ЛАБОРАТОРНА РОБОТА №1
КЛАСИ І ОБ'ЄКТИ В С++

Мета роботи – Отримати практичні навички реалізації класів на С++.

Основні теоретичні відомості

Об'єктно-орієнтоване програмування побудовано на понятті класу.

Клас визначає новий тип даних, який задає формат об'єкта. Клас включає як дані, так і код, призначений для виконання дій над цими даними. Отже, клас пов'язує дані з кодом. У С++ специфікація класу використовується для побудови об'єктів. Об'єкти – це екземпляри класу. Клас – це логічна абстракція, яка реально не існує до тих пір, поки не буде створено об'єкт цього класу, тобто то, що стане фізичним представленням цього класу в пам'яті комп'ютера.

При визначенні класу оголошуються дані, які він містить, і код, який виконується над цими даними. Хоча дуже прості класи можуть містити тільки код або тільки дані, більшість реальних класів містять обидва компоненти. У класі дані оголошуються у вигляді змінних, а код оформляється у вигляді функцій. Функції та змінні, складові класу, називаються його членами або функціями та змінними екземпляра.

Оголошення класу починається з ключового слова class.

Оголошення класу синтаксично подібно оголошенню структури. Розглянемо приклад.

Наступний клас визначає тип queue, який призначений для реалізації черги. (Під чергою розуміється список обслуговування в порядку надходження, тобто «першим прибув – першим обслужений».)

```
// Створення класу queue.
  class queue
  {
  int q[100];
  int sloc, rloc;
  public:
  void init();
  void qput(int i);
  int qget();
  };
```

Розглянемо детально оголошення цього класу.

Всі члени класу queue оголошені в тілі інструкції class. Членами даних класу queue є змінні q, sloc і rloc. Крім того, тут визначено три функції-члена: init(), qput() і qget().

Будь-який клас може містити як закриті, так і відкриті члени. За замовчуванням всі елементи, які визначені в класі, є закритими (private-члени). Наприклад, змінні q, sloc і rloc є закритими. Це означає, що до них можуть отримати доступ тільки інші члени класу queue; ніякі інші частини програми цього зробити не можуть. У цьому полягає один із проявів інкапсуляції: програміст в повній мірі може керувати доступом до певних елементів даних. Закритими можна оголосити і функції (в цьому прикладі таких немає), і тоді їх зможуть викликати тільки інші члени цього класу.

Ключове слово **public** використовується для оголошення відкритих членів класу.

Щоб зробити частини класу відкритими (тобто доступними для інших частин програми), необхідно оголосити їх після ключового слова public. Всі змінні або функції, визначені після модифікатора public, доступні для всіх інших функцій програми. Отже, в класі queue функції init (), qput () і qget () є відкритими. Зазвичай в програмі організовується доступ до закритих членів класу через його відкриті функції. Зверніть увагу на те, що після ключового слова public слід двокрапка.

Слід мати на увазі, що об'єкт утворює свого роду зв'язок між кодом і даними. Так, будь-яка функція-член має доступ до закритих елементів класу. Це означає, що функції init (), qput () і qget () мають доступ до змінних q, sloc і rloc. Щоб додати функцію-член в клас, треба визначити її прототип в оголошенні цього класу.

В оголошенні класу queue містяться прототипи функцій-членів. Оскільки функції-члени забезпечені своїми прототипами у визначенні класу, їх не потрібно поміщати більше ні в яке інше місце програми.

Щоб реалізувати функцію, яка є членом класу, необхідно повідомити компілятору, до якого класу вона належить, назвавши ім'я цієї функції разом з ім'ям класу. Наприклад, ось як можна записати код функції qput():

```
void queue::qput(int i)
{
if(sloc==100)
   {
    cout << "Черга заповнена.";
    return;
   }
sloc++;
q[sloc] = i;
}
```

Оператор дозволу області видимості кваліфікує ім'я члена разом з ім'ям його класу.

Оператор "::" називається оператором дозволу області видимості. По суті, він повідомляє компілятору, що дана версія функції qput() належить класу queue. Іншими словами, оператор "::" заявляє про те, що функція qput() знаходиться в області видимості класу queue. Різні класи можуть використовувати однакові імена функцій. Компілятор ж визначить, до якого класу належить функція, за допомогою оператора дозволу області видимості і імені класу.

Функції-члени можна викликати тільки з заданого об'єкта. Щоб викликати функцію-член з частини програми, яка знаходиться поза класом, необхідно використовувати ім'я об'єкта і оператор "крапка". Наприклад, при виконанні цього коду буде викликана функція init () для об'єкта ob1.

```
queue ob1, ob2;
ob1.init();
```

Конструктор – це функція, яка викликається при створенні об'єкта.

Як правило, деяку частину об'єкта, перш ніж його можна буде використовувати, необхідно ініціалізувати. Наприклад, розглянемо клас queue (він представлений вище в цьому розділі). Перш ніж клас queue можна буде використовувати, змінним rloc і sloc потрібно присвоїти нульові значення. В даному конкретному випадку ця вимога виконувалося за допомогою функції init(). Але, оскільки вимога ініціалізації членів класу вельми поширена, в C++ передбачена реалізація цієї можливості при створенні об'єктів класу. Така автоматична ініціалізація виконується завдяки використанню конструктора.

Конструктор – це спеціальна функція, яка є членом класу і ім'я якої збігається з ім'ям класу. Ось, наприклад, як став виглядати клас queue після переробки, пов'язаної із застосуванням конструктора для ініціалізації його членів.

```
// Визначення класу queue.
class queue {
int q [100];
int sloc, rloc;
public:
queue (); // конструктор
void qput (int i);
int qget ();
};
```

Зверніть увагу на те, що в оголошенні конструктора queue() відсутній тип значення, що повертається. У C++ конструктори не повертають значень і, отже, немає сенсу у вказуванні їх типу. (При цьому не можна вказувати навіть тип void.)

Тепер наведемо код функції queue ().

```
// Визначення конструктора.
queue :: queue ()
{
sloc = rloc = 0;
cout << "Черга ініціалізована.";
}
```

Конструктор об'єкту викликається при створенні об'єкта. Це означає, що він викликається при виконанні інструкції оголошення об'єкта. Конструктори

глобальних об'єктів викликаються на самому початку виконання програми, ще до звернення до функції main(). Що стосується локальних об'єктів, то їх конструктори викликаються кожен раз, коли зустрічається оголошення такого об'єкта.

Деструктор – це функція, яка викликається при руйнуванні об'єкта.

У багатьох випадках при руйнуванні об'єкта необхідно виконати деяку дію або навіть деяку послідовність дій. Локальні об'єкти створюються при вході в блок, в якому вони визначені, і руйнуються при виході з нього. Глобальні об'єкти руйнуються при завершенні програми. Існує безліч факторів, що обумовлюють необхідність деструкції. Наприклад, об'єкт повинен звільнити раніше виділену для нього пам'ять. У C++ саме деструкторам доручається обробка процесу дезактівізация об'єкта. Ім'я деструктора збігається з ім'ям конструктора, але передує символом "~" Подібно конструкторам деструктори не повертають значень, а отже, в їх оголошеннях відсутня тип значення.

Розглянемо вже знайомий нам клас queue, але тепер він містить конструктор і деструктор. Розглянемо вже знайомий нам клас queue, але тепер він містить конструктор і деструктор.

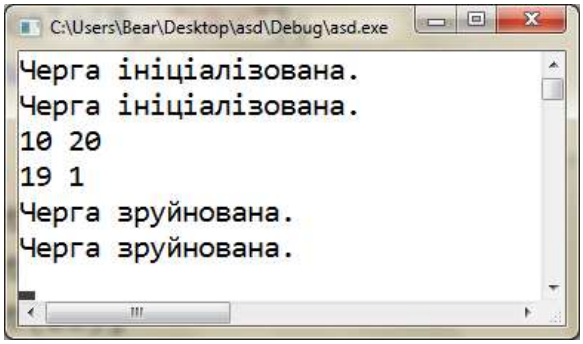
```
// Визначення класу queue.
class queue {
int q [100];
int sloc, rloc;
public:
queue(); // конструктор
~queue (); // деструктор
void qput (int i);
int qget();
};
// Визначення конструктора.
queue::queue()
{
sloc = rloc = 0;
cout << "Черга ініціалізована.";
}
// Визначення деструктора.
queue::~queue ()
{
cout << "Черга зруйнована.";
}
```

Ось як виглядає нова версія програми реалізації черги, в якій демонструється використання конструктора і деструктора.

```
#include <conio.h>
#include <iostream>
// Демонстрація доступу до членів класу.
using namespace std;

// Визначення класу queue.
class queue {
int q [100];
int sloc, rloc;
public:
queue(); // конструктор
~queue(); // деструктор
void qput(int i);
int qget();
};
// Визначення конструктора.
queue::queue ()
{
sloc = rloc = 0;
cout << "Черга ініціалізована."<<endl;
}
// Визначення деструктора.
queue::~queue ()
{
cout << "Черга зруйнована."<<endl;
}
// Занесення в чергу цілочисельного значення.
void queue::qput(int i)
{
  if(sloc==100)
  {
  cout << "Черга заповнена."<<endl;
  return;
  }
 sloc++;
 q[sloc] = i;
}

// Вилучення з черги цілочисельного значення.
int queue::qget()
{
if(rloc == sloc) {
  cout << "Черга пуста."<<endl;
  return 0;
  }
 rloc++;
```

```
 return q[rloc];
}

int main()
{
if(true)
{
queue a, b; // Створення двох об'єктів класу queue.
a.qput(10);
b.qput(19);
a.qput(20);
b.qput(1);
cout << a.qget() << " ";
cout << a.qget() << " "<<endl;
cout << b.qget() << " ";
cout << b.qget() << " "<<endl;
// руйнування об'єктів після виходу з блоку if
}
}
```

При виконанні цієї програми виходять такі результати:



Параметризрвані конструктори

Конструктор може мати параметри. З їх допомогою при створенні об'єкта членам даних (змінним класу) можна привласнити деякі початкові значення, які визначаються в програмі. Це реалізується шляхом передачі аргументів конструктору об'єкта. У наступному прикладі ми вдосконалюємо клас queue так, щоб він приймав аргументи, які будуть служити ідентифікаційними номерами (ID) черги. Перш за все необхідно внести зміни в визначення класу queue. Тепер воно виглядає так:

```
// Визначення класу queue.
class queue {
int q [100];
int sloc, rloc;
int who; // містить ідентифікаційний номер черги
public:
```

```
queue(int id); // параметризований конструктор
~queue (); // деструктор
void qput (int i);
int qget ();
};
```

Змінна who використовується для зберігання ідентифікаційного номера (ID) черзі. Її реальне значення визначається значенням, переданим конструктору як параметр id, при створенні змінної типу queue. Конструктор queue () виглядає тепер таким чином:

```
// Визначення конструктора.
queue :: queue (int id)
{
sloc = rloc = 0;
who = id;
cout << "Черга " << who << " ініціалізована."<<endl;
}
```

Щоб передати аргумент конструктору, необхідно зв’язати цей аргумент з об'єктом при оголошенні об'єкту. C++ підтримує два способи реалізації такого зв'язування. Ось як виглядає перший спосіб:

```
queue a = queue(101);
```

В цьому оголошенні створюється черга на ім'я a, якій передається значення (ідентифікаційний номер) 101. Але ця форма (в такому контексті) використовується рідко, оскільки другий спосіб має більш короткий запис і зручніше для використання.

У другому способі аргумент повинен слідувати за ім'ям об'єкта і полягати в круглі дужки. Наприклад, така інструкція еквівалентна попередньому оголошенню:

```
queue a(101);
```

Це найпоширеніший спосіб оголошення параметризованих об'єктів.

Вказівники на об'єкти

Щоб отримати доступ до окремого члена об'єкта виключно «силами» самого об'єкта, використовується оператор «крапка». А якщо для цього служить вказівник на цей об'єкт, необхідно використовувати оператор «стрілка». (Застосування операторів «крапка» і «стрілка» для об'єктів відповідає їх застосування для структур та об'єднань.)

Приклад визначення класу

```
class Student {
 string name; // Ім'я
 int age; // Вік
 double grade; // Рейтинг
public:
 Student(); // Конструктор без параметрів
 Student(string, int, double); // Конструктор з параметрами
 ~Student();
 void Set(string, int, double);
 void Show();
};
```

Приклад реалізації конструктора з видачею повідомлення.

```
Student::Student(string NAME, int AGE, double GRADE)
{
 name=NAME;
 age = AGE;
 grade = GRADE;
 cout << "Конструктор з параметрами викликаний для об'єкта" <<
this << endl;
}
```

Приклад реалізації функції Show (вивід інформації про об’єкт)

```
void Student::Show()
{
 cout << "Студент " << name << ", вік " << age << ", має
рейтинг " << grade << endl;
}
```

У програмі необхідно передбачити розміщення об'єктів як в статичній, так і в динамічній пам'яті, а також створення масивів об'єктів.

Приклади.

а) масив студентів розміщується в статичної пам'яті:

```
Student group[3] = {  Student("Іванопуло", 19, 50.9),
                      Student("Петрополіс", 18, 25.5),
                      Student("Сидоракіс", 18, 45.5)
                   };
```

б) масив студентів розміщується в динамічної пам'яті:

```
Student *P = new Student[3];
P->Set("Іванопуло", 19, 50.9);
(P + 1)->Set("Петрополіс", 18, 25.5);
(P + 2)->Set("Сидоракіс", 18, 45.5);
```

Приклад використання покажчика на компонентну функцію:

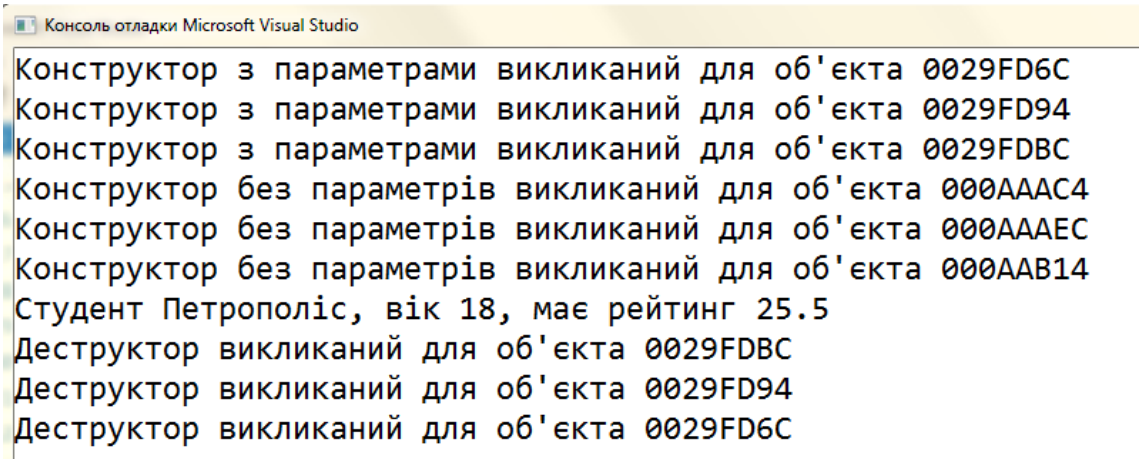
```
void (Student:: * pf)();
pf = &Student::Show;
(P[1].*pf)();
```

Текст всієї програми:

```
class Student {
     string name; // Ім'я
     int age; // Вік
     double grade; // Рейтинг
public:
     Student(); // Конструктор без параметрів
     Student(string, int, double); // Конструктор з параметрами
     ~Student();
     void Set(string, int, double);
     void Show();
};

Student::Student()
{
     name = "Ковіді";
     age = 18;
     grade = 10;
     cout << "Конструктор без параметрів викликаний для об'єкта "
<< this << endl;
}
Student::Student(string NAME, int AGE, double GRADE)
{
     name=NAME;
     age = AGE;
     grade = GRADE;
```

```
	cout << "Конструктор з параметрами викликаний для об'єкта " <<
this << endl;
}
Student::~Student()
{

	cout << "Деструктор викликаний для об'єкта " << this << endl;
}
void Student::Set(string NAME, int AGE, double GRADE)
{
	name = NAME;
	age = AGE;
	grade = GRADE;

}

void Student::Show()
{
	cout << "Студент " << name << ", вік " << age << ", має
рейтинг " << grade << endl;
}

int main()
{
	setlocale(0, "");
	Student group[3] = { Student("Іванопуло", 19, 50.9),
	Student("Петрополіс", 18, 25.5),
	Student("Сидоракіс", 18, 45.5)
	};


	Student *P = new Student[3];
	P->Set("Іванопуло", 19, 50.9);
	(P + 1)->Set("Петрополіс", 18, 25.5);
	(P + 2)->Set("Сидоракіс", 18, 45.5);

	void (Student:: * pf)();
	pf = &Student::Show;
	(P[1].*pf)();
}
```

Результат роботи програми наведений нижче:



Зверніть увагу, що після завершення головної функції знищуються тільки об’єкти, які були розміщені в статичної пам'яті, як знищуються об’єкти в динамічної пам'яті, розглянемо в лабораторної роботі 4.

Ключове слово this

Ключове слово *this* – це вказівник на поточний об'єкт даного класу. Відповідно через *this* ми можемо звертатися всередині класу до будь-яких його членів.

Для звернення до змінних використовується вказівник *this*. Причому після this ставиться не крапка, а стрілка `->`.

Завдання до лабораторної роботи

1. Визначити користувальницький клас у відповідності з варіантом завдання.

2. Визначити в класі конструктор без параметрів і конструктор з параметрами.

3. Визначити в класі деструктор.

4. Визначити в класі компоненти-функції для перегляду та ініціалізації полів даних.

5. Визначити покажчик на компоненту-функцію.

6. Визначити покажчик на екземпляр класу.

7. Написати демонстраційну програму, в якій створюються і руйнуються об'єкти певного класу і кожен виклик конструктора і деструктора супроводжується видачею відповідного повідомлення. (який об'єкт який конструктор або деструктор викликав).

8. Показати в програмі використання покажчика на об'єкт і покажчика на компоненту функцію.

Варіанти завдання

Варіант 1.

Створити клас «квадрат», такий, що:

a) його екземпляр містить розмір сторони квадрата.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням сторони.

c) його методи дозволяють отримувати і присвоювати значення сторони і обчислювати значення площі.

d) функція print_sqr () виводить на екран значення його сторони і площі.

Варіант 2.

Створити клас «трикутник», такий, що:

a) його екземпляр містить розміри трьох сторін трикутника.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням сторін.

c) його методи дозволяють отримувати і присвоювати значення сторін і обчислювати значення площі (наприклад, за формулою Герона), $s = \sqrt{p(p-a)(p-b)(p-c)}, p = (a+b+c)/2$.

d) функція print_sqr () виводить на екран значення його сторін і площі.

Варіант 3

Створити клас «циліндр», такий, що:

a) його екземпляр містить розмір радіуса і висоти.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням радіуса і висоти.

c) його методи дозволяють отримувати і присвоювати значення радіуса і висоти та обчислювати значення об'єму.

d) функція print_val () виводить на екран значення його радіуса, висоти та об’єму.

Варіант 4

Створити клас «куля», такий, що:

a) його екземпляр містить розмір радіусу.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням радіусу.

c) його методи дозволяють отримувати і присвоювати значення радіусу та обчислювати значення об'єму.

d) функція print_val () виводить на екран значення його радіуса та об’єму.

Варіант 5.

Створити клас «конус», такий, що:

a) його екземпляр містить розмір радіуса і висоти.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням радіуса і висоти.

c) його методи дозволяють отримувати і присвоювати значення радіуса і висоти та обчислювати значення об'єму.

d) функція print_val () виводить на екран значення його радіуса, висоти та об’єму.

Варіант 6.

Створити клас «трапеція», такий, що:

a) його екземпляр містить розмір двох основ і висоти.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням підстав і висоти.

c) його методи дозволяють отримувати і присвоювати значення основ і висоти і обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення її основ, висоти і площі.

Варіант 7.

Створити клас «смайлик», такий, що: 

a) його екземпляр містить розмір радіуса особи R і радіуса r очей.

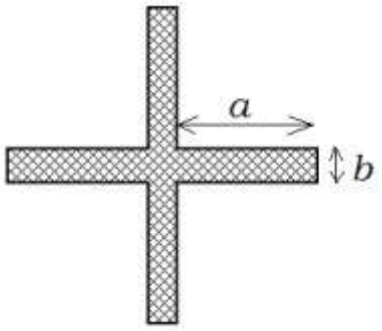
b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням R і r.

c) його методи дозволяють отримувати і присвоювати значення R і r і обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення її R, r площі (без площі очей).

Варіант 8.

Створити клас «хрест», такий, що:



a) його екземпляр містить розмір два розміри a і b.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням a і b.

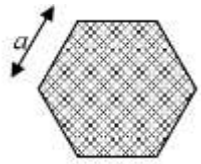
c) його методи дозволяють отримувати і присвоювати значення a, b та обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення a, b і площі цієї фігури.

Варіант 9.

Створити клас «шайба» (правильний шестикутник), такий, що:

$$S = \frac{3\sqrt{3}}{2}a^2$$



a) його екземпляр містить розмір сторони a.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням a.

c) його методи дозволяють отримувати і присвоювати значення a і обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення сторони і площі цієї фігури.

Варіант 10. Створити клас «ромб»:

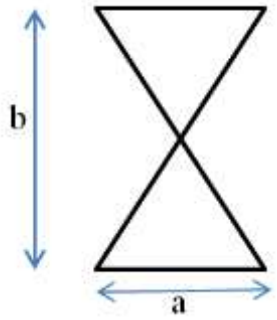
a) його екземпляр містить розміри його діагоналей a і b.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням a і b.

c) його методи дозволяють отримувати і присвоювати значення a, b і обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення a, b і площі цієї фігури.

Варіант 11. Створити клас «пісочний годинник»:



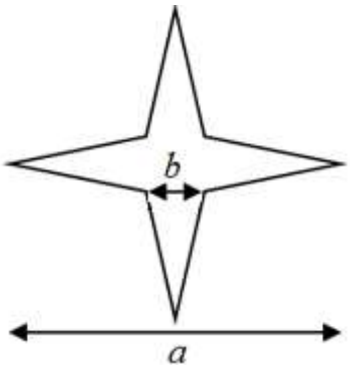
a) його екземпляр містить розміри a і b.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням a і b.

c) його методи дозволяють отримувати і присвоювати значення a, b і обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення a, b і площі цієї фігури.

Варіант 12. Створити клас «зірка»:



a) його екземпляр містить розміри a і b.

b) його конструктор без параметра створює екземпляр зі значенням 0, а конструктор з параметрами створює екземпляр з відповідним значенням a і b.

c) його методи дозволяють отримувати і присвоювати значення a, b і обчислювати значення площі.

d) функція print_sqrt () виводить на екран значення a, b і площі цієї фігури цієї фігури.

Варіант 13. Створити клас «комплексне число», такий, що:

a) його екземпляр містить дійсну і уявну частину - змінні з плаваючою точкою.
b) його конструктор без параметра створює екземпляр зі значенням 0.0 + i0.0, а конструктор з параметрами створює екземпляр з відповідною дійсною та уявною частиною.
c) його методи дозволяють отримувати і присвоювати значення дійсної та уявної частини.
d) функція print () виводить на екран значення екземпляра у вигляді (0.00 ± i0.00).

Варіант 14. Створити клас «вектор», такий, що:

a) його екземпляр містить три координати - змінні з плаваючою точкою. Особливість значень координат: знак третьої координати автоматично стає таким, що їх множення завжди додатне.
b) його конструктор без параметра створює екземпляр зі значенням 0.0 0.0 0.0, а конструктор з параметрами створює екземпляр з відповідними значеннями координат.
c) його методи дозволяють отримувати і присвоювати значення координат.
d) функція print () виводить на екран значення екземпляра в <0.0 0.0 0.0>.

Варіант 15. Створити клас «людина», такий, що:

a) він містить рядкові поля «ім'я», «прізвище» «телефон». Поле цілого типу «ідентифікаційний код», «вік»
b) його конструктор без параметра створює екземпляр з ващими значеннями, а конструктор з параметрами створює екземпляр з відповідними значеннями параметрів.
c) його методи дозволяють отримувати і присвоювати значення.
d) функція print () виводить на екран всю інформацію про людину.

Зміст звіту:

1. Титульний лист: назва дисципліни; номер та найменування роботи; прізвище, ім'я, по батькові студента; дата виконання.

2. Постановка завдання. Слід дати конкретну постановку, тобто вказати, який клас повинен бути реалізований, які повинні бути в ньому конструктор, компоненти-функції тощо.

3. Визначення користувацького класу з коментарями.

4. Реалізація конструкторів і деструктора.

5. Фрагмент програми, що показує використання покажчика на об'єкт і покажчика на функцію з поясненням.

6. Лістинг основної програми, в якому має бути зазначено, в якому місці та який конструктор або деструктор викликаються.

Контрольні запитання

1. Навести приклад опису класу, визначення об’єкта.
2. Чи обов’язково в описі класу повинні міститись конструктор і деструктор?
3. Скільки може бути в класі конструкторів і деструкторів?
4. Коли викликається конструктор і деструктор?
5. Що таке конструктор за замовчуванням?
6. Що таке «посилання на себе»?