

Шаблони (Templates)

Любомир Чорбаджиев¹
lchorbadjiev@elsys-bg.org

¹Технологическо училище “Електронни системи”
Технически университет, София

9 април 2008 г.

Съдържание

- 1 Нужда от шаблони
- 2 Решение в стил C
- 3 Решение в стил C++: използване на шаблони
- 4 Дефиниране на шаблонна функция
- 5 Дефиниране на шаблонен клас
- 6 Екземпляри на шаблона
- 7 Проверка на шаблона
- 8 Пример: шаблонен стек (статичен)
- 9 Шаблонни функции
- 10 Пример: шаблонен масив с проверка на границите
- 11 Пример: шаблонен стек (динамичен)

Пример: копиране на масив от цели числа

- Даден е масив от цели числа. Трябва да се копират стойностите на елементите на масива в друг масив.

```
1 void copy(int dst[], int src[], int size) {  
2     for(int i=0;i<size;i++)  
3         dst[i]=src[i];  
4 }
```

Пример: копиране на масив от цели числа

```
1 #include <iostream>
2 int main(void) {
3     int a[10], b[10];
4     for(int i=0;i<10;i++)
5         a[i]=i*10;
6     copy(b,a,10);
7     for(int i=0;i<10;i++)
8         std::cout << b[i] << " ";
9     std::cout << std::endl;
10    return 0;
11 }
```

Пример: копиране на масив от числа с плаваща запетая

- Даден е масив от числа с плаваща запетая. Трябва да се копират стойностите на елемента на масива в друг масив.
- Използването на функцията `void copy(int a[], int b[], int size)` води до непредвидими резултати.

Пример: копиране на масив от числа с плаваща запетая

```
1 #include <iostream>
2 int main(void) {
3     double c[10],d[10];
4     for(int i=0;i<10;i++)
5         c[i]=i*1000.0;
6     copy((int*)d,(int*)c,10);
7     for(int i=0;i<10;i++)
8         std::cout << d[i] << ";\n";
9     std::cout << std::endl;
10    return 0;
11 }
```

```
0; 1000; 2000; 3000; 4000; 2.12203e-314; 6.95327e-310;\n2.07343e-317; 6.95327e-310; 2.07362e-317;
```

Пример: копиране на масив от числа с плаваща запетая

- Директното решение е да се дефинира нова функция:

```
1 void copy(double dst[], double src[], int size) {  
2     for(int i=0; i<size; i++)  
3         dst[i]=src[i];  
4 }
```

Пример: копиране на масив от числа с плаваща запетая

```
1 #include <iostream>
2 int main(void) {
3     double c[10], d[10];
4     for(int i=0; i<10; i++)
5         c[i]=i*1000.0;
6     copy(d, c, 10);
7     for(int i=0; i<10; i++)
8         std::cout << d[i] << "; ";
9     std::cout << std::endl;
10    return 0;
11 }
```

0; 1000; 2000; 3000; 4000; 5000; 6000; 7000; 8000; 9000;

Проблеми

- Ако се подхожда по този начин за всеки тип трябва да се дефинира отделна версия на функцията `copy()`.
- Използването на `copy/paste` за размножаване на кода води до:
 - труден за поддържане код;
 - вмъкване на трудни за откриване грешки.
- Необходимо е дефиницията на функцията `copy()` да бъде обобщена по някакъв начин.
 - За всички типове алгоритъма на функцията `copy()` е един и същ.
 - Да се дефинира една функция, която да е в състояние да работи с всички типове.

Решение в стил C

- За аргументи на функцията `copy()` могат да се използват указатели от типа `void*`.

```
1 void copy_array(void* dst, void* src, int size) {  
2     for(int i=0;i<size;i++) {  
3         static_cast<char*>(dst)[i]=  
4             static_cast<char*>(src)[i];  
5     }  
6 }
```

- При използването на тази функция, параметърът `int size` трябва да се интерпретира като размер на копирувания масив в байтове, а не като брой елементи в масива.

Решение в стил С

```
1 #include <iostream>
2 int main(void) {
3     int a[10],b[10];
4     double c[10],d[10],e[10];
5     for(int i=0;i<10;i++) {
6         a[i]=i*1000;
7         c[i]=i*1.1;
8         e[i]=0;
9     }
```

Решение в стил C

```
1 copy_array(b,a, sizeof(a));  
2 for(int i=0;i<10;i++)  
3     std::cout<< b[i] << ";_";  
4 std::cout << std::endl;
```

0; 1000; 2000; 3000; 4000; 5000; 6000; 7000; 8000; 9000;

Решение в стил C

```
1 copy_array(d,c, sizeof(c));  
2 for(int i=0;i<10;i++)  
3     std::cout<< d[i] << ";␣";  
4 std::cout << std::endl;
```

0; 1.1; 2.2; 3.3; 4.4; 5.5; 6.6; 7.7; 8.8; 9.9;

Решение в стил C

```
1  copy_array(e,a, sizeof(a));  
2  for(int i=0;i<10;i++)  
3      std::cout<< e[i] << ";␣";  
4  std::cout << std::endl;  
5  return 0;  
6 }
```

```
2.122e-311; 6.36599e-311; 1.061e-310; 1.4854e-310;\n1.9098e-310; 0; 0; 0; 0; 0;
```

Решение в стил C

- В този стил са дефинирани и функциите `memcpy()` и `memcpy_s()` от стандартната C-библиотека.
- За решаване на задачата за копиране на масив директно може да се използва функцията `memcpy()`.

Решение в стил C

```
1 #include <cstdlib>
2 #include <iostream>
3 int main(void) {
4     int a[10];
5     double b[10];
6     for(int i=0;i<10;i++){
7         a[i]=i*1000;
8         b[i]=0;
9     }
10    std::memcpy(b,a,sizeof(a));
11    for(int i=0;i<10;i++)
12        std::cout << b[i] << ";\n";
13    std::cout << std::endl;
14 }
```

```
2.122e-311; 6.36599e-311; 1.061e-310; 1.4854e-310;\n
1.9098e-310; 0; 0; 0; 0; 0;
```


Решение в стил C: проблеми

- Лесно се правят грешки в използваните типове данни.
- Компиляторът не е в състояние да проследи грешната употреба на променливи с различни типове.
- Резултатът от подобни програми зависи от архитектурата на процесора, върху който се изпълняват.

Решение в стил C++: използване на шаблони

- Решението в стил C++ е да се дефинира шаблонна функция `copy()`.

```
1 template <class T>
2 void copy(T dst[], T src[], int size) {
3     for(int i=0;i<size;i++)
4         dst[i]=src[i];
5 }
```

- Използването на тази функция е тривиално:

Решение в стил C++: използване на шаблони

```
1 #include <iostream>
2 int main(void) {
3
4     // версия на copy() за масиви от int
5     int a[10], b[10];
6     for(int i=0; i<10; i++)
7         a[i]=i*10;
8
9     copy<int>(b, a, 10);
10    for(int i=0; i<10; i++)
11        std::cout << b[i] << " ";
12    std::cout << std::endl;
```

0; 10; 20; 30; 40; 50; 60; 70; 80; 90;

Решение в стил C++: използване на шаблони

```
1 // версия на copy() за масиви от double
2 double c[10],d[10];
3 for(int i=0;i<10;i++)
4     c[i]=i*1000.0;
5
6 copy<double>(d,c,10);
7 for(int i=0;i<10;i++)
8     std::cout << d[i] << "; ";
9 std::cout << std::endl;
10
11 return 0;
12 }
```

0; 1000; 2000; 3000; 4000; 5000; 6000; 7000; 8000; 9000;

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.

Шаблонни функции

- Шаблонните функции описват поведение, което може да бъде прилагано към различни типове.
- Една шаблонна функция описва семейство от функции, които имат еднакво поведение, но могат да бъдат прилагани към различни типове аргументи.
- Конструкцията за дефиниране на шаблонна функция е:

```
1 template <class R, class T, ...>  
2 R function_name(T arg1, ...) {  
3 ...  
4 }
```

Пример: повдигане на куб

- Задача: да се напише функция, която за дадено число x изчислява x^3 .
- За решаването на тази задача е възможно да се използват макроси на препроцесора.

```
1 #define CUBE(x) ((x)*(x)*(x))
```

- Такова решение работи за всякакви типове.

```
1 #include <iostream>
2 int main(void) {
3     std::cout << "CUBE (3) = " << CUBE (3) << std::endl;
4     std::cout << "CUBE (3+3) = " << CUBE (3+3) << std::endl;
5     return 0;
6 }
```

- $CUBE(3+3)$ се изчислява като $((3+3)*(3+3)*(3+3))$.

Пример: повдигане на куб

- Друг вариант за решаване на тази задача е да се дефинира шаблонна функция `cube()`.

```
1 template <typename T>
2 T cube(const T& x) {
3     return x*x*x;
4 }
5
6 #include <iostream>
7 int main(void) {
8     std::cout<<"cube (3)="
9         <<cube<int>(3)<<std::endl;
10    std::cout<<"cube (3+3.3)="
11        <<cube<double>(3+3.3)<<std::endl;
12    std::cout<<"cube (9.9)="
13        <<cube(9.9)<<std::endl;
14 }
```


Пример: намиране на максимум

- Задача: да се дефинира функция `max()`, която връща по-големия от два аргумента.

```
1 template <class T>
2 const T& maxvalue(const T& a, const T& b) {
3     return a>b?a:b;
4 }
5
6 #include <iostream>
7 int main(void) {
8     std::cout<<"max(5,6)="
9         <<maxvalue<int>(5,6)<<std::endl;
10    std::cout<<"max(\"hello\", \"bye\")="
11        <<max<string>("hello", "bye")<<endl;
12    return 0;
13 }
```

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 даден шаблон, на мястото на “формалния параметър” **class T** се предава фактическият тип.
- В дефиницията на шаблона формалното име на тип **T** се използва точно по същия начин, по който се използват и имената на другите типове.

Дефиниране на шаблон

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

- 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 не само типове:

Пример:

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

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.

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};
```

Проверка на типовете: пример

- Да приемем, че за класа `Rec` не е дефиниран оператор за изход `operator<<(ostream& out, const Rec& r)`.
- Тогава екземплярът `recStack` на шаблона `stack<T>` дефиниран в ред 2 съдържа грешка в метода `print_all()`, тъй като този метод разчита елементите на стека да имат предефиниран оператор за изход.

```
1 class Rec { /* ... */ };  
2 stack<Rec> recStack; // ?? error;  
3 recStack.print_all(); // error;
```


Пример: заглавен файл `stack.hpp`

```
1 #ifndef STACK_HPP__
2 #define STACK_HPP__
3
4 #include <exception>
5
6 template<class T>
7 class stack {
8     static const unsigned size_=128;
9     T data_[size_];
10    int top_;
11 public:
12    stack(void);
```

Пример: заглавен файл `stack.hpp`

```
1  const T& top(void) const ;
2  void pop(void) ;
3  void push(const T& val) ;
4  bool empty(void) const ;
5  };
6
7  template <class T>
8  stack<T>::stack(void)
9    : top_(-1)
10 {}
```

Пример: заглавен файл `stack.hpp`

```
1
2 template<class T> const T&
3 stack<T>::top(void) const {
4     if (top_ < 0) {
5         throw std::exception();
6     }
7     return data_[top_];
8 }
9 template<class T> void
10 stack<T>::pop(void) {
11     if (top_ < 0){
12         throw std::exception();
13     }
14     top_--;
15 }
```

Пример: заглавен файл `stack.hpp`

```
1 template<class T> void
2 stack<T>::push(const T& val){
3     if( size_ <= top_+1 ) {
4         throw std::exception();
5     }
6     data_[++top_]=val;
7 }
8 template<class T> bool
9 stack<T>::empty(void) const {
10     return top_ < 0;
11 }
12 #endif
```

Пример: използване на шаблонен стек

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

Пример: използване на шаблонен стек

```
1 stack<float> sf;  
2 for(int i=0; i<10; ++i){  
3     sf.push(10.0*i);  
4 }  
5 while(! sf.empty() ){  
6     std::cout << sf.top() << "␣";  
7     sf.pop();  
8 }  
9 std::cout << std::endl << std::endl;
```

Пример: използване на шаблонен стек

```
1 stack<stack<int>> ssi;  
2 for(int i=0;i<5;++i){  
3     stack<int> temp;  
4     for(int j=0;j<10;++j){  
5         temp.push(i);  
6     }  
7     ssi.push(temp);  
8 }
```

Пример: използване на шаблонен стек

```
1  while(!ssi.empty()){
2      stack<int> ts=ssi.top();
3      while(!ts.empty()){
4          std::cout << ts.top() << "␣";
5          ts.pop();
6      }
7      std::cout << std::endl;
8      ssi.pop();
9  }
10
11 return 0;
12 }
```


Пример: стек

Резултати:

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

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 }
```

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 }
```

Шаблони на функции

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

Пример:

```
1 template <class T> T& fun(const T& val) { /*...*/ }
2 int i=0, p=10;
3 i=fun(p);
4
5 template <class T> const T& fun1(void) { /*...*/ }
6 int x=fun1(); //error
7 int y=fun1<int>();
```

Шаблони на функции

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

Използване на шаблони: масив с проверка на границите

```
1 #include <iostream>
2 #include <exception>
3 using namespace std;
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    {}
```

Използване на шаблони: масив с проверка на границите

```
1  Array(const Array& other)
2      : size_(other.size_), data_(new T[size_])
3  {
4      for(unsigned int i=0; i< size_; i++)
5          data_[i]=other.data_[i];
6  }
7  ~Array(void) {
8      delete [] data_;
9  }
10 unsigned size() const {
11     return size_;
12 }
```

Използване на шаблони: масив с проверка на границите

```
1  Array& operator=(const Array& other) {  
2      if(this!=&other) {  
3          delete [] data_;  
4          size_=other.size_;  
5          data_=new T[size_];  
6          for(unsigned i=0;i<size_;i++)  
7              data_[i]=other.data_[i];  
8      }  
9      return *this;  
10 }
```


Използване на шаблони: масив с проверка на границите

```
1 T& operator [] (unsigned int index)
2     throw (exception)
3     {
4         if (index >= size_) {
5             throw exception();
6         }
7         return data_[index];
8     }
9 };
```

Използване на шаблони: масив с проверка на границите

```
1 int main(void) {
2     Array<int> a1(3), a2;
3     for(int i=0;i<3;++i) {
4         a1[i]=i;
5     }
6     a2=a1;
7     for(int i=0;i<3;i++) {
8         cout << "a2[" << i << "]= " << a2[i] << endl;
9     }
```

Използване на шаблони: масив с проверка на границите

```
1  try {  
2      cout << "a2[" << 3 << "]= " << a2[3] << endl;  
3  } catch(exception &e) {  
4      cout << "exception caught..."  
5          << endl;  
6  }
```

Използване на шаблони: масив с проверка на границите

```
1 Array<double> a3(3), a4;  
2 for(int i=0;i<3;++i) {  
3     a3[i]=(i+1)*3.14;  
4 }  
5 a4=a3;  
6 for(int i=0;i<3;i++) {  
7     cout << "a4[" << i << "]= " << a4[i] << endl;  
8 }
```

Използване на шаблони: масив с проверка на границите

```
1  try {  
2      cout << "a3[" << 3 << "]=" << a3[3] << endl;  
3  } catch(exception &e) {  
4      cout << "exception_ caught..."  
5          << endl;  
6  }  
7  
8  return 0;  
9 }
```

Използване на шаблони: масив с проверка на границите

```
lubo@dobby:~/school/cpp/notes> ./a.out  
a2[0]=0  
a2[1]=1  
a2[2]=2  
exception caught...  
a4[0]=3.14  
a4[1]=6.28  
a4[2]=9.42  
exception caught...
```

Използване на шаблони: динамичен стек

```
1 #include <iostream>
2 #include <exception>
3 using namespace std;
4
5 template<class T>
6 class Stack {
7     const static unsigned int chunk_=2;
8     int size_;
9     T *data_;
10    int top_;
```

Използване на шаблони: динамичен стек

```
1 public :  
2     Stack(void)  
3         : size_(chunk_),  
4           data_(new T[size_]),  
5           top_(-1)  
6     {}  
7     ~Stack(void) {  
8         delete [] data_;  
9     }
```


Използване на шаблони: динамичен стек

```
1 Stack(const Stack& other)
2     : size_(other.size_),
3       data_(new T[size_]),
4       top_(other.top_)
5 {
6     for(int i=0; i<=top_; i++)
7         data_[i]=other.data_[i];
8 }
```

Използване на шаблони: динамичен стек

```
1 Stack& operator=(const Stack& other) {  
2     if(this!=&other) {  
3         delete [] data_;  
4         size_=other.size_;  
5         top_=other.top_;  
6         data_=new T[size_];  
7         for(int i=0;i<=top_;i++)  
8             data_[i]=other.data_[i];  
9     }  
10    return *this;  
11 }
```

Използване на шаблони: динамичен стек

```
1 void push(const T& v) {
2     if(top_ >=(size_-1)) {
3         resize();
4     }
5     data_[++top_]=v;
6 }
7 T pop(void) {
8     if(top_ <0){
9         throw exception();
10    }
11    return data_[top_--];
12 }
```

Използване на шаблони: динамичен стек

```
1 private :  
2 void resize(void) {  
3     T *oldData=data_;  
4     data_=new T[size_+chunk_];  
5     for(int i=0;i<size_;i++)  
6         data_[i]=oldData[i];  
7     delete [] oldData;  
8     size_+=chunk_;  
9 }  
10 };
```

Използване на шаблони: динамичен стек

```
1 int main(void) {
2     Stack<int> st;
3     st.push(1);
4     st.push(2);
5     st.push(3);
6
7     Stack<int> st1=st;
8     cout << st.pop() << endl;
9     cout << st.pop() << endl;
10    cout << st.pop() << endl;
11
12    cout << st1.pop() << endl;
13    cout << st1.pop() << endl;
14    cout << st1.pop() << endl;
```

Използване на шаблони: динамичен стек

```
1  try {  
2      cout << st1.pop() << endl;  
3  } catch(const exception& e) {  
4      cout << "exception caught..." << endl;  
5  }  
6  return 0;  
7 }
```

Използване на шаблони: динамичен стек

```
lubo@kid:~/school/cpp/notes$ ./a.out
3
2
1
3
2
1
exception caught...
```