

C++

Некоторые стандартные
шаблоны классов

Класс string

```
#include <string>
using std::string;
```

заголовки без .h содержат
описания пространств имен и
другую специфическую
информацию в стиле C++

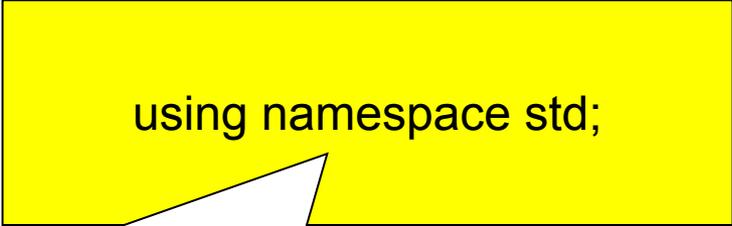
```
string s1; // Инициализация пустой строкой
string s2(s1); // Инициализация копией строки s1
string s3("value"); // Инициализация копией литерала
string s4(n, 'c'); // n символов 'c'
```

Класс string

```
#include "stdafx.h"
```

```
using std::cout;  
using std::cin;  
using std::string;  
using std::endl;
```

```
int main()  
{  
    string s;  
    cin >> s;  
    cout << s << endl;  
    return 0;  
}
```



using namespace std;

Класс string

```
int main()
{
    string::size_type n;
    string s;
    cin >> s;
    if (s.empty()) cout << "string is empty!";
    else
        { n=s.size();
          cout << n << endl;
        }
    return 0;
}
```

Операции со строками

<code>s.empty()</code>	Возвращает true, если строка пуста
<code>s.size()</code>	Возвращает количество символов в строке
<code>s[n]</code>	Возвращает n-ый символ строки
<code>s1 + s2</code>	Возвращает «склейку» строк s1 и s2
<code>s1 = s2</code>	Заменяет символы строки s1 строкой s2
<code>s1 == s2</code>	Проверяет совпадение строк
<code>!=, <, <=, >, >=</code>	Имеют обычное значение

Класс string

```
string s1("Hello");  
string s2("World!");  
s=s1+", "+s2;  
cout << s << endl << s[s.size()-1] << endl;;
```

s = "abc"+"efg"; // так нельзя!

s[s.size] // непредсказуемая ошибка

Работа с символами строки

<code>isalnum(c)</code>	Возвращает true, если c буква или цифра
<code>isalpha(c)</code>	c – буква
<code>isdigit(c)</code>	c – цифра
<code>ispunct(c)</code>	Знак пунктуации
<code>isspace(c)</code>	пробел
<code>isgraph(c)</code>	печатаемый символ
<code>islower(c)</code>	символ в нижнем регистре
<code>isxdigit(c)</code>	c – шестнадцатичная цифра
<code>toupper(c)</code>	возвращает прописную букву
<code>tolower(c)</code>	возвращает букву в нижнем регистре

Шаблон класса `vector`

```
vector<T> v1;      // Вектор, содержащий объекты типа T
vector<T> v2 (v1); // Вектор v2 - копия v1
vector<T> v3 (n, i); // Вектор из n элементов со знач. i
vector<T> v4 (n);  // Вектор из n элементов
```

Размер вектора увеличивается динамически

Шаблон класса vector

```
#include <vector>
```

```
using std::vector;
```

```
using std::string;
```

```
using std::cout;
```

```
using std::endl;
```

```
void main()
```

```
{
```

```
    vector<int> a(100,0);
```

```
    vector<int> c;
```

```
    vector<int> b(a);
```

```
    vector<string> s(10,"Hello!");
```

```
    cout << s[0] << endl;
```

```
}
```

Операции с векторами

<code>v.empty()</code>	Возвращает true, если вектор пуст
<code>v.size()</code>	Возвращает количество элементов
<code>v[n]</code>	Возвращает n-й элемент вектора
<code>v.push_back(t)</code>	Добавляет элемент t в конец вектора
<code>v1 = v2</code>	Заменяет элементы вектора v1 копиями элементов вектора v2
<code>s1 == s2</code>	Проверяет элементов совпадение векторов
<code>!=, <, <=, >, >=</code>	Имеют обычное значение

Динамическое добавление элементов

```
#include <vector>

using std::vector;
using std::cout;
using std::endl;

void main()
{

    vector<int> c;
    cout << c.size() << endl;
    c.push_back(10);
    c.push_back(20);
    for (int i=0; i!=c.size(); i++)
        cout << c[i] << ' ';
    // c[2]=30 - так нельзя!
}
```

Итераторы

```
#include <vector>
using std::vector;
using std::string;
using std::cout;
using std::endl;

void main()
{
    vector<int> c(10);
    for (int i=0; i!=c.size(); i++) c[i]=i*10;

    vector<int>::iterator ic;

    for (ic=c.begin(); ic!=c.end(); ic++)
        cout << *ic << endl;
}
```

Шаблон класса `bitset`

```
bitset<n> b;      // Набор из n нулевых битов
bitset<n> b(u);   // Копия значения unsigned long
bitset<n> b(s);   // Биты из текстовой строки "10011"
bitset<n> b(s, pos, n); // из n символов текстовой
                        // строки, начиная с pos.
```

```
bitset<16> b(0xFFFF);
```

```
bitset<32> d("0111110000110111");
```

Операции с наборами битов

<code>b.any()</code>	Все ли биты установлены
<code>b.none()</code>	Нет ли в наборе установленных битов
<code>b.count()</code>	Число установленных битов
<code>b.size()</code>	Число битов в наборе
<code>b[pos]</code>	Доступ к биту с номером <code>pos</code>
<code>b.reset()</code>	Сброс всех битов
<code>b.set()</code>	Установка всех битов
<code>b.flip()</code>	Инвертирует все биты
<code>b.flip(pos)</code>	Инвертирует бит
<code>b.to_ulong()</code>	Возвращает <code>unsigned long</code> с теми же битами

Примеры работы с набором бит

```
#include <iostream>
#include <bitset>

using std::bitset;
using std::string;
using std::cout;
using std::endl;

void main()
{
    bitset<32> b1 (0xFFFFFFFF) ;
    b1[10]=0;
    b1.flip(1);
    b1.flip();
    cout << b1 << endl;
    cout << b1.count() << endl;
    cout << b1.to_ulong() << endl;
}
```

Дополнительные стандартные классы

- Контейнеры
 - **vector** – быстрый произвольный доступ
 - **list** – быстрая вставка удаление
 - **deque** – двухсторонняя очередь
 - **stack** – стек, последним пришел, первым вышел
 - **queue** – очередь, первым пришел, последним вышел
 - **priority_queue** – приоритетная очередь
- Ассоциативные контейнеры
 - Тип **map**
 - Тип **set**
 - Типы **multimap** и **multiset**

Приведение типов

- `static_cast`
- `dynamic_cast`
- `const_cast`
- `reinterpret_cast`
- В старом стиле (два варианта)

static_cast<T> (x)

Выполняет преобразование типов, которое компилятор может выполнить неявно, а также которое не может выполнить неявно

```
void main()  
{  
    double d = 3.14;  
    int     n = 0;  
    n = static_cast<int> (d);  
  
    void *p = &d;  
    double *dp = static_cast<double *> (p);  
}
```

const_cast<T> (x)

Преобразование констант

```
const char *s;  
char *p =string_copy(const_cast<char *> (p) );
```

reinterpret_cast<T> (x)

Машинно-зависимая интерпретация бит

```
char c = 'A';
```

```
    unsigned short int *w =  
reinterpret_cast<unsigned short int*> (&c);
```

```
    cout << *w << endl;
```

Приведение типов в старом стиле

```
char *pc = (char *) ip;
```

```
double d;
```

```
int n = int(d);
```

Перегрузка операторов преобразования

```
class SmallInt
{
public:
    int val;
    SmallInt(int i=0)
    {
        val=i;
    }
    operator int() { return val; }
};
```

Эта функция будет использоваться и в операторах явного преобразования типов

Преобразование классов

```
class Aaa
{ public: int a;
};

class Bbb : public Aaa
{ public: int b;
};

void main()
{ Aaa* a = new Aaa();
  Bbb* b = new Bbb();

  try
  { a = dynamic_cast<Aaa*> (b);
  }
  catch(bad_cast)
  { cout << "Error in " << __FILE__
    << ", line: " << __LINE__ << endl;
  }
  cout << typeid(a).name() << endl;
}
```

Размещаемый оператор new

- new (адрес размещения) тип;
- new (адрес размещения) тип(параметры);

Встраиваемые функции

```
inline double sqr(double x)
{ return x*x;
}
```

Помещать лучше в заголовочный файл