

Типове II: Шаблони на функции. Указатели към функции

Калин Георгиев

26 февруари 2016 г.

Шаблони (на функции)

Еднообразни функции за различни типове

```
int findIndexMax
(int arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] < arr[i])
            indexMax = i;
}
```

```
return indexMax;
}
```

```
int findIndexMax
(char arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] < arr[i])
            indexMax = i;
}
```

```
return indexMax;
}
```

```
int findIndexMax
(double arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] < arr[i])
            indexMax = i;
}
```

```
return indexMax;
}
```

...

Създаване на “Шаблон на функция”

```
template <typename T>
int findIndexMax (T arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] < arr[i])
            indexMax = i;

    return indexMax;
}
```

Използване на шаблона на функция

```

int main ()
{
    int arri[] = {1,5,6,7};
    cout << findIndexMax<int> (arri,4);

    double arrd[] = {2.1,17.5,6.0};
    cout << findIndexMax<double> (arrd,3);

    char arrc[] = "Hello";
    cout << findIndexMax<char> (arrc,5);

    char* arrstr[] = {"Hello", "World", "!"};
    cout << findIndexMax<char*> (arrstr,3); //!!!
}

template <typename T>
int findIndexMax
    (T arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] < arr[i])
            indexMax = i;

    return indexMax;
}

```

- Конкретния тип трябва да е съвместим с всички операции в шаблона (в горния пример - <)!

Още един пример: печатане на “всякакви” масиви

```
template <typename T>
void printArray (T arr[], int arrSize)
{
    cout << "{"
    for (int i = 0; i < arrSize-1; i++)
        cout << arr[i] << ",";

    if (arrSize > 0) //no comma
        cout << arr[arrSize-1];

    cout << "}";
}
```

Още по-дълбока параметризация: функции като параметри

Пример за еднотипни функции

```
int findIndexMax (int arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] < arr[i])
            indexMax = i;

    return indexMax;
}
```

```
int findIndexMin (int arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (arr[indexMax] > arr[i])
            indexMax = i;

    return indexMax;
}
```


Функции вместо операторите < и >

```
bool compareGt (int a, int b)
{
    return a > b;
}

bool compareLt (int a, int b)
{
    return a < b;
}
```

```
int findIndexMax
(int arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (compareLt (arr[indexMax],arr[i]))
            indexMax = i;

    return indexMax;
}

int findIndexMin
(int arr[], int arrSize)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (compareGt (arr[indexMax],arr[i]))
            indexMax = i;

    return indexMax;
}
```

Функциите имат тип

comparator : int \times int \rightarrow bool

```
bool compareGt (int a, int b)
{return a > b;}
bool compareLt (int a, int b)
{return a < b;}
```

ptrFn : T₁ \times T₂ \times ... \times T_k \rightarrow T_{res}

```
int main (){
    //variable definition:
    //pComparator
    bool (*pComparator) (int,int);

    //pointer assignment
    pComparator = compareLt;
    cout << pComparator (1,2);

    pComparator = compareGt;
    cout << pComparator (1,2);
}
```

*Tres (*ptrFn) (T1,T2,...,Tk);*

Функциите имат тип

comparator : int \times int \rightarrow bool

```
bool compareGt (int a, int b)
{return a > b;}
bool compareLt (int a, int b)
{return a < b;}
```

ptrFn : $T_1 \times T_2 \times \dots \times T_k \rightarrow T_{res}$

```
int main (){
    //variable definition:
    //pComparator
    bool (*pComparator) (int,int);

    //pointer assignment
    pComparator = compareLt;
    cout << pComparator (1,2);

    pComparator = compareGt;
    cout << pComparator (1,2);
}
```

*Tres (*ptrFn) (T1,T2,...,Tk);*

Функциите имат тип

$comparator : int \times int \rightarrow bool$

```
bool compareGt (int a, int b)
{return a > b;}
bool compareLt (int a, int b)
{return a < b;}
```

$ptrFn : T_1 \times T_2 \times \dots \times T_k \rightarrow T_{res}$

```
int main (){
    //variable definition:
    //pComparator
    bool (*pComparator) (int,int);

    //pointer assignment
    pComparator = compareLt;
    cout << pComparator (1,2);

    pComparator = compareGt;
    cout << pComparator (1,2);
}
```

$T_{res} \text{ } (*ptrFn) \text{ } (T_1, T_2, \dots, T_k);$

Функциите имат тип

```
bool compareGt (int a, int b)
{return a > b;}
bool compareLt (int a, int b)
{return a < b;}
```

```
int main (){
    //variable definition:
    //pComparator
    bool (*pComparator) (int,int);

    //pointer assignment
    pComparator = compareLt;
    cout << pComparator (1,2);

    pComparator = compareGt;
    cout << pComparator (1,2);
}
```

$$comparator : int \times int \rightarrow bool$$

$$ptrFn : T_1 \times T_2 \times \dots \times T_k \rightarrow T_{res}$$

$$T_{res} \text{ } (*ptrFn) \text{ } (T_1, T_2, \dots, T_k);$$

Предаване на функции като параметри

```
int findExtremum
(int arr[],
 int arrSize,
 bool (*pComparator)(int,int))
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (pComparator (arr[indexMax],arr[i]))
            indexMax = i;

    return indexMax;
}
```

```
bool compareGt (int a, int b)
{return a > b;}
bool compareLt (int a, int b)
{return a < b;}
```

Предаване на функции като параметри

```
void sort (int arr[],
          int arrSize,
          bool (*pComparator)(int,int))
{
    for (int i = 0; i < arrSize-1; i++)
    {
        //find subarray extremum and
        //swap with a[i]
        swap (arr[i],
              arr[i+findExtremum(arr+i,
                                arrSize-i,
                                pComparator)]);
    }
}
```

Предаване на функции като параметри

```
int main ()
{
    int arr[] = {1,7,3,5,2,3,2,4};

    sort (arr,8,compareLt);
    printArray (arr,8);

    sort (arr,8,compareGt);
    printArray (arr,8);

    return 0;
}
```

```
bool compareGt (int a, int b)
{return a > b;}
bool compareLt (int a, int b)
{return a < b;}

int findExtremum
(int arr[],
 int arrSize,
 bool (*pComparator)(int,int))
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (pComparator (arr[indexMax],arr[i]))
            indexMax = i;
    return indexMax;
}

void sort (int arr[],
           int arrSize,
           bool (*pComparator)(int,int))
{
    for (int i = 0; i < arrSize-1; i++)
    {
        swap (arr[i],
              arr[i+findExtremum(arr+i,
                                  arrSize-i,
                                  pComparator)]);
    }
}
```


Внимание: `-std=c++11`

Предефиниране на типове

Предефиниране на типове

- Използване на “сложен” тип:

```
void doSomething (int myMatrix[10][20])
```

```
int main ()  
{  
    int m[10][20] = {...};  
    doSomething (m);  
}
```

Предефиниране на типове

- “Полагане” на ново име на тип:

```
using myarr = int[10][20];
```

```
void doSomething (myarr myMatrix)
```

```
int main ()
```

```
{  
    myarr m = {...};  
    doSomething (m);  
  
    ///???  
    myarr x[10];  
}
```

```
void doSomething  
    (int myMatrix[10][20])
```

```
int main ()
```

```
{  
    int m[10][20] = {...};  
    doSomething (m);  
}
```

Тип на указател към функцията

Указател към функция

```
using Comparator =  
    bool (*)(int, int);
```

```
bool compareGt (int a, int b)  
{return a > b;}  
bool compareLt (int a, int b)  
{return a < b;}
```

comparator : int × int → bool

Предаване на функции като параметри

```
using Comparator = bool (*)(int,int);

//int findExtremum (int arr[],
//                int arrSize,
//                bool (*pComparator)(int,int));

int findExtremum (int arr[],
                 int arrSize,
                 Comparator pComparator);
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (pComparator (arr[indexMax],arr[i]))
            indexMax = i;

    return indexMax;
}
```

Шаблони на указатели към функции

Шаблон на указател

```

template <typename T>
using Comparator = bool (*)(T,T);

template <typename T>
int findExtremum (int arr[],
                 int arrSize,
                 Comparator<T> pComparator);
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (pComparator (arr[indexMax],arr[i]))
            indexMax = i;

    return indexMax;
}

//int findExtremum
//    (int arr[],
//     int arrSize,
//     bool (*pComparator)(int,int));

using Comparator = bool (*)(int,int);

int findExtremum (int arr[],
                 int arrSize,
                 Comparator pComparator);

```

Пример

```
template <typename T>
bool compareGt (T a, T b)
{return a > b;}
```

```
bool compareGt (char a, char b)
{return a < b;}
```

```
template <typename T>
bool compareLt (T a, T b)
{return a < b;}
```

```
int main ()
{
    int ia[] = {1,3,5};
    double da[] = {1.7,6.5,3.4,5.8};
    char ca = "abz";
```

```
cout << findExtremum<int> (ia,3,compareGt<int>);
cout << findExtremum<double> (da,4,compareGt<double>);
cout << findExtremum<char> (ca,3,compareGt<char>);
```

```
template <typename T>
using Comparator = bool (*)(T,T);

template <typename T>
int findExtremum (T arr[],
                  int arrSize,
                  Comparator<T> pComparator)
{
    int indexMax = 0;
    for (int i = 1; i < arrSize; i++)
        if (pComparator (arr[indexMax],arr[i]))
            indexMax = i;

    return indexMax;
}
```

Map.Reduce

Задача: еднотипна промяна на всеки елемент на масив

```
void increase (int arr[], int arrsize)
{
    for (int i = 0; i < arrsize; i++)
        arr[i] = arr[i]+1;
}
void multiply (int arr[], int arrsize)
{
    for (int i = 0; i < arrsize; i++)
        arr[i] = arr[i]*2;
}
void increaseEvens (int arr[], int arrsize)
{
    for (int i = 0; i < arrsize; i++)
        if (arr[i] %2 == 0)
            arr[i] = arr[i] + 1;
}
```

Map

$$\text{map} : T \rightarrow T$$

- еднотипна обработка на всеки от елементите на масив

```
template <typename T>
using mapFn = T (*) (T);

template <typename T>
void map (T arr[], int arrsize, mapFn<T> f)
{
    for (int i = 0; i < arrsize; i++)
        arr[i] = f(arr[i]);
}
```

Пример: добавяне на единица

```
int plusOne (int x)
{return x+1;}

int multTwo (int x)
{return x*2;}

int main ()
{
    int arr[] = {1,2,3};
    map<int> (arr,3,plusOne);
    map<int> (arr,3,multTwo);

    printArray<int> (arr,3);
}
```

```
template <typename T>
using mapFn = T (*) (T);

template <typename T>
void map (T arr[], int arrsize, mapFn<T> f)
{
    for (int i = 0; i < arrsize; i++)
        arr[i] = f(arr[i]);
}
```

Пример: добавяне на единица само на четните елементи

```
int evenPlusOne (int x)
{
    if (x%2 == 0)
        return x+1;
    return x;
}
```

```
int main ()
{
    int arr[] = {1,2,3};
    map<int> (arr,3,evenPlusOne);

    printArray (arr,3);
}
```

```
template <typename T>
using mapFn = T (*) (T);

template <typename T>
void map (T arr[], int arrsize, mapFn<T> f)
{
    for (int i = 0; i < arrsize; i++)
        arr[i] = f(arr[i]);
}
```

Задача: намиране на сума, произведение, брой и пр.

```
int sum (int arr[], int arrsize)
{
    int result = arr[0];
    for (int i = 1; i < arrsize; i++)
        result = result + arr[i];
    return result;
}

int prod (int arr[], int arrsize)
{
    int result = arr[0];
    for (int i = 1; i < arrsize; i++)
        result = result * arr[i];
    return result;
}

int countEvens (int arr[], int arrsize)
{
    int result = 0;
    for (int i = 1; i < arrsize; i++)
        if (arr[i] % 2 == 0)
            result = result + 1;
    return result;
}
```


Reduce

$$reduce : R \times E \rightarrow R$$

- Сумиране (“акумулиране”, “обединяване”) на всички елементи в един резултат

```
template <typename ResT, typename ElemT>
using reduceFn = ResT (*) (ResT, ElemT);
```

```
template <typename ResT, typename ElemT>
ResT reduce (ElemT arr[],
             int arrsize,
             reduceFn<ResT, ElemT> f,
             ResT init)
{
    ResT result = init;

    for (int i = 0; i < arrsize; i++)
        result = f (result, arr[i]);

    return result;
}
```

Пример: Събиране и умножение

```
int sum (int accumulated, int x)
{return accumulated + x;}
```

```
int prod (int accumulated, int x)
{return accumulated * x;}
```

```
int main ()
{
    int arr[] = {1,2,3};
    cout << reduce<int,int> (arr,3,sum,0);
    cout << reduce<int,int> (arr,3,prod,1);
}
```

```
template <typename ResT, typename ElemT>
using reduceFn = ResT (*) (ResT, ElemT);
```

```
template <typename ResT,typename ElemT>
ResT reduce (ElemT arr[],
             int arrsize,
             reduceFn<ResT,ElemT> f,
             ResT init)
{
    ResT result = init;

    for (int i = 1; i < arrsize; i++)
        result = f (result,arr[i]);

    return result;
}
```

Пример: Събиране само на четните числа

```

int sumEvens (int accumulated, int x)
{
    if (x % 2 == 0)
        return accumulated + x;
    return accumulated;
}

int main ()
{
    int arr[] = {1,2,3};
    cout << reduce<int,int> (arr,3,sumEvens,0);
}

template <typename ResT, typename ElemT>
using reduceFn = ResT (*) (ResT, ElemT);

template <typename ResT,typename ElemT>
ResT reduce (ElemT arr[],
             int arrsize,
             reduceFn<ResT,ElemT> f,
             ResT init)
{
    ResT result = init;

    for (int i = 1; i < arrsize; i++)
        result = f (result,arr[i]);

    return result;
}

```

Пример: Проверка дали има четни числа

```
bool isEven (bool accumulated, int x)
{
    if (x % 2 == 0)
        return true;
    return accumulated;
}
```

```
int main ()
{
    int arr[] = {1,2,3};
    cout << reduce<bool,int> (arr,3,isEven,false);
}
```

```
template <typename ResT, typename ElemT>
using reduceFn = ResT (*) (ResT, ElemT);
```

```
template <typename ResT,typename ElemT>
ResT reduce (ElemT arr[],
             int arrsize,
             reduceFn<ResT,ElemT> f,
             ResT init)
{
    ResT result = init;

    for (int i = 1; i < arrsize; i++)
        result = f (result,arr[i]);

    return result;
}
```

Пример: Брой срещания на символ

```
int countLs (int accumulated, char x)
{
    if (x == 'l')
        return accumulated + 1;
    return accumulated;
}
```

```
int main ()
{
    cout << reduce<int, char> ("Hello_!World!", 12, countLs, 0);
}
```

```
template <typename ResT, typename ElemT>
using reduceFn = ResT (*) (ResT, ElemT);
```

```
template <typename ResT, typename ElemT>
ResT reduce (ElemT arr[],
             int arrsize,
             reduceFn<ResT, ElemT> f,
             ResT init)
{
    ResT result = init;

    for (int i = 1; i < arrsize; i++)
        result = f (result, arr[i]);

    return result;
}
```

Благодаря за вниманието!