C++指针详解

搞清指针的四方面的内容：指针的类型，指针所指向的类型，指针的值或者叫指针所指向的内存区，还有指针本身所占据的内存区。

例一：

int \*ptr;   
char \*ptr;   
int \*\*ptr;   
int (\*ptr)[3];   
int \*(\*ptr)[4];

**指针的类型**：

从语法的角度看，你只要把指针声明语句里的指针名字去掉，剩下的部分就是这个指针的类型。这是指针本身所具有的类型。让我们看看例一中各个指针的类型：

int \*ptr; //指针的类型是int \*   
char \*ptr; //指针的类型是char \*   
int \*\*ptr; //指针的类型是 int \*\*   
int (\*ptr)[3]; //指针的类型是 int(\*)[3]   
int \*(\*ptr)[4]; //指针的类型是 int \*(\*)[4]

**指针所指向的类型**

当你通过指针来访问指针所指向的内存区时，指针所指向的类型决定了编译器将把那片内存区里的内容当做什么来看待。  
  
从语法上看，你只须把指针声明语句中的指针名字和名字左边的指针声明符\*去掉，剩下的就是指针所指向的类型。例如：

int \*ptr; //指针所指向的类型是int   
char \*ptr; //指针所指向的的类型是char   
int \*\*ptr; //指针所指向的的类型是 int \*   
int (\*ptr)[3]; //指针所指向的的类型是 int()[3]   
int \*(\*ptr)[4]; //指针所指向的的类型是 int \*()[4]

**指针的值：**

指针的值是指针本身存储的数值，这个值将被编译器当作一个地址，而不是一个一般的数值

长度sizeof(指针所指向的类型)

我们说一个指针的值是XX，就相当于说该指针指向了以XX为首地址的一片内存区域；我们说一个指针指向了某块内存区域，就相当于说该指针的值是这块内存区域的首地址。

指针所指向的内存区和指针所指向的类型是两个完全不同的概念。在例一中，指针所指向的类型已经有了，但由于指针还未初始化，所以它所指向的内存区是不存在的，或者说是无意义的。

**指针本身所占据的内存区**

指针本身占了多大的内存？你只要用函数sizeof(指针的类型)

**指针的算术运算**

指针可以加上或减去一个整数。指针的这种运算的意义和通常的数值的加减运算的意义是不一样的。例如：    
例二：

char a[20];   
 int \*ptr=a;   
...   
...   
 ptr++;

在上例中，指针ptr的类型是int\*,它指向的类型是int，它被初始化为指向整形变量a。接下来的第3句中，指针ptr被加了1，编译器是这样处理的：它把指针ptr的值加上了sizeof(int)，在32位程序中，是被加上了4。由于地址是用字节做单位的，故ptr所指向的地址由原来的变量a的地址向高地址方向增加了4个字节。  
由于char类型的长度是一个字节，所以，原来ptr是指向数组a的第0号单元开始的四个字节，此时指向了数组a中从第4号单元开始的四个字节。  
我们可以用一个指针和一个循环来遍历一个数组，看例子：

int array[20];   
int \*ptr=array;   
...   
//此处略去为整型数组赋值的代码。   
...   
for(i=0;i<20;i++)   
{   
 (\*ptr)++;   
 ptr++；   
}

[复制代码](javascript:void(0);)

这个例子将整型数组中各个单元的值加1。由于每次循环都将指针ptr加1，所以每次循环都能访问数组的下一个单元。再看例子：

char a[20];   
int \*ptr = a;   
...   
...   
ptr += 5;

在这个例子中，ptr被加上了5，编译器是这样处理的：将指针ptr的值加上5乘sizeof(int)，在32位程序中就是加上了5乘4=20。由于地址的单位是字节，故现在的ptr所指向的地址比起加5后的ptr所指向的地址来说，向高地址方向移动了20个字节。在这个例子中，没加5前的ptr指向数组a的第0号单元开始的四个字节，加5后，ptr已经指向了数组a的合法范围之外了。虽然这种情况在应用上会出问题，但在语法上却是可以的。这也体现出了指针的灵活性。   
  
如果上例中，ptr是被减去5，那么处理过程大同小异，只不过ptr的值是被减去5乘sizeof(int)，新的ptr指向的地址将比原来的ptr所指向的地址向低地址方向移动了20个字节。   
  
总结一下，一个指针ptrold加上一个整数n后，结果是一个新的指针ptrnew，ptrnew的类型和ptrold的类型相同，ptrnew所指向的类型和ptrold所指向的类型也相同。ptrnew的值将比ptrold的值增加了n乘sizeof(ptrold所指向的类型)个字节。就是说，ptrnew所指向的内存区将比ptrold所指向的内存区向高地址方向移动了n乘sizeof(ptrold所指向的类型)个字节。一个指针ptrold减去一个整数n后，结果是一个新的指针ptrnew，ptrnew的类型和ptrold的类型相同，ptrnew所指向的类型和ptrold所指向的类型也相同。ptrnew的值将比ptrold的值减少了n乘sizeof(ptrold所指向的类型)个字节，就是说，ptrnew所指向的内存区将比ptrold所指向的内存区向低地址方向移动了n乘sizeof(ptrold所指向的类型)个字节。

**运算符&和\***

这里&是取地址运算符，\*是...书上叫做“间接运算符”。&a的运算结果是一个指针，指针的类型是a的类型加个\*，指针所指向的类型是a的类型，指针所指向的地址嘛，那就是a的地址。\*p的运算结果就五花八门了。总之\*p的结果是p所指向的东西，这个东西有这些特点：它的类型是p指向的类型，它所占用的地址是p所指向的地址。  
  
例五：

[复制代码](javascript:void(0);)

int a=12;   
int b;   
int \*p;   
int \*\*ptr;   
p=&a;//&a的结果是一个指针，类型是int\*，指向的类型是int，指向的地址是a的地址。   
\*p=24;//\*p的结果，在这里它的类型是int，它所占用的地址是p所指向的地址，显然，\*p就是变量a。  
ptr=&p;//&p的结果是个指针，该指针的类型是p的类型加个\*，在这里是int\*\*。该指针所指向的类型是p的类型，这里是int\*。该指针所指向的地址就是指针p自己的地址。   
\*ptr=&b;//\*ptr是个指针，&b的结果也是个指针，且这两个指针的类型和所指向的类型是一样的，所以?amp;b来给\*ptr赋值就是毫无问题的了。  
\*\*ptr=34;//\*ptr的结果是ptr所指向的东西，在这里是一个指针，对这个指针再做一次\*运算，结果就是一个int类型的变量。

C++在一定范围内生成不重复随机数序列

int a[100];

for(i=0; i<=99; ++i) a[i]=i;

for(i=99; i>=1; --i) swap(a[i], a[rand()%i]);

上面这段代码只需要遍历一次就可以产生这100个不重复的随机数，它是如何做到的呢？首先第二行按顺

序用0到99填满整个数组；第三行，是随机产生从0到m-2个数组下标，把这个下标的元素值跟m-1下标的元

素值交换，一直进行到下标为1的元素。因此它只需要遍历一次就能产生全部的随机数。