МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Волинський національний університет імені Лесі Українки

Кафедра комп’ютерних наук та кібербезпеки

Т.О. Гришанович, Л.Я. Глинчук

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

Навчальний посібник

Луцьк 2022

УДК 004.4(075.8)
Г 85

Рекомендовано до друку вченою радою
Волинського національного університету імені Лесі Українки
(протокол №4 від 31.03.2022 р.)

Рецензенти:

Чернящук Н. Л. - доктор пед. наук, професор, завідувач кафедри комп'ютерної інженерії та кібербезпеки Луцького національного технічного університету

Ліщина Н. М. – кандидат техн. наук, доцент, завідувач кафедри інженерії програмного забезпечення Луцького національного технічного університету

Хомяк М. Я. – кандидат фіз.-мат. наук, доцент, завідувач кафедри загальної математики та методики навчання інформатики Волинського національного університету імені Лесі Українки

Основи об'єктно-орієнтованого програмування : навч. посібник / Гришанович Т. О., Глинчук Л. Я.; ВНУ імені Лесі Українки. Електронні текстові данні (1 файл: 998 КБ). Луцьк : ВНУ імені Лесі Українки, 2022. – 120 с.

Посібник призначено для тих, хто прагне оволодіти технікою об'єктно-орієнтованого програмування на основі мови С++. Використання даної книги передбачає, що читач уже знайомий із основами програмування на мові С++, вміє використовувати алгоритмічні конструкції та структури даних, знайомий із базовими алгоритмами програмування.
Посібник містить теоретичні відомості із об'єктно орієнтованого програмування, починаючи із поняття класу та об'єкту, закінчуючи множинним наслідуванням. До кожної теми наведено питання для самоконтролю. Також посібник містить набір завдань для практичного виконання. Ці завдання можуть бути використані для практикування на будь-якій мові програмування, що підтримує об'єктно-орієнтовану парадигму.

Зміст

Тема 1
Вступ до об’єктно-орієнтованого програмування. Поняття класу та об’єкту

Необхідність використання об’єктно-орієнтованого підходу зумовлена бар’єром можливостей технологій процедурного програмування, що виник у 60-х роках XX століття. Причиною цієї кризи стало значне зростання складності та розміру програмних систем. При використанні процедурного підходу ускладнюється структура програми та можливість її модифікації. Це пов’язано із зростанням числа глобальних змінних, функцій та зв’язків між ними. Крім того, процедурний підхід розділяє дані та методи їх опрацювання , що не відповідає картині реального світу, що складається із об’єктів, які одночасно характеризуються властивостями (даними) та поведінкою (діями). Логічне об’єднання даних та операцій над ними є головною ідеєю об’єктно-орієнтованої методології, що забезпечила подолання цих труднощів.

Об’єктно-орієнтоване проектування полягає у формуванні класів, об’єктів, та відношень між ними на основі характеристик (атрибутів) та дій (операцій).

Визначальною ідеєю об’єктно-орієнтованого підходу до розробки сучасних програмних продуктів є поєднання даних і дій, що виконуються над ними, в єдине ціле, яке називають об’єктом.

Функції об’єкта, які у мові програмування C++ називають методами або функціями-членами класу, зазвичай призначені для доступу до даних об’єкта та виконання певних дій над ними. Наприклад, якщо необхідно зчитувати будь-які значення даних об’єкта, то потрібно викликати відповідну функцію, яка їх зчитає та поверне об’єкту. Зазвичай прямий доступ до даних є неможливим, тому вони приховані від зовнішніх дій, що захищає їх від випадкових змін. Вважають, що дані та методи класу між собою інкапсульовані. Терміни приховання та інкапсуляція даних є ключовими в описі об’єктно-орієнтованих мов програмування.

Зв’язок між даними та методами їх обробки в об’єктно-орієнтованому програмуванні можна зобразити за допомогою наступної схеми:

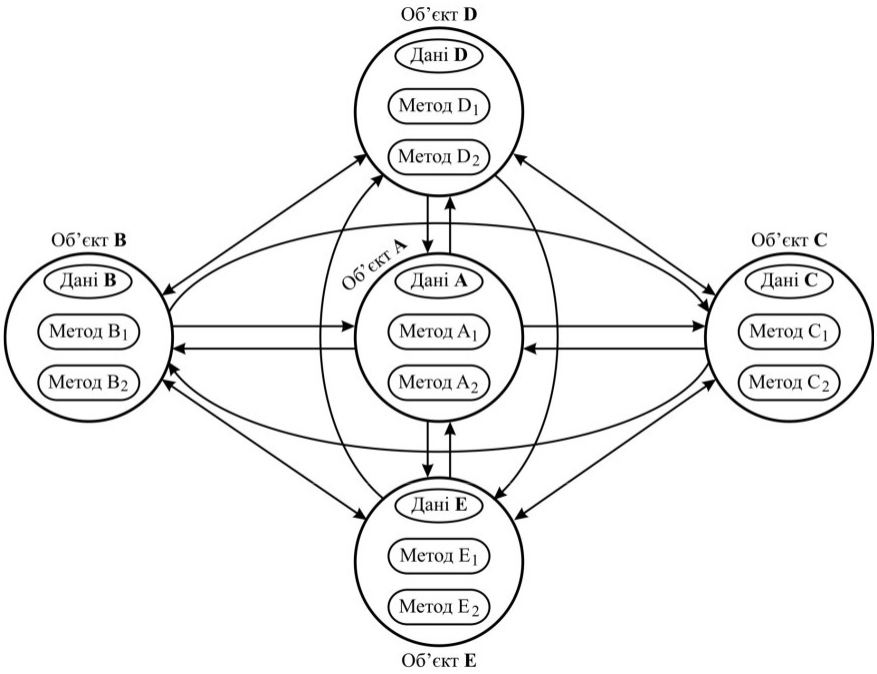

Рисунок 1 - Зв’язок між даними та методами їх обробки в об’єктно-орієнтованому програмуванні

Об’єктно-орієнтоване програмування ґрунтується на трьох основних принципах:

- інкапсуляція даних та функцій (методів)їх опрацювання в єдиній інформаційній структурі, що називають класом:
- імпорт властивостей базових класів у похідні класи - успадкування класів;
- поліморфізм, який полягає у використанні однакових інтерфейсів для роботи з різними за функціональністю об’єктами.

Об’єктно-орієнтоване програмування ґрунтується на пріоритеті даних над кодом, на передачі повідомлень об’єктам за допомогою викликів відповідних функцій. На відміну від процедурного програмування, в об’єктно-орієнтованому підході не код керує даними, а дані керують кодом програми.

Об’єктно-орієнтований підхід до розробки програмних продуктів базується на такому понятті як клас. Клас визначає новий тип даних, який задає формат об’єкта. Клас містить як дані, так і коди програм, призначені для виконання дій над ними. Загалом, клас пов’язує дані з кодами програми. У мові програмування С++ специфікацію класу використовують для побудови об’єктів. Об’єкти – це примірники класового типу. Загалом, клас є набором планів і дій, які вказують на властивості об’єкта та визначають його поведінку. Важливо розуміти, що клас – це логічна абстракція, яка реально не існує доти, доки не буде створено об’єкт цього класу, тобто це те, що стане фізичним представленням цього класу в пам’яті комп’ютера.

Визначаючи клас, оголошують ті глобальні дані, які міститимуть об’єкти, і програмні коди, які виконуватимуться над цими даними. Хоча інколи прості класи можуть містити тільки програмні коди або тільки дані, проте більшість реальних класів містять обидва компоненти. У класі дані оголошуються у вигляді змінних, а програмні коди оформляють у вигляді функцій. Змінні та функції називаються членами класу. Змінну, оголошену в класі, називають членом даних, а оголошену в класі функцію – функцією-членом класу. Іноді замість терміну член даних класу використовують термін змінна класу, а замість терміну функція-член класу використовують термін метод класу.

Тобто клас (class) – це структурований тип даних, що інкапсулює (приховує, об’єднує, обмежує область досяжності) оголошення полів даних (атрибутів) та функцій для їх опрацювання. Функції класу називають методами. Дані, які мають класовий тип, називають об’єктами або екземплярами класу.

Клас використовують для означення множини об’єктів, які мають однакову структуру, поведінку та відношення між об’єктами класів.

Структура (зміст) класу називається його протоколом.

Дані та методи класу захищені оголошенням класу.

class *ім’я_класу* {

private:

дані та методи класу

public:

дані та методи класу

protected:

дані та методи класу

}

C++ надає даним та методам 3 рівні захисту:

- закриті (використовуються за замовчуванням) – private – можуть використовуватись тільки у межах класу;
- відкриті – public – досяжні у класі та поза класом у межах дії встановленого простору імен;
- захищені – protected – доступні у самому класі та похідних від нього.

Дозволяється довільний порядок розміщення розділів private, public та protected у протоколі класу.

Оголошення класу синтаксично є подібним до оголошення структури. Загальні формати оголошення класу мають такий вигляд:

Варіант А

class *ім’я_класу* {

private:

закриті дані та функції класу

public:

відкриті дані та функції класу

} *перелік_об’єктів_класу*;

Варіант B

class ім’я_класу {

private:

закриті дані та функції класу

public:

відкриті дані та функції класу

};

ім’я_класу перелік_об’єктів_класу;

У цих оголошеннях елемент *ім’я_класу* означає ім’я "класового" типу. Воно стає іменем нового типу, яке можна використовувати для побудови об’єктів цього класу. Об’єкти класу інколи створюють шляхом вказання їх імен безпосередньо за закритою фігурною дужкою оголошеного класу як елемент *перелік_об’єктів_класу* (варіант А). Проте найчастіше об’єкти створюють за потребою після оголошення класу (варіант В).

Наприклад, наведений нижче клас під іменем ClassPoint визначає тип майбутнього об’єкта, призначеного для зберігання координат точки на площині:

```
class ClassPoint {  // оголошення класового типу
     double x;
     double y;
     public:
        void Init();// ініціалізація даних класу ClassPoint
        void Set(double, double);   // зміна значень даних об’єкта класу
        void Get(); // виведення з об’єкта значення
};
```

Розглянемо детально механізм визначення цього класу. Усі члени класу ClassPoint оголошені в тілі настанови class. Членом даних класу ClassPoint є змінні x та y. Окрім цього, тут визначено три функції-члени класу: Init() –

ініціалізація об’єкта, Set() – зміна даних об’єкта класу, Get() – виведення з об’єкта значення.

Кожен клас може містити як закриті, так і відкриті члени. За замовчуванням усі члени, визначені в класі, є закритими (private- членами). Наприклад, змінна a є закритим членом даних класу. Це означає, що до неї можуть отримати доступ тільки функції-члени класу ClassPoint; ніякі інші частини програми цього зробити не можуть. У цьому полягає один з проявів інкапсуляції: розробник повною мірою може керувати доступом до певних елементів даних. Закритими можна оголосити функції (у наведеному вище прикладі таких немає), внаслідок чого їх зможуть викликати тільки інші члени цього класу.

Щоб зробити члени класу відкритими (тобто доступними для інших частин програми), необхідно визначити їх після ключового слова public. Усі змінні або функції, визначені після специфікатора доступу public, є доступними для всіх інших функцій програми, в тому числі і функції main(). Отже, в класі myClass функції Init(), Set() і Get() є відкритими. Зазвичай у програмі організовується доступ до закритих членів класу через його відкриті функції. Зверніть увагу на те, що після ключового слова public знаходиться двокрапка.

Визначивши клас, можна створити об’єкт цього "класового" типу, якщо використати ім’я класу. Отож, ім’я класу стає специфікатором нового типу. Наприклад, у процесі виконання наведеної нижче команди створюється два об’єкти PointA та PointB класу ClassPoint.

Після створення об’єктів класу кожен з них набуває власні копії членів-даних, які утворює клас. Це означає, що кожний з об’єктів класу ClassPoint матиме власні копії змінних x та y. Отже, дані, пов’язані з об’єктом PointA, відокремлені (ізольовані) від даних, які пов’язані з об’єктом PointB.

Щоб отримати доступ до відкритого члена класу через об’єкт цього класу, використовують крапкову нотацію (оператор "крапка"). Наприклад, щоб вивести

на екран значення змінної x, яка належить об’єкту PointA, використовують таку команду:

```
cout << " PointA.x= " << PointA.x;
```

Закриті члени класу доступні тільки функціям, які є членами цього класу. Наприклад, таку вказівку PointA.x = 1.5; не можна помістити у функцію main(). Закриті дані класу можна змінити лише методами самого класу.

Для зміни даних класу використовують метод Set(). Даний метод не є обов’язковим, але вважається “правилом хорошого тону” використовувати метод саме з таким іменем для змін даних у об’єкті. Наведемо приклад його реалізації

```
class ClassPoint {  // оголошення класового типу
     double x;
     double y;
     public:
        void Set(double tmpX, double tmpY){  // зміна значень даних об’єкта класу
           x = tmpX;
           y = tmpY;
        }
        void Get(); // виведення з об’єкта значення
   };
```

Для отримання даних класу використовують метод Get(). Для описаного нами класу ClassPoint взагалі доцільно оголошувати два Get()-методи: GetX() та GetY() із такою реалізацією:

```
class ClassPoint {  // оголошення класового типу
     double x;
     double y;
     public:
        void Set(double tmpX, double tmpY);  // зміна значень даних об’єкта класу
```

```
        double GetX(){    // виведення з об’єкта значення
            return x;
        }
        double GetY(){    // виведення з об’єкта значення
            return y;
        }
};
```

Такий підхід спростить отримання доступу до даних та полегшить його підтримку.

Розглянемо приклад звернення до методів та даних класу ClassPoint. А також додамо до протоколу класу метод, який обчислюватиме відстань від точки, що викликає метод, до точки заданої як параметр методу.

```
class ClassPoint {  // оголошення класового типу
double x;
double y;
public:
   void Set(double tmpX, double tmpY){  // зміна значень даних об’єкта класу
      x = tmpX;
      y = tmpY;
   }
   double GetX(){    // виведення з об’єкта значення
       return x;
   }
   double GetY(){    // виведення з об’єкта значення
       return y;
   }
   double Distn(ClassPoint Point){ //метод обчислення відстані між точками
   return sqrt(pow((x-Point.x), 2)+ pow((y-Point.y), 2));
```

```
};

int main (){
   ClassPoint PointA, PointB;
   PointA.Set(12, -5); // демонстрація викликів методів у наступних 3 рядках
   PointB.Set(4, -3);
   cout<<PointA.GetX()<<endl;
   cout<<PointB.GetY()<<endl;
   cout<<PointA.Distn(PointB); // обчислення відстані між точками PointA та PointB
return 0;
}
```

Результат роботи цієї програми наступний:

12

-3

10

Ще одним важливим поняттям у класах є покажчик this. Під час кожного виклику функції-члена класу їй автоматично передається покажчик на об’єкт, який іменується ключовим словом this, для якого викликається ця функція. Покажчик this – це *неявний* параметр, який приймається всіма функціями-членами класу. Отже, в будь-якій функції-члені класу покажчик this використовується для посилання на об’єкт, що викликається. За допомогою покажчика this можна отримати доступ до всіх елементів класу (даних та методів).

Щоби зрозуміти, як працює покажчик this, розглянемо модифікацію методів для класу ClassPoint.

```
class ClassPoint { // Оголошення класового типу
   double x;
   double y;
   public:
```

```
        void Set(double tmpX, double tmpY){};      // Зміна значень даних об’єкта класу
        double GetX(){    // Виведення з об’єкта значення
            return this->x;
        }
        double GetY(){    // Виведення з об’єкта значення
            return this->y;
        }
};
int main (){
    ClassPoint PointA, PointB;
    PointA.Set(12, -5); // демонстрація викликів методів у наступних 3 рядках
    PointB.Set(4, -3);
    cout<<PointA.GetX()<<endl;
    cout<<PointB.GetY()<<endl;
return 0;
}
```

Результат роботи цієї програми наступний:

12

-3

Цей приклад тривіальний, але у ньому показано, як можна використовувати покажчик this. Практика у написанні програм мовою С++ надасть можливість впевнитись і зручності та важливості використання покажчика this.

Питання для самоконтролю:

1. Що таке об’єктно-орієнтоване програмування?
2. Чим відрізняється об’єктно-орієнтований підхід у програмуванні від структурного підходу?

3. Що таке інкапсуляція?
4. Що таке поліморфізм?
5. Що таке наслідування?
6. Що таке клас?
7. Яка різниця між класом та об’єктом?
8. Які бувають типи доступу до членів класу?
9. Як звернутись до даних у класі?
10. Як звернутись до методу у класі?
11. Опишіть призначення методу Get().
12. Опишіть призначення методу Set().
13. Для чого використовується покажчик this?

Тема 2
Конструктори та деструктор класу

Як правило, певну частину об’єкта, перш ніж його можна буде використовувати, необхідно ініціалізувати. Але, оскільки вимога ініціалізації членів-даних класу є достатньо поширеною, то у мові програмування C++ передбачено реалізацію цієї потреби при створенні об’єктів класу. Така автоматична ініціалізація членів-даних класу здійснюється завдяки використанню конструктора.

Конструктор – це спеціальна функція-член класу, яка викликається при створенні об’єкта, а її ім’я обов’язково збігається з іменем класу.

Особливості конструктора:

- призначений для виділення динамічної пом’яті та ініціалізації даних класу;
- якщо конструктор не оголошений, то компілятор згенерує конструктор за замовчуванням (без параметрів);
- для конструктора явно не вказується результуючий тип, конструктор не повертає значення;
- визначається в середині класу або поза протоколом класу;
- може перевантажуватися;
- може мати параметри за замовчуванням;
- не успадковується;
- може мати список ініціалізацій, який задається у заголовку після символу «:». (Так ініціалізуються константні поля (const), посилання на довільний тип, поля з типом іншого класу, конструктори базових класів в разі успадкування.)
- може бути конструктором копіювання, якщо має один параметр з типом T& або const T&;

- не може бути викликаний явно з використанням об’єкта класу (або вказівника чи посилання на клас);
- при успадкуванні раніше викликаються конструктори базових класів, а потім – похідних.
- може бути розміщений у private, public, protected-частинах класу.
- автоматично викликається при створенні об’єкта у сегменті даних, стеку, типів, коли значення певного типу перетворюється у в об’єкт класу у виразі, передачі аргументів у функцію, поверненні результату із функції;
- не викликається при оголошенні вказівника або посилання на клас;
- при успадкуванні раніше викликаються конструктори базових класів, а потім – похідних.

Розглянемо приклад застосування конструктора для ініціалізації членів-даних класу:

```
class TestClass{ // оголошення класового типу
     int first;
     public:
          TestClass (int parametr){ // конструктор класу
               first = parametr;
               cout<<”Object inicialized”;
          }
          void setTestClass(int a) {first = a}; //set-метод
          int getTestClass() {return first}; //get-метод
}
```

Зверніть увагу на те, що в оголошенні конструктора TestClass відсутній тип значення, що повертається. У мові програмування C++ конструктори не повертають значень і, як наслідок, немає сенсу вказувати їх тип. У описаному вище коді у процесі виконання конструктора ним виводиться повідомлення ”Object inicialized” – " Об’єкт ініціалізовано", яке слугує виключно ілюстративній меті. На практиці ж здебільшого конструктори не виводять ніяких повідомлень.

Конструктор об’єкта викликається при створенні об’єкта. Це означає, що він викликається у процесі виконання вказівки створення об’єкта. Конструктори глобальних об’єктів викликаються на самому початку виконання програми, тобто ще до звернення до функції main(). Що стосується локальних об’єктів, то їх конструктори викликаються кожного разу, коли виникає потреба створення такого об’єкта.

Існує декілька типів конструкторів. Класифікуються вони за своїм призначенням:

- конструктори ініціалізації (звичайні);
- копіювання;
- перетворення типів.

Конструктор ініціалізації викликається:

- при створенні нового об’єкта у сегменті даних, стеку, динамічній пам’яті та ініціалізації його полів (всіх або деяких з них);
- для створення локального об’єкта у виразі тип_класу (список параметрів конструктора).

У прикладі виклику конструктора для класу TestClass нами був реалізований саме конструктор ініціалізації.

Реалізація конструктора копіювання для класу TestClass матиме наступний вигляд:

```
class TestClass{ // оголошення класового типу
     int first;
     public:
          TestClass (int parametr){ // конструктор ініціалізації
               first = parametr;
          }
          TestClass (const TestClass &Object){// конструктор копіювання
               first = Object.first;
          }
```

```
    void setTestClass(int a) {first = a}; //set-метод
    int getTestClass() {return first}; //get-метод
}
```

Властивості конструктор копіювання:

- має один параметр з посиланням на тип класу. Може мати більше одно параметра, які задаються за замовчуванням;
- не допускає перевантаження — у класі можу бути лише один конструктор копіювання;
- якщо не визначений у класі, то конструктор копіювання генерується автоматично і виконує порозрядне копіювання об’єктів;
- виконує копіювання усіх видів полів даних;
- якщо поля класу мають тип вказівника або посилання на будь-який тип, то конструктор копіювання повинен бути визначений користувачем;
- якщо визначення конструктора копіювання визначено у закритій або захищеній частинах класу, то неможливо створити копію об’єктів поза межами класу (крім друзів класу).

Конструктор копіювання викликається:

- при створення нового об’єкта та його ініціалізації вже існуючим об’єктом цього ж класу;
- при передачі об’єктів через список параметрів функцій «за значенням» (не через вказівники або посилання);
- при поверненні функціями локальних об’єктів класу «за значенням» (не через вказівники або посилання).

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

Реалізація конструктора перетворення типів для класу TestClass матиме наступний вигляд:

```
class TestClass{ // оголошення класового типу
     int first;
     public:
          TestClass (float prm){ // конструктор перетворення типів
               first = int(prm);
          }
          void setTestClass(int a) {first = a}; //set-метод
          int getTestClass() {return first}; //get-метод
   }
```

Викликається конструктор перетворення типіву таких випадках:

- для ініціалізації об’єкта класу значенням, тип якого є відмінним від цього класу;
- у виразах виду «класовий тип • інший тип»;
- для явного перетворення будь-якого типу до типу класу;
- в операції присвоєння «класовий тип = інший тип»;
- для передавання параметрів у функцію, коли формальний параметр має класовий тип, а фактичний – інший тип;
- для повернення значення функцією, коли функція має класовий тип, а вираз оператора return має інший тип.

Доповненням до конструктора слугує деструктор – це функція, яка викликається під час руйнування об’єкта. У багатьох випадках під час руйнування об’єкта необхідно виконати певну дію або навіть певні послідовності дій. Локальні об’єкти створюються під час входу в блок, у якому вони визначені, і руйнуються при виході з нього. Глобальні об’єкти руйнуються внаслідок завершення програми. Існує багато чинників, які заставляють використовувати деструктори. Наприклад, об’єкт повинен звільнити раніше виділену для нього

пам’ять. У мові програмування C++ саме деструкторам доручається оброблення процесу деактивізації об’єкта.

Ім’я деструктора має збігатися з іменем конструктора, але йому передує сим-вол "~" (тильда). Подібно до конструкторів, деструктори не повертають значень, а отже, в їх оголошеннях відсутній тип значення, що повертається.

Розглянемо вже знайомий нам клас TestClass, але тепер він має містити конструктор і деструктор:

```
class TestClass{ // оголошення класового типу
     int first;
     public:
          TestClass (int parametr){ // конструктор класу
               first = parametr;
               cout<<”Object inicialized”;
          }
          ~TestClass (){ // деструктор класу
               cout<<”Object deleted”;
          }
          void setTestClass(int a) {first = a}; //set-метод
          int getTestClass() {return first}; //get-метод
}
```

Властивості:

- не має параметрів;
- не повертає значення;
- не перевантажується;
- не успадковується;
- існує за замовчуванням, але повинен бути оголошений, якщо для полів класу було виділено динамічну пам’ять.

Питання для самоконтролю:

1. Поясніть різницю між оголошенням та ініціалізацією об’єкта.
2. Який метод виконує ініціалізацію об’єкта?
3. Як відбувається ініціалізація об’єкта, якщо конструктор не описано розробником?
4. Назвіть особливості конструктора.
5. Яка мінімальна кількість параметрів у конструкторі?
6. Назвіть види конструкторів.
7. Як оголошується коструктор без параметрів?
8. Як оголошується коструктор з параметром?
9. Назвіть властивості конструктора ініціалізації.
10. Як оголошується конструктор копіювання?
11. Назвіть властивості конструктора копіювання.
12. Як оголошується конструктор перетворення типів?
13. Назвіть властивості конструктора перетворення типів.
14. Яка різниця між конструктором та методом set()?
15. Що таке деструктор?
16. Назвіть властивості деструктора.
17. Скільки деструктрів може бути оголошено для класу?

Тема 3
Робота із масивами об’єктів

Масиви об’єктів можна організувати так само, як і масиви значень стандартних типів. Масиви об’єктів можуть оголошуватись одним із таких способів:

```
Тип_класу ідентифікатор_масиву [кількість об’єктів];
Тип_класу *вказівник = new Тип_класу [кількість об’єктів];
Тип_класу & посилання = *new Тип_класу [кількість об’єктів];
```

Вцілому робота із масивами об’єктів нічим не відрізняється від роботи із масивами елементів простих типів або структур (елементи типу struct): звертання відбувається за ідентифікатором масиву та номером елемента. Звернення до даних та методів класу здійснюється через крапкову нотацію. При оголошенні посилання на динамічний масив слід зважати на те, що ім’я посилання еквівалентно лише елементу з індексом 0. Для доступу до елемента слід враховувати зміщення відносно адреси нульового елемента.

Розглянемо наступний приклад. У наведеному нижче коді створюється клас displayClass, який містить значення розширення для різних режимів роботи монітора. У функції main() створюється масив для зберігання трьох об’єктів типу displayClass, а доступ до них здійснюється за допомогою звичайної процедури індексування елементів масиву.

```
enum resolution {low, medium, high};
class displayClass {
     int width;
     int height;
     resolution res;
     public:
```

```
            void Set(int w, int h) {width = w; height = h; }
            void Get(int &w, int &h) {w = width; h = height; }
            void Show(resolution r) {res = r; }
            resolution getRes() {return res; }
};
            char names[3][9] = { "Low", "Medium", "Height"};
int main (){
displayClass Monitor[3];
Monitor[0].Show(low);
Monitor[0].Set(640, 480);
Monitor[1].Show(medium);
Monitor[1].Set(800, 600);
Monitor[2].Show(high);
Monitor[2].Set(1600, 1200);
cout << "Regimes: " << endl;
int w, h;
     for (int i=0; i<3; i++) {
          cout << names[Monitor[i].getRes()] << ": ";
          Monitor[i].Get(w, h);
          cout << w << " x " << h << endl; }
return 0;
}
```

Внаслідок виконання ця програма відображає на екрані такі результати:

Regimes:

Low: 640 x 480

Medium: 800 x 600

Height: 1600 x 1200

Варто звернути увагу на використання двовимірного символьного масиву names для перетворення перерахованого значення (enum) в еквівалентний символьний рядок. В усіх перерахунках (enum), які не містять безпосередньо заданої ініціалізації, перша константа має значення 0, друга – значення 1 і т.д.

Отже, значення, що повертається функцією getRes(), можна використовувати для індексації елементів масиву names, що дає змогу вивести на екран відповідну назву режиму відображення. Багатовимірні масиви об’єктів індексуються так само, як і багатовимірні масиви значень інших типів.

Розглянемо, як відбувається ініціалізація масивів об’єктів. Якщо клас містить параметризований конструктор, то масив об’єктів такого класу можна ініціалізувати. Наприклад, у наведеному нижче коді програми використовується клас myClass і параметризований масив об’єктів типу array цього класу, що ініціалізується конкретними значеннями.

```
class myClass { // оголошення класового типу
     int a;
     public:
          myClass(int b) { a = b; }
          double Put() { return a; } };
int main() {
myClass array[4] = { -1, -2, -3, -4 };
     for (int i=0; i<4; i++)
          cout << "array[" << i << "]= " << array[i].Put() << endl;
return 0;
}
```

Результати виконання цієї програми наступні:

array[0]= -1

array[1]= -2

array[2]= -3

array[3]= -4

Вони підтверджують, що конструктору myClass дійсно були передані значення від -1 до -4. Насправді синтаксис ініціалізації масиву, виражений рядком myClass array[4] = { -1, -2, -3, -4 };, є скороченим варіантом такого (довшого) формату: myClass array[4] = { myClass(-1), myClass(-2), myClass(-3), myClass(-4)};

Формат ініціалізації, представлений у наведеній вище програмі, використовується частіше, ніж його довший еквівалент, проте необхідно пам’ятати, що він працює для масивів таких об’єктів, конструктор яких приймає тільки один аргумент. При ініціалізації масиву об’єктів, конструктор яких приймає декілька аргументів, необхідно використовувати довший формат ініціалізації. Розглянемо такий приклад.

```
class myClass { // оголошення класового типу
     int a, b;
     public:
          myClass(int c, int d) { a = c; b = d; }
          int PutA() { return a; }
          int PutB() { return b; }
};
int main() {
myClass array[4][2] = {
myClass(1, 2), myClass(3, 4),
myClass(5, 6), myClass(7, 8),
myClass(9, 10), myClass(11, 12),
myClass(13, 14), myClass(15, 16) };
for (int i=0; i<4; i++)
     for (int j=0; j<2; j++) {
          {cout << "array[" << i << "," << j << "] ==> a= ";
          cout << array[i][j].PutA() << "; b= ";
          cout << array[i][j].PutB() << endl;
      }
return 0;
}
```

Внаслідок виконання ця програма відображає на екрані такі результати:

array[0,0] ==> a= 1; b=2

array[0,1] ==> a= 3; b=4

array[1,0] ==> a= 5; b=6

array[1,1] ==> a= 7; b=8

array[2,0] ==> a= 9; b=10

array[2,1] ==> a= 11; b=12

array[3,0] ==> a= 13; b=14

array[3,1] ==> a= 15; b=16

У наведеному вище прикладі параметризований конструктор класу myClass приймає два аргументи. В основній функції main() оголошується та ініціалізується масив array об’єктів шляхом безпосередніх викликів конструктора myClass(). Для ініціалізації масиву можна завжди використовувати довгий формат ініціалізації, навіть якщо об’єкт приймає тільки один аргумент (коротка форма просто зручніша для застосування).

Крім одновимірних, можна використовувати і двовимірні масиви об’єктів. Оголошуються такі масиви аналогічно до одновимірних:

```
Тип_класу ідентифікатор_масиву [кількість рядків] [кількість стовпців];
```

Динамічна пам’ять для двовимірного масиву виділяється так:

```
Тип_класу (*вказівник) [кількість стовпців] = new Тип_класу [кількість рядків] [кількість стовпців];
```

Кількість рядків та стовпців у цих оголошеннях повинні бути константами.

Звільнення пам’яті, що виділяється під об’єкти динамічного масиву, відбувається так:

```
delete [] вказівник;

delete [] & посилання;
```

Питання для самоконтролю:

1. Що таке масив?
2. Як відбувається звернення до елемента масиву?

3. Як оголосити одновимірний масив об’єктів?
4. Як оголосити одновимірний динамічний масив об’єктів?
5. Чим відрізняється масив об’єктів від масивів, що складаються із елементів простих типів?
6. Як ініціалізувати масив, який складається із об’єктів?
7. Як відбувається звернення до елементів масиву об’єктів?
8. Як викликати метод елемента масиву об’єктів?
9. Як оголосити двовимірний масив об’єктів?
10. Як оголосити двовимірний динамічний масив об’єктів?
11. Як звільнити пам’ять, яка була виділена під одновимірний динамічний масив об’єктів?
12. Як звільнити пам’ять, яка була виділена під двовимірний динамічний масив об’єктів?

Тема 4
Види класів. Вкладені класи

Об’єктно-орієнтована програма здебільшого ґрунтується на основі декількох класів. Тому важливо визначити правила взаємовідношення між класами та їх розміщення у програмі.

Отже, класи можуть бути:

- глобальні, оголошення яких здійснюється поза функціями;
- локальні, які розміщуються всередині функції;
- дружні, яким надають права доступу до усіх частин класу;
- не вкладені, коли у протоколі класу немає оголошення інших класів;
- вкладені, коли у протоколі класу є оголошення інших класів;
- контейнери та колекції, які містять об’єкти інших класів;
- ітератори, призначені для забезпечення доступу до елементів інших класів;
- успадковані, коли похідний клас використовує оголошення даних та методів базового класу;
- шаблонні, які є узагальненим визначенням множини класів;
- абстрактні, що містять нереалізовані віртуальні методи;
- інтерфейсні, призначені для однакового доступу до різних версій об’єктів (компонентів) відкомпільованого класу, незалежно від їх внутрішньої реалізації.

До цього часу ми розглянули лише один вид класів – глобальні невкладені. Глобальними називаються класи, оголошення яких розміщено поза функціями. Допускається використання імен вище розміщених класів для оголошення імен у нижче розміщених класах.

Одним із важливих видів класів є контейнерні. Контейнерним називається клас, який містить дані з типом іншого класу. Такий вид класів забезпечує можливість повторного використання коду програми.

Контейнерні класи будуються на основі аналізу внутрішньої структури складного об’єкта або системи та виявленні їх складових елементів або підсистем. Під час аналізу виділяються функціонально повні або самодостатні елементи (підсистеми), які дають змогу побудувати класи із можливістю їх подальшого повторного використання для побудови інших програмних продуктів. Між контейнерними класами існує відношення “містить”: контейнерний клас містить об’єкти-представники інших класів. Контейнерний клас називають зовнішнім, а інкапсульовані об’єкти – внутрішніми.

Наприклад, розглянемо клас ClassPoint, з яким ми знайомились у першій темі даного посібника. До протоколу цього класу додаємо метод printPoint виведення координат точки на екран.

```
class ClassPoint {  // оголошення класового типу
     double x;
     double y;
     public:
        void Set(double tmpX, double tmpY){};     // зміна значень даних об’єкта класу
          double GetX(){ // виведення з об’єкта значення
            return this->x;
        }
        double GetY(){    // виведення з об’єкта значення
            return this->y;
        }
        void printPoint(){     // виведення координат точки на екран
            cout<<”(“<<x<<”; “<<y<<”)”<<endl;
        }
};
```

Побудуємо також клас ClassCircle, який містить дані для побудови кола на площині. Такий клас буде контейнером для об’єкта класу ClassPoint, що використовується для задання координат центру кола на площині. До того ж клас ClassCircle містить поле radii для визначення радіусу кола. У відкритій частині класу ClassCircle розміщено конструктор ініціалізації даних та метод їх виведення на екран.

```
class ClassCircle {   // оголошення контейнерного класу
     ClassPoint center;  // інкапсульований об’єкт з координатами центру кола
     double radii;        // радіус кола
     public:
       ClassCircle (ClassPoint &p, double r):  center(p)  // конструктор ініціалізації
                    { radii = r; }
       void printCircle (){    // виведення координат точки та радіусу кола на екран
                    center.printPoint;
                    cout<<”radii = ”<<radii<<endl;
       }
}
```

Якщо у класі внутрішнього об’єкта перекрито конструктор за замовчуванням, то ініціалізація такого об’єкта повинна здійснюватися у списку ініціалізації конструктора контейнерного класу (одразу після символу “:”). За інших обставин ініціалізацію інкапсульованого об’єкта можна виконати у тілі контейнерного класу (у фігурних дужках “{ }”).

Розглянемо функцію main() для демонстрації роботи описаних нами вище класів. Головна функція оголошує об’єкти класів ClassCircle та ClassPoint, потім викликає метод printCircle () для того, аби вивести на екран інформацію про коло.

```
int main(){
```

```
ClassPoint point(-4, 7);              // об’єкт класу ClassPoint
ClassCircle circle (point, 10);       // об’єкт класу ClassCircle
circle.printCircle ();      // виклик методу контейнерного класу ClassCircle
return 0;
}
```

Контейнери восновному будуються на основі шаблонних класів. Вони (контейнери) не допускають явного задання числа елементів і не підтримують розгалуженої структури. Найпростіші види контейнерів вбудовані безпосередньо у мову C++.

Розглянемо приклад, коли до складу класу-контейнера входить не один інкапсульований об’єкт, а масив об’єктів. Нехай необхідно розробити клас File, який містить дані про назву, розмір, дату зміни та розширення файлу. Клас Catalog містить масив елементів класу File. Визначити методи роботи з файлами та каталогами.

Реалізація для такої задачі буде виглядати наступним чином.

```
class File{
     private:
          string name, date, attributes;
          float size;
     public:
        File (){                  // конструктор без параметрів
          name = "New file";
          date = "00.00.00";
          attributes = " - ";
          size = 0;
       }
     // конструктор з параметрами
       void Set ( string name, string date, float size, string
attributes){
                    this->name = name;
```

```
                this->date = date;
                this->size = size;
                this->attributes = attributes;
     }
        void Set (){
              cout << "Enter a file name - ";
              cin >> name;
              cout << "Enter the date the file was created - ";
              cin >> date ;
              cout << "Enter the file size - ";
              cin >> size;
              cout << "Enter the file attributes - ";
              cin >> attributes;
              cout << endl;
        }
    void Get (){
        cout << name << "  " << date<< " " << time << "  " << size
<< " kb\t\t" << attributes << "  ";
    }
    ~File (){}
};
class Catalog { // контейнерний клас
     private:
       File arr[3];
     public:
           void SetCatalog(File f1, File f2, File f3){
                  arr[0] = f1;
              arr[1] = f2;
                  arr[2] = f3;
           }
         void GetCatalog(){
                  arr[0].Get();
                  arr[1].Get();
```

```
                    arr[2].Get();
          }

    ~Catalog (){}
};

int main(){
    File f1, f2, f3;
    Catalog k;
    cout << "За замовчування коструктор створює об’єкт із такими значеннями полів";
    f3.Get();
    cout << "Відкоригуйте дані";
    f1.Set("foto", "17.06.14", 1371, "A");
    f2.Set("System", "31.03.20", 4567, "DA");
    f3.Set();
    k.SetCatalog(f1, f2, f3);
    cout << "Ваш файл потрапив до каталогу";
    k.GetCatalog();
    return 0;
}
```

Питання для самоконтролю:

1. Назвіть вид класів?
2. Що таке глобальний клас?
3. Що таке локальний клас?
4. Який клас називають контейнером?
5. Який об’єкт називають інкапсульованим?
6. Що викликається раніше: конструктор класу-контейнера чи інкапсульованого об’єкту?

Тема 5
Перевантаження операторів

У математиці зміст операції залежить від її операндів. Якщо a і b є цілими числами, їх сума a+b обчислюється по одним правилах, якщо матрицями — по іншим. У людини, що знає природу операндів, не виникає складнощів при інтерпретації змісту операції +.

Як відомо, кожен вбудований тип даних пов’язаний з визначеним набором операцій, які можна до нього застосовувати. Ці операції позначаються відповідними символами, зміст яких відомий компілятору заздалегідь. Однак цілком природно спробувати розширити застосування операторів на класи, що створюються програмістом. Наприклад, матричні обчислення стають набагато наочнішими, якщо сума двох матриць записується як a+b, а не як summa(a,b). Для цього в мові C++ існує механізм перевантаження операторів.

Операції у C++ є контекстнозалежними і можуть бути перевантажені безпосередньо у класі.

Перевантаження допускається до таких операцій:

1. Первинних

[]	()	->

2. Бінарних

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

Існують також оператори, заборонені до перевантаження. Зміна їх змісту зруйнувала би логіку програми. До таких операторів належать :: (оператор дозволу області видимості), . (“ точка” — оператор доступу до члена класу), ?: (тернарний оператор), .* (доступ до розіменованого вказівника-члена класу), sizeof, typeid, static_cast, dynamic_cast, const_cast і reinterpret_cast. Крім того, не рекомендується перевантажувати логічні оператори && і ||, оскільки на їхні перевантажені версії не поширюється правило скорочених обчислень логічних виразів. (Нагадаємо, що це правило полягає в наступному: якщо на деякому етапі значення усього вираження стає визначеним, подальші обчислення припиняються.)

Перевантаження операторів відбувається за допомогою операторних функцій.

Синтаксис операторних функцій виглядає наступним чином.

```
  тип_значення_що_повертається operator символ_операції(параметри)
{
   ...
  }
```

Наприклад, операторна функція, що перевантажує операцію +, має вигляд operator+().

Операторні функції повинні мати прямий доступ до членів класу. Отже, необхідно, щоб вони були або членами класу, або дружніми функціями.

Необхідно пам’ятати про обмеження, що супроводжують застосування перевантажених операторів.

- Перевантажені функції не можуть змінити пріоритет операторів.
- Кількість операндів фіксована: жодного, один чи два.
- Значення операндів не можна задавати за замовчуванням.

Розглянемо, як відбувається перевантаження унарних операторів за допомогою функцій-членів.

Оператори, як відомо, можуть бути унарними і бінарними. Унарний оператор має один операнд, а бінарний — два. Нагадаємо, що до унарних операторів, що перевантажуються, належать такі оператори, як +, -, ++, --, &, ~ і !. До бінарних операторів, що перевантажуються, належать всі інші оператори, перераховані в таблиці вище.

Операторні функції-члени, що перевантажують унарний оператор, мають одну особливість: їх операнди передаються неявно за допомогою вказівника this. Отже, така функція-член класу не має явних параметрів.

Розглянемо приклад перевантаження унарного мінуса.

```
class Tcomplex // оголошення класу для роботи із комплексними числами
{
   double Re;
   double Im;
public:
   TComplex(double x, double y):Re(x),Im(y){ } //конструктор з
   параметром
   TComplex(TComplex& z){ Re = z.Re; Im = z.Im;} //конструктор
   копіювання
   ~TComplex(){}
   void printTComplex(){
   cout<<x << " +i"<<y;
   }
   TComplex operator-() {Re = -Re; Im = -Im; return *this;}
   //перевантаження оператора "-"
};

int main()
{
   TComplex z(1,1),u(0,0);
   z.print();
```

```
    u=-z;
    u.printTComplex();
    return 0;
}
```

Помітимо, що операторна функція може повертати об’єкт класу (точніше, розіменований вказівник this) чи посилання на об’єкт, а може і нічого не повертати. Вибираючи тип значення, що повертається, необхідно керуватися здоровим глуздом. Якщо унарний оператор повинен використовуватися всередині виразів, то операторна функція повинна повертати об’єкт або посилання. Якщо ж оператор використовується ізольовано, функція може нічого не повертати.

Рядок програми u = - z; можна переписати в еквівалентному вигляді:

```
u = z.operator-();
```

Цей рядок демонструє, що виклик операторної функції operator-() здійснюється об’єктом z. Варто мати на увазі, що хоча зміст оператора перевантажувати можна, його природу змінювати заборонено. Це значить, що унарний оператор не можна перевантажити як бінарний і навпаки.

Розглянемо найбільш важливі приклади унарних операторів.

У мові C++ передбачено дві форми операторів інкремента і декремента: префіксна і постфіксна. Для того щоб розрізняти їх, використовується звичайний механізм перевантаження функцій – вводиться фіктивний цілочисловий параметр.

Наприклад, для класу TComplex прототипи відповідних операторних функцій можуть виглядати в такий спосіб.

```
TComplex operator++();          // префіксна форма
TComplex operator++(int x);     // постфіксна форма
TComplex operator--(); // префіксна форма
TComplex operator--(int x);     // постфіксна форма
```

Спробуємо розібратися на конкретному прикладі перевантаження постфіксної та префіксної форм інкременції.

```
class TComplex
 {
    double Re;
    double Im;
 public:
    TComplex(double x, double y):Re(x),Im(y){ }
    TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}
    ~Tcomplex(){}
    void printTComplex(){
    cout<<x << “+i ”<<y;
    }
    TComplex& operator++()
    {
        ++Re;
        ++Im;
        cout<<"Префіксна форма ++"<<endl;
        return *this;
    }
    const TComplex operator++(int i)
    {
        ++Re;
        ++Im;
        cout<<"Постфіксна форма ++"<<endl;
        return *this;
    }
    TComplex& operator--()
    {
        --Re;
        --Im;
        cout<<"Префіксна форма --"<<endl;
        return *this;
```

```
    }
    const TComplex operator--(int i)
    {
        --Re;
        --Im;
        cout<<"Постфіксна форма --"<<endl;
        return *this;
    }
};

int main()
{
    TComplex z(1,1);
    ++z;
    z.printTComplex();
    z++;
    z.printTComplex();
    --z;
    z.printTComplex();
    z--;
    z.printTComplex();
    return 0;
}
```

У функції main() до об’єкта z послідовно застосовуються префіксна і постфіксна форми операторів ++ і --.

Результат роботи описаної вище програми наступний:

Префіксна форма ++

2.000000 + i*2.000000

Постфіксна форма ++ 0

3.000000 + i*3.000000

Префіксна форма --

000000 + i*2.000000

Постфіксна форма -- 0

1.000000 + i*1.000000

Якщо символ операції ++ стоїть перед операндою, викликається операторна функція operator++(), якщо після — операторна функція operator++(int i). Змінна і відіграє роль прапорця, що повідомляє компілятору, що дана функція перевантажує постфіксну форму оператора інкремента і декремента.

Незважаючи на те що розробник вільний самостійно трактувати перевантажений оператор, прагнучи зберегти аналогію з його убудованими значеннями, необхідно дотримуватись визначених правил. Наприклад, як відомо, оператор інкрементації цілих чисел повертає посилання на неконстантний об’єкт. Ця властивість повинна зберігатися і при перевантаженні.

Оператори заперечення (!), взяття адреси (&) і побітового заперечення (~) допускають перевантаження, але не мають універсальних альтернатив, хоча варто було б реалізувати. Їх можна перевантажувати, наприклад, для підвищення наочності програми. Скажемо, за допомогою оператора ! можна позначати операцію звертання матриці, а за допомогою символу ~ — її транспонування. Щоправда, застосування тильди закріплене за деструкторами, тому варто виявляти обережність, щоб не створити плутанину. У будь-якому випадку зміст перевантаження операторів залежить від конкретної задачі.

Оператор посилання на член об’єкта -> є унарним. Операторна функція, що перевантажує його, виглядає таким чином.

об’єкт -> елемент

Цей запис еквівалентний наступному виразу.

об’єкт.operator->(елемент);

Функція operator ->() повинна бути нестатичним членом класу. Як параметр вона одержує об’єкт чи класу посилання на нього, повертаючи вказівник this на об’єкт, де виконується виклик, або посилання на об’єкт будь-якого іншого класу, у якому визначений оператор ->. її зручно використовувати в контейнерних класах,

що містять усередині себе вказівник на інший клас. Основний зміст перевантаження оператора -> полягає в додатковій функціональності, що розширює можливості звичайних вказівників.

Наприклад, у приведеній нижче програмі функція operator->() веде підрахунок посилань на кожен об’єкт класу.

```
class TClass
{
   int n;
   int counter;
public:
   TClass(int x):n(x),counter(0) { }
   TClass* operator->();
   int get(void) { return n;}
   int ref(void) { return counter; }
};

TClass* TClass::operator ->()
{
   counter++;
   return this;
}
int main()
{
   TClass a(1), b(2);
   cout<<"n = "<<a->get())<<endl;
   cout<<"n = "<<b->get())<<endl;
   cout<<"n = "<<a->get())<<endl;
   cout<<"counter = "<<a->ref())<<endl;
   cout<<"counter = "<<b->ref())<<endl;
   return 0;
}
```

Врезультаті роботи програми на екран виводяться наступні рядки.

n = 1

n = 2

n = 1

counter = 3

counter = 2

Розглянемо, як відбувається перевантаження бінарних операторів за допомогою функцій-членів. Бінарний оператор має два операнди. Його виклик виконується об’єктом, розташованим у лівій частині оператора. Отже, бінарний оператор

a+b

еквівалентний такому оператору.

a.operator+(b)

Таким чином, бінарна операторна функція-член повинна має тільки один параметр, що задає другий операнд. Вказівник this на перший операнд вона одержує неявно.

Для того щоб бінарну операторну функцію можна було застосовувати всередині виразів, необхідно, щоб вона повертала об’єкт свого класу. Розглянемо приклад перевантаження бінарного оператора + на прикладі класу TComplex.

```
class TComplex{
   double Re;
   double Im;
public:
   TComplex(double x, double y):Re(x),Im(y){ }
   TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}
   ~TComplex(){}
   void printTComplex(){
   cout<<x << “ +i”<<y;
   }
   TComplex operator+(TComplex z){
```

```
      TComplex w(0,0);
      w.Re = Re+z.Re;
      w.Im = Im+z.Im;
      return w;
    }
  };

int main()
{
  TComplex u(1,1),v(2,2),z(0,0);
  z=u+v;
  z.printTComplex();
  return 0;
}
```

У цій програмі сума комплексних чисел u і v присвоюється об’єкту z. Виклик операторної функції operator+() здійснюється з об’єкта u. Оскільки сума двох об’єктів класу TComplex є об’єктом цього ж класу, не обов’язково створювати новий об’єкт і записувати в нього результат підсумовування. Можна скористатися наступною синтаксичною конструкцією.

(u+v).printTComplex();

У цьому випадку функція printTComple() буде викликатися тимчасовим об’єктом, створеним операторної функцією operator+(). Цікаво, що цей механізм допускає вживання “безглуздих виразів”. Наприклад, компілятор не заперечує проти такого оператора.

(u+v)=z;

Для того щоб запобігти цьому, оператор + повинний повертати константний об’єкт. Крім того, бажано, щоб він не змінював другий операнд. Отже, параметр повинний бути константним. І останнє: тому що об’єкт може бути досить громіздким, його варто передавати за значенням. Отже, одержуємо наступну реалізацію.

```
const TComplex operator+(const TComplex& z){
   TComplex
   w(0,0); w.Re =
   Re+z.Re; w.Im=
   Im+z.Im;
   return w;
}
```

Продемонструємо, як здійснюються обчислень над комплексними числами із використанням перевантажених операторів.

```
 class TComplex
{
   double Re;
   double Im;
public:
   TComplex(double x, double y):Re(x),Im(y){ }
   TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}
   ~TComplex(){}
   void printTComplex();
   TComplex operator+(const TComplex& z)     // додавання
   {
      TComplex w(0,0);
      w.Re = Re+z.Re;
      w.Im = Im+z.Im;
      cout<<"Operator + "<<endl;
      return w;
   }
   TComplex operator-(const TComplex& z)     // віднімання
   {
      TComplex w(0,0);
      w.Re = Re-z.Re;
      w.Im = Im-z.Im;
      cout<<"Operator - "<<endl;
```

```
    return w;
}
TComplex operator*(const TComplex& z)     // множення
{
    TComplex w(0,0);
    w.Re = Re*z.Re-Im*z.Im;
    w.Im = Re*z.Im+Im*z.Re;
    pcout<<"Operator * "<<endl;
    return w;
}
TComplex operator-()       // унарний мінус
{
    Re = -Re;
    Im = -Im;
    cout<<"Unary operator - "<<endl;
    return *this;
}
TComplex operator~()       // спряжене комплексне число
{
    TComplex w(0,0);
    w.Re = Re;
    w.Im = -Im;
    cout<<"Unary operator ~ "<<endl;
    return w;
}
TComplex operator/(TComplex& z)      // ділення
{
    TComplex w(0,0);
    w=(*this)*(~z);
    w.Re = w.Re/modul2(z);
    w.Im = w.Im/modul2(z);
    cout<<"Operator / "<<endl;
```

```
        return w;
    }
    double modul2(const TComplex& z)
    {
    return z.Re*z.Re+z.Im*z.Im;
    }
};

int main()
{
    TComplex u(1,1),v(2,2),z(0,0);
    z=(u+v)*u/v;
    z.printTComplex();
    return 0;
}
```

У класі TComplex операція розподілу двох комплексних чисел реалізована через обчислення спряженого числа (операція ~) і розподіл дійсної і уявної частин на квадрат модуля дільника (функція modul2()).

От у якому порядку виконувалися зазначені оператори.

Operator +

Operator * Unary

operator ~

Operator *

Operator /

У підсумку, як і слід було очікувати, одержуємо наступне число.

1.500000 + i*1.500000

Реалізуючи ланцюжок обчислень, необхідно обережно використовувати ключове слово const. Безперечно, захист параметра від необережної модифікації необхідний. У той же час повернення константних об’єктів не завжди виправданий.

У програмах, розглянутих вище, ми вільно маніпулювали об’єктами, привласнюючи їх один одному. Це відбувалося завдяки вбудованому оператору присвоювання, що виконує побітове копіювання. Як і у випадку з конструктором копіювання, це не краще рішення. Якщо об’єкт містить вказівник на деяку область пам’яті, його копія також буде посилатися на неї. Для того, щоб цього не відбулося, перевантажений оператор присвоювання повинен виконувати глибоке копіювання, уникаючи подвійної адресації.

```
class TArray{
   int *p;
   int size;
 public:
   TArray(long n, int x);
   TArray(TArray&);
   TArray& operator=(TArray& X);
   void view();
 };

 int main()
 {
   TArray x(10,1),y(10,0);
   x.view();
   y = x;
   y.view();
   return 0;
}

 TArray::TArray(long n, int x)
 {
   size = n;
   p = new int[size];
   for (long i=0; i<size; i++)p[i] = x;
   cout<<"Адреса = "<<p<<endl;
```

```
}

TArray::TArray(TArray& X)
{
   size=X.size;
   p = new int[size];                    // глибоке копіювання
   for (long i=0; i<X.size; i++) p[i] =
   X.p[i];
   cout<<"Адреса = "<<p<<endl;
}

void TArray::view()
{
    for(long i=0; i<size; i++) cout<<"  "<<p[i]);
}

TArray& TArray::operator=(TArray& X)
{
   if(this == &X) return *this;              // перевірка
   самоприсвоювання.
   if(size==X.size)              // глибоке копіювання
                                 // (вказівник не копіюється!)
   for (long i=0; i<X.size; i++) p[i] =
   X.p[i];
   else cout<<" Size !"<<endl;
   cout<<endl;
   return *this;
}
```

На екрані будуть виведені наступні рядки.

Адреса = 008803A0

Адреса = 00880340

1111111111

1111111111

Зверніть увагу на два моменти. По-перше, у перевантаженому операторі присвоювання виконується не побітове, а глибоке копіювання. Адреса об’єкта у залишається незмінною, міняється лише зміст. По-друге, у функції operator=() передбачена перевірка самоприсвоювання. Це дозволяє коректно обробляти вирази виду x = x;

Незважаючи на зовнішню схожість перевантаженого оператора присвоювання і конструктора копіювання, між ними існує принципова різниця. При виклику конструктора копіювання створюється новий об’єкт, що ініціалізується раніше існуючим об’єктом. При присвоюванні обидва об’єкти уже існують.

В арифметичних обчисленнях дуже корисні скорочені оператори присвоювання. Розумно спробувати перевантажити їх для знову створюваних класів. При цьому варто врахувати обмеження, що накладаються на оператор присвоювання й арифметичних операторів. Причому, як правило, перевантажені арифметичні оператори реалізуються за допомогою скорочених операторів присвоювання.

```
class TComplex{
   double Re;
   double Im;
public:
   TComplex(double x, double y):Re(x),Im(y){ }
   TComplex(TComplex& z){ Re = z.Re; Im = z.Im;}
   ~TComplex(){}
   void printTComplex();
   const TComplex operator+=(const TComplex& z)  // додавання
   {
       Re = Re+z.Re;
       Im = Im+z.Im;
```

```
        cout<<Operator += ";
        return *this;
    }
    const TComplex operator+(const TComplex& z)   // додавання
    {
        TComplex w=*this;
        w+=z;
        cout<<"Operator + ";
        return w;
    }
};

int main()
{
    TComplex u(1,1),v(2,2),z(3,3);
    u+=v;
    u.print();
    z=u+v;
    z.printTComplex();
    return 0;
}

void TComplex::print()
{
 cout<<Re<<" +i"<<Im);
}
```

На початку цієї теми ми виділили оператор [] як особливий клас операторів для перевантаження — первинний. Розглянемо, як здійснюється зміна його змісту.

Бінарний оператор доступу до члена масиву перевантажується за допомогою наступної операторної функції, що повинна бути нестатичним членом класу.

тип_ що повертається_значення& ім’я_класу::operator[](інтегральний_тип i)

```
{
  // ...
}
```

Відзначимо два моменти. По-перше, оскільки операція доступу застосовується до індексованих масивів, параметр операторної функції повинний бути цілочисловим. По-друге, елементи масиву можуть стояти як у лівої, так і в правій частині оператора присвоювання. Отже, функція operator[]() повинна повертати посилання або вказівник.

Хоча зовні функція operator[]() виглядає унарною, насправді вона є бінарною. Її перший параметр явно задає індекс елемента, а другий параметр, що представляє собою вказівник this на об’єкт, де виконується виклик, передається неявно.

От типовий приклад використання цього оператора. Для демонстрації прикладу використаємо клас, що містить в собі розмірність двовимірного масиву дійсних чисел (кількість стовпців та кількість рядків) та вказівник на сам масив.

```
class TMatrix{
   int n;    //кількість рядків
   int m;    //кількість стовпців
   double *p; //масив
public:
   TMatrix(int x,int y);     //конструктор з параметром
   TMatrix();        //конструктор без параметрів

   ~TMatrix() { if (p!=NULL) delete p; p=NULL; }  //деструктор
   double* operator[](int i) { return p+i*m; }
   //перевантажений оператор []
   void output();         //виведення матриці на екран
```

```
};

TMatrix::TMatrix(int x,int y):n(x),m(y)
{
   p=new double[n*m];
   for (int i=0; i<n; i++)
   for (int j=0; j<m; j++)
   if(i==j)p[i*m+j]=1; else p[i*m+j]=0;
}

void TMatrix::output()
{
   for (int i=0; i<n; i++)
   {
     for (int j=0; j<m; j++)
     {
       cout<<" ", (*this)[i][j];
     }
   cout<<endl;
   }
   cout<<endl;
}

int main()
{
   TMatrix c(3,3);
   c[1][2]=2.0;
   c[2][1]=2.0;
   c.output();

   return 0;
}
```

Як відомо, у мові C++ за замовчуванням не передбачена перевірка виходу індексу масиву за припустимі межі. З цієї причини визначення класу TMatrix варто було б доповнити генеруванням відповідних виняткових ситуацій.

Розглянемо також особливості перевантаження оператора (). Об’єкти, що містять операторну функцію operator()(), називаються функціями-об’єктами, чи функторами. Ця функція може одержувати довільну кількість параметрів і повертати значення будь-яких типів. Такі об’єкти бувають корисними при виконанні операцій, зв’язаних з декількома індексами. Як відзначалася вище, операторна функція operator()() повинна бути нестатичним членом класу.

Проілюструємо створення функторів програмою, у якій функція operator()() повертає заданий рядок матриці.

```
class TMatrix;
class Str
 {
    int m;
    double *q;
 public:
    Str(int x):m(x) {q = new double[m];}

    double& operator[](int i) { return
    *(q+i); }
    void output();
    friend class TMatrix;
 };

class TMatrix
 {
    int n;
    int m;
     double *p;
```

```
public:
   TMatrix(int x,int y);

   ~TMatrix() { if (p!=NULL) delete p;
   p=NULL; }
   Str operator()(int);

   double* operator[](int i) { return p+i*m; }
   void output();
};

TMatrix::TMatrix(int x,int y):n(x),m(y)
{
   p=new double[n*m];
   for (int i=0; i<n; i++)
   for (int j=0; j<m; j++)
   if(i==j)  p[i*m+j]=1; else p[i*m+j]=0;
}

Str TMatrix::operator()(int i)
{
   Str s(3);

   for (int j=0; j < m; j++) s[j]=(*this)
   [i][j]; return s;
}

void TMatrix::output()
{
   for (int i=0; i<n; i++)
   {
     for (int j=0; j<m; j++)
     {
```

```
      cout<<"  ", (*this)[i][j];
    }
    cout<<endl;
  }
    cout<<endl;
}

void Str::output()
{
  for(int j = 0; j < m; j++)
  cout<<" ",q[j];
}


int main()
{
  TMatrix c(3,3);
  Str s(3);
  c[1][2]=2.0;
  c[2][1]=2.0;
  cout<<"Матриця: "<<endl;
  c.output();
  cout<<"Рядок: "<<endl;
  s=c(1);
  s.output();

  return 0;
}
```

Виконавши цю програму, ми одержимо наступний результат.

Матриця:

1.000 0.000 0.000

0.000 1.000 2.000

0.000 2.000 1.000

Рядок:

0.000 1.000 2.000

Зверніть увагу на те, що в класі Str операторна функція operator[]() повертає посилання на число типу double. Це дозволяє використовувати вираження s[j] у лівій частині оператора присвоювання.

Питання для самоконтролю:

1. Які операції можна перевантажити?
2. Які операції не можна перевантажити?
3. Як виглядає синтаксис операторних функцій?
4. Які вимоги висуваються до операторних функцій?
5. Які обмеження накладаються на застосування перевантажених операторів?
6. Назвіть особливість операторних функцій-членів, що перевантажують унарний оператор.
7. Назвіть дві форми перевантаження операторів інкремента і декремента.
8. Назвіть особливості перевантаження операторів інкремента і декремента.
9. Як перевантажується оператор посилання на член об’єкта?
10. Які обмеження накладаються на функцію operator->()?
11. Назвіть особливості бінарних операторних функцій-членів.
12. Назвіть вимоги до перевантаженого оператору присвоювання.
13. Назвіть вимоги до перевантаженого операторної функції operator,().
14. Як виглядає перевантажена операторна функція для оператору new.
15. Як перевантажується бінарний оператор доступу до члена масиву.

Тема 6
Успадкування. Одинарне успадкування

Успадкування – один з трьох фундаментальних механізмів об’єктно-орієнтованого програмування, оскільки саме завдяки йому уможливлюється створення ієрархічних класифікацій. Використовуючи механізми успадкування, можна розробити загальний клас, який визначає характеристики, що є властиві множині взаємопов’язаним між собо юелементам. Цей клас потім може успадковуватися іншими, вузькоспеціалізованими класами з додаванням у кожен з них своїх, властивих тільки їм унікальних особливостей. У стандартній термінології мови програмування C++ початковий клас називається базовим. Клас, який успадковує базовий клас, називається похідним. Похідний клас можна використовувати як базовий для іншого похідного класу. Затаким механізмом якраз і будується багаторівнева ієрархія класів.

В основі механізму, що дозволяє створювати ієрархії класів, лежить принцип успадкування. Клас, що лежить в основі ієрархії, називається базовим. Класи, що успадковують властивості базового класу, називаються похідними. Похідні класи, у свою чергу, можуть бути базовими стосовно своїх спадкоємців, що в результаті приводить до ланцюжка успадкування. Процес утворення похідного класу на основі базового називається виводом класу. З одного базового класу можна вивести декілька похідних. Крім того, похідний клас може бути спадкоємцем декількох базових класів. Успадкування буває одинарним і множинним. При одинарному успадкуванні в кожного похідного класу є лише один базовий клас, а при множинному — декілька. Розглянемо механізм успадкування ближче.

Мова програмування C++ підтримує механізм успадкування, який дає змогу в оголошенні класу вбудовувати інший клас. Для цього базовий клас задається під час оголошення похідного класу. Щоб зрозуміти сказане, почнемо з конкретного

прикладу. Розглянемо клас vehicle, який загалом визначає дорожній транспортний засіб. Його члени даних дають змогу зберігати наявну кількість коліс і можливу кількість пасажирів, яких може перевозити транспортний засіб:

```
class vehicle {
     int wheel; // кількість коліс
     int passenger; // кількість пасажирів
     public:
          void setWheel(int f) { wheel = f; }
          int getWheel() { return wheel; }
          void setPassenger(int t) { passenger = t; }
          int getPassenger() { return passenger; }
};
```

Таке загальне визначення дорожнього транспортного засобу є частиною визначення будь-якого конкретного типу автотранспорту. Наприклад, у наведеному нижче оголошенні класу шляхом успадкування класу vehicle створюється клас truck – вантажних автомобілів:

```
class truck : public vehicle{
     float loadCapacity; // вантажомісткість у кубічних метрах
     public:
          void setCapacity(int h) { mistkist = h; }
          int getCapacity() { return mistkist; }
}
```

Той факт, що клас truck успадковує клас vehicle, означає, що клас truck успадковує весь вміст класу vehicle. До вмісту класу vehicle клас truck додає свого члена даних loadCapacity, а також функції-члени, необхідні для його підтримки. Зверніть увагу на те, як успадковується клас vehicle. Загальний формат для забезпечення механізму успадкування має такий вигляд:

class ім’я_похідного_класу: доступ ім’я_базового_класу{

тіло нового класу

}

У такому оголошенні похідного класу елемент доступ є необов’язковим. У разі потреби він може бути виражений одним із специфікаторів доступу: public, private або protected. Покищо у визначеннях усіх успадкованих класів будемо використовувати специфікатор доступу public. Це означає, що всі public-члени базового класу також будуть public-членами похідного класу. Отже, у наведеному вище прикладі члени класу truck мають доступ до відкритих функцій-членів класу vehicle, неначе вони (ці функції) були оголошені в тілі класу truck. Проте клас truck не має доступу до private-членів класу vehicle. Наприклад, для класу truck закритий доступ до членів даних wheel і passenger. Розглянемо кодпрограми, яка демонструє механізм успадкування двох підкласів класу vehicle: truch і car.

```
// Оголошення базового класу транспортних засобів
class vehicle {
     int wheel; // кількість коліс
     int passenger; // кількість пасажирів
     public:
          void setWheel(int f) { wheel = f; }
          int getWheel() { return wheel; }
          void setPassenger(int t) { passenger = t; }
          int getPassenger() { return passenger; }
};
// Оголошення похідного класу вантажівок.
class truck : public vehicle{
     float loadCapacity; // вантажомісткість у кубічних метрах
     public:
          void setCapacity(int h) { mistkist = h; }
          int getCapacity() { return mistkist; }
          void Show() ;
}
enum type {car, van, wagon};   // перерахунковий тип даних
// оголошення похідного класу автомобілів.
```

```
class car : public vehicle {
     enum type carType;
     public:
          void setType(type t) { carType = t; }
          enum type getType() { return carType; }
          void Show() ;
};
void truck::Show() {      // метод виведення інформації про об’єкт класу truck на екран
cout << "Транспортний засіб: " << endl;
cout << "коліс: " << getWheel() << " шт" << endl;
cout << "пасажирів: " << getPassenger() << " осіб" << endl;
cout << "вантажомісткість: " << loadCapacity << " мкуб" << endl; }
void car::Show() { // метод виведення інформації про об’єкт класу car на екран
cout << "Транспортний засіб: " << endl;
cout << "коліс: " << getWheel() << " шт" << endl;
cout << "пасажирів: " << getPassenger() << " осіб" << endl;
cout << "тип: ";
switch(getType()) {
     case van: cout << "van" << endl; break;
     case car: cout << "car" << endl; break;
     case wagon: cout << "wagon" << endl; }
}

int main() {
truck ObjT, ObjF;
car ObjG;
// ініціалізація об’єкта типу truck
ObjT.setWheel(18);
ObjT.setPassenger(2);
ObjT.setCapacity(160);
```

```
// Ініціалізація об’єкта типу truck
ObjT.setWheel(6);
ObjT.setPassenger(3);
ObjT.setCapacity(80);
// виведення інформації про об’єкт типу truck
ObjT.Show();
ObjF.Show();
// ініціалізація об’єкта типу car
ObjG.setWheel4);
ObjG.setPassenger(6);
ObjG.setType(van);
// Виведення інформації про об’єкт типу car
ObjG.Show(;
return 0;
}
```

Розглянемо, як відбувається управління механізмом доступу до членів базового класу. Якщо один клас успадковує інший, то члени базового класу стають членами похідного. Статус доступу до членів базового класу у похідному класі визначається специфікатором доступу, який використовують для успадкування базового класу. Специфікатор доступу до членів базового класу виражається одним з ключових слів: public, private або protected. Якщо специфікатор доступу не вказано, то за замовчуванням використовується специфікатор private, колий деться про успадкування типу class. Якщо базовий клас успадковується, як public-клас, то всі його public-члени стають public-членами похідного класу.

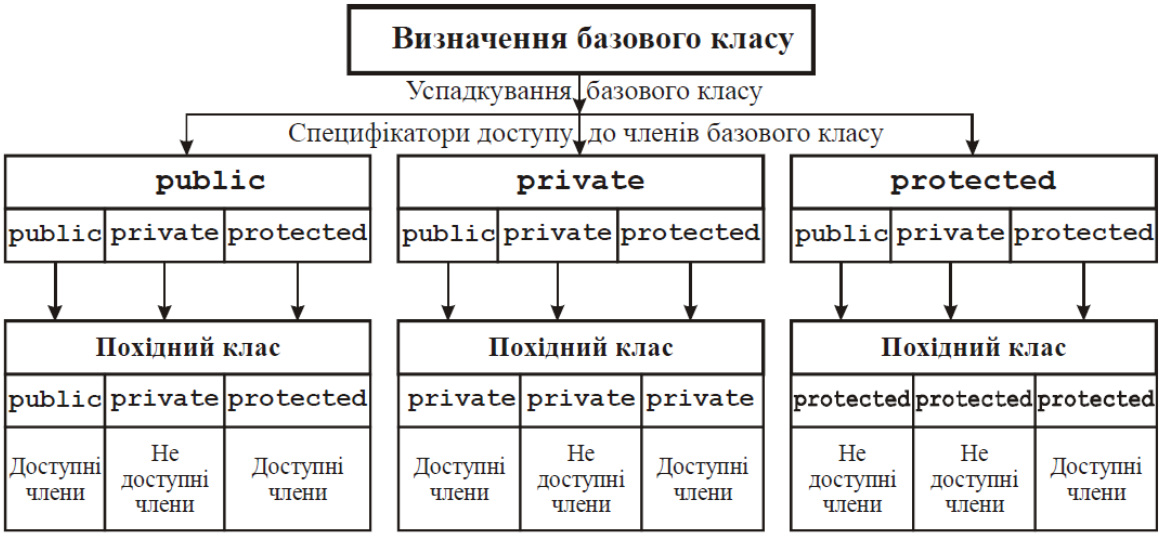

Рисунок 2 – Успадкування специфікаторів доступу

В усіх випадках private-члени базового класу залишаються закритими у межах цього класу і недоступні для членів похідного. Наприклад, у наведеному нижче коді програми public-члени класу baseClass стають public-членами класу derived. Отже, вони будуть доступними і для інших частин програми.

Продемонструємо розмежування доступу на такому навчальному прикладі.

```
class baseClass { // оголошення базового класу
     int c, d;
     public:
          void setB(int a, int b) { c = a; d = b; }
          void showB() { cout << "c= " << c << "; d= " << d << endl;
}
};
class derived : public baseClass {  // оголошення похідного класу
     int f;
          public: derived(int x) { f = x; }
          void showF() {
          showB();
          cout << "f= " << f << endl; }
};
```

```
int main() { derived ObjD(3); // доступ до членів класу
baseClass ObjD.setB(1, 2); // доступ до членів класу
baseClass ObjD.showB();
cout << endl;
// доступ до члена класу derived
ObjD.showF();
return 0;
}
```

Результатом роботи такоє програми є наступна інформація виведена на еран:

c = 1; d = 2 *// виведено дані базового класу*

c = 1; d = 2 *// виведено дані успадкованого класу*

f = 3

Питання для самоконтролю:

1. Що називається повторним використанням коду?
2. Який механізм дозволяє створювати ієрархії класів?
3. Який клас називається базовим?
4. Який клас називається похідним?
5. Що називається виводом класу?
6. Назвіть види успадкування залежно від кількості базових класів.
7. Опишіть механізм успадкування.
8. Назвіть види успадкування залежно від рівня доступу.
9. Опишіть відкрите успадкування.
10. Опишіть захищене успадкування.
11. Опишіть закрите успадкування.

Тема 7
Множинне успадкування. Механізми успадкування декількох базових класів

Як уже було сказано вище, успадкування буває одинарним і множинним. При одинарному успадкуванні в кожного похідного класу є лише один базовий клас, а при множинному – декілька. Розглянемо механізм множинного успадкування ближче. Похідний клас може успадковувати два або більше базових класів. Наприклад, у навчальному прикладі клас derivedClass успадковує обидва класи base1 і base2.

```
class base1 { // оголошення базового класу
     protected:
          int x;
     public:
          void showX() { cout << x << endl; }
};
class base2 { // оголошення базового класу
     protected:
          int y;
     public:
          void showY() { cout << y << endl; }
};
// Успадкування двох базових класів.
// Оголошення похідного класу
class derivedClass : public base1, public base1 {
     public:
          void setXY(int c, int d) {x = c; y = d; }
};
int main() {
derived ObjD;        // створення об’єкта класу
ObjD.setXY(10, 20);       // член класу
```

```
derived ObjD.showX();      // функція з класу base1
ObjD.showY();          // функція з класу base2
return 0;
}
```

Як видно з цього коду програми, щоб забезпечити успадкування декількох базових класів, необхідно через кому перерахувати їх імена у вигляді списку. При цьому потрібно вказати специфікатор доступу для кожного успадкованого базового класу.

З’ясуємо, у чому полягають собливості використання конструкторів і деструкторів при реалізації механізму успадкування. Базовий і/або похідний клас може містити конструктор і/або деструктор. Важливо розуміти послідовність, у якій виконуються ці функції при створенні об’єкта похідного класу і його (об’єкта) руйнування. Для розуміння сказаного розглянемо такий приклад.

```
class basedClass { // Оголошення базового класу
     public:
          basedClass() { cout << "Creating basedClass-object" <<
endl; }
          ~basedClass() { cout << "Destructing basedClass-object" <<
endl; }
};
// Оголошення похідного класу
class derivedClass : public basedClass {
     public:
          derivedClass() { cout << "Creating derivedClass-object" <<
endl; }
          ~derived() { cout << "Destructing derivedClass-object" <<
endl; }
};
int main() {
derivedClass ObjD; // створення об’єкта класу
```

```
// Ніяких дій, окрім створення і руйнування об’єкта ObjD.
return 0;
}
```

Як зазначено у коментарях для функції main(), ця програма тільки створює і відразу руйнує об’єктObjD, який має тип derived. Внаслідок виконання ця програма відображає на екрані такі результати:

Creating basedClass-object

Creating derivedClass-object

Destructing derivedClass-object

Destructing basedClass-object

Аналізуючи отримані результати, бачимо, що спочатку виконується конструктор класу basedClass, а за ним – конструктор класу derivedClass. Потім (через руйнування об’єкта ObjD у цьому коді програми) викликається деструктор класу derivedClass, а за ним – деструктор класу basedClass. Результати описаного вище експерименту можна узагальнити. При створенні об’єкта похідного класу спочатку викликається конструктор базового класу, а за ним – конструктор похідного класу. Під час руйнування об’єкта похідного класу спочатку викликається його "рідний" конструктор, а за ним – конструктор базовогокласу.

Конструктори викликаються у порядку походження класів, а деструктори - у зворотньому.

Цілком логічно, що функції конструкторів виконуються у порядку походження їх класів. Оскільки базовий клас "нічого не знає" ні про який похідний клас, операції з ініціалізації, які йому потрібно виконати, не залежать від операцій ініціалізації, що виконуються похідним класом, але, можливо, створюють попередні умови для подальшої роботи. Тому конструктор базового класу повинен виконуватися першим. Така ж логіка простежується і у тому, що деструктори виконуються у порядку, зворотному порядку походження класів. Оскільки базовий клас знаходиться в основі похідного класу, то руйнування

першого передбачає руйнування другого. Отже, деструктор похідного класу є сенс викликати до того, як об’єкт буде повністю зруйнований. При розширеній ієрархії класів (тобто за ситуації, коли похідний клас стає базовим класом для ще одного похідного) застосовується таке загальне правило: конструктори викликаються у порядку походження класів, а деструктори – у зворотному порядку. Для розуміння сказаного розглянемо такий приклад.

```
class basedClass { // оголошення базового класу
     public:
          baseClass() { cout << "Creating basedClass-object" <<
endl; }
          ~baseClass() { cout << "Destructing baseddClass-object" <<
endl; }
};
// Оголошення похідного класу
class derived1 : public baseClass {
     public:
          derived1() { cout << "Creating derived1-object" << endl; }
          ~derived1() { cout << "Destructing derived1-object" <<
endl; }
};
// Оголошення похідного класу
class derived2 : public derived1 {
     public:
          derived2() { cout << "Creating derived2-object" << endl; }
          ~derived2() { cout << "Destructing derived2-object" <<
endl; }
};
int main() {
derived2 Obj;  // створення об’єкта класу
// створення і руйнуванняоб’єкта Obj
```

```
return 0;
}
```

Внаслідок виконання ця програма відображає на екрані такі результати:

Creating basedClass-object

Creating derived1-object

Creating derived2-object

Destructing derived2-object

Destructing derived1-object

Destructing basedClass-object.

Те саме загальне правило застосовується і у ситуаціях, коли похідний клас успадковує декілька базових класів.

Проемонструємо послідовності виконання конструкторів і деструкторів під час успадкування декількох базових класів

```
class basedClass1 { // оголошення базового класу1
     public:
          baseA() { cout << "Creating basedClass1-object" << endl; }
          ~baseA() { cout << "Destructing basedClass1-object" <<
endl; }
};
class basedClass2 { // оголошення базового класу2
     public:
          baseA() { cout << "Creating basedClass2-object" << endl; }
          ~baseA() { cout << "Destructing basedClass2-object" <<
endl; }
};
// оголошення похідного класу
class derivedClass : public basedClass1, public basedClass1 {
     public:
```

```
        baseA() { cout << "Creating derivedClass-object" <<
endl; }
        ~baseA() { cout << "Destructing derivedClass-object" <<
endl; }
};
};

int main() {
derivedClass Obj; // створення об’єкта похідного класу
// створення і руйнуванняо б’єкта Obj
return 0;
}
```

Внаслідок виконання ця програма відображає на екрані такі результати:

Creating basedClass1-object

Creating basedClass2-object

Creating derivedClass-object

Destructing derivedClass-object

Destructing basedClass2-object

Destructing basedClass1-object

Як бачите, конструктори викликаються у порядку походження їх класів, зліва на право, у порядку їх задавання у переліку успадкування для класу derivedClass. Деструктори викликаються у зворотному порядку, зправа на ліво. Це означає, що якби клас base2 знаходився передкласом base1 у переліку класу derivedClass, тобто відповідно до такої настанови: class derivedClass : public base2, public base1, то результати виконання попереднього коду програми були б такими:

Creating basedClass2-object

Creating basedClass1-object

Creating derivedClass-object

Destructing derivedClass-object

Destructing basedClass1-object

Destructing basedClass2-object

До тепер жоден з попередніх прикладів не містив конструкторів, для яких потрібно було б передавати аргументи. У випадках, коли конструктор тільки похідного класу вимагає передачі одного або декількох аргументів, до статньо використовувати стандартний синтаксис параметризованого конструктора. Але як передати аргументи конструктору базового класу? У цьому випадку необхідно використовувати розширену форму оголошення конструктора похідного класу, у якому передбачено можливість передачі аргументів одному або декільком конструкторам базового класу. Загальний формат розширеного оголошення конструктора похідного класу маєтакий вигляд:

конструктор_похідного_класу(перелік_аргументів): base1(перелік_аргументів), base2(перелік_аргументів), ... baseN(перелік_аргументів); {

тіло конструктора похідного класу}

Тут елементи base1, ..., baseN означають імена базових класів, що успадковуються похідним класом. Зверніть увагу на те, що оголошення конструктора похідного класу відділяється від переліку базових класів двокрапкою, а імена базових класів розділяються між собою комами (у разі успадкування декількох базових класів). Для демонстрації опсаного принципу розгялнемо наступну програму.

```
class baseClass {     // оголошення базового класу
     protected:
          int c;
     public:
          baseClass(int x) { c = x; cout << "Creating basedClass-
object" << endl; }
          ~baseClass() { cout << "Destructing basedClass-object" <<
endl; }
};
```

```
// Оголошення похідного класу
class derivedClass : public baseClass {
     int d;
     public: // Клас derivedClass використовує параметр x, а параметр y
передається конструктору класу baseClass.
          derivedClass(int x, int y) : baseClass(y) {
               d = x;
               cout << "Creating derivedClass-object" << endl;
          }
          ~derived() { cout << "Destructing derivedClass-object" <<
endl;
          }
          void show(string s) {
               cout << s << "c= " << c << "; d= " << d << endl;
          }
};
int main() {
derivedClass Obj(12, -1);
Obj.show("Base Class: "); // відображає числа 12 -1
return 0;
}
```

У цьому коді програми конструктор класу derived оголошується з двома параметрами x і y. Проте конструктор derivedClass() використовує тільки параметр x, а параметр y передається конструктору baseClass(). У загальному випадку конструктор похідного класу повинен оголошувати параметри, які приймає його клас, а також ті, які потрібні базовому класу. Як це показано у наведеному вище прикладі, будь-які параметри, що потрібні базовому класу, передаються йому у переліку аргументів базового класу, які вказуються після двокрапки. Розглянемо приклад програмного коду, де продемонстровано механізм передачі параметрів конструкторам декількох базових класів.

```
class baseClass1 {        // оголошення базового класу
     protected:
          int c;
     public:
          baseClass1(int x) {
               c = x;
               cout << "Creating baseClass1-object"" << endl;
          }
          ~baseClass1() {
               cout << "Destructing baseClass1-object" << endl;
          }
};
class baseClass2 {        // оголошення базового класу
     protected:
          int c;
     public:
          baseClass2(int x) {
               c = x;
               cout << "Creating baseClass2-object"" << endl;
          }
          ~baseClass1() {
               cout << "Destructing baseClass2-object" << endl;
          }
};
// Оголошення похідного класу
class derivedClass : public baseClass1, public baseClass2 {
     int d;
     public:
          derivedClass (int x, int y, int z): baseA(y), baseB(z) {
               d = x;
               cout << "Creating derivedClass-object" << endl;
```

```
        }
        ~derived() {
            cout << "Destrucring derivedClass-object" << endl;
        }
        void show(string s) {
        cout << s << "c= " << c << "; d= " << d << "; f= " << f <<
endl;
        }
};
int main() {
derived Obj(8, 9, 10);
Obj.show("Based class"); // Відображає числа c= 8; d= 9; f= 10
return 0;
}
```

Важливо розуміти, що аргументи для конструктора базового класу передаються через аргументи, які приймаються конструктором похідного класу. Навіть якщо конструктор похідного класу не використовує ніяких аргументів, то він повинен оголосити один або декілька аргументів, якщо базовий клас приймає один або декілька аргументів. У цій ситуації аргументи, що передаються похідному класу, "транзитом" передаються базовому. Наприклад, у наведеному нижче коді конструктори baseClass1() і baseClass2(), на відміну від конструктора класу derivedClass, приймають аргументи.

```
class baseClass1 {        // оголошення базового класу
    protected:
        int c;
    public:
        baseClass1(int x) {
            c = x;
            cout << "Creating baseClass1-object"" << endl;
        }
```

```
          ~baseClass1() {
               cout << "Destructing baseClass1-object" << endl;
          }
};
class baseClass2 {        // оголошення базового класу
     protected:
          int c;
     public:
          baseClass2(int x) {
               c = x;
               cout << "Creating baseClass2-object"" << endl;
          }
          ~baseClass1() {
               cout << "Destructing baseClass2-object" << endl;
          }
};
// Оголошення похідного класу
class derivedClass : public baseClass1, public baseClass2 {
          public:     /* конструктор класу derived не використовує
параметрів, але повинен оголосити їх, щоб передати конструкторам базових
класів. */
               derivedClass( int x, int y) : baseClass1(x),
baseClass2(y) {
                    cout << "Creating derivedClass-object" << endl;
               }
               ~derived() {
                    cout << "Destructing derivedClass-object " <<
endl;
               }
               void show(string s) { cout << s << "c =" << c << ";
f= " << f << endl; }
};
```

```
int main() {
derived Obj (5, -17);
ObjD.showB("Base class: "); // відображає числа c= 5; f= -17
return 0;
}
```

Конструктор похідного класу може використовувати будь-які (або всі) параметри, які ним оголошені для прийняття, незалежно від того, чи передаються вони (один або декілька) базовому класу. Іншими словами, той факт, що деякий аргумент передається базовому класу, не заважає його використанню і самим похідним класом.

Наприклад, наведений нижче фрагмент коду програми є абсолютно допустимим:

```
class derivedClass : public baseClass {
     int d;
          public: /* Клас derived використовує обидва параметри x і y, а також передає їх класу baseClass */
               derivedClass(int x, int y) : baseClass(x, y) {
                    d = x*y;
                    cout << "Creating derivedClass-object" << endl; }
//. . . }
```

При передачі аргументів конструкторам базового класу необхідно мати на увазі, що аргумент, який передається, може містити будь-який (дійсний на момент передачі) вираз, що містить виклики функцій і змінних. Це можливо завдяки тому, що мова C++ дає змогу виконувати динамічну ініціалізацію даних.

Питання для самоконтролю:

1. Як відбуваються виклики конструкторів при успадкуванні класів?
2. Як відбуваються виклики деструкторів при успадкуванні класів?

3. Як передаються параметри у констуктори, якщо клас успадкований від кількох класів?
4. Назвіть недоліки множинного успадкування.

Тема 8
Оголошення класів у заголовочних файлах

Усі класи, які ми використовували до цього часу, були досить простими, тому ми описували методи безпосередньо всередині тіла класів. Наприклад:

```
class Date{     // клас, що описує дату
     private:
          int m_day;      // день
          int m_month;    // місяць
          int m_year;     // рік
     public:
          Date(int day, int month, int year){      // конструктор з параметром
               m_day = day;
               m_month = month;
               m_year = year;
          }
          void setDate(int day, int month, int year){  // set()-метод
               m_day = day;
               m_month = month;
               m_year = year;
}
          int getDay() { return m_day; }           // get()-методи
          int getMonth() { return m_month; }
          int getYear(){ return m_year; }
};
```

Однак, як тільки класи стають більшими і складнішими, наявність всіх методів всередині тіла класу може утруднити його управління і роботу з ним. Використання вже написаного класу вимагає розуміння тільки його відкритого інтерфейсу, а не того, як він реалізований. На допомогу уцьому випадку приходть

можливість відокремлення оголошення від реалізації. C ++ надає спосіб відокремити «оголошення» від «реалізації». Це робиться шляхом визначення методів поза тілом самого класу. Для цього просто визначають методи класу, ніби це звичайні функції, але в якості префікса додають до імені функції ім’я класу з оператором дозволу області видимості :: (того, що використовується з просторами імен). Ось наш клас Date з конструктором Date() і методом setDate (), оголошеними поза тілом класу. Зверніть увагу, прототипи цих функцій все ще знаходяться всередині тіла класу, але їх фактична реалізація перебуває за його межами:

```
class Date{
     private:
          int m_day;
          int m_month;
          int m_year;
     public:
          Date(int day, int month, int year);
          void SetDate(int day, int month, int year);
          int getDay() { return m_day; }
          int getMonth() { return m_month; }
          int getYear() { return m_year; }
};
// Конструктор класу Date
Date::Date(int day, int month, int year)
{
SetDate(day, month, year);
}
// Метод класу Date
void Date::SetDate(int day, int month, int year)
{
m_day = day;
m_month = month;
```

```
m_year = year;
}
```

Просто, чи не так? Оскільки в багатьох випадках функції доступу можуть складатися всього з одного рядка, то їх зазвичай залишають в тілі класу, хоча перемістити їх за межі класу можна завжди. Ось ще один приклад класу з конструктором, оголошеним ззовні, та списком параметрів ініціалізації.

```
class Mathem{
     private:
          int m_value = 0;
     public:
          Mathem(int value=0): m_value(value) {}
          Mathem& add(int value) { m_value += value; return *this; }
          Mathem& sub(int value) { m_value -= value; return *this; }
          Mathem& divide(int value) { m_value /= value;  return
*this; }
          int getValue() { return m_value ; }
};
```

Конвертуємо у наступний код:

```
class Mathem{
     private:
          int m_value = 0;
     public:
          Mathem(int value=0);
          Mathem& add(int value);
          Mathem& sub(int value);
          Mathem& divide(int value);
          int getValue() { return m_value; }
};
Mathem::Mathem(int value): m_value(value){
}
Mathem& Mathem::add(int value)
{
```

```
m_value += value;
return *this;
}
Mathem& Mathem::sub(int value)
{
m_value -= value;
return *this;
}
Mathem& Mathem::divide(int value){
m_value /= value;
return *this;}
```

Оголошення функцій можна помістити у заголовочні файли, щоб потім мати можливість використовувати ці функції в декількох файлах або навіть в декількох проєктах. Класи в цьому плані нічим не відрізняються від функцій. Визначення класів можуть бути поміщені в заголовки для полегшення їх повторного використання в декількох файлах або проектах. Зазвичай, визначення класу поміщається в заголовки з тим же ім’ям, що й у класу, а методи, оголошені поза тілом класу, поміщаються в файл .cpp з тим же ім’ям, що й у класу. Ось наш Date, але вже розбитий на файли .cpp і .h:

Date.h:

```
#ifndef DATE_H
#define DATE_H
class Date{
     private:
          int m_day;
          int m_month;
          int m_year;
     public:
          Date(int day, int month, int year);
          void SetDate(int day, int month, int year);
          int getDay() { return m_day; }
```

```
        int getMonth() { return m_month; }
        int getYear() { return m_year; }
};
#endif
```

Date.cpp

```
#include "Date.h"
Date::Date(int day, int month, int year){ // конструктор класу Date
    SetDate(day, month, year);
}
void Date::SetDate(int day, int month, int year){ // метод класу Date
    m_day = day;
    m_month = month;
    m_year = year;
}
```

Тепер у будь-який інший файл .h або .cpp, який захоче використовувати клас Date, можна просто додати #include "Date.h". Зверніть увагу, Date.cpp також необхідно додати до компіляції в проєкт, який використовує Date.h, щоб лінкер зміг розібратися з реалізацією класу Date.

При використанні такого типу оголошень класу можуть виникнути настпуні питання:

Питання №1: Хіба визначення класу в заголовному файлі не порушує правило одного визначення?

Ні. Класи - це призначені для користувача типи даних, які звільняються від визначення тільки в одному місці. Тому клас, оголошений у заголовочному файлі, можна вільно підключати до інших файлів.

Питання №2: Хіба визначення методів класу в заголовному файлі не порушує правило одного визначення?

Методи, означені всередині тіла класу, вважаються неявно вбудованими. Вбудовані функції звільняються від правила одного визначення. А це означає, що

проблем з визначенням простих методів (таких як функції доступу) всередині самого класу виникати не повинно. Методи, оголошені поза тілом класу, розглядаються як звичайні функції і підпорядковуються правилу одного визначення. Тому ці функції повинні бути визначені в файлі .cpp, а не всередині .h.

Параметри за замовчуванням для методів повинні бути оголошені в тілі класу (в заголовках), де вони будуть видні всім, хто вказав #include для заголовка з класом.

Поділ оголошення класу і його реалізації дуже подібне до використання бібліотек, які використовуються для розширення можливостей вашої програми. Ви також підключали заголовки зі стандартної бібліотеки, такі як iostream, string, vector, array і інші. Зверніть увагу, ви не додавали iostream.cpp, string.cpp, vector.cpp або array.cpp в ваші проекти. Ваша програма потребує тільки в оголошеннях з заголовних файлів, щоб компілятор міг перевірити коректність вашого коду відповідно до правил синтаксису. Однак, реалізації класів, які перебувають в стандартній бібліотеці, містяться в попередньо скомпільованому файлі, який додається на етапі лінкінгу. Ви ніколи не маєте доступу до цього коду.

Поза програмою з відкритим вихідним кодом (де надаються обидва файли: .h і .cpp), більшість сторонніх бібліотек надають тільки заголовки разом з попередньо скомпільований файлом бібліотеки. На це є кілька причин:

- На етапі лінкінгу швидше буде підключити попередньо скомпільовану бібліотеку, ніж виконувати перекомпіляцію кожного разу, коли вона потрібна.
- Захист інтелектуальної власності (розробники не хочуть, аби хтось брав їхній код).
- Наявність власних файлів, розділених на оголошення (файли .h) та реалізацію (файли .cpp) є не тільки хорошою формою змісту коду, але і спрощує створення власних користувацьких бібліотек.

Можливо, у вас виникне бажання помістити всі визначення методів класу в заголовки всередині тіла класу. Хоча це скомпілюється, але тут є кілька нюансів:

- По-перше, як згадувалося вище, це призведе до захаращення визначення вашого класу.
- По-друге, функції, певні всередині класу, є неявно вбудованими. Великі функції, які викликаються з багатьох файлів, можуть сприяти, таким чином, до «роздування» вашого коду.
- По-третє, якщо ви зміните щось у заголовковому файлі, то слід перекомпілювати кожен файл, який містить цей заголовок. Це може мати "ефект метелика", коли одна незначна зміна призводить до перекомпільовування всього проекту, що може бути достатньо довго та повільно. Якщо був змінений тільки файл .cpp, то слід перекомпілювати лише його!

Тому рекомендації до використання заголовочних файлів наступні:

- Класи, призначені для повторного використання, оголошуються у окремому файлі з тим же іменем, що й ім’я класу.
- Тривіальні методи (які будуть повторно використані), оголошуються в класі, у файлі .h.
- Нетривіальні методи оголошуються у файлі .cpp

Питання для самоконтролю:

1. Що міститься у заголовкових файлах із розширенням *.h?
2. Як під’єднати файл із розширенням *.h до проєкту?
3. Чому при використанні стандартних файлів C++ розширення *.h не використовується?
4. З якої причини оголошення класів виносять у окремі файли?
5. У чому перевага оголошення класів у окремих файлах?
6. На якому етапі середовище програмування приєднує заголовкові файли?

7. Чи слід перекомпільовувати весь проєкт, якщо був змінений заголовковий файл?
8. Чи слід перекомпільовувати весь проєкт, якщо був змінений файл головної програми?
9. Чи слід перекомпільовувати весь проєкт, якщо був змінений файл із реалізацією методів класу (має розширення *.cpp)?

Лабораторна робота №1

Тема: Вступ до об’єктно-орієнтованого програмування. Програмування класів

Мета: Навчитись проектувати та оголошувати класи, створювати об’єкти класів

Теоретичні відомості: тема 1

Завдання для практичного виконання

Варіант 1. Описати клас Square, з властивістю довжина сторони квадрата. Методи: set- та get-, обчислення площі та периметра. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 2. Описати клас Circle, з властивістю радіус. Методи: set- та get-, обчислення площі круга та довжини кола. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 3. Описати клас Date з властивостями день, місяць, рік. Методи: set- та get-, попередній день, наступний день. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 4. Описати клас Equation, з властивостями a, b, c. Методи: set- та get-, розв’язування рівняння виду ax+b=c. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 5. Описати клас Time, з властивостями година, хвилина, секунда. Методи: set- та get-, збільшення значення часу на 1 секунду, на 1 хвилину, на 1 годину. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 6. Описати клас Rectangle, з властивостями довжина, ширина. Методи: set- та get-, обчислення периметру та площі. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 7. Описати клас Date з властивостями день, місяць, рік. Методи: set- та get-, наступний день, наступний місяць. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 8. Описати клас SquareEquation, з властивостями a, b, c. Методи: set- та get-, розв’язування рівняння виду $ax^2+bx+c= 0$. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 9. Описати клас Triangle, з властивостями 3 сторони трикутника. Методи: set- та get-, знаходження площі та периметра трикутника. Написати програму, яка демонструватиме роботу з цим класом.

Варіант 10. Описати клас Calculation, з властивостями число_1, число_2. Методи: set- та get-, обчислення суми, різниці, добутку, частки цих двох чисел. Написати програму, яка демонструватимероботу з цим класом.

Лабораторна робота №2

Робота з класами. Методи класу

Мета: Навчитись проектувати та оголошувати власні класи, а також модифікувати уже розроблені

Теоретичні відомості: тема 1-2

Завдання для практичного виконання

1. Розгляньте оголошення класу ZooAnimal, подане нижче.

```
class ZooAnimal{
     private:
          string name;
          int cageNumber;
          int weightDate;
          int weight;
     public:
          void Destroy (); // destroy function
          string reptName ();
          int daysSinceLastWeighed (int today);
     };
```

2. У цьому класі пропущено прототип для функції Create. Ця функція повинна мати наступні параметри: рядок символів для імені та 3 значення цілого типу (саме у такому порядку). Ці параметри повинні отримуватись від об’єкта класу ZooAnimal, коли він викликає метод Create. Create не повертає значення.

2. Запишіть заголовок функції daysSinceLastWeighed, що є членом класу ZooAnimal. Параметром цієї функції є одне ціле значення, а повертає вона ціле значення - кількість днів від коли тварина була востаннє зважена.

```
void ZooAnimal::Destroy (){
delete [] name;
```

```
}
// -------- member function to return the number of days
// -------- since the animal was last weighed   {       }
```

3. У функції main присутній оператор cout, який має вивести на екран ім’я тварини. Але такий запис не є коректним, оскільки ім’я входить до приватних даних класу. Напишіть функцію, яка буде виводити ім’я тварини (name) на екран.

```
// ========== an application to use the ZooAnimal class
int main () {
ZooAnimal bozo;
bozo.Create ("Bozo", 408, 1027, 400);
 cout << "This animal's name is " << bozo.name << endl;
bozo.Destroy ();
return 0;
}
```

4. Напишіть функцію reptWeightDate для заданого класу. Вона повинна просто повертати дату зважування weightDate.

enum scale {ounces, kilograms};

```
class ZooAnimal{
     private:
      string name;
      int cageNumber;
      int weightDate;
      int weight;
public:
      void Create (string, int, int, int); // create function
      void Destroy (); // destroy function
      void changeWeight (int pounds);
      void changeWeightDate (int today);
      string reptName ();
      int reptWeight ();
      void reptWeight (scale which);
       int reptWeightDate ();
```

```
      int daysSinceLastWeighed (int today);
};
```

5. Створіть для класу ZooAnimal конструктор з параметрами. Конструктор має 4 параметри: рядок символів та 3 цілих числа cageNumber,weightDate, and weight, відповідно.

```
class ZooAnimal{
     private:
      string name;
      int cageNumber;
      int weightDate;
      int weight;
public:
      // constructor function
      ~ZooAnimal (); // destroy function
      void changeWeight (int pounds);
      string reptName ();
      int reptWeight ();
      int daysSinceLastWeighed (int today);
};
```

6. Створіть для класу ZooAnimal конструктор, що створюватиме за замовчуванням об’єкт із наступними параметрами з параметрами: "Nameless" для name, 9999 для whichCage, 101 (January 1) для weighDay, та 100 для pounds.

7. Для заданого класу напишіть функцію int daysSinceLastWeighed (int today), яка рідрахує, скільки днів минуло від останнього зважування. Параметром функції є сьогоднішня дата.

Результатом даного завдання є програма, що демонструє роботу із класом. Для деяких методів не вказано, як описувати реалізацію, тому для них залишаєте тільки оголошення.

Індивідуальні завдання.

Варіант 1. Створити клас поліномів розмірності 3. Членами класу є коефіцієнти полінома. Методами класу є: ввід коефіцієнтів полінома; вивід полінома на екран; обчислення та вивід значення полінома для заданого значення змінної.

Варіант 2. Створити клас для роботи із кільцем. Членами класу є внутрішній та зовнішній радіуси кільця. Методами класу є: ввід кільця із клавіатури; вивід кільця на екран (виведення інформації про об’єкт); обчислення площі та товщини кільця і вивід результатів на екран. Написати програму, що демонструє роботу з класом.

Варіант 3. Створити клас поліномів розмірності 4 . Членами класу є коефіцієнти полінома. Методами класу є: ввід коефіцієнтів полінома; вивід полінома на екран; обчислення та вивід значення полінома для заданого значення змінної.

Варіант 4. Створити клас для роботи із колом та точкою. Членами класу є радіус кола, координати його центру та координати деякої точки. Методами класу є: ввід кола та точки із клавіатури; визначення положення точки відносно кола (в колі чи за його межами) і вивід результату на екран. Написати програму, що демонструє роботу з класом.

Варіант 5.Створити клас трикутників. Членами класу є координати вершин в просторі. Методами класу є: ввід координат вершин із клавіатури; вивід трикутника на екран (виведення інформації про об’єкт); обчислення периметра та площі і вивід результату на екран. Написати програму, що демонструє роботу з класом.

Варіант 6. Створити клас прямокутників. Членами класу є координати вершин на площині. Методами класу є: ввід координат вершин із клавіатури; вивід прямокутника на екран (виведення інформації про об’єкт); обчислення периметра та площі і вивід результату на екран. Написати програму, що демонструє роботу з класом.

Варіант 7. Створити клас прямих на площині. Членами класу є коефіцієнти рівняння прямої ax+by+ c = 0. Методами класу є: ввід прямої з клавіатури; вивід прямої на екран (виведення інформації про об’єкт); обчислення та вивід на екран координат точок перетину із осями. Написати програму, що демонструє роботу з класом.

Варіант 8. Створити клас відрізків на площині. Членами класу є координати кінців відрізка. Методами класу є: ввід відрізка з клавіатури; вивід відрізка на екран (виведення інформації про об’єкт); обчислення та вивід на екран довжини відрізка. Написати програму, що демонструє роботу з класом.

Варіант 9. Створити клас для роботи із колом та точкою. Членами класу є радіус кола, координати його центру та координати деякої точки. Методами класу є: ввід кола та точки із клавіатури; визначення положення точки відносно кола (в колі чи за його межами) і вивід результату на екран. Написати програму, що демонструє роботу з класом.

Варіант 10. Створити клас для роботи із датою. Членами класу є рік, місяць та день місяця. Методами класу є: ввід дати з клавіатури; вивід дати на екран; обчислення та вивід на екран пори року, що відповідає даній даті. Написати програму, що демонструє роботу з класом.

Лабораторна робота №3

Тема: Методи класу. Конструктор та деструктор класу

Мета: Навчитись програмувати дані-методи класу, викликати методи, програмувати та викликати конструктори різних типів для класу, програмувати та викликати деструктори класу

Теоретичні відомості: тема 2

Завдання для практичного виконання

Для кожного завдання розробити клас, створити конструктор ініціалізації, конструктор копіювання та перетворення типів, set-, get-методи. Розробити програму, яка демонструватиме роботу із даним класом.

Варіант 1. Клас RightTrpezoid із наступними атрибутами: висота, бічна сторона та менша основа. Визначити для даного класу функції, що будуть визначати площу та периметр трапеції.

Варіант 2. Клас Purchase із наступними атрибутами: ціна, термін придатності (до якої дати товар придатний для використання). Визначити для цього класу функції, які повертають вартість покупки із знижкою у 15% та перевіряють, чи товар придатний для використання, якщо зараз 2.03.2020.

Варіант 3. Клас Ticket, який має наступні атрибути: дата сеансу, час сеансу, ряд у кінотеатрі (максимально 15 рядів). Визначити для класу функції, які встановлюють вартість даного квитка: для ранішніх сеансів (з 10 до 17 год) ціна = 15 + 1.5 за кожен ряд, для вечірніх сеансів (з 18 до 22 год) ціна = 25 + 2 за кожен ряд.

Варіант 4. Клас Range, де поле first – ціле додатне число, ліва межа діапазону, second – ціле додатне число,права межа діапазону. Пара чисел являє собою напіввідкритий діапазон [first, second). Реалізувати метод перевірки приналежності цілого числа заданому діапазону. Метод здійснює перевірку правильності задання інтервалу, first < second.

Варіант 5. Клас Parallelepipped із 3 атрибутами: довжина ребра 1, довжина ребра 2, довжина ребра 3. Визначити для цього класу методи, що визначають його площу бічної поверхні та об’єм фігури.

Варіант 6. Клас Bill, де поля ціле додатне число – номінал купюри та ціле додатне число –кількістю купюр заданого номіналу. Реалізувати метод summa() – обчислення суми усіх купюр. Метод здійснює перевірку рівності нулю поля second.

Варіант 7. Клас IntUnsNumber, де полями є ціла частина десяткового числа та дробова частина десяткового числа (максимум 3 знаки). Реалізувати методи множення заданого числа на довільне ціле число.

Варіант 8. Клас Prisma із наступними атрибутами: висота та ребро правильного трикутника, що є основою. Визначити для даного класу функції, що будуть визначати площу бічної поверхні та об’єм призми.

Варіант 9. Клас Parallelogram, який має наступні атрибути: довжина більшої сторони, висота, проведена до неї, розмір гострого кута. Визначити для даного класу функції, що будуть визначати площу та периметр паралелограма.

Варіант 10. Клас Date із 3 атрибутами: рік, місяць, день. Визначити для цього класу функції-елементи, які повертають відомості про те, чи високосний рік, і яка пора року.

Лабораторна робота №4

Тема: Програмування задач, що використовують масиви об’єктів

Мета: Навчитись використовувати масиви об’єктів при розв’язуванні задач, ініціалізувати такі масиви, викликати методи для елементів масивів

Теоретичні відомості: тема 3

Завдання для практичного виконання

Варіант 1.

1. Створити клас Storage, який містить наступні поля:
   - Name – назва товару;
   - Type – одиниця вимірювання:
   - Quantity – кількість одиниць товару;
   - Cost – ціна одиниці товару.
2. Написати програму, що використовує даний клас і виконує наступні дії:
   - вводить з клавіатури масив даних Mall що складається з N змінних типу Storage;
   - Виводить на екран ціну та кількість товару, назва якого вводиться з клавіатури, або виводить повідомлення про його відсутність.

Варіант 2.

1. Створити клас Bus, який містить наступні поля:
   - Name – назва пункту призначення;
   - Numer – номер рейсу;
   - Date – дата відправлення;
   - Time – час відправлення.
2. Написати програму, що використовує даний клас і виконує наступні дії:

- вводить з клавіатури масив даних BusStation що складається з n змінних типу Bus;
- виводить на екран всі рейси, які відправляються після 17.00 у задану дату.

Варіант 3.

1. Створити клас Subscriber, який містить наступні поля:
   - Name – прізвище абонента;
   - Init – ініціали абонента;
   - Nomer – номер телефону;
   - Adress – домашня адреса.
2. Написати програму, що використовує даний клас і виконує наступні дії:
   - вводить з клавіатури масив даних Phone, що складається з m змінних типу Subscriber;
   - Виводить на екран прізвище, ініціали та домашню адресу за введеним номером телефону, або виводить повідомлення про його відсутність.

Варіант 4.

1. Написати клас Flight, який містить наступні поля:
   - Name – назва пункту призначення;
   - Numer – номер рейсу;
   - Type – тип літака;
   - Time – час відправлення.
2. Написати програму, що використовує даний клас і виконує наступні дії:
   - Вводить з клавіатури масив даних FlightSchedule, що складається з n змінних типу flight;
   - Виводить на екран всі номери рейсів та час відправлення

літаків, які відправляються в заданий пункт призначення або повідомляє про відсутність таких рейсів.

Варіант 5.

1. Написати клас Tool, який містить наступні поля:
   - Name – назва деталі;
   - Sort – сорт виробу;
   - Date –дата виготовлення
   - Quant – кількість;
   - Cost - ціна деталі.
2. Написати програму, що використовує даний клас і виконує наступні дії:
   - вводить з клавіатури масив даних Order, що складається з т змінних типу Tool;
   - Виводить на екран всі деталі I сорту, які виготовлені раніше дани, заданої користувачем.

Варіант 6.

1. Написати клас Book, який містить наступні поля:
   - Title – назва книги;
   - Author – автори книги;
   - Date –дата друку;
   - Cost - ціна книги.
2. Написати програму, що використовує даний клас і виконує наступні дії:
   - вводить з клавіатури масив даних BookShop, що складається з n змінних типу Book;
   - Виводить на екран всі книги, що були надруковані в заданому році.

Варіант 7.

1. Написати клас Movie, яка містить наступні поля:
    - Name – назва фільму;
    - Date – дата сеансу;
    - Cost - ціна квитка.
    - DayOfWeek – день тижня;
2. Написати програму, що використовує даний клас і виконує наступні дії:
    - вводить з клавіатури масив даних MovieTheater, що складається з n змінних типу Movie;
    - Виводить на екран всі сеанси і дати їх трансляції, які відбудуться в заданий день тижня.

Варіант 8.

1. Написати клас Stuff, яка містить наступні поля:
    - Name – ім’я працівника;
    - Surname – прізвище працівника ;
    - DateOfBirth – дата народження;
    - Position - посада працівника.
    - Payment – заробітна плата працівника
2. Написати програму, що використовує даний клас і виконує наступні дії:
    - вводить з клавіатури масив даних Office, що складається з n змінних типу Stuff;
    - Виводить на екран всіх працівників молодших 40 років, які мають заробітну плату меншу введеної з клавіатури.

Варіант 9.

1. Написати клас Coin, яка містить наступні поля:
    - Country – держава виготовлення;

- Year – рік виготовлення;
- Nominal – номінал;
- Price – ціна;

2. Написати програму, що використовує даний клас і виконує наступні дії:
   - вводить з клавіатури масив даних coincollection, що складається з n змінних типу coin;
   - виводить на екран список монет 1950 року.

Варіант 10.

1. Написати клас Movie, який містить наступні поля:
   - Title – назва відео фільму;
   - Year – рік зйомки фільму;
   - Ganre – жанр фільму;
   - Editor – режисер;

2. Написати програму, що використовує даний клас і виконує наступні дії:
   - вводить з клавіатури масив даних MovieCollection, що складається з n змінних типу Movie;
   - виводить на екран список фільмів за введеним з клавіатури прізвищем режисера.

Лабораторна робота №5

Тема: Програмування задач на використання композицій класів

Мета: Навчитись використовувати контейнерні класи про розробці програм

Теоретичні відомості: тема 4

Завдання для практичного виконання

В усіх завданнях розробити конструктор для об’єктів класу, деструктор, set-, get-методи та програму, що буде демонструвати роботу із створеним класом.

Варіант 1. Розробити клас Computer, який містить об’єкти класів Mouse, Monitor, Keyboard, SystemUnit. Визначити необхідні елементи даних. Створити масив об’єктів класу Computer та визначити комп’ютер із мінімальним відхиленням параметрів від заданих.

Варіант 2. Розробити клас Car, який містить об’єкт класу Engine. У класі Engine визначити дані про об’єм двигуна, потужність та заводський номер. Клас Car додатково містить номер державної реєстрації, марку та колір. Визначити методи для зміни марки, номера та заводського номера двигуна для класу Car.

Варіант 3. Розробити клас File, який містить дані про назву, розмір, дату зміни та розширення файлу. Клас Catalog містить масив елементів класу File. Визначити методи роботи з файлами та каталогами.

Варіант 4. Розробити клас University, який містить прізвище ректора та масив об’єктів Faculty. Клас Faculty містить прізвище декана та масив об’єктів Department. Клас Department містить прізвище завідувача кафедрою та кількість

працівників. Визначити необхідні методи для роботи з елементами даних та підрахувати загальну кількість працівників університету.

Варіант 5. Розробити клас Group, який містить масив об’єктів класу Student. Клас Student містить прізвище, номер залікової книжки та масив оцінок. Визначити необхідні методи для роботи з елементами даних та вивести на екран список із трьох студентів з найвищим середнім балом.

Варіант 6. Клас Shop містить масив об’єктів класу Goods. Клас Goods містить назву товару, свідотство якості та вартість. Клас Buyer містить масив об’єктів класу Goods. Визначити необхідні методи для роботи з елементами даних та здійснити купівлю і визначити вартість товарів згідно преліку.

Варіант 7. Клас Library містить масив об’єктів Book. Клас Book містить прізвище автора, рік видання, назву, кількість сторінок, видавництво. Визначити необхідні методи для роботи з елементами даних та здійснити пошук книги за автором, видавництвом, роком видання та назвою.

Варіант 8. Клас Player містить масив об’єктів класу MovieLibrary. MovieLibrary містить дані про відеофільм: заголовок, жанр, прізвище режисера та тривалість. Визначити необхідні методи для роботи з елементами даних та здійснити пошук фільму за жанром, режисером, тривалістю та назвою.

Варіант 9. Клас Schedule містить масив об’єктів класу Information про час відправлення потягів. У класі Information задано номер потяга, час відправлення, час прибуття до пункту призначення та платформу. Визначити необхідні методи для роботи з елементами даних. Задати поточний час та визначити найближчий потяг у заданому напрямку.

Варіант 10. Створити клас Engine із вказаними об’ємом двигуна та потужністю. Створити клас Truck, який містить об’єкт класу Engine. Додатково задається марка, вантажопідйомність та номер реєстрації. Визначити методи для зміни марки, номера та заводського номера двигуна для класу Truck.

Лабораторна робота №6

Тема: Програмування задач, що використовують перевантаження операторів

Мета: Навчитись здійснювати перевантаження бінарних та унарних операторів, використовувати нове значення оператора у програмах

Теоретичні відомості: тема 5

Завдання для практичного виконання

В усіх завданнях розробити конструктор для об’єктів класу, деструктор, set-, get-методи та програму, що буде демонструвати роботу із створеним класом та дію операторів, для яких було змінено значення.

Варіант 1. Створити клас Fraction, що є відношенням двох цілих чисел. Перевантажити операції додавання, віднімання, множення та ділення.

Варіант 2. Створити клас RealNumber (у закриті частині класу знаходиться дане дійсного типу). Перевантажити оператори () – повертає дробову частину числа, [] – повертає цілу частину числа.

Варіант 3. Розробити клас Binom. Перевантажити операції ==, !=, >=, <=, <, > для порівняння біномів.

Варіант 4. Створити клас Calendar. Перевантажити операцію = для присвоювання дат, +=, -= для додавання та віднімання заданої кількості днів, відповідно, операції ==, !=, >=, <=, <, > для порівняння дат.

Варіант 5. Створити клас IntegerNumber (у закриті частині класу знаходиться дане цілого типу). Перевантажити оператори ^–піднесення до степеня, [] – перевірка числа на парність, ~ -перевірка, чи є число простим.

Варіант 6. Створити клас DecimalFraction (у закриті частині класу знаходяться два числа цілого типу). Перевантажити оператори () – повертає дробову частину числа, [] – повертає цілу частину числа.

Варіант 7. Створити клас CorrectFraction, що є відношенням двох цілих чисел. Перевантажити операції ==, !=, >=, <=, <, > для порівняння дробів.

Варіант 8. Розробити клас PolynomPow3 (степінь 3). Коефіцієнти представити у вигляді поля-масиву класу. Перевантажити операції + додавання двох поліномів, - віднімання двох поліномів, * множення полінома на число.

Варіант 9. Створити клас UncorrectFraction (у закриті частині класу знаходиться 2 числа цілого типу). Перевантажити оператори () –перетворення дробу у десяткове число, [] –ціла частина числа, % -дробова частина числа, * -множення дробу на ціле число.

Варіант 10. Розробити клас PolynomPow2 (степінь 2). Коефіцієнти представити у вигляді поля-масиву класу. Перевантажити операції + додавання двох поліномів, - віднімання двох поліномів, * множення полінома на число.

Лабораторна робота №7

Тема: Програмування задач із застосуванням одинарного успадкування

Мета: Навчитись реалізовувати одинарне успадкування у класах, використовувати цей принцип об’єктно-орієнтованого програмування у прикладних програмах

Теоретичні відомості: тема 6

Завдання для практичного виконання

Варіант 1. Створити класи транспортних засобів: Auto, Truck, Boat і Plane. Створити з них ієрархії. В основу ієрархії покласти клас Vehicle зі спільними для усіх цих класів елементами. Визначити функції виведення, конструктори і деструктори. Розробити програму, що демонструватиме роботу з класом.

Варіант 2. Використовуючи ієрархію та успадкування, створити класи Window, WindowWithTitle і WindowWithButton. Визначити необхідні конструктори, деструктори та метод зміни назви заголовку вікна. Зімітувати натискання кнопки для закривання вікна. Розробити програму, що демонструватиме роботу з класом.

Варіант 3. Створити клас Liquid, що має назву (вказівник на рядок), густину. Визначити конструктори, деструктор і функцію виведення даних. Створити public-похідний клас Drink, що має колір та ознаку смаку. Визначити конструктори за замовчуванням і з різним числом параметрів, деструктори, функцію виведення. Визначити функції зміни густини, кольору та ознаки смаку. Розробити програму, що демонструватиме роботу з класом.

Варіант 4. Створити клас Person, що має ім’я (вказівник на рядок), вік, вагу. Визначити конструктори, деструктор і функцію виведення. Створити public-

похідний клас Pupil, який має рік навчання. Визначити конструктори за замовчуванням та з різним числом параметрів, деструктори, функцію виведення. Визначити функції перепризначення віку і класу. Розробити програму, що демонструватиме роботу з класом.

Варіант 5. Створити ієрархію класів Person, Student і PostGradStudent. Перевизначити оператори виведення у потік і введення з потоку, визначити конструктор копіювання, операцію присвоювання через відповідні функції базового класу. Розробити програму, що демонструватиме роботу з класом.

Варіант 6. Створити клас Window, що має координати верхнього лівого і нижнього правого кутів, колір фону (вказівник на рядок). Визначити конструктори, деструктор і функцію виведення. Створити public-похідний клас - WindowMenu (меню задається масивом рядків символів). Визначити конструктори за замовчуванням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення кольору фону і рядка меню. Розробити програму, що демонструватиме роботу з класом.

Варіант 7. Створити ієрархію типів, що описує SoftWare та OperationSystem. Визначити клас Lunix як операційну систему. Класи повинні містити необхідні дані, конструктори, деструктори, методи, перевантажені функції виведення у потік і введення з потоку. Розробити програму, що демонструватиме роботу з класом.

Варіант 8. Створити клас Rectangle (довжина, ширина), з методом обчислення його площі. Створити похідний від нього клас Box (довжина, ширина, висота) з методом обчислення об’єму. Всі дані для створення об’єктів задаються у програмі.

Вивести на екран характеристики об’єктів, їх розміри, площу та об’єм. Розробити програму, що демонструватиме роботу з класом.

Варіант 9. Створити клас DataStoradge, що має обсяг (Гбайт). Створити похідні класи HardDrive, OpticalDisc, що додатково мають параметри форматування (кількість циліндрів, доріжок, секторів) та марку. Визначити необхідні конструктори, методи, деструктор. Перевантажити операції потокового введення-виведення. Розробити програму, що демонструватиме роботу з класом.

Варіант 10. Створити клас Dot, що має координати. Визначити конструктори, деструктор і функцію виведення. Створити public-похідний клас ColorDot, що має колір крапки. Визначити конструктори за замовчуванням і з різним числом параметрів, деструктор, функцію виведення. Визначити функції перепризначення кольору і координат крапки, виведення крапки на екран. Розробити програму, що демонструватиме роботу з класом.

Лабораторна робота №8

Тема: Програмування задач із застосуванням множинного успадкування

Мета: Навчитись реалізовувати множжинне успадкування у класах, використовувати цей принцип об’єктно-орієнтованого програмування у прикладних програмах

Теоретичні відомості: тема 7

Завдання для практичного виконання

Варіант 1. Описати класи Вид, Літаюче та ЛітаючаІстота. Клас ЛітаючаІстота є нащадком класів Вид та Літаюче. Клас Вид має поля «назва виду», «клас» та методи, створення та виведення виду. Клас Літаюче має поля «спосіб польоту», «кількість крил», «напрям польоту», та методи створення та виведення. Клас ЛітаючаІстота має власне поле «назва» та методи створення, виведення усіх характеристик літаючої істоти. З використанням цих класів скласти програму опису деякої літаючої істоти та вивести інформацію щодо неї.

Варіант 2. Описати класи Жанр, Носій та Фільм. Клас Фільм є нащадком класів Жанр та Носій.Клас Жанр має поля «назва жанру», «характерні особливості» та методи, створення та виведення жанру.Клас Носій має поля «тип носія», «кольорова гама» та «тривалість зберігання»та методи створення та виведення. Клас Фільм має власні поля «назва», «автор сценарію», «режисер», «тривалість» та методи створення, виведення усіх характеристик фільму. З використанням цих класів скласти програму опису деякого фільму та вивести інформацію щодо нього.

Варіант 3. Описати класи ХудожнійТвір, Техніка та Картина. Клас Картина є нащадком класів ХудожнійТвір та Техніка. Клас ХудожнійТвір має поля «назва твору», «рік написання», «автор», «жанр» та методи, створення, виведення художнього твору. Клас Техніка описує використану техніку живопису. Цей клас має поля «назва техніки» та «матеріал» та методи створення та виведення. Клас Картина має власні поля «ширина», «висота», «вартість» та методи створення та виведення усіх характеристик картини, а також методи читання та зміни вартості. З використанням цих класів скласти програму опису деякої картини та вивести інформацію щодо неї.

Варіант 4. Описати класи Професія, РангДержслужби та ДержавнаПосада. Клас ДержавнаПосада є нащадком класів Професія та РангДержслужби. Клас Професія має поля «назва професії», «спеціальність за освітою» та методи, створення та виведення професії. Клас РангДержслужби має поля «номер рангу», «вимоги для отримання рангу» та методи створення та виведення. Клас ДержавнаПосада має власні поля «назва», «заробітна платня» та методи створення та виведення усіх характеристик державної посади. З використанням цих класів скласти програму опису деякої державної посади та вивести інформацію щодо неї.

Варіант 5. Описати класи МузичнийТвір, Звукозапис та МузичнийЗапис. Клас МузичнийЗапис є нащадком класів МузичнийТвір та Звукозапис. Клас МузичнийТвір має поля «назва твору», «рік створення», «автор», «жанр» та методи, створення та виведення музичного твору. Клас Звукозапис має поля «частота», «стандарт запису» та «тривалість» та методи створення та виведення. Клас МузичнийЗапис має власні поля «носій», «розташування» та методи створення, виведення усіх характеристик музичного запису. З використанням цих класів скласти програму опису деякого музичного запису та вивести інформацію щодо нього.

Варіант 6. Описати класи Текст, Шрифт та ТекстовийНадпис. Клас ТекстовийНадпис є нащадком класів Текст та Шрифт. Клас Текст має поля «зміст тексту», «мова» та методи, створеннята виведення тексту.Клас Шрифт має поля «назва шрифту», «кегль», «напівгрубий», «нахилений», «підкреслений» та методи створення та виведення. Клас ТекстовийНадпис має власне поле «колір» та методи створення тавиведення усіх характеристик текстового надпису. З використанням цих класів скласти програму опису деякого текстового надпису та вивести інформації щодо нього.

Варіант 7. Описати класи Музичний центр, Колонки та Блок живлення. Клас Музичний центр є нащадком класів Колонки та Блок живлення.Клас Колонки має поля «кількість», «максимальну потужність», «тип колонок» та методи, створення та виведення колонки. Клас Блок живлення має поля «потужність», «вихідний вольтаж» та методи створення та виведення.Клас Музичний центр має власні поля «бренд», «кількість USB-портів» та методи створення, виведення усіх характеристик музикального центру. З використанням цих класів скласти програму опису деякого музикального центру та вивести інформацію щодо нього.

Варіант 8. Описати класи Матеріал, ФункціональнеПристосування та Меблі. Клас Меблі є нащадком класів Матеріал та ФункціональнеПристосування. Клас Матеріал має поля «тип матеріалу», «назва матеріалу», «природність» та методи, створення та виведення матеріалу. Клас ФункціональнеПристосування має поля «назва пристосування», «вікові рекомендації» та методи створення та виведення. Клас Меблі має власні поля «тип», «назва» та методи створення, виведення усіх характеристик меблів. З використанням цих класів скласти програму опису деякого функціонального пристосування та вивести інформацію щодо нього.

Варіант 9. Описати класи НастільнаЛампа, Блок живлення та Лампа. Клас НастільнаЛампа є нащадком класів Блок живлення та Лампа. Клас Блок живлення має поля «потужність», «вихідний вольтаж», «вихідний струм» та методи, створення та виведення блоку живлення. Клас Лампа має поля «тип цоколю», «потужність лампи» та методи створення та виведення. Клас НастільнаЛампа має власні поля «бренд», «розміри» та методи створення, виведення усіх характеристик настільної лампи. З використанням цих класів скласти програму опису деякої настільної лампи та вивести інформацію щодо неї.

Варіант 10. Описати класи РозумнийГодинник, Акумулятор та Дисплей. Клас РозумнийГодинник є нащадком класів Акумулятор та Дисплей. Клас Акумулятор має поля «кількість mAh», «вихідний вольтаж», «робоча температура» та методи, створення та виведення акумулятора. Клас Дисплей має поля «розширення дисплею», «тип матриці» та методи створення та виведення. Клас РозумніЧаси має власні поля «бренд», «колір ланцюжка» та методи створення, виведення усіх характеристик розумних часів. З використанням цих класів скласти програму опису деякого розумних часів та вивести інформацію щодо них.

Лабораторна робота №9

Тема: Класи і заголовочні файли

Мета: Навчитись оголошувати класи та їх реалізацію у файлах, відокремлених від головного файлу проєкту. Навчитись під’єднувати заголовочні файли до різних проєктів з метою їх повторного використання

Теоретичні відомості: тема 8

Завдання для практичного виконання

Модифікувати реалізацію лабораторної роботи №6 "Робота з масивами об’єктів" таким чином, аби оголошення класу та реалізація методів були розміщені у заголовочному файлі .h, а реалізація програми - у файлі main.cpp.

Наприклад,

1. Створити клас Student, яка містить наступні поля:

- name – прізвище та ініціали;
- birthYear – рік народження:
- points – оцінки з 4 предметів (масив з 4 елементів)

2. Написати програму, що використовує даний клас і виконує наступні дії:

- вводить з клавіатури масив даних Group, що складається з n змінних типу Student;
- Виводить на екран прізвища і рік народження студентів середній бал яких > 72.0.

Слід створити файл Student.h із оголошенням класу та реалізацією його методів.

У файлі main.cpp реалізовано програму, що демострує роботу із масивом об’єктів. При виконанні програми вводиться із клавіатури масив об’єктів та відбираються дані за вказаною ознакою.

Лабораторна робота №10

Тема: Підсумкове завдання із об’єктно-орієнтованого програмування

Мета: Закріпити навички розробки програм із використанням об’єуктно-орієнтованого підходу

Теоретичні відомості: теми 1-8

Завдання для практичного виконання

Варіант 1

1. Розробити базові класи StudentT (група, прізвище, оцінка) та Teacher (прізвище та ініціали викладача, посада викладача).

2. Визначити конструктори, деструктор та методи встановлення і виведення значень полів даних.

3. Перевантажити операції: () встановлення значень полів даних, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів.

4. Використовуючи множинне успадкування, визначити похідний клас Class з додатковими полями даних: дисципліна, вид заняття (лекція, практичне, лабораторне), дата, час, місце проведення. Визначити конструктори, деструктор, методи встановлення та визначення значень полів даних.

6. Розробити клас Schedule занять, який містить масив об’єктів класу Class. Вивести розклад занять на екран.

Варіант 2

1. Створити клас Person (прізвище, ім’я, по батькові, дата народження, стать та ін.).

2. Визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів даного класу.

3. Перевантажити операцію ( ) для встановлення значень полів даних, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів.

4. Створити похідний клас Emloyee з додатковими полями: табельний номер, оклад, стаж роботи, кількість відпрацьованих годин, зарплата за годину роботи. Визначити необхідні дані, методи, конструктори та деструктори, методи або операторні функції введення-виведення.

5. Розробити клас Payment, у закритій частині якого розміщено список об’єктів з даними про співробітників, для яких розраховується зарплата. Обчислити зарплату співробітників.

Варіант 3

1. Розробити класи Date (день, місяць, рік) та Time (година, хвилина, секунда).

2. Визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів розроблених класів.

3. Перевантажити операцію += для нарощування значення дати і часу на задану величину, -= для зменшення значення дати і часу, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів. Перевіряти коректність значень полів даних.

4. Створити похідний від Date і Time клас File, який містить дані про файл: назва, розмір, дата та час створення, атрибути. Визначити необхідні дані, методи, конструктори та деструктори, методи або операторні функції введення-виведення.

5. Розробити клас Dir, що містить масив об’єктів класу File. Визначити необхідні конструктори, деструктори, методи роботи з файлами та каталогами. Вивести дані про файли, що відповідають заданій масці пошуку.

Варіант 4

1. Розробити клас Organisation, який містить дані про назву організації, телефон, адресу та ін.

2. Визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів цього класу.

3. Перевантажити операції для встановлення значень полів даних, операцію присвоєння об’єктів, потокові операції введення та виведення об’єктів.

4. Створити похідний клас Department з додатковими полями: спеціальність, кількість бакалаврів, спеціалістів і магістрів. Визначити необхідні дані, методи, конструктори та деструктори, методи або операторні функції введення-виведення.

5. Розробити клас Faculty, у закритій частині якого розміщено масив об’єктів з даними про кафедри. Визначити загальну кількість студентів.

Варіант 5

1. Розробити клас Date (рік, місяць, день).

2. Визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів цього класу.

3. Перевантажити операції для встановлення значень полів даних, операцію присвоєння об’єктів, потокові операції введення та виведення об’єктів.

4. Створити похідний клас Book (прізвище автора, назва, видавництво, рік видання, кількість сторінок). Визначити необхідні дані, методи, конструктори та деструктори, методи або операторні функції введення- виведення.

5. Розробити клас Library, що містить масив об’єктів класу Book. Виконати пошук книг за прізвищем автора, назвою, видавництвом, роком видання.

Варіант 6

1. Розробити клас Goods, який містить назву товару, дату виготовлення, свідоцтво якості, вартість.

2. Визначити конструктори, деструктор та методи встановлення і виведення значень полів даних.

3. Перевантажити операції: + ("плюс") та - ("мінус") для зміни вартості товару, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів.

4. Визначити похідний клас GoodsInShop з додатковими полями даних: націнка, термін придатності. Визначити конструктори, деструктор, методи роботи з полями даних.

5. Розробити клас Shop, що містить масив об’єктів класу GoodsInShop. Визначити прибуток магазину від продажу товарів.

Варіант 7

1. Розробити класи Date (рік, місяць, день) і Time(година, хвилина, секунда).

2. Визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів цього класу.

3. Перевантажити операції для встановлення значень полів даних, операцію присвоєння об’єктів, потокові операції введення та виведення об’єктів.

4. На основі множинного успадкування класів створити похідний клас TVShow (дата, час трансляції, вид та назва передачі). Визначити необхідні дані, методи, конструктори та деструктори, операторні функції введення- виведення.

5. Розробити клас Schedule, що містить масив об’єктів класу TVShow. Виконати пошук передачі за її назвою.

Варіант 8

1. Розробити клас Goods, який містить назву товару, кількість одиниць товару, інструкцію застосування, вартість.

2. Визначити конструктори, деструктор та методи встановлення і читання значень полів даних.

3. Перевантажити операції += та -= для зміни кількості одиниць товару, + та - для зміни вартості товару, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів.

4. Визначити похідний клас Medicine з додатковими даними: концентрація, термін придатності. Визначити операторні методи введення, виведення, корегування полів даних класу.

5. Розробити клас Pharmacy, який містить масив об’єктів класу Medicine. Визначити вартість ліків за рецептом.

Варіант 9

1. Розробити клас Produce, який містить назву продукту та ціну одиниці продукту.

2. Визначити конструктори, деструктор та методи встановлення і читання значень полів даних.

3. Перевантажити операції + та - для зміни вартості продукту, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів.

4. Визначити похідний клас Ingredient з додатковими даними про дозування продукту. Визначити операторні методи введення, виведення, корегування полів даних класу.

5. Розробити клас Recipe кулінарного виробу, який містить масив об’єктів класу Ingredient. Визначити вартість виготовлення кулінарного виробу.

Варіант 10

1. Створити клас Person (прізвище, ім’я, по батькові, дата народження).

2. Визначити конструктори ініціалізації, копіювання, деструктори та методи для зміни і читання значень полів даного класу.

3. Перевантажити операцію () для встановлення значень полів даних, операцію присвоєння об’єктів =, потокові операції введення >> та виведення << об’єктів.

4. Створити похідний клас Subscriber з додатковим полем - номер телефону. Визначити необхідні дані, методи, конструктори та деструктори, методи або операторні функції введення- виведення.

5. Розробити клас PhoneBook, у закритій частині якого розміщено масив об’єктів з даними про абонентів. Виконати пошук номера телефону за прізвищем абонента.

Список використаних джерел

1. Бєлов Ю. А., Карнаух Т. О., Коваль Ю. В., Ставровський А. Б.. Вступ до програмування мовою С++. Організація обчислень: навч. посіб. Київ : ВПЦ “Київський університет”, 2012. 176 с.
2. Богач І. В., Довгалець С. М., Дубовой В. М. Алгоритми розв’язання задач з програмування. Розв’язник. Вінниця: ВНТУ, 2017. 119 с.
3. Бойко Б. І., Омельчук Л. Л., Русіна Н. Г. Об’єктно-орієнтоване програмування. Лабораторний практикум: навч. посіб. Київ : КНУ ім. Тараса Шевченка, 2016. 90 с.
4. Бублик В. В. Об’єктно-орієнтоване програмування. Київ : ІТ-книга, 2015. 640 с.
5. Вакалюк Т. А., Шевчук Л. Д., Постова С. А. Структурне та візуальне програмування Навчальний посібник для студентів фізико-математичного факультету. Переяслав-Хмельницький : вид-во ПХДПУ, 2019. 318 с.
6. Грицюк Ю., Рак Т. Програмування мовою С++. Львів: Вид-во ЛДУ БЖД, 2011. 290 с.
7. Грицюк Ю., Рак Т. Об’єктно-орієнтоване програмування мовою С++. Львів: Вид-во ЛДУ БЖД, 2011. 403 с.
8. Жуковський С. С., Вакалюк Т. А. Об’єктно-орієнтоване програмування мовою С++. Навчально-методичний посібник для студентів напряму 6.040302 “Інформатика”. Житомир: Вид-во ЖДУ, 2016. 100 с.
9. Зубенко В. В., Омельчук Л. Л. Програмування. Поглиблений курс. Ктїв : ВПЦ “Київський університет”, 2011. 624 с.
10. Каплун В. А., Баришев Ю. В., Остапенко А. В. Технологія програмування. Лабораторний практикум: навч. посіб. Вінниця : ВНТУ, 2015. 125 с.
11. Коротеєєва Т. О. Алгоритми та структури даних: навч. посіб. Львів : Видавництво Львівської політехніки, 2014. 280 с.

12. Кравець О. П. Об’єктно-орієнтоване програмування: навч. посіб. Львів: Видавництво Львівської політехніки, 2012. 626 с.
13. Креневич А. П., Обвінцев О. В. С++ у задачах і прикладах : навчальний посібник із дисципліни "Інфор-матика та програмування". К : Видав-ничо-поліграфічний центр "Київський університет", 2011. 208 с.
14. Нікітченко М. С. Теорія програмування. Частина 1. Навчальний посібник Ніжин: Видавництво НДУ імені Миколи Гоголя, 2010. 121 с.
15. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 1: навчальний посібник. Львів : Видавництво «Новий Світ-2000», 2020. 320 с.
16. Ришковець Ю. В., Висоцька В. А. Алгоритмізація та програмування. Частина 2: навчальний посібник. Львів : Видавництво «Новий Світ-2000», 2020. 337 с.
17. С++. Основи програмування. Теорія та практика: підручник / О. Г. Трофименко та ін. Одеса: Фенікс, 2010. 544 с.
18. С++. Теорія та практика: навч. посіб.з грифом МОНУ/ О. Г. Трофименко та ін. Одеса : ВЦ ОНАЗ, 2011. 587 с.
19. Татарчук Д. Д., Діденко Ю. В. Програмування мовами С та С++: навч. посіб. Ктїв : НТУУ" КПІ", 2012. 112 с.
20. Трофименко О. Г., Буката Л. М., Леонов Ю. Г. Алгоритмізація обчислювальних процесів і особливості програмування в С++: метод. посіб. Одеса : ІЦОНАЗ, 2009. 93 с.
21. Трофименко О. Г., Прокоп Ю. В., Логінова Н. І., Задерейко О. В. С++. Алгоритмізація та програмування: підручник. Одеса : Фенікс, 2019. 477 с.
22. .Шевчук І.Б. Конспект лекцій з навчальної дисципліни “Алгоритмізація та програмування”. Львів : Львівський національний університет ім. Івана Франка, 2018. 30с.

УДК 004.4(075.8)

Електронне мережне навчальне видання

Т. О. Гришанович, Л. Я. Глинчук

ОСНОВИ ОБ’ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ

навчальний посібник

Друкується в авторській редакції