ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

«МЕТОДИЧНІ ВКАЗІВКИ ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ З ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ НА С++»

за освітньо-професійною програмою
першого (бакалаврського) рівня спеціальності
122 «Компютерні науки»

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 6 від «24» травня 2024 р.)
Обов'язково до розміщення в репозитарії

Запоріжжя 2024

Методичні вказівки до виконання лабораторних робіт з об’єктно-орієнтоване програмування на С++ за освітньо-професійною програмою першого (бакалаврського) рівня спеціальності 122 «Комп’ютерні науки» / Уклад. Нікуліна О.М. Запоріжжя, ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА», 2024. 53 с.

Методичні вказівки включають теми, мету, теоритичні основи, приклади програм та завдання для кожної лабораторної роботи, вимоги до оформлення звіту з лабораторних робіт, список рекомендованої літератури.

Рекомендовано для студентів спеціальності 122 Комп’ютерні науки першого (бакалаврського) рівня освіти.

Самостійне електронне текстове мережеве видання

Затверджено на засіданні кафедри
цифрових технологій та проєктно-
аналітичних рішень
Протокол № 8 від «02» квітня 2024 р.

Узгоджено:
Секретар Редакційної ради

_____ Малій Х. В.
«13» травня 2024 р.

ЗМІСТ

ВСТУП

Вказівки призначені для студентів, які вивчають мову С++ на семінарах або самостійно. Класи, шаблони, успадкування, винятки, віртуальні функції, абстрактні класи розглядаються на прикладах, супроводжуваних необхідними теоретичними відомостями.

Предметом навчальної дисципліни «Об'єктно-орієнтованого програмування» є методи алгоритмізації та програмування на мовах С++ та Java. Завдання дисципліни – освоєння методів та засобів об'єктно-орієнтованого програмування у візуальному середовищах Visual Studio та IntelliJ IDEA.

Мета цих методичних вказівок – допомогти студентам засвоїти основи об'єктно-орієнтованого програмування на мові С++.

Лабораторні роботи передбачають вивчення розділів: побудова та об'ява класів, перевантаження, відкрите та закрите успадкування, віртуальні та абстрактні класи, винятки, використання контейнерів мови програмування С++. Для виконання завдань студент повинен в достатньому ступені володіти процедурним програмуванням на мові С++. До кожної лабораторної роботи в методичних вказівках наведені: мета роботи, теоретичні основи, приклади та варіанти контрольних задач за розділами курсів, що вивчаються. Варіанти робіт відповідають вимогам навчального плану та силабусу і можуть бути застосовані для здійснення контролю знань студентів зі спеціальностями 122 – «Комп'ютерні науки».

Дисципліна спрямована на отримання здобувачами наступних загальних та спеціальних (фахових) компетентностей:

ЗК1. Здатність до абстрактного мислення, аналізу та синтезу.

ЗК2. Здатність застосовувати знання у практичних ситуаціях.

ЗК3. Знання та розуміння предметної області та розуміння професійної діяльності.

ЗК6. Здатність вчитися й оволодівати сучасними знаннями.

ЗК9. Здатність працювати в команді.

СК3. Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проєктування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв'язності та нерозв'язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем.

СК8. Здатність проєктувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об'єктно-орієнтованого, функціонального, логічного, відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмами управління.

Дисципліна спрямована на отримання здобувачами наступних програмних результатів :

ПР1. Застосовувати знання основних форм і законів абстрактно-логічного мислення, основ методології наукового пізнання, форм і методів вилучення, аналізу, обробки та синтезу інформації в предметній області комп'ютерних наук.

ПР5. Проєктувати, розробляти та аналізувати алгоритми розв'язання обчислювальних та логічних задач, оцінювати ефективність та складність алгоритмів на основі застосування формальних моделей алгоритмів та обчислюваних функцій.

ПР9. Розробляти програмні моделі предметних середовищ, вибирати парадигму програмування з позицій зручності та якості застосування для реалізації методів та алгоритмів розв'язання задач в галузі комп'ютерних наук.

У результаті виконання лабораторних робіт здобувач вищої освіти повинен продемонструвати достатній рівень сформованості наступних результатів навчання:

- Здатність до логічного мислення, побудови логічних висновків, використання формальних мов і моделей алгоритмічних обчислень, проєктування, розроблення й аналізу алгоритмів, оцінювання їх ефективності та складності, розв'язності та нерозв'язності алгоритмічних проблем для адекватного моделювання предметних областей і створення програмних та інформаційних систем.
- Здатність застосовувати існуючі і розробляти нові алгоритми розв'язування задач у галузі комп'ютерних наук.
- Здатність розробляти і реалізовувати проекти зі створення програмного забезпечення, у тому числі в непередбачуваних умовах, за нечітких вимог та необхідності застосовувати нові стратегічні підходи, використовувати програмні інструменти для організації командної роботи над проєктом.
- Здатність проектувати та розробляти програмне забезпечення із застосуванням різних парадигм програмування: узагальненого, об'єктно-орієнтованого, функціонального, логічного, з відповідними моделями, методами й алгоритмами обчислень, структурами даних і механізмів управління.

ЛАБОРАТОРНА РОБОТА 1
КЛАСИ ТА ОБ'ЄКТИ

Мета лабораторної роботи – опанувати написання програм з використанням об'єктів.

1.1 Теоретичні основи

Класи у C++ є основним засобом визначення користувацьких типів. Класи аналогічні структурам. *Клас* (class) – це структурований тип даних, набір елементів даних різних типів і функцій для роботи з цими даними. Властивості об'єктів описуються за допомогою полів класів, а дії над об'єктами описуються за допомогою функцій, які називаються методами класу. Клас має ім'я, складається з полів, званих членами класу і функцій – методів класу.

Опис класу має такий вигляд:

```
class name  // name – ім'я класу
{
private:
//Опис закритих членів і методів класу
protected:
// Опис захищених членів і методів класу
public:
// Опис відкритих членів і методів класу
```

Одним з важливих принципів об'єктно-орієнтованого підходу є *інкапсуляція* (приховування даних). Зміст інкапсуляції полягає у приховуванні від зовнішнього користувача деталей реалізації об'єкта. Для реалізації інкапсуляції мова C++ надає рівні доступу до елементів класу: public, protected та private. Елементи класу, декларовані як private (закриті) можуть використовуватись лише у функціях-елементах цього класу а також у його друзях (класах та функціях). Елементи, декларовані як protected (захищені), додатково можуть бути використані у похідних класах. Елементи, декларовані як public (публічні) можуть бути використані у будь-якій частині програми. Відповідні ключові слова з двокрапкою використовують для групування елементів класу.

Функції-елементи, які надають доступ до закритих елементів класу, мають назву функцій доступу. Їх ще називають геттерами і сеттерми (getters / setters). Функції-елементи, які були оголошені всередині тіла класу, можуть бути реалізовані поза класом. Для того, щоб визначити приналежність функції до класу, використовується операція дозволу області видимості (::). Функція-елемент має вільний доступ до елементів даних. Функції-елементи можна повністю визначити всередині класу. Функції-елементи мають доступ до інших елементів класу через так званий вказівник this. Функції-елементи отримують цей вказівник як неявний аргумент. Цей вказівник вказує на об'єкт, для якого викликана функція-елемент. Найчастіше вказівник this застосовують для того, щоб уникнути конфліктів імен.

Після опису класу необхідно описати змінну типу class.

```
name_class name;
```

де name_class – ім'я класу, name – ім'я змінної.

Надалі змінну типу class будемо називати «об'єкт» або «екземпляр класу». Оголошення змінної типу class (в нашому прикладі змінна name типу name_class) називається створенням (инициализацией) об'єкта (екземпляра класу). Після опису змінної можна звертатися до членів і методам класу. Звернення до членів і методам класу здійснюється аналогічно поводженню до полів структури за допомогою оператора «.» (точка, крапка).

```
name.p1; // Звернення до поля p1
// екземпляра класу name.
name.f1(par1, par2, ... parn); // Звернення до методу f1
// екземпляра класу name, par1, par2, ..., parn – список
// формальних параметрів функції f1.
```

Члени класу доступні з будь-якого методу класу і їх не треба передавати в клас, як параметри функцій-методів.

Використання відкритих членів і методів дозволяє отримати повний доступ до елементів класу, однак це не завжди добре. Якщо всі члени класу оголосити відкритими, то при безпосередньому зверненні до них з'явиться потенційна можливість внести помилку в функціонування взаємопов'язаних між собою методів класу. Тому, загальним принципом є наступне: «Чим менше відкритих даних про клас використовується в програмі, тим краще ». Зменшення кількості публічних членів і методів дозволить мінімізувати кількість помилок.

Бажано, щоб всі члени класу були закритими і тоді неможливо буде звертатися до членів класу безпосередньо за допомогою оператора «.» Кількість відкритих методів також слід мінімізувати. Якщо в описі елементів класу відсутня вказівка на метод доступу, то члени і методи вважаються закритими (private). Прийнято описувати методи за межами класу.

При створенні в програмі екземпляра класу, формується покажчик this, в якому зберігається адреса змінної – екземпляра класу. Покажчик this виступає в ролі покажчика на поточний об'єкт.

У попередньому прикладі метод `chislo.vvod()` використовувався для присвоєння початкового значення деяким членам класу, однак для спрощення процесу ініціалізації об'єкта передбачена спеціальна функція, яка називається конструктором. Ім'я конструктора збігається з ім'ям класу, конструктор запускається автоматично при створенні екземпляра класу (при оголошенні змінної типу class) функції-конструктори не повертають значення, але при описі а не слід вказувати в якості значення, що повертається тип void.

Конструктор автоматично запускається на виконання для кожного примірника класу при його описі. Найчастіше конструктор служить для ініціалізації полів екземпляра класу. Конструктори можуть перевантажуватися, а також можна використовувати параметри за замовчуванням. Ще одним видом конструктора є конструктор копіювання, який дозволяє створити копію примірника класу. Це актуально коли необхідні два примірника класу з одними і тими ж значеннями членів класу. Синтаксис заголовка конструктора копіювання наступний:

```
public: name_constructor (type_class & name);
```

де `name_constructor` – ім'я конструктора копіювання, `type_class` – ім'я класу, переданого в конструктор копіювання (в конструктор копіювання можна передавати тільки екземпляри класу), `name` – передається в конструктор копіювання екземпляр класу.

C++ дозволяє побудувати функцію-деструктор, яка виконується при знищенні екземпляра класу, що відбувається в наступних випадках: при завершенні програми або виході з функції та при звільненні пам'яті, виділеної для екземпляра класу. Деструктор має ім'я ~ ім'я_класу, не має параметрів і не може перевантажуватися. Якщо деструктор не визначений явно, то буде використовуватися стандартний деструктор.

1.2. Приклади

Приклад 1.1. Розглянемо клас `complex` для роботи з комплексними числами. У класі complex будуть члени класу: `double x` – дійсна частина комплексного числа, `double y` – уявна частина комплексного сила; методи класу: `double modul()` – функція обчислення модуля комплексного числа, `double argument()` – функція обчислення аргументу комплексного числа, `void show_complex()` – функція виводить комплексне число на екран.

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
class complex
{
     public:
     double x;
     double y;
     double modul() {return pow (x * x + y * y, 0.5);}
     double argument() {return atan2 (y, x) * 180 / PI;}
     void show_complex () { if (y>= 0)
     cout << x << "+" << y << "i" << endl;
     else
     cout << x << y << "i" << endl;
     }
};
int main ()
{
     complex chislo;
     chislo.x = 3.5;
     chislo.y = -1.432;
     chislo.show_complex();
     cout << "Modul ‘chisla =" << chislo.modul ();
     cout << endl << "Argument chisla =" << chislo.argument ()
<< endl;
}
```

Приклад 1.2. Змінимо розглянутий раніше приклад класу `complex`.

Додамо метод vvod, призначений для введення дійсної та уявної частини числа, члени класу і метод show_complex зробимо закритими, а інші методи відкритими. Текст програми буде мати вигляд:

```
#include <iostream>
#include <math.h>
#define PI 3.14159
using namespace std;
class complex
{
     public:
     void vvod();
     double modul();
     double argument();
     void show_complex();
     private:
     double x;
     double y;
     };
     void complex :: vvod ()
     {
     cout << "Vvedite x \ t";
     cin >> x;
     cout << "Vvedite y \ t";
     cin >> y;
     show_complex ();
     }
     double complex :: argument()
     {
     return atan2(y, x) * 180 / PI;
     }
     void complex :: show_complex()
     {
     if (y>= 0)
     cout << x << "+" << y << "i" << endl;
     else cout << x << y << "i" << endl;
     }
```

```
int main ()
{
     complex chislo;
     chislo.vvod();
     cout<< "Modul kompleksnogo chisla =" << chislo.modul() <<
endl;
     cout<<"Argument     kompleksnogo     chisla     ="     <<
chislo.argument();
}
```

У розглянутому прикладі показано спільне використання відкритих і закритих елементів класу. Поділ на відкриті і закриті в цьому прикладі кілька штучне, воно проведено лише для ілюстрації механізму спільного використання закритих або відкритих елементів класу. Якщо спробувати звернутися до методу show_complex() або до членам класу x, y з функції main, то компілятор видасть повідомлення про помилку (доступ до елементів класу заборонений).

Приклад 1.3. Додамо в створений в попередньому прикладі клас complex конструктор:

```
     #include <iostream>
     #include <string>
     #include <math.h>
     #define PI 3.14159
     using namespace std;
     class complex
     {
     public:
          complex();
          double modul();
          double argument();
          void show_complex();

     private:
          double x;
          double y;
     };
     int main()
{
```

```
    complex chislo;
    cout<<"Modul kompleksnogo chisla
="<<chislo.modul()<<endl;
    cout<<"Argument kompleksnogo chisla
="<<chislo.argument();
}
complex :: complex ()
{
    cout << "Vvedite x \t";
    cin >> x;
    cout << "Vvedite y \t";
    cin >> y;
    show_complex ();
}
double complex :: modul ()
{
    return pow (x * x + y * y, 0.5);
}
double complex :: argument ()
{
    return atan2 (y, x) * 180 / PI;
}
void complex :: show_complex ()
{
    if (y>= 0) cout << x << "+" << y << "i" << endl;
    else cout << x << y << "i" << endl;
}
```

1.3. Задачі

У всіх завданнях, крім зазначених операцій, повинні бути реалізовані наступні методи: метод ініціалізації *Init*, введення з клавіатури *Read*, виведення на екран *Display*, перетворення в рядок *toString*. Всі поля закриті.

Задача 1.1

Створити клас *Vector3d*, що задається трійкою координат. Реалізувати методи: додавання і віднімання векторів, скалярний добуток векторів, множення на скаляр, порівняння векторів, обчислення

довжини вектора, порівняння довжини вектора.

Задача 1.2

Створити клас *Money* для роботи з грошовими сумами. Число представлено двома полями: гривні і копійки. Реалізувати методи: додавання і віднімання, ділення сум, розподіл суми на дробове число, множення суми на дробове число, операції порівняння.

Задача 1.3

Створити клас *Triangle* для подання трикутника. Поля: сторони і кути. Реалізувати операції: отримання і зміни полів даних, обчислення площі, обчислення периметра, обчислення висот, визначення виду трикутника.

Задача 1.4

Створити клас *Angle* для роботи з кутами на площині. Поля: градуси і хвилини. Реалізувати методи: переклад в радіани, приведення до діапазону 0–360, збільшення і зменшення кута на задану величину, отримання синуса, порівняння кутів.

Задача 1.5

Створити клас *Data* для роботи з датами. Поля: рік, місяць, день. Реалізувати операції: віднімання, порівняння дат, визначення високосного року, переклад в кількість днів.

Задача 1.6

Створити клас *Time* для роботи з часом. Поля: години, хвилини, секунда. Реалізувати операції: віднімання, порівняння часу, переклад в хвилини і секунди.

Задача 1.7

Створити клас *Account*, який представляє собою банківський рахунок. Поля: прізвище власника, номер рахунку, відсоток нарахування, сума в гривнях. Реалізувати операції: зміна власника рахунку, зняття деякої суми з рахунку, покласти гроші на рахунок, нарахувати відсоток.

Задача 1.8

Створити клас *Fraction*, для роботи з дробовими числами. Поля: ціла частина і дрібна. Реалізувати операції: додавання, віднімання, множення, операції порівняння.

Задача 1.9

Створити клас *Goods* (Товар). Поля: найменування товару, дата оформлення, ціна товару, кількість одиниць товару, номер накладної.

Реалізувати методи: зміни ціни товару, зміни кількості товару, обчислення вартості товару.

Задача 1.10

Створити клас *Payment* (Зарплата). Поля: ПІБ, оклад, рік надходження на роботу, прибутковий податок, відсоток надбавки, кількість відпрацьованих днів на місяць, нарахована і утримана суми. Реалізувати методи: обчислення нарахованої суми, обчислення утриманої суми, сума, що видається, обчислення стажу.

ЛАБОРАТОРНА РОБОТА 2
ПЕРЕВАНТАЖЕННЯ ОПЕРАЦІЙ

Мета лабораторної роботи – навчитися використовувати дружні функції та перевантажувати операції у класі.

2.1. Теоретичні основи

Будь-яка зовнішня функція, певна як «друг» класу, має необмежений доступ до будь-яких елементів класу, в тому числі до закритих. Для того щоб зовнішня функція стала одним класу, необхідно в інтерфейсі класу задати її прототип, додавши попереду слово friend. Друзі не є частиною області видимості класу та не отримують покажчика this.

```
class SomeClass
{
        friend void f(SomeClass &sc);
    private:
        int i;
};
void f(SomeClass &sc)
{
        cout << sc.i; // Доступ до закритого елементу даних
}
void main()
{
        SomeClass s1;
        f(s1);
}
```

Дружні функції можуть бути реалізовані у тілі класу. Такі функції мають неявний модифікатор `inline`. Друзі можуть бути оголошені у будь-якій частині класу (`public`, `private` або `protected`). Це не впливає на механізм доступу.

Можна здійснити оголошення класу без визначення. Завдяки цьому можна декларувати взаємну дружбу:

```
class Y;          // оголошення класу
class X
{
        friend Y;
        . . .
};
class Y;
    {                  // визначення класу
        friend X;
        . . .
};
```

Функції-елементи класу `X` можуть бути оголошені як друзі класу `Y`:

```
class X
{
        . . .
        void member_funcX();
};
class Y
{
        friend void X::member_funcX();
        . . .
};
```

Якщо `X` – друг класу `Y`, `Y` – друг класу `Z`, `X` не є автоматично другом `Z`. Дружба не успадковується.

В C ++ існує можливість перевантаження операції всередині класу, наприклад, можна домогтися того, що операція * при роботі з матрицями здійснювала множення матриць, а при роботі з комплексними числами – множення комплексних чисел.

Для перевантаження операцій всередині класу потрібно написати спеціальну функцію-метод класу. При перевантаженні операцій слід пам’ятати наступне:

– при перевантаженні не можна змінити пріоритет операцій;
– не можна змінити тип операції (з унарною операції не можна зробити бінарну або навпаки);
– перевантажена операція є членом класу і може використовуватися тільки у виразах з об’єктами свого класу;
– не можна створювати нові операції;
– заборонено перевантажувати операції: . (доступ до членів класу), унарна операцію * (значення за адресою покажчика), :: (розширення області видимості), ?: (операція if);
– допустима перевантаження наступних операцій: +, –, *, /,%, =, <,>, + =, - =, * =, / =, <,>, &&, ||, ++, —, ( ), [], new, delete.

Для перевантаження бінарної операції всередині класу необхідно створити функцію-метод:

```
type operator symbols (type1 parametr)
{
     операторы;
}
```

тут type – тип повертається операцією значення, operator – службове слово, symbols – перевантажуючи операція, type1 – тип другого операнда, першим операндом є екземпляр класу, parametr – ім’я змінної другого операнда.

2.2. Приклади

Приклад 2.1. Створити клас для роботи з комплексними числами, в якому перевантажити операції додавання. Текст програми:

```
#include <iostream>
using namespace std;
class complex
{
     public:
     complex (bool pr = true);
     //  Метод,  який  реалізує  перевантаження  операції
додавання.
     complex operator + (complex M);
     private:
     double x;
     double y;
```

```
     void show_complex();
     };
     int main ()
     {
     complex chislo1, chislo2, chislo3 (false);
     chislo3 = chislo1 + chislo2;
     cout << "chislo3 =";
     chislo3.show_complex();
     }
     complex :: complex (bool pr)
     {
     if (pr)
     {
          cout << "VVedite x \t";
          cin >> x;
          cout << "Vvedite y \t";
          cin >> y;
          show_complex();
     }
     else {x = 0; y = 0;}
}
void complex :: show_complex()
{
     if (y> = 0) cout << x << "+" << y << "i" << endl;
     else cout << x << y << "i" << endl;
}
complex complex :: operator + (complex M)
{
     complex temp (false);
     temp.x = x + M.x;
     temp.y = y + M.y;
     return temp;
};
```

Розглянемо, як реалізується перевантаження операції ++ x і x ++ на прикладі класу комплексних чисел.

Приклад 2.2. Нехай операція ++ x збільшує Справді і уявну частину

комплексного числа x на 1, а x ++ збільшує на 1 тільки дійсну частину комплексного числа x.

```
#include <iostream>
using namespace std;
class complex
{
     public:
     complex (bool pr = true)
     {
          if (pr)
          {
               cout << "Vvedite x \t";
               cin >> x;
               cout << "Vvedite y \t";
               cin >> y;
               show_complex ();
          };
     }
     complex operator ++ ()
     {
          x ++;
          y ++;
          return * this;
     }
     complex operator ++ (int)
     {
          x ++;
          return * this;
     }
     void show_complex ()
     {
          if (y> = 0)
               cout << x << "+" << y << "i" << endl;
          else cout << x << y << "i" << endl;
     };
     double x;
```

```
    double y;
};
int main ()
{
    complex chislo2;
    ++ chislo2;
    cout << "++ chislo2 =";
    chislo2.show_complex();
    chislo2 ++;
    cout << "chislo2 ++ =";
    chislo2.show_complex();
}
```

2.3. Задачі

Виконати завдання лабораторної роботи 1 (свій варіант) з конструктором і перевантаженням операцій. Реалізувати завдання двома способами: всі операції визначити як зовнішні дружні функції і як методи класу.

Задача 2.1

У класі *Vector3d* перевантажити операції: додавання і віднімання векторів, порівняння векторів, порівняння довжини вектора.

Задача 2.2

У класі *Money* перевантажити операції: додавання і віднімання, операції порівняння.

Задача 2.3

У класі *Triangle* перевантажити операції: обчислення площі, обчислення периметра, обчислення висот.

Задача 2.4

У класі *Angle* перевантажити операції: переклад в радіани, збільшення і зменшення кута на задану величину, порівняння кутів.

Задача 2.5

У класі *Data* перевантажити операції: додавання, віднімання, порівняння дат.

Задача 2.6

У класі *Time* перевантажити операції: додавання, віднімання, порівняння часу.

Задача 2.7

У класі *Account* перевантажити операції: зняття деякої суми з рахунку, покласти гроші на рахунок.

Задача 2.8

У класі *Fraction* перевантажити операції: додавання, віднімання, множення, операції порівняння.

Задача 2.9

У класі *Goods* перевантажити операції: зміни ціни товару, обчислення вартості товару.

Задача 2.10

У класі *Payment* перевантажити операції: сума, що видається, обчислення стажу.

ЛАБОРАТОРНА РОБОТА 3
УСПАДКУВАННЯ

Мета лабораторної роботи – опанувати написання програм з об'єктами, які успадковуються.

3.1. Теоретичні основи

Успадкування – це третій «кит» (поряд з інкапсуляцією і поліморфізмом), на якому стоїть об'єктно-орієнтоване програмування, це спосіб повторного використання програмного забезпечення. При успадкуванні обов'язково є клас-предок (батько, який породжує клас) і клас-спадкоємець (нащадок, породжений клас). В С ++ породжує клас називається базовим, а породжений клас – похідним. Всі терміни еквівалентні.

Відносини між батьківським класом і його нащадками називаються ієрархією успадкування. Глибина успадкування стандартом не обмежена. Порядок успадкування в ієрархії не регламентується: все залежить від потреб завдання і досвіду програміста.

Простим (або одиночним) називається успадкування, при якому похідний клас має тільки одного з батьків. Формально успадкування одного класу від іншого задається наступною конструкцій:

```
class ім'я_класу_нащадка: [модифікатор_доступу] ім'я_базового_класу
  {тіло_класу}
```

Модифікатор доступу визначає доступність елементів базового

класу в класі-нащадка. Квадратні дужки не є елементом синтаксису, а показують, що модифікатор може бути відсутнім. Цей модифікатор називається модифікатором успадкування.

При розробці окремих класів використовуються два модифікатора доступу до елементів класу: public і private. При спадкуванні застосовується ще один: protected. Захищені елементи класу доступні тільки прямим спадкоємцям і нікому іншому

Доступність елементів базового класу з класів-спадкоємців змінюється в залежності від модифікаторів доступу в базовому класі і модифікатора успадкування. Яким би не був модифікатор спадкування, приватні елементи базового класу недоступні в його нащадках.

Слід розрізняти тип доступу до елементів в базовому класі і тип спадкування. У C++ прийняті такі типи успадкування:

– *відкритий базовий клас* (public): відкриті елементи базового класу залишаються відкритими у похідному класі, захищені елементи базового класу залишаються захищеними у похідному класі, закриті елементи базового класу є закритими у похідному класі;

– *захищений базовий клас* (protected): відкриті та захищені елементи базового класу є захищеними у похідному класі, закриті елементи базового класу є закритими у похідному класі;

– *закритий базовий клас* (private): відкриті та захищені елементи базового класу є закритими у похідному класі, закриті елементи базового класу є закритими у похідному класі; цей спосіб є прийнятим за умовчанням.

Конструктори не успадковуються – вони створюються в похідному класі (якщо не визначені програмістом явно). Система надходить з конструкторами наступним чином:

– якщо в базовому класі немає конструкторів або є конструктор без аргументів (або аргументи присвоюються за замовчуванням), то в похідному класі конструктор можна не писати будуть створені конструктор копіювання і конструктор без аргументів;

– якщо в базовому класі всі конструктори з аргументами, похідний клас зобов’язаний мати конструктор, в якому явно повинен бути викликаний конструктор базового класу;

– при створенні об’єкта похідного класу спочатку викликається конструктор базового класу, потім – похідного.

Деструктор класу, як і конструктори, не успадковується, а

створюється. З деструкторами система надходить наступним чином:

– при відсутності деструктора в похідному класі система створює деструктор за замовчуванням;

– деструктор базового класу викликається в деструкторі похідного класу автоматично незалежно від того, визначено він явно або створений системою;

– деструктори викликаються (для знищення об’єктів) в порядку, зворотному виклику конструкторів.

Створення і знищення об’єктів виконується за принципом LIFO: останнім створений – першим знищений.

Клас-нащадок успадковує структуру (всі елементи даних) і поведінку (всі методи) базового класу. Клас-нащадок отримує в спадок все поля базового класу (хоча, якщо вони були приватні, доступу до них не має). Якщо новина поля не повинні додаватися, розмір класу-спадкоємця збігається з розміром базового класу. Породжений клас може додати власні поля.

Додаткові поля повинні ініціалізуватися конструктором спадкоємця. Додаткові поля похідного класу можуть збігатися і по імені, і за типом з полями базового класу – в цьому випадку нове поле приховує поле базового класу, тому для доступу до останнього поля в класі-нащадка необхідно використовувати префікс-кваліфікатор базового класу.

Клас-нащадок успадковує всі методи базового класу, крім операції присвоювання – вона створюється для нового класу автоматично, якщо не визначена явно. У класі-нащадка можна визначати нові методи. Якщо в класі-нащадка ім’я методу і його прототип збігаються з ім’ям метод базового класу, то кажуть, що метод похідного класу приховує метод базового класу. Щоб викликати метод батьківського класу, потрібно вказувати його з кваліфікаторів класу.

Статичні поля нормально успадковуються. Однак все нащадки поділяють єдину копію статичних полів. Нормально успадковуються і статичні методи; при виклику можна писати або префікс спадкоємця, або префікс базового класу.

Обмежень в спадкуванні вкладених класів немає: зовнішній клас може успадковувати від вкладеного і навпаки; вкладений клас може успадковувати від вкладеного.

Відкрите спадкування встановлює між класами відношення «є»:

клас-нащадок є різновидом класу-предка. Це означає, що всюди, де може бути використаний об’єкт базового класу (при присвоєнні, при передачі параметрів і повернення результату), замість нього дозволяється підставляти об’єкт похідного класу. Дане положення називається принципом підстановки і підтримується компілятором. Цей принцип працює і для посилань, і для покажчиків: замість посилання (покажчика) на базовий клас може бути підставлена посилання (покажчик) на клас-спадкоємець. Зворотне – невірно! Наприклад, спортсмен є людиною, але не кожна людина – спортсмен. Тут людина – базовий клас, а спортсмен – похідний.

Крім конструкторів, не успадковуються два види функцій: операція присвоювання і дружні функції. Дружні функції не успадковуються, оскільки не є методами базового класу, хоча і мають доступ до внутрішньої структурі класу. При відкритому спадкуванні можна не дублювати дружні функції для похідного класу, так як принцип підстановки забезпечує приміщення аргументів похідного класу на місце параметрів базового класу.

Закрите успадкування – це успадкування реалізації: клас реалізований за допомогою класу. Воно принципово відрізняється від відкритого: принцип підстановки не дотримується. Це означає, що не можна привласнити (у всякому разі, без явного перетворення типу) об’єкт похідного класу базового. Тому закрите спадкоємство добре застосовувати в тих випадках, коли потрібно мати функціональність базового класу, але не потрібні ні копіювання, ні присвоювання.

При закритому спадкуванні всі елементи класу-спадкоємця стають приватними і недоступними програмі-клієнту.

Можна відкрити методи базового класу за допомогою using – оголошення, яке має наступний синтаксис:

```
using <ім’я базового класу> :: <ім’я методу в базовому класі>
```

Таким чином, закрите спадкоємство дозволяє обмежити надається похідним класам функціональність.

Множинне успадкування (multiple inheritance) застосовується в тому випадку, коли об’єкт являє собою поняття, що поєднує в собі кілька загальних понять, кожне з яких може бути представлено базовим класом. У випадку використання множинного успадкування клас може походити від кількох базових класів. Наприклад,

```
class X                 // Базовий клас
```

```
{
        . . .
};
class Z            // Базовий клас
{
        . . .
};
class Y : public X, public Z // Похідний клас
{
        . . .
};
```

Множинне успадкування класів незалежно від способів його реалізації є небезпечним з точки зору можливого конфлікту імен, його використання може призвести до виникнення суперечливих і дубльованих даних. Використання множинного успадкування слід уникати в більшості випадків.

3.2. Приклади

Приклад 3.1. Створити композиції класів для роботи з каленадарем. Текст програми:

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cmath>
using namespace std;
class Data
{
    private:
        unsigned int year;
        unsigned int month;
        unsigned int day;
    public:
        unsigned int GetYear(){return this->year;};
        unsigned int GetMonth(){return this->month;};
        unsigned int GetDay(){return this->day;};
        void SetData(unsigned int, unsigned int , unsigned int);
```

```
};
class Calendar
{
    private:
        Data obj;
    public:
        void Input(unsigned int, unsigned int, unsigned int);
        void Output();
        Calendar operator +(int);
        Calendar operator -(int);
        void LeapYear();
        int Dif(Calendar);
        void Change();
        void Comparison(Calendar);
};
int main()
{
    Calendar Date1,Date2;
    unsigned int i;
    int DataMas[3];
    const char Differ[2] = ".";
    char DateChar[20];
    LabString:
    i = 0;
    cout << "\nInput date in yyyy.mm.dd format: ";
    cin >> DateChar;
    char* buf;
    buf = strtok(DateChar,Differ);
    while (buf != 0)
    {
        DataMas[i] = atoi(buf);
        i++;
        buf = strtok(0, Differ);
    }
    if ((DataMas[1] <= 0) || (DataMas[1] > 12) || (DataMas[2] <= 0) ||
(DataMas[2] > 31))
```

```
    {
        cout << "\nFormat error. Proceeding to input...\n";
        goto LabString;
    }
    Date1.Input(DataMas[0],DataMas[1],DataMas[2]);
    LabNumber:
    cout << "\n Input year: yyyy = ";
    cin >> DataMas[0];
    cout << "\n Input month: mm = ";
    cin >> DataMas[1];
    cout << "\n Input day: dd = ";
    cin >>DataMas[2];
    if ((DataMas[1] <= 0) || (DataMas[1] > 12) || (DataMas[2] <= 0) ||
(DataMas[2] > 31))
    {
        cout << "\nFormat error. Proceeding to input...\n";
        goto LabNumber;
    }
    Date2.Input(DataMas[0],DataMas[1],DataMas[2]);
    Date1.Output();
    cout << "\n";
    Date2.Output();
    unsigned int menu = 10;
    while (menu != 0)
    {
        cout << "\nWhat to do..?\n1 - Add\n2 - Sub\n3 - Check Leap
Year\n4 - Change value\n5 - Date comparison\n";
        cout << "6 - Output\n7 - Difference in days\n0 - Exit\n";
        cin >> menu;
        switch (menu)
        {
            case 1:
                {
                    cout <<"\nTo what date add? 1 or 2 ";
                    unsigned int Choose;
                    cin >> Choose;
```

```
                cout << "\nInput amount of days to add to Date: ";
                unsigned int a;
                cin >> a;
                if (Choose == 1)
                Date1 = Date1 + a;
                if (Choose == 2)
                Date2 = Date2 + a;
                break;
            }
        case 2:
            {
                cout <<"\nFrom which date sub? 1 or 2 ";
                unsigned int Choose2;
                cin >> Choose2;
                cout << "\nInput amount of days to sub from Date:
";
                unsigned int s;
                cin >> s;
                if (Choose2 == 1)
                Date1 = Date1 - s;
                if (Choose2 == 2)
                Date2 = Date2 - s;
                break;
            }
        case 3:
            {
                cout <<"\nWhat date check? 1 or 2 ";
                unsigned int Choose3;
                cin >> Choose3;
                if (Choose3 == 1)
                Date1.LeapYear();
                if (Choose3 == 2)
                Date2.LeapYear();
                break;
            }
        case 4:
```

```
                {
                    cout <<"\nWhat date to change? 1 or 2 ";
                    unsigned int Choose4;
                    cin >> Choose4;
                    if (Choose4 == 1)
                    Date1.Change();
                    if (Choose4 == 2)
                    Date2.Change();
                    break;
                }
            case 5:
                {
                    Date1.Comparison(Date2);
                    break;
                }
            case 6:
            {
                Date1.Output();
                cout << "\n";
                Date2.Output();
                break;
            }
            case 7:
                {
                    unsigned int DifM;
                    DifM = Date1.Dif(Date2);
                    cout << "Difference in days = " << DifM << "
days\n";
                    break;
                }
            case 0: exit(0);break;
        }
    }
}
```

Приклад 3.2. Розв’язати задачу 3.1 через успадкування класів. Текст програми:

```
class Data
{
    private:
        unsigned int year;
        unsigned int month;
        unsigned int day;
    public:
        unsigned int GetYear(){return this->year;};
        unsigned int GetMonth(){return this->month;};
        unsigned int GetDay(){return this->day;};
        void SetData(unsigned int, unsigned int , unsigned int);
        virtual void Input(unsigned int, unsigned int, unsigned int)=0;
        virtual void Output()=0;
        void LeapYear();
        void Change();
};

class Calendar: public Data
{
    public:
        void Input(unsigned int, unsigned int, unsigned int);
        void Output();
        void Comparison(Calendar);
        int Dif(Calendar);
        Calendar operator +(int);
        Calendar operator -(int);

};
```

3.2. Задачі

Реалізувати класи завдань лабораторної роботи 1 (свій варіант) з конструкторами, але без перевантаження операцій, в якості базових класів. Реалізувати для базових класів функції введення / виводу як дружні. Реалізувати класи-нащадки, в яких методи базових класів використовуються для реалізації перевантажених операцій. Реалізувати похідні класи в двох варіантах: як відкриті і як закриті класи-нащадки. При відкритому спадкуванні не визначати функції введення /

виводу об’єктів для класів-нащадків.

Задача 3.1

У класі *Vector3d* перевантажити операції у класі-нащадка: додавання і віднімання векторів, порівняння векторів, порівняння довжини вектора.

Задача 3.2

У класі *Money* перевантажити операції у класі-нащадка: додавання і віднімання, операції порівняння.

Задача 3.3

У класі *Triangle* перевантажити операції у класі-нащадка: обчислення площі, обчислення периметра, обчислення висот.

Задача 3.4

У класі *Angle* перевантажити операції у класі-нащадка: переклад в радіани, збільшення і зменшення кута на задану величину, порівняння кутів.

Задача 3.5

У класі *Data* перевантажити операції у класі-нащадка: додавання, віднімання, порівняння дат.

Задача 3.6

У класі *Time* перевантажити операції у класі-нащадка: додавання, віднімання, порівняння часу.

Задача 3.7

У класі *Account* перевантажити операції у класі-нащадка: зняття деякої суми з рахунку, покласти гроші на рахунок.

Задача 3.8

У класі *Fraction* перевантажити операції у класі-нащадка: додавання, віднімання, множення, операції порівняння.

Задача 3.9

У класі *Goods* перевантажити операції у класі-нащадка: зміни ціни товару, обчислення вартості товару.

Задача 3.10

У класі *Payment* перевантажити операції у класі-нащадка: сума, що видається, обчислення стажу.

ЛАБОРАТОРНА РОБОТА 4
ВИНЯТКИ

Мета лабораторної роботи – опанувати написання програм з використанням блоків з винятками.

4.1. Теоретичні основи

Дуже часто функція програми, у якій виникає певна помилка, не має можливості виправити цю помилку, бо невідомим є контекст виклику цієї функції. Помилку необхідно передати до тієї частини програми, у якій її можна обробити. Для вирішення подібних проблем в C++ були введені засоби генерації і обробки виключень (exception).

Для того щоб працювати з винятками, необхідно:

• Виділити контрольований блок – блок try.

• Передбачити генерацію одного або декількох винятків операторами throw всередині блоку try або всередині функцій, що викликаються з цього блоку.

• Розмістити відразу за блоком try один або кілька обробників винятків catch.

Контрольований блок – це складений оператор, перед яким записано ключове слово try:

```
try {
// фрагмент коду
}
```

Оператор `throw`, призначений для генерації виключення, має вигляд throw вираз. Тип виразу, що стоїть після throw, визначає тип породжується виключення. При генерації виключення виконання поточного блоку припиняється, і відбувається пошук відповідного обробника і передача йому управління.

Синтаксис оброблювачів нагадує визначення функції з одним параметром і ім’ям catch:

```
catch (/ * ... V) {
// дії по обробці виключення
```

Оголошення параметра обробника можливо в одній з трьох форм:

```
catch (Type) {// обробка виключення типу Type}
catch (Type info) {
// обробка виключення типу Type
```

```
// з використанням значення info
}
catch (...) {// обробка винятків всіх типів}
```

Після обробки виключення управління передається першому оператору, що знаходиться безпосередньо за обробниками винятків. Туди ж, минаючи код всіх обробників, передається керування, якщо виключення в try-блоці не було згенеровано.

Якщо обробник не в змозі повністю обробити помилку, він може згенерувати виключення повторно за допомогою оператора throw без параметрів. У цьому випадку передбачається наявність зовнішніх блоків, в яких може перебувати інший обробник для цього типу виключення.

Список винятків функції є частиною її заголовку. Його треба наводити в усіх оголошеннях функції.

Для винятків можна застосовувати фундаментальні типи даних, але кращий підхід базується на створенні власних типів, оскільки розпізнавання винятків базується на типах. Найчастіше це нові класи, які в окремому випадку можуть бути порожніми.

На відміну від багатьох інших мов програмування, C++ не вимагає, щоб усі класи винятків походили з одного базового класу. Проте іноді доцільно створювати власні ієрархії винятків:

```
class Error
{
  // ...
};

class MathError : public Error
{
  // ...
};
```

Оброблювач базового типу обробляє також винятки похідних типів:

```
...
catch (Error)
{
  // обробляє виняток типу Error та MathError
}
```

В першу чергу перехоплюються більш специфічні винятки, далі винятки базових класів, наостанку слід розміщувати оброблювач за умовчанням. Наприклад,

```
...
catch (MathError)
{
   // обробляє виняток типу MathError
}
catch (Error)
{
   // обробляє виняток типу Error
}
catch (...)
{
   // обробляє всі інші винятки
}
```

Однією з основних проблем використання фундаментальних типів даних (наприклад, типу int) в якості типів винятків є те, що вони, по своїй суті, є невизначеними. Ще серйознішою проблемою є неоднозначність того, що означає виняток, коли в блоці try є кілька стейтментів або викликів функцій:

Одним із способів вирішення цієї проблеми є використання класів-винятків. Клас-виняток — це звичайний клас, який генерується в якості винятку. Використовуючи такий клас, ми можемо генерувати виняток, який повертає опис виниклої проблеми. Зверніть увагу, в обробниках винятків об’єкти класу-винятку приймати потрібно по посиланню, а не по значенню. Це запобіжить створенню копії винятку компілятором, що є витратною операцією (особливо, коли виняток є об’єктом класу), і запобіжить обрізці об’єктів при роботі з дочірніми класами-винятками. Передачу по адресі краще не використовувати, якщо у вас немає на це вагомих причин.

Оскільки ми можемо генерувати об’єкти класів в якості винятків, а класи можуть бути отримані з інших класів, то нам потрібно враховувати, що станеться, якщо ми будемо використовувати успадковані класи в якості винятків. Виявляється, обробники можуть обробляти винятки не тільки одного певного класу, а й винятки дочірніх йому класів!

По-перше, як ми вже говорили, дочірні класи можуть бути спіймані обробником батьківського класу. По-друге, коли C++ намагається знайти обробник для згенерованого винятку, він робить це послідовно.

Правило: Обробники винятків дочірніх класів повинні знаходитися перед обробниками винятків батьківського класу.

Багато класів та операторів зі Стандартної бібліотеки C++ генерують класи-винятки при збої. Наприклад, оператор `new` і `std::string` можуть генерувати `std::bad_alloc` при нестачі пам'яті. Невдале динамічне приведення типів за допомогою оператора dynamic_cast генерує виняток `std::bad_cast` тощо. Починаючи з C++14, існує більше 20 класів-винятків, які можуть бути згенеровані, а в C++17 їх ще більше.

Хорошою новиною є те, що всі ці класи-винятки є дочірніми класу `std::exception`. `std::exception` — це невеликий інтерфейсний клас, який використовується в якості батьківського класу для будь-якого винятку, який генерується в Стандартній бібліотеці C++.

У більшості випадків, якщо виняток генерується Стандартною бібліотекою C++, то нам все рівно, чи було це невдале виділення, конвертування чи що-небудь інше. Нам досить знати, що сталося щось катастрофічне, через що в нашій програмі стався збій. Завдяки std::exception ми можемо налаштувати обробник винятків типу std::exception, який ловитиме і оброблятиме як std::exception, так і всі (20+) дочірні йому класи-винятки!

Іноді нам потрібно буде обробляти певний тип винятків трохи інакше, ніж інші типи винятків. У такому випадку ми можемо додати обробник винятків для цього конкретного типу, а всі інші винятки «перенаправити» в батьківський обробник.

Винятки типу `std::exception` і всіх інших дочірніх йому класів-винятків обробляються другим обробником. Такі ієрархії спадкування дозволяють використовувати певні обробники для перехоплення певного типу винятків або для перехоплення одним (батьківським) обробником всієї ієрархії винятків. Звичайно, ви можете створити свої власні класи-винятки, дочірні класу `std::exception`, і перевизначити віртуальний константний метод `what()`.

4.2. Приклади

Приклад 4.1. Обробка помилок з використанням винятків:

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

int main()
{
    try
    {
        throw - 1;
    }
    catch (int a)
    {
        cout << "Int exception with value: " << a << '\n';
    }
    catch (double)
    {
        cout << "Exception of type double" << '\n';
    }
    catch (const std::string& str)
    {
        cout << "Exception of type string" << '\n';
    }

    cout << "Continuing our way!\n";

    return 0;
}
```

Приклад 4.2. Для більш зручної обробки помилок можна описати більш складні класи винятків:

```
class MyClass
{
public:
    class Bad_Data
    {
        int bad_value;
    public:
        Bad_Data(int value) : bad_value(value) {}
        int getBadValue() const { return bad_value; }
    };
    void f(int i)
```

```
    {
        if (i < 0)
            throw Bad_Data(i);
    }
};
void main()
{
    try
    {
        MyClass m;
        m.f(-2);
    }
    catch (MyClass::Bad_Data b)
    {
        cout << "Bad value: " << b.getBadValue();
    }
}
```

4.3. Задачі

Реалізувати обробку винятків в трьох варіантах:

– використовувати відповідні стандартні винятки;

– використовувати винятки-нащадки від стандартних винятків;

– визначити власні винятки.

Передачу об’єкту винятку виконати різними способами: за значенням, за посиланням, за вказівником.

Виконати завдання лабораторної роботи 1 (свій варіант).

Задача 4.1

Клас *Vector3d* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.2

Клас *Money* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.3

Клас *Triangle* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.4

Клас *Angle* з конструкторами, перевантаженням операцій і

обробкою винятків.

Задача 4.5

Клас *Data* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.6

Клас *Time* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.7

Клас *Account* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.8

Клас *Fraction* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.9

Клас *Goods* з конструкторами, перевантаженням операцій і обробкою винятків.

Задача 4.10

Клас *Payment* з конструкторами, перевантаженням операцій і обробкою винятків.

ЛАБОРАТОРНА РОБОТА 5
ВІРТУАЛЬНІ ТА АБСТРАКТНІ КЛАСИ

Мета лабораторної роботи – навчитися будувати віртуальні та абстрактні класи.

5.1. Теоретичні основи

Поліморфізм (polymorphism) – це механізм визначення серед функцій з однаковими іменами функції для виклику, заснований на типі параметрів (поліморфізм часу компіляції) або об’єкта, для якого викликається метод (поліморфізм часу виконання). Поліморфізм часу виконання у C++ використовує ієрархії успадкування.

Зв’язування – це зіставлення виклику функції з тілом. Для звичайних методів зв’язування виконується на етапі трансляції до запуску програми. Таке зв’язування називають «раннім», або статичним. При спадкуванні звичайного методу його поведінка не змінюється в спадкоємця. Однак буває необхідно, щоб поведінка деяких

методів базового класу і класів-спадкоємців відрізнялися. Щоб домогтися різного поведінки в залежності від типу, необхідно оголосити функцію-метод віртуальної; в C++ це робиться за допомогою ключового слова virtual. Віртуальні функції в сукупності з принципом підстановки забезпечують механізм «пізнього» (відкладеного) або динамічного зв'язування, яке працює під час виконання програми.

Клас, в якому визначені віртуальні функції (хоча б одна), називається поліморфним класом.

Ключове слово virtua1 можна писати тільки в базовому класі – це досить зробити в оголошенні функції.

```
class SomeClass
{
        . . .
    public:
        virtual void firstFunc();
        virtual int secondFunc(int x);
};
```

Правила опису та використання віртуальних функцій-методів наступні.

1. Віртуальна функція може бути тільки методом класу.

2. Будь-яку перевантажували операцію-метод класу можна зробити віртуальної, наприклад операцію присвоювання або операцію перетворення типу.

3. Віртуальна функція успадковується.

4. Віртуальна функція може бути константної.

5. Якщо в базовому класі визначена віртуальна функція, то метод похідного класу з таким же ім'ям і прототипом (включаючи і тип значення, що повертається, і константність методу) автоматично є віртуальним і заміщає функцію-метод базового класу.

6. Статичні методи не можуть бути віртуальними.

7. Конструктори не можуть бути віртуальними.

8. Деструктори можуть (частіше – повинні) бути віртуальними – це гарантує коректне звільнення пам'яті через покажчик базового класу.

Усередині конструкторів і деструкторів динамічне зв'язування не працює, хоча виклик віртуальних функцій не заборонений. У конструктори і деструкції завжди викликається «рідна» функція класу.

Віртуальні функції-методи можна перевантажувати і

перевизначити (у спадкоємців) з іншим списком аргументів. Якщо віртуальна функція перевизначена з іншим списком аргументів, вона заміщає (приховує) батьківські методи.

Віртуальна функція, яка не має визначення тіла, називається чистою (pure) і оголошується наступним чином:

```
virtual тип ім’я (параметри) = 0;
```

Передбачається, що дана функція буде реалізована в класах-спадкоємців. Клас, в якому є хоч одна чиста віртуальна функція, називається *абстрактним класом*.

```
class AbstractBaseClass
{
    public:
    virtual void f() = 0;
    virtual int secondFunc(int x);
};
```

Об’єкти абстрактного класу створювати заборонено навіть операцією new при передачі параметра в функцію неможливо передати об’єкт абстрактного класу за значенням. Однак покажчики (і посилання) визначати можна (згадайте оголошення класу).

При спадкуванні абстрактність зберігається: якщо клас-спадкоємець не реалізує успадковану чисту віртуальну функцію, то він теж є абстрактним. В C ++ абстрактний клас визначає поняття інтерфейсу. Спадкування від абстрактного класу – це успадкування інтерфейсу.

5.2. Приклади

Приклад 5.1. Розглянемо абстрактний базовий клас `figure` (фігура), на базі якого можна побудувати похідні класи для реальних фігур (еліпс, коло, квадрат, ромб, прямокутник, трикутник і т.п.). На лістингу приведений базовий клас figure і похідні класи _circle (окружність) і RecTangle (прямокутник).

```
#include <iostream>
#include <math>
#define PI 3.14159
using namespace std;
class figure
{
```

```
     public:
     int n;
     float *p;
     figure();
     float perimetr();
     virtual float square();
     virtual void show_parametri();
};
figure::figure() {cout<<"This is abstract constructor"<<endl;}
float figure::perimetr()
{
     int i;
     float psum;
     for (psum = 0, i = 0; i <n; psum + = p [i], i ++);
     return psum;
}
float figure::square()
{
     cout<<"No square abstract figure"<<endl;
     return 0;
}
void figure::show_parametri() {cout<<"Abstract figure";}
class _circle: public figure
{
     public:
     _circle();
     float perimetr();
     virtual float square();
     virtual void show_parametri();
};
class RecTangle: public figure
{
     public:
     RecTangle();
     virtual float square();
     virtual void show_parametri();
```

```
};
void main()
{
     _circle RR;
     RR.show_parametri();
     RecTangle PP;
     RR.show_parametri();
}
_circle::_circle()
{
     cout<<"Parametri okruzhnosti"<<endl;
     n = 1;
     p = new float[n];
     cout<<"Vvedite radius";
     cin>>p[0];
}
float _circle::perimetr()
{
     return 2*PI*p[0];
}
float _circle::square()
{
     return PI*p[0]*p[0];
}
void _circle::show_parametri()
{
     cout<<"This is circle"<<endl;
     cout<<"Radius ="<<p [0]<<endl;
     cout<<"Perimetr ="<<perimetr()<<endl;
     cout<<"Square ="<<square()<<endl;
}
RecTangle::RecTangle()
{
     cout<<"Parametri rectangle"<<endl;
     n = 4;
     p = new float[n];
```

```
    cout<<"Vvedite dlini storon";
    cin>>p[0]>>p[1];
    p[2] = p[0];
    p[3] = p[1];
}
float RecTangle::square()
{
    return p[0]*p[1];
}
void RecTangle::show_parametri()
{
    cout<<"This is Rectangle"<<endl;
    cout<<"a =" << p[0] <<"b ="<<p[1]<<endl;
    cout<<"Perimetr ="<<perimetr()<<endl;
    cout<<"Square ="<<square()<<endl;
}
```

5.3. Задачі

У наступних завданнях потрібно реалізувати абстрактний базовий клас, визначивши в ньому чисті віртуальні функції. Ці функції визначаються в похідних класах. В базових класах повинні бути оголошені чисті віртуальні функції введення / виводу, які реалізуються в похідних класах.

Програма, яка викликає, повинна продемонструвати всі варіанти виклику віртуальних функцій за допомогою покажчиків на базовий клас. Написати функцію виведення, яка одержує параметри базового класу по посиланню і демонструє віртуальний виклик.

Задача 5.1

Створити абстрактний базовий клас Figure з віртуальними методами обчислення площі та периметра. Створити похідні класи: Rectangle (прямокутник), *Trapezium* (трапеція) зі своїми функціями площі і периметра. Самостійно визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних.

Задача 5.2

Створити абстрактний базовий клас *Number* з віртуальними методами – арифметичними операціями. Створити похідні класи *Integer* (ціле) і *Real* (дійсне).

Задача 5.3

Створити абстрактний базовий клас *Body* (тіло) з віртуальними функціями обчислення площі поверхні і об’єму. Створити похідні класи: *Parallelepiped* (паралелепіпед) і *Ball* (куля) зі своїми функціями площі поверхні і об’єму.

Задача 5.4

Створити абстрактний клас *Currency* (валюта) для роботи з грошовими сумами. Визначити віртуальні функції перекладу в рублі і виведення на екран. Реалізувати похідні класи *Dollar* (долар) і *Euro* (євро) зі своїми функціями перекладу і виведення на екран.

Задача 5.5

Створити абстрактний базовий клас *Triangle* для подання трикутника з віртуальними функціями обчислення площі та периметра. Поля даних повинні включати дві сторони і кут між ними. Визначити класи-нащадки: прямокутний трикутник, рівнобедрений трикутник, рівносторонній трикутник зі своїми функціями обчислення площі та периметра.

Задача 5.6

Створити абстрактний базовий клас *Root* (корінь) з віртуальними методами обчислення коренів і виведення результату на екран. Визначити похідні класи *Linear* (лінійне рівняння) і *Square* (квадратне рівняння) з власними методами обчислення коренів і виведення на екран.

Задача 5.7

Створити абстрактний базовий клас *Function* (функція) з віртуальними методами обчислення значення функції в заданій точці і виведення результату на екран. Визначити похідні класи *Ellipse* (еліпс), *Hyperbola* (гіпербола) з власними функціями обчислення у залежно від вхідного параметра.

Задача 5.8

Створити абстрактний базовий клас *Integer* (ціле) з віртуальними арифметичними операціями і функцією виведення на екран. Визначити похідні класи *Decimal* (десяткове) і *Binary* (двійкове), що реалізують власні арифметичні операції і функцію виведення на екран. Число представляється масивом, кожен елемент якого – цифра.

Задача 5.9

Створити абстрактний базовий клас *Series* (прогресія) з

віртуальними функціями обчислення і-го елемента прогресії і суми прогресії. Визначити похідні класи: *Linear* (арифметична) і *Exponential* (геометрична).

Задача 5.10

Створити абстрактний базовий клас *Container* з віртуальними методами *sort*() і поелементної обробки контейнера *foreach*. Розробити похідні класи *Bubble* (бульбашка) і *Choice* (вибір). У першому класі сортування реалізується методом бульбашки, а поелементна обробка полягає в добуванні квадратного кореню. В другом класі сортування реалізується методом вибору, а поелементна обробка обчисленні логарифму.

ЛАБОРАТОРНА РОБОТА 6
КОНТЕЙНЕРИ

Мета лабораторної роботи – опанувати написання програм з набором однотипних елементів.

6.1. Теоретичні основи

Контейнер – це набір однотипних елементів. Вбудовані масиви в C++ – окремий випадок контейнера. Рядок символів – ще один приклад контейнера.

Ім’я контейнера – це ім’я змінної в програмі, яке підкоряється правилам видимості C++. Будучи об’єктом, контейнер має час життя в залежності від місця і часу створення; час життя контейнера в загальному випадку не залежить від тривалості існування його елементів.

Тип контейнера складається з типу самого контейнера і типу входять до нього елементів. Тип контейнера – це не тип його елементів. Тип елементів може бути або вбудованим, або реалізованим; елементами контейнера можуть бути контейнери.

Контейнери можуть бути фіксованого розміру і змінного розміру. У контейнері фіксованого розміру, кількість елементів постійне, воно зазвичай задається при створенні контейнера. Прикладом контейнера фіксованої довжини є масив. Для контейнера змінного розміру кількість елементів при оголошенні зазвичай не задається. Елементи в такому контейнері додаються і видаляються під час роботи програми.

Якщо елементи контейнера ніяк не впорядковані, то додавання і видалення елементів зазвичай виконується на початку або в кінці контейнера. Спосіб вставки і видалення визначає вид контейнера. Якщо вставка і видалення виконуються тільки на одному кінці, такий контейнер називається стеком. Кажуть, що стек працює відповідно до дисципліною LIFO (Last In First 0ut – останнім прийшов, першим пішов). Якщо елементи додаються на одному кінці контейнера, а видаляються з іншого кінця, такий контейнер називається чергою. Черга працює за принципом FIFO (First In First 0ut – першим прийшов, першим пішов). Можна виконувати і вставку, і видалення на обох кінцях контейнера такий контейнер називається двосторонньої чергою (від англійського терміна deque, абревіатури від «double ended queue»).

Якщо контейнер будь-яким чином впорядкований, операція вставки працює відповідно. з порядком елементів в контейнері. Операція видалення при цьому може виконуватися по-різному: на початку, в кінці контейнера або заданого елемента.

Серед всіх операцій з контейнерами можна виділити кілька типових груп:

– операції доступу до елементів, які забезпечують і операцію заміни значень елементів;

– операції додавання і видалення окремих елементів або груп елементів;

– операції пошуку елементів і груп елементів;

– операції об’єднання контейнерів; інші (спеціальні) операції, що залежать від виду контейнера.

Найважливішими є операції доступу і операції об’єднання контейнерів.

Доступ до елементів контейнера буває: послідовний, прямий, асоціативний. Прямий доступ до елементу – це доступ за номером (за індексом) елемента. Нумерація елементів може починатися з будь-якого числа, проте в C ++ прийнято нумерацію починати з нуля, аналогічно масивів.

Асоціативний доступ теж виконується за індексом, однак індексом є не номер елемента, а його вміст. Асоціативний контейнер впорядкований певним чином за ключем.

При послідовному доступі здійснюється переміщення від елемента до елементу контейнера. Можна вважати, що існує якийсь

об’єкт, який перебирає елементи контейнера за допомогою деякого безлічі операцій. Той елемент, на якому в даний момент об’єкт позиціонується, називається поточним елементом.

У більш загальному випадку об’єкт, який перебирає елементи контейнера, називається ітератором. Ітератор – це об’єкт, що забезпечує послідовний доступ до елементів контейнера. Ітератор може бути реалізований як частина класу-контейнера у вигляді набору методів.

```
v.first(); // Перейти до першого
v.last (); // Перейти до останнього.
v.next (); // Перейти до наступного
v.prev (); // Перейти до попереднього
v.skip (n); // Перейти на n елементів вперед
v.skip (-n); // перейти на n елементів назад
v.current (); // отримати поточний
```

Найчастіше ітератор реалізується як клас, що надає той же набір операцій. У практиці програмування на C++ ітератор реалізується з інтерфейсом покажчика (для сумісності з масивами).

Контейнери реалізуються за допомогою динамічної пам’яті. Виділення пам’яті здійснюють конструктори класу-контейнера при створенні об’єкта-контейнера. Зазвичай в класі реалізується кілька перевантажених конструкторів. Пам’ять можна виділити за допомогою операції new, яка має дві форми: а виділення масиву однотипних елементів new[]; а виділення одиночного елемента new.

Виділення пам’яті операцією new[] забезпечує надання безперервної області пам’яті. Кількість елементів можна задавати виразом, який обчислюється під час роботи програми. Така форма практично завжди використовується для реалізації динамічних масивів. В цьому випадку в класі обов’язково задається поле-покажчик, який і отримує адресу динамічного масиву. Зазвичай задається також поле для зберігання розміру виділеної пам’яті. Якщо кількість елементів динамічного масиву фіксоване, то це ж поле визначає кількість елементів контейнера-масиву. Якщо ж кількість елементів змінюється під час роботи програми, то в класі зазвичай присутній ще і поле для зберігання поточного кількості елементів.

Для динамічних контейнерів обов’язкові реалізація конструктора копіювання і операції привласнення. Конструктор копіювання,

створюваний за замовчуванням, виконує поелементне копіювання полів класу. Конструктор копіювання повинен виконувати інші дії: запросити пам'ять для нового контейнера такого ж розміру і скопіювати в новий контейнер елементи не започатковано контейнера.

Чисто послідовний контейнер зазвичай реалізується як пов'язаний список елементів (вузлів). Виділення пам'яті виконується для кожного елементу окремо операцією new. Прямий доступ відсутній; послідовний доступ забезпечується за допомогою ітераторів, причому як в прямому, так і в зворотному напрямку. Ітератор реалізується або як набір методів класу-контейнера, або як вкладений клас. Вставка виконується на початку контейнера, в кінці контейнера і після елемента, на який вказує ітератор. Видалення елементів зазвичай реалізується в двох варіантах: з ітератором, коли видаляється елемент, на який вказує ітератор, і за значенням, коли видаляється елемент, який має задане значення. Часто реалізуються операції пошуку елементів і різні варіанти об'єднання контейнерів. Операції порівняння контейнерів і введення / виведення, як зазвичай, реалізуються зовнішніми дружніми функціями.

6.2. Приклади

Приклад 6.1. Числовий зростаючий масив.

```
class Array
{
public:
    typedef double  value_type;
    typedef double*  iterator;
    typedef const double*  const_iterator;
    typedef double&  reference;
    typedef const double*  const_reference;
    typedef std::size_t  size;
    Array (const size _type& n=minsize);
    Array (iterator first, iterator last);
    Array (const Array & array);
    ~ Array();
    Array& operator=( const Array&);
    iterator begin() { return elems;}
    iterator end() { return elems+Count;}
    const_iterator begin() const { return elems;}
```

```
    const_iterator end() const { return elems+Count;}
    bool empty() const;
    size_type capacity() const;
    void resize(size_type newsize);
    reference operator[](size_type);
    const_reference operator[](size_type) const;
    reference front(){return elems[0];};
    const_reference front() const {return elems[0];};
    reference back(){return elems[size()-1];};
    const_reference back() const {return elems[size()-1];};
    void push_back(const value_type& v);
    void pop_back();
    void clear() {Count=0;};
    void swap(Array& other);
    void assign(const value_type& v);
private:
    static const size_type minsize=10;
    size_type Size;
    size_type Count;
    value _type *elems;
};
```

Приклад 6.2. Послідовний контейнер-список.

```
class List
{
public:
    typedef double  value_type;
    typedef std::size_t  size;
    class iterator;
    List(const value_type& a, size n=1);
    List(iterator, iterator);
    List(const List& x);
    ~List();
    List& operator=(const List&);
    iterator begin() {return head;}
    iterator end() {return tail;}
    iterator begin() const {return head;}
```

```
    iterator end() const {return tail;}
    bool empty() const {return (Head==Tail);};
    size length() const {return count;};
    value_type& front(){return *begin;};
    value_type& back(){iterator it=end(); --it; return *it;};
    value_type& find(const value_type& a);
    void push_front(const value_type&);
    void pop_front();
    void push_back(const value_type&);
    void pop_back();
    void insert(iterator, const value_type&);
    void erase(iterator);
    void erase(iterator, iterator);
    void remove(const value_type&);
    void swap(List &);
    void clear(List &);
    void splice(List &);
    void splice(iterator, List &);
    void sort();
    void splice(List &);
private:
    struct Node
     { Node(const value_type& a) {}
        Node() {}
       value_type item;
       Node *next;
       Node *prev;
      };
      long count;
      Node *Head;
      Node *Tail;
     iterator head, tail;
};
```

6.3. Задачі

Наступні завдання виконати двома способами:

– з використанням масивом, з виділенням пам’яті, розмір заздалегідь не відомий. Елементи можуть вставлятися в будь-яке місце масиву; може бути видалений будь-який елемент або група елементів (див. лістинг прикладу 7.1). Реалізувати операцію індексування.

– з використанням двозв’язного списку (див. лістинг прикладу 7.2). Замість операції індексування реалізувати ітератор як набір методів списку.

Для роботи з контейнерами реалізувати методи: додавання та видалення елементу, вводу та виводу всього контейнеру.

Задача 6.1

Створити клас «Відділ кадрів». Поля: прізвище співробітника, ім’я, по батькові, посаду, стаж роботи, оклад.

Задача 6.2

Створити клас «Червона книга». Поля: вид тварини, рід, сімейство, місце проживання, чисельність популяції.

Задача 6.3

Створити клас «Бібліотека». Поля: автор книги, назва, рік видання, код УДК, ціна, кількість в бібліотеці.

Задача 6.4

Створити клас «Супутники планет». Поля: назву, назву планети-господаря, рік відкриття, діаметр, період обертання.

Задача 6.5

Створити клас «Радіодеталі». Поля: позначення, тип, номінал, кількість на схемі, позначення можливого замінника.

Задача 6.6

Створити клас «Побут студентів». Поля: прізвище студента, ім’я, по батькові, факультет, розмір стипендії, число членів сім’ї.

Задача 6.7

Створити клас «Міський транспорт». Поля: вид транспорту, номер маршруту, початкова зупинка, кінцева зупинка, час у дорозі.

Задача 6.8

Створити клас «Спортивні змагання». Поля: прізвище спортсмена, ім’я, команда, вид спорту, заліковий результат, штрафні очки.

Задача 6.9

Створити клас «Оптова база». Поля: назву товару, кількість на

складі, вартість одиниці, назва постачальника, термін поставки.

Задача 6.10

Створити клас «Зоопарк». Поля: вид тварини, кличка, вік, категорія рідкості, вага, добовий раціон м’яса, овочів, молока.

ВИМОГИ ДО ЗВІТУ З ЛАБОРАТОРНОЇ РОБОТИ

Звіт з лабораторної роботи повинин включати наступне:

1. Титульний лист.
2. Постанова задачі.
3. Опис програми (класів, методів).
4. Керівництво користувача(скріни працюючої програми).
5. Висновки.
6. Додаток (код програми).

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1 Нікуліна О.М., Коцюба Н. В. Об’єктно-орієнтоване програмування мовою С++: методичні вказівки до лаб. занять з курсу «Об’єктно-орієнтоване програмування» для студентів спеціальностей 122 – Комп’ютерні науки, 126 – Інформаційні системи та технології. Х. : НТУ «ХПІ», 2022. – 68 с.

2 Schildt H. C++: The Complete Reference: 4th Edition, McGraw-Hill Education, 2002, 1056 p.

3 Трофименко О.Г. С++. Алгоритмізація та програмування : підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. – Одеса : Фенікс, 2019. – 477 с.

4 Пекарський Б.Г. Основи програмування: Навчальний посібник. Кондор, 2018. - 364 с.

5 Джордж Хайнеман, Гері Полліс, Стенлі Селков. Алгоритми. Довідник з прикладами на C, C++, Java і Python. - Діалектика, 2017. 432 с.

6 Грицюк Ю.І., Рак Т.Є. Програмування мовою С++: навчальний посібник. – Львів : Вид-во Львівського ДУ БЖД, 2011. – 292 с.

Web-ресурси

7 Уроки програмування на С++. https://acode.com.ua/uroki-po-cpp/

8 Основи програмування. Розробник курсу Л.В. Іванов. http://www.iwanoff.inf.ua/programming_2_ua/index.html

9 C/C++ language and standard libraries reference // https://msdn.microsoft.com/en-us/library/hh875057.aspx