

Лабораторне заняття №5 (2-й семестр)

Перевантаження операторів

Ключовий показчик *this*

Ключовий показчик *this* — це показчик на об'єкт, який викликає метод класу. При кожному виклику методу йому автоматично передається показчик з ключовим словом *this*, на об'єкт, для якого викликається метод. Показчик *this* — це неявний параметр, що приймається усіма методами класу. Відповідно, в будь-якому методі показчик *this* можна використовувати для посилання на об'єкт, що його викликає.

«Дружні» функції

В C++ є можливість дозволити доступ до закритих членів класу функціям, які не є членами цього класу. Для цього достатньо оголосити ці функції «дружніми» (або «друзями») по відношенню до класу, що розглядається. Для того, щоб зробити функцію «другом» класу, необхідно включити її прототип в *public*-розділ оголошення класу з ключовим словом *friend*. Функція може бути «другом» декількох класів.

```
class someClass {  
    //...  
public:  
    friend void frnd (someClass obj);  
    //...  
};
```

Ключове слово *friend* надає функції, що не є членом класу, доступ до його закритих членів.

```
#include <iostream>  
using namespace std;  
class myclass {  
    int a, b;  
public:  
    myclass(int i, int j) { a=i; b=j; }  
    friend int sum(myclass x);  
};  
int sum(myclass x)  
{  
    return x.a + x.b;  
}  
int main ()  
{  
    myclass n (3, 4);  
    cout << sum(n);  
    return 0;  
}
```

Перевантаження операторів

C++ дозволяє перевантажувати більшість операцій в такий спосіб, щоб стандартні операції можна було використовувати і для об'єктів створених користувачем класів.

Перевантаження операцій надає можливість використовувати власні типи даних як стандартні, а складний та малозрозумілий текст програми перетворювати на інтуїтивно зрозумілий. Позначення власних операцій вводити не можна.

Можна перевантажувати будь-які операції, існуючі в C++, за винятком:

. .* ? : :: # ## sizeof

Перевантаження операцій здійснюється за допомогою методів спеціальної форми (функцій-операцій) і підпорядковується таким правилам:

- при перевантаженні операцій зберігаються кількість аргументів, пріоритети операцій та правила асоціації (зліва направо чи справа наліво), які використовуються у стандартних типах даних);

- для стандартних типів даних перевизначати операції не можна;
- функції-операції не можуть мати аргументів за замовчуванням;
- функції-операції успадковуються (за винятком “=”);
- функції-операції не можуть визначатися як static.

Функцію-операцію можна визначити трьома способами:

- як метод класу;
- як «дружню» функцію класу;
- як звичайну функцію.

У двох останніх випадках функція повинна мати хоча б один аргумент, який має тип класу, покажчик чи посилання на клас.

Функція-операція містить ключове слово *operator*, за яким слідує знак операції, яку треба перевизначити:

```
<тип> operator <операція> (<список параметрів>)
{ <тіло функції> }
```

Такий синтаксис повідомляє компіляторові про те, що, якщо операнд належить до визначеного користувачем класу, треба викликати функцію з таким ім'ям, коли зустрічається в тексті програми ця операція.

Перевантаження операторів з використанням методів класу.

Можливо перевантажувати як бінарні оператори (наприклад, «+», «=»), так і унарні (наприклад, «++», «--»).

При перевантаженні бінарної операції функція-оператор має один параметр. В усіх випадках, об'єкт, що активує функцію-оператор, передається неявним чином за допомогою покажчика *this*. Об'єкт, що знаходиться справа від знаку операції, передається методу як параметр.

```
class three_d
{
    int x, y, z;
public:
    three_d operator+ (three_d op2);
```

```
};
three_d three_d::operator+(three_d op2)
{
    three_d temp;
    temp.x=x+op2.x;
    temp.y=y+op2.y;
    temp.z=z+op2.z;
    return temp;
}
```

Зверніть увагу, в рядку

```
temp.x=x+op2.x;
```

`x` відповідає `this->x`, де `x` асоційовано з об'єктом, який викликає функцію-оператор.

При перевантаженні унарної операції функція-оператор не має параметрів, тобто жоден об'єкт не передається явним чином, а операція виконується над об'єктом, який генерує виклик цього методу через неявно переданий покажчик *this*.

Розглянемо приклад, в якому для об'єктів типу *three_d* визначається операція інкременту.

```
class three_d
{
    int x,y,z;
public:
    three_d operator--();
};
three_d three_d::operator--()
{
    x--;
    y--;
    z--;
}
```

Оператори інкременту та декременту мають як префіксну, так і постфіксну форми.

В вищенаведеному прикладі функція `operator++` визначає префіксну форму оператора «++» для класу *three_d*. Якщо необхідно перевантажити постфіксну форму оператора «++», це можна зробити наступним чином:

```
three_d three_d::operator++(int notused)
```

Параметр *notused* не використовується самою функцією. Він є індикатором для компілятора, який дозволяє відрізнити префіксну форму оператора інкремента від постфіксної. Так само він використовується для оператора декремента («--»).

Перевантаження операторів за допомогою «дружніх» функцій

Оскільки «дружні» функції не є членами класу, вони не можуть мати неявний аргумент *this*. Тому, при перевантаженні бінарної функції оператора обидва операнди

передаються функції, а при перевантаженні унарних операторів передається один операнд.

Наступні оператори не можуть використовувати перевантаження за допомогою «дружніх» функцій:

= () [] ->.

```
class three_d {
    int x, y, z;
public:
    friend three_d operator*(three_d op1, three_d op2);
};
three_d operator*(three_d op1, three_d op2)
{
    three_d temp;
    temp.x = op1.x * op2.x;
    temp.y = op1.y * op2.y;
    temp.z = op1.z * op2.z;
    return temp;
}
```

Якщо необхідно перевантажити оператори інкременту або декременту з використанням «дружньої» функції, необхідно передати їх посилання на об'єкт.

Оскільки параметр у такому вигляді являє собою неявний покажчик на аргумент, то зміни, які будуть вноситися в параметр, впливатимуть і на аргумент. Тобто використання посилання в якості параметру функції дозволяє їй успішно інкрементувати або декрементувати об'єкт, який передається як операнд.

Якщо для перевантаження операторів інкременту та декременту використовується функція-«друг», її префіксна форма приймає один параметр (який і є операндом), а постфіксна форма – два параметри (другим є цілочисельне значення, яке не використовується).

```
friend three_d operator++(three_d &op1)
{
    op1.x++;
    op1.y++;
    op1.z++;
    return op1;
}
friend three_d operator++(three_d &op1, int notused)
{
    three_d temp = op1;
    op1.x++;
    op1.y++;
    op1.z++;
    return temp;
}
```

Перевантаження операторів за допомогою звичайних функцій

Оскільки принцип перевантаження операторів через звичайні та «дружні» функції практично ідентичний (вони мають різні рівні доступу до закритих членів класу), єдина відмінність – у випадку «дружньої» функції її необхідно обов'язково

оголосити в класі, на відміну від звичайної функції, яку достатньо визначити поза тілом класу, без вказування додаткового прототипу функції. Доступ до закритих членів класу, у випадку використання звичайної функції для перевантаження операторів, відбувається через геттери.

```
class three_d
{
    int x,y,z;
public:
    int getX(){return x;}
    int getY(){return y;}
    int getZ() {return z;}
};
three_d operator+ (three_d &op1,three_d &op2)
{
    return
three_d(op1.getX()+op2.getX(),op1.getY()+op2.getY(),op1.getZ()+op2.getZ(
));
}
```

Приклад.

Розглянемо перевантаження операторів на прикладі класу “three_d”. Перевантажимо:

- оператори «+», «=» та префіксний та постфіксний декременти як методи класу;
- оператори «*», префіксний та постфіксний інкременти як «дружні» функції;
- оператор «-» як звичайну функцію.
- Заголовковий файл «three_d.h»

```
#ifndef THREE_D_H
#define THREE_D_H
class three_d
{
    int x,y,z;
public:
    three_d(){x=y=z=0;}
    three_d(int i,int j,int k) {x=i;y=j;z=k;}
    int getX(){return x;}
    int getY(){return y;}
    int getZ() {return z;}
    three_d operator+ (three_d op2);
    three_d operator= (three_d op2);
    three_d operator-- ();
    three_d operator-- (int notused);
    friend three_d operator++(three_d &op1);
    friend three_d operator++(three_d &op1, int notused);
    friend three_d operator* (three_d &op1, three_d &op2);
    void show();
};
three_d operator- (three_d &op1,three_d &op2);

#endif // THREE_D_H
```

- файл з реалізацією класу «three_d.cpp»

```

#include "three_d.h"
#include <iostream>

using namespace std;

three_d operator- (three_d &op1, three_d &op2)
{
    return three_d(op1.getX()-op2.getX(), op1.gexY()-
op2.gexY(), op1.getZ()-op2.getZ());
}

three_d three_d::operator+(three_d op2)
{
    three_d temp;
    temp.x=x+op2.x;
    temp.y=y+op2.y;
    temp.z=z+op2.z;
    return temp;
}

three_d three_d::operator= (three_d op2)
{
    x=op2.x;
    y=op2.y;
    z=op2.z;
    return *this;
}

three_d three_d::operator--()
{
    x--;
    y--;
    z--;
    return *this;
}

three_d three_d::operator--(int notused)
{
    three_d temp = *this; //збереження початкового значення
    x--;
    y--;
    z--;
    return temp;
}

three_d operator* (three_d &op1, three_d &op2)
{
    three_d temp;
    temp.x = op1.x * op2.x;
    temp.y = op1.y * op2.y;
    temp.z = op1.z * op2.z;
    return temp;
}

three_d operator++(three_d &op1)
{
    op1.x++;
    op1.y++;
    op1.z++;
    return op1;
}

three_d operator++(three_d &op1, int notused)

```

```

{
    three_d temp = op1;
    op1.x++;
    op1.y++;
    op1.z++;
    return temp;
}
void three_d::show()
{
    cout<<x<<"<<y<<"<<z<<"\n";
}

```

- файл «main.cpp» з прикладом використання перевантажених операторів

```

#include <iostream>
#include "three_d.h"
using namespace std;

int main() {
    setlocale(LC_ALL, "Russian");
    three_d a(1,2,3), b(10,10,10), c;
    cout<<"c=a+b \n";
    c=a+b;
    a.show(); b.show(); c.show();
    cout<<"c=b=a \n";
    c=b=a;
    a.show(); b.show(); c.show();
    cout<<"a--c \n";
    a=a--c; //префіксна форма декременту - об'єкт отримує
            //значення c після його декрементування
    a.show(); c.show();
    cout<<"a=c-- \n";
    a=c--; //постфіксна форма декременту - об'єкт отримує
            //значення c до його декрементування
    a.show(); c.show();
    cout<<"c=a*b \n";
    c=a*b;
    c.show();
    cout<<"a++c; \n";
    a=a++c; a.show(); c.show();
    cout<<"a=c++ \n";
    a=c++; a.show(); c.show();
    cout<<"c=a-b \n";
    c=a-b; c.show();
}

```

Лабораторна робота 5.

Хід виконання завдання:

1. Змодельовати клас для заданого поняття (згідно варіанту).
2. Перевантажити вказані у варіанті оператори – необхідно реалізувати кожен з трьох способів перевантаження:
 - як метод класу,
 - як «дружню» функцію
 - як звичайну функцію.

3. Написати програму, в якій користувач матиме можливість використовувати перевантажені оператори, проводячи операції з об'єктами.

Варіанти завдань.

Варіант 1.

Клас – комплексне число. Перевантажити оператори: віднімання (-), порівняння (>), декремент (--), присвоювання (=).

Варіант 2.

Клас – комплексне число. Перевантажити оператори: додавання (+), порівняння (==), декремент (--), присвоювання (=).

Варіант 3.

Клас – час. Перевантажити оператори: віднімання (-), порівняння (>,<), інкремент (++), присвоювання (=).

Варіант 4.

Клас – час. . Перевантажити оператори: додавання (+), порівняння (==), декремент (--), присвоювання (=).

Варіант 5.

Клас – вектор. Перевантажити оператори: додавання (+), скалярний добуток (%), множення вектора на число (*=), присвоювання (=), декремент (--).

Варіант 6.

Клас – вектор. Перевантажити оператори: віднімання(-), векторний добуток (*), інкремент (++), присвоювання (=).

Варіант 7.

Клас – дріб. Перевантажити оператори: віднімання (-), порівняння (>,<), ділення (/), інкремент (++), присвоювання (=).

Варіант 8.

Клас – дріб. Перевантажити оператори: додавання (+)порівняння (==, !=), добуток (*), декремент (--), присвоювання (=).

Варіант 9.

Клас- пряма (задається через координати двох точок). Перевантажити оператори: визначення паралельності двох прямих (||), визначення кута між двома прямими (%), присвоювання (=), декремент (--).

Варіант 10.

Клас- пряма (задається рівнянням $y = ax + b$). Перевантажити оператори: визначення паралельності двох прямих (||), визначення кута між двома прямими (%), присвоювання (=), інкремент (++).

Варіант 11.

Клас – матриця. Перевантажити оператори: присвоювання (=), додавання (+), добуток двох матриць (*), інкремент (++).

Варіант 12.

Клас – матриця. Перевантажити оператори: присвоювання (=), віднімання (-), добуток матриці на число (*=), декремент (--).

Варіант 13.

Клас – парабола. $y = ax^2 + bx + c$. Перевантажити оператори: присвоювання (=), перевірка співпадіння парабол (||), перевірка перетину парабол (/).

Варіант 14.

Клас – вектор. Перевантажити оператори: інкремент(--), визначення кута між двома векторами (/), присвоювання (=).

Варіант 15.

Клас – вектор. Перевантажити оператори: додавання (+), множення вектора на число (*=), присвоювання (=), інкремент (++).

Варіант 16.

Клас – комплексне число. Перевантажити оператори: порівняння (<), ділення (/), інкремент (++), присвоювання (=).

Варіант 17.

Клас – комплексне число. Перевантажити оператори: порівняння (!=), добуток (*), інкремент (++), присвоювання (=).

Варіант 18.

Клас – дріб. Перевантажити оператори: додавання (/) порівняння (<), добуток (*), інкременту (++), присвоювання (=).

Варіант 19.

Клас – матриця. Перевантажити оператори: присвоювання (=), додавання (+), добуток матриці на число (*=), інкременту (--).

Варіант 20.

Клас – матриця. Перевантажити оператори: остатку від ділення (%), порівняння (!=), добутку матриць (*), декремент (--).

Варіант 21.

Клас – вектор. Перевантажити оператори: додавання числа(+=), скалярний добуток (*), інкремент (++), присвоювання (=).

Варіант 22.

Клас – час. Перевантажити оператори: додавання (+), порівняння (>), інкременту (++), присвоювання (=).

Варіант 23.

Клас- пряма (задається через координати двох точок). Перевантажити оператори: визначення паралельності двох прямих (||), визначення кута між двома прямими (%), присвоювання (=), кутів з осями координат (%).

Запитання для самоконтролю.

- 1) Що таке перевантаження операцій?
- 2) Які операції не можуть бути перевантаженими?
- 3) Механізм перевантаження операторів
- 4) Чим відрізняється дія перевантаженої операції ++ при її використанні у префіксній формі від її використання у постфіксній формі?
- 5) Скільки має бути аргументів у функції, яка перевантажує бінарну операцію за допомогою дружньої функції?
- 6) Показчик this, яка його функція?
- 7) Чим відрізняється перевантаження операцій через звичайну функцію та через дружню?
- 8) У чому особливості перевантаження операції за допомогою методів класу.
- 9) Що таке дружня функція? Її особливості.