**一. 解读复杂指针声明**

要理解复杂类型其实很简单,一个类型里会出现很多运算符,他们也像普通的表达式一样,有优先级,其优先级和运算优先级一样.  
记住原则:**从变量名处起,根据运算符优先级结合,一步一步分析.**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6180640)

1. **int** p;
2. /\* 这是一个普通的整型变量 \*/
3. **int** \*p;
4. /\* 首先从P 处开始,先与\*结合,所以说明P 是一个指针,
5. 然后再与int 结合,说明指针所指向的内容的类型为int 型.
6. 所以P 是一个返回整型数据的指针
7. \*/
8. **int** p[3];
9. /\* 首先从P 处开始,先与[]结合,说明P 是一个数组,
10. 然后与int 结合,说明数组里的元素是整型的.
11. 所以P 是一个由整型数据组成的数组
12. \*/
13. **int** \*p[3];
14. /\* 首先从P 处开始,先与[]结合,因为其优先级比\*高,所以P 是一个数组,
15. 然后再与\*结合,说明数组里的元素是指针类型,
16. 然后再与int 结合,说明指针所指向的内容的类型是整型的
17. 所以P 是一个由返回整型数据的指针所组成的数组
18. \*/
19. **int** (\*p)[3];
20. /\* 首先从P 处开始,先与\*结合,说明P 是一个指针
21. 然后再与[]结合,说明指针所指向的内容是一个数组
22. 然后再与int 结合,说明数组里的元素是整型的.
23. 所以P 是一个指向由整型数据组成的数组的指针
24. \*/
25. // 想想与 int a[3]是不是一样啊，a也可以看成一个指向整型数据组成的数组的指针
26. **int** \*\*p;
27. /\* 首先从P 开始,先与\*结合,说是P 是一个指针,
28. 然后再与\*结合,说明指针所指向的元素是指针,
29. 然后再与int 结合,说明该指针所指向的元素是整型数据.
30. \*/
31. **int** p(**int**);
32. /\* 从P 处起,先与()结合,说明P 是一个函数,
33. 然后进入()里分析,说明该函数有一个整型变量的参数
34. 然后再与外面的int 结合,说明函数的返回值是一个整型数据
35. \*/
36. **int** (\*p)(**int**);
37. /\* 从P 处开始,先与指针结合,说明P 是一个指针,
38. 然后与()结合,说明指针指向的是一个函数,
39. 然后再与()里的int 结合,说明函数有一个int 型的参数,
40. 再与最外层的int 结合,说明函数的返回类型是整型,
41. 所以P 是一个指向有一个整型参数且返回类型为整型的函数的指针
42. \*/
43. // 函数指针是也！
44. **int** \*(\*p(**int**))[3];
45. /\* 从P 开始,先与()结合,说明P 是一个函数,
46. 然后进入()里面,与int 结合,说明函数有一个整型变量参数,
47. 然后再与外面的\*结合,说明函数返回的是一个指针,
48. 然后到最外面一层,先与[]结合,说明返回的指针指向的是一个数组,
49. 然后再与\*结合,说明数组里的元素是指针,
50. 然后再与int 结合,说明指针指向的内容是整型数据.
51. 所以P 是一个参数为一个整数据且返回一个指向由整型指针变量组成的数组的指针变量的函数.
52. \*/
53. **int** (\*p[5])(**int**\*);
54. /\* 从p开始，先与[]结合，所以p是一个数组，
55. 然后与\*结合，说明数组里的元素是指针类型，
56. 然后与()结合，说明指针指向是一个函数，
57. 这个函数有一个指针参数，返回值int类型。
58. 所以p 是一个返回整形数据的函数指针数组。
59. \*/

int p;

/\* 这是一个普通的整型变量 \*/

int \*p;

/\* 首先从P 处开始,先与\*结合,所以说明P 是一个指针,

然后再与int 结合,说明指针所指向的内容的类型为int 型.

所以P 是一个返回整型数据的指针

\*/

int p[3];

/\* 首先从P 处开始,先与[]结合,说明P 是一个数组,

然后与int 结合,说明数组里的元素是整型的.

所以P 是一个由整型数据组成的数组

\*/

int \*p[3];

/\* 首先从P 处开始,先与[]结合,因为其优先级比\*高,所以P 是一个数组,

然后再与\*结合,说明数组里的元素是指针类型,

然后再与int 结合,说明指针所指向的内容的类型是整型的

所以P 是一个由返回整型数据的指针所组成的数组

\*/

int (\*p)[3];

/\* 首先从P 处开始,先与\*结合,说明P 是一个指针

然后再与[]结合,说明指针所指向的内容是一个数组

然后再与int 结合,说明数组里的元素是整型的.

所以P 是一个指向由整型数据组成的数组的指针

\*/

// 想想与 int a[3]是不是一样啊，a也可以看成一个指向整型数据组成的数组的指针

int \*\*p;

/\* 首先从P 开始,先与\*结合,说是P 是一个指针,

然后再与\*结合,说明指针所指向的元素是指针,

然后再与int 结合,说明该指针所指向的元素是整型数据.

\*/

int p(int);

/\* 从P 处起,先与()结合,说明P 是一个函数,

然后进入()里分析,说明该函数有一个整型变量的参数

然后再与外面的int 结合,说明函数的返回值是一个整型数据

\*/

int (\*p)(int);

/\* 从P 处开始,先与指针结合,说明P 是一个指针,

然后与()结合,说明指针指向的是一个函数,

然后再与()里的int 结合,说明函数有一个int 型的参数,

再与最外层的int 结合,说明函数的返回类型是整型,

所以P 是一个指向有一个整型参数且返回类型为整型的函数的指针

\*/

// 函数指针是也！

int \*(\*p(int))[3];

/\* 从P 开始,先与()结合,说明P 是一个函数,

然后进入()里面,与int 结合,说明函数有一个整型变量参数,

然后再与外面的\*结合,说明函数返回的是一个指针,

然后到最外面一层,先与[]结合,说明返回的指针指向的是一个数组,

然后再与\*结合,说明数组里的元素是指针,

然后再与int 结合,说明指针指向的内容是整型数据.

所以P 是一个参数为一个整数据且返回一个指向由整型指针变量组成的数组的指针变量的函数.

\*/

int (\*p[5])(int\*);

/\* 从p开始，先与[]结合，所以p是一个数组，

然后与\*结合，说明数组里的元素是指针类型，

然后与()结合，说明指针指向是一个函数，

这个函数有一个指针参数，返回值int类型。

所以p 是一个返回整形数据的函数指针数组。

\*/

**二. 指针的算术运算**

**1. 指针可以加上或减去一个整数。**指针的这种运算是**以单元为单位**。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6180640)

1. **char** a[20]="You\_are\_a\_girl";
2. **int** \*ptr=(**int** \*)a;
3. ptr+=5;

char a[20]="You\_are\_a\_girl";

int \*ptr=(int \*)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。

**2. 指针和指针进行加减：  
两个指针不能进行加法运算**，这是非法操作，因为进行加法后，得到的结果指向一个不知所向的地方，而且毫无意义。  
**两个指针可以进行减法操作**，但必须类型相同，一般用在数组方面。

**三. 数组首地址 a, &a, &a[0]**

int a[10]

**1. a 作为右值时，代表数组首元素的首地址，而非数组首地址。**

也就是 a[0] 的地址。int i = \*(a+1)，这里a是右值，所是代表首元素的首地址，a+1代表下一个元素的首地址，即&a[1]。

**2. a 是整个数组的名子。**所以 sizeof(a) 的值为 size(int) \* 10 = 40，代表整个数组的大小。**3. & a 即为取 a 的首地址，也即是整个数组的首地址。**所以 sizeof(&a) 的值为 4。

但是在VC6.0里显示的是40，这是不对的。code:block里显示的是4。  
int \* p = (int\*)(&a+1) 中 &a+1 代表下一个数组的首地址，显然是越界的。

**4. & a[0] 代表首元素的首地址。**所以 sizeof(&a[0]) 的值为 4。

**5. &a[10] 很显然，数组越界了，但是他的sizeof是多少呢？**也是4，因为关键字sizeof求值是在编译的时候，虽然并不存在a[10]这个元素，但是这里并没有真正访问a[10]，而是根

据数组元素的类型来确定其值。所以sizeof(a[10])不会出错。

**6. a[-1]代表什么意思？**

首先要明白下标的形式被编译器解析成指针的形式，即a[1] 解析成 \*(a+1)，那么a[-1]就被解析成\*(a-1)。

**四. 指针数组 & 数组指针**

**指针数组**：首先它是一个数组，数组的元素都是指针，也称为"储存指针的数组"。  
**数组指针**：首先它是一个指针，它指向一个数组。也可以理解为"数组的指针。"

int \*p1[10]  
int (\*p2)[10] 各代表什么？要弄清这个问题，首先要知道**[ ]优先级比\*要高**。

p1先与[ ]结合，构成一个数组定义，数组名为p1，int\*修饰的是数组的内容，即数组的每个元素。

p2中()的优先级比[ ]高，所以\*号先与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。数组在这里没有名子。

**五. 函数指针 & 指针函数**

**函数指针：**指向函数的指针变量。

**指针函数：**带指针的函数，也就是返回指针的函数。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6180640)

1. **char** \* fun(**char**\* a, **char**\* b) //定义为 指针函数
2. {...... }
3. **int** main()
4. {
5. **char**\* (\*p)(**char**\* p1, **char**\* p2); //定义为 函数指针
6. p = &fun; //把函数地址赋给他
7. //p = fun; //这样写也行
8. (\*p)("aa", "bb"); //使用函数指针
9. return 0;
10. }

char \* fun(char\* a, char\* b) //定义为 指针函数

{...... }

int main()

{

char\* (\*p)(char\* p1, char\* p2); //定义为 函数指针

p = &fun; //把函数地址赋给他

//p = fun; //这样写也行

(\*p)("aa", "bb"); //使用函数指针

return 0;

}

**指针参数：**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6180640)

1. **int** fun(**char** \*s)
2. {
3. **int** num=0;
4. num+=\*s;
5. s++;
6. return num;
7. }
8. **int** b;
9. **char** str[]="abcdefghijklmn";
10. b=fun(str);

int fun(char \*s)

{

int num=0;

num+=\*s;

s++;

return num;

}

int b;

char str[]="abcdefghijklmn";

b=fun(str);

在函数调用中，当把str作为实参传递给形参s 后，实际是把str 的值传递给了s，  
s 所指向的地址就和str 所指向的地址一致，但是str 和s 各自占用各自的存储空间。  
在函数体内对s 进行自加1 运算，并不意味着同时对str 进行了自加1 运算。

**六. 指针常量 & 常量指针**

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/lwbeyond/article/details/6180640)

1. const **char**\* p1; //常量指针，指向常量的指针
2. **char** const\* p2;
3. **char**\* const p3; //指针常量，指针是常量

const char\* p1; //常量指针，指向常量的指针

char const\* p2;

char\* const p3; //指针常量，指针是常量

怎么记？

a. 可以先把类型名去掉，然后看 const 离谁近，就修饰谁。

b. 也可以const 在 \* 左边的为常量指针，const 在 \* 右边的为指针常量。

**七. 野指针**

野指针是怎么造成的？

1. 指针变量被创建而没有初始化。

2. 指针 p 被 free 或者 delete 之后， 没有置为 NULL。