МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

КЛАСИ ЯК ОСНОВА ООП

**Методичні рекомендації та варіанти завдань
до виконання лабораторних робіт
з дисципліни «Програмування та алгоритмічні мови»**

бакалаврами напряму підготовки 12 Інформаційні технології

Дніпро
2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ІНСТИТУТ ЕЛЕКТРОЕНЕРГЕТИКИ
Факультет інформаційних технологій
Кафедра безпеки інформації та телекомунікацій

КЛАСИ ЯК ОСНОВА ООП

Методичні рекомендації та варіанти завдань до виконання лабораторних робіт з дисципліни «Програмування та алгоритмічні мови»

бакалаврами напряму підготовки 12 Інформаційні технології

Дніпро
НТУ «ДП»
2021

Класи як основа ООП. Методичні рекомендації та варіанти завдань до виконання лабораторних робіт з дисципліни «Програмування та алгоритмічні мови» бакалаврами напряму підготовки 12 Інформаційні технології / Упоряд.: Г.М. Саксонов, О.А. Жукова ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро: НТУ «ДП», 2021. – 20 с.

Упорядники:
Г.М. Саксонов, ст. викл;
О.А. Жукова, ст. викл.

Затверджено методичною комісією за напрямом «Кібербезпека» (протокол № 4 від 10.03.2021) за поданням кафедри безпеки інформації та телекомунікацій (протокол № 10 від 09.03.2021).

Подано методичні рекомендації до виконання лабораторних робіт з дисципліни «Програмування і алгоритмічні мови» для бакалаврів напряму 12 Кібербезпека.

Відповідальний за випуск зав. кафедри БІТ В.І. Корнієнко, д-р техн. наук, проф.

ЗМІСТ

ПЕРЕДМОВА

Класи та об'єкти в С ++ є основними концепціями об'єктно-орієнтованого програмування (ООП). Об'єктно-орієнтоване програмування – розширення структурного програмування, в якому основними концепціями є поняття класів і об'єктів. Основна відмінність мови програмування С++ від С полягає в тому, що в С немає класів, а отже мова С не підтримує ООП, на відміну від С++.

Класи в С++ – це абстракція, що описує методи і властивості ще не існуючих об'єктів. ***Об'єкти*** – конкретне уявлення абстракції, що має свої властивості та методи. Створені об'єкти на основі одного класу називаються ***екземплярами*** цього класу. Ці об'єкти можуть мати різну поведінку, властивості, але все одно будуть об'єктами одного класу. В ООП існує три основних принципи побудови класів:

1. ***Інкапсуляція*** – це властивість, що дозволяє об'єднати в класі і дані, і методи, які працюють з ними і приховати деталі реалізації від користувача.

2. ***Успадкування*** – це властивість, що дозволяє створити новий клас-нащадок на основі вже існуючого, при цьому всі характеристики класу батька присвоюються класу-нащадку.

3. ***Поліморфізм*** – властивість класів, що дозволяє використовувати об'єкти класів з однаковим інтерфейсом без інформації про тип і внутрішній структурі об'єкта.

1. ЛАБОРАТОРНА РОБОТА № 1
ЗНАЙОМСТВО З КЛАСАМИ

1.1. Мета роботи

Ознайомитись на практиці з класами, об'єктами та головними елементами об'єктного підходу.

Навчитись створювати і використовувати об’єкти типу клас.

1.2. Постановка задачи

Використовуючи лекційний матеріал і інші літературні джерела, навчитись описувати класи, засвоїти способи створення об’єктів.

Розробити програму, що реалізовує два об’єкта класу *steck*. За основу узяти приклад, наведений нижче.

Введення і виведення символів організувати з клавіатури.

Доповнити клас *steck* членами-функціями що реалізовують:

- вивід на екран количества символів в стеку;
- Вивід на екран символу, що знаходиться в стеку із заданим номером.

1.3. Теоретичні відомості

Клас використовується для створення об'єктів. Основна форма має вигляд:

```
class <имя класса>
        {
        <закрытые функции и переменные>
        public:
        <функции-члены и переменные>
        }
```

Закриті функції і ***змінні - члени*** доступні тільки для інших членів цього класу.

Відкриті функції і ***змінні*** доступні для будь-якої частини програми, в якій знаходиться клас.

Функції, оголошені усередині описи класу, називаються ***функціями членами***.

Для визначення функцій-членів використовується форма:

<тип> <имя класса> :: <**имя функции-члена>** (параметры)
{ тело функции }

Дві двокрапки після імені класу (**::**) називаються ***операцією розширення області видимості.***

Визначення класу тільки визначає тип об'єктів, а самі об'єкти не задає, пам'ять не виділяється. Для створення об'єктів ім'я класу використовується як специфікатор типу даних.

Після створення об'єкту до відкритих членів класу можна ***звертатися***, використовуючи ***операцію крапка*** – *d.pop()*.

set-функції та get-функції класів

Кожен об'єкт має якісь свої властивості або атрибути, які характеризують його впродовж усього життя. Атрибути об'єкта зберігаються в змінних, оголошених всередині класу, якому належить даний об'єкт. Причому, оголошення змінних повинно виконуватися зі специфікатором доступу *private*. Такі змінні називаються елементами даних або полями класу. Оскільки елементи даних оголошені в *private*, то і доступ до них можуть отримати тільки методи класу, зовнішній доступ до елементів даних заборонений. Тому прийнято оголошувати в класах спеціальні методи – так звані *set* і *get* функції, за допомогою яких можна маніпулювати елементами даних. *set*-функції ініціалізують елементи даних, *get*-функції дозволяють переглянути значення елементів даних.

Доопрацюємо клас *CppClass* так, щоб у ньому можна було зберігати дату в форматі *дд.мм.гг*. Для зміни і перегляду дати реалізуємо *set*- і *get*-функції.

```
class CppClass //ім'я класу
{
  private:
  int day, // день
  month, // місяць
  year; // рік
  public:
  void message () //функція виводить повідомлення на екран
       {
```

```
        cout << "\n\tClasses and Objects in C++ \n\n";
      }
  void setDate (int date_day, int date_month, int date_year)
      {
        day = date_day; / Ініціалізація дня
        month = date_month; //Ініціалізація місяця
        year = date_year; //Ініціалізація року

  void getDate () //відобразити поточну дату
  {
  cout<<"\n\tDate:"<<day<<"."<<month<<"."<<year<<endl<<endl;
  }
}; // Кінець оголошення класу CppStudio
int main ()
{
  setlocale (LC_ALL, "rus"); // Установка локалі
  int day, month, year;
  cout << "\n\tВведіть поточний день, місяць і рік: \n\n";
  cout << "\tдень : "; cin >> day;
  cout << "\tмісяць: "; cin >> month;
  cout << "\tрік : "; cin >> year;
  CppClass obj; // Оголошення об'єкта
  obj.message (); // Виклик функції класу message
  obj.setDate (day, month, year); // Ініціалізація дати
  obj.getDate (); // Відобразити дату
  system ("pause");
  return 0;
}
```

У визначенні класу специфікатор доступу ***private*** обмежує доступ до змінних, які оголошені після нього і до початку специфікатора доступу ***public***. Таким чином, до змінних *day*, *month*, *year*, можуть отримати доступ тільки методи класу. Функції, що не належать класу, не можуть звертатися до цих змінних. Дані або методи класу, оголошені після специфікатора доступу private, але до початку наступного специфікатора доступу називаються закритими елементами даних і закритими методами класу.

Доцільно оголошувати елементи даних після специфікатора доступу ***private***, а методи класу – після специфікатора ***public***. Тоді, для маніпулювання елементами даних, оголошуються спеціальні функції – *get* і *set*.

В клас *CppClass* ми додали два методи *setDate()* і *getDate()*. Метод *setDate()* (set-функція) ініціалізує елементи даних. Тобто метод *setDate()* ініціалізує змінні *day*, *month*, *year*.

Щоб переглянути, значення закритих елементів даних, оголошена функція *getDate()* (get-функція), яка повертає значення з змінних *day*, *month*, *year* у вигляді дати.

1.4. Приклад A

Розробити програму, в якій оголошено найпростіший клас, в якому оголошена одна функція, яка друкує повідомлення.

```
// оголошення класу
  class CppClass
  {
       public: // специфікатор доступу
       void message () // функція виводить повідомлення на екран
                {
                  cout << " Classes and Objects in C++ \n\n\n";
                }
  }; // Кінець оголошення класу CppStudio
  int main (int argc, char * argv [])
  {
       CppClass obj; // Оголошення об'єкта
       obj.message(); // Виклик функції класу message
       system("pause");
       return 0;
  }
```

1.5. Приклад B

Розробити клас, що реалізовує стек символів.

```
  #define size 10
  class steck
  {
       char stck [size];
       int tos;
```

```
        public:
        void init();
        void puch(char ch);
        char pop();
   } ;
//----------------------
   void steck::init()
   {
        tos=0;
   }
   //--------------------------
   void steck::puch(char h)
   {
        if (tos==size) {cout<<"Стек заповнений\n";return;}
        stck[tos]=h;tos++;
   }
//-------------------------
   char steck::pop()
   {
        if (tos==0){cout<<"Стек порожній";return(0) ;}
        tos--;
        return stck[tos];
   }
//=========================================
   main()
   {    steck d;
        d.init();
        d.puch('1');
        d.puch('2');
        d.puch('f');
        d.puch('f');
        cout<<d.pop()<<endl;
        d.puch('d');
        d.puch('d');
        d.puch('f');
        d.puch('a');
        d.puch('f');
        d.puch('a');
        d.puch('f');
        d.puch('a');
        cout<<d.pop()<<endl;
```

```
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
    cout<<d.pop()<<endl;
}
```

1.6. Оформлення звіту

Звіт повинен містити:

- номер і назву лабораторної роботи;
- умову задачі та дані свого варіанту;
- *C*-програму з коментарями;
- результати роботи програми.

1.7. Контрольні питання

1. Що визначає клас?
2. Поясните принцип інкапсуляції.
3. Для чого використовуються ключові слова **public** і **private**?
4. Що називають операцією розширення області видимості? Наведіть приклад.
5. Поясните принцип успадкування.

2. ЛАБОРАТОРНА РОБОТА № 2
СТВОРЕННЯ КЛАСІВ З ВИКОРИСТАННЯМ КОНСТРУКТОРІВ РІЗНОГО ТИПУ

2.1. Мета роботи

Ознайомитись на практиці з класами, об'єктами та головними елементами об'єктного підходу.

Навчитись створювати і використовувати об’єкти типу клас, використовуючи спеціальні методи класу – конструктори і деструктори.

2.2. Постановка задачі

На мові C++ розробити програму, що використовує клас *«Massiv»*. Цей клас повинен описувати *N*-мерный масив даних, розміщений в динамічній області пам'яті. Використовуючи два об’єкта цього класу виконати завдання згідно заданого варіанту.

Розміри масивів і значення елементів задавати довільно. Клас «Massiv» повинен містити три функції–елементи:

- клас для введення елементів масиву з клавіатури і запису їх в динамічну область пам'яті;
- клас доступу до будь-якого елементу даних, розміщеного в динамічній області пам'яті, по заданих індексах масиву;
- клас визначення деякого значення, відповідно до заданого варіанту.

В класі використовувати конструктори за умовчанням і конструктори з параметрами.

2.3. Теоретичні відомості

Конструктори і деструктори

Коли ми створюємо елементи (змінні) класу, ми не можемо присвоїти їм значення у самому визначенні класу. Компілятор видасть помилку. Тому нам

необхідно створювати окремий метод (так звану *set*-функцію) класу, за допомогою якого і буде відбуватися ініціалізація елементів. При цьому, якщо необхідно створити, наприклад, 20 об'єктів класу, то прийдеться 20 разів викликати *set*-функції. Тут нам якраз зможе допомогти ***конструктор*** класу.

Конструктор (*construct* – створювати) – це спеціальний метод класу, призначений для ініціалізації елементів класу деякими початковими значеннями.

Деструктор (*destruct* – руйнувати) – спеціальний метод класу, який служить для «знищення» елементів класу. Найчастіше його використовують тоді, коли в конструкторі, при створенні об'єкта класу, динамічно була виділена ділянка пам'яті і необхідно цю пам'ять очистити, якщо ці значення вже не потрібні для подальшої роботи програми.

Важливо пам'ятати:

- конструктор і деструктор завжди оголошуємо в розділі *public*;
- при оголошенні конструктора тип повернення не вказується, в тому числі – *void*;
- у деструктора так само немає типу повернення, деструктору не можна передавати ніяких параметрів;
- ім'я класу і конструктора повинні бути ідентичними;
- ім'я деструктора ідентично імені конструктора, але з приставкою ~;
- у класі допустимо створювати декілька конструкторів, якщо це необхідно. Імена будуть однаковими, а компілятор буде їх розрізняти по переданим параметрами (перевантаження функцій). Якщо ми не передаємо в конструктор параметри, він вважається конструктором за замовчуванням;
- у класі може бути оголошений лише **один** деструктор.

2.4. Приклад А

Розробити С-програму з використанням конструкторів.

```
class AB //класс
{
private:
int a;
```

```
int b;
public:
AB() //це конструктор
{
 a = 0; b = 0; //початкові значення полів
 cout << "\n\n\tРабота конструктора: " << endl;
 cout << "\ta = " << a << endl;
 cout << "\tb = " << b << endl << endl;
}
void setAB() // змінюємо початкові значення
  {
      cout << "\tВведите целое число a: "; cin >> a;
      cout << "\tВведите целое число b: "; cin >> b;
  }
void getAB()
 {
      cout << "\ta = " << a << endl;
      cout << "\tb = " << b << endl << endl;
  }
};
int main()
{
setlocale(LC_ALL, "rus");
AB obj1; //спрацює конструктор для І-го об'єкта
obj1.setAB(); //задаємо нові значення
obj1.getAB(); //виводимо їх на екран
AB obj2; //спрацює конструктор для ІІ-го об'єкта
system("pause");    return 0; }
```

Результат роботи програми:

```
Работа конструктора при создании нового объекта:
a = 0
b = 0

Введите целое число a: 13
Введите целое число b: 17
a = 13
b = 17



Работа конструктора при создании нового объекта:
a = 0
b = 0
```

Успадкування класів

Успадкування класів дозволяє створювати ***похідні*** класи (класи спадкоємці), взявши за основу всі методи й елементи ***базового*** класу (класу батька). Таким чином, заощаджується маса часу на написання і налагодження коду нової програми. Об'єкти похідного класу вільно можуть використовувати все, що створено і налагоджено в базовому класі. При цьому ми можемо в похідний клас дописати необхідний код для удосконалення програми: додати нові поля, методи і т. д. Базовий клас залишиться недоторканим.

Основна інформація про успадкування класів:

Успадкування – це визначення ***похідного*** класу, який може звертатися до всіх елементів і методів базового класу, за винятком тих, що перебувають у розділі ***private***.

Похідний клас ще називають ***нащадком*** або підкласом, а базовий – ***батьківським***, або надкласом, або суперкласом.

Синтаксис ***визначення*** похідного класу:

class Імя_Похідн_Класу**: специфікатор доступу** Імя_Баз_Класу **{ . . . };**

Похідний клас має доступ до всіх полів і методів базового класу, а ***базовий*** клас може використовувати тільки свої власні поля і методи.

У похідному класі необхідно ***явно*** визначати свої ***конструктори***, ***деструктори*** і ***перевантажені*** оператори присвоювання через те, що вони не успадковуються від базового класу. Але їх можна викликати ***явним чином*** при визначенні конструктора, деструктора або перевантаження оператора присвоєння похідного класу, наприклад, таким чином (для конструктора):

Конструктор_Похідн_Класу (...): **Конструктор**_Баз_Класу (...) {...}.

2.5. Приклад В

Розробити програму, в якій було би створено два класи: базовий – *FirstClass* і похідний від нього *SecondClass*.

```
class FirstClass // базовий клас
{
protected: // специфікатор доступу до элементу value
int value;
public:
FirstClass() { value = 0; }
FirstClass(int x) { value = x; }
void show_value() { cout << value << endl; }
};
class SecondClass : public FirstClass // похідний клас
{
public: // конструктор класу SecondClass викликає конструктори класу
// FirstClass
SecondClass() : FirstClass() {}
SecondClass(int inputS) : FirstClass(inputS) {}
void ValueSqr() // Без специфікатора protected не змогли б змінити value
{
value *= value;
}
};
int main()
{ setlocale(LC_ALL, "rus");
FirstClass F_object(3); // об'єкт базового класу
cout << "\n\tvalue F_object = ";
F_object.show_value();
SecondClass S_object(4); // об'єкт похідного класу
cout << "\tvalue S_object = ";
S_object.show_value(); // виклик методу базового класу
S_object.ValueSqr(); // підносимо value до квадрату
cout << "\tквадрат value S_object = ";
S_object.show_value();
// F_object.ValueSqr(); // ПОМИЛКА: немає доступу
cout << endl;
system("pause");
return 0;
}
```

Результат роботи програми:

```
value F_object = 3
value S_object = 4
квадрат value S_object = 16
```

2.6. Оформлення звіту

Звіт повинен містити:

- номер і назву лабораторної роботи;
- умову задачі та дані свого варіанту;
- опис класів;
- таблицю символічних імен;
- *C*-програму з коментарями;
- результати розрахунку програми.

2.7. Варіанти завдань

№№ п/п	Розмірність масиву	Тип елементів масиву	Обчислюване значення
1	1	Цілий	Сума всіх елементів
2	1	Дійсний	Сума від’ємних елементів
3	1	Символьний	Кількість голосних
4	1	Цілий	Добуток всіх елементів
5	1	Дійсний	Добуток додатніх елементів
6	1	Символьний	Кількість приголосних
7	1	Цілий	Сума парних елементів
8	2	Дійсний	Сума елементів з непарними індексами
9	2	Символьний	Кількість символу ‘A’
10	2	Цілий	Сума елементів з парними індексами
11	2	Дійсний	Сума ASCII кодів всіх елементів
12	2	Символьний	Кількість символу ‘C’
13	2	Цілий	Добуток елементів з непарними індексами
14	2	Дійсний	Сума ASCII кодів парних елементів
15	2	Символьний	Кількість символу ‘B’
16	2	Цілий	Сума додатніх елементів з непарними індексами
17	2	Дійсний	Сума елементів, відмінних від нуля
18	1	Символьний	Кількість символу ‘+’
19	1	Цілий	Сума елементів з парними індексами, відмінних від нуля
20	1	Дійсний	Сума від’ємних елементів з непарними індексами
21	1	Символьний	Кількість символу ‘D’
22	1	Цілий	Сума елементів більших за 10.
23	2	Дійсний	Кількість від’ємних елементів
24	2	Символьний	Кількість символу ‘?’
25	2	Цілий	Добуток додатніх елементів

2.8. Контрольні питання

1. Чим відрізняється клас від об'єкту?
2. Що таке метод? Як викликається метод?
3. Чи обов'язково робити поля класу приватними?
4. Чи може метод бути приватним?
5. Поясніть принцип поліморфізму.
6. Що таке успадкування?
7. Що таке екземпляр класу?
8. До яких полів і методів базового класу має доступ похідний клас?
9. Які поля і методи може використовувати базовий клас?
10. Як треба визначати конструктори і деструктори у похідному класі? Чому?

ЛІТЕРАТУРА

1. Пахомов Б. C/C++ и MS Visual C++ 2010 для начинающих. – СПб.: БХВ-Петербург, 2011. – 736 с. ISBN: 978-5-9775-0599-4.

2. Дейтел Х.М. Как программировать на С: 3-е издание / Х.М. Дейтел, П.Дж. Дейтел ; пер. с англ. В.В. Тимофеев. – М: Бином-Пресс, 2002. – 1168 с. – ISBN 5-9518-0002-1 (в пер.).

3. Дейтел Х.М. Как программировать на С++: 5-е издание / Х.М. Дейтел, П. Дж. Дейтел ; пер. с англ. В.В. Тимофеев. – М: Бином-Пресс, 2008. – 1456 с. – ISBN 978-5-9518-0224-8 (в пер.).

4. Шилдт Г., Полный справочник по С / Г. Шилдт. – СП.: Вильямс, 2003. – 800 с. – ISBN 5-8459-0226-6 (в пер.).

5. Алгоритмы: построение и анализ, 2-е издание / Т.Х. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К. Штайн ; пер. с англ. И.В. Красикова и др. – М. : Издательский дом «Вильямс», 2005. – 1296 с. – ISBN 5-8459-0857-4 (в пер.).

6. Лаптев В.В. С++. Объектно-ориентированное программирование: Учебное пособие / В.В. Лаптев. – СПб.: Питер, 2008. – 464 с. – ISBN 978-5-91180-200-4.

7. Керниган Б.У. Язык программирования С, 2-е издание / Б.У. Керниган, Д.М. Ритчи ; пер. с англ. В.Л. Бродового. – М. : Издательский дом «Вильямс», 2009. – 304 с. – ISBN 978-5-8459- 0891-9 (в пер.).

8. Страуструп Б. Язык программирования С++. Специальное издание / Б. Страуструп ; пер. с англ. Н.Н. Мартынова. – М.: Бином, 2011. – 1136 с. – ISBN 978-57989-0425-9 (в пер.).

Упорядники:

Саксонов Геннадій Михайлович

Жукова Олена Андріївна

КЛАСИ ЯК ОСНОВА ООП

Методичні рекомендації та варіанти завдань до виконання лабораторних робіт з дисципліни «Програмування та алгоритмічні мови»

бакалаврами напряму підготовки 12 Інформаційні технології

Видано в редакції упорядників

Комп'ютерний дизайн, верстка та обробка – **Г.М. Саксонов**

Підписано до друку 27.04.2021. Формат 30х42/4.
Папір офсет. Ризографія. Ум. друк. арк. 1,1.
Обл.-вид. арк. 1,1. Тираж 6 пр. Зам. №

Національний технічний університет «Дніпровська політехніка»
49005, м. Дніпро, просп. Д. Яворницького, 19.