联想计算机丛书之一

# TURBO C

## 实用子程序集

李春葆　译　王廷俊　校


北京联想计算机集团公司

1991·3

# 《Turbo C 实用子程序集》

李春葆　翻译
王廷俊　审校

- 数百个通用精巧的子程序
- 管道和 I/O 换向设计
- 文件操作
- DOS. BIOS 系统调用
- ANSI 支持
- I/O 处理
- 存贮器映象
- 菜单设计

# 目　　录

# 第一章　概述

编写本书的目的旨在节省你开发、编程和调试 Turbo C 程序的时间，你可以使用书中提供的库函数来完成你的 Turbo C 程序，这些子程序可以简单地插入到你的 Turbo C 程序中，而且它们都调试过，因此也减少了你程序开发的调试时间。如果你对 Turbo C、DOS 或 IBM PC 还是新手，这些子程序将会帮助你掌握这些主题。书中所列出的源码，你可以根据可读性、易修改性和通用性来作适当的修改。

当你通过这些子程序获得提高后，记住它们仅是基本的 Tubro C 库函数，你还可以开发更多的库函数，也可以修改这些子程度以满足你的实际需要，有了这些库函数，你的程序开发变得更容易了。

· Turbo C 标准库函数

库函数的主要功能是减少代码重复。如果别人编好了一个执行特定功能的代码，你为什么要重复编码呢？Borland 公司提供了功能强大的子程序集，称之为标准库函数，你要尽可能地使用它们，Borland 公司雇用许多 Turbo C 专家编写这些标准库函数，其中所有子程序编制得既好又优，你应花时间熟悉它们。本书附录 B 列出了全部标准库函数。

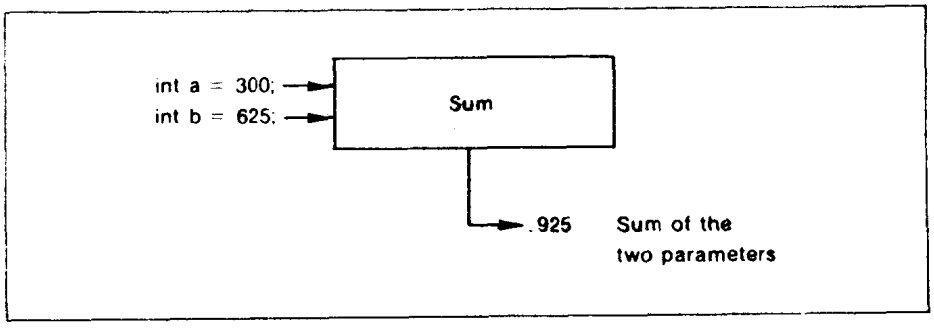
你可能认为本书的子程序与标准库函数功能重复，在很多情况下，这可以简单回答成是为了源程序代码。在字符串方面，Turbo C 标准库函数提供了功能强大的字符串操作子程序。由于字符串使用十分广泛，要充分理解字符串操作的唯一方式是要有标准库函数的源代码，如果没有它们，你就不能修改它们以满足需要，本书中列出的子程序，可以帮助你更好地理解 Turbo C 标准库函数。

如果你要进行严格的 Turbo C 开发，就要考虑从 Borland 公司购买标准库函数的源代码，它提供了开发 Turbo C 程序的标准例子。

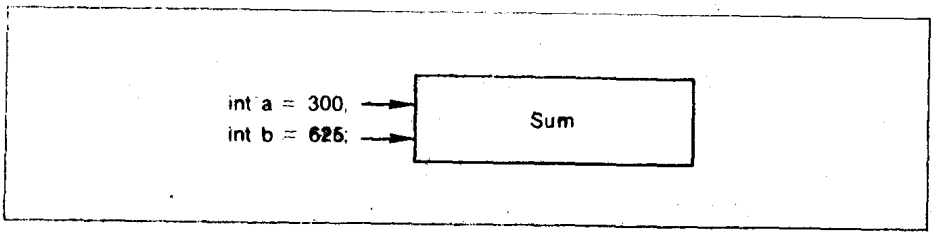
· 子程序的表示

本书中所有子程序采用相同的描述格式。

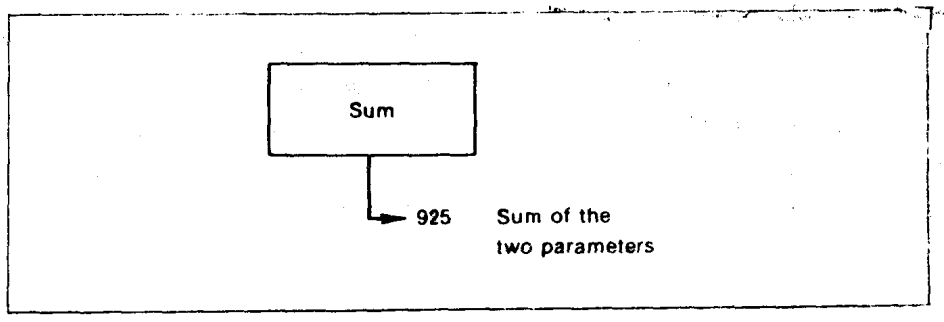
首先用图解描述，然后给出源程序代码，例如子程序 Sum 接收两个数并返回它们的和，其图解描述如下。



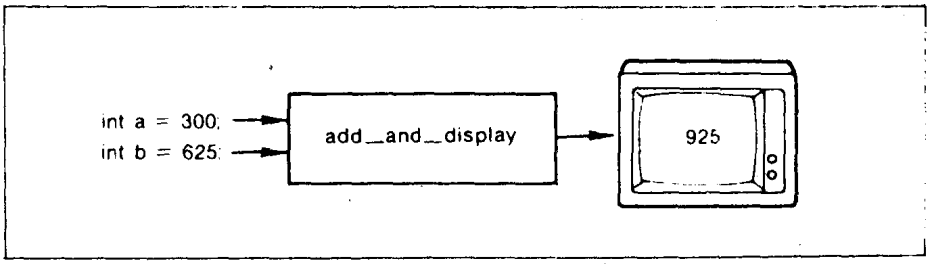
首先看到有两个变量传递到该子程序：




该描述告诉你这两个变量的数据类型为整型，并给出了你可以赋的值，在这里，值 300 和 625 要进行加法运算。

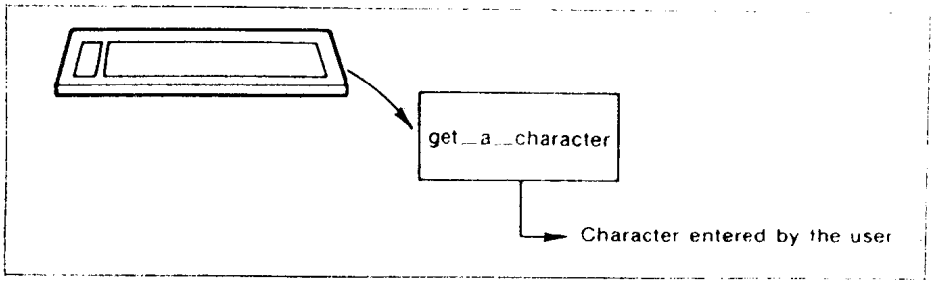



返回一个值的所有子程序都指明所返回的值来自顶部的方盒，子程序 add_and_display 不仅返回相加的结果，而且把结果显示在屏幕上，如下图：

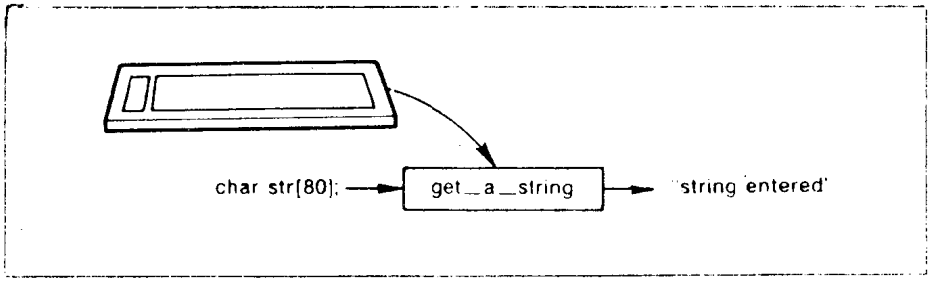



任何把数据写到屏幕上的子程序都以这种格式描述。

类似地，如果子程序 get_a_character 使用了键盘，则采用如下描述：




如果一个子程序修改了一个或多个参数，那么修改的参数放在方盒的右边，如下图：




图解描述子程序的目的是在列出这些源码之前勾画出它的轮郭，很多情况下，你不必知道一个子程序如何做，只需知道它做什么，这种描述就是帮助你达到这个目的。

描述每个子程序源码的格式也是一样的，前面的子程序 Sum 的源码描述如下：

```
/*
 * sum (a, b)
 *
 * Return the sum of the two integer values specified.
 *
 * a (in): First value to sum.
 *
 * b (in): Second value to sum
 *
 * result = sum (6, 7);
 *
 */

sum (a, b)
  int a, b;

  return (a + b);
```

注意功能代码之前的头部描述：

```
/*
 * sum (a, b)
 *
 * Return the sum of the two integer values specified.
 *
 * a (in): First value to sum.
 * b (in): Second value to sum.
 *
 * result = sum (6, 7);
 *
 */
```

通过头部信息，在读源码之前了解该子程序的功能，变量和用途，描述块中每个参数标有 in 或者 out，在一个函数中其值始终不变的参数是 in 参数，其初始值不被函数使用的参数称为 out 参数，如果参数的初值被函数使用且对它作了修改，这样的参数为 in/out 参数。

• 使用函数原型

如果你从未使用过 Turbo C 原型，你必须改正这种习惯。Turbo C 允许你定义和宣称一个函数。当你定义一个函数时，你提供它的源码，如：

```
float sum (a, b, c)
  float a, b, c;
 {
  return (a + b + c);
 }
```

当你宣称一个函数时，你把有关该函数的信息告诉其它函数，如：

```
float sum ();
```

多年来，很多 C 程序员仅当函数返回一个非整型值时才宣称它，其实，通过函数宣称，你可以避免很多运行错误，因为这些错误可能是由编译器引起的。看如下例子：

```
float sum (a, b, c)
  float a, b, c;
 {
  return (a + b + c);
 }

main ()
 {
  float sum ();

  printf ("%f\n", sum (1.2, 2.4));
 }
```

这里，函数 sum 要有三个参数，但你只给了两个参数，因为这时编译器没有关于 sum 的知识，便接受了这个程序代码，很难觉察要出现的运行错误。

使用原型就避免了这个错误，注意你如何改变 sum 的函数头部，按如下方式把参数定义移进圆括号里：

```
float sum (float a, float b, float c)
 {
  return (a + b + c);
 }
```

在主程序中，你必须按如下方式把 sum 宣称为一个函数并指明每个参数的类型：

```
float sum (float a, float b, float c)
 {
  return (a + b + c);
 }

main ()
 {
  float sum (float, float, float);

  printf ("%f\n", sum (1.2, 2.4));
 }
```

因为主程序有了 sum 的知识，它就发现调用：

```
  printf ("%f\n", sum (1.2, 2.4));
```

是错误的。

在开发本书的子程序中，函数原型节省了作者大量的时间，你也应该宣称所有使用的子程序及其参数，这也会节省你调试时间。

·汇编语言子程序

第 14、15 章描述的几个子程序是在两个提供硬件接口的汇编语言子程序基础上开发的，为了使用这些子程序，你必须具备 Microsoft 宏汇编或目标代码库磁盘，任何程序开发要尽可能多地使用高级语言，这个要求在本书里是符合的，但为了执行快速屏幕显示，这两个接口子程序必须要用汇编语言编写。

·结语

有了编程技巧、经验和成百个子程序，这会大大减少你开发程序的时间，利用这些时间，你便可以好好研究书中给出的源程序代码了。

# 第二章　字符串操作

任何C程序库中使用最广泛的子程序是字符串操作子程序。许多C编译器提供了一个通用的字符串操作函数库，Turbo C也不例外，由于大量使用字符串，所以你必须充分了解C是如何存贮和操作字符串的。

本章详细讨论字符串，并描述了很多解决问题的工具，学完本章后，你应该能够识别一个程序比另一个程序更优是由哪些要素决定的。这章提供了一个完整的字符串操作子程序集合，来帮助你了解一些Turbo C标准库函数是怎样做的，从而在规范化基础上开发你的每个程序。

·引言

字符串是一个或多个字符的序列，C语言象存贮数组一样存贮字符串，字符串的字符存贮在一片相邻存贮器区域中。例如，采用如下方式宣称的字符串：

```
char some_string [255];
```

C用255个字符的存贮空间产生一个字符串变量，和所有C数组一样，C字符串对应一个指标(index)，其开始编移值为0，前面的宣称产生一个如图2－1所示指标的数组。

C没有内部方法用来确定一个C字符串的字符数目，标准方法是把一个空字符(ASCII0)放在字符串最后一个字符之后。因此，C存贮字符串"Turbo C"如图2－2所示，末尾增加了一个空字符。

由于每个字符串都用一个空字符标志结束，所以，你可以采用如下简单方式搜索字符(\0)来确定字符串S的字符数目：

for (i=0;s[i]! ='\0';i++)

每当你在双引号里指定一个字符串时，Turbo C便把空字符放在它的末尾。例如：

```
#define COMPILER "Turbo C"
```

在大多数情况下，保证把一个空字符放在字符串末尾，成为程序员的责任，如同下面的程序：

```
main ()
 {
   char alphabet [27];    /* 26 letters and space for null */
   char letter;
   int index;
   for (index = 0, letter = 'A'; letter <= 'Z'; index++, letter++)
      alphabet [index] = letter;
   alphabet [index] = '\0';        * append the null character */
   printf ("%s\n", alphabet);
 }
```

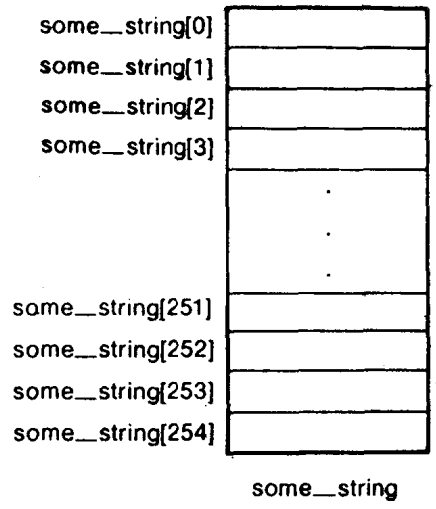



图 2—1 字符串数组的指标

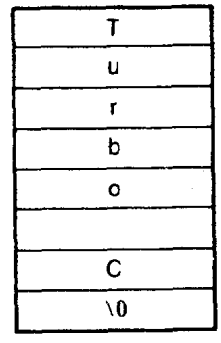



图 2—2 把一个空字符放进字符串中

• 作为参数的字符串

帮助你编写通用字符串操作子程序的重要因素之一是可以在 C 中按如下方式处理传递给函数的数组、假设你有一个称为 string_length 函数，它返回字符串的字符数目，在你的程序中采用如下方式调用它：

```
count = string_length (strvar);
```

既然传递的是字符数组，你不妨在该函数内部宣称一个无界字符串：

```
string_length (char str[])
 {
  /* code here */
 }
```

本章后面的所有子程序都采用这种方式宣称字符串参数。

• 字符串长度

如下的子程序通过判定后继字符是否为空字符来计算字符串的长度：

```
/*
 * string_length (string)
 *
 * Return the number of characters in the string.
 *
 * string (in): string to return the length of.
 *
 * count = string_length (string);
 *
 */

int string_length (char string[])
 {
  int i;

  for (i = 0; string[i]; i++)
    ;

  return (i);
 }
```

• 数组界限

在大多数程序设计应用中，程序运行所花时间与所占空间总是矛盾的，字符串操作子程序也不例外。

如下子程序，把一个字符串的内容拷贝到另一个字符串中去。

```
/*
 * void first_copy (source, target)
 *
 * Copy the contents of the source string to the target.
 *
 * s1 (in): source string containing characters to copy.
 * s2 (out): string receiving characters copied.
 *
 * first_copy ("This is a test", stringvar);
 *
 * first_copy does not perform bounds checking.
 *
 */

void first_copy (char s1[], char s2[])
 {
  int i;

  for (i = 0; s1[i] != '\0'; ++i)
    s2[i] = s1[i];

  s2[i] = '\0';
 }
```

该子程序把字符从第一个字符串(S1)拷贝到第二个字符串(S2),每次拷贝一个字符,直到找到空字符为止(如图 2—3 所示)。

该子程序在大多数情况下运行是正确的,但是,看如下例子,它使用了 first_copy:

```
main ()
 {
   s2[5];

   first_copy ("long string", s2);
 }
```

这里,first_copy 看起来象是从 S1 到 S2 拷贝字符,实际上,拷贝已经超过了 S2 的数组界限,字符串"long string"包含 11 个字符(包括空字符),而 S2 仅有 5 个字符的空间,结果,first_copy 覆盖了子程序栈空间的内容,从而产生错误,为了纠正这个问题,你可以增加一个参数用来指定可以赋给目标字符串的最大字符个数。如:

```
second_copy ("long string", s, sizeof(s));
```

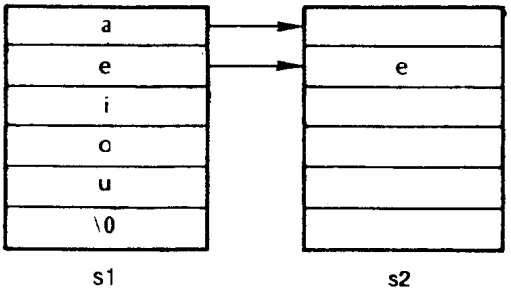



图 2—3 把字符从第一个字符串拷贝到第二个字符串中

```
/*
 * int second_copy (source, target, array_bound)
 *
 * Copy the source string to the target string variable.
 *
 * s1 (in): Contains the characters to be copied.
 * s2 (out): Receives the characters copied.
 * maxchar (in): specifies the maximum number of characters
 *               that s2 can store.
 *
 * status = second_copy ("This is", stringvar, sizeof (stringvar));
 *
 * If the array bounds are exceeded, second_copy returns the value
 * 1; otherwise it returns the value 0.
 *
 */

int second_copy (char s1[], char s2[], int maxchar)
 {
  int index;

  maxchar--;              /* leave space for null */

  for (index = 0; (s1[index] != '\0') && index < maxchar; index++)
    s2[index] = s1[index];

  s2[index] = '\0';

  return (s1[index] && (index == maxchar));
 }
```

该子程序避免发生前面的错误，但由于包含了两个判定：

```
(s1[index] != '\0') && (index < maxchar)
```

从而增加了每次迭代的处理时间。

·最小源代码

虽然前面的子程序可读性很强(假设你熟悉 C 数组)，你也可以简化这些代码。

看如下代码段：

```
for (i = 0; s1[i] != '\0'; ++i)
  s2[i] = s1[i];
s2[i] = '\0';
```

C 允许你把它改变成：

```
for (i = 0; (s2[i] = s1[i]) != '\0'; ++i)
  ;
```

这两种情况，执行相同的功能，在第二个代码段中，Turbo C 采用循环迭代判定赋给 S2 的值，如果该值为空，终止循环，否则，把 S1 的下一个字符赋给 S2，再重复判定，一旦把空字符赋给 S2，终止循环，由于在循环中包含把空字符赋给 S2，所以你可以删除代码行：

```
s2[i] = '\0';
```

当把空字符赋给 S2 时，从判定语句：

```
(s2[i] = s1[i]) != '\0'
```

返回的值为 0(空是 ASCII0)，由于 C 中假(false)与 0 等价，你可以再一次修改这个代码：

```
for (i = 0; (s2[i] = s1[i]) ; ++i)
  ;
```

在你开发 C 字符串操作子程序时，考虑如下几点：

1)、执行速度比可靠性更重要吗？

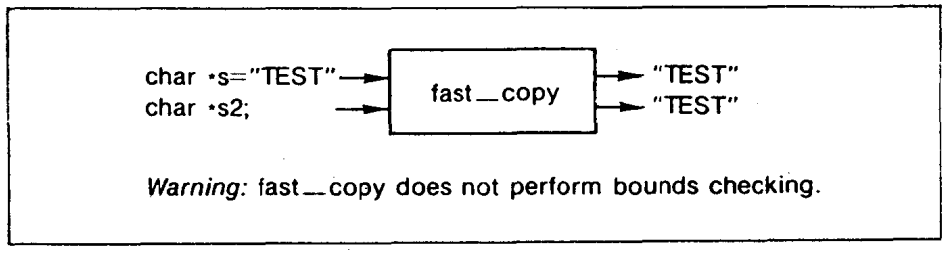
2)、代码要尽可能简单吗？

3)、要保持代码的可读性吗？

开发字符串操作子程序库时，要不断地在速度、可靠性和可读性之间作平衡，究竟突出哪个特性，这与你程序设计的需求有关。

·字符串拷贝(copy)

有两个执行字符串拷贝的子程序，第一个是 fast_copy，它不作界限检查，把第一个字符串的内容直接拷贝到第二个字符串中，第二个是 copy_string，也作同样的处理，但包含界限检查。



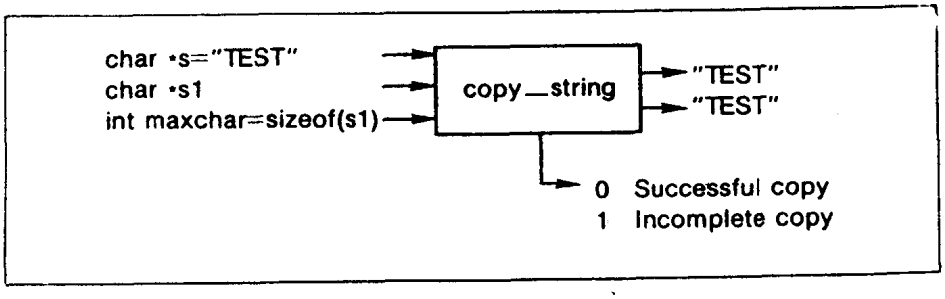

```
/*
 * void fast_copy (source, target)
 *
 * Copy the contents of the source string to the target.
 *
 * s1 (in): source string containing characters to copy.

 * s2 (out): string receiving characters copied.
 *
 * fast_copy ("This is a test", stringvar);
 *
 * fast_copy does not perform bounds checking.
 *
 */

void fast_copy (char s1[], char s2[])
 {
  int i;

  for (i = 0; (s2[i] = s1[i]) ; ++i)
     ;
 }
```

如果该子程序不能执行成功的拷贝，则返回值 1，否则返回值 0。




```
/*
 * int copy_string (source, target, array_bound)
 *
 * Copy the source string to the target string variable.
 *
 * s1 (in): contains the characters to be copied.
 * s2 (out): receives the characters copied.
 * maxchar (in): specifies the maximum number of characters
 *               that s2 can store.
 *
 * status = copy_string ("This is", stringvar, sizeof (stringvar));
 *
 * If the array bounds are exceeded, string_copy returns the value
 * 1; otherwise it returns the value 0.
 *
 */

int copy_string (char s1[], char s2[], int maxchar)
 {
  int i;

  maxchar--;            /* leave space for null */
```

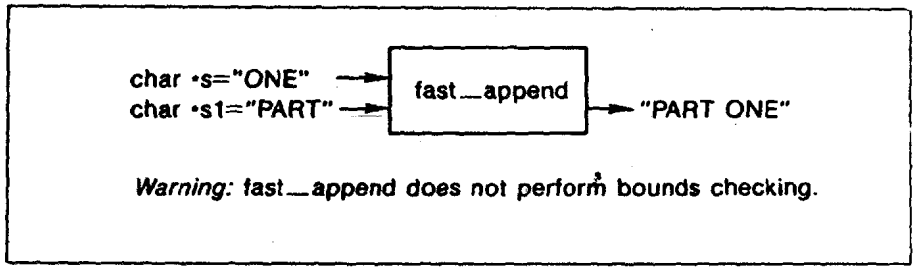
```
for (i = 0; (s2[i] = s1[i]) && i < maxchar; i++)
   ;

if (i == maxchar && s1[i])   /* see if characters remain in s1 */
  {
   s2[i] = '\0';

   return (1);
  }
 else
   return (0);
}
```

• 字符串连接(APPEND)

下面的两个子程序是把第一个字符串的内容连接到第二个字符串的后面，它们先找到第二字符串的末尾位置(空字符)，然后把第一个字符串的字符连接到后面，一旦第一个字符串的空字符连接到第二个字符串后面，循环终止，和前面一样，fast_append 不作界限检查，子程序 append_string 作界限检查。




```
/*
 * void fast_append (source, target)
 *
 * Append the contents of the source string to the target.
 *
 * s1 (in): source string containing characters to append.
 * s2 (out): string receiving characters copied.
 *
 * fast_append ("This is a test", stringvar);
 *
 * fast_append does not perform bounds checking.
 *
 */

void fast_append (char s1[], char s2[])
{
 int i, j;

 for (i = 0; s2[i] ; ++i)   /* find the end of s2 */
    ;

 for (j = 0; s2[i] = s1[j]; i++, j++)  /* append s1 */
    ;

 }
```

如果如下子程序不能成功地进行字符串连接，则返回值 1，否则返回值 0。