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

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

isalphanum(c)	Возвращает true, если с буква или цифра
isalpha(c)	с – буква
isdigit(c)	с – цифра
ispunct(c)	Знак пунктуации
isspace(c)	пробел
isgraph(c)	печатаемый символ
islower(c)	символ в нижнем регистре
isxdigit(c)	с – шестнадцатиричная цифра
toupper(c)	возвращает прописную букву
tolower(c)	вовзращает 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

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

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