char 型指针：

从本质上来看，字符指针其实就是一个指针而已， 只不过该指针用来指向一个字符串/字符串数组。

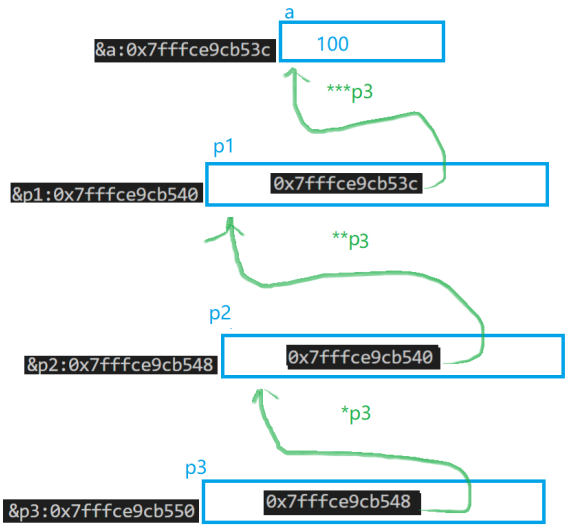
```
char * msg = "Hello Even" ;
```

多级指针：

- 如果一个指针p1 它所指向的是一个普通变量的地址，那么p1就是一个一级指针
- 如果一个指针p2 它所指向的是一个指针变量的地址，那么p2就是一个二级指针
- 如果一个指针p3 它所指向的是一个指向二级指针变量的地址，那么p3就是一个三级指针

```
int a = 100 ;
int * p1 = &a ; // 那么p1就是一个一级指针
int ** p2 = &p1 ; // 那么p2就是一个二级指针
int *** p3 = &p2 ; // 那么p3就是一个三级指针
```

```
01 C语言基础 > 09 指针 > C ptr_demo.c > main(int, char const * [])
 9
10      printf("a:%d\n" , a );
11      printf("*p1:%d\n" , *p1 );
12      printf("**p2:%d\n" , **p2 );
13      printf("***p3:%d\n" , ***p3 );
14
15      printf("&a:%p\n" , &a );
16      printf("&p1:%p --> p1:%p\n" , &p1 , p1 );
17      printf("&p2:%p --> p2:%p\n" , &p2 , p2 );
18      printf("&p3:%p --> p3:%p\n" , &p3 , p3 );

问题  输出  调试控制台  终端                1: wsl

a:100
*p1:100
**p2:100
***p3:100
&a:0x7fffce9cb53c
&p1:0x7fffce9cb540 --> p1:0x7fffce9cb53c
&p2:0x7fffce9cb548 --> p2:0x7fffce9cb540
&p3:0x7fffce9cb550 --> p3:0x7fffce9cb548
```



指针的万能拆解方法：

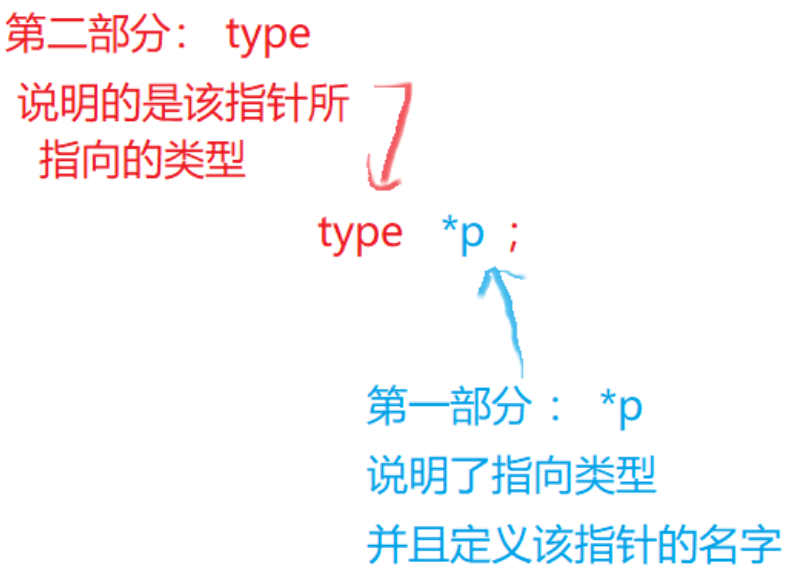
对于任何的指针都可以分为两部分：

第一部分： 说明他是一个指针 （*p）

第二部分： 说用它所指向的内容的类型 （*p）以外的东西

```
char * p1 ; // 第一部分： * p1 ， 第二部分 char 说明p1 指向的类型为char
char **p2 ; // 第一部分： * p2 , 第二部分 char * 说明p2 指向的类型为char *
```

```
int **p3 ; // 第一部分： * p3 ，第二部分 int * 说明p3 指向的类型为int *
char (*p4) [3] ; // 第一部分： * p4 ，第二部分 char [3] ，说明p4 指向一个拥有3个元素的char 数组
char （*p5） (int , float) ; // 第一部分： * p5, 第二部分char (int , float) ， 说明
 // 说明该指针指向一个 拥有char类型返回， 并需要 一个int 和 float 参数的函数
void *(*p6) (void *); //第一部分： * p6, 第二部分 void * (void *)
 // 说明p6 指向一个 拥有 void * 返回并需要一个void * 参数的函数 函数指针）
```



总结：

- 以上指针 p1 p2 p3 p4 p5 p6 本质上都是指针，因此它们的大小都是 8字节（64位系统）
- 以上指针 p1 p2 p3 p4 p5 p6 本质上都是指针， 唯一的不容是它们所指向的内容的类型不同

void 型指针：

概念： 表示该指针的类型暂时是不确定

要点：

- void 类型的指针，是没有办法直接索引目标的。必须先进行强制类型转换。
- void 类型指针，无法直接进行加减运算。

void关键字的作用：

- 修饰指针，表示该指针指向了一个未知类型的数据。
- 修饰函数的参数列表， 则表示该函数不需要参数。
- 修饰函数的返回值，则表示该函数没有返回值。

```c
void * p =  malloc(4) ; // 使用malloc 来申请 4个字节的内存， 并让p来指向该内存的入口地址

*(int *)p = 250 ; // 先使用(int*) 来强调p是一个整型地址 ， 然后再解引用
printf("*p:%d\n", *(int*)p);// 输出时也应该使用对应的类型来进行输出

*(float*)p = 3.14 ;
printf("*p:%f\n", *(float*)p);

int * a ;
 char * b ;
 float * f ;

 void * k ;

```

注意：

以上写法 void * p， 在实际开发中不应该出现。以上代码只是为了说明语法问题。

const 指针：

- const修饰指针有两种效果：
  - 常指针 修饰的是指针本身， 表示该指针变量无法修改、

```c
char * const p ;
```

```c
char arr [] = "Hello" ;
char msg [] = "Even" ;
char * const p = arr ;

// p = msg ; // p 被const 所修改，
        //说明P是一个常量 ， 他的内容（所指向的地址）无法修改
*(p + 1 ) = 'E' ; // p所指向的内容是可以通过p 来修改 （只要保持P所指向的地址不变即可）
printf("%s\n" , p );
```

  - 常目标指针 修饰的是指针所指向的目标，表示无法通过该指针来改变目标的数据

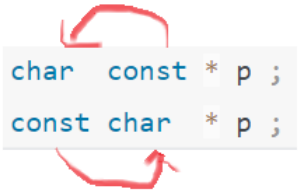
```c
char const * p ;
const char * p ;
```

```c
char arr [] = "Hello" ;
char msg [] = "Even" ;

const char * p1 = arr ;
p1 = msg ;  // p1 的指向是可以被修改的
// *(p1+1) = 'V' ; // 常目标指针， 不允许通过该指针来它所指向的内容

*(msg+1) = 'V' ; // 虽然p1不能修改所指向的内容， 但是内容本身是可以被修改的

printf("%s\n" , p1 );
```

总结：

- 常指针并不常见。
- 常目标指针，在实际开发过程中比较常见，用来限制指针的权限为 只读

p是常量--> 常指针



p是一个常目标指针
修饰的是指针所指向的目标