

Лекція 3

Об'єктно-орієнтоване програмування

Лектор: *Розова Людмила Вікторівна*

План лекції 3

- 1 Дружні функції, класи
- 2 Перевантаження операторів
- 3 Статичні змінні

Матеріали курсу:

<https://github.com/LRozova/oop1>

Дружні функції

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 декільком класам.

```
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;  
}
```

В одному класі можуть бути визначені декілька дружніх функцій

Дружні функції можуть бути оголошені в іншому класі:

```
class Car; //прототип класу
class Man
{public:
    Man(string n)
        { name = n; }
    void driveCar(Car &a); // повертає назву авто, яким керує людина
    void setPrice(Car &a, int price); // встановлює ціну для
авто
private:
    string name; // ім'я
};

class Car
{
    friend void Man::driveCar(Car &);
    friend void Man::setPrice(Car &, int price);
public:
    Car (string carName, int carPrice)
        {name = carName; price = carPrice; }
private:
    string name; // назва авто
    int price; // ціна авто
};
```

Дружні класи

Дружні класи - це класи, які мають доступ до всіх закритих членів класу, до якого вони є друзями.

```
class Car
```

```
{ friend class Man
```

```
.....
```

```
};
```

```
class Man
```

```
{.....
```

```
void driveCar(Car &a);
```

```
void setPrice(Car &a, int price);
```



```
};
```

!!! Але **class Car**, при цьому, не є другом **class Man**, тому не має доступу до його приватних елементів

Перевантаження операцій

Перевантаження операторів (operator overloading) - це можливість застосовувати вбудовані оператори мови до різних типів, в тому числі і створених користувачем.

Переваги цього: Перевантаження операцій надає можливість використовувати власні типи даних як стандартні, це перетворює текст програми на інтуїтивно зрозумілий

➤ !Позначення власних операцій вводити не можна

Перевантаження операцій здійснюється за допомогою методів спеціальної форми «функцій-операцій» за такими правилами:

- при перевантаженні операцій зберігаються кількість аргументів, пріоритети операцій та правила асоціації (зліва направо чи справа наліво, як 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.

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

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

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.

Операції, які **можуть** бути перевантажені:

+ - * / % ^ & | ~ ! = < > += -=
 *= /= %= ^= &= << >> == != <= >=
 && || ++ -- -> [] () new delete

Операції, які **НЕ можуть** бути перевантажені:

. .* ? : :: # ## sizeof

Операція	Рекомендована форма перевантаження
Всі унарні операції	Зовнішня функція/ friend / метод класу
= [] () ->	Метод класу
+= -= *= /= %= &= ^=	Метод класу
Інші бінарні операції	Зовнішня функція/ friend / метод класу
<< >>	Зовнішня функція / friend

Перевантаження операторів з використанням методів класу

Можливо перевантажувати як унарні операції,
наприклад, ++, --, -, +

Так і бінарні оператори:

наприклад, +, -, =, *, /

Перевантаження унарної операції методом класу:

- функція-оператор не має параметрів;
- операція виконується над об'єктом, який генерує виклик цього методу через неявно переданий покажчик *this*.

Розглянемо перевантаження унарних операцій на прикладі операція **інкременту (префіксна форма)**.

```
class Point
{
    int x, y, z;
public:
    ...
    Point operator++();
};
Point Point::operator++()
{
    ++x; ++y; ++z; //інкремент координат x, y, z
    return *this; //повертання значення
}
...
Point a(1, 2, 3);
++a;
```

Операція інкременту (постфіксна форма).

```
class Point
```

```
{
```

```
    int x, y, z;
```

```
public:
```

```
    Point operator++(int notused) ;
```

```
...};
```

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

```
{Point temp = *this; //збереження вихідного значення
```

```
    x++; //інкремент координат x, y, z
```

```
    y++; z++;
```

```
    return temp; //повертання вихідного значення
```

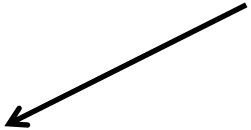
```
}
```

```
...
```

```
Point a(1, 2, 3);
```

```
a++;
```

Вказівка компілятора,
що це постфіксна форма



Перевантаження бінарних операцій

- В цьому випадку **функція-оператор** має тільки **один параметр**.
- Інший параметр передається неявним чином за допомогою покажчика *this* на об'єкт, для якого викликається функція-оператор.
- Об'єкт, що знаходиться справа від знаку операції, передається методу як параметр

Розглянемо на прикладі *бінарної операції* +

```
class Point
{   int x,y,z;
public:
    Point operator+ (Point op2);
};
Point Point::operator+(Point op2)
{
    Point temp;
    temp.x=x+op2.x;
    temp.y=y+op2.y;
    temp.z=z+op2.z;
    return temp;
}
.....
Point a(1,2,3), b(2,3,4), c;
c=a+b;...
```

x,y,z відповідають this->x, this->y, this->z для об'єкту, що викликає операцію (який зліва)

Розглянемо на прикладі *бінарної операції* >

```
class Person
{
    string Name;
    int age;
public:
    ...
    bool operator >(const Person &Man) {
        if( age > Man.age() )
            return true;
        return false; }
};
...
Person M1 ("Ivan", 25), M2 ("Petro", 30);
if (M1>M2) ...//порівняння за полем age
```

Перевантаження операторів з використанням «дружніх» функцій

- «дружні» функції мають бути оголошені в класі;
- «дружні» функції мають доступ до приватних елементів класу;
- «дружні» функції не є членами класу, вони не можуть мати неявний аргумент *this*, тому:
 - ❑ при перевантаженні **бінарних** операторів обидва операнди передаються функції-оператору,
 - ❑ при перевантаженні **унарних** операторів передається один операнд.
- не можна перевантажувати як дружні функції
 - = () [] ->

Перевантаження унарних операцій на прикладі операції декременту

Префіксна форма

```
friend Point operator--(Point &op1)
{
    op1.x--;
    op1.y--;
    op1.z--;
    return op1;
}
```

Постфіксна форма

```
friend Point operator--(Point &op1, int noused)
{
    Point temp = op1;
    op1.x--;
    op1.y--;
    op1.z--;
    return temp;
}
```


Перевантаження операторів як «дружні» функції

Перевантаження бінарних операцій на прикладі операції *

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

Два операнди



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

```
Point a(1,2,3), b(2,3,4), c;  
c=a*b;...
```

Перевантаження операторів з використанням звичайних функцій

- Звичайні функції не мають доступу до приватних елементів класу, тому доступ до закритих членів класу відбувається через спеціальні методи класу (геттери);
- Механізм перевантаження:
 - ❑ при перевантаженні *бінарних* операторів обидва операнди передаються функції-оператору,
 - ❑ при перевантаженні *унарних* операторів передається один операнд.

Перевантаження бінарних операцій на прикладі операції +

```
class Point
{
    int x, y, z;
public:
    int getX() {return x;}
    int getY() {return y;}
    int getZ() {return z;}
};

Point operator+ (Point &op1, Point &op2)
{
    return
    Point (op1.getX()+op2.getX(), op1.getY()+op2.ge
    xY(), op1.getZ()+op2.getZ());
}
```

Доступ до приватних даних за допомогою спеціальних методів

Перевантаження бінарних операцій на прикладі операції <

```
bool operator <(const Person &M1, const
Person &M2)
{
    if( M1.getAge() < M2.getAge())
        return true;
    return false;
}
```

- Виконується за допомогою **методу класу**.
- Операторна-функція повинна повертати посилання на об'єкт, для якого вона викликана, і приймати в якості параметра єдиний аргумент - посилання на об'єкт, який присвоюється
- Може виконуватися за замовчуванням

```
Person& operator = (const Person &M)
{
    // !Треба робити перевірку на самоприсвоювання:
    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;
    age = M.age;
    return *this;}

```

Можна заборонити виконання операції = та створення копій об'єктів

```
Person& operator = (const Person &M) =delete;
```

Перевантаження оператору індексування []

- Використовувати стандартний запис C++ для доступу до елементів членів класу.
- Виконується за допомогою **методу класу** для елементів класу.

```
class Vector
{
    int size;
    int *mas;
public:
    Vector(int s);
    int operator [] (int i) //int & operator [] (int i);
    {
        if (i < 0) cout << "index < 0";
        if (i >= size) cout << "index > size";
        return mas[i];
    }
    ... } ; ...
    Vector x(15); //об'єкт класу Vector
    cout << x[5] << endl; //повертає mas[5]
```

Параметр операторної функції **operator[]()** може мати будь який тип даних: *символ, дійсне число, строка*

- Створюється не новий засіб виклику функції, а операторна функція **operator()**, якій можна передати довільне число параметрів.
- Виконується за допомогою **методу класу**.
- Об'єкт, який генерує виклик операторної функції, адресується покажчиком **this**

```
void Vector:: operator() (int n)
{
    for(int i=0; i<size; i++)
        mas[i]=n*mas[i];
} ...
};
...
Vector x(10); //об'єкт класу Vector
x(3); //повертає x.operator()(3);
```

Перевантаження операторів помістити в потік << та взяти з потоку >>

- Перевантажуються для вводу виведення типів даних, які визначені користувачем (класи)
- Перевантажуються за допомогою **дружніх функцій**
- повертають посилання на об'єкт типу **ostream** чи **istream**

```
class Point
{ double x, y;
public:
    Point();
    friend ostream& operator<< (ostream &output,
const Point &point);
    friend istream &operator >>(istream &input,
Point &point);
...};
```


Перевантаження операторів помістити в потік << та взяти з потоку >>

25

```
ostream& operator<< (ostream &output, const Point  
&point)  
{ output << "Point(" << point.x << ", " << point.y  
<< ") "; //має доступ до приватних елементів, бо є другом  
  return output;  
}
```

```
istream &operator >>(istream &input, Point &point)  
{  cout<<"Input x"; cin>> point.x;  
  cout<<"Input y"; cin>> point.y;  
  return input;  
}
```

...

```
Point p1;
```

```
cin>>p1;//для об'єкту класа Point викликає перевантажений
```

оператор >>

```
cout<<p1;//для об'єкту класа Point викликає перевантажений
```

оператор <<

Статичні змінні

В C++ є можливість доступу всіх створених об'єктів конкретного класу до однієї змінної (полю), вміст якої зберігається в одному місці. Для цього оголошують змінну:

static тип ім'я ;

- Ключове слово **static** може бути використано як для атрибутів, так і для методів класу.
- Особливістю елементів **static** є те, що вони належать класу, а не об'єкту цього класу, тому можуть бути використані навіть без створення об'єкту класу, і незалежно від кількості створених об'єктів даного класу. В пам'яті буде знаходитись **лише одна копія елементу**, що об'явлено як статичний.
- Доступ до статичних змінних відбувається з використанням імені класу, оператору розширення видимості “**::**” та можливий **тільки після ініціалізації**:

тип ім'я_класу:: ім'я_змінної = початк.значення;

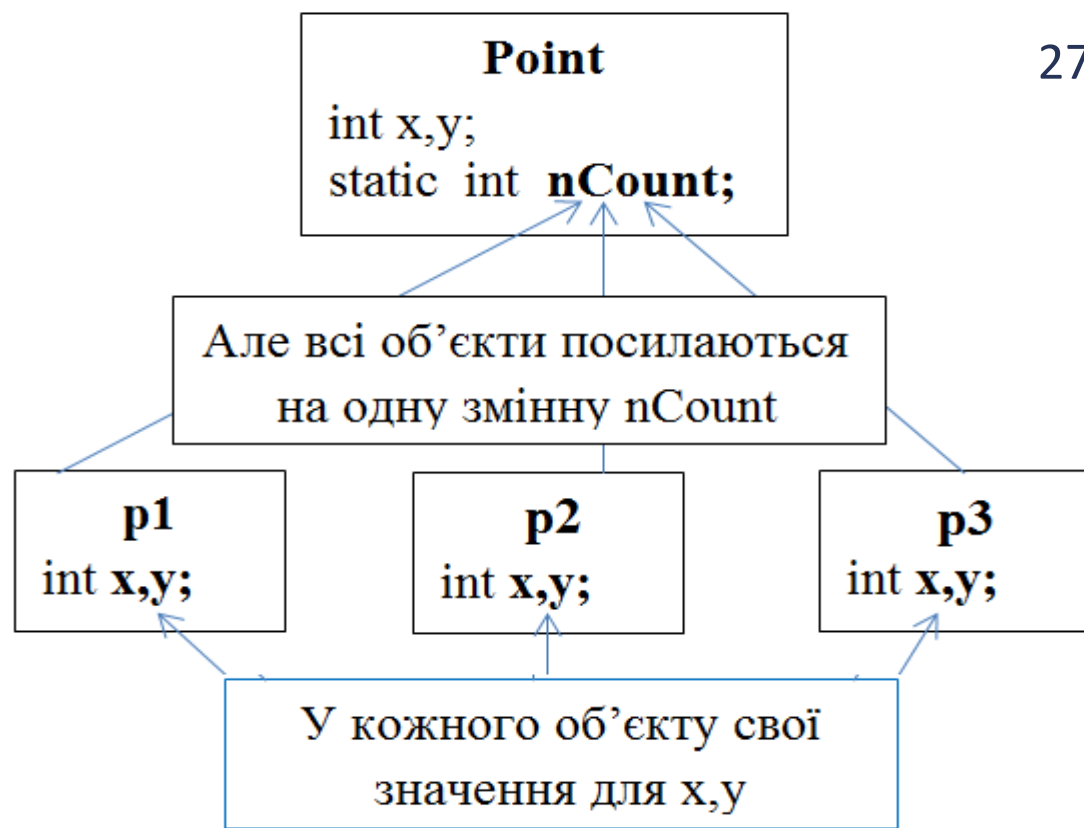
Статичні змінні

27

```
class Point {  
public:  
    int x,y;  
    static int nCount;  
    Point(){};  
};
```

```
int Point::nCount=0;
```

```
int main()  
{Point p1,p2,p3;  
    cout<<"nCount:"<<Point::nCount<<endl;  
    cout<<p1.nCount<<" "<<p2.nCount<<" "<<p3.nCount  
    <<endl;  
    Point::nCount=1;  
    cout<<p1.nCount<<" "<<p2.nCount<<" "  
    <<p3.nCount<< endl;  
    return 0;}
```

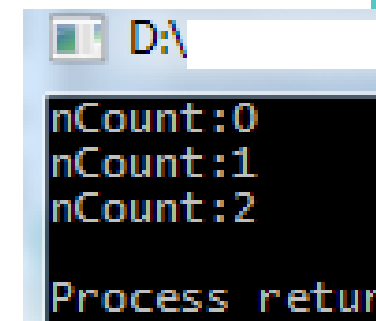


The screenshot shows a command prompt window with the following output:
D:\
nCount:0
0 0 0
1 1 1
Process returned

Статичні змінні

Підрахунок створених об'єктів класу

```
class Point {  
    public:  
        int x,y;  
        static int nCount;  
        Point () {++nCount};  
};  
  
int Point::nCount=0;  
  
int main()  
{cout<<"nCount:"<<Point::nCount<<endl;  
  Point p1;  
  cout<<"nCount:"<<Point::nCount<<endl;  
  Point p2;  
  cout<<"nCount:"<<Point::nCount<<endl;  
  return 0;}
```



```
D:\  
nCount:0  
nCount:1  
nCount:2  
Process retur
```

Дякую за увагу!