Шаблони

Класът `MyVector` е колекция от `int`. Какво бихме направили, ако искаме вместо `int` да имаме `MyVector`, който държи `char` ?

Единият вариант е да направим нов клас `MyVector_char`, който да изпълни задачата.

А ако трябва вместо примитивни типове (`int`, `char`, `double`, `string`) трябва да направим `MyVector` за някой друг клас (примерно `Rational`) ?

C++ ни дава начин да създаваме т.нар. **шаблонни функции и класове**, които ни позволяват да напишем кода за неизвестен тип, и след това той да се замести с желаният от нас тип. Този код не се компилира. При поискване на даден тип, тогава се случва заместването на кода.

Шаблонни функции

Нека имаме следната задача - да се напише **една** функция, която събира две числа. Числата са от тип `int`, `double` и `Rational`.

За целта ще създадем шаблонна функция `add()`. Когато искаме да зададем шаблонен тип, го указваме по следният начин:

```
template <class T>
```

И навсякъде в кода, където искаме да използваме този тип, пишем `T`

```
template <class T>

T add(T first, T second)
{
    T result = first + second;
    return result;
}
```

Извикването на функцията става по следният начин:

```
int int_one = 5, int_two = 4;
double double_one = 3.213, double_two = 6.123;
Rational rational_one(2,3), rational_two(1,3);

std::cout << add(int_one, int_two) << std::endl;
std::cout << add(double_one, double_two) << std::endl;
std::cout << add(rational_one, rational_two) << std::endl;
```

Забележка: класът `Rational` има предефинирани оператори + и <<

Шаблонни класове

Същото нещо може да се приложи и към класовете. Когато се използват шаблонии класове или функции, този код не се компилира. Когато извикаме функцията или класа с конкретен код, се генерира кода на функцията или класа **само за този тип**.

Поради този факт, е прието този тип класове да се пишат в header файлове (.h или .hpp) (Малко допълнение - за целта на курса е прието да се пише в .h файлове, но те отговарят на C header файлове. Правилното разширение на C++ header файлове е .hpp)

Когато искаме да използваме клас с конкретен тип, а не шаблонния, това се случва със следният синтаксис: `Class_A<int>`, т.е. името на класа вече има < > в себе си.

Декларация на шаблонен клас

За да използваме шаблонен тип, трябва преди декларацията на класа да напишем `template <class T>` и след това навсякъде където искаме да ползваме този тип, използваме `T`. Важно е да се отбележи, че когато искаме типа на някой от методите да е инстанция на класа (operator=), типа на класа е Class.

Декларацията на шаблонния клас `MyVector` изглежда така:

```
template <class T>
class MyVector
{
    private:
        T* values;
        int size;
        int maxSize;

        void resize();
    public:
        MyVector(); //Default
        MyVector(const MyVector&); //Copy
        MyVector<T>& operator=(const MyVector&); //Operator=

        void set_element(int, T); //pos, val
        void push_back(T);
        void insert(int, T); //pos, val
        void remove(int); //pos
        void print() const;

        T get_element(int) const;
        T pop_back();
        T begin() const;
        T end() const;
        int get_size() const;
        T operator[](int) const;

        friend std::ostream& operator << (std::ostream& out, const MyVector&);
        friend std::istream& operator >> (std::istream& in, MyVector&);
        ~MyVector();
};
```

Навсякъде където имахме `int`, го заменихме с `T`.

Дефиниция на шаблонен клас

Има два начина да се напишат дефинициите на шаблонен клас - където са декларациите, или отделно, но отново в същият файл:

Дефинициите и декларациите са на едно

```
public:
...
void push_back(T val)
{
    if(size == maxSize)
        resize();
    values[size] = val;
    size++;
}
...
```

Дефинициите са разделени от декларациите

```
template <class T>
void MyVector<T>::push_back(T val)
{
    if(size == maxSize)
        resize();

    values[size] = val;
    size++;
}
```

Забелязваме, че ако дефиницията е разделена от декларациите, трябва преди всеки метод да напишем `template <class T>`

Използването на нашият шаблонен `MyVector` става по следния начин:

```
MyVector<int> a;
MyVector<std::string> b;
MyVector<char> c;
```