

## Шаблони (Templates)

Любомир Чорбаджиев  
Технологическо училище "Електронни системи"  
Технически университет, София  
lchorbadjiev@elsys-bg.org  
*Revision : 1.1*

9 март 2005 г.

## Шаблони

- Шаблоните обезпечават непосредствената поддръжка на така нареченото **обобщено програмиране**, т.е. програмиране, при което като параметри се използват типове.
- Механизмът на шаблоните в C++ позволява използването на типове в качеството на параметри при дефинирането на функции и класове.
- Шаблонът зависи само от тези свойства на параметъра-тип, които той явно използва; поради това не е необходимо различните типове, които се използват като параметри на шаблона да бъдат свързани по какъвто и да било начин.

## Дефиниране на шаблон

```
template<class T> class stack {  
    T data_[128];  
public:  
    const T& pop(void) const;  
    //...  
};
```

- Префиксът **template<class T>** се използва за дефиниране на шаблон (**template**).
- При използване на шаблона на мястото на "формалния параметър" **class T** се предава фактическият тип.
- В дефиницията на шаблона **T** точно по същия начин, по който се използват и имената на другите типове.
- Областта на видимост за **T** завършва в края на обявата, започнала с **template<class T>**.
- В дефиницията **template<class T>** **T** е име на произволен тип; не е задължително **T** да бъде име на клас.

## Екземпляри на шаблона

```
stack<double> doubleStack;  
stack<int> intStack;
```

- Процесът на генериране на клас от (1) шаблон на клас и (2) аргумент на шаблона се нарича **създаване на екземпляр на шаблона (template instantiation)**.
- Генерирането на клас от шаблон на клас се изпълнява от компилатора;
- Класът, генериран от шаблон на клас, е обикновен C++ клас. Използването на шаблони не предполага допълнителни механизми по време на изпълнение на кода.
- Шаблоните обезпечават ефективен начин за генериране на код.

## Параметри на шаблона

- Като параметри на даден шаблон могат да се използват и параметри, които не са типове:

Пример:

```
template<class T, int size> class Buffer {
    T data_[size];
    int size_;
public:
    Buffer(void) : size_(size)
    {}
    //...
};
```

4

## Проверка на типовете

- Проверка в точката на дефиниция: проверка за синтактични грешки и грешки, които не зависят от фактическите параметри-типове на шаблона.
- Проверка при създаване на екземпляр на шаблона: проверка за съответствие на фактическите типове, предадени на шаблона.
- Проверка в момента на свързване.

5

## Проверка на типовете: пример

```
1 template<class T> class stack {
2     T data_[128];
3     int top_;
4 public:
5     stack(void) : top_(-1) {}
6     //...
7     void print_all(void) {
8         for(int i=0;i<=top_;++i)
9             cout << data_[i] << '␣';
10        cout << endl;
11    }
12 };
13 class Rec { /*...*/};

1 stack<int> intStack;
2 stack<Rec> recStack; // ?? error;
3 recStack.print_all(); // error;
```

6

## Пример: стек

```
1 #ifndef STACK_HPP__
2 #define STACK_HPP__
3 #include <exception>
4 template<class T>
5 class stack {
6     static const unsigned size_=128;
7     T data_[size_];
8     int top_;
9 public:
10    stack(void);
11    const T& top(void) const;
12    void pop(void);
13    void push(const T& val);
14    bool empty(void) const;
15 };
16 template<class T>
17 stack<T>::stack(void)
18     : top_(-1)
19 {}
20 template<class T> const T&
21 stack<T>::top(void) const {
22     if (top_ < 0) {
```

7

```

23     throw std::exception();
24 }
25     return data_[top_];
26 }
27 template<class T> void
28 stack<T>::pop(void) {
29     if (top_<0){
30         throw std::exception();
31     }
32     top_--;
33 }
34 template<class T> void
35 stack<T>::push(const T& val){
36     if( size_ <= top_+1 ) {
37         throw std::exception();
38     }
39     data_[++top_]=val;
40 }
41 template<class T> bool
42 stack<T>::empty(void) const {
43     return top_<0;
44 }
45 #endif

```

## Пример: стек

```

1 #include <iostream>
2 #include "stack.hpp"
3
4 int main(void) {
5     stack<int> si;
6     for(int i=0;i<10; ++i){
7         si.push(i);
8     }
9     while(! si.empty() ){
10        std::cout << si.top() << " ";
11        si.pop();
12    }
13    std::cout << std::endl;
14    stack<float> sf;
15    for(int i=0; i<10; ++i){
16        sf.push(10.0*i);
17    }
18    while(! sf.empty() ){
19        std::cout << sf.top() << " ";
20        sf.pop();
21    }
22    std::cout << std::endl << std::endl;

```

8

```

23
24     stack<stack<int> > ssi;
25     for(int i=0;i<5;++i){
26         stack<int> temp;
27         for(int j=0;j<10;++j){
28             temp.push(i);
29         }
30         ssi.push(temp);
31     }
32
33     while(!ssi.empty()){
34         stack<int> ts=ssi.top();
35         while(!ts.empty()){
36             std::cout << ts.top() << " ";
37             ts.pop();
38         }
39         std::cout << std::endl;
40         ssi.pop();
41     }
42
43     return 0;
44 }

```

## Пример: стек

Результати:

```

9 8 7 6 5 4 3 2 1 0
90 80 70 60 50 40 30 20 10 0

```

```

4 4 4 4 4 4 4 4 4 4
3 3 3 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2 2 2
1 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0

```

9

## Шаблони на функции

- Механизмът на шаблоните може да се ползва за обобщено дефиниране на функции;

Пример:

```
1 template<class R, class T>
2 const R& fun(T& a){
3     //...
4 }

1 template<class T>
2 void sort(vector<T>& v){
3     //...
4 }
```

10

## Шаблони на функции

```
1 template<class T>
2 void swap(T& a, T& b) {
3     T tmp=a;
4     a=b;
5     b=tmp;
6 }
```

11

## Шаблони на функции

- При шаблоните на функции съществен момент се явява възможността за **извеждане (deduction)** на типа на аргументите на шаблона.

Пример:

```
template<class T> T& fun(const T& val) { /*...*/ }
int i=0, p=10;
i=fun(p);

template<class T> const T& fun1(void) { /*...*/ }
int x=fun1(); //error
int y=fun1<int>();

template<class R, class T> R fun2(T& v) { /*...*/ }
int z=0;
double w=0;
w=fun2<double, int>(z);
w=fun2<double>(z);
w=fun2(z); // error!!
```

## Използване на шаблони: масив с проверка на границите<sup>12</sup>

```
1 #include <iostream>
2 using namespace std;
3 class IndexOutOfBounds {};
4
5 template<class T>
6 class Array {
7     unsigned int size_;
8     T* data_;
9 public:
10    Array(unsigned int size=10)
11        : size_(size), data_(new T[size_])
12    {}
13    Array(const Array& other)
14        : size_(other.size_), data_(new T[size_])
15    {
16        for(unsigned int i=0; i< size_; i++)
17            data_[i]=other.data_[i];
18    }
19    ~Array(void) {
20        delete [] data_;
21    }
```

13

```

22 unsigned size() const {
23     return size_;
24 }
25 Array& operator=(const Array& other) {
26     if(this!=&other) {
27         delete [] data_;
28         size_=other.size_;
29         data_=new int[size_];
30         for(unsigned i=0;i<size_;i++)
31             data_[i]=other.data_[i];
32     }
33     return *this;
34 }
35 T& operator[](unsigned int index)
36     throw (IndexOutOfBounds)
37     {
38     if(index>=size_) {
39         throw IndexOutOfBounds();
40     }
41     return data_[index];
42 }
43 };

```

```

44 int main(void) {
45     Array<int> a1(3), a2;
46     for(int i=0;i<3;++i) {
47         a1[i]=i;
48     }
49     a2=a1;
50     for(int i=0;i<3;i++) {
51         cout << "a2[" << i << "]= " << a2[i] << endl;
52     }
53     try {
54         cout << "a2[" << 3 << "]= " << a2[3] << endl;
55     } catch(IndexOutOfBounds toCatch) {
56         cerr << "IndexOutOfBounds exception caught..."
57             << endl;
58     }
59     return 0;
60 }

```

```

lubo@dobby: ~/school/cpp/notes> ./a.out
a2[0]=0
a2[1]=1
a2[2]=2
IndexOutOfBounds exception caught...

```

## Исползване на шаблони: динамичен стек

```

1 #include <iostream>
2 using namespace std;
3 class StackEmptyException {};
4
5 template<class T>
6 class Stack {
7     const static unsigned int chunk_=2;
8     int size_;
9     T *data_;
10    int top_;
11 public:
12    Stack(void)
13        : size_(chunk_),
14          data_(new T[size_]),
15          top_(-1)
16    {}
17    ~Stack(void) {
18        delete [] data_;
19    }

```

14

```

20 Stack(const Stack& other)
21     : size_(other.size_),
22       data_(new T[size_]),
23       top_(other.top_)
24 {
25     for(int i=0; i<=top_; i++)
26         data_[i]=other.data_[i];
27 }
28 Stack& operator=(const Stack& other) {
29     if(this!=&other) {
30         delete [] data_;
31         size_=other.size_;
32         top_=other.top_;
33         data_=new T[size_];
34         for(int i=0;i<=top_;i++)
35             data_[i]=other.data_[i];
36     }
37     return *this;
38 }

```

```

39 void push(const T& v) {
40     if(top_>=(size_-1)) {
41         resize();
42     }
43     data_[++top_]=v;
44 }
45 T pop(void) {
46     if(top_<0){
47         throw StackEmptyException();
48     }
49     return data_[top_--];
50 }
51 private:
52 void resize(void) {
53     T *oldData=data_;
54     data_=new T[size_+chunk_];
55     for(int i=0;i<size_;i++)
56         data_[i]=oldData[i];
57     delete [] oldData;
58     size_+=chunk_;
59 }
60 };

```

```

61 int main(void) {
62     Stack<int> st;
63     st.push(1);
64     st.push(2);
65     st.push(3);
66
67     Stack<int> st1=st;
68
69     cout << st.pop() << endl;
70     cout << st.pop() << endl;
71     cout << st.pop() << endl;
72
73     cout << st1.pop() << endl;
74     cout << st1.pop() << endl;
75     cout << st1.pop() << endl;
76
77     try {
78         cout << st1.pop() << endl;
79     } catch(const StackEmptyException& toCatch) {
80         cout << "StackEmptyException caught..." << endl;
81     }
82     return 0;
83 }

```

```
lubo@kid:~/school/cpp/notes$ ./a.out
```

```
3
```

```
2
```

```
1
```

```
3
```

```
2
```

```
1
```

```
StackEmptyException caught...
```