МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

**Об'єктно-орієнтоване програмування**

**Лабораторний практикум**

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського   
як навчальний посібник для здобувачів ступеня бакалавра*

*за освітньою програмою «Прикладна математика»  
спеціальності 113 «Прикладна математика»*

Київ

КПІ ім. Ігоря Сікорського

2021

Об'єктно-орієнтоване програмування: Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 113 «Прикладна математика» / Н. М. Куссуль, А. Ю. Шелестов, А. М. Лавренюк, Д. В. Тітков; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 1441 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 52 с.

*Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № \_8\_ від 24.06.2021 р.)   
за поданням Вченої ради Фізико-технічного інституту Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського»*

*(протокол № 10 від 26.04.2021 р.)*

Електронне мережне навчальне видання

Об'єктно-орієнтоване програмування Лабораторний практикум

Укладачі: *Куссуль Наталія Миколаївна,* д. техн. наук, проф.

*Шелестов Андрiй Юрiйович*, д. техн. наук, проф.

*Лавренюк Алла Миколаївна,* канд. техн. наук, доц.

*Тітков Дмитро Валерійович*

|  |  |
| --- | --- |
| Відповідальний редактор | *Смирнов С.А.,* к.ф.-м.н., доц. |
| Рецензент | *Яйлимов Б. Я.*, к.т.н, с.н.с., Інституту космічних досліджень НАН України та ДКА України |

Навчальний посібник «Об'єктно-орієнтоване програмування. Лабораторний практикум» присвячено вивченню принципів об’єктно-орієнтованого програмування та їх реалізації на прикладі мови програмування С++. Метою є отримання навичок об’єктно-орієнтованого програмування та проектування об’єктно-орієнтованих систем. Посібник містить необхідний теоретичний матеріал, приклади програмного коду, а також завдання для виконання лабораторного практикуму студентами за спеціальністю 113 «Прикладна математика».

© КПІ ім. Ігоря Сікорського, 2021

ЗМІСТ

[**Лабораторна робота №1  
Класи та об’єкти**](#_30j0zll)5

[1.1 Теоретичні відомості](#_3znysh7) 5

[1.2 Приклад виконання](#_cofkb3h1m4n5) 8

[1.3 Порядок виконання роботи](#_b3wz1xljdrm) 9

[1.4 Варіанти завдань](#_7ev6dbeybtyp) 9

[1.5 Контрольні питання](#_ru5dxbjozv08) 9

[**Лабораторна робота №2  
Проектування класів та їх методів**](#_26in1rg)10

[2.1 Теоретичні відомості](#_wkuody1yqg3z) 10

[2.2 Приклад виконання](#_j72lnhwilt2r) 13

[2.3 Порядок виконання роботи](#_omyl6rpfmrde) 16

[2.4 Варіанти завдань](#_e76djolzkq0n) 17

[2.5 Контрольні питання](#_kaznbnmqosih) 21

[**Лабораторна робота №3  
Перевантаження операцій**](#_4i7ojhp)23

[3.1 Теоретичні відомості](#_mgud2zyu9w7m) 23

[3.2 Приклад виконання](#_x03p02ecwmai) 24

[3.3 Порядок виконання роботи](#_n6v3f4adnpt2) 28

[3.4 Варіанти завдань](#_yhnrgth94qq4) 28

[3.5 Контрольні питання](#_5mvpljch9uld) 28

[**Лабораторна робота №4  
Перевантаження операторів потокового введення-виведення**](#_2p2csry)30

[4.1 Теоретичні відомості](#_5idv0p3iktu6) 30

[4.2 Приклад виконання](#_jo80fccyv8ld) 32

[4.3 Порядок виконання роботи](#_pyng5kjqznrr) 33

[4.4 Варіанти завдань](#_4t2gxqwvebmz) 33

[4.5 Контрольні питання](#_wcktw8izzh4d) 33

[**Лабораторна робота № 5  
Параметризовані класи (шаблони)**](#_41mghml)35

[5.1. Теоретичні відомості](#_fb3x5hq7hkbg) 35

[5.2 Приклад виконання](#_p94x8zcxg9kp) 37

[5.3 Порядок виконання роботи](#_5fq2iawfvwe2) 39

[5.4 Варіанти завдань](#_3tshse7pujkm) 39

[5.5 Контрольні питання](#_en87zox2hhpd) 39

[**Лабораторна робота №6  
Програмне середовище Qt Creator**](#_ajoq5osfco7b)40

[6.1 Теоретичні відомості](#_sayft232nwly) 40

[6.2 Приклад виконання](#_izvz9hihredz) 42

[6.3 Порядок виконання роботи](#_uu2wybbugnhy) 47

[6.4 Варіанти завдань](#_mtc55zk041fa) 47

[6.5 Контрольні питання](#_nmf14n) 49

[**Лабораторна робота № 7  
Використання об’єктно-орієнтованого підходу для розробки програмного забезпечення**](#_dhof1ksjk6hr)50

[7.1 Теоретичні відомості](#_16qb73lefyv) 50

[7.2 Приклад виконання](#_db57qnher18p) 53

[7.3 Порядок виконання роботи](#_mzi63hc083m0) 53

[7.4 Варіанти завдань](#_wipxlo54c4d0) 54

[7.5 Контрольні питання](#_olbbnjhrpo4m) 55

[**Список використаної літератури**](#_8yi803ml5pxd)56

# Лабораторна робота №1 Класи та об’єкти

**Мета роботи:** засвоїти основні поняття об’єктно-орієнтованого програмування на прикладі мови програмування С++.

## 1.1 Теоретичні відомості

**Основні поняття, потрібні для виконання роботи.**

Це:

1. класи
2. область доступу
3. ієрархія класів

Перейдемо до послідовного розгляду цих понять.

Головна мета введення концепції класів в С++ - забезпечення програміста засобами для створення нових типів, які були б такими ж зручними у використанні, як і вбудовані.

**Тип** - конкретний представник деякої концепції.

**Клас** - це тип, що визначає користувач. Для визначення концепції, яка не виражається безпосередньо вбудованими типами, створюються нові типи. Ретельно підібраний набір типів, які визначає користувач, робить програму коротшою і виразнішою.

Основний сенс введення нових типів полягає в обмеженні доступу до даних ззовні та у використанні для цього спеціальних процедур у межах чітко визначеного інтерфейсу.

**Функції-члени**

Функції, визначені всередині опису класу (до речі, структура - це один з видів класу), називаються ***функціями-членами***, і їх можна викликати тільки для змінної відповідного типу, використовуючи стандартний синтаксис доступу до членів структури.

Оскільки різні структури можуть мати функції-члени з однаковими назвами, то, визначаючи функцію-член треба вказати ім’я структури:

**void** Date::init(**int** dd, **int** mm, **int** yy)

{…}

Визначенням класу є конструкція class C {…}

**Контроль доступу.** Розглянемо приклад:

**class** X {

**public**:

**void** init();

**int** getITnow();

**private**:

**int** x,y,z;

**int** px,py,pz;

};

Імена в закритій private частині можна використовувати тільки у функціях-членах класу. Відкрита publicчастина утворює відкритий інтерфейс об’єктів класу. (Структура - клас, члени якого відкриті за замовчуванням). Крім того, існує мітка protected(захищений), тобто всі змінні будуть доступні тільки прямим "спадкоємцям" цього класу.

Захист закритих даних базується на обмеженні використання імен членів класу. Цей захист можна обійти, маніпулюючи з адресами і явним перетворенням типу. Захист проти зловмисного доступу до закритих даних мовою високого рівня можна забезпечити тільки на апаратному рівні, хоча навіть це - досить складне завдання в реальній системі.

**Статичні члени**

Змінну, яка є частиною класу, а не частиною об’єкта цього класу, називають ***статичним членом*** і позначають специфікаторомstatic***.***

**Константні функції-члени.** Нехай у класі Х існують функції, які надають і змінюють значення об’єкта типу Х. Але, на жаль, не існує способу для перевірки значення об’єкта Х. Проте цю проблему можна легко вирішити, описавши ці функції як ***константні функції-члени***, тобто функції, які не змінюють стан Х:

**class** Date {

**int** d,m,y;

**public**:

**int** day() **const** {**return** d;}

**int** month() **const** {**return** m;}

**int** year() **const** {**return** y;}

//…

};

Коли константна функція-член описується зовні, а не всередині класу, то потрібно додати суфікс const:

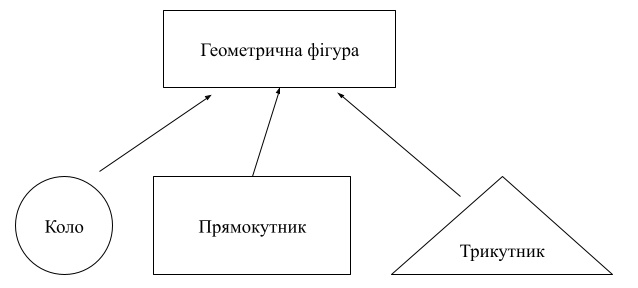
**inline int** Date::year() **const**  //правильно

{ **return** y;}

Константну функцію-член можна викликати як для константного, так і для неконстантного об’єкта, тоді як неконстантну функцію-член можна викликати тільки для об’єкта, який не є константою.

**Підкласи**

***Підкласи*** - це класи, які успадковують усі "властивості" суперкласу ("батьківського класу").(рис. 1.1).



**Рис. 1.1 - Суперклас та підкласи**

**Віртуальні функції**

Віртуальні функції визначаються специфікатором virtual і дозволяють програмісту описати в базовому класі функції, які можна було б замінити у кожному наступному класі.

**Ієрархія класів**

Об’єкти різних класів і самі класи можуть перебувати у відношенні успадкування, за якого формується ієрархія об’єктів, що відповідає заздалегідь передбаченій ієрархії класів.

Ієрархія класів дозволяє визначати нові класи на основі вже існуючих. Існуючі класи зазвичай називають ***базовими*** (інколи ***породжувальними***),   
а нові класи, що формуються на основі базових, - ***похідними*** (***породженими***), інколи ***класами-нащадками*** або ***спадкоємцями***. Похідні класи “отримують спадок” - дані і методи своїх базових класів - і, крім того, можуть поповнюватись власними компонентами (даними і власними методами).

Наприклад, за таким визначенням

class S: X{…};

клас S породжений класом X, звідки він успадковує компоненти.

## 1.2 Приклад виконання

#**include** <string>

#**include** <iostream>

**using namespace** std;

**class** Student {

**private**:

string name;

**int** age, course;

**public**:

**void** setData();

**void** getData();

};

**void** Student::setData()

{

cout<<"Enter name"<< endl;

cin>>name;

}

**void** Student::getData()

{

cout<< " Name=" <<name<<";"<<endl;

cout<< " Age=" <<age;

}

//-------------------------------------------------

**int** main()

{

Student Olga; //створення об’єкта

Student\* Nick;//створення вказівника

Olga.setData();//виклик метода setData() об’єкта Olga

Olga.getData();

Nick=**new** Student; //виділення пам’яті під об’єкт

Nick->setData(); //виклик функції об’єкта \*Nick

Nick->getData();

cin.get();

cin.get();

**return** 0;

}

## 1.3 Порядок виконання роботи

Створити консольний проект, у якому використовується наведений вище клас Student. Створити екземпляри цього класу і дослідити можливість доступу до закритої та відкритої областей.

Створити простий клас згідно з номером варіанта, використовуючи властивість інкапсуляції даних та звернення до об’єктів цього класу напряму́ і через вказівник.

## 1.4 Варіанти завдань

1. Клас котів.
2. Клас собак.
3. Клас викладачів.
4. Клас робітників.
5. Клас дітей.
6. Клас тварин.
7. Клас птахів.
8. Клас рослин.
9. Клас прямокутників.
10. Клас трикутників.
11. Клас кіл.
12. Клас відрізків.
13. Клас свят.
14. Клас автомобілів.
15. Клас грошей.
16. Клас книг.

## 1.5 Контрольні питання

1. Що таке клас?
2. Чим клас відрізняється від структури?
3. Що таке область доступу?
4. Наведіть усі модифікатори доступу, які знаєте, поясніть їх роботу?
5. Що таке поле класу?
6. Що таке метод(функція-член) класу?
7. Що таке статичне поле (член) класу, як позначають?
8. Що таке константні методи (функції-члени) класу, як позначають?
9. Що таке віртуальні методи (функції-члени) класу, як позначають?
10. Розкажіть усе, що знаєте про наслідування?

# Лабораторна робота №2 Проектування класів та їх методів

**Мета роботи**: засвоїти поняття конструктора, деструктора та функцій-членів класів.

## 2.1 Теоретичні відомості

**Конструктори**

Одне з основних завдань об’єктно-орієнтованого програмування полягає у тому, щоб об’єкти описаного раз і назавжди класу працювали «правильно» - тобто так, як це визначає модель. Кожний об’єкт перед тим як почати роботу, потрібно створити, тобто перевести в якийсь початковий стан. Отже, треба якимось чином описати можливі механізми створення об’єктів даного класу. Для цього в мові C++ існують ***конструктори****.* Це особливі методи класу, які й повинні перевести об’єкт у той самий початковий стан. Конструктор описується як метод, ім’я якого збігається з іменем класу, а тип поверненого значення опущений.

**Приклад:**

**class** Point

{

**public**:

Point(**int** x0, **int** y0);

**private**:

int x, y;

};

Point::Point(**int** x0, **int** y0)

{

x=x0;

y=y0;

}

Тепер для створення об’єкта класу Point потрібно після імені змінної вказати параметри, як для виклику функції:

Point A(1, 1), B(2, 0);

**Типи конструкторів**

Існують деякі типи конструкторів, які, крім безпосереднього використання, автоматично викликаються у деяких особливих ситуаціях.

**Конструктор за замовчуванням.** Це конструктор, що викликається без параметрів:

Point();

Point(int a=5);

Його використовують для створення масиву об’єктів, оскільки не зрозуміло, які конструктори і з якими параметрами треба викликати для кожного елемента масиву. Наприклад:

Point A[10];

Point\* B = **new** Point[10];

Конструктор за замовчуванням викликається також тоді, якщо не вказано параметри для ініціалізації об’єкта, як у цьому випадку:

Point p;

**Конструктор копіювання.** Цей конструктор викликається тоді, коли потрібно створити копію об’єкта. Аргументом цього конструктора має бути посилання на об’єкт цього самого класу:

Point(Point& p);

Важливим випадком, коли викликається конструктор копіювання, є передавання об’єкта у функцію як параметра за значенням. Тоді створюється новий об’єкт і для нього автоматично викликається конструктор копіювання. Створення конструкторів копіювання потрібне у випадку, якщо об’єкт потребує якихось спеціальних операцій при копіюванні, оскільки під час стандартного копіювання вміст одного об’єкта просто побайтно переноситься в інший.

**Приклад:**

**class** String

{

**public**:

String();// конструктор за замовчуванням

String(**const** String& s); // конструктор копіювання

String(**const** **char**\* s);// конструктор з параметром

// const char\*, який являє собою

// стандартний рядок s

~String();// деструктор

**private**:

**char**\* array; // масив символів

**int** size; // розмір масиву

};

**Приклад виклику конструкторів:**

**int** main()

{ String a, b; // конструктор за замовчуванням

String c(a); // конструктор копіювання

print(a); // конструктор копіювання, оскільки

// аргумент передається у функцію за значенням

String d(“One”); // конструктор з параметром

//...

}

**Деструктори**

Конструктори ініціалізують об’єкт, тобто вони створюють середовище, у якому "працюють" функції-члени. Іноді створення такого середовища зумовлює "захоплення" якихось ресурсів: пам’яті, файлу, процесорного часу, які повинні бути "звільнені" після їх використання. Тобто класам потрібна функція, яка б знищувала об’єкт аналогічно тому, як його створює конструктор. Такі функції називають ***деструкторами.***

**Приклад:**

**class** Name

{

**const char**\* s;

// ...

};

**class** Table

{

Name\* p;

size\_t sz;

**public**:

Table(size\_t s=15) //конструктор

{ p=**new** Name[sz=s]; }

~Table() {**delete**[] p;} //деструктор

// ...

};

**Успадкування класів**

Об’єкти різних класів і самі класи можуть перебувати у відношенні успадкування, за якого формується ієрархія об’єктів, що відповідає заздалегідь передбаченій ієрархії класів.

Ієрархія класів дозволяє визначати нові класи на основі вже існуючих. Існуючі класи зазвичай називають ***базовими*** (інколи ***породжувальними***), а нові класи, що формуються на основі базових, - ***похідними*** (***породженими***), інколи ***класами-нащадками*** або ***спадкоємцями***. Похідні класи “отримують спадок” - дані і методи своїх базових класів - і, крім того, можуть поповнюватись власними компонентами (даними і власними методами). Елементи, які успадковуються, не переміщуються в похідний клас, а залишаються в базових класах. Повідомлення, обробку якого не можуть виконати методи похідного класу, автоматично передається в базовий клас.

Будь-який похідний клас може, в свою чергу, стати базовим для інших класів, і таким чином формується напрямлений граф ієрархії класів та об’єктів. В ієрархії похідний об’єкт успадковує дозволені для успадкування компоненти всіх базових об’єктів. Іншими словами, в об’єкта є можливість доступу до даних і методів усіх своїх базових класів.

Успадкування в ієрархії класів може відображатись і у вигляді дерева, і у вигляді більш загального ***напрямленого ациклічного графу***. Дозволяється ***множинне успадкування*** - можливість для деякого класу успадковувати компоненти кількох ніяк не зв’язаних між собою базових класів.

Наприклад, за таким визначенням

class S: X, Y, Z {...};

клас S породжений класами X, Y, Z, звідки він успадковує компоненти.

## 2.2 Приклад виконання

***Клас точок***

**Базовий клас (точки у двовимірній площині з цілочисельними координатами):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* обчислення відстані між точками;
* додавання координат двох точок;
* введення - виведення на екран;
* перевірка збіжності двох точок.

**Похідний клас**: пікселів на екрані (точки, які мають колір).

#**include** <iostream>

#**include** <math.h>

#**include** <string>

**using namespace** std;

**class** Point {

**protected**:

**int** x, y;

**public**:

Point();

Point(**int** k, **int** g);

Point(**const** Point& p);

**int** getx() { **return** x; }

**int** gety() { **return** y; }

**void** setx(**int** t) { x = t; }

**void** sety(**int** t) { y = t; }

**float** distance(**const** Point& p);

**void** plus(**const** Point& p);

**bool** testconv(**const** Point& p);

**void** displayin();

**virtual void** displayout();

};

Point::Point()

{

x = 5;

y = 5;

}

Point::Point(**int** k, **int** g)

{

x = k;

y = g;

}

Point::Point(**const** Point& p)

{

x = p.x;

y = p.y;

}

**void** Point::plus(**const** Point& p)

{

x = x + p.x;

y = y + p.y;

}

**bool** Point::testconv(**const** Point& p)

{

**if** ((x == p.x) && (y == p.y)) **return** true;

**return** false;

}

**float** Point::distance(**const** Point& p)

{

**return** sqrt((float)(x-p.x)\*(x-p.x)+(y-p.y)\*(y-p.y));

}

**void** Point::displayin()

{

cout << "enter point\n";

cin >> x >> y;

}

**void** Point::displayout()

{

cout << "Point(" << x << "," << y << ")\n";

}

**class** PColor : **public** Point {

**private**:

string c;

**public**:

**void** displayout();

PColor()

{ c = "white"; }

PColor(**int** x1, **int** y1, **string** c1):Point(x1, y1)

{ c = c1; }

PColor(const PColor& p) : Point(p)

{ c = p.c; }

};

**void** PColor::displayout()

{

Point::displayout();

cout << "Color " << c<<endl;

}

**int** main()

{

**int** x, y;

Point a;

a.displayout();

a.displayin();

cout<<"???: "<<a.getx()<<" "<<a.gety()<< endl;

a.displayout();

cout << "copy constructor test: ";

Point b(a);

b.displayout();

cout << "enter x,y:\n";

cin >> x >> y;

a.setx(x);

a.sety(y);

cout << "setter test: ";

a.displayout();

cout << "parameter constructor test: ";

Point c(x, y);

c.displayout();

cout << "distance: " << c.distance(b) << endl;

if (b.testconv(c))

cout << "they are equal\n";

else

cout << "they are not equal\n";

cout << "adding points result: ";

a.plus(b);

a.displayout();

PColor f;

f.displayout();

PColor f1(3,3,"green");

f1.displayout();

PColor f2(f1);

f2.displayout();

cin.get();

cin.get();

**return** 0;

}

## 2.3 Порядок виконання роботи

Згідно з номером варіанта спроектувати класи, реалізувавши конструктори та відповідні методи. В головній функції проілюструвати їх використання.

## 

## 2.4 Варіанти завдань

**1.*Клас точок***

**Базовий клас (точки у двовимірній площині з цілочисельними координатами):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* обчислення відстані між точками;
* додавання координат двох точок;
* введення - виведення на екран;
* перевірка збіжності двох точок.

**Похідний клас**: пікселів на екрані (точки, які мають колір).

**2.*Клас комплексних чисел***

**Базовий клас (уявних чисел):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* додавання;
* віднімання;
* множення на дійсне число;
* виведення інформації на екран.

**Похідний клас**: комплексних чисел.

**3.*Відомість успішності студентів (2 класи: базовий - студент; похідний - студент з відомістю успішності)***

**Базовий клас (студент):**

*Змінні:* ім’я, вік, вищий навчальний заклад.

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* завдання вищого навчального закладу;
* змінювання імені;
* введення-виведення інформації про вік;
* виведення всіх даних.

**Похідний клас**: студент з відомістю успішності - до базового класу додається масив оцінок.

**4.*Динамічний одновимірний масив цілих чисел***

**Базовий клас:**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції (дві-три на вибір):*

* запис / одержання елемента масиву;
* виведення масиву на екран;
* пошук елемента масиву;
* інформація про розмір масиву;
* очищення масиву.

**Похідний клас**: динамічний одновимірний масив цілих чисел з довільними (включаючи й від’ємні) межами.

**5.*Текст (2 класи: рядок, текст перебувають у відношенні агрегації)***

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор.*

*Функції (дві-три на вибір):*

* вставлення (видалення) символу у даному рядку, у даному місці;
* очищення заданого рядка;
* видалення заданого рядка;
* вставлення рядка у задане місце;
* пошук рядка у тексті;
* очищення тексту.

**6.*Працівник***

**Базовий клас (основна інформація про працівника):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* введення-виведення інформації про працівника;
* змінення посади, тощо.

**Похідний клас**: працівник з відомістю отриманих премій - до базового класу додається динамічний масив премій.

**7.*Клас прямокутників***

**Базовий клас (прямокутник з цілочисельними сторонами):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* обчислення площі прямокутника;
* обчислення периметра прямокутника;
* зміна довжин сторін;
* виведення інформації про сторони на екран;
* перевірка чи є прямокутник квадратом.

**Похідний клас**: прямокутник, який має колір.

**8.*Клас кіл***

**Базовий клас (коло – координати центру, радіус):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* обчислення площі кола;
* обчислення довжини кола;
* зміна радіусу, координат центру;
* виведення інформації про коло на екран;
* перевірка, чи площа перевищує задану величину.

**Похідний клас**: коло, яке має колір.

**9.*Клас трапецій***

**Базовий клас (задаються координатами):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* обчислення площі трапеції;
* обчислення периметру;
* зміна координат;
* виведення інформації про трапецію на екран;
* перевірка, чи площа перевищує задану величину.

**Похідний клас**: трапеція, яке має колір.

**10.*Клас дробів***

**Базовий клас (правильний дріб, менший 1):**

*Конструктори*: за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції:*

* додавання;
* віднімання;
* множення;
* виведення інформації на екран.

**Похідний клас**: довільний дріб (з цілою частиною).

**Завдання підвищеної складності**

**11.*Динамічний двовимірний масив цілих чисел (матриця)\****

**Базовий клас:**

*Конструктори:* за замовчуванням, копіювання, з різною кількістю параметрів.

*Деструктор.*

*Функції (дві-три на вибір):*

* запис / одержання елемента масиву;
* виведення масиву на екран;
* пошук елемента масиву;
* інформація про розмір масиву;
* очищення масиву.

**Похідний клас**: динамічний двовимірний масив цілих чисел з довільними (включаючи й від’ємні) межами.

**12.*Послідовні кільцеві структури даних (стеки, черги)\****

**Базовий клас (кільцева структура даних):**

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор.*

*Функції:*

* виведення всіх елементів на екран;
* інформація про максимальну кількість елементів.

**Похідний клас**: стек.

*Функції:*

* додавання елемента в стек;
* інформація про поточну кількість елементів в стеку;
* видалення елемента (заданої кількості елементів) зі стеку.

**13.*Послідовні кільцеві структури даних (стеки, черги)\****

**Базовий клас (кільцева структура даних):**

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор.*

*Функції:*

* виведення всіх елементів на екран;
* інформація про максимальну кількість елементів.

**Похідний клас**: черга.

*Функції:*

* додавання елемента в чергу;
* інформація про поточну кількість елементів в черзі;
* видалення елемента (заданої кількості елементів) з черги.

**14.*Динамічний список (2 класи: елемент списку, список перебувають у відношенні агрегації)\****

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор*.

*Функції (дві-три на вибір):*

* вставлення елемента з голови (хвоста) у заданому місці;
* видалення елемента з голови (хвоста) із заданого місця;
* виведення списку на екран;
* пошук елемента списку;
* інформація про кількість елементів списку;
* очищення списку;
* одержання голови (хвоста) списку;
* одержання наступного (попереднього) елемента.

**15.*Реалізація бінарного дерева (2 класи: вузол дерева, дерево перебувають у відношенні агрегації)\****

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор.*

*Функції (дві-три на вибір):*

* включення вузла дерева у заданому місці;
* знищення частини дерева із заданого місця;
* виведення дерева на екран;
* пошук елемента по дереву;
* інформація про структуру дерева;
* очищення дерева;
* обхід дерева (перебирання всіх елементів).

**16.*Граф (2 класи: вузол графу, граф перебувають у відношенні агрегації)\****

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор.*

*Функції (дві-три на вибір):*

* включення до графу елемента зі своїми зв’язками;
* видалення із графу елемента зі своїми зв’язками;
* виведення графу на екран;
* пошук елемента графу;
* інформація про структуру графу;
* очищення графу;
* обхід графу (перебирання всіх елементів);
* отримання шляхів між двома заданими вузлами.

**17.*Множина (2 класи: елемент множини, множина перебувають у відношенні агрегації)\****

*Конструктори:* за замовчуванням, з параметрами та копіювання.

*Деструктор.*

*Функції (дві-три на вибір):*

* додавання елемента до множини;
* видалення елемента із множини;
* виведення всіх елементів множини на екран;
* перевірка входження елемента до множини;
* очищення множини;
* перебирання всіх елементів множини.

## 2.5 Контрольні питання

1. Що таке конструктор, види конструкторів?
2. Розкажіть усе, що знаєте про деструктори?
3. Розкажіть усе, що знаєте про успадкування класів? Що таке базові та похідні класи?
4. Як звернутися до поля чи методу базового класу і похідного, якщо у похідному ці поля або методи перекриті?
5. В який момент роботи програми здійснюється виклик конструктора класу?
6. Чи може конструктор мати параметри? Приклади конструкторів з різною кількістю параметрів
7. Чи обов’язково в класі описувати конструктор?
8. Що таке конструктор за замовчуванням (default constructor)? Приклади
9. Скільки конструкторів за замовчуванням може мати клас?
10. Чи може конструктор повертати значення?
11. Чи може конструктор оголошуватись у розділі private?
12. Чи може в класі бути оголошено два конструктори, що приймають однакову кількість параметрів?
13. Види організації взаємодії між класами?
14. Що таке агрегація?
15. Шо таке наслідування?
16. Шо таке композиція?

# Лабораторна робота №3 Перевантаження операцій

**Мета роботи**: засвоїти принципи перевантаження операцій.

## 3.1 Теоретичні відомості

**Перевантаження операцій**

На відміну від інших мов програмування С++ допускає перевантаження операцій =, +, -, \*, ==, && та інших для будь-яких типів даних. У результаті компілятор визначає типи операндів, і запускається відповідний варіант операції. Перевантажувати можна навіть операцію виклику функції. Перевантажувати оператори можна двома способами.

1. Звичайний опис функції з назвою operator *знак*. Наприклад:

Point **operator** +(Point A, Point B)

{

// тіло функції

}

Тоді вираз A+B інтерпретується як operator +(A, B).

2. Опис у тілі класу методу з тією самою назвою. Тоді викликається метод для першого операнда, і йому як аргументи передаються інші операнди:

**class** Point {

// тіло класу

Point **operator** +(Point p);

};

У цьому випадку вираз A+B інтерпретується як A.operator +(B).

Деякі операції - такі як =, new, delete, можна перевантажувати тільки другим способом.

**Вказівник \*this**

Вираз \*thisозначає об’єкт, для якого викликано функцію-член. Але це не звичайна змінна: неможливо отримати її адресу чи присвоїти їй якесь значення. У константній функції-члені класу Х this має тип constX\*для упередження будь-яких модифікацій самого об’єкта.

## 3.2 Приклад виконання

**Приклад виконання 1**

Розглянемо клас String із лабораторної роботи 2 і наведемо приклад перевантаження стандартних операцій:

**class** String {

**public**:

String();// конструктор за замовчуванням

String(**const** String& s);// конструктор копіювання

//конструктор з параметром **const char**\*, який являє //собою стандартний рядок s

String(**const** **char**\* s);

~String(); // деструктор

// отримати рядок

**const char**\* GetString();

//перевантаження операції присвоювання

String& **operator** =(**const** String& s);

//визначимо функцію operator + як дружню, щоб вона

//могла працювати з private-полями.

**friend** String **operator**+(**const** String& a, **const** String& b);

**private**:

**char**\* array;// масив символів

**int** size;// розмір масиву

};

String::String()

{

size=1;

array=**new** char[1];

array[0]=0;

}

String::String(**const** String& s)

{

size=s.size;

array=**new** char[size];

**for**(**int** i=0; i<size; i++) array[i]=s.array[i];

}

String::String(**const** **char**\* s)

{

size=1;

**while**(s[size-1]!=0) size++;

array=**new** char[size];

**for**(**int** i=0; i<size; i++) array[i]=s[i];

}

String::~String()

{

**delete**[] array;

}

**const char**\* String::GetString()

{

**return** array;

}

String& String::**operator** =(**const** String& s)

{

**delete**[] array;

size=s.size;

array=**new** char[size];

**for**(**int** i=0; i<size; i++) array[i]=s.array[i];

**return** \***this**;

}

//перевантаження операції +

String **operator**+(**const** String& a, **const** String& b)

{

String s;

**delete**[] s.array;

s.size=a.size+b.size-1;

s.array=**new** **char**[s.size];

**int** i;

**for**(i=0; i<a.size-1; i++) s.array[i]=a.array[i];

**for**(i=0; i<b.size; b++) s.array[i+a.size-1]=b.array[i];

**return** s;

}

Виклик перевантажених операцій здійснюється так:

int main()

{ String a, b, c;

c=a+b;

//...

}

**Приклад виконання 2**

Розглянемо клас Point із лабораторної роботи 2 і наведемо приклад перевантаження стандартних операцій:

#**include** <iostream>

#**include** <math.h>

#**include** <string>

**using namespace** std;

**class** Point {

**protected**:

**int** x, y;

**public**:

Point();

Point(**int** k, **int** g);

Point(**const** Point& p);

**int** getx() { **return** x; }

**int** gety() { **return** y; }

**void** setx(**int** t) { x = t; }

**void** sety(**int** t) { y = t; }

**float** distance(const Point& p);

**void** plus(const Point& p);

**bool** testconv(const Point& p);

**void** displayin();

**virtual void** displayout();

**friend** Point **operator** +(**const** Point& a, **const** Point& b);

**friend** Point **operator** -(**const** Point& a, **const** Point& b);

Point& **operator**=(**const** Point& d);

};

//...

Point **operator** +(**const** Point& a, **const** Point& b)

{

Point c;

c.x = a.x + b.x;

c.y = a.y + b.y;

**return** c;

}

Point **operator** -(**const** Point& a, **const** Point& b)

{

Point j;

j.x = a.x - b.x;

j.y = a.y - b.y;

**return** j;

}

Point& Point::**operator**=(**const** Point& d)

{

x = d.x;

y = d.y;

**return** \***this**;

}

**int** main()

{

Point a, b, c, d;

a.displayin();

b.displayin();

c = a + b;

d = a - b;

c.displayout();

d.displayout();

cin.get();

**return** 0;

}

## 3.3 Порядок виконання роботи

Перевантажити оператори. Використати спроектовані класи структур даних згідно з варіантом, та спроектовані об’єкти цих класів у лабораторній роботі 2.

## 3.4 Варіанти завдань

**1.** Операції: = (присвоювання)

+, – (поелементні математичні операції).

**2.** Операції: = (присвоювання)

+, – (поелементні математичні операції).

**3.** Операції: = (присвоювання)

== (перевірка на рівність масивів оцінок).

**4.** Операції: [ ] (одержання елемента масиву)

= (присвоєння масивів)

+, –, \* (поелементні математичні операції).

**5.** Операції: = (присвоювання)

+ (додавання одного тексту до іншого).

**6.** Операції: = (присвоювання)

== (перевірка на рівність масивів премій).

**7.** Операції: = (присвоювання)

+ (додавання площі одного прямокутника до площі іншого).

**8.** Операції: = (присвоювання)

+ (додавання площі одного кола до площі іншого).

**9.** Операції: = (присвоювання)

+ (додавання площі одної трапеції до площі іншої).

**10.** Операції: = (присвоювання)

+ (додавання дробів).

**11.** Операції: = (присвоювання масивів)

+, –, \* (поелементні математичні операції).

**12.** Операції: = (присвоювання).

**13.** Операції: = (присвоювання).

**14.** Операції: = (присвоювання).

**15.** Операції: = (присвоювання).

**16.** Операції: = (присвоювання).

**17.** Операції: = (присвоювання)

+ (об’єднання множин)

\* (перетин множин)

– (різниця множин).

## 3.5 Контрольні питання

1. Дайте визначення терміну “перевантаження”
2. Що можна перевантажувати в С++?
3. Сформулюйте базове правило перевантаження
4. Поясніть, чому перевантаження взагалі можливе?
5. Скільки операторів можна перевантажувати в С++?
6. Які оператори не можна перевантажувати і чому?
7. Які оператори можна, перевантажувати
8. Назвіть 3 способи перевантаження операторів
9. Як можна змінити початкову кількість операндів при перевантаженні
10. Що таке static? Варіанти використання
11. Різниця між overloading (перевантаженням) і overriding (перевизначенням)?
12. Що таке вказівник this?
13. Який спосіб перевантаження можна використовувати для операторів присвоєння (=), індексу ([]), виклику функції (()) бо вибору члена (->) використовуйте
14. Які оператори не можуть бути перевантажені через методи класу?
15. Перевантаження унарних та бінарних операцій. В чому різниця?

# Лабораторна робота №4 Перевантаження операторів потокового введення-виведення

**Мета роботи:** ознайомитись із принципами перевантаження операторів потокового введення-виведення.

## 4.1 Теоретичні відомості

**Введення-виведення стандартних типів**

Для виведення стандартних типів у С++ визначено клас ostream з оператором <<.

**class** ostream {

// ...

**public**:

ostream& **operator**<<(char\*);

ostream& **operator**<<(int i);

//…

};

cout, cerr, clog - об’єкти класу ostream\_withassign, що успадковує ostream. Тому виведення здійснюється за допомогою конструкцій

cout<<“Hi”;

cerr<<“x=”<<x<<“\n”;

Аналогічно для введення стандартних типів у С++ визначено клас istream з оператором >>.

**class** istream {

// ...

**public**:

istream& **operator**>>(char\*);

istream& **operator**>>(char&);

istream& **operator**>>(short&);

istream& **operator**>>(int&);

istream& **operator**>>(long&);

istream& **operator**>>(float&);

istream& **operator**>>(double&);

// ...

};

cin-об’єкт класу istream\_withassign, що успадковує istream. Тому введення здійснюється за допомогою конструкції

cin>>c;

**Введення-виведення типів, визначених користувачем**

Для виведення типів, визначених користувачем, треба перевантажити оператор << для даного типу. Наведемо приклад для типу complex:

ostream& **operator**<<(ostream& s,complex& a){

s << ”(” << a.re << ”,” << a.im << ”)”;

**return**(s);

}

Зазначимо, що посилання на об’єкт ане обов’язкове, оскільки оператор не змінює цей об’єкт. Але з метою збереження пам’яті його варто робити. Тому декларація оператора може виглядати так:

ostream& operator<<(ostream& s,const complex& a)

Для введення типів, визначених користувачем, потрібно перевантажити оператор >> для цього типу. Для типу complex:

istream& **operator**>>(istream& s, complex& a)

{

**double** re=0, im=0;

**char** c=0;

s>>c;

**if** (c=='(')

{

s>>re>>c;

**if** (c==',') s>>im>>c;

}

**if** (c==')') a=complex(re,im);

**return** s;

}

У цьому випадку посилання на об’єкт а обов’язкове, оскільки оператор змінює цей об’єкт.

Зазначимо, що посилання на ostream та istream, що повертають функції, потрібні для збереження транзитивності операторів << та >>, тобто можливості використовувати конструкції вигляду:

cin>>c1>>c2;

cout<<”x=”<<x;

Визначені таким чином оператори << та >> не можна зробити функціями-членами класу complex, оскільки їх перші параметри - це об’єкти класу ostream та istream відповідно (що неможливо у разі перевантаження операторів). Тому оператори << та >> можуть бути лише функціями-друзями класу complex:

**class** complex {

//…

**public**:

//...

**friend** ostream& **operator**<<(ostream& s,const complex& a)

**friend** istream& **operator**>>(istream& s,complex& a)

};

Перевантажені оператори введення-виведення викликаються так:

**int** main()

{

complex a;

//…

cin>>a;

cout<<a;

//…

}

## 4.2 Приклад виконання

#**include** <iostream>

#**include** <math.h>

#**include** <string>

**using namespace** std;

**class** Point {

**protected**:

**int** x, y;

**public**:

Point();

Point(**int** k, **int** g);

Point(**const** Point& p);

//...

**friend** istream& **operator**>>(istream& s, Point& a);

**friend** ostream& **operator**<<(ostream& s, **const** Point& a);

};

//...

ostream& **operator**<<(ostream& s, **const** Point& a)

{

s << "Point(" << a.x << "," << a.y << ")\n";

**return** s;

}

istream& operator>>(istream& s, Point& a)

{

cout << "enter point\n";

s >> a.x >> a.y;

**return** s;

}

**int** main()

{//...

Point a, b, c, d;

cin >> a;

cin >> b;

c = a + b;

d = a - b;

cout << c;

cout << d;

//...

}

## 4.3 Порядок виконання роботи

Перевантажити оператори потокового введення-виведення (>>,<<) для класу, спроектованого у лабораторній роботі 2 згідно зі своїм варіантом. Введення виконувати в форматі, зручному для подання відповідної структури даних. При цьому реалізувати обробку помилкового введення і встановлення відповідного стану потоку. В основній програмі продемонструвати роботу цих операторів.

## 4.4 Варіанти завдань

Номери варіантів визначають згідно з варіантами лабораторних робіт 2, 3.

## 4.5 Контрольні питання

1. Які оператори використовують для потокового вводу-виводу?
2. Як впливає пріоритет виконання операцій на їх роботу?
3. Які є особливості при організації їх перевантаження?
4. Чому обрано саме ці оператори?
5. Що таке friend функції ( оператори ), як працюють, чим відрізняються від методів класів і звичайних функцій?

# Лабораторна робота № 5 Параметризовані класи (шаблони)

**Мета роботи:** ознайомитись з принципами створення шаблонів

## 5.1. Теоретичні відомості

***Шаблон (template)*** - це механізм параметризації визначення класу чи функції довільним типом. Шаблони дозволяють застосовувати простий спосіб введення різного роду загальних концепцій та методів їх спільного використання. Результуючий код за ефективністю не поступається написаному вручну спеціалізованому коду. Клас, що утворюється із шаблону класу у разі конкретного завдання значення параметра, є цілком нормальним класом, тому шаблон класу називають ***генератором типів***. Такий процес генерації оголошення класу чи функції за шаблоном та аргументом шаблону називають ***інстанціюванням***. Якщо має місце генерація різних версій шаблону для різних значень параметра, то кожну з цих версій називають ***спеціалізацією*.**

Стандартна бібліотека С++ містить класи-контейнери, реалізовані у формі шаблонів. Ось деякі з них:

vector одновимірний масив елементів

list двозв’язний список елементів

deque черга елементів з двома кінцями

stack стек елементів

map асоціативний масив елементів

set множина елементів

Шаблон може мати кілька параметрів, які можуть бути: класами, стандартними типами, шаблонами; аргумент шаблону може бути константним виразом, адресою об’єкта чи функції; не може бути рядковим літералом. Допускаються значення аргументів шаблонів за замовчуванням, синтаксис та правила використання аналогічні з аргументами функцій за замовчуванням. Одна спеціалізація вважається більш спеціалізованою за іншу, якщо список аргументів першої спеціалізації відповідає другій, а не навпаки. Генеруючи типи, перевагу віддають більш спеціалізованим версіям:

**template**<**class** T> **class** Vector// шаблон

{

T\* v;

**int** size;

**public**:

Vector();

Vector(**int**);

//...

};

Наведемо приклад створення об’єкта параметризованого класу:

**int** main()

{ //...

Vector<**int**> vi;//інстанціювання

Vector<**char**\*> vpc;

Vector<**double**> vd;

//...

}

**template**<**class** T T default\_value=0> **class** Vector<T\*>

{ /\*…\*/};

//часткова спеціалізація Vector для значень параметра,

//поданих у вигляді T\*

Vector<**int**\*\*> vppi;// тут T - **int**\*

**template**<> **class** Vector<**void**\*>{/\*…\*/};

//повна спеціалізація без указання параметра шаблону

Vector<**void**\*> vpv;

Іноді параметр шаблону функції може бути виведений з типів її аргументів під час виклику, тому його не потрібно вказувати під час інстанціювання:

**template**<**class** T> **void** sort(Vector<T>& v) {/\*…\*/}//шаблон функції

sort(vi);//vi типу Vector<**int**>, не потрібно писати sort<**int**>

Шаблони функцій можна перевантажувати звичайними методами. Розв’язуючи відповідності, перевагу віддають більш спеціалізованим шаблонам:

**template**<**class** T> T sqrt(T);

**template**<**class** T> complex<T> sqrt(complex<T>);

**double** sqrt(**double**);

**void** f(complex<**double**> z)

{

sqrt(2); //sqrt<int>(int)

sqrt(2.0);//sqrt(double)

sqrt(z);//sqrt<double>(complex<double>)

}

Шаблони можна ефективно використовувати для успадкування. Методику абстракції за допомогою шаблонів називають ***параметричним поліморфізмом*** або ***поліморфізмом часу компіляції*** на відміну від ***поліморфізму часу виконання***, що забезпечується віртуальними функціями.

Можна створювати шаблони класів, що походять як від класу, що не є шаблоном, так і від шаблону класу. Класи чи шаблони класів можуть мати члени-шаблони. Члени-шаблони не можуть бути віртуальними.

## 5.2 Приклад виконання

#**include** <iostream>

#**include** <math.h>

#**include** <string>

**using namespace** std;

**template** <**class** T>

**class** Point

{

**protected**:

T x, y;

**public**:

Point();

Point(T k, T g);

Point(**const** Point<T>& p);

//...

**void** setx(T t) { x = t; }

**void** sety(T t) { y = t; }

**void** plus(**const** Point<T>& p);

**void** displayin();

**virtual void** displayout();

};

**template** <**class** T> Point<T>::Point()

{

x = 5;

y = 5;

}

**template** <**class** T> Point<T>::Point(T k, T g)

{

x = k;

y = g;

}

**template** <**class** T> Point<T>::Point(**const** Point<T>& p)

{

x = p.x;

y = p.y;

}

**template** <**class** T> **void** Point<T>::plus(**const** Point<T>& p)

{

x = x + p.x;

y = y + p.y;

}

**template** <**class** T> **void** Point<T>::displayin()

{

cout << "enter point\n";

cin >> x >> y;

}

**template** <**class** T> **void** Point<T>::displayout()

{

cout << "Point(" << x << "," << y << ")\n";

}

**int** main()

{

**int** x, y;

Point<**float**> a;

a.displayin();

a.displayout();

cout << "copy constructor test: ";

Point<**float**> b(a);

b.displayout();

cout << "enter x,y:\n";

cin >> x >> y;

a.setx(x);

a.sety(y);

cout << "setter test: ";

a.displayout();

cout << "parameter constructor test: ";

Point<float> c(x, y);

c.displayout();

cout << "adding points result: ";

a.plus(b);

a.displayout();

//...

}

## 5.3 Порядок виконання роботи

Спроектувати шаблон для класу, створеного у лабораторній роботі 2, згідно зі своїм варіантом. В основній програмі створити відповідні структури простих (наприклад, список чисел або рядків) і складних даних (наприклад, список об’єктів іншого класу) і продемонструвати роботу з ними.

## 5.4 Варіанти завдань

Номери варіантів визначають згідно з варіантами лабораторних робіт 2, 3.

## 5.5 Контрольні питання

1. Що таке шаблони (template)?
2. З якою метою використовуються шаблони?
3. Що таке інсталювання, спеціалізація?
4. Що таке шаблони функцій?
5. Як оформлюються шаблони функцій?
6. У чому переваги і недоліки класів з шаблонами?

# 

# Лабораторна робота №6 Програмне середовище Qt Creator

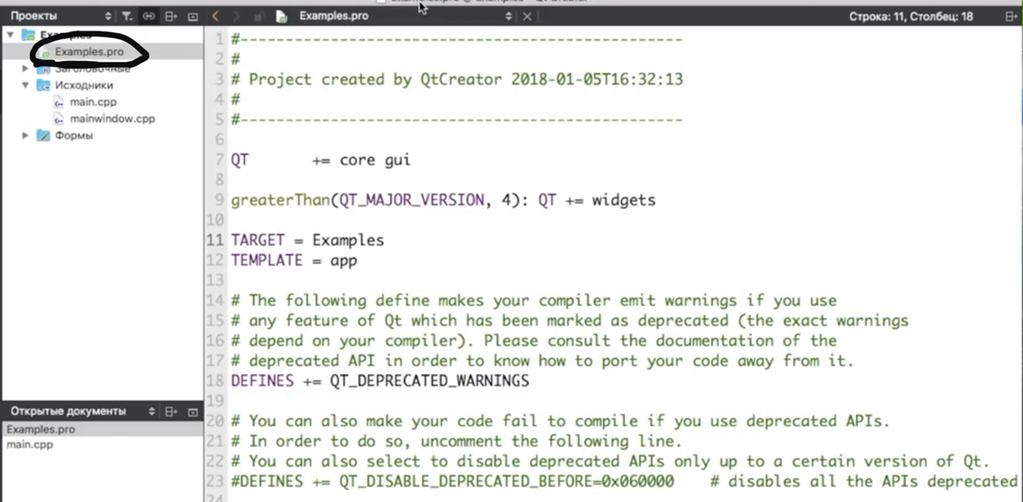
**Мета роботи**: освоїти принципи візуального програмування, вивчити основні функціональні можливості Qt Creator, навчитися працювати з головними візуальними компонентами Qt Creator.

## 6.1 Теоретичні відомості

**Файл-Examples.pro**

Після запуску програми Qt Creator виявимо велику кількість різних файлів, одним із них є файл-***Examples.pro*** (назва проєкту з розширенням ***.pro***).

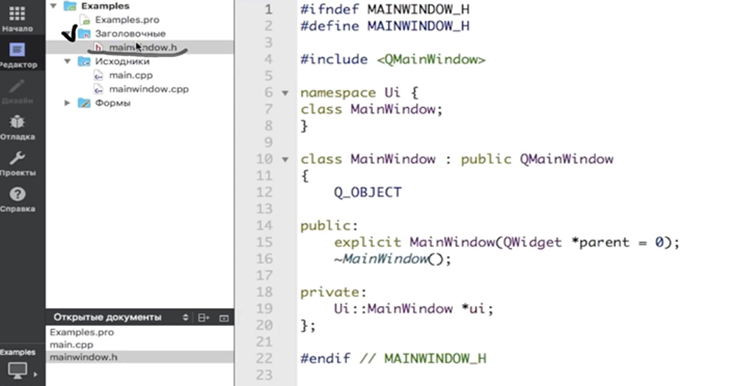
В файлі **Examples.pro** (Рис. 6.1) розташовані можливі налаштування, які дозволяють програмі працювати, також тут знаходяться посилання на ті файли, які наявні в нашій програмі. Наприклад: ***main.cpp***, mainwindow, заголовні файли, які відповідають за графічний інтерфейс, (наприклад: ***mainwindow.ua***).



**Рис. 6.1 - Файл проекту**

**Папка «Заголовочные»**

В даному каталозі (Рис. 6.2) описано заголовки файлів бібліотек нашої програми, включаючи класи, прототипи методів, які зможемо надалі реалізувати.



**Рис. 6.2 - Заголовочний файл**

**Папка «Исходники»**

В цій папці (Рис. 6.3) розміщені файли основного коду проєкту. Тут головним файлом є ***main.cpp***. Тут міститься функція ***main()***, що є точкою входу програми. Саме з неї розпочинається виконання нашого додатку.

***Mainwindow.cpp***  - це основне вікно нашого додатку.

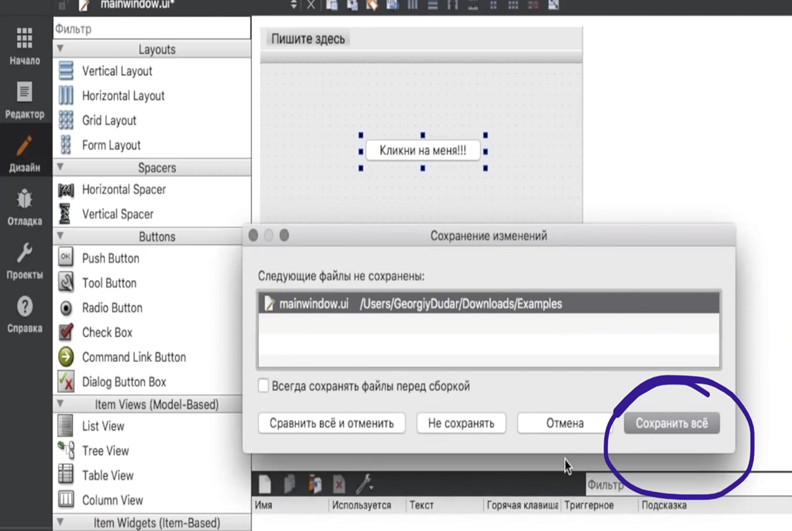
**Папка «Формы»**

Папка «Формы» пов’язана з графічним редактором



Рис. 6.3 - Файл форми

Наприклад, якщо відкрити ***mainwindow.ua***, то форма головного вікна відкриється в графічному редакторі, куди можна додавати різні елементи інтерфейсу. Створивши необхідну форму, її необхідно зберегти (Рис. 6.4).

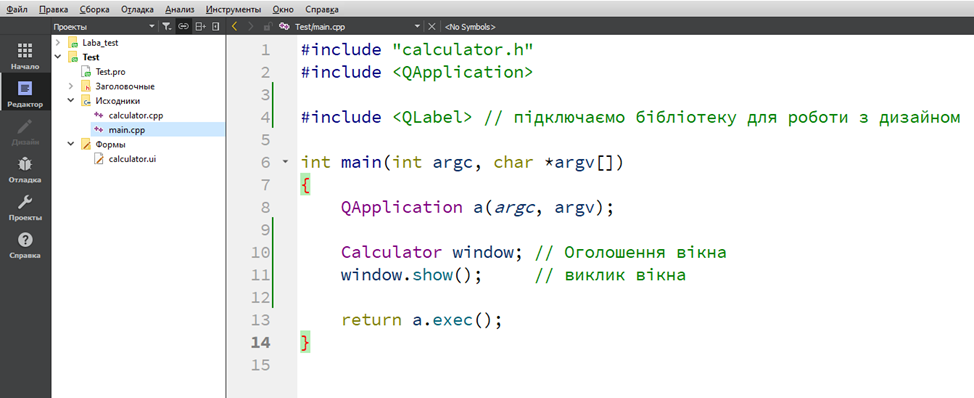


**Рис. 6.4 - Збереження форми**

Відеокурс по Qt Creator - <https://www.youtube.com/watch?v=g6fw5n9Gt-E>.

## 6.2 Приклад виконання

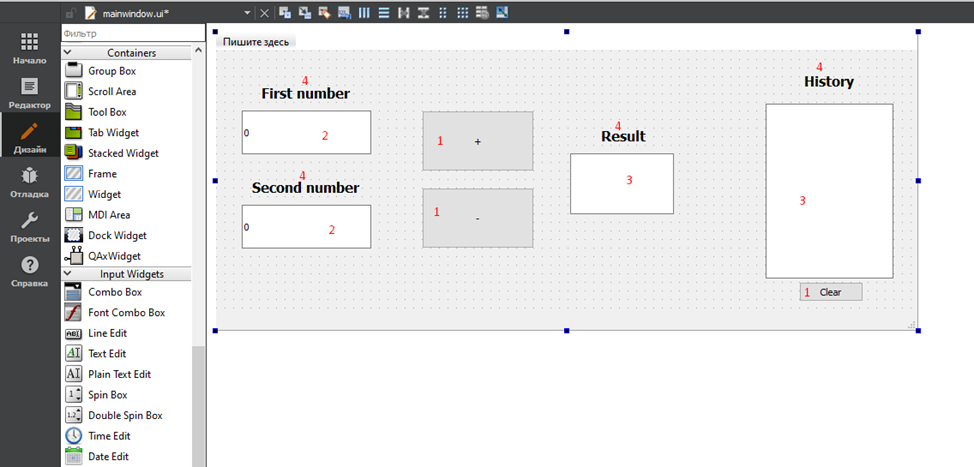
Після створення об’єкта, підключаємо бібліотеку **<QLabel>** та створюємо об’єкт головного класу (в нашому випадку calculator). Використовуємо до нього функцію **show(),** яка викличе графічне зображення об’єкта (Рис. 6.5).



**Рис. 6.5 - Головний файл проету**

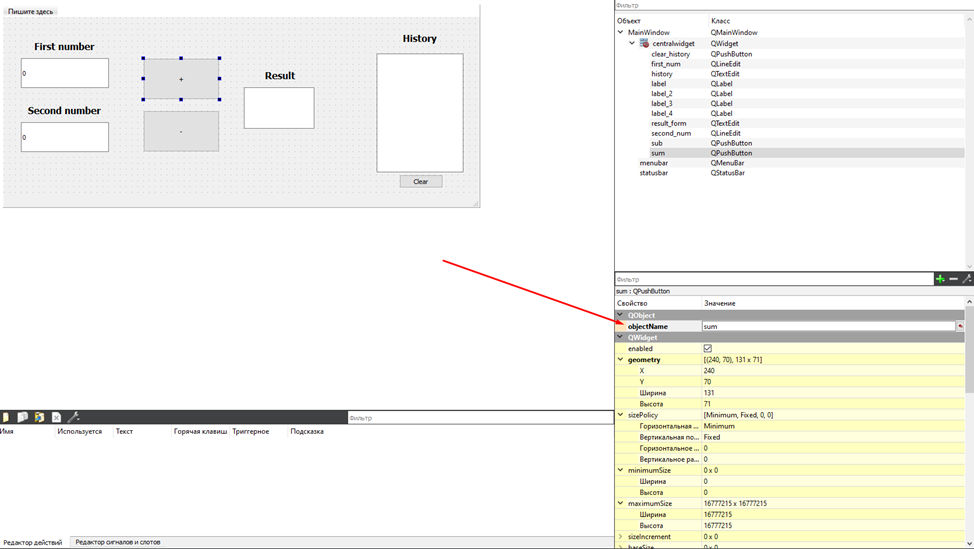
Для створення та редагування дизайну відкриємо файл з розширенням \*.ui (у нашому випадку формы/calculator.ui).

Створимо декілька кнопок та поля для введення і виведення даних (Рис. 6.6). Для цього виберемо зліва елементи: 1 ***Push Button***(кнопка), 2 ***Line Edit*** ( форма для введення), 3 ***Text Edit***(форма для виведення), 4 ***Label***(текст).



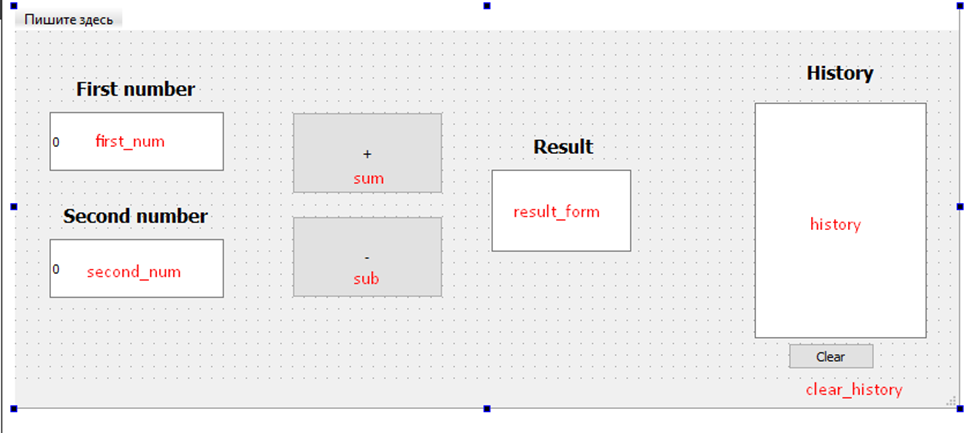
**Рис. 6.6 - Візуальний конструктор форм**

Натиснувши на елемент можна дізнатися та змінювати його властивості (Рис. 6.7).



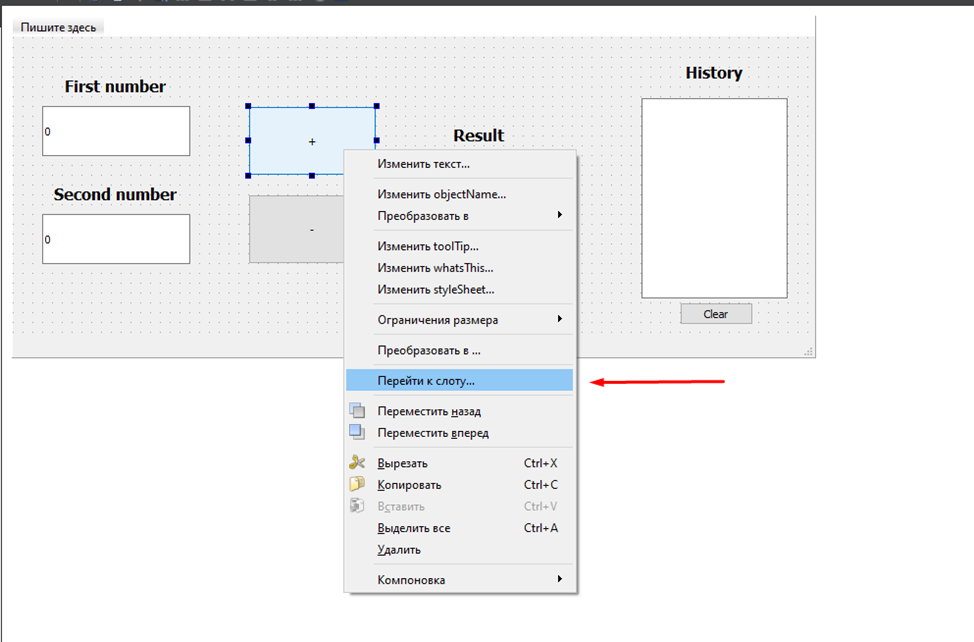
**Рис. 6.7 - Інспектор властивостей форми**

Головне дізнатися назву елемента, змінимо її для подальшого використання. Для цього двічі натиснемо на “objectName” та напишемо свою назву. Червоним написано назви об’єктів (Рис. 6.8).



**Рис. 6.8 - Форма з елементами**

Щоб запрограмувати кнопку на дію, потрібно натиснути на неї правою кнопкою миші та вибрати “Перейти до слоту” (Рис. 6.9).



**Рис. 6.9 - Підключення реакції на натискання кнопки**

Відкриється вікно вибору події.

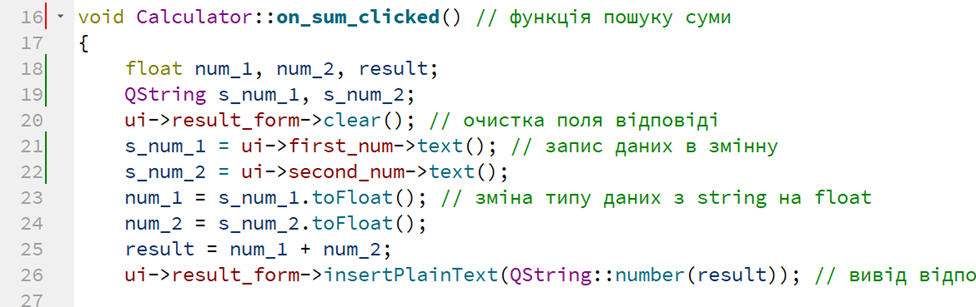
Виберемо “натиснути” - ***clicked().*** У файлі “calculator.cpp” автоматично створиться функція ***on\_sum\_clicked()***(Рис. 6.10)***.***



**Рис. 6.10 - Метод для обробки клика**

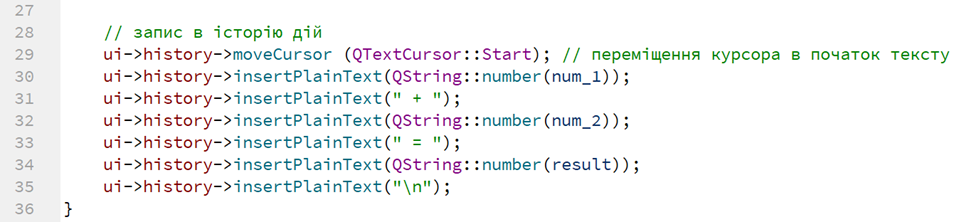
Оголосимо три змінні типу float num\_1, num\_2, result та ще дві змінні типу «QString» s\_num\_1, s\_num\_2 для запису введених даних. Для s\_num\_1, s\_num\_2 присвоїмо значення завдяки конструкції: ***ui->nameObject->text().*** Зверніть увагу що ми присвоюємо дані типу QString, тому потрібно перетворити їх в тип float через функцію toFloat()(Рис. 6.11).

Щоб вивести результат, потрібно знову перетворити тип в QString та використати функцію ***ui->nameObject->insertPlainText(“текст”).*** Використовуємо функцію ***ui->nameObject->clear()*** щоб очистити поле відповіді.



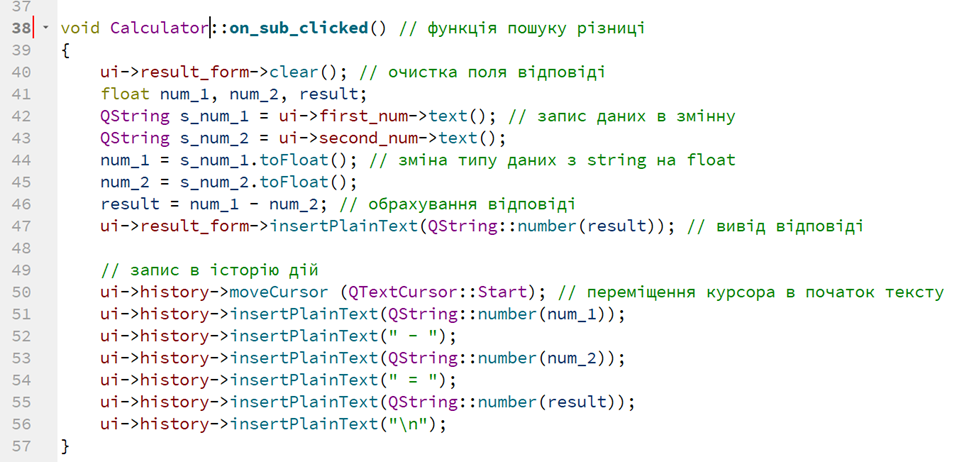
**Рис. 6.11 - Приклад обробки “кліка”**

Допишемо вивід операції в історію (Рис. 6.12).



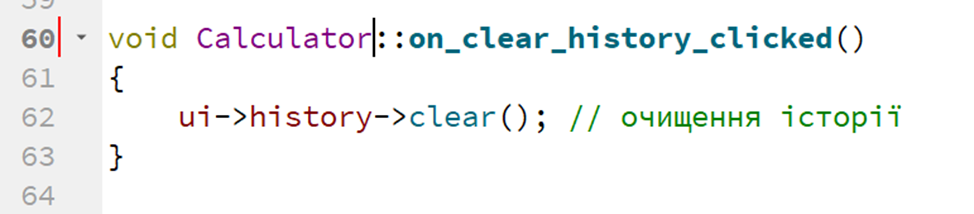
**Рис. 6.12 - Код занесення тексту у компонент**

Створимо аналогічну функцію для пошуку різниці (Рис. 6.13).



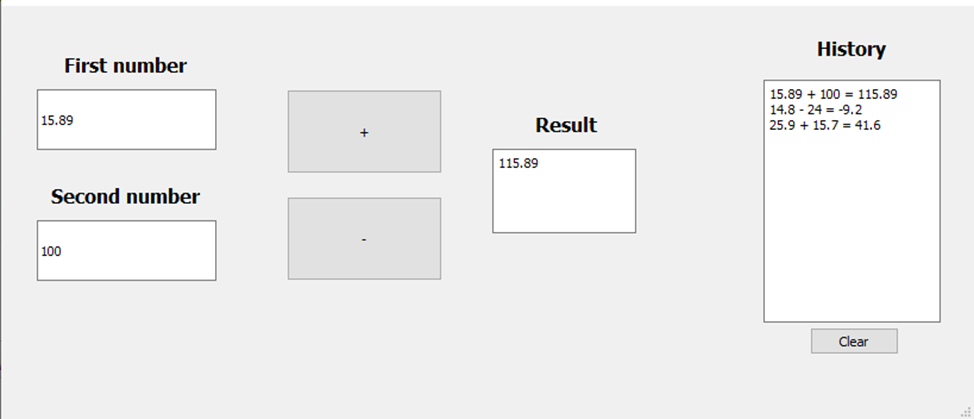
**Рис. 6.13 - Приклад обробки “кліка” кнопки “-”**

Створимо функцію для очищення історі (Рис. 6.14) ї.



**Рис. 6.14 - Обробки кнопки “очищення”**

Тепер запустимо нашу програму та протестуємо (Рис. 6.15).



**Рис. 6.15 - Готовий запущений додаток**

## 6.3 Порядок виконання роботи

1. Створити калькулятор за наведеним прикладом (для ознайомлення з Qt Creator).
2. Створити власний проект згідно з номером варіанта.

## 6.4 Варіанти завдань

1. Створити проект, який буде мати дві форми. З використанням компонентів Edit i Button створити першу форму, в яку потрібно буде ввести довжини трьох сторін трикутника. У другій формі відобразити значення площі та периметра трикутника.
2. Створити проект, який буде мати дві форми. З використанням компонентів Edit i Button створити форму, в яку потрібно буде ввести значення радіуса кола. У другій формі відобразити значення площі круга та довжини кола.
3. Створити примітивний варіант гри “Перший мільйон” (“О, щасливчик!”). Використати компоненти Button та Label для відображення питань та відповідей. Обмежитись одним запитанням, в разі відповіді вивести результат (Так чи Ні) в іншій формі.
4. Створити проект, який буде складатись з двох форм. На першій формі повинна бути лише кнопка виклику другої форми, причому після виклику другої форми перша повинна зникати. На другій формі повинна бути кнопка, яка буде з’являтись у різних місцях форми випадково з кожним натисканням на ній миші.
5. Створити примітивний текстовий редактор, в якому будуть функції очистки всього тексту, вставка дати та часу, зміни кольору фону і тексту.
6. Створити калькулятор, подібний до наведеного прикладу, який буде обчислювати sin(x), cos(x), tg(x), ctg(x).
7. Створити проект, який буде складатись з двох форм. На першій формі створити декілька полів введення назв міст, а навпроти них - кількість населення цих міст. На другій формі після її відображення показати місто з найбільшою кількістю населення.
8. Створити проект, який буде складатись з двох форм. На першій формі створити декілька полів для введення імен дівчат і хлопців, а навпроти них - їхній вік. На другій формі після її відображення показати імена хлопців та дівчат однакового віку.
9. Створити секундомір. На формі розмістити кнопки вмикання і вимикання секундоміра. Ще розмістити кнопку, що викликає другу форму, на якій відображається час на момент натискання.
10. Створити проект, в якому з головної форми будуть викликатися ще дві форми. А за допомогою компонента Button з першої форми можна буде змінювати розміри двох інших, а також їх видимість, позиції на екрані.
11. Створити проект, в якому за допомогою кнопок можна буде керувати типом фігури та її кольором, а також виводити на окрему форму площу фігури. Примітка.
12. Створити текстовий редактор. Реалізувати функції завантаження та зберігання файлу, можливість зміни розміру та назви шрифту.
13. Створити форму для перегляду графічних файлів.
14. Створити програму для прослуховування звукових файлів.
15. Створити програму для перегляду відео-файлів.
16. Завдання підвищеної складності:
17. Створити програму, яка реалізує виведення графіка функції: y = A\*sin(B\*x)-B. Графік обов’язково повинен бути відцентрованим, тобто проходити не по системі координат форми, а у звичному для людини вигляді.
18. Створити програму, яка реалізує виведення графіка функції: y = A\*x^B + C. Графік обов’язково повинен бути відцентрованим, тобто проходити не по системі координат форми, а у звичному для людини вигляді.
19. Створити програму, яка реалізує виведення графіка функції: y = A\*ln(x)/ln(B)+C. Графік обов’язково повинен бути відцентрованим, тобто проходити не по системі координат форми, а у звичному для людини вигляді.
20. Створити форму, у якій було б реалізовано шифрування та дешифрування тексту за методом Цезаря. Користувач повинен сам вирішувати, яким ключем потрібно шифрувати інформацію. Шифр Цезаря реалізується заміною однієї літери іншою, яка стоїть на деяке значення попереду. Наприклад, шифруючи слово “Secret” зсувом літери на одну вперед за порядком, отримаємо “Tfdsfu”.
21. Створити форму, у якій було б реалізовано головне меню. Через це меню обов’язково має виконуватись кілька різних операцій. Наприклад, закриття додатка, виведення привітання, виведення інформації про творця програми, виведення дати, часу тощо.
22. Створити графічний редактор з простими можливостями: малювання прямих, кривих, та стандартних геометричних фігур
23. Створити діалог введення пароля. Реалізувати шифрування та дешифрування пароля та виведення його у зашифрованому (та дешифрованому) вигляді. Метод шифрування - будь-який.
24. Створити програму-браузер на прикладі Провідника Windows. Реалізувати лише перегляд списку файлів на носії інформації, наявному в системі, та запуску цих файлів.

## 6.5 Контрольні питання

1. Що дозволяє робити Qt Creator?
2. Для яких ОС можна використовувати Qt Creator?
3. Яка мова програмування використовується у Qt Creator?
4. Яка бібліотека візуальних компонентів використовується у Qt Creator?
5. Яка парадигма програмування використовується у Qt Creator?
6. У чому переваги і недоліки візуального програмування?
7. З яких “об’єктів” (сутностей) складається Qt Creator програма?

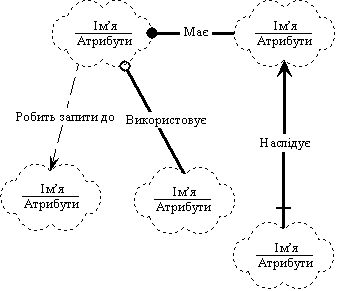
# 

# Лабораторна робота № 7 Використання об’єктно-орієнтованого підходу для розробки програмного забезпечення

**Мета роботи:** засвоїти принципи об’єктно-орієнтованого проектування, включаючи аналіз предметної області та побудову фізичних і логічних моделей

## 7.1 Теоретичні відомості

**Система позначень Буча.** Для фізичного чи логічного проектування предметної області програмного додатка прийнято використовувати систему позначень Буча (Рис. 7.1).

****

**Рис. 7.1 - Статична діаграма класів у нотації Буча**

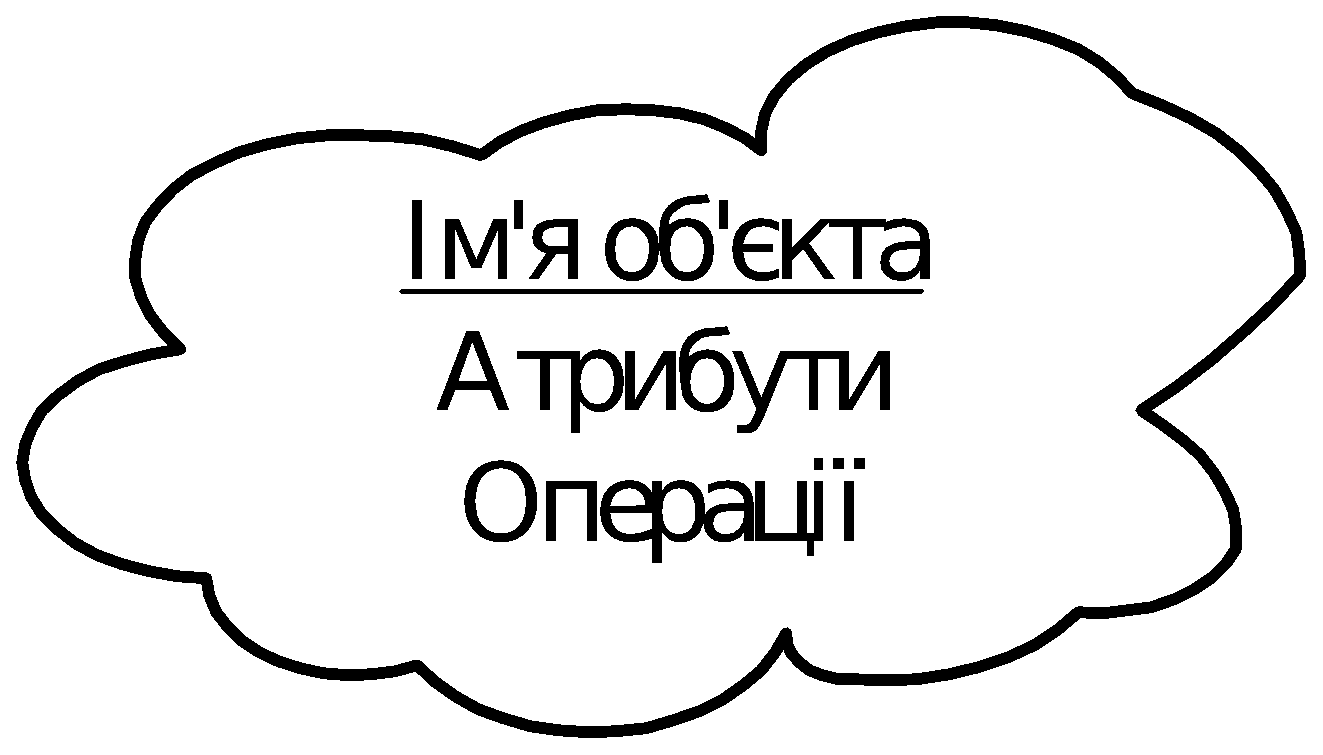
Для побудови цих діаграм використовуються такі основні елементи:

1. Ілюстрація класу (Рис. 7.2).

****

**Рис. 7.2 - Зображення класу**

2. Ілюстрація об’єкта (Рис. 7.3).

**Рис. 7.3 - Зображення об’єкта**

3. Спеціалізація класів:

- абстрактний клас (наприклад, клас тварин тощо) (Рис. 7.4).

****

**Рис. 7.4 - Зображення абстрактного класу**

- дружній клас (такий клас, що має доступ до закритих членів інших класів) (Рис. 7.5).

****

**Рис. 7.5 - Зображення дружнього класу**

- статичний клас, що забезпечує даними програмний додаток (Рис. 7.6).

****

**Рис. 7.6 - Зображення статичного класу**

- родовий базовий клас - найбільш загальний клас у системі (Рис. 7.7).

****

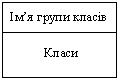
**Рис. 7.7 - Зображення родового класу**

4. Метаклас - такий клас, екземплярами якого є класи (Рис. 7.8).

****

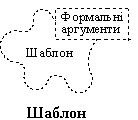
**Рис. 7.8 - Зображення метакласу**

1. Категорії однакових класів (Рис. 7.9).



**Рис. 7.9 - Зображення категорії однакових класів**

6. Шаблони класів (Рис. 7.10).

****

б)

a)

**Рис. 7.10 - Зображення шаблону і класу:**

**а) шаблон; б) клас, реалізований за шаблоном**

7. Утиліти класів зображують функції та процедури - не об’єкти класів; їх використовують для розробки програмного додатка з використанням принципів об’єктно-орієнтованого проектування (Рис. 7.11).



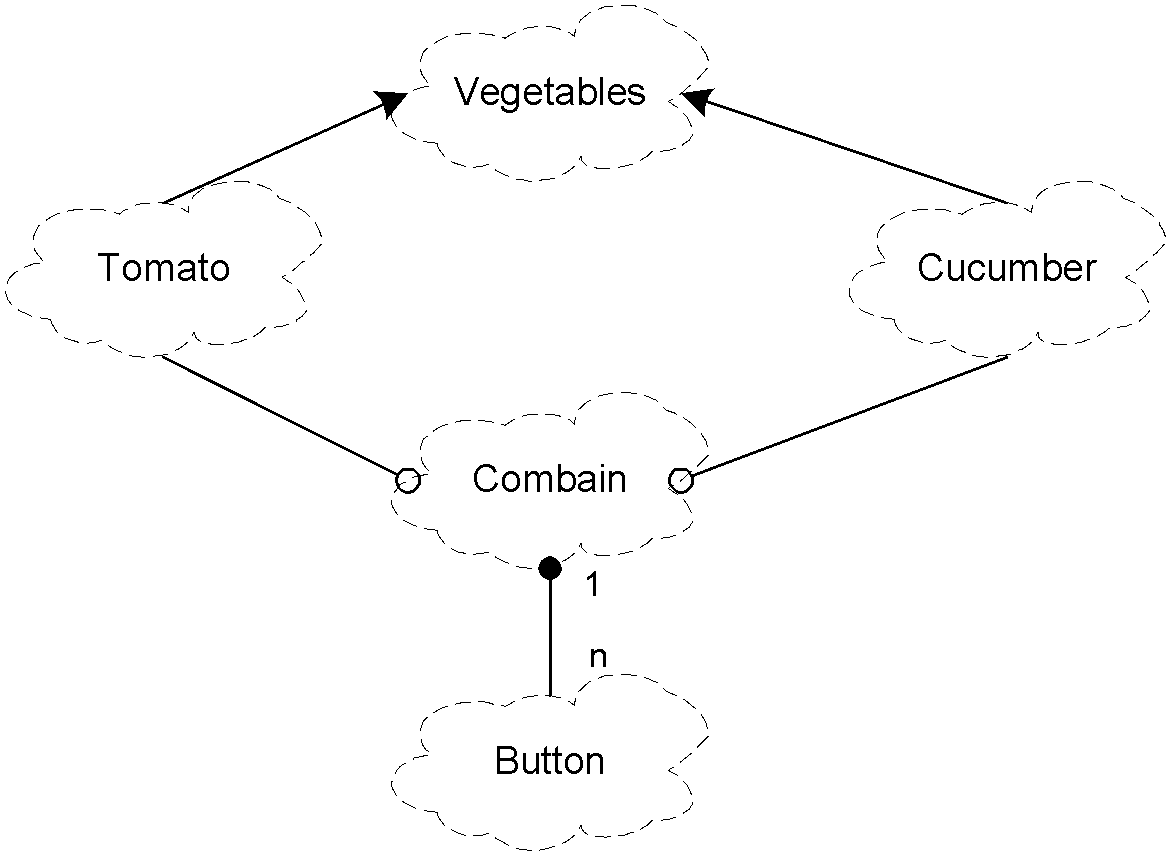
**Рис. 7.11 - Зображення утиліти класу**

Таблиця 7.1 - Умовні позначення відношень на схемах

|  |  |
| --- | --- |
|  | Проста агрегація |
|  | Агрегація за значенням |
|  | Агрегація за посиланням |
|  | Використання |
|  | Успадкування  - однакові типи |
|  | Успадкування  - нові типи |
|  | Асоціація |

## 7.2 Приклад виконання

Приклад діаграми класів у нотації Буча для моделі кухонного комбайна показано на Рис. 7.12.



**Рис. 7.12 – Діаграма класів у нотації Буча для простої моделі кухонного комбайна**

## 7.3 Порядок виконання роботи

1. Виконати аналіз предметної області.
2. Виконати об’єктну декомпозицію складного об’єкта відповідно до варіанта.
3. Побудувати фізичну та логічну моделі з використанням системи позначень Буча або UML.
4. Обговорити та затвердити розроблені моделі у викладача[[1]](#footnote-2).
5. Реалізувати програму об’єктно-орієнтованою мовою програмування С++.
6. Оформити пояснювальну записку до комп’ютерного практикуму №7.
7. Захистити комп’ютерний практикум №7.

Кожний з проектів має містити в собі декілька класів, у тому числі один абстрактний та декілька агрегатних. Ці класи мають бути пов’язані різними відношеннями, які треба встановити під час аналізу предметної області. У процесі аналізу основну увагу слід приділяти коректності відношень між класами, що відповідають конкретним предметам чи об’єктам реального світу.

**Оформлення комп’ютерного практикуму:**

1. Роздільна компіляція.
2. Титульна сторінка.
3. Діаграма Буча (UML-діаграма).
4. Пояснення всіх відношень між класами на діаграмі.
5. Код програми.
6. Пояснення до функцій (що в якості аргументів, що повертає функція).
7. Копії екранів (з допомогою кнопки PrintScreen) процесу виконання програми.

На оцінку "відмінно" повинні бути реалізовані такі відношення між класами: успадкування, агрегація, використання. На оцінку "добре" - відношення між класами успадкування та агрегація.

## 7.4 Варіанти завдань

1. Розробити проект пульта дистанційного керування телевізором.
2. Розробити проект пульта дистанційного керування розумним будинком.
3. Розробити проект керування телефоном.
4. Розробити проект керування холодильником.
5. Розробити проект роботи підіймального ліфта.
6. Розробити проект підтримки системи сигналізації.
7. Розробити проект турнікета метрополітену.
8. Розробити проект автомата з продажу цукерок.
9. Розробити проект керування дитячою іграшкою.
10. Розробити проект  системи продажу залізничних квитків.
11. Розробити проект системи керування супутником.
12. Розробити проект керування кухонним комбайном.
13. Розробити проект бортового комп’ютера автомобіля.
14. Розробити проект автопілота літака.
15. Розробити проект моделі бібліотеки.
16. Розробити проект аудіоцентру.
17. Розробити проект автомобіля.
18. Скласти систему класифікації геометричних фігур.
19. Скласти систему класифікації тварин.
20. Скласти модель взаємодії комп’ютера з периферійними пристроями (комп’ютер подати як складний пристрій).
21. Скласти модель котеджу та “взаємодії” з ним людини.
22. Змоделювати комп’ютерний клас.
23. Змоделювати роботу магазину з продажу запчастин.
24. Змоделювати відношення програмного забезпечення, встановленого на типових апаратних засобах.
25. Скласти модель *аеропорт - допоміжні технічні засоби - літаки - обслуговуючий персонал*.
26. Змоделювати структуру та функціонування метрополітену.
27. Змоделювати факсимільний апарат.
28. Змоделювати космічний апарат.
29. Змоделювати роботу пошти.
30. Скласти модель взаємодії людини з телефонним апаратом.

## 7.5 Контрольні питання

1. У чому полягає об'єктно-орієнтована парадигма програмування?
2. Порівняйте об'єктно-орієнтовану і процедурну парадигми програмування?
3. Що таке логічна модель?
4. Що таке фізична модель?
5. Що таке і для чого використовується діаграма класів?
6. У чому полягає проектування об'єктно-орієнтованої програми?
7. У чому полягає відношення агрегації?
8. У чому полягає відношення використання?
9. У чому полягає відношення успадкування?
10. У чому полягає відношення асоціації?

# Список використаної літератури

1. Вступ до алгоритмів. Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест і Кліфорд Стайн. Переклад з англійської (третього видання). - К.: К.І.С., 2019 - 1288 с., іл.
2. Програмування С++ в прикладах і задачах. Васильєв О. - К.: Ліра-К., 2019 - 382 с., іл.
3. Чиста архітектура. Роберт Сесіл Мартін. - К.: Фабула., 2019 - 368 с., іл.
4. С++. Основи програмування. Теорія та практика. О.Г. Трофименко - К.: Фенікс., 2010 - 554с., іл.
5. С++. Основи програмування. Теорія та практика : підручник / [О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.] ; за ред.О.Г.Трофименко. – Одеса: Фенікс, 2010. – 544 с.
6. Объектно-ориентированный анализ и проектирование с примерами приложений. 3-е издание Гради Буч, Роберт А. Максимчук, Майкл У. Энгл, Бобби Дж. Янг, Джим Коналлен, Келли А. Хьюстон. - М.: Вильямс., 2020 - 720 с., іл.
7. Об’єктно-орієнтоване програмування. Частина 1. Основи об’єктно-орієнтованого програмування на мові C#.: Навчальний посібник. / Д.В. Настенко, А. Б. Нестерко. – К.: НТУУ «КПІ», 2016. - 76с.
8. ТЕХНОЛОГІЇ ОБ'ЄКТНО-ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ: ЧАСТИНА І. КОМП’ЮТЕРНИЙ ПРАКТИКУМ [Електронний ресурс]: навч. посіб. для студ. спеціальності 151 – «Автоматизація та комп’ютерно-інтегровані технології» / КПІ ім. Ігоря Сікорського; уклад.: В. І. Бендюг, Б. М. Комариста. – Електронні текстові данні (1 файл: 2,84 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2018. – 225 с.
9. ДСТУ 3008-2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. – На заміну ДСТУ 3008–95; чинний з 2017–07–01. – Київ: ДП «УкрНДНЦ», 2016. – 26 с.
10. ДСТУ ГОСТ 7.1-2006. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : чинний з 2007-07-01. – К. : Держспоживстандарт України, 2007. – 47 с. (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).
11. Введення в дію нового стандарту з бібліографічного опису ДСТУ ГОСТ 7.1:2006. Основні відмінності від ГОСТ 7.1.-84 [Електронний ресурс] : нові правила бібліогр. опису / Кн. палата України. – Режим доступу : http://www.ukrbook.net/DSTU\_pabl.htm

1. Затвердження та обговорення моделей - це дуже серйозний момент, від якого залежить успішність та коректність (або **не**успішність та **не**коректність) усіх подальших дій групи розробників. Тому цьому виду діяльності слід приділяти багато уваги. Якщо на стадії здачі комп’ютерного практикуму №7 з’ясується, що розроблені моделі не були затверджені та коректно побудовані, весь додаток буде розглядатися як некоректна реалізація. [↑](#footnote-ref-2)