# My\_strcpy

已知strcpy函数的原型是：

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

1.不调用库函数，实现strcpy函数。

2.解释为什么要返回char \*。

参考答案：

1）不调用C++/C的字符串库函数，请编写函数 strcpy  
char \*strcpy(char \*strDest, const char \*strSrc)  
{

    assert((strDest!=NULL) && (strSrc !=NULL));    // 2分

    char \*address = strDest;                                          // 2分  
    while( (\*strDest++ = \* strSrc++) != '\0' )         // 2分  
              NULL ;  
    return address ;                                                  // 2分  
}

（2）strcpy能把strSrc的内容复制到strDest，为什么还要char \* 类型的返回值？  
答：为了实现链式表达式。                                              // 2分  
例如       int length = strlen( strcpy( strDest, “hello world”) );

错误的做法：  
[1](A)不检查指针的有效性，说明答题者不注重代码的健壮性。  
    (B)检查指针的有效性时使用((!strDest)||(!strSrc))或(!(strDest&& strSrc))，说明答题者对**[C语言](http://lib.csdn.net/base/c" \o "C语言知识库" \t "_blank)**中类型的隐式转换没有深刻认识。在本例中char \*转换为bool即是类型隐式转换，这种功能虽然灵活，但更多的是导致出错概率增大和维护成本升高。所以C++专门增加了bool、true、false 三个关键字以提供更安全的条件表达式。  
    (C)检查指针的有效性时使用((strDest==0)||(strSrc==0))，说明答题者不知道使用常量的好处。直接使用字面常量（如本例中的 0）会减少程序的可维护性。0虽然简单，但程序中可能出现很多处对指针的检查，万一出现笔误，编译器不能发现，生成的程序内含逻辑错误，很难排除。而使用 NULL代替0，如果出现拼写错误，编译器就会检查出来。  
[2](A)return new string("Invalid argument(s)");，说明答题者根本不知道返回值的用途，并且他对内存泄漏也没有警惕心。从函数中返回函数体内分配的内存是十分危险的做法，他把释放内存的义务抛给不知情的调用者，绝大多数情况下，调用者不会释放内存，这导致内存泄漏。  
    (B)return 0;，说明答题者没有掌握异常机制。调用者有可能忘记检查返回值，调用者还可能无法检查返回值（见后面的链式表达式）。妄想让返回值肩负返回正确值和异常值的双重功能，其结果往往是两种功能都失效。应该以抛出异常来代替返回值，这样可以减轻调用者的负担、使错误不会被忽略、增强程序的可维护性。  
[4](A)循环写成while (\*strDest++=\*strSrc++);，同[1](B)。  
    (B)循环写成while (\*strSrc!='\0') \*strDest++=\*strSrc++;，说明答题者对边界条件的检查不力。循环体结束后，strDest字符串的末尾没有正确地加上'\0'。

 对于第一个错误，可能很多人会写成这样

if((strDest==NULL) || (strSrc ==NULL))   return NULL;    // 2分

其实这样也是不太妥的，因为这之后程序会继续运行，而用断言assert就会中断程序。

关于assert的用法，下一节总结。

assert() 函数用法  
assert宏的原型定义在<assert.h>中，其作用是如果它的条件返回错误，则终止程序执行，原型定义：  
#include <assert.h>  
void assert( int expression );  
assert的作用是现计算表达式 expression ，如果其值为假（即为0），那么它先向stderr打印一条出错信息，  
然后通过调用 abort 来终止程序运行。

另外，对于strcpy()函数看看**[Linux](http://lib.csdn.net/base/linux" \o "Linux知识库" \t "_blank)**标准中时如何写的，如下：

[**linux**](http://lib.csdn.net/base/linux)下的标准strcpy函数的写法：（tring.c在/usr/src/linux-.../lib/string.c下面）  
/\*\*  
 \* strcpy - Copy a %NUL terminated string  
 \* @dest: Where to copy the string to  
 \* @src: Where to copy the string from  
 \*/  
char \*strcpy(char \*dest, const char \*src)  
{  
char \*tmp = dest;  
while ((\*dest++ = \*src++) != '\0')  
/\* nothing \*/;  
return tmp;  
}

附:linux下string.c中一些函数的源码：

**[cpp]** [view plain](http://blog.csdn.net/abilly/article/details/24481269) [copy](http://blog.csdn.net/abilly/article/details/24481269)

1. /\*\*
2. \* strncpy - Copy a length-limited, %NUL-terminated string
3. \* @dest: Where to copy the string to
4. \* @src: Where to copy the string from
5. \* @count: The maximum number of bytes to copy
6. \*
7. \* The result is not %NUL-terminated if the source exceeds
8. \* @count bytes.
9. \*
10. \* In the case where the length of @src is less than  that  of
11. \* count, the remainder of @dest will be padded with %NUL.
12. \*
13. \*/
14. **char** \*strncpy(**char** \*dest, **const** **char** \*src, **size\_t** count)
15. {
16. **char** \*tmp = dest;

19. **while** (count) {
20. **if** ((\*tmp = \*src) != 0)
21. src++;
22. tmp++;
23. count--;
24. }
25. **return** dest;
26. }
27. /\*\*
28. \* strcmp - Compare two strings
29. \* @cs: One string
30. \* @ct: Another string
31. \*/
32. **int** strcmp(**const** **char** \*cs, **const** **char** \*ct)
33. {
34. unsigned **char** c1, c2;

37. **while** (1) {
38. c1 = \*cs++;
39. c2 = \*ct++;
40. **if** (c1 != c2)
41. **return** c1 < c2 ? -1 : 1;
42. **if** (!c1)
43. **break**;
44. }
45. **return** 0;
46. }

49. /\*\*
50. \* strlen - Find the length of a string
51. \* @s: The string to be sized
52. \*/
53. **size\_t** strlen(**const** **char** \*s)
54. {
55. **const** **char** \*sc;

58. **for** (sc = s; \*sc != '\0'; ++sc)
59. /\* nothing \*/;
60. **return** sc - s;
61. }



66. /\*\*
67. \* memset - Fill a region of memory with the given value
68. \* @s: Pointer to the start of the area.
69. \* @c: The byte to fill the area with
70. \* @count: The size of the area.
71. \*
72. \* Do not use memset() to access IO space, use memset\_io() instead.
73. \*/
74. **void** \*memset(**void** \*s, **int** c, **size\_t** count)
75. {
76. **char** \*xs = s;

79. **while** (count--)
80. \*xs++ = c;
81. **return** s;
82. }

85. /\*\*
86. \* memcpy - Copy one area of memory to another
87. \* @dest: Where to copy to
88. \* @src: Where to copy from
89. \* @count: The size of the area.
90. \*
91. \* You should not use this function to access IO space, use memcpy\_toio()
92. \* or memcpy\_fromio() instead.
93. \*/
94. **void** \*memcpy(**void** \*dest, **const** **void** \*src, **size\_t** count)
95. {
96. **char** \*tmp = dest;
97. **const** **char** \*s = src;

100. **while** (count--)
101. \*tmp++ = \*s++;
102. **return** dest;
103. }