

## Шаблини функцій

```
1 int max(int a, int b)
2 {
3     return (a > b) ? a : b;
4 }
```

```
1 double max(double a, double b)
2 {
3     return (a > b) ? a : b;
4 }
```

```
1 T max(T a, T b)
2 {
3     return (a > b) ? a : b;
4 }
```

```
1 template <typename T> // объявление параметра шаблона функции
2 T max(T a, T b)
3 {
4     return (a > b) ? a : b;
5 }
```

```
1 template <typename T1, typename T2>
2 // Шаблин функції здесь
```

```
1 template <typename T>
2 const T& max(const T& a, const T& b)
3 {
4     return (a > b) ? a : b;
5 }
```

```
1 #include <iostream>
2
3 template <typename T>
4 const T& max(const T& a, const T& b)
5 {
6     return (a > b) ? a : b;
7 }
8
9 int main()
10 {
11     int i = max(4, 8);
12     std::cout << i << '\n';
13
14     double d = max(7.56, 21.434);
15     std::cout << d << '\n';
16
17     char ch = max('b', '9');
18     std::cout << ch << '\n';
19
20     return 0;
21 }
```

результат:

8

21.434

b

*Примітка:* Стандартна бібліотека C++ має в своєму арсеналі шаблон функції `max()` (який знаходиться в заголовку `<algorithm>`), тому ви можете не реалізовувати цю функцію вручну в майбутньому. Плюс, якщо ви пишете свої власні шаблони функцій і використовуєте стейтмент **using namespace std;**, то не забувайте про можливість виникнення конфліктів імен, так як компілятор не зможе визначити, чи хочете ви використовувати свою версію функції `max()` або версію `std::max()`.

<http://www.cplusplus.com/reference/algorithm/>

<https://ru.cppreference.com/w/cpp/algorithm>

### Примірки шаблонів функцій

```
1 template <typename T> // объявление параметра шаблона функции
2 const T& max(const T& a, const T& b)
3 {
4     return (a > b) ? a : b;
5 }
```

```
1 int i = max(4, 8); // вызывается max(int, int)
```

```
1 const int& max(const int &a, const int &b)
2 {
3     return (a > b) ? a : b;
4 }
```

```
1 double d = max(7.58, 19.378); // вызывается max(double, double)
```

```
1 const double& max(const double &a, const double &b)
2 {
3     return (a > b) ? a : b;
4 }
```

### Оператори, виклики функцій і шаблони функцій

```
1 class Dollars
2 {
3 private:
4     int m_dollars;
5 public:
6     Dollars(int dollars)
7         : m_dollars(dollars)
8     {
9     }
10 };
```

```

1  template <typename T> // объявление параметра шаблона функции
2  const T& max(const T& a, const T& b)
3  {
4      return (a > b) ? a : b;
5  }
6
7  class Dollars
8  {
9  private:
10     int m_dollars;
11 public:
12     Dollars(int dollars)
13         : m_dollars(dollars)
14     {
15     }
16 };
17
18 int main()
19 {
20     Dollars seven(7);
21     Dollars twelve(12);
22
23     Dollars bigger = max(seven, twelve);
24
25     return 0;
26 }

```

```

1  const Dollars& max(const Dollars &a, const Dollars &b)
2  {
3      return (a > b) ? a : b;
4  }

```

Ошибка C2676 бинарный ">": "const T" не определяет этот оператор или преобразование к типу приемлемо к встроенному оператору

```

1  class Dollars
2  {
3  private:
4      int m_dollars;
5  public:
6      Dollars(int dollars)
7          : m_dollars(dollars)
8      {
9      }
10
11     friend bool operator>(const Dollars &d1, const Dollars &d2)
12     {
13         return (d1.m_dollars > d2.m_dollars);
14     }
15 };

```

Шаблон функції, яка обчислює середнє арифметичне елементів масиву:

```
1  #include <iostream>
2
3  template <class T>
4  T average(T *array, int length)
5  {
6      T sum = 0;
7      for (int count=0; count < length; ++count)
8          sum += array[count];
9
10     sum /= length;
11     return sum;
12 }
13
14 int main()
15 {
16     int array1[] = { 6, 4, 1, 3, 7 };
17     std::cout << average(array1, 5) << '\n';
18
19     double array2[] = { 4.25, 5.37, 8.44, 9.25 };
20     std::cout << average(array2, 4) << '\n';
21
22     return 0;
23 }
```

результат:

4

6.8275

```

1  #include <iostream>
2
3  class Dollars
4  {
5  private:
6      int m_dollars;
7  public:
8      Dollars(int dollars)
9          : m_dollars(dollars)
10     {
11     }
12
13     friend bool operator>(const Dollars &d1, const Dollars &d2)
14     {
15         return (d1.m_dollars > d2.m_dollars);
16     }
17 };
18
19 template <class T>
20 T average(T *array, int length)
21 {
22     T sum = 0;
23     for (int count=0; count < length; ++count)
24         sum += array[count];
25
26     sum /= length;
27     return sum;
28 }
29
30 int main()
31 {
32     Dollars array3[] = { Dollars(7), Dollars(12), Dollars(18), Dollars(15) };
33     std::cout << average(array3, 4) << '\n';
34
35     return 0;
36 }

```

Ошибка E0349 отсутствует оператор "<<", соответствующий этим операндам

Ошибка C2679 бинарный "<<": не найден оператор, принимающий правый операнд типа "T" (или приемлемое преобразование отсутствует)

```

1  template <class T>
2  Dollars average(Dollars *array, int length)
3  {
4      Dollars sum = 0;
5      for (int count=0; count < length; ++count)
6          sum += array[count];
7
8      sum /= length;
9      return sum;
10 }

```

```
sum += array[count];
```

```
1 class Dollars
2 {
3 private:
4     int m_dollars;
5 public:
6     Dollars(int dollars)
7         : m_dollars(dollars)
8     {
9     }
10
11     friend bool operator>(const Dollars &d1, const Dollars &d2)
12     {
13         return (d1.m_dollars > d2.m_dollars);
14     }
15
16     friend std::ostream& operator<< (std::ostream &out, const Dollars &dollars)
17     {
18         out << dollars.m_dollars << " dollars ";
19         return out;
20     }
21
22     Dollars& operator+=(Dollars dollars)
23     {
24         m_dollars += dollars.m_dollars;
25         return *this;
26     }
27
28     Dollars& operator/=(int value)
29     {
30         m_dollars /= value;
31         return *this;
32     }
33 };
```

13 dollars

## Шаблоны классов

Array.h:

```
1  #ifndef ARRAY_H
2  #define ARRAY_H
3
4  #include <assert.h> // для assert()
5
6  template <class T> // это шаблон класса с T
7  class Array
8  {
9  private:
10     int m_length;
11     T *m_data;
12
13 public:
14     Array()
15     {
16         m_length = 0;
17         m_data = nullptr;
18     }
19
20     Array(int length)
21     {
22         m_data = new T[length];
23         m_length = length;
24     }
25
26     ~Array()
27     {
28         delete[] m_data;
29     }
30
31     void Erase()
32     {
33         delete[] m_data;
34         // Присваиваем значение nullptr для m_data,
35         m_data = nullptr;
36         m_length = 0;
37     }
38
39     T& operator[](int index)
40     {
41         assert(index >= 0 && index < m_length);
42         return m_data[index];
43     }
44
45     // Длина массива всегда является целочисленным
46     // она не зависит от типа элементов массива
47     int getLength(); // определяем метод и шаблон м
48 };
49
50
51 template <typename T> // метод, определённый вне те
52 int Array<T>::getLength() { return m_length; } // о
53
54 #endif
```

```
1  #include <iostream>
2  #include "Array.h"
3
4  int main()
5  {
6      Array<int> intArray(10);
7      Array<double> doubleArray(10);
8
9      for (int count = 0; count < intArray.getLength(); ++count)
10     {
11         intArray[count] = count;
12         doubleArray[count] = count + 0.5;
13     }
14
15     for (int count = intArray.getLength()-1; count >= 0; --count)
16         std::cout << intArray[count] << "\t" << doubleArray[count] << '\n';
17
18     return 0;
19 }
```

результат:

9	9.5
8	8.5
7	7.5
6	6.5
5	5.5
4	4.5
3	3.5
2	2.5
1	1.5
0	0.5

## Шаблони класів і Заголовки

Array.h:

```
1 #ifndef ARRAY_H
2 #define ARRAY_H
3
4 #include <assert.h> // для assert()
5
6 template <class T>
7 class Array
8 {
9 private:
10     int m_length;
11     T *m_data;
12
13 public:
14     Array()
15     {
16         m_length = 0;
17         m_data = nullptr;
18     }
19
20     Array(int length)
21     {
22         m_data = new T[length];
23         m_length = length;
24     }
25
26     ~Array()
27     {
28         delete[] m_data;
29     }
30
31     void Erase()
32     {
33         delete[] m_data;
34         // Присваиваем значение nullptr для m_data, чтобы на выходе не получить висячий указатель!
35         m_data = nullptr;
36         m_length = 0;
37     }
38
39     T& operator[](int index)
40     {
41         {
42             assert(index >= 0 && index < m_length);
43             return m_data[index];
44         }
45
46         // Длина массива всегда является целочисленным значением,
47         // она не зависит от типа элементов массива
48         int getLength();
49 };
50
```

Array.cpp:

```
1 #include "Array.h"
2
3 template <typename T>
4 int Array<T>::getLength() { return m_length; }
```



main.cpp:

```
1 #include "Array.h"
2
3 int main()
4 {
5     Array<int> intArray(10);
6     Array<double> doubleArray(10);
7
8     for (int count = 0; count < intArray.getLength(); ++count)
9     {
10         intArray[count] = count;
11         doubleArray[count] = count + 0.5;
12     }
13
14     for (int count = intArray.getLength()-1; count >= 0; --count)
15         std::cout << intArray[count] << "\t" << doubleArray[count] << '\n';
16
17     return 0;
18 }
```

Програма вище скомпілюється, але викличе таку помилку линкера:

```
unresolved      external      symbol      "public:      int      __thiscall
Array::getLength(void)"      (?GetLength@?$Array@H@@@QAEHXZ)
```

templates.cpp:

```
1 // Таким образом, мы гарантируем, что компилятор увидит полное определение шаблона класса Array
2 #include "Array.h"
3 #include "Array.cpp" // мы нарушаем правила хорошего тона программирования, но только в этом месте
4
5 // Здесь вы #include другие файлы .h и .cpp с определениями шаблонов, которые вам нужны
6
7 template class Array<int>; // явно создаём экземпляр шаблона класса Array<int>
8 template class Array<double>; // явно создаём экземпляр шаблона класса Array<double>
9
10 // Здесь вы явно создаёте другие экземпляры шаблонов, которые вам нужны
```

## Параметр non-type

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

Створимо шаблон класу StaticArray, який використовує як параметр типу, так і параметр non-type. Параметр типу відповідає за тип даних елементів статичного масиву, а параметр non-type відповідає за розмір виділяється масиву:

```

1  #include <iostream>
2
3  template <class T, int size> // size является параметром non-type в шаблоне класса
4  class StaticArray
5  {
6  private:
7      // Параметр non-type в шаблоне класса отвечает за размер выделяемого массива
8      T m_array[size];
9
10 public:
11     T* getArray();
12
13     T& operator[](int index)
14     {
15         return m_array[index];
16     }
17 };
18
19 // Синтаксис определения шаблона метода и самого метода вне тела класса с параметром non-type
20 template <class T, int size>
21 T* StaticArray<T, size>::getArray()
22 {
23     return m_array;
24 }

```

```

26 int main()
27 {
28     // Объявляем целочисленный массив из 10 элементов
29     StaticArray<int, 10> intArray;
30
31     // Заполняем массив значениями
32     for (int count=0; count < 10; ++count)
33         intArray[count] = count;
34
35     // Выводим элементы массива в обратном порядке
36     for (int count=9; count >= 0; --count)
37         std::cout << intArray[count] << " ";
38     std::cout << '\n';
39
40     // Объявляем массив типа double из 5 элементов
41     StaticArray<double, 5> doubleArray;
42
43     // Заполняем массив значениями
44     for (int count=0; count < 5; ++count)
45         doubleArray[count] = 5.5 + 0.1*count;
46
47     // Выводим элементы массива
48     for (int count=0; count < 5; ++count)
49         std::cout << doubleArray[count] << ' ';
50
51     return 0;
52 }

```

Результат виконання програми вище:

```

9 8 7 6 5 4 3 2 1 0
5.5 5.6 5.7 5.8 5.9

```

## Явна спеціалізація шаблону функції

```
1 template <class T>
2 class Repository
3 {
4     private:
5         T m_value;
6     public:
7         Repository(T value)
8         {
9             m_value = value;
10        }
11
12        ~Repository()
13        {
14        }
15
16        void print()
17        {
18            std::cout << m_value << '\n';
19        }
20 };

1 int main()
2 {
3     // Инициализируем объекты класса
4     Repository<int> nValue(7);
5     Repository<double> dValue(8.4);
6
7     // Выводим значения объектов класса
8     nValue.print();
9     dValue.print();
10 }
```

результат:

7  
8.4

Частина `template <>` повідомляє компілятору, що це шаблон функції, але без параметрів (так як в цьому випадку явно вказуємо потрібний тип даних).

```
1 template <>
2 void Repository<double>::print()
3 {
4     std::cout << std::scientific << m_value << '\n';
5 }
```

Результат виконання програми вище:

7  
8.400000e+00