C 语言

数组与指针

张晓平

武汉大学数学与统计学院

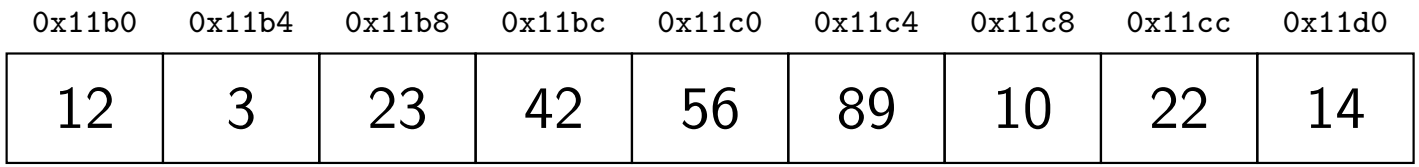
2018 年 5 月 17 日

数组

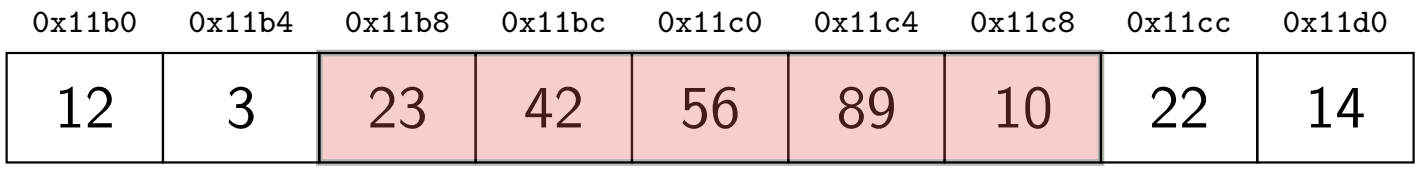
数组由一系列类型相同的元素组成，数组声明必须包括元素的个数（即数组长度）与类型。如

```
float  farray[20];
char  carray[12];
int  iarray[50];
```

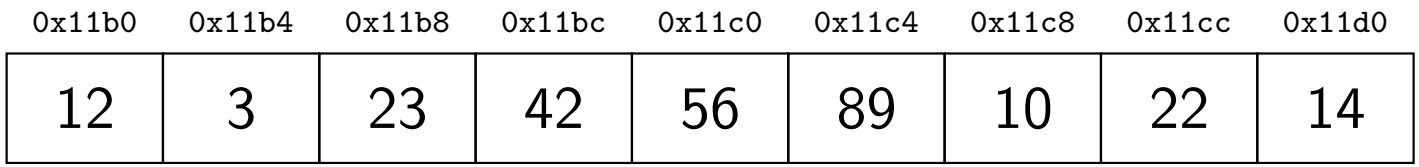
数组初始化



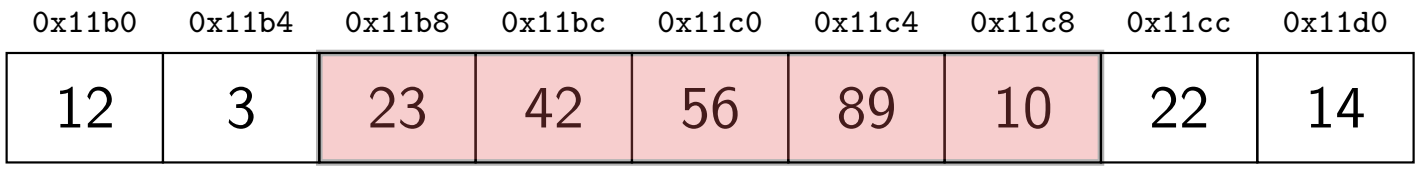

```
int array[5];
```



数组初始化




```
int array[5];
```



数组名表示的是数组首元素的地址。故 `array` 的值为 `0x11b8` 。

数组初始化

0x11b0	0x11b4	0x11b8	0x11bc	0x11c0	0x11c4	0x11c8	0x11cc	0x11d0
12	3	23	42	56	89	10	22	14

```
int array[5] = {1,2,3};
```

0x11b0	0x11b4	0x11b8	0x11bc	0x11c0	0x11c4	0x11c8	0x11cc	0x11d0
12	3	1	2	3	0	0	22	14

数组初始化

0x11b0	0x11b4	0x11b8	0x11bc	0x11c0	0x11c4	0x11c8	0x11cc	0x11d0
12	3	23	42	56	89	10	22	14

```
int array[5] = {2, [2]=5, 6, [0]=8};
```

0x11b0	0x11b4	0x11b8	0x11bc	0x11c0	0x11c4	0x11c8	0x11cc	0x11d0
12	3	8	0	5	6	0	22	14

指针

- 计算机的硬件指令很大程度上依赖于地址，而指针为我们使用地址提供了一种方法。
- 使用指针能让我们以类似于计算机底层的方式来表达意愿，从而让程序能更高效地工作。
- 需要强调的是，指针能非常有效地处理数组。事实上，数组是一种变相使用指针的形式。

指针

再一次强调：数组名是数组首元素的地址。

指针

再一次强调：数组名是数组首元素的地址。

若 array 为一个数组，则以下关系式为真：

```
array == &array[0];
```

指针 I


```
1  #include <stdio.h>
2  #define SIZE 4
3  int main(void)
4  {
5    short dates [SIZE];
6    short * pti;
7    short index;
8    double bills[SIZE];
9    double *ptf;
10
11   pti = dates;
```

指针 II


```
12    ptf = bills;
13    printf("%20s %6s\n", "short", "double");
14    for (index = 0; index < SIZE; index ++)
15      printf("pointers + %d: %p %p\n",index,
          pti + index, ptf + index);
16
17    return 0;
18  }
```

指针

```
                    short  double
pointers + 0:  0xf7d0  0xf7b0
pointers + 1:  0xf7d2  0xf7b8
pointers + 2:  0xf7d4  0xf7c0
pointers + 3:  0xf7d6  0xf7c8
```

指针

在 C 中，对指针加 1 的结果是对该指针增加一个存储单元。对数组而言，地址会增加到下一个元素的地址，而不是下一个字节。

指针定义小结

- 指针的数值就是它所指向的对象的地址。地址的内部表达方式由硬件决定，很多计算机都是以字节编址的。
- 在指针前用运算符 * 就可以得到该指针所指向的对象的值。
- 对指针加 1，等价于对指针的值加上它所指向的对象的字节大小。

指针

```
dates + 2 == &dates[2];     // same address
*(dates + 2) == dates[2];   // same value
```

可以用指针标识数组的每个元素，并得到每个元素的值。从本质上讲，这是对同一对象采用了两种不同的符号表示方法。

指针

在描述数组时，C 确实借助了指针的概念。例如，定义 `array[n]` 时，

- 即：`*(array + n)`，
- 含义：“寻址到内存中的 `array`，然后移动 `n` 个单元，再取出数值”。

指针

请注意 *(dates+2)和 *dates+2 的区别。取值运算符* 的优先级高于+ ，故后者等价于 (*dates)+2 。

```
*(dates + 2)   //   same as dates[2]
*dates + 2     //   same as dates[0] + 2
```

数组、指针与函数

编写函数，求一个数组的各元素之和。

数组、指针与函数

方式一：在函数中给定固定的数组大小。

```
int sum(int * ar)
{
  int i, total = 0;

  for (i = 0; i < 10; i++)
    total += ar[i];
  return total;
}
```

数组、指针与函数

方式一：在函数中给定固定的数组大小。

```
int sum(int * ar)
{
  int i, total = 0;

  for (i = 0; i < 10; i++)
    total += ar[i];
  return total;
}
```

但该函数仅在数组长度为 10 时可工作。

数组、指针与函数

方式二：将数组大小作为参数传递给函数。

```
int sum(int * ar, int n)
{
  int i, total = 0;

  for (i = 0; i < n; i++)
    total += ar[i];
  return total;
}
```

数组、指针与函数

方式二：将数组大小作为参数传递给函数。

```
int sum(int * ar, int n)
{
  int i, total = 0;

  for (i = 0; i < n; i++)
    total += ar[i];
  return total;
}
```

该方式更为灵活，第一个参数把数组地址和数组类型的信息传递给函数，第二个参数把数组的元素个数传递给函数。

数组、指针与函数

给定数组

```
int ar[5] = {1, 2, 3, 4, 5};
```

- 可以求数组的整体和，如

```
int result = sum(ar, 5);
```

- 也可以求数组的部分和，如

```
int result = sum(ar+1, 3);
```

计算的是ar的第 2 个至第 4 个元素之和。

数组、指针与函数

在做函数声明时，以下四种函数原型是等价的：

```
int sum(int * ar, int n);
int sum(int *, int);

int sum(int ar[], int n);
int sum(int [], int);
```

数组、指针与函数

在定义函数时，名称不可以省略。故在定义时以下两种形式是等价的：

```
int sum(int * ar, int n)
{
   ...
}

int sum(int ar[], int n)
{
   ...
}
```