jiangxinyu的专栏

叶子的离开，是因为风的追求还是树的不挽留？

目录视图 摘要视图 RSS 订阅

个人资料

jiangxinyu

访问： 2046471次
积分： 23248
等级： BLOG 7
排名： 第124名

原创： 57篇
转载： 1369篇
译文： 0篇
评论： 309条

文章分类

设计模式 (31)
[你必须知道的.NET] (2)
程序设计 (64)
WPF (7)
软件工程 (9)

数组引用:C++ 数组做参数 深入分析

分类： C++ windows 2012-07-20 13:51 5716人阅读 评论(1) 收藏 举报

c++ vector reference iostream 编译器 语言

"数组引用"以避免"数组降阶"（本文曾贴于VCKBASE\C++论坛）

受[hpho]的一段模板函数的启发，特写此文，如有雷同，实在遗憾。
数组降阶是个讨厌的事，这在C语言中是个无法解决的问题，先看一段代码，了解什么是"数组降阶"

```
#include <IOSTREAM>
using namespace std;

void Test( char array[20] )
{
    cout << sizeof(array) << endl; // 输出 4
}

int main( void )
{
    char array[20] = { 0 };
    cout << sizeof(array) << endl; // 输出 20
    Test( array );
}
```

文章存档

文章搜索

阅读排行

为什么同样申明的array一个输出20一个输出4？这是因为void Test(char array[20])中的array被降阶处理了，void Test(char array[20])等同于void Test(char array[])，也等同于void Test(char* const array)，如果你BT（开玩笑），它也等同于void Test(char array[999])。

就是说

```
void Test( char array[20] )
{
    cout << sizeof(array) << endl;
}
```

被降成

```
void Test( char* const array )
{
    cout << sizeof(array) << endl; // 既然是char*，当然输出4
}
```

这样一来问题大了，你完全可以定义一个不足20个元素的数组，然后传给Test，坐等程序崩溃。在一些要求较高的场合就不能使用数组做参数，真TMD心有不甘。

那么在C语言中怎样解决这个问题？

没办法，应该说没有好办法。a：做个结构，其中仅一个char array[20]，然后用这个结构指针代替char array[20]。可见这是个很繁琐的办法，且不直观；b：在Test内部使用_msize来计算array长度。这更不行，首先它使得错误的发现被推迟到运行期，而不是编译期，其次_msize长度/元素大小>=array长度，也就是说就是new char[19]和new array[20]分配的大小是一样的，这样一来，虽不至于导致程序崩溃，但运算结果却不正确。

感谢[hpho]，受其启发，C++中有C所没有的"引用"，但数组引用是怎样申明的呢？经过几番试验，Look

```
#include <IOSTREAM>
using namespace std;

void Test( char (&array)[20] ) // 是不是很像 char *p[20] 和 char (*p)[20] 的区别？
{
    cout << sizeof(array) << endl;
}

int main( void )
{
    char array[20] = { 0 };
    cout << sizeof(array) << endl;
    Test( array );
}
```

在 C++中，数组永远不会按值传递，它是传递第一个元素，准确地说是第 0个 的指针。

例如，如下声明 ：

```
void putValues( int[ 10 ] );
```

被编译器视为

```
void putValues( int* );
```

数组的长度与参数声明无关，因此，下列三个声明是等价的：

```
// 三个等价的 putValues()声明
void putValues( int* );
void putValues( int[] );
void putValues( int[ 10 ] );
```

因为数组被传递为指针 所以这对程序员有两个含义：

1. 在被调函数内对参数数组的改变将被应用到数组实参上而不是本地拷贝上，当用作实参的数组必须保持不变时，程序员需要保留原始数组的拷贝函数可以通过把参数类型声明为 const 来表明不希望改变数组元素。

```
void putValues( const int[ 10 ] );
```

2. 数组长度不是参数类型的一部分，函数不知道传递给它的数组的实际长度，编泽器也不知道，当编译器对实参类型进行参数类型检查时，并不检查数组的长度。例如：

```
void putValues( int[ 10 ] ); // 视为 int*

int main() {
int i, j[ 2 ];
putValues( &i ); // ok: &i 是 int*; 潜在的运行错误
putValues( j ); // ok: j 被转换成第 0 个元素的指针
// 实参类型为 int*: 潜在的运行错误
return 0;
}
```

参数的类型检查只能保证putValues()的两次调用都提供了int*型的实参，类型检查不能检验实参是一个 10元素的数组 。习惯上， C风格字符串是字符的数组，它用一个空字符编码作为结尾。但是所有其他类型，包括希望处理内含空字符的字符数组必须以某种方式在向函数传递实参时使其知道它的长度。

一种常见的机制是提供一个含有数组长度的额外参数。例如：

```
void putValues( int[], int size );
```

```
int main() {
            int i, j[ 2 ];
            putValues( &i, 1 );
            putValues( j, 2 );
            return 0;
}
```

另外一种机制是将参数声明为数组的引用

当参数是一个数组类型的引用时，数组长度成为参数和实参类型的一部分，编译器检查数组实参的长度与在函数参数类型中指定的长度是否匹配。

```
// 参数为 10 个 int 的数组
// parameter is a reference to an array of 10 ints
void putValues( int (&arr)[10] );//不能写成&arr[10]，因为下标操作符的优先级较高
int main() {
            int i, j[ 2 ];
            putValues( i ); // 错误: 实参不是 10 个 int 的数组
            putValues( j ); // 错误: 实参不是 10 个 int 的数组
            return 0;
}
```

"数组引用"以避免"数组降阶"（本文：http://blog.vckbase.com/bruceteen/archive/2004/05/20/232.aspx）

受[hpho]的一段模板函数的启发，特写此文，如有雷同，实在遗憾。

数组降阶是个讨厌的事，这在C语言中是个无法解决的问题，先看一段代码，了解什么是"数组降阶"

```
#include <IOSTREAM>
using namespace std;

void Test( char array[20] )
{
    cout << sizeof(array) << endl; // 输出 4
}

int main( void )
{
    char array[20] = { 0 };
    cout << sizeof(array) << endl; // 输出 20
    Test( array );
```

codeproject
C/C++开发（网络大本营）
正则表达式引擎下载
易软开源
关于BugFree
NUnit Test
软件工程和研发技术的公益网站
深之JohnChen的专栏

```
}
```

为什么同样申明的array一个输出20一个输出4？这是因为void Test(char array[20])中的array被降阶处理了，void Test(char array[20])等同于void Test(char array[])，也等同于void Test(char* const array)，如果你BT（开玩笑），它也等同于void Test(char array[999])。

就是说

```
void Test( char array[20] )
{
    cout << sizeof(array) << endl;
}
```

被降成

```
void Test( char* const array )
{
    cout << sizeof(array) << endl; // 既然是char*，当然输出4
}
```

这样一来问题大了，你完全可以定义一个不足20个元素的数组，然后传给Test，坐等程序崩溃。在一些要求较高的场合就不能使用数组做参数，真TMD心有不甘。

那么在C语言中怎样解决这个问题？

没办法，应该说没有好办法。a：做个结构，其中仅一个char array[20]，然后用这个结构指针代替char array[20]。可见这是个很繁琐的办法，且不直观；b：在Test内部使用_msize来计算array长度。这更不行，首先它使得错误的发现被推迟到运行期，而不是编译期，其次_msize长度/元素大小>=array长度，也就是说就是new char[19]和new array[20]分配的大小是一样的，这样一来，虽不至于导致程序崩溃，但运算结果却不正确。

感谢[hpho]，受其启发，C++中有C所没有的"引用"，但数组引用是怎样申明的呢？经过几番试验，Look

```
#include <IOSTREAM>
using namespace std;

void Test( char (&array)[20] ) // 是不是很像 char *p[20] 和 char (*p)[20] 的区别？
{
    cout << sizeof(array) << endl;
}

int main( void )
{
    char array[20] = { 0 };
    cout << sizeof(array) << endl;
    Test( array );
}
```

本文来自：http://blog.vckbase.com/smileonce/archive/2004/11/05/1295.aspx

去年，周星星大哥曾经在VCKBASE/C++论坛发表过一篇文章《"数组引用"以避免"数组降阶"》*1，当时我不能深入理解这种用法的含义；时隔一年，我的知识有几经锤炼，终于对此文章渐有所悟，所以把吾所知作想详细道来，竟也成了一篇文章。希望本文能对新手有所启迪，同时也希望大家发现本文中的疏漏之处后不吝留言指教。

故事起源于周星星大哥给出的两个Demo，为了节省地方，我把两个Demo合二为一，也能说明同样的问题：

```
#include using namespace std;void Foo1(int arr[100]){ cout << "pass by pointer:   " << sizeof(arr) << endl;}void Foo2(int (&arr)[100]){ cout << "pass by reference: " << sizeof(arr) << endl;}void main(){ int a[100]; cout << "In main function : " << sizeof(a) << endl; Foo1(a); Foo2(a); }
```

其运行结果如下：

In main function : 400
pass by pointer: 4
pass by reference: 400

这段代码说明了，如果数组形参是数组名形式(或者指针形式，下文讨论)时，使用sizeof运算符，将得不到原来数组的长度；如果用传递原数组引用的方法，则没有问题。

这段代码的确很难理解，因为这短短的十几行涉及到了形参与实参的关系、数组名和指针的关系、引用的意义、声名和表达式的关系这4大类问题，只要有1条理解不透、或者理解不正确，就理解不透上面的这段代码。本文也就从这4个问题入手，把这4个问题首先解决掉，然后再探讨上面的这段代码。虽然这样看来很是繁复，但是我认为从根上入手来理解、学习，是条似远实近的道路。

一、函数形参和实参的关系

```
void Foo(int a);Foo(10);
```

这里的a叫做形式参数（parameter），简称形参；这里的10叫做实际参数（argument），简称实参。形参和式参之间是什么关系呢？他们是赋值的关系，也就是说：把实参传递给形参的过程，可以看作是把实参赋值给形参的过程。上面的例子中，实参10传递给形参a，就相当于a=10;这个赋值的过程。（因为数据类型多的很，无法举例子举全面，所以这里就不举例子了；如果觉得不好理解，就在vc中写个sample调试一下各种数据类型的情况，你就能够验证这个结论了。）

二、数组名和指针的关系

这个问题是个历史性的问题了，在C语言中，数组名是当作指针来处理的。更确切的说，数组名就是指向数组首元素地址的指针，数组索引就是距数组首元素地址的偏移量。理解这一点很重要，很多数组应用的问题就是有此而起的。这也就是为什么C语言中的数组是从0开始计数，因为这样它的索引就比较好对应到偏移量上。在C语言中，

编译过程中遇到有数组名的表达式，都会把数组名替换成指针来处理；编译器甚至无法区分a[4]和4[a]的区别！*2
但是下面这一点需要注意：

```
int a[100];int *b;
```

这两者并不等价，第一句话声明了数组a，并定义了这个数组，它有100个int型元素，sizeof(a)将得到整个数组所占的内存大小，是400；第二句话只是声明并定义了一个int型的指针，sizeof(b)将得到这个指针所占的内存大小，是4。所以说，虽然数组名在表达式中一般会当作指针来处理，但是数组名和指针还是有差距的，最起码有a==&a[0]但是sizeof(a)!=sizeof(a[0])。

并且在ANSI C标准中，也明文规定：在函数参数的声明中，数组名北边一起当作指向该数组第一个元素的指针。所以，下面的几种书写形式是等效的：

```
void Foo1(int arr[100]){}void Foo2(int arr[]){}void Foo3(int *arr){}
```

C++尽可能的全面兼容C语言，所以这一部分的语法相同。

三、引用的意义

“引用“是C++中引进的概念，C语言中没有。它的目的在于，在某些方面取代指针。如果你认为引用和指针并无大不同，肯定会为指针报不平，颇有一种“即生亮何生瑜”的感慨；但是，引用确实有新的特色，也确实在很多地方的表现和指针有所不同，本文就是一例。使用引用，我们要把握这它最最最重要的一点，这也是它和指针最大的区别：引用一经定义，就和被它引用的变量紧紧地结合在一起，再不分开，对引用的任何操作都反映在它引用的变量上；而指针，只是访问它指向变量的另一种方式，两者虽有联系，但是并不像引用那样密不可分。:)

```
#include using namespace std;
void main(){ int a = 10; int & a_ref = a; int b = 20; //int & b_ref ; // error C2530: 'b_ref' : references must be initialized
                // 定义引用时就要初始化,说明引用跟它指向的元素密不可分 int & b_ref = b; int * p; int * q;
 //下面的结果证明了：引用一经定义，就不能再指向其他目标；
 //把一个引用b_ref赋值给另一个引用a_ref，其实就是把b赋值给了a. cout << a_ref << " " << b_ref << endl; a_ref = b_ref; cout << a_ref << " " << b_ref << endl; cout << a << " " << b << endl; cout << endl;
 //即使对一个引用a_ref取地址，取得也是a的地址。已经“恶鬼附体”了:) p = &a; q = &a_ref; cout << p << " " << q << endl; cout << endl;  //下面这段代码展示了指针与引用的不同 p = &a; q = &b; cout << p << " "<< q << endl;  p = q; cout << p << " "<< q << endl; cout << endl; system("pause");}
```

下面是运行的结果，以供参考：

```
10 20
20 20
```

20 20

0012FED4 0012FED4

0012FED4 0012FEBC
0012FEBC 0012FEBC

四、声明和表达式的关系

这里想说明的是，分析一个声明可以把它看作一个表达式，按照表达式中的运算符优先级顺序来声明。比如int **(&arr)[**100**]**，你首先要找到声明器arr，那么&arr说明arr是一个引用。什么引用呢？在看括号外面，[]说明了这一个数组，100说明这个数组有100个元素，前面的int说明了这个数组的每个元素都是int型的。所以，这个声明的意思就是：arr就是指向具有100个int型元素的数组的引用。如果你觉得这种理解很晦涩，那你就不妨用typedef来简化声明中的复杂的运算符优先级关系，比如下面的形式就很好理解，其效果是和最初的那个例子是一样的：

```
#include using namespace std;typedef int INTARR[100]; //这个，这个...也可以用表达式来理解，有点“GNU is not UNIX“的味道是吧？void Foo(INTARR &arr)    //noh,这样看就很明白了，就是传了个引用进去{ cout << "pass by reference: " << sizeof(arr) << endl;}void main(){ INTARR a;            //用类型别名来定义a INTARR &a_ref=a;     //用类型别名来定义引用a_ref  cout << "In main function : " << sizeof(a) << endl; Foo(a);  system("pause");}
```

===大结局**===**

吐沫星乱飞了半天，大家感觉还好吧，快结束了，大家再忍耐一下。看看下面这段程序：

```
#include using namespace std;void main(){ int a[100]; int * pa = a; int (&a_ref)[100] = a; cout << sizeof(a) << endl; cout << sizeof(pa) << endl; cout << sizeof(a_ref) << endl; system("pause");}
```

怎么样，是不是对输出结果感到很自然呢？如果是，那就好办了。我总结一下就下课哈！^_^ 数组名在表达式中，往往被当作是指向首元素a[0]地址的指针，但是在sizeof(a)中，返回的结果是数组a占用内存的大小；pa是指向a的指针，他也指向a[0]，但是sizeof(pa)中，返回结果是pa这个指针所占内存空间的大小，之所以这样，因为pa这个指针和数组a的结合不够紧密，属于访问数组a的第二被选方案；a_ref这个引用，就是对数组a的引用，就像“恶鬼附体”一样，一旦附体附上了，你怎么也甩不掉它，对它的任何操作，全部都反映在a上。在看本文最初的那个例子，比这个例子所增加的操作就是函数实参到形参的传递，我们在上面说过了，从实参到形参的传递可以看作是把实参赋值给形参。所以本文最初的那个例子，其实际的操作过程就和本文最后的这个例子是一样的。所以，并非函数把数组给“降阶”了，而是它原原本本就该这样，千万不必奇怪。 :p

意犹未尽，在**PS**一段：在C语言中，没有引用，是怎么解决这种问题呢。下面是常用的几种作法：

1.传递数组的时候，在增加一个参数，用来记录数组的元素个数或者长度。main(int argc, char ** args)就是这种做法；这种方法还可以防止溢出，安全性比较高。
2.在数组的最后一个有效元素后面作一个标志，指明数组已经结束了。C语言中用char数组表示字符串，传给相关

的字符串函数，用的就是这种做法。这种方法保证了C的所谓字符串是无限长度的，因为用一个变量表示数组的长度的话，终归会受到这个变量类型的限制，比方说这个变量是unsigned byte型的，那么字符串长度就不能超过256，否则这个变量就溢出了。

3.对于多维数组，通常的方法是在最后一个有效维后面做一行标志，比如a[3][3]={{1,0,2},{2,2,5},{-1,-1,-1}}。如果我的程序用不到-1，我可以拿-1来填充最后一行，作为标志。这样在函数内部检测到某一维的元素都是-1，就说明到底了。

方法是灵活多变的，关键看人怎么用了。C老爹Dennis Ritchie曾经说过：C诡异离奇，缺陷重重，却获得了巨大的成功。

注1：本文将不再引用“降阶”这个术语，原因是我认为这个“降阶”的概念有种把类似2维数组压扁到1维的意思，其实本文讨论的并不是这个问题，本文讨论的是数组形参传递过程中数组长度损失的问题（这么说也不准确，还是看文中的讨论吧）。

注2：C语言的编译器遇到数组元素arr[i]，就会替换成*(arr+i)的形式。

```
int main(int argc, char* argv[])
{
int arr1[10] = {4,8,3,5,98};
for (int i=0;i<10;i++)
{
   cout<<arr1[i]<<" ";
}
cout<<endl;
for (int i=0;i<10;i++)
{
   cout<<i[arr1]<<" ";
}
cout<<endl;
return 0;
}
```

第一次知道C++竟然不能区分这样的数组元素arr1[i]和i[arr1].两个循环都可以输出数组arr1的元素。刚知道，差点被雷倒！！

1.变长一维数组

这里说的变长数组是指在编译时不能确定数组长度，程序在运行时需要动态分配内存空间的数组。实现变长数组最简单的是变长一维数组，你可以这样做：

```
//文件名:  array01.cpp
#include<iostream>
using  namespace  std;

int  main()
{
  int  len;
  cin>>len;
  //用指针p指向new动态分配的长度为len*sizeof(int)的内存空间
  int  *p=new  int[len];
  ..........
  delete[]  p;
  return  0;
}
```

注意int *p=new int[len];这一句，你不能这样做：

```
int  p[len];
```

C++编译器会报错说len的大小不能确定，因为用这种形式声明数组，数组的大小需要在编译时确定。而且这样也不行：

```
int  p[]=new  int[len];
```

编译器会说不能把int*型转化为int[]型，因为用new开辟了一段内存空间后会返回这段内存的首地址，所以要把这个地址赋给一个指针，所以要用int *p=new int[len];

array01.cpp实现了一个变长的一维数组，但是要养成一个好习惯，就是注意要注销指针p，使程序释放用new开辟的内存空间。

当然使用C++标准模版库（STL）中的vector（向量）也可以实现变长数组：

```
//文件名:  array02.cpp
#include<iostream>
#include<vector>
using  namespace  std;
```

```
int   main()
{
  int   len;
  cin>>len;
  vector<int>   array(len);//声明变长数组

  for(int   i=0;i<len;i++)
  {
    array[i]=i;
    cout<<array[i]<<"\t";
  }
  return   0;
}
```

这里的变长数组让我联想到了java的java.util包中的vector和C#中的ArrayList，它们也可以在各自的语言中实现变长数组。不过C++中的vector不能像C#一样有托管的垃圾回收机制回收被占用的内存空间，但是你可以在使用完vector后调用~vector()析构函数释放内存。

2.变长n维数组

变长的n维数组实现起来有些麻烦，但是在工程与软件设计应用中常使用的是二维数组，所以在这里着重介绍变长的二维数组，变长的n维数组可以按照类似的方法实现。首先看一个经典的用C实现变长二维数组的例子：

```
//文件名:  array03.c
#include     <stdio.h>
#include     <malloc.h>

void     main()

{
                int     x,y,i,j;
                float     **a,*b;
                                printf("请输入你所求解的线性方程组的行数x：x=");
                scanf("%d",&x);
                                printf("请输入你所求解的线性方程组的列数y：y=");
                scanf("%d",&y);

          a=(float     **)malloc(sizeof(float     *)     *x);
```

```
        b=(float     *)malloc(sizeof(float)     *x);
            for(i=0;i<x;i++)
            {
                        *(a+i)=(float     *)malloc(sizeof(float)     *y);
            }

/*读入数据*/

            printf("请按行的顺序依次输入系数的值(共%d项)：",x*y);
            for(i=0;i<=x-1;i++)
                        for(j=0;j<=y-1;j++)
                                    scanf("%f",&a[i][j]);
            printf("请按列的顺序依次输入常数的值(共%d项)：",x);
            for(j=0;j<=x-1;j++)
                                    scanf("%f",&b[j]);

            printf("您输入方程组的增广矩阵为：\n");
            for(i=0;i<=x-1;i++)
            {
                        for(j=0;j<=y-1;j++)
                                    printf("%.5f         ",a[i][j]);
                        printf("%.5f         ",b[i]);
                        printf("\n");
            }
            free(b);
            for(i=0;i<x;i++)
                        free     (*(a+i));
}
```

那么用C++怎样实现呢？在C++中可以通过new和delete运算符动态开辟和释放空间，其中new与C中malloc函数的功能相似，delete与C中free函数的功能相似。用C++实现变长二维数组时可以采用两种方法：双指针方法和使用STL中vector（向量）的方法。

首先介绍一下双指针方法，在这里双指针就是指像指针的指针，比如你可以这样声明一个数组：

```
int   **p   =   new   int*[num1];
```

而对每一个*p（一共num1个*p）申请一组内存空间：

```
for(int   i=0;   i<num1;   ++i)
```

```
  p[i]  =  new  int[num2];
```

其中，num1是行数，num2是数组的列数。测试的源程序如下：

```
//文件名:  array04.cpp
#include  <iostream>
#include  <iomanip>
using  namespace  std;

int  main()
{
  int  num1,//行数
       num2;//列数

  cout<<"Please  enter  the  number  for  row  and  column:  "<<endl;
  cin  >>  num1  >>  num2;

  //为二维数组开辟空间
  int  **p  =  new  int*[num1];
  for(int  i=0;  i<num1;  ++i)
    p[i]  =  new  int[num2];

for(int  j=0;j<num1;j++)
{
  for(int  k=0;k<num2;k++)
  {
    p[j][k]=(j+1)*(k+1);
    cout<<setw(6)<<p[j][k]<<':'<<setw(8)<<&p[j][k];
  }
  cout<<endl;
}

//释放二维数组占用的空间
for(int  m=0;m<num1;m++)
  delete[]  p[m];
delete[]  p;

return  0;
```

```
}
```

以下是运行结果：

```
Please enter the number for row and column:
4 5
        1:004915F0      2:004915F4      3:004915F8      4:004915FC      5:00491600
        2:00491180      4:00491184      6:00491188      8:0049118C      10:00491190
        3:00491140      6:00491144      9:00491148      12:0049114C     15:00491150
        4:00491100      8:00491104      12:00491108     16:0049110C     20:00491110
Press any key to continue
```

程序清单array04.cpp可以显示分配的内存空间单元的地址，大家可以看到，由于数组空间是动态分配的，数组行之间的地址空间是不连续的，因为不同行的数组元素的地址空间是用不同的new来分配的。而每一行之中列之间的地址空间是连续的。

那么用vector（向量）怎样实现二维数组呢？以下给出源程序：

```
//文件名:  array05.cpp
#include  <iostream>
#include  <vector>
#include  <iomanip>
using  namespace  std;
int  main()
{
 int  i,
      j,
      m,  //行数
      n;  //列数

 cout  <<  "input  value  for  m,n:";
 cin>>m>>n;

 //注意下面这一行：vector<int后两个">"之间要有空格！否则会被认为是重载">>"。
 vector<vector<int>  >  vecInt(m,  vector<int>(n));
 for  (i  =  0;  i  <  m;  i++)
  for  (j  =  0;  j  <  n;  j++)
```

```
      vecInt[i][j]  =  i*j;

  for  (i  =  0;  i  <  m;  i++)
  {
    for  (j  =  0;  j  <  n;  j++)
      cout<<setw(5)<<vecInt[i][j]<<":"<<setw(9)<<&vecInt[i][j];
    cout<<endl;
  }
  return  0;
}
```

以下是运行结果：

```
input  value  for  m,n:3  4
      0:  00491180      0:  00491184      0:  00491188      0:  0049118C
      0:  00491140      1:  00491144      2:  00491148      3:  0049114C
      0:  00491100      2:  00491104      4:  00491108      6:  0049110C
Press  any  key  to  continue
```

大家可以看到，这里vector中元素的内存的地址分配也有同双指针实现的二维数组有同样的特点。不过用vector的方法比使用双指针简单地多，分配内存空间时会更安全，数组初始化代码也更简单，所以本人建议使用STL中的vector来实现变长多维数组。以下是一个变长三维数组：）

```
//文件名:  array06.cpp
#include  <iostream>
#include  <vector>
#include  <iomanip>
using  namespace  std;
int  main()
{
  int  i,
    j,
    k,
    m,  //一维坐标
    n,  //二维坐标
    l;  //三维坐标
```

```
cout << "input value for m,n,l:";
cin>>m>>n>>l;
vector<vector<vector<int> > > vecInt(m, vector<vector<int> >(n, vector<i
for (i = 0; i < m; i++)
 for (j = 0; j < n; j++)
  for(k = 0; k < l; k++)
   vecInt[i][j][k] = i+j+k;

for (i = 0; i < m; i++)
{
 for (j = 0; j < n; j++)
 {
  for(k = 0; k<l; k++)
   cout<<setw(5)<<vecInt[i][j][k]<<":"<<setw(9)<<&vecInt[i][j][k];
  cout<<endl;
 }
 cout<<endl;
}

 return 0;
}
```

运行结果：

```
input value for m,n,l:2 3 4
     0: 00492FE0     1: 00492FE4     2: 00492FE8     3: 00492FEC
     1: 00492FA0     2: 00492FA4     3: 00492FA8     4: 00492FAC
     2: 00492F60     3: 00492F64     4: 00492F68     5: 00492F6C

     1: 00492EC0     2: 00492EC4     3: 00492EC8     4: 00492ECC
     2: 00492E80     3: 00492E84     4: 00492E88     5: 00492E8C
     3: 00492E40     4: 00492E44     5: 00492E48     6: 00492E4C
```

上一篇 一个VS2005无法调试C++的问题

下一篇 解决了一个隐蔽的内存泄漏——pthread_create后没有detach导致内存持续增长

猜你在找

C语言及程序设计初步
Part 1：基础语言-Cocos2d-x手机游戏开发必备C++语
C语言及程序设计提高
三维游戏引擎开发-渲染
J2SE轻松入门第二季
第四章 运算符的本质
C++实现strcmp字符串比较
毕业论文知网查重心得体会吐血奉献
OpenMP四线程同步之互斥锁函数
对引用型参数是否加const限定一定要慎重

准备好了么？跳吧 ！

更多职位尽在 CSDN JOB

iOS，Android，HTML5 程序员	我要跳槽	html5高级开发工程师	我要跳槽
上海胜因软件技术有限公司	8-10K/月	广州欧凸欧健康科技有限公司	10-20K/月
初级C/C++开发工程师	我要跳槽	HTML5游戏开发工程师	我要跳槽
东软集团华东大区	4-6K/月	微屏软件科技（上海）有限公司	4-7K/月

查看评论

1楼 proteus2 2013-05-09 08:02发表

感觉这篇文章并不深入，且很多概念错误，主要是没有去学习语言标准所致，有点知道现象，而不知道原因的样子。
关于数组和指针的关系，推荐飞天御剑流写的再再论指针，写的很好，当然他写的东西也是不断发现bug，不断修正的。对于解释数组为什么不能作为返回值，即便是修正版的（后记）论述，还是觉得不满意，当然我本人也不知道为什么。

发表评论

用 户 名： zyp2524153

评论内容：

提交