关于strcpy函数形参类型的解析和指针作为输入型输出型参数的不同

在C语言中，字符串一直都是热点，关于strcpy函数大家都很熟悉，但是真正了解的很少，一旦用到总会报一大堆莫名其妙错误，今天我就来给大家详细剖析一下strcpy函数。

虽然不能看到strcpy的内部实现，但是我们通过查阅<string.h>可以看到strcpy函数的声明。

char * __cdecl strcpy(char *, const char *);

那个_cdecl是一个函数调用约定，暂且不讨论，我们今天就来说一下strcpy指针形参加const与不加的区别，帮助大家更好使用这个函数

首先我们要理解这两种语句有何不同

```
1 char *p="abcd";
2 char str[5]="abcd";
```

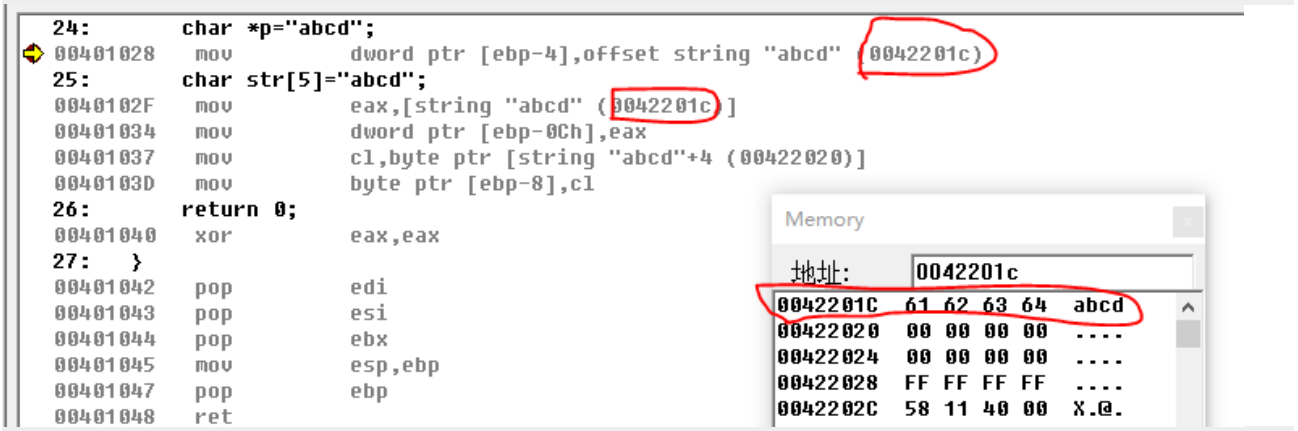
这两条语句都是存储abcd字符串，但是经过编译链接后，会产生不同的结果

语句1，常量字符串会保存在程序的常量区，编译时会将该字符串在常量区的起始地址拷贝过来存在指针p中，

语句2 常量字符串也会保存在程序的常量区，但是在字符数组str初始化时i，会将字符串拷贝到str中，即数组中存储字符串副本

即str[0]='a';str[1]='b';str[2]='c';str[3]='d';str[4]='\0';

如图所示



我们看到在地址0x0042201C 处存储的是字符串常量abcd，而语句1汇编指令mov dword ptr ds:[ebp-4], 0xoo42201C，作用就是把常量字符串地址存到[ebp-4]这个内存空间（即变量名为p的内存地址）

语句2则是将该常量值一个个拷贝到数组中，即字符串存储在数据段中

那么**这样区分之后，会产生一个是否允许修改的差异，我们都知道常量区中的内容不允许修改，而数据区的内容是可读可写的**，因此，如果我们这样写

```
1 p[0]='w';
2 str[0]='q';
```

会发现 虽然语句1编译链接都通过，但是运行时会报错，这是因为**语句1试图非法修改常量区的值，而常量区是允许修改，只能读取**

而语句2则可以使程序正常运行，因为str数组只是常量字符串的一份副本，这份副本存在数据区，可以修改，而且不会影响到常量区字符串的值

明白了关于**指向字符串常量的指针和存储字符串常量的数组之间的差异**后，我们接下来讨论指针形参输出型和输入型问题

对于形参是指针类型的，我们都知道是传址调用，也明白函数内形参的改变会影响到形参，这就涉及一个实参是否允许修改的问题

char * strcpy(char * strDest,const char * strSrc);

这个函数形参，一个是不加const修饰，一个是加上了const修饰，有何区别呢？这个函数是将strSrc指向的字符串复制到strDest中，（连同字符串结束符'\0'一起复制），

那么就是说，strDest指向的值是可以修改的，而strSRC指向的值在函数中是不允许修改的，我们把加上const修饰的参数称为输入型参数，即只允许读取，不允许写入，把不加const修饰的参数称为输出型参数，即可以在函数内部进行读写改变，从而在主调函数中看到改变。

通过刚才的探讨，我们可以很容易知道如下四条语句哪些会使程序运行时出错

```
char *p="1234";
char *q="abcd";
char str1[5]="6789";
char str2[5]="hijk";
strcpy(p,q);①
strcpy(p,str1);②
strcpy(str1,p);③
strcpy(str1,str2);④
```

很明显，标号①②的语句都会运行时出错，因为strcpy的第一个形参要求是可以写入的，而p，q都是指向了常量区字符串的首地址，不可写入

标号③④都是可以正常运行，但是推荐写法③，因为写法④设计一个隐式转换问题，将str2转换成了常指针了。

下面给出一个strcpy函数的实现

```
char * strcpy(char *strDest,const char *strSrc)
{
assert((strDest!=NULL)&&(strSrc!=NULL)); //断言两个指针都不是空指针
char *address=strDest;                                    //函数要返回复制后的字符串首地址
while((*(strDest++)=*(strSrc++))!='\0');//连同结束符一起复制
return address;                                                  //返回复制后的字符串首地址
}
```

总结

1 char *p="1234";指针p指向常量区，不允许修改内容及p[0]='a'非法

2 char str[5]="abcd" 字符数组str复制了常量区字符串“abcd”的值，而字符数组是在数据段，可以进行修改及str[0]='1'合法

3 对于strcpy函数的第一个形参，是输出型形参，因此只能是数组名，而不能是字符指针变量

4 对strcpy的第二个形参，是输入型形参，可以是数组名或者是字符指针变量，但最好是字符指针变量

strcpy()函数详解

strcpy()函数是C语言中的一个复制字符串的库函数，以下将详细解释说明一下：

- **函数声明以及实现代码**

```
char *strcpy(char *dst, const char *src);
```

```
1. char * strcpy(char *dst,const char *src)
2. {
3.     if((dst==NULL)||(src==NULL))
4.     
5.         return NULL;
6. 
7.     char *ret = dst; //[1]
8. 
9.     while ((*dst++=*src++)!='\0'); //[2]
10. 
11.     return ret;//[3]
12. }
```

（1）const 修饰：源字符串参数用const修饰，防止修改源字符串；

（2）空指针检查：源指针和目的指针都有可能会出现空指针的情况，所以应该对其进行检查；

（3）为什么要设置ret 指针以及返回ret指针的位置[3]，由于目的指针dst已经在进行移动了，所以用辅助指针ret表明首指针；

（4）以上所示[2]处，为简单的字符串的复制过程，正好表明strcpy函数遇到'\0'将会停止；

- **示例说明**

```
char a[7]="abcdef";
char b[4]="ABC";
strcpy(a,b);
printf ("%c",a[5]);

a这个数组是什么：是"ABC\0ef"
还是"ABC\0"
或者其他答案呢?
```

之前在某处看到过这个问题，以下将对其进行详解，

· 最终答案输出是 "ABC\0"；

· 为什么答案不是"ABC\0ef"；

[1]从strcpy函数的实现代码可以看出当src指针指向为 ‘\0’ 时将会停止字符串的复制，由此可以得知返回ret指针所指向的数组a内容应该是 “ABC\0ef”,也就是说实际内存数组a中的内容应该是“ABC\0ef “；

但是为什么最终显示会是"ABC\0"呢，原因在于，strcpy的本身属性：

即strcpy只用于字符串复制，并且它不仅复制字符串内容之外，还会复制字符串的结束符；

基于此种原因，但是字符串的特性是什么呢？字符串最后一个字节存放的是一个空字符——“\0”，用来表示字符串的结束。把b复制到a之后会令b的空字符把复制后的字符串隔断,所以最终printf输出只能是"ABC\0"；

【注】此函数的第一个属性，会返回ret,也就是第一次，dst赋给ret的首地址，如

```
1. char *a="coda";
2.     char b[MAX]="you are the best one.";
3.     char *p;
4.     p=strcpy(b+8,a);
5.     puts(p);
```

输出结果为：coda

可能到这里你已经发现了一些问题，如果想把一个字符串的一部分复制到另一个字符串的某个位置，该怎么办呢，显然strcpy()函数是满足不了这个功能的，strncpy()函数是为了弥补strcpy()函数不能检查目标字符串是否容纳下源字符串的不足而设定的一个函数。并且完全可以实现这个功能。

以上如果有错误，还请提出。