

# Лекція 7

## Об'єктно-орієнтоване програмування

Лектор: *Розова Людмила Вікторівна*

# План лекції 7



Шаблони функцій

---



Шаблони класів

---

Матеріали курсу:

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

# Шаблони функцій

**Шаблони в C++ (template)** дозволяють визначати сімейства функцій або класів.

**Шаблони** призначені для кодування узагальнених алгоритмів, без прив'язки до деяких параметрів (наприклад, типів даних).

**Шаблони функцій** (шаблонна функція, узагальнена функція) - це узагальнений опис поведінки функцій, яка може бути викликана для об'єктів різних типів. Шаблон функції являє собою сімейство різних функцій (або опис алгоритму).

За описом шаблон функції схожий на звичайну функцію: різниця в тому, що деякі елементи не визначені (типи, константи) і є параметризовані.

**template** <**typename** або **class** параметр>

//може бути декілька параметрів

Заголовок функції

{тіло функції}

Не має відношення  
до класів в ООП

## Приклад 1

```
#include <iostream>
using namespace std;

//шаблон функції (трафарет) для додавання чисел різного типу
template<typename T> //або <class T>,
//задавання параметру
T add(T x, T y) //заголовок функції
{
    return x + y;
}

int main()
{
    double a1 = 4.7, b1 = 5.3;
    double n1 = add(a1, b1) ; //заміна параметра
//на тип double у функції add()
```

```
int a2 = 4, b2 = 5;
```

```
int n2 = add(a2, b2); //заміна параметра
```

```
//на тип int у функції add()
```

```
short int a3 = 3, b3 = 2;
```

```
short int n3 = add(a3, b3); //заміна параметра
```

```
//на тип short int у функції add()
```

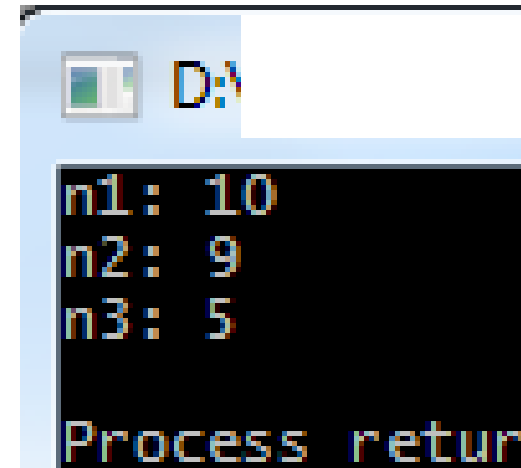
```
cout << "n1: " << n1 << endl;
```

```
cout << "n2: " << n2 << endl;
```

```
cout << "n3: " << n3 << endl;
```

```
return 0;
```

```
}
```



```
n1: 10
n2: 9
n3: 5
Process retur
```

**Шаблони функцій** не компілюються безпосередньо. Коли компілятор зустрічає виклик шаблонної функції, він копіює шаблон функції і замінює типи параметрів шаблону функції фактичними (переданими) типами даних.

Функція з фактичними типами даних називається **екземпляром шаблону функції** (або «об'єктом шаблону функції»).

**Шаблони функцій** працюють як з фундаментальними типами даних (char, int, double і т.д.), так і з класами (але повинні бути визначені операції для об'єктів класу, які використовуються в шаблоні).

**Шаблони функцій** також можуть бути **перевантажені**. Тоді вони повинні відрізнатись списками параметрів.

**Шаблони функцій** можуть мати також **параметри за замовчуванням**:

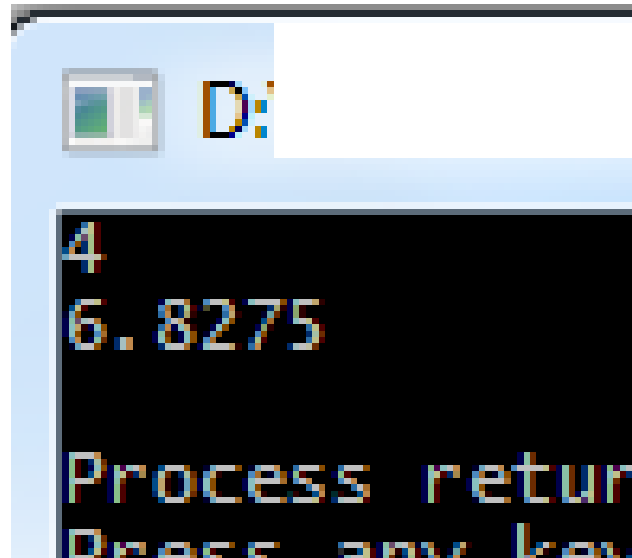
```
template<class T1=double, class T2=int>
```

## Приклад 2

```
#include <iostream>
using namespace std;

template <class T>
T average(T *array, int length)
{
    T sum = 0;
    for (int count=0; count < length;
        ++count)
        sum += array[count];
    sum /= length;
    return sum;
}
```

```
int main()
{
    int array1[5] = { 6, 4, 1, 3, 7 };
    cout << average(array1, 5) << '\n';
    double array2[4] = { 4.25, 5.37, 8.44,
9.25 };
    cout << average(array2, 4) << '\n';
    return 0;
}
```





# Шаблони класів

**Шаблони класів** – узагальнений опис класу (типу даних, який створює користувач), в якому можуть бути параметризовані атрибути і операції типу. Являють собою конструкції, за якими можуть бути згенеровані дійсні класи шляхом підстановки замість параметрів конкретних аргументів.

Створення *шаблону класу* аналогічно створенню шаблону функції:

```
template <class T> //або список параметрів  
//також можуть бути параметри за замовчуванням  
class Ім'я  
{ ... };
```

*Інстанціювання шаблону класа* – створення конкретних класів з шаблону

## Приклад 3

```
#include <iostream>
using namespace std;
template <class T>
class Point
{private:
    T x, y;
public:
    Point(T x, T y)
    {
        this->x=x; this->y=y;
    }
    void show();
};
```

//при багатофайловому проекті визначення методів шаблону краще розміщувати в заголовковому файлі

**template <class T>** //визначення методів поза шаблона класу

**void Point<T>::show()**

11

```
{  
    cout<<"type x: "<<typeid(x).name()<<endl;  
    cout<<"type y: "<<typeid(y).name()<<endl;  
    cout<<"x= "<<x<<" y= "<<y<<endl;  
}
```

int main()

{//створюючи об'єкти класу,

визначаємо тип параметра шаблону

**Point<int>** a(1,1);

a.show();

**Point<double>** b(2.2,1.1);

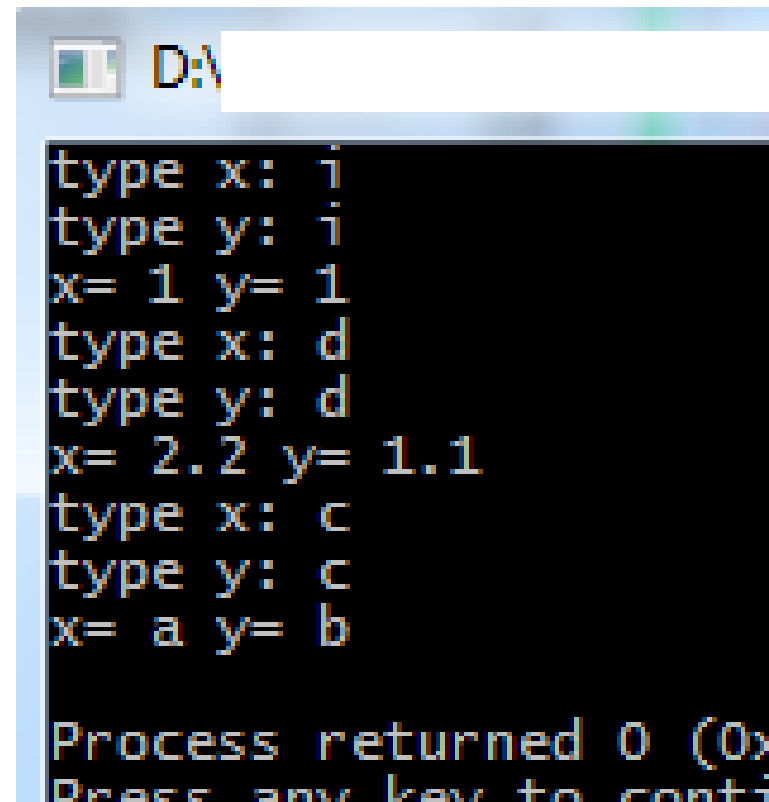
b.show();

**Point<char>** c('a','b');

c.show();

return 0;

}



```
D:\...  
type x: i  
type y: i  
x= 1 y= 1  
type x: d  
type y: d  
x= 2.2 y= 1.1  
type x: c  
type y: c  
x= a y= b  
  
Process returned 0 (0x0)  
Press any key to continue . . .
```

## Спеціалізація шаблону

При програмуванні іноді буває потрібно робити спеціалізовану версію шаблону.

**Спеціалізація шаблону** буває:

- **повною**, якщо конкретизовані всі аргументи шаблону;
- **частковою**, вказується тільки частина шаблону, наприклад спеціалізація його методів.

При оголошені шаблону та всіх його спеціалізованих версій повинен дотримуватися порядок слідування: спочатку шаблон, далі його спеціалізації.

...

**template <class T>** //створення шаблону класу

**class Point**

{private:

**T** x, y;

public:

Point(**T** x, **T** y)

{ this->x=x; this->y=y; }

void show();

};

**template <class T>** //визначення метода поза шаблоном

**void Point<T>::show()**

{cout<<"type x: "<<typeid(x).name()<<endl;

cout<<"type y: "<<typeid(y).name()<<endl;

cout<<"x= "<<x<<" y= "<<y<<endl;

}

```
template <> //повна спеціалізація шаблону
class Point<string>
{private:
    string x,y;
public:
    Point(string x, string y)
    { this->x=x; this->y=y;}
    void show();
};
```

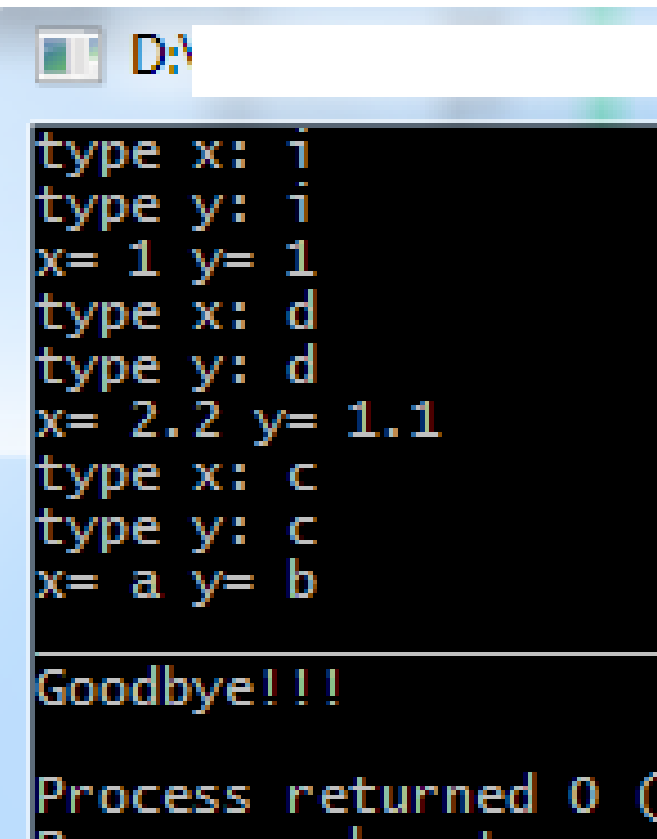
// визначення метода поза спеціалізації шаблону

```
void Point<string>::show()
{ cout<<"_____ "<<endl;
  cout<<x<<y<<endl;
}
```

```

int main()
{
    Point<int> a(1,1);
    a.show();
    Point<double> b(2.2,1.1);
    b.show();
    Point<char> c('a','b');
    c.show();
    Point<string> d("Good","bye!!!");
    d.show();
    return 0;
}

```



```

D:\
type x: i
type y: i
x= 1 y= 1
type x: d
type y: d
x= 2.2 y= 1.1
type x: c
type y: c
x= a y= b
Goodbye!!!
Process returned 0 (

```

## Часткова спеціалізація шаблону класу

При частковій спеціалізації вказується тільки частина шаблону, наприклад спеціалізація його методів.

```
template <> //часткова спеціалізація шаблону.
```

```
//Якщо є ще одим тип, або параметр в шаблоні,
```

```
//він вказується в <>
```

```
void Point<string>::show()
```

```
{
```

```
    cout<<"_____ "<<endl;
```

```
    cout<<x<<y<<endl;
```

```
}
```



## Параметр non-type в шаблоні класу

Параметр **non-type** в шаблоні - це спеціальний параметр шаблону, який замінюється не типом даних, а конкретним значенням.

Цим значенням може бути:

- цілочисельне значення;
- покажчик або посилання на об'єкт класу;
- покажчик або посилання на функцію;

Наприклад:

```
template <class T, int size>
```

// size являється параметром non-type в шаблоні класу

//параметри можуть бути у тому числі за замовчуванням

```
template <class T=int, int size=10>
```

## Приклад 5

```
#include <iostream>
using namespace std;

template <class T=int, int size=10>
// size відповідає за розмір масиву
class StaticArray
{ private:
    T m_array[size];
public:
    T* getArray()
    { return m_array;
    }
    T& operator[] (int index)
    { return m_array[index];
    }
};
```

```
int main()  
{
```

//Оголошуємо масив типу int 10-ти елементів (за замовч)

```
    StaticArray<> intArray;
```

// Заповнюємо масив значеннями

```
    for (int count=0; count < 10; ++count)  
        intArray[count] = count;
```

// виведення елементів

```
    for (int count=9; count >= 0; --count)  
        cout << intArray[count] << " ";  
    cout << '\n';
```

// Оголошуємо масив типу double з 5-ти елементів

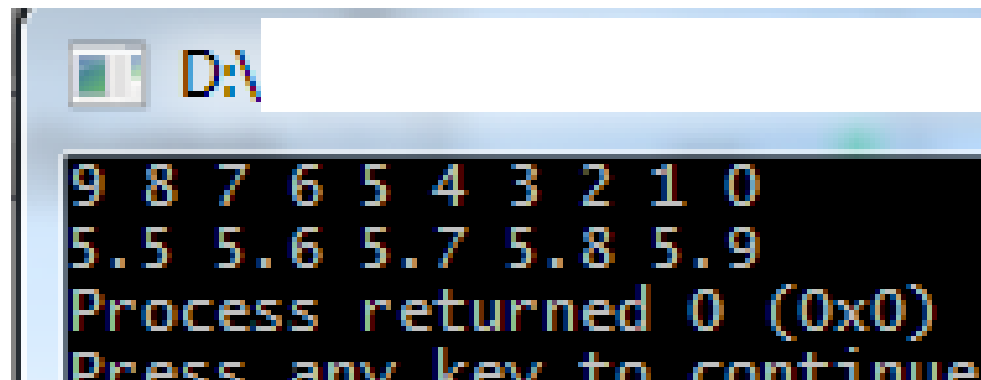
```
StaticArray<double, 5> doubleArray;
```

// Заповнення та виведення масиву

```
for (int count=0; count < 5; ++count)  
    doubleArray[count] = 5.5 + 0.1*count;
```

```
for (int count=0; count < 5; ++count)  
    cout << doubleArray[count] << ' ';
```

```
return 0;  
}
```



```
D:\  
9 8 7 6 5 4 3 2 1 0  
5.5 5.6 5.7 5.8 5.9  
Process returned 0 (0x0)  
Press any key to continue
```

## Статичні елементи шаблонів

В шаблоні класу можна оголошувати статичні елементи. Тоді при різних варіантах параметрів шаблону створюються різні класи зі своїми копіями статичних полів та методів.

Наприклад:

```
template <class T>
class StaticClass
{ static T n;
  ... };
```

...

```
template <int> StaticClass <int> :: n=5;
```

//спеціалізація статичного поля

## Успадкування шаблонів класів

- Шаблон може бути успадкований від простого класу.
- Простий клас може бути нащадком шаблону.
- Також шаблон класу може бути успадкований від шаблону класу.

Наприклад:

```
template <class T>
```

```
class Base
```

```
{ ... };
```

```
template <class T>
```

```
class Derive: public class Base <T>
```

```
{ ... };
```

## Приклад 6

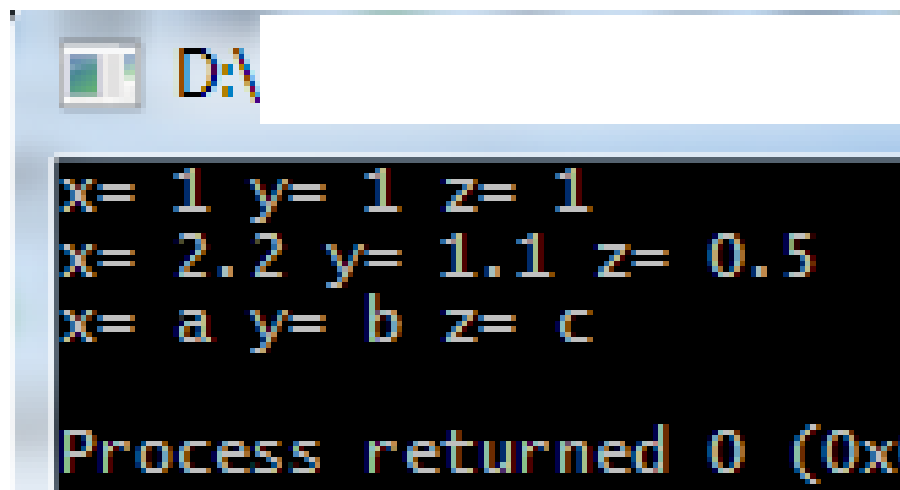
```
#include <iostream>
using namespace std;

template <class T>
class Point
{protected:
    T x, y;
public:
    Point(T x, T y)
    {
        this->x=x; this->y=y;
    }
    void show()
    {
        cout<<"x= "<<x<<" y= "<<y<<endl;
    }
};
```

```
template <class T>//створення похідного класу
class Point3D:public Point<T>
{private:
    T z;
public:
    Point3D(T x, T y, T z):Point<T>(x,y)
    {
        this->z=z;
    }
    void show()
    {
        cout<<"x= "<<this->x<<" y= "<<
            this->y<<" z= "<<this->z<<endl;
    }
};
```



```
int main()  
{  
    Point3D <int> a(1,1,1);  
    a.show();  
    Point3D <double> b(2.2,1.1,0.5);  
    b.show();  
    Point3D <char> c('a','b','c');  
    c.show();  
  
    return 0;  
}
```



```
D:\  
x= 1 y= 1 z= 1  
x= 2.2 y= 1.1 z= 0.5  
x= a y= b z= c  
Process returned 0 (0x0)
```

Дякую за увагу!