Шаблоны и библиотека STL

Функции-шаблоны

```
template<class тип1, class тип2, ...>
тип имя_функции(список параметров)
{
 // тело функции
}
```

Функции-шаблоны

```
#include <iostream>
template <class X> void swap(X &a, X &b)
{
    X temp = a; a = b; b = temp;
}
int main()
{
    int i = 10, j = 20;
    float x = 10.1, y = 23.3;
    char a = 'x', b = 'z';
    swap(i, j);        // обмен целых
    swap(x, y) ;       // обмен вещественных
    swap(a, b) ;       // обмен символов
    return 0;
}
```

Функции с двумя типами-шаблонами

```
#include <iostream>
template <class type1, class type2>
void F(type1 &x, type2 y)
{
    cout << x << ' ' << y << endl;
}
int main()
{
    int i = 10;  char c = ‘q’;
    F(i, "hi");  F(c, 0.23);
    point x;  F(x, 2);
    return 0;
}
```

Ограничения на функции-шаблоны

- функции-шаблоны должны выполнять одни и те же общие действия, и только тип данных может быть различным

- виртуальная функция не может быть функцией-шаблоном

Классы-шаблоны

```
template<class тип1, class тип2, ...>
class имя_класса
{
   // тело класса
};

имя_класса<тип> объект;
```

Шаблонный класс stack

```
#include <iostream>
template<class X>
class stack
{
    X *arr; int size, top;
public:
    stack(int s)
    {
        arr = new X[s]; size = s; top = -1;
    }
    ~stack()  {  delete [] arr;  }
    void push(X val);
    X pop();
};
```

Реализация методов класса stack

```
template<class X>
void stack<X>::push(X val)
{
   if (top < size-1)  arr[++top] = val;
   else throw "Stack is full";
}

template<class X>
X stack<X>::pop()
{
  if (top >= 0) return arr[top--];
  throw "Stack is empty";
}
```

Классы-шаблоны

```
int main()
{
    stack<int> a(9);  stack<double> b(5);  stack<char> c(10);
    try
    {
        a.push(1); b.push(-12.23);
        for (int i = 0; i < 10; i++)  c.push((char)('A' + i));
        for (int i = 0; i < 10; i++)  cout << c.pop();
    }
    catch (char *err)
    {
        cout << err;
    }
    return 0;
}
```

Классы-шаблоны

Пусть определен некоторый класс:

```
class addr
{
    char name[40];
    char street[40];
    char city[30];
    char state[3];
    char zip[12];
public: …
};
```

и функция, которая генерирует случайную строку:

```
char* rand_string_gener(int leng);
```

Классы-шаблоны

```
int main()
{
    addr a, b, c, *p;
    stack<char *> s_str(5);
    stack<addr> s_addr(8);
    stack<addr *> s_ptr_addr(10);

    for (int i = 0; i < 5; i++)
        s_str.push(rand_string_gener(rand() % 10 + 1));

    s_addr.push(a); b = s_addr.pop();

    s_ptr_addr.push(&c);
    s_ptr_addr.push(new addr); p = s_ptr_addr.pop();
    return 0;
}
```

Пример с двумя типами-шаблонами

```
#include <iostream>
template <class Type1, class Type2> class myclass
{
    Type1 i;  Type2 j;
public:
    myclass(Type1 a, Type2 b) { i = a; j = b; }
    void show() { cout << i << “ “ << j << “\n”; }
};
int main()
{
    myclass<int, double> o1(10, 0.23);
    myclass<char, char*> o2('X', "This is a test");
    o1.show();      // вывод int, double
    o2.show();      // вывод char, char*
    return 0;
}
```

Обзор библиотеки STL

- контейнеры – это вспомогательные объекты, содержащие другие объекты
  - vector (динамический массив)
  - queue (очередь)
  - list (линейный список)
  - stack (стек)
  - string (строка)
  - map (словарь)
  - …
- алгоритмы
  - заполнение элементов контейнера инициализирующими значениями или множеством значений
  - поиск значений среди элементов контейнера
  - проверка элементов двух контейнеров на попарное равенство
  - …
- итераторы – объекты, предоставляющие возможности манипуляции элементами контейнеров

Обзор библиотеки STL

```
#include <vector>
using namespace std;

#include <string>
#include <list>
```

Класс vector

```
#include <vector>
using namespace std;
void main()
{
    vector<int> v;
}
```

Класс vector

```
#include <vector>
#include <iostream>
using namespace std;
void main()
{
    vector<int> v;
    v.push_back(1);
    if (v.size() != 0) cout << "Last element: " <<
    v.back() << endl;
    v.push_back(2);
    if (v.size() != 0) cout << "New last element: " <<
    v.back() << endl;
}
```

Класс vector

```
#include <vector>
#include <iostream>
using namespace std;
void main()
{
    vector<int> v;
    v.push_back(1);
    cout << "Current capacity of v = " << v.capacity() << endl;
    v.reserve(20);
    cout << "Current capacity of v = " << v.capacity() << endl;
}
```

Класс vector

```
#include <vector>
#include <iostream>
using namespace std;
int main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    int i = v[1];
    cout << "The second element of v is " << i << endl;
}
```

Класс vector

- clear() – очищает вектор, т.е. удаляет все хранящиеся в нем элементы
- empty() – возвращает истину, если в векторе нет элементов, т.е. если он пустой
- front() – возвращает ссылку на первый элемент вектора
- pop_back() – удаляет последний элемент вектора

Класс string

- operator<< - оператор вставки строки в поток вывода
- operator>> - оператор извлечения строки из потока ввода
- operator= - оператор присваивания строк
- operator+ - оператор сложения (конкатенации) двух объектов string, объекта string с обычной строкой языка C, т.е. char*, и объекта string с одиночным символом
- operator<, operator>, operator<=, operator>=, operator==, operator!= - операторы сравнения строк

Класс string

```
#include <string>
#include <iostream>
using namespace std;
void main()
{
    string message = "Hello!";
    cout << message << endl;
}
```

Класс string

```
#include <string>
#include <iostream>
using namespace std;
void main()
{
    string s;
    cout << "Enter a word: ";
    cin >> s;
    cout << "You entered: " << s << endl;
}
```

Класс string

```
#include <string>
#include <iostream>
using namespace std;
void main()
{
    string result;
    string s1 = "ABC",  s2 = "DEF";
    char cp1[] = "GHI";  char c = 'J';
    result = s1 + cp1;     // Сложение string и char*
    result = cp1 + s1;     // Сложение char* и string
    result = s1 + s2;      // Сложение двух string
    result = s1 + c;       // Сложение string и char
    result = c + s1;       // Сложение char и string
}
```

Класс string

```
#include <string>
#include <iostream>
using namespace std;
void main()
{
    string s1="ABC“, s2="ABC“, s3="DEF";
    if (s1 == s2) cout << "s1 and s2 are equal" << endl;
    else cout << "s1 and s2 are NOT equal" << endl;

    if (s1 < s3) cout << "s1 < s3" << endl;
    else cout << "s1 >= s3" << endl;
}
```

Класс list

```
#include <list>
using namespace std;
void main()
{
    list<int> l;
}
```

Класс list

```
#include <list>
#include <iostream>
using namespace std;
void main()
{
    list<int> l;
    l.push_back(1);
    if (l.size() != 0)
     cout << "Last element: " << l.back() << endl;
    l.push_back(2);
    if (l.size() != 0)
     cout << "New last element: " << l.back() << endl;
}
```

Класс list

```
#include <list>
#include <iostream>
using namespace std;
void main()
{
    list<int> l;
    list<int>::iterator i;
    l.push_back(10);
    l.push_back(20);
    l.push_back(30);
    cout << "The list is now:";
    for (i = l.begin(); i != l.end(); i++)
        cout << " " << *i;
}
```

Класс list

```
#include <iostream>
#include <list>
using namespace std;
void main()
{
    list<int> l;
    list<int>::iterator i;
    l.push_back(10); l.push_back(20); l.push_back(30);
    i = l.begin();  i++;
    l.insert(i, 999);
    cout << "l =";
    for(i = l.begin(); i != l.end(); i++) cout << " " << *i;
    cout << endl;
}
```

Класс list

```
#include <iostream>
#include <list>
using namespace std;
void main()
{
    list<int> l;
    list<int>::iterator i;
    l.push_back(10); l.push_back(20); l.push_back(30);
    l.push_back(40); l.push_back(50);
    i = l.begin();  i++;
    l.erase(i);
    for (i = l.begin(); i != l.end(); i++) cout << " " << *i;
    cout << endl;
}
```

Класс list

- clear() – очищает список, т.е. удаляет все хранящиеся в нем элементы
- empty() – возвращает истину, если в списке нет элементов, т.е. если он пустой
- front() – возвращает ссылку на первый элемент списка
- push_front() – добавляет элемент в начало списка
- pop_front() – удаляет первый элемент списка
- pop_back() – удаляет последний элемент списка

Класс map

```
#include <utility>
using namespace std;
template <class T1, class T2> struct pair;

int main ()
{
    pair <string, double> a;
    pair <string, double> b("Pi", 3.14);
    pair <string, double> c (b);
    a = make_pair(string("half"), 0.5);
    return 0;
}
```

Класс map

first, second – открытые поля структуры pair

```
#include <utility>
int main ()
{
    pair <int, int> a(1, 2);
    a.first = 10;
    a.second *= 2;
    cout << "Pair: " << a.first << ", " << a.second;
    return 0;
}
```

Класс map

```
#include <map>
int main ()
{
    map<double, string> md;
    md.insert(
        pair<double, string>(0.1, string("abc"))
    );
    md.insert(
        pair<double, string>(2.71, string('f'))
    );
    return 0;
}
```

Класс map

```
#include <map>
int main ()
{
    map<char, int> m;
    m.insert(pair<char, int>('a', 300));
    m.insert(pair<char, int>('d', 400));
    map<char, int>::iterator it = m.begin();
    m.insert (it, pair<char, int>('b', 100));
    m.insert (it, pair<char, int>('c', 200));
    for (it = m.begin(); it != m.end(); ++it)
        cout << it->first << " : " << it->second << endl;
    return 0;
}
```

Класс map

```
#include <map>
int main ()
{
    map<char, int> m;
    map<char, int>::iterator it;
    …      // заполнение контейнера
    it = m.find('b');
    if (it != m.end())
    {
        cout << “Removing ” << it->second;
        m.erase(it);
    }
    return 0;
}
```

Класс map

```
#include <map>
int main ()
{
    map<char, int> m;
    m['b'] = 100;
    m['a'] = 200;
    m['c'] = m['a'] + m['d'];
    m['a']++;
    return 0;
}
```