

# Указатели. Матриси, указатели, параметри на функции

Калин Георгиев

9 ноември 2015 г.

Указатели!

# Дефиниране

```
long a=1,b=2;  
long *pi = &a;
```

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

## Операции: “през” указател, С указател

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

`*pi = 10;`

...	100	104	108	...
...	a	b	pi	...
...	10	2	100	...

`pi = 104; //won't compile`

...	100	104	108	...
...	a	b	pi	...
...	10	2	104	...

`*pi = 104; //won't compile`

...	100	104	108	...
...	a	b	pi	...
...	10	104	104	...

## Операции: “през” указател, C указател

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

`*pi = 10;`

...	100	104	108	...
...	a	b	pi	...
...	10	2	100	...

`pi = 104; //won't compile`

...	100	104	108	...
...	a	b	pi	...
...	10	2	104	...

`*pi = 104; //won't compile`

...	100	104	108	...
...	a	b	pi	...
...	10	104	104	...

## Операции: “през” указател, C указател

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

`*pi = 10;`

...	100	104	108	...
...	a	b	pi	...
...	10	2	100	...

`pi = 104; //won't compile`

...	100	104	108	...
...	a	b	pi	...
...	10	2	104	...

`*pi = 104; //won't compile`

...	100	104	108	...
...	a	b	pi	...
...	10	104	104	...

## Операции: “през” указател, C указател

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

\*pi = 10;

...	100	104	108	...
...	a	b	pi	...
...	10	2	100	...

```
pi = 104; //won't compile
```

...	100	104	108	...
...	a	b	pi	...
...	10	2	104	...

```
*pi = 104; //won't compile
```

...	100	104	108	...
...	a	b	pi	...
...	10	104	104	...

## Операции: “през” указател, С указател

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

\*pi = 10;

...	100	104	108	...
...	a	b	pi	...
...	10	2	100	...

pi = 104; //won't compile

...	100	104	108	...
...	a	b	pi	...
...	10	2	104	...

\*pi = 104; //won't compile

...	100	104	108	...
...	a	b	pi	...
...	10	104	104	...



## Операции: “през” указател, C указател

...	100	104	108	...
...	a	b	pi	...
...	1	2	100	...

\*pi = 10;

...	100	104	108	...
...	a	b	pi	...
...	10	2	100	...

pi = 104; //won't compile

...	100	104	108	...
...	a	b	pi	...
...	10	2	104	...

\*pi = 104; //won't compile

...	100	104	108	...
...	a	b	pi	...
...	10	104	104	...

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10

# Пример с функции

```
void f (long x)
{
    x = x + 10; //(2')
    cout << x;
}

void g (long *px)
{
    *px = *px + 10; //(5)
    cout << *px;
}

long main ()
{
    long x = 0; //(1)
    f(x);        //(2)
    cout << x;   //(3)
    g (&x);      //(4)
    cout << x;   //(6)
}
```

(1)main:

x | 0

(2,2')main:

x | 0

f:

x | 0

x | 10

(3)main:

x | 0

(4)main:

x | 0

g:

px | 100

(5)main:

x | 10

g:

px | 100

(6)main:

x | 10



# Параметър по стойност vs. по указател

```
long division (long x, long y, long *remainder)
{
    *remainder = x - (x/y)*y;
    return x/y;
}

long main ()
{
    long r = 0;
    cout << " [14/4]="
         << division (14,4,&r)
         << " |14/4|="
         << r
         << endl;
}
```

## Какво е нередно в този пример?

```
long division (long x, long y, long *remainder)  
.....
```

```
cout << division (2,2,0);
```

# Масиви и указатели

# Какво е a? Какво е arr?

```
long a = 5; b = 10; long arr[3] = {1,2,3}; arr2[3] = {4,5,6};
```

- a и b: заместител на адреса / “наименование” на буфера

```
a = 2; a = a + 3; b = a + b; if (a > b) {...}
```

- Операциите с масиви са операции с елементите им
- Отеделните елементи имат свойства на обикновени променливи от съответния тип

```
arr[0] = 2; arr[0] = arr[1] + 3; ...
```

- Какво са arr и arr2 сами по себе си?

```
arr = ???; arr2 = arr???; if (arr > arr2) ???
```

## Какво е a? Какво е arr?

```
long a = 5; b = 10; long arr[3] = {1,2,3}; arr2[3] = {4,5,6};
```

- a и b: заместител на адреса / “наименование” на буфера

```
a = 2; a = a + 3; b = a + b; if (a > b) {...}
```

- Операциите с масиви са операции с елементите им
- Отеделните елементи имат свойства на обикновени променливи от съответния тип

```
arr[0] = 2; arr[0] = arr[1] + 3; ...
```

- Какво са arr и arr2 сами по себе си?

```
arr = ???; arr2 = arr???; if (arr > arr2) ???
```

## Какво е a? Какво е arr?

```
long a = 5; b = 10; long arr[3] = {1,2,3}; arr2[3] = {4,5,6};
```

- a и b: заместител на адреса / “наименование” на буфера

```
a = 2; a = a + 3; b = a + b; if (a > b) {...}
```

- Операциите с масиви са операции с елементите им
- Отеделните елементи имат свойства на обикновени променливи от съответния тип

```
arr[0] = 2; arr[0] = arr[1] + 3; ...
```

- Какво са arr и arr2 сами по себе си?

```
arr = ???; arr2 = arr???; if (arr > arr2) ???
```

## Какво е a? Какво е arr?

```
long a = 5; b = 10; long arr[3] = {1,2,3}; arr2[3] = {4,5,6};
```

- a и b: заместител на адреса / “наименование” на буфера

```
a = 2; a = a + 3; b = a + b; if (a > b) {...}
```

- Операциите с масиви са операции с елементите им
- Отеделните елементи имат свойства на обикновени променливи от съответния тип

```
arr[0] = 2; arr[0] = arr[1] + 3; ...
```

- Какво са arr и arr2 сами по себе си?

```
arr = ???; arr2 = arr???; if (arr > arr2) ???
```

# Указател към първия елемент

```
long arr[3] = {1,2,3};
```

arr	...	*arr			
80	...	100	104	108	...
100	...	1	2	3	...



# Указател към първия елемент

```
long arr[3] = {1,2,3};
```

arr	...	*arr			
80	...	100	104	108	...
100	...	1	2	3	...

# Операции

arr	...	*arr			
80	...	100	104	108	...
100	...	1	2	3	...

- Отново “писане и четене през” указател

```
*arr = 10;  
cout << *arr;
```

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	3	...

# Операции

arr	...	*arr			
80	...	100	104	108	...
100	...	1	2	3	...

- Отново “писане и четене през” указател

```
*arr = 10;  
cout << *arr;
```

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	3	...

# Адресна аритметика

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	3	...

- Аритметични операции с указател:  
<указател>+<ест. число>=<указател>

`*(arr+2) = 10;`

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	10	...

# Адресна аритметика

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	3	...

- Аритметични операции с указател:  
<указател>+<ест. число>=<указател>

`*(arr+2) = 10;`

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	10	...

## Още за адресната аритметика

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	10	...

- `arr+i == &arr[i]`
- `*(arr+i) <==> arr[i]`
- `(long)(arr+1)==104 // != 101`

```
char charArr[6] = "Hello";
```

- `(long)(charArr+1) - (long)(charArr+2) == 1 == sizeof(char)`
- `(long)(arr+1) - (long)(arr+2) == 4 == sizeof(long)`
- Следователно `arr[i] <==> *(arr+i)`, независимо от размера на елементите

## Още за адресната аритметика

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	10	...

- `arr+i == &arr[i]`
- `*(arr+i) <==> arr[i]`
- `(long)(arr+1)==104 // != 101`

```
char charArr[6] = "Hello";
```

- `(long)(charArr+1) - (long)(charArr+2) == 1 == sizeof(char)`
- `(long)(arr+1) - (long)(arr+2) == 4 == sizeof(long)`
- Следователно `arr[i] <==> *(arr+i)`, независимо от размера на елементите

# Още за адресната аритметика

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	10	...

- `arr+i == &arr[i]`
- `*(arr+i) <==> arr[i]`
- `(long)(arr+1)==104 // != 101`

```
char charArr[6] = "Hello";
```

- `(long)(charArr+1) - (long)(charArr+2) == 1 == sizeof(char)`
- `(long)(arr+1) - (long)(arr+2) == 4 == sizeof(long)`
- Следователно `arr[i] <==> *(arr+i)`, независимо от размера на елементите



## Още за адресната аритметика

arr	...	*arr			
80	...	100	104	108	...
100	...	10	2	10	...

- `arr+i == &arr[i]`
- `*(arr+i) <==> arr[i]`
- `(long)(arr+1)==104 // != 101`

```
char charArr[6] = "Hello";
```

- `(long)(charArr+1) - (long)(charArr+2) == 1 == sizeof(char)`
- `(long)(arr+1) - (long)(arr+2) == 4 == sizeof(long)`
- Следователно `arr[i] <==> *(arr+i)`, независимо от размера на елементите

# Масиви и функции

```
long sum (long arr[])
{
    long sum = 0;
    for (int i = 0; i < 3; i++)
    {
        sum += arr[i];
    }
    return sum;
}

int main ()
{
    int a[3] = {1,2,3};
    int b[4] = {1,2,3,4};

    cout << sum(a) << " " << sum (b) << endl;
}
```

## Не можем да определим размера на масиви динамично

```
long sum (long arr[], int n)
{

    long sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += arr[i];
    }
    return sum;
}

int main ()
{
    int a[3] = {1,2,3};
    int b[4] = {1,2,3,4};

    cout << sum(a,3) << " " << sum (b,4) << endl;
}
```

## "Подмасиви"

	c		c+2							
...	1	2	3	4	5	6	7	8	9	...

```
int main ()
{
    int c[8] = {1,2,3,4,6,7,8,9};

    cout << sum(c,4);
    cout << sum(c+2,4);
}
```

# Внимание! Странични ефекти!

```
void test (long arr[], long x)
//void test (long *arr, long x)
{

    x++;
    arr[0]++; // <==> *arr=*arr+1
    arr[1]++; // <==> *(arr+1)=*(arr+1)+1

}

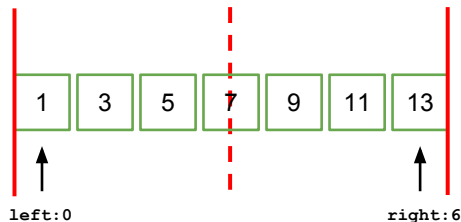
int main ()
{

    long arr[2]={0,1};
    long x = 0;
    test (arr,x);
    cout << arr[0];
    cout << x;
```

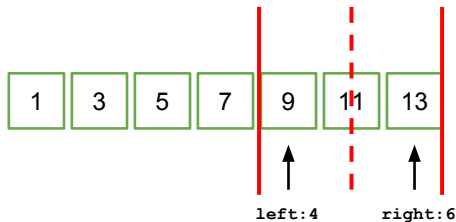
Пример: Двоично търсене

# Алгоритъм на двоичното търсене

#0 left:0 right:6 a[3]<11



#1 left:4 right:6 a[5]==11

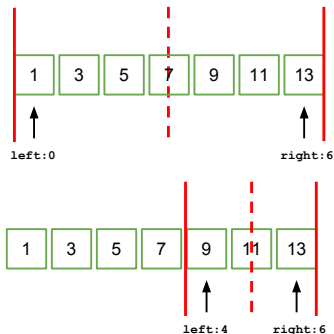


# Реализация с цикъл

```
bool find (int x, int a[], int size)
{
    int left=0, right = size-1;

    while (left < right &&
           a[(left+right)/2] != x)
    {
        if (a[(left+right)/2] < x)
            left = (left+right)/2 + 1;
        else
            right = (left+right)/2;
    }

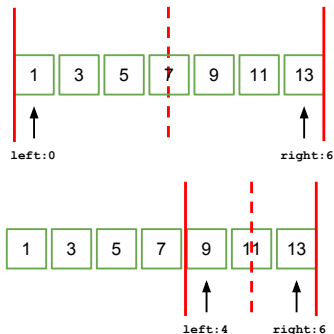
    return a[(left+right)/2] == x;
}
```





# Реализация с рекурсия

```
bool findrec (int x, int a[], int size)
{
    if (size == 0)
    {
        return false;
    }
    if (size == 1)
    {
        return a[0] == x;
    }
    if (a[size/2] > x)
    {
        return findrec (x,a,size/2);
    }
    if (a[size/2] < x)
    {
        return findrec (x,a+(int)ceil(size/2.0),ceil(size/2.0)-1);
    }
    return true;
}
```



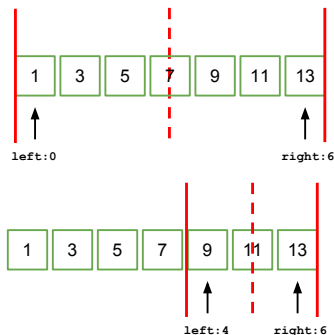
# А това защо работи?

```
bool findreclogic (int x, int a[], int size)
{
    return
        (size == 1 &&
         a[0] == x) ||

        (size > 1 &&
         a[size/2] > x &&
         findreclogic (x,a,size/2)) ||

        (size > 1 &&
         a[size/2] < x &&
         findreclogic (x,
                        a+((int)ceil(size/2.0),
                           ceil(size/2.0)-1)) ||

        (size > 1 &&
         a[size/2] == x);
}
```



Благодаря за вниманието!