрізняється параметр шаблона від параметру функції?
3. Як створити шаблонний клас?
4. Що являє собою стандартна бібліотека шаблонів STL?
5. Які основні шаблонні класи колекцій ви знаєте?
6. Що представляють собою ітератори?

СПИСОК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Stroustrup B. The C++ Programming Language 4th Edition. Boston : Addison-Wesley, 2013. 1281 p.

2. Armstrong D. J. The Quarks of Object-Oriented Development. *An Introduction to Database Systems.* 6th ed. Boston, 1994. P. 650.

3. Іванов Є. О., Ліндер Я. М., Жереб К. А. Основи мови програмування C++ : навч. посіб. Київ : Логос, 2020. 90 с.

4. Єфіменко С. В. Методичний посібник з курсу «Об’єктно-орієнтоване програмування. Мови C/C++». Київ : КНУ імені Тараса Шевченка, 2021. 124 с.

5. Основи програмування на C++ : навч. посіб. / О. О.Водка та ін. Харків : НТУ «ХПІ», 2021. 112 с.

6. Порєв В. М. Об'єктно-орієнтоване програмування: конспект лекцій [Електронний ресурс] : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2022. 271 с.

7. Potok T. E., Vouk M., Rindos A. Productivity analysis of object-oriented software developed in a commercial environment. *Software: Practice and Experience.* 1999. Vol. 29, no. 10. P. 833–847. URL: https://doi.org/10.1002/(sici)1097-024x(199908)29:10%3C833::aid-spe258%3E3.0.co;2-p (date of access: 05.08.2024).

8. Грицюк Ю. І., Рак Т. Є. Програмування мовою C++ : навч. посіб. Львів : Вид-во Львівського ДУ БЖД, 2011. 292 с.

9. Пінчук В. П., Лозовська Л. І. Програмування мовою C/C++ з прикладами та вправами : навч. посіб. Запоріжжя : ЗНТУ, 2009. 204 с.

10. Бублик В. В. Об’єктно-орієнтоване програмування : підручник. Київ : ІТ-книга, 2015. 624с.

11. Жуковский С. С., Вакалюк Т. А. Об‘єктно-орінтоване програмування мовою С++ : навч.-метод. посіб. Житомир : Вид-во ЖДУ, 2016. 100 с.

12. Кузнєцов, М. С. Об’єктно-орієнтоване програмування з використанням UML та мови С++ : навч. посіб. Дніпропетровськ : НМетАУ, 2003. 90 с.

11. Search. *Это госы! Wiki*. URL: https://gos-it.fandom.com/wiki/Special:Search?query=Wiki (date of access: 05.08.2024).

12. IT-Academy. ⊳ IT-Academy • IT курсы программирования в Минске | Образовательный центр ПВТ. URL: https://www.it-academy.by/course/osnovy-programmirovaniya/ (date of access: 05.08.2024)

13. Is learning C++ still worthwhile to learn?. Quora. URL: https://www.quora.com/C++-programming-language/Is-learning-C++-still-worthwhile-to-learn (date of access: 05.08.2024).

Навчально-методичне видання

Лозовська Людмила Іванівна,
Савчук Лариса Миколаївна,
Климкович Тетяна Олександрівна

ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

Навчально-методичні рекомендації
з виконання лабораторних робіт

В авторській редакції
Комп'ютерна верстка Л. І. Лозовська

Експертний висновок склав канд. екон. наук, доц. К. О. Удачина

Зареєстровано НМВ УДУНТ (№ 786 від 07.11.2024)

Формат 60х84 1/16. Ум. друк. арк. 3,49. Обл.-вид. арк. 1,75.
Зам. № 100

Видавець: Український державний університет науки і технологій
вул. Лазаряна, 2, ауд.2216, м. Дніпро, 49010.
Свідоцтво суб'єкта видавничої справи ДК №7709 від 14.12.2022

Адреса видавця та оперативної поліграфії:
вул. Лазаряна, 2, Дніпро, 49010