**关于C语言中的数组指针、指针数组以及二级指针**

**概念解释**

**数组指针**：首先它是一个指针，它指向一个数组，即指向数组的指针；在32 位系统下永远是占4 个字节，至于它指向的数组占多少字节，不知道。数组指针指向的是数组中的一个具体元素，而不是整个数组，所以数组指针的类型和数组元素的类型有关。   
**指针数组**：首先它是一个数组，数组的元素都是指针，数组占多少个字节由数组本身决定。它是“储存指针的数组”的简称，即每个元素都是指针。   
**二级指针**： 如果一个指针指向的是另外一个指针，我们就称它为二级指针，或者指向指针的指针。

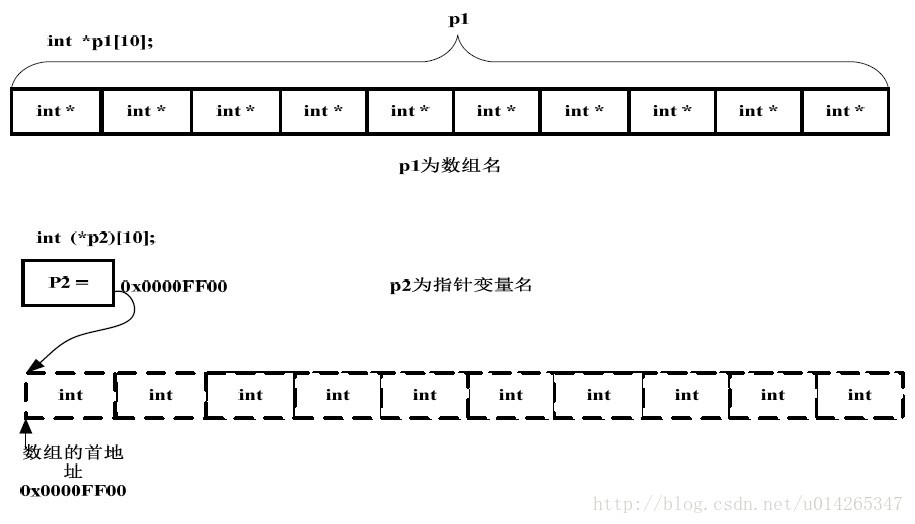
**实例解释**

**判断哪个为指针数组哪个为数组指针?**

int \*p1[10];

int (\*p2)[10];

**解析**

1. “[]”的优先级比“*”要高。p1 先与“[]”结合，构成一个数组的定义，数组名为p1，int*修饰的是数组的内容，即数组的每个元素.因此这是一个数组，其包含10 个指向int 类型数据的指针，即指针数组
2. “()”的优先级比“[]”高，“\*”号和p2 构成一个指针的定义，指针变量名为p2，int 修饰的是数组的内容，即数组的每个元素。数组在这里并没有名字，是个匿名数组。因此p2 是一个指针，它指向一个包含10 个int 类型数据的数组，即数组指针   
   

**关于p2的定义问题**

平时我们定义指针不都是在数据类型后面加上指针变量名么？这个指针p2 的定义怎么不是按照这个语法来定义的呢？也许我们应该这样来定义p2：   
int (\*)[10] p2;   
int (\*)[10]是指针类型，p2 是指针变量。这样看起来的确不错，不过就是样子有些别扭。其实数组指针的原型确实就是这样子的，只不过为了方便与好看把指针变量p2 前移了而已。

**利用指针遍历数组元素**

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

int arr[] = { 1, 3, 5, 7, 9};

int len = sizeof(arr) / sizeof(int); //求数组长度

int i;

for(i=0; i<len; i++)

{

printf("%d ", \*(arr+i) ); //\*(arr+i)等价于arr[i]

}

printf("\n");

return 0;

}

1. *(arr+i)这个表达式，arr 是数组名，指向数组的第 0 个元素，表示数组首地址， arr+i 指向数组的第 i 个元素，*(arr+i) 表示取第 i 个元素的数据，它等价于 arr[i]。其中arr 是int\*类型的指针，每次加 1 时它自身的值会增加 sizeof(int)，加 i 时自身的值会增加 sizeof(int) \* i
2. 还可以如此表示

int arr[] = { 1, 3, 5, 7, 9};

int \*p = arr;

arr 是数组第 0 个元素的地址，所以int \*p = arr;也可以写作int \*p = &arr[0];。也就是说，arr、p、&arr[0] 这三种写法都是等价的，它们都指向数组第 0 个元素，或者说指向数组的开头。

**利用数组指针遍历数组**

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

int arr[] = { 1, 3, 5, 7, 9};

int len = sizeof(arr) / sizeof(int); //求数组长度

int i, \*p = arr;

for(i=0; i<len; i++)

{

printf("%d ", \*(p+i) );

}

printf("\n");

return 0;

}

1. 数组在内存中只是数组元素的简单排列，没有开始和结束标志，在求数组的长度时不能使用sizeof(p) / sizeof(int)，因为 p 只是一个指向 int 类型的指针，编译器并不知道它指向的到底是一个整数还是一系列整数（数组），所以 sizeof(p) 求得的是 p 这个指针变量本身所占用的字节数，而不是整个数组占用的字节数。
2. 根据数组指针不能逆推出整个数组元素的个数，以及数组从哪里开始、到哪里结束等信息。不像字符串，数组本身也没有特定的结束标志，如果不知道数组的长度，那么就无法遍历整个数组。
3. 对指针变量进行加法和减法运算时，是根据数据类型的长度来计算的。如果一个指针变量 p 指向了数组的开头，那么 p+i 就指向数组的第 i 个元素；如果 p 指向了数组的第 n 个元素，那么 p+i 就是指向第 n+i 个元素；而不管 p 指向了数组的第几个元素，p+1 总是指向下一个元素，p-1 也总是指向上一个元素

更改上面的代码，让 p 指向数组中的第二个元素：

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

int arr[] = { 1, 3, 5, 7, 9};

int \*p = &arr[2]; //也可以写作 int \*p = arr + 2;

printf("%d, %d, %d, %d, %d\n", \*(p-2), \*(p-1), \*p, \*(p+1), \*(p+2) );

return 0;

}

会发现结果和上面的一致

**总结**

引入数组指针后，我们就有两种方案来访问数组元素了，一种是使用下标，另外一种是使用指针。   
**1. 使用下标**   
也就是采用 arr[i] 的形式访问数组元素。如果 p 是指向数组 arr 的指针，那么也可以使用 p[i] 来访问数组元素，它等价于 arr[i]。   
**2. 使用指针**   
也就是使用 *(p+i) 的形式访问数组元素。另外数组名本身也是指针，也可以使用*(arr+i) 来访问数组元素，它等价于 \*(p+i)。   
不管是数组名还是数组指针，都可以使用上面的两种方式来访问数组元素。不同的是，数组名是常量，它的值不能改变，而数组指针是变量（除非特别指明它是常量），它的值可以任意改变。也就是说，数组名只能指向数组的开头，而数组指针可以先指向数组开头，再指向其他元素。

**借助自增运算符来遍历数组元素**

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

int arr[] = { 1, 3, 5, 7, 9};

int i, \*p = arr, len = sizeof(arr) / sizeof(int);

for(i=0; i<len; i++)

{

printf("%d ", \*p++ );

}

printf("\n");

return 0;

}

**解释**

*p++ 应该理解为*(p++)，每次循环都会改变 p 的值（p++ 使得 p 自身的值增加），以使 p 指向下一个数组元素。该语句不能写为 \*arr++，因为 arr 是常量，而 arr++ 会改变它的值，这显然是错误的

**关于数组指针的几个问题**

**假设 p 是指向数组 arr 中第 n 个元素的指针，那么 *p++、*++p、(\*p)++ 分别是什么意思呢？**   
1. \*p++上面已经叙述   
2. *++p 等价于*(++p)，会先进行 ++p 运算，使得 p 的值增加，指向下一个元素，整体上相当于 \*(p+1)，所以会获得第 n+1 个数组元素的值   
3. (\*p)++ 会先取得第 n 个元素的值，再对该元素的值加 1。假设 p 指向第 0 个元素，并且第 0 个元素的值为 1，执行完该语句后，第 0 个元素的值就会变为 2

**实例中的指针数组和二级指针**

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

int a = 1, b = 2, c = 3;

//定义一个指针数组

int \*arr[3] = {&a, &b, &c};//也可以不指定长度，直接写作 int \*parr[]

//定义一个指向指针数组的指针,即二级指针

int \*\*parr = arr;

printf("%d, %d, %d\n", \*arr[0], \*arr[1], \*arr[2]);

printf("%d, %d, %d\n", \*\*(parr+0), \*\*(parr+1), \*\*(parr+2));

return 0;

}

1. arr 是一个**指针数组**，它包含了 3 个元素，每个元素都是一个指针，在定义 arr 的同时，我们使用变量 a、b、c 的地址对它进行了初始化，这和普通数组很类似。
2. parr 是指向数组 arr 的指针，确切地说是指向 arr 第 0 个元素的指针，它的定义形式应该理解为int *(\*parr)，括号中的 \** 表示 parr 是一个指针，括号外面的int *表示 parr 指向的数据的类型。arr 第 0 个元素的类型为 int*，所以在定义 parr 时要加两个 \*,即可称parr为**二级指针**，或者**指向指针的指针**。

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

char \*lines[5] =

{

"COSC1283/1984",

"Programming",

"Techniques",

"is",

"great fun"

};

char \*str1 = lines[1];

char \*str2 = \*(lines + 3);

char c1 = \*(\*(lines + 4) + 6);

char c2 = (\*lines + 5)[5];

char c3 = \*lines[0] + 2;

printf("str1 = %s\n", str1);

printf("str2 = %s\n", str2);

printf(" c1 = %c\n", c1);

printf(" c2 = %c\n", c2);

printf(" c3 = %c\n", c3);

return 0;

}

运行结果

str1 = Programming

str2 = is

c1 = f

c2 = 9

c3 = E

为了更加直观，将上述代码改成下面的形式

#include <stdio.h>

#include<iostream>

using namespace std;

int main()

{

char \*string0 = "COSC1283/1984";

char \*string1 = "Programming";

char \*string2 = "Techniques";

char \*string3 = "is";

char \*string4 = "great fun";

char \*lines[5];

lines[0] = string0;

lines[1] = string1;

lines[2] = string2;

lines[3] = string3;

lines[4] = string4;

char \*str1 = lines[1];

char \*str2 = \*(lines + 3);

char c1 = \*(\*(lines + 4) + 6);

char c2 = (\*lines + 5)[5];

char c3 = \*lines[0] + 2;

printf("str1 = %s\n", str1);

printf("str2 = %s\n", str2);

printf(" c1 = %c\n", c1);

printf(" c2 = %c\n", c2);

printf(" c3 = %c\n", c3);

return 0;

}

1. char \*lines[5]; 定义了一个指针数组，数组的每一个元素都是指向char类型的指针。最后5行，为数组的每一个元素赋值，都是直接赋给指针。

2. 而lines，是一个指向指针的指针，它的类型为 char \*\*，所以 \*lines 是一个指向字符的指针，\*\*lines是一个具体的字符。这一点很重要，一定要明白。

3. 指针是可以进行运算的，lines 为lines[5]数组的首地址，即第0个元素的地址；lines+0, lines+1, lines+2 ... 分别是第0, 1, 2 ...个元素的首地址，\*(lines+0)或lines[0], \*(lines+1)或lines[1], \*(lines+2)或lines[2] ... 分别是字符串 str0, str1, str2 ... 的首地址。所以：

\*lines == \*(lines+0) == lines[0] == str0

\*(lines+1) == lines[1] == str1

\*(lines+2) == lines[2] == str2

**注意**   
lines为指向指针的指针，所以\* (lines+n)为指针，\*\*(lines+n)才为具体的字符。

**解析**

1. lines[1]：它是一个指针，指向字符串string1，即string1的首地址。

2. \*(lines + 3)：lines + 3 为lines[5]数组第3个元素的地址， \*(lines + 3)为第3个元素，它是一个指针，指向字符串string3。

3. \*(\*(lines + 4) + 6)：\*(lines + 4) + 6 == lines[4] + 6 == string4 + 6，为字符串string4第6个字符的地址，即 f 的地址，\*(\*(lines + 4) + 6) 就表示字符 f。

4. (\*lines + 5)[5]：\*lines + 5 为字符串 string0 第5个字符的地址，即 2 的地址，(\*lines + 5)[5]等价于\*(\*lines + 5 + 5)，表示第10个字符，即9。

5. \*lines[0] + 2：\*lines[0] 为字符串string0 第0个字符的地址，即C的地址。字符与整数运算，首先转换为该字符对应的ASCII码值，然后再运算，所以 \*lines[0] + 2 = 67 + 2 = 69。不过要求输出字符，所以还要转换成69所对应的字符，即E。

**输入5个国名并按字母顺序排列后输出**

#include<stdio.h>

#include<iostream>

using namespace std;

void sort(char \*name[],int n)

{

char \*pt;

int i,j,k;

for(i=0;i<n-1;i++)

{

k=i;

for(j=i+1;j<n;j++)

if(strcmp(name[k],name[j])>0)

k=j;

if(k!=i)

{

pt=name[i];

name[i]=name[k];

name[k]=pt;

}

}

}

void print(char \*name[],int n)

{

int i;

for (i=0;i<n;i++)

printf("%s\n",name[i]);

}

int main()

{

static char \*name[]={ "CHINA","AMERICA","AUSTRALIA","FRANCE","GERMAN"};

int n=5;

sort(name,n);

print(name,n);

return 0;

}

**说明：**   
1. 在以前的例子中采用了普通的排序方法，逐个比较之后交换字符串的位置。交换字符串的物理位置是通过字符串复制函数完成的。反复的交换将使程序执行的速度很慢，同时由于各字符串(国名)的长度不同，又增加了存储管理的负担。用指针数组能很好地解决这些问题。把所有的字符串存放在一个数组中，把这些字符数组的首地址放在一个指针数组中，当需要交换两个字符串时，只须交换指针数组相应两元素的内容(地址)即可，而不必交换字符串本身。   
2. 本程序定义了两个函数，一个名为sort完成排序，其形参为指针数组name，即为待排序的各字符串数组的指针。形参n为字符串的个数。另一个函数名为print，用于排序后字符串的输出，其形参与sort的形参相同。主函数main中，定义了指针数组name 并作了初始化赋值。然后分别调用sort函数和print函数完成排序和输出。值得说明的是在sort函数中，对两个字符串比较，采用了strcmp函数，strcmp函数允许参与比较的字符串以指针方式出现。name[k]和name[j]均为指针，因此是合法的。字符串比较后需要交换时，只交换指针数组元素的值，而不交换具体的字符串，这样将大大减少时间的开销，提高了运行效率。   
3. 这题用algorithm中的sort()也可以很好的解决。