Шаблонизация

Шаблоны – создание универсальных функций или классов, которые будут работать с любыми типами данных

Пример встроенных шаблонов – vector<>, array<>, это и есть шаблоны, и нужный тип данных мы указываем при их создании внутри <char>

Примеры, если забыли

```
Vector<int> mas1={1, 2, 3, 4};
Vector<char> mas2={‘q’, ’w’, ’e’, ’r’};
Vector<Button> mas3;
```

Выделенное зеленым это как раз то, что мы можем сделать с помощью шаблонов

То есть шаблоны нужны чтобы сделать функцию/класс универсальным для любых типов данных.

С функциями мы могли сделать это с помощью перегрузки, но если функция делает одно и то же, с разными типами данных, то это сильно увеличивает объем кода

Например:

```
1  // перегрузка функции printArray для вывода массива на экран
2  void printArray(const int * array, int count)
3  {
4      for (int ix = 0; ix < count; ix++)
5          cout << array[ix] << "   ";
6      cout << endl;
7  }
8
9  void printArray(const double * array, int count)
10 {
11     for (int ix = 0; ix < count; ix++)
12         cout << array[ix] << "   ";
13     cout << endl;
14 }
15
16 void printArray(const float * array, int count)
17 {
18     for (int ix = 0; ix < count; ix++)
19         cout << array[ix] << "   ";
20     cout << endl;
21 }
22
23 void printArray(const char * array, int count)
24 {
25     for (int ix = 0; ix < count; ix++)
26         cout << array[ix] << "   ";
27     cout << endl;
28 }
```

Чтобы упростить , мы можем создать шаблон, в котором описываем все типы данных(C++)

```
template <“параметры данных шаблона”>
“сама функция или класс, как обычно”

template <typename T>
void printArray(T *array, int count){
    for(int i=0; i<count; i++){ cout<<array[i]<<‘ ‘;}
    cout<<endl;
}
```

Использование:

```
printArray<int>( {1,2,3,4} , 4);//функция типа int
printArray<char>( {‘q’, ’w’, ‘e’} , 3); //тип char
auto printArray( {1.5, 5.1, 2.4} , 3);//автоматически решает
```

Как параметры мы можем использовать ключевые слова typename или class и дальше любые название этих шаблонных типов данных

template<typename Type1, typename Type2>...

template<class C1>...

template<typename T1, typename T2, class C1>...

Пример сортировки пузырьком с шаблоном и суммы 2х чисел

```
template< class T >
T Add(T n1, T n2)
{
    T result;
    result = n1 + n2;

    return result;
}
```

```
template < class ElementType > //Использовал class, но можно и typename - без разницы
void bubbleSort(ElementType * arr, size_t arrSize)
{
    for(size_t i = 0; i < arrSize - 1; ++i)
        for(size_t j = 0; j < arrSize - 1; ++j)
            if (arr[j + 1] < arr[j])
                my_swap ( arr[j] , arr[j+1] ) ;
}
```

С классами все то же самое, мы используем типы данных из шаблона в классе, чтобы сделать его универсальным. Например те же классы vector и array, в которых мы можем использовать массивы любых типов данных, используя один и тот же класс

Пример классов без шаблонов, в каждом свой тип переменной

```
4  class IntAccount {
5  private:
6      int id;
7  public:
8      IntAccount(int id) : id(id)
9      { }
10     int getId() {
11         return id;
12     }
13 };
```

```
15 class StrAccount {
16 private:
17     std::string id;
18 public:
19     StrAccount(std::string id) : id(id)
20     { }
21     std::string getId() {
22         return id;
23     }
24 };
```

А теперь то же самое с шаблоном, который будет работать для любых типов данных

```
1  template <typename T>
2  class Account {
3  private:
4      T id;
5  public:
6      Account(T id) : id(id)
7      { }
8      T getId() {
9          return id;
10     }
11 };
```

Шаблонные классы обычно нужны для хранения разных типов данных в массивах(как vector) или их обработки(например универсальный класс для реальзации стека, то есть 2х функций pop и push)

Создавать объекты шаблонных классов можно аналогично vector, например для нашего класса Account:

```
Account<int> acc1(5);
Account<char> acc2('q');
Account<Account<int>>(new Account(1));
```

В C# использование и смысл шаблонов тот же, но записываются они даже проще, просто сразу после имени функции/класса в скобках <>

```
 9  using System;
10
11  class HelloWorld {
12    static void Main() {
13      Account<string> ac=new Account<string>("qwerty");
14      Console.WriteLine(ac.get_Id());
15    }
16  }
17
18  class Account<T>{
19      private T id;
20      public Account(T id){this.id=id;}
21      public T get_Id(){return id;}
22  }
```