**Лабораторна робота №8.** **Використання в класах статичних членів та методів.**

**Мета:** Набуття навичок в розробці програм, де використовуються класи зі статичними членами і методами та об’єкти класів.

**Порядок виконання роботи**

1. Повторити синтаксис оголошення конструкторів.
2. Ознайомитися з теоретичними відомостями.
3. Продовжити працювати над програмою минулої лабораторної роботи. Створити для класу *Person* створити, використовуючи статичні члени-класу та статичні методи, лічильник кількості створених об’єктів та індикатор заповнення класу, який приймає значення; 0 – об’єкти відсутні, 1 – є дані по об’єктах, 2 – об’єкти відсортовані.
4. Створити позицію меню "The state of the objects of the Person class" (Стан об'єктів класу *Person*), де виводити інформацію про кількість об’єктів та стан їхньої обробки.
5. Надати коментарі до всіх членів класу та методів щодо їх призначення.
6. В першому рядку програми та заголовкового файлу повинні бути записаними в коментарі номер групи та прізвище, а також номер ЛР (через кому до попередньої).
7. Результати надсилати на електронну адресу викладача

[**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді заголовкового файлу та cpp-файлу з іменем у форматі

Для заголовкового файлу

**<Прізвище англійською>.h**

Наприклад, Ivanov.h

Для cpp-файлу:

**<Номер групи> <Номер лабораторної><Прізвище англійською>.cpp**

Наприклад, 31-01Ivanov.cpp.

Тему в заголовку листа записати

**ООП<Номер групи>-ЛР <Номер лабораторної>-<Прізвище>**

**Строк відсилки ЛР ІПЗ-31 11.10.2022**

Всі запитання, що виникнуть, надсилайте на електронну адресу викладача, тему в заголовку листа записати

**ООП<Номер групи>-Запитання-<Прізвище >**.

**Теоретичні відомості.**

**Константними функціями-членами класу**

Функції, які повертають константний об’єкт називаються константними функціями. Якщо такі функції оголошені в класі, то ці функції називаються константними функціями-членами класу.

Щоб оголосити функцію, яка повертає константний об’єкт, потрібно перед оголошенням функції розмістити ключове слово ‘const’

const returned\_type **FunName**(parameters)

{

// function's body

// ...

}

де

* returned\_type – тип, що повертається функцією;
* parameters – параметри функції;
* FunName – ім’я функції, що повертає константний об’єкт (константної функції).

Якщо функція оголошена в класі, то приблизне її оголошення буде таким:

class CMyClass

{

private:

// приватні члени та методи класу

// ...

public:

// загальнодоступні члени та методи класу

// ...

// оголошення константної функції в класі

const returned\_type **FunName**(parameters);

}

// реалізація функції FunName

const returned\_type CMyClass::FunName(parameters)

{

// тіло функції FunName()

// ...

}

##### Приклад оголошення класу, що містить константні (const) функції-члени

У прикладі оголошується клас CMyClass, що містить константну функцію GetX(), яка повертає константне значення типу double.

// клас, що містить оголошення константної функції

class CMyClass

{

double x; // внутрішня змінна

public:

CMyClass(void); // конструктор

void SetX(double nx);

const double GetX(void); // константна функція-член класу

};

Використання константної функції в деякому програмному коді:

// використання константної функції

CMyClass MC;

MC.SetX(25.08); // встановлення значення x

// використання константної функції GetX() класу CMyClass

double t1 = MC.GetX(); // t1 = 25.08 - ініціалізація змінної

const double t2 = MC.GetX(); // t2 = 25.08 - ініціалізація константи

##### Загальний вигляд функції-члена класу, що повертає об’єкт типу volatile

Об’єкт (змінна), що оголошений зі специфікатором volatile може бути змінений у програмі неявно без застосування явно заданих команд. Ключове слово volatile інформує компілятор, що значення змінної у програмі може бути змінене неявно (програмою обробки переривань, фоновим процесом, тощо). Знаючи про неявну мінливість змінної у програмі (ключове слово volatile), компілятор сформує код, що буде опитувати значення змінної перед кожним її використанням у програмі. Таким чином, буде сформовано реальне значення змінної (а не те, що було до зміни).

**Наприклад.** Нехай у деякій програмі P реалізована глобальна змінна X, що містить адресу, яка може бути змінена програмою-обробником переривання. Програма-обробник переривання викликається незалежно від виконання даної програми P і відповідним чином змінює значення цієї глобальної змінної X. Якщо у програмі P не вказати ключового слова volatile перед X, то, при зчитуванні значення X у програмі P може відображатись старе значення X. Якщо вказати слово volatile, то компілятор буде оновлювати значення змінної X при кожному звертанні до неї, і, спотворення результату не буде.

Функції-члени, які оголошені зі специфікатором volatile просто повертають непостійний (volatile) об’єкт. Такий об’єкт може бути присвоєний змінній, що оголошена з специфікатором volatile.

Загальний вигляд класу, що містить функцію, яка повертає volatile-значення

class CMyClass

{

private:

// приватні члени та методи класу

// ...

public:

// загальнодоступні члени та методи класу

// ...

// оголошення константної функції в класі

volatile returned\_type **FunName**(parameters);

}

// реалізація функції FunName

volatile returned\_type CMyClass::**FunName**(parameters)

{

// тіло функції FunName()

// ...

}

де

* returned\_type – тип, що повертається функцією;
* parameters – параметри функції;
* FunName – ім’я функції, що повертає volatile-об’єкт.

##### Приклад оголошення класу, що містить функції-члени volatile

У класі CMyVolatileClass оголошено функцію Get(), що повертає volatile-значення типу double.

// клас, що містить volatile-функцію

class CMyVolatileClass

{

int d;

public:

CMyVolatileClass(void);

// функції-члени класу

// звичайна функція-член класу

void Set(int nd) { d = nd; }

// функція-член класу, що повертає volatile-значення

volatile int Get(void);

};

Демонстрація використання класу в деякому методі чи програмному коді:

// клас, що містить volatile-функцію

CMyVolatileClass VC;

VC.Set(33); // виклик звичайної функції

// виклик volatile-функції для звичайної змінної

int t1 = VC.Get(); // t1 = 33

volatile int t2;

// виклик volatile-функції для volatile-змінної

t2 = VC.Get(); // t2 = 33

Як видно з вищенаведеного коду, volatile-функції можна використовувати у поєднанні зі звичайними та volatile-змінними.

##### Загальна форма оголошення функції з константним покажчиком this

Оголошення функції у класі (на прикладі функцій MyFun1(), MyFun2()) з константним покажчиком this має такий загальний вигляд:

class CMyClass

{

private:

// приховані члени даних та методи класу

// ...

public:

// функція оголошена з константним покажчиком this

return\_type MyFun1(parameters) const; // out-of-line

return\_type MyFun2(parameters) const // inline реалізація

{

// тіло функції

// ...

}

}

// реалізація out-of-line функції

return\_type CMyClass::MyFun1(parameters) const

{

// тіло функції MyFun1()

// ...

}

де

* return\_type – тип, що повертається тією чи іншою функцією;
* parameters – параметри, які отримує функція.

У вищенаведеному фрагменті, у класі CMyClass оголошуються дві функції з константним покажчиком this. Функція MyFun1() реалізована за межами класу (типу out-of-line). Функція MyFun2() реалізована в класі (типу inline). Перед тілом кожної функції стоїть ключове слово const.

##### Суть оголошення функції з константним покажчиком this

У функції з константним покажчиком this перед тілом функції ставиться ключове слово const. Це означає, що тіло функції має обмежені можливості використання.

Слід зауважити, що у цьому випадку ключове слово const не має відношення до значення, що повертається функцією. Воно має значення тільки для покажчика this, що використовується у функції.

Якщо функція оголошена з константним покажчиком this, то в тілі функції заборонено змінювати дані класу. При спробі змінити дані класу, буде виникати помилка компіляції.

##### Приклад, що демонструє відмінність між функціями з константним покажчиком this та без нього

Відмінність між звичайною функцією та функцією з константним покажчиком this добре видно в реалізації нижченаведеного класу.

Нехай задано клас CRadius, що реалізує радіус деякого об’єкта чи геометричної фігури. У класі реалізовано:

* внутрішній член даних типу double;
* конструктор CRadius();
* звичайну функцію GetRadius(), яка збільшує радіус удвічі і повертає його значення;
* функцію GetRadius2(), оголошену з константним покажчиком this. Дана функція повертає значення внутрішньої змінної radius.

// клас CRadius

class CRadius

{

double radius;

public:

CRadius(void); // конструктор класу за замовчуванням

// Звичайна функція-член класу,

// цій функції неявно передається покажчик: CRadius \* const this

double GetRadius(void)

{

radius \*= 2.0; // дозволено

return radius;

}

// Функція-член класу, оголошена з const

// цій функції передається покажчик: const CRadius \* const this

double GetRadius2(void) const

{

//radius = 2.0; // Заборонено! Помилка компіляції

return radius;

}

};

Якщо у функції GetRadius2() спробувати змінити значення внутрішньої змінної radius, то вийде помилка компіляції:

**Error: 'radius' cannot be modified because it is being accessed through a const object**

Ключове слово const перед тілом функції в класі означає, що функції заборонено вносити будь-які зміни в члени даних класу.

Використання класу CRadius в деякому програмному коді, наприклад, обробнику події:

CRadius CR;

double d;

d = CR.GetRadius2(); // d = 1.0 - виклик функції з константним this

d = CR.GetRadius(); // d = 2.0 - виклик звичайної функції-члена

##### Модифікація покажчика this, що передається функції-члену, яка оголошена з константним покажчиком this

Нехай задано клас CRadius, в якому є функція-член з константним покажчиком this:

// клас CRadius

class CRadius

{

double radius;

public:

CRadius(void); // конструктор класу за замовчуванням

// Звичайна функція-член класу,

// цій функції неявно передається покажчик: CRadius \* const this

double GetRadius(void)

{

radius \*= 2.0; // дозволено

return radius;

}

// Функція-член класу, оголошена з const

// цій функції передається покажчик: const CRadius \* const this

double GetRadius2(void) const

{

//radius = 2.0; // Заборонено! Помилка компіляції

return radius;

}

};

Покажчик this у класі є невидимий, функціям класу він передається неявно. Це означає, що при спробі явного опису покажчика this у класі, компілятор видасть помилку.  
Звичайній функції-члену класу GetRadius() покажчик this передається в неявному вигляді як:

CRadius \* const this;

Функції-члену GetRadius2() невидимий покажчик this передається з ключовим словом const:

// модифікований покажчик this

const CRadius \* const this;

Запис

const CRadius

означає, що об’єкт типу CRadius є константним об’єктом. І тому, змінювати значення об’єкту у тілі функції GetRadius2() не можна. Отже, не можна змінювати значення внутрішніх членів-даних класу у функції GetRadius2(). Спроба змінити значення внутрішнього члену даних radius у тілі функції GetRadius2() викличе помилку компіляції.

##### Приклад використання функції класу з константним покажчиком this

Функції з константним покажчиком this доцільно використовувати у випадках, коли потрібно, щоб функція-член базового класу випадково не перевизначилась у похідних класах. У цьому випадку константний покажчик this служить захистом даних базового класу від випадкової їх зміни у похідних класах.

##### Що буде, якщо ключове слово const розмістити перед іменем функції, а не перед тілом функції?

Це важливо. У цьому випадку, функція повертає константне значення. Але змінювати внутрішні дані класу у тілі функції можна.  
Ключове слово const перед іменем функції відноситься до значення, яке повертається функцією. Ключове слово const перед тілом функції відноситься до покажчика this.

##### Оголошення ключового слова const для статичних функцій-членів

Оголошення статичної функції-члена з ключовим словом const не має змісту. При виклику статичної функції-члена, невидимий покажчик this не передається цій функції. При спробі оголошення статичної функції-члена з константним покажчиком this компілятор видасть помилку

**... modifiers not allowed on static members functions**

##### Загальна форма оголошення функції-члена з непостійним покажчиком this. Ключове слово volatile

Для того, щоб оголосити функцію-член з непостійним покажчиком this використовується ключове слово volatile.  
Загальна форма оголошення функції з непостійним покажчиком this в класі має приблизно наступний вигляд:

class CMyClass

{

// приховані члени даних та методи класу

// ...

public:

// загальнодоступні члени даних та методи класу

// функція-член з непостійним покажчиком this

return\_type MyFun1(parameters) volatile; // out-of-line

return\_type MyFun2(parameters) volatile // inline

{

// тіло функції

// ...

}

}

// реалізація out-of-line функції

return\_type CMyClass::MyFun1 volatile

{

// тіло функції MyFun1()

// ...

}

де

* return\_type – тип, що повертається тією чи іншою функцією;
* parameters – параметри, які отримує функція.

У вищенаведеному фрагменті, у класі CMyClass оголошуються дві функції з непостійним (volatile) покажчиком this. Функція MyFun1() реалізована за межами класу (типу out-of-line). Функція MyFun2() реалізована у класі (типу inline). Перед тілом кожної функції стоїть ключове слово volatile.

##### Приклад оголошення функції з непостійним (volatile) покажчиком this?

Оголошення функції як volatile доцільно використовувати у випадках, коли потрібно обробляти об’єкт класу різними процесами, до яких можна віднести:

* обробка об’єкту процесором;
* обробка з допомогою фонових прикладних програм;
* обробка з допомогою програм обробки переривань.

##### Використання ключового слова volatile для функції-члена класу

Функція-член класу, що оголошена з ключовим словом volatile трактується компілятором у різних випадках по різному. Для таких функцій компілятор відключає деякі види оптимізації в залежності від типу оголошеного об’єкту.  
Функції-члени класу, що оголошені з ключовим словом volatile можна використовувати як для автоматичних об’єктів так і для непостійних (volatile) об’єктів.  
Для константних об’єктів використання функцій з непостійним покажчиком заборонено. Знову ж таки, все залежить від налаштувань та реалізації компілятора.

##### Приклад використання функцій-членів класу з непостійним покажчиком this

Нехай дано клас CRadius. У класі реалізовано:

* внутрішній член-даних radius;
* конструктор CRadius();
* звичайну функцію-член класу GetRadius();
* дві фунції-члени класу з непостійним покажчиком this: GetRadius2() та SetRadius().

// клас CRadius

class CRadius

{

double radius;

public:

CRadius(void); // конструктор класу за замовчуванням

// Звичайна функція-член класу,

// цій функції неявно передається CRadius \* const this

double GetRadius(void)

{

return radius;

}

// Функція-член класу, оголошена з volatile

// цій функції передається volatile CRadius \* const this

double GetRadius2(void) volatile

{

return radius;

}

// Функція-член класу, оголошена з volatile

void SetRadius(double r) volatile

{

radius = r;

}

};

Використання класу в іншому програмному коді

CRadius CR1; // автоматичний об'єкт

volatile CRadius CR2; // непостійний об'єкт

double d;

// автоматичний об'єкт

CR1.SetRadius(5.5);

d = CR1.GetRadius2(); // d = 5.5

// непостійний об'єкт

CR2.SetRadius(6.7);

d = CR2.GetRadius2(); // d = 6.7

Як видно з програмного коду, функції-члени з непостійним (volatile) покажчиком this можна використовувати для автоматичних та непостійних об’єктів. У цьому випадку, компілятор включає різні види оптимізації в залежності від типу об’єкту.

***Вкладені класи***

Мова С++ дозволяє створювати вкладені (або внутрішні класи). Це дозволяє сховати внутрішній клас від зовнішнього світу, обмеживши його область видимості зовнішнім класом. Розглянемо на прикладі класів опису комплексних чисел TDouble та TInteger в середині класу TComplex.

Визначення внутрішніх класів

**#include <stdio.h>**

**#include <Windows.h>**

**class TComplex**

**{**

**class Double**

**{**

**double x;**

**public:**

**Double(double y):x(y){printf(" Ctor Double \n");}**

**};**

**class Integer**

**{**

**int n;**

**public:**

**Integer(int m):n(m){printf(" Ctor Integer \n");}**

**};**

**Double Re;**

**Integer Im;**

**public:**

**void print();**

**TComplex(Integer k, Double v):Re(v), Im(k){};**

**};**

**int main()**

**{system("color F0");**

**TComplex Z(10.0, 20);**

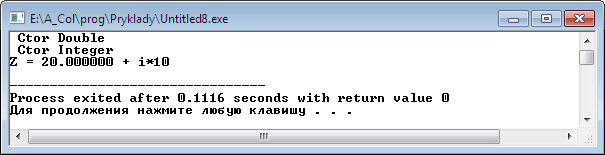
**Z.print();**

**return 0;**

**}**

**void TComplex::print() {printf("Z = %lf + i\*%d \n",Re,Im);}**

**Результати роботи**

****

Внутрішній (чи локальний) клас можна визначити не тільки усередині класу, але й усередині функції, наприклад усередині функції main().

Визначення локального класу

**#include <stdio.h>**

**#include <Windows.h>**

**int main()**

**{ system("color F0");**

**class TComplex**

**{**

**double Re;**

**int Im;**

**public:**

**void print(){printf("Z = %lf + i\*%d \n",Re,Im);};**

**TComplex(int k, double v):Re(v), Im(k){};**

**};**

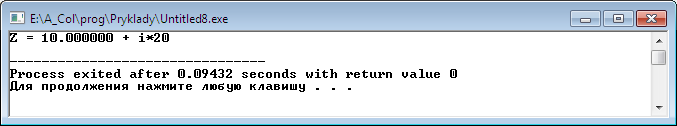
**TComplex Z(20.0, 10);**

**Z.print();**

**return 0;**

**}**

**Результати роботи**



Оскільки функції не можна визначати усередині інших функцій, їх визначення необхідно занурити усередину локального класу. Тільки під цією оболонкою вони доступні для зовнішньої функції. У силу обмеженості області видимості локальних класів, їм недоступні інші локальні змінні, оголошені усередині функції, крім статичних і зовнішніх. Крім того, усередині локальних класів неможливо визначити статичні змінні.

Запитання

1. Назвіть основні засади об’єктно-орієнтованого програмування
2. Дайте визначення об’єкта та чим він характеризується?
3. Назвіть фундаментальні властивості об'єктно-орієнтованого програмування.
4. Дайте визначення інкапсуляції, спадкуванню та поліморфізму
5. Назвіть переваги об'єктно-орієнтованого програмування.
6. Дайте визначення класу та об’єкту в ООП
7. Що може містити оголошення класу?
8. Як оголосити клас на мові С++?
9. Де може бути визначена (описана) функція класу?
10. Порівняйте звернення члена класу у функціях класу та поза межами цього класу.
11. Назвіть специфікатори доступу до членів класу.
12. Чи може з'являтися у визначенні класу один і той же специфікатор доступу багато разів?
13. У якому порядку можуть з'являтися специфікатори доступу до членів класу?
14. Як співвідносяться структури і класи в С++?
15. Що означають вбудовані методи класу? Коли вони застосовуються? Чи є вони обов’язковими?
16. Як пов’язані селектори та модифікатори з гетерами та сеттерами?
17. В чому сутність ключового слова **this**?
18. Чи може бути масив членом класу? Якого типу елементи може мати такий масив?
19. Як оголосити масив структур? Як здійснюється доступ до полів структури в масиві структур?
20. Чи можуть бути об’єкти елементами масиву? Як одержати конкретний об’єкт з такого масиву?
21. Що таке конструктор?
22. Які бувають типи конструкторів?
23. Що таке Конструктор за замовчуванням?
24. Що таке перевантаження конструкторів?
25. Дайте визначення конструктора класу.
26. Для чого призначений конструктор класу?
27. Які види конструкторів ви знаєте?
28. Скільки конструкторів можна оголосити у класі?
29. Як у класі оголосити конструктор з параметрами?
30. Для чого призначений деструктор класу?
31. Скільки деструкторів можна оголосити у класі?
32. Як викликається конструктор і деструктор?
33. Якщо клас має декілька конструкторів, то який з конструкторів викликається при створенні об’єкта цього класу?
34. Чи завжди необхідно описувати в класі деструктор? Якщо ні, то в яких випадках?
35. Особливості створення та ініціалізація об’єктів.
36. Як можна використати статичні члени?
37. Для чого призначені статичні методи класу?
38. В чому різниця між статичними та нестатичними методами?
39. Які особливості мають статичні методи?
40. Коли доцільно використовувати модифікатор **mutable?**
41. В чому полягає делегування конструкторів?
42. Що таке вкладений клас і які його особливості?
43. Які залежності існують між класами?