

Об'єктно-орієнтоване програмування

на основі мови C++
2й семестр

- показчик this
- дружні функції
- перевантаження операцій

показчик * This.

Кожен об'єкт супроводжується показником на самого себе - званним показником **this** - це неявний аргумент у всіх посиланнях на елементи всередині цього об'єкта.

Приклад 6. Використання показчика * this.

```
#include <iostream>
using namespace std;
class Test
{
public:
    Test (int = 0);
    void print() const;
private:
    int x;
};

Test::Test(int a) { x = a; }

void Test::print() const
{
    cout << "        x = " << x << endl;

    cout << "    this->x = " << this->x << endl;
    cout << " (*this).x = " << (*this).x << endl;
}

int main()
{
    Test a(12);
    a.print();
    return 0;
}
```

C:\

```
x = 12  
this->x = 12  
(*this).x = 12
```

```
Process returned 0 (0x0)   execution time : 0.141 s  
Press any key to continue.
```

Ще один приклад використання покажчика this:

```
Time &   setHour (int);  
Time &   setMinute (int);  
Time &   setSecond (int);
```

```
Time &   Time::setHour(int h)  
{  
    hour = (h >= 0 && h < 24) ? h : 0;  
    return *this;  
}
```

```
t.setHour(18).setMinute(30).setSecond(22);
```

- Дружні функції

Дружні функції і класи

дружня функція - це функція, яка не є членом класу, але має доступ до членів класу, оголошеним в полях **private** або **protected**.

дружня функція оголошується всередині класу, до елементів якого їй потрібен доступ, з ключовим словом `friend`.

Дружня функція може бути звичайною функцією або методом іншого раніше визначеного класу.

Одна функція може бути дружньою відразу декількома класами.

Дружні функції - приклад

```
class monstr;
```

```
class hero {
```

```
    public:
```

```
    void kill (monstr &);
```

```
};
```

```
class monstr{
```

```
    friend int steal_ammo(monstr &);
```

```
    friend void hero :: kill (monstr &);
```

```
};
```

```
int steal_ammo(monstr & M) {return -M.ammo;}
```

```
void hero :: kill (monstr & M) {
```

```
M.health = 0; M.ammo = 0;
```

```
}
```

Дружні класи - приклад

```
class hero{  
    ...  
    friend class mistress;  
}  
class mistress {  
    ...  
    void f1 ();  
    void f2 ();  
}
```



- перевантаження операцій

2. Перевантаження операцій.

Операції перевантажуються шляхом складання опису функції (з заголовком і тілом), ім'я функції складається з ключового слова **operator** і знака функції. наприклад:

```
int operator < (const String &) const;
```

Щоб використовувати операцію над об'єктами класів, ця операція повинна бути перевантажена, але є два винятки:

1) операція присвоювання (=) може бути використана з кожним класом без явної перевантаження. За замовчуванням операція присвоювання зводиться до побітові копіювання полів класу, але таке побітове копіювання неприпустимо для класів з динамічно створеними полями; для таких класів треба явно перевантажувати операцію присвоювання.

2)

операція отримання адреси (&) може бути використана з об'єктами будь-яких класів без перевантаження; вона просто повертає адресу об'єкта в пам'яті. Але операцію адресації можна також і перевантажувати.

Операції, які можуть бути перевантажені:

+ - * /% ^ & |

~! = <> + = - = * =

/ =% = ^ = & = << >>

==! = <=> = && || ++ -

-> [] () new delete

Операції, які не можуть бути перевантажені:

.. * ::?: Sizeof

Рекомендована форма перевантаження операцій.

Операция	Рекомендуемая форма перегрузки
Все унарные операции	Метод класса
= [] () ->	Обязательно метод класса
+= -= *= /= %= &= ^=	Метод класса
Остальные бинарные операции	Внешняя функция друг / метод класса
Поместить в поток << Взять из потока >>	Только внешняя функция друг

- при перевантаженні операцій зберігаються кількість аргументів, пріоритети операцій і правила асоціації (справа наліво або зліва направо), які використовуються в 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.

формат:

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

Функцію-операцію можна визначити:

- як метод класу
- як дружню функцію класу
- як звичайну функцію

Перевантаження унарних операцій

1. Усередині класу:

```
class monstr{  
    ...  
    monstr & Operator ++ ()  
    {++ health; return * this;}  
};  
  
monstr Vasia;  
cout << (++Vasia).get_health();
```

Перевантаження унарних операцій

2. Як дружню функцію:

```
class monstr{  
    ...  
    friend      monstr & Operator ++ ( monstr & M);  
};  
monstr& Operator ++ (monstr & M) {++M.health; return M;}
```


3. Поза класу:

```
void change_health (int he) {health = he;}  
...  
monstr & operator ++ (monstr & M) {  
    int h = M.get_health (); h ++;  
    M.change_health (h) ;  
    return M;}  
}
```


Перевантаження Постфіксий инкремента

```
class monstr{  
    ...  
    monstr operator ++ (int) {  
        monstrM (* this); health ++;  
        return M;  
    }  
};  
monstr Vasia;  
cout << (Vasia++) .get_health();
```

Наявність аргументу
(int) Вказує, що це
постфіксий інкремент



1. Усередині класу:

```
class monstr {  
    ...  
    bool operator> (const monstr & M) {  
        if (health> M.get_health ())  
            return true;  
        return false; }  
};
```

2. Поза класу:

```
bool operator> (const monstr & M1, const monstr &  
M2) {  
    if (M1.get_health ()> M2.get_health ())  
        return true;  
    return false;  
}
```

Перевантаження операції присвоювання

операція-функція повинна повертати посилання на об'єкт, для якого вона викликана, і приймати в якості параметра єдиний аргумент - посилання на присвоюється об'єкт

```
const monstr& Operator = (const monstr & M) {  
    // Перевірка на самоприсваївання:  
    if (& M == this) return * this;  
    if (name) delete [] name;  
    if (M.name) {name = new char [strlen(M.name) + 1];  
        strcpy(Name, M.name);}  
    else name = 0;  
    health = M.health; ammo =M.ammo; skin =M.skin;  
    return *this;}
```

```
monstr A (10), B, C;  
C = B = A;
```

перевантаження операцій new і delete

- їм не потрібно передавати параметр типу класу;
першим параметром функцій new і new [] повинен передаватися розмір об'єкта типу size_t (це тип, що повертається операцією sizeof, він визначається в заголовки <stddef.h>); при виклику він передається в функції неявним чином;
- вони повинні визначатися з типом значення, що повертається void *, навіть якщо return повертає покажчик на інші типи (найчастіше на клас);
- операція delete повинна мати тип повернення void і перший аргумент типу void *;
- операції виділення і звільнення пам'яті є статичними елементами класу.

```
class Obj {...};
```

```
class pObj{
```

```
...
```

```
private:
```

```
Obj * P;
```

```
};
```

```
pObj * P = new pObj;
```

```
static void * operator new (size_t size);
```

```
void operator delete (void * ObjToDie, size_t  
size);
```

```
#include <new.h>
```

```
SomeClass a = new (buffer) SomeClass(his_size);
```

Перевантаження операції приведення типу

```
operator ім'я_нового_типа ();
```

```
monstr :: operator int () {  
    return health;  
}
```

```
...
```

```
monstr Vasia; cout << int (Vasia);
```

Перевантаження операції виклику функції

```
class if_greater {  
    public:  
        int operator () (Int a, int b) const {  
            return a > b;  
        }  
};
```

```
if_greater x;  
cout << x (1, 5) << endl; //x.operator () (1, 5))  
cout << if_greater () (5, 1) << endl;
```

Перевантаження операції індексування

```
class Vect{
public:
    explicit Vect(int n = 10);
    //ініціалізація масивом:
    Vect(const int a [], int n);
    ~Vect() {Delete [] p; }
    int& Operator [] (int i);
    void Print ();
private:
    int* P;
    int size;
};
```


Перевантаження операції індексування

```
Vect::Vect(int n): size (n) {p = new int[Size];}
Vect::Vect(const int a [], int n): size (n) {
    p = new int[Size];
    for (int i = 0; i <Size; i++) p [i] = A [i]; }

int& Vect::operator [] (int i) {
    if (i <0 || i >= Size) {
        cout << "невірний індекс (i = "<< i << ")" <<
endl;
        cout << "завершення програми"<< endl;
        exit (0); }
    return p [i];
}
```

Перевантаження операції індексування

```
void Vect:: Print () {  
    for (int i = 0; i <Size; i++) cout << p [i]  
    << " ";  
    cout << endl; }
```

```
int main () {  
    int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9,  
10};  
    Vect a (arr, 10);  
    a.Print();  
    cout << a [5] << endl;  
    cout << a [12] << endl;  
    return 0;  
}
```