程序设计与算法(一)

李文新 郭炜

指针(一)

指针的基本概念

- 每个变量都被存放在从某个内存地址（以字节为单位）开始的若干个字节中

指针的基本概念

- 每个变量都被存放在从某个内存地址（以字节为单位）开始的若干个字节中

- “指针”，也称作“指针变量”，大小为4个字节（或8个字节）的变量，其内容代表一个内存地址。

指针的基本概念

●每个变量都被存放在从某个内存地址（以字节为单位）开始的若干个字节中

●“指针”，也称作“指针变量”，大小为4个字节（或8个字节）的变量，其内容代表一个内存地址。

●通过指针，能够对该指针指向的内存区域进行读写。

指针的基本概念

●每个变量都被存放在从某个内存地址（以字节为单位）开始的若干个字节中

●“指针”，也称作“指针变量”，大小为4个字节（或8个字节）的变量，其内容代表一个内存地址。

●通过指针，能够对该指针指向的内存区域进行读写。

●如果把内存的每个字节都想像成宾馆的一个房间，那么内存地址相当于就是房间号，而指针里存放的，就是房间号。

指针的定义

类型名 * 指针变量名;

```
int * p;          //  p 是一个指针，变量 p的类型是 int *
char * pc;        //  pc 是一个指针，变量 pc 的类型是 char *
float *pf;        // pf 是一个指针，变量 pf 的类型是 float *
```

指针的内容

```
int * p = ( int * ) 40000;
```

p内容:

十进制	40000
十六进制	0x9C40
二进制每个比特	0000 0000 0000 0000 1001 1100 0100 0000

p指向地址40000，地址p就是地址40000

指针的内容

```
int * p = ( int * ) 40000;
```

p内容:

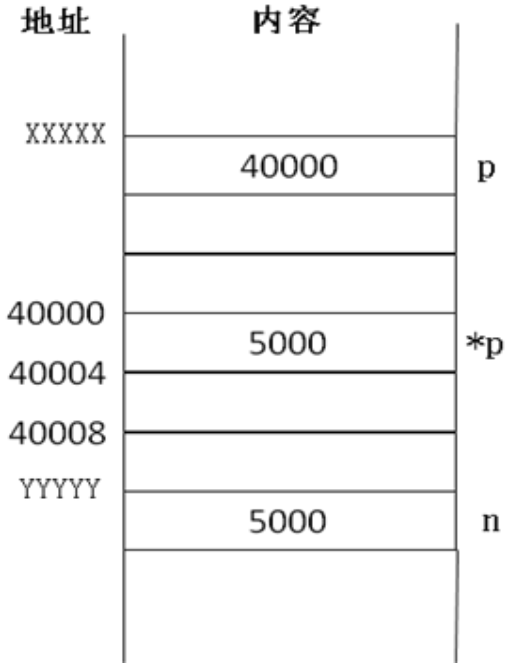
十进制	40000
十六进制	0x9C40
二进制每个比特	0000 0000 0000 0000 1001 1100 0100 0000

p指向地址40000，地址p就是地址40000

*p 就代表地址40000开始处的若干个字节的内容

通过指针访问其指向的内存空间

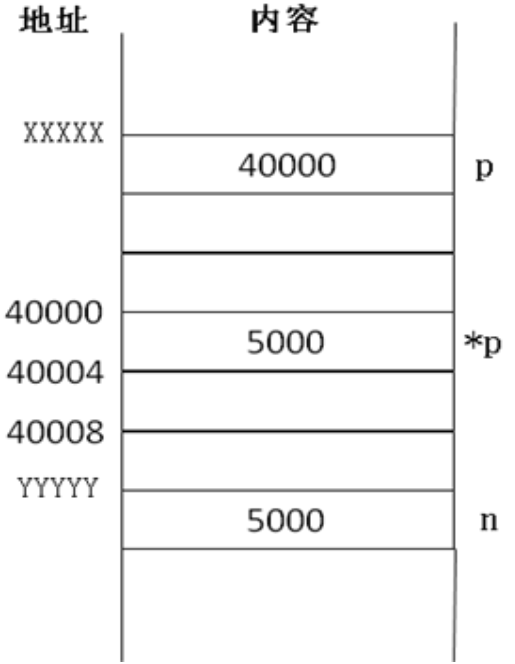
```
int * p = ( int * ) 40000;
//往地址40000处起始的若干个字节的内存空间里写入 5000
*p = 5000;
//将地址40000处起始的若干字节的内容赋值给 n
int n = *p;
```



通过指针访问其指向的内存空间

```
int * p = ( int * ) 40000;
//往地址40000处起始的若干个字节的内存空间里写入 5000
*p = 5000;
//将地址40000处起始的若干字节的内容赋值给 n
int n = *p;
```

“若干” = sizeof(int), 因为 int * p;



指针定义总结

```
T * p ;          // T 可以是任何类型的名字，比如 int, double ,char 等等。
```

p　　的类型：　T *
*p　　的类型：　T

通过表达式　* p，可以读写从地址p开始的 sizeof(T)个字节

*p　等价于存放在地址p处的一个 T 类型的变量

*　　间接引用运算符
sizeof(T*)　　4字节（64位计算机上可能8字节）

指针用法

```
char ch1 = 'A';
char * pc = &ch1; //使得pc指向变量ch1
```

& ： 取地址运算符

&x ： 变量x的地址（即指向x的指针）

对于类型为 T 的变量 x，&x 表示变量 x 的地址(即指向x的指针)
&x 的类型是 T *。

指针用法

```
char ch1 = 'A';
char * pc = &ch1; // 使得pc 指向变量ch1
* pc = 'B';       // 使得ch1 = 'B'
```

指针用法

```
char ch1 = 'A';
char * pc = &ch1; // 使得pc 指向变量ch1
* pc = 'B';       // 使得ch1 = 'B'
char ch2 = * pc;  // 使得ch2 = ch1
```

指针用法

```
char ch1 = 'A';
char * pc = &ch1; // 使得pc 指向变量ch1
* pc = 'B';       // 使得ch1 = 'B'
char ch2 = * pc;  // 使得ch2 = ch1
pc = & ch2;       // 使得pc 指向变量ch2
```

指针用法

```
char ch1 = 'A';
char * pc = &ch1; // 使得pc 指向变量ch1
* pc = 'B';       // 使得ch1 = 'B'
char ch2 = * pc;  // 使得ch2 = ch1
pc = & ch2;       // 使得pc 指向变量ch2
* pc = 'D';       // 使得ch2 = 'D'
```

指针的作用

有了指针，就有了自由访问内存空间的手段

指针的作用

有了指针，就有了自由访问内存空间的手段

- 不需要通过变量，就能对内存直接进行操作。通过指针，程序能访问的内存区域就不仅限于变量所占据的数据区域

指针的作用

有了指针，就有了自由访问内存空间的手段

●不需要通过变量，就能对内存直接进行操作。通过指针，程序能访问的内存区域就不仅限于变量所占据的数据区域

●在C++中，用指针p指向a的地址,然后对p进行加减操作，p就能指向a后面或前面的内存区域，通过p也就能访问这些内存区域

指针的互相赋值

不同类型的指针，如果不经过强制类型转换，不能直接互相赋值

```
int * pn, char * pc, char c = 0x65;
pn = pc;                    //类型不匹配，编译出错
pn = & c;                   //类型不匹配，编译出错
```

指针的互相赋值

不同类型的指针，如果不经过强制类型转换，不能直接互相赋值

```
int * pn, char * pc, char c = 0x65;
pn = pc;                    //类型不匹配，编译出错
pn = & c;                   //类型不匹配，编译出错
pn = (int * ) & c;
```

指针的互相赋值

不同类型的指针，如果不经过强制类型转换，不能直接互相赋值

```
int * pn, char * pc, char c = 0x65;
pn = pc;                    //类型不匹配，编译出错
pn = & c;                   //类型不匹配，编译出错
pn = (int * ) & c;
int n = * pn;                       //n值不确定
* pn = 0x12345678;
```

指针的互相赋值

不同类型的指针，如果不经过强制类型转换，不能直接互相赋值

```
int * pn, char * pc, char c = 0x65;
pn = pc;                    //类型不匹配，编译出错
pn = & c;                   //类型不匹配，编译出错
pn = (int * ) & c;
int n = * pn;               //n值不确定
* pn = 0x12345678;          //编译能过但运行可能出错
```

指针的运算

1） 两个同类型的指针变量，可以比较大小

地址p1<地址p2，⇔　p1< p2　值为真。
地址p1=地址p2，⇔　p1== p2　值为真
地址p1>地址p2，⇔　p1 > p2　值为真

指针的运算

2）两个同类型的指针变量，可以相减

两个T * 类型的指针 p1和p2

p1 - p2 = (地址p1 - 地址 p2) / sizeof(T)

指针的运算

3) 指针变量加减一个整数的结果是指针

p ： T * 类型的指针
n ： 整数类型的变量或常量

➔

p+n ： T * 类型的指针，指向地址：
地址p + n × sizeof(T)

n+p， p-n ， *(p+n)， *(p-n) 含义自明

指针的运算

4） 指针变量可以自增、自减

T* 类型的指针p指向地址n

➔

`p++, ++p` ：p指向 **`n + sizeof(T)`**
`p--, --p` ：p指向 **`n - sizeof(T)`**

指针的运算

5）指针可以用下标运算符“[ ]”进行运算

p 是一个 T * 类型的指针，
n 是整数类型的变量或常量

➔

`p[n]` 等价于 `*(p+n)`

通过指针实现自由内存访问

如何访问int型变量 a 前面的那一个字节?

通过指针实现自由内存访问

如何访问int型变量 a 前面的那一个字节?

```
int a;
char * p = (char * ) &a;   // &a是 int *类型
--p;
printf("%c", * p); //可能导致运行错误
* p = 'A';  //可能导致运行错误
```

指针运算示例

```
#include <iostream>
using namespace std;
int main()  {
        int * p1, * p2;  int n = 4;
        char * pc1, * pc2;
        p1 = (int *) 100;                                   //地址p1为100
        p2 = (int *) 200;                                   //地址p2为200
        cout<< "1) " << p2 - p1 << endl;
        //输出 1) 25, 因(200-100)/sizeof(int) = 100/25 = 4
        pc1 = (char * ) p1;                                 //地址pc1为100
        pc2 = (char * ) p2;                                 //地址pc2为200
        cout<< "2) " << pc1 - pc2 << endl;                  //输出 2) -100
        //输出 2) -100,因为(100-200)/sizeof(char) = -100
        cout<< "3) " << (p2 + n) - p1 << endl;              //输出 3) 29
        int * p3 = p2 + n;      // p2 + n 是一个指针，可以用它给 p3赋值
        cout<< "4) " << p3 - p1 << endl;                    //输出 4) 29
        cout<< "5) " << (pc2 - 10) - pc1 << endl;   //输出 5) 90
        return 0;
}
```

空指针

- 地址0不能访问。指向地址0的指针就是空指针

空指针

●地址0不能访问。指向地址0的指针就是空指针

●可以用“NULL”关键字对任何类型的指针进行赋值。NULL实际上就是整数0,值为NULL的指针就是空指针:

```
int * pn = NULL; char * pc = NULL; int * p2 = 0;
```

空指针

●地址0不能访问。指向地址0的指针就是空指针

●可以用“NULL”关键字对任何类型的指针进行赋值。NULL实际上就是整数0,值为NULL的指针就是空指针:

```
int * pn = NULL; char * pc = NULL; int * p2 = 0;
```

●指针可以作为条件表达式使用。如果指针的值为NULL，则相当于为假，值不为NULL，就相当于为真

```
if(p) ⇔ if(p!=NULL)          if(!p) ⇔ if( p==NULL )
```

指针作为函数参数

```
#include <iostream>
using namespace std;
void Swap( int *p1,  int * p2)
{
        int tmp = *p1;          // 将p1指向的变量的值，赋给tmp
        *p1 = *p2;              // 将p2指向的变量的值，赋给p1指向的变量
        *p2 = tmp;              // 将tmp 的值赋给p2指向的变量。
}
int main()
{
    int m = 3,n = 4;
    Swap( &m, &n); //使得p1指向m,p2指向n
    cout << m << " " << n << endl; //输出 4 3
    return 0;
}
```

指针和数组

- 数组的名字是一个指针常量
  指向数组的起始地址

```
T a[N];
```

  - a的类型是 T *

指针和数组

- 数组的名字是一个指针常量
  指向数组的起始地址

```
T a[N];
```

  - a的类型是 T *
  - 可以用a给一个T * 类型的指针赋值

指针和数组

- 数组的名字是一个指针常量
  指向数组的起始地址

```
T a[N];
```

- a的类型是 T *
- 可以用a给一个T * 类型的指针赋值
- a是编译时其值就确定了的常量，不能够对a进行赋值

指针和数组

- 作为函数形参时，T *p 和 T p[ ] 等价

```
void Func( int * p) { cout << sizeof(p);}
```

⇔

```
void Func( int p[]) { cout << sizeof(p);}
```

指针和数组

```
#include <iostream>
using namespace std;
int main()  {
        int a[200];   int * p ;
        p = a;                  // p指向数组a的起始地址，亦即p指向了a[0]
        * p = 10;               //使得a[0] = 10
        *( p + 1 ) = 20;        //使得 a[1] = 20
        p[0] = 30;              //p[i] 和 *(p+i) 是等效的，使得a[0] = 30
        p[4] = 40;              //使得 a[4] = 40
        for( int i = 0;i < 10; ++i) //对数组a的前10个元素进行赋值
          *( p + i) = i;
        ++p;                    // p指向 a[1]
        cout << p[0] << endl; //输出1  p[0]等效于*p，p[0]即是a[1]
        p = a + 6;              // p指向a[6]
        cout << * p << endl;// 输出 6
        return 0;
}
```

指针和数组

```
#include <iostream>
using namespace std;
void Reverse(int * p,int size) { //颠倒一个数组
        for(int i = 0;i < size/2; ++i) {
                int tmp = p[i];
                p[i] = p[size-1-i];
                p[size-1-i] = tmp;
        }
}
int main()
{
        int a[5] = {1,2,3,4,5};
        Reverse(a,sizeof(a)/sizeof(int));
        for(int i = 0;i < 5; ++i) {
            cout << *(a+i) << "," ;
        }
        return 0;
} => 5,4,3,2,1,
```