C 语言字符串操作函数

1.函数名: strcpy

功　能: 拷贝一个字符串到另一个字符串

用　法: char *strcpy(char *destin, char *source);

程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
int main(void) {
      char string[10];
 char *str1 = "abcdefghi";
      stpcpy(string, str1);
 printf("%s\n", string);
 return 0;
 }
```

2.函数名: strcat

功　能: 字符串拼接函数

用　法: char *strcat(char *destin, char *source);

程序例:

```
＃i nclude <string.h>
＃i nclude <stdio.h>
int main(void) {
 char destination[25];
 char *blank = " ",*c = "C++", *Borland = "Borland";
    strcpy(destination, Borland);
    strcat(destination, blank);
    strcat(destination, c);
    printf("%s\n", destination);
    return 0;
    }
```

3.函数名: strchr

功　能: 在一个串中查找给定字符的第一个匹配之处\

用　法: char *strchr(char *str, char c);

程序例:

```
＃i nclude <string.h>  ＃i nclude <stdio.h>
int main(void)   {
 char string[15];
 char *ptr, c = 'r';
      strcpy(string, "This is a string");
 ptr = strchr(string, c);
 if (ptr)
 printf("The character %c is at position: %d\n", c, ptr-string);
 else
```

```
 printf("The character was not found\n");
 return 0;
 }
```

4.函数名: strcmp
功　能: 串比较
用　法: int strcmp(char *str1, char *str2);
看 Asic 码，str1>str2，返回值 > 0；两串相等，返回 0 程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int main(void)  {
 char *buf1 = "aaa", *buf2 = "bbb", *buf3 = "ccc";
 int ptr;
     ptr = strcmp(buf2, buf1);
 if (ptr > 0)
  printf("buffer 2 is greater than buffer 1\n");
 else
  printf("buffer 2 is less than buffer 1\n");
     ptr = strcmp(buf2, buf3);
 if (ptr > 0)
  printf("buffer 2 is greater than buffer 3\n");
 else
  printf("buffer 2 is less than buffer 3\n");
     return 0;
 }
```

5.函数名: strncmpi
功　能: 将一个串中的一部分与另一个串比较, 不管大小写
用　法: int strncmpi(char *str1, char *str2, unsigned maxlen);
程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int main(void)
{
    char *buf1 = "BBB", *buf2 = "bbb";
    int ptr;
    ptr = strcmpi(buf2, buf1);
    if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");
    return 0;
```

```
    }
```

6.函数名: strcpy
功　能: 串拷贝
用　法: char *strcpy(char *str1, char *str2);
程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
int main(void)
{
  char string[10];
  char *str1 = "abcdefghi";
       strcpy(string, str1);
  printf("%s\n", string);
  return 0;
}
```

7.函数名: strcspn
功　能: 在串中查找第一个给定字符集内容的段
用　法: int strcspn(char *str1, char *str2);
程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
＃i nclude <alloc.h>
int main(void)    {
  char *string1 = "1234567890";
  char *string2 = "747DC8";
  int length;
       length = strcspn(string1, string2);
  printf("Character where strings intersect is at position %d\n", length);
       return 0;
  }
```

8.函数名: strdup
功　能: 将串拷贝到新建的位置处
用　法: char *strdup(char *str);
程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
＃i nclude <alloc.h>
int main(void)
{
  char *dup_str, *string = "abcde";
       dup_str = strdup(string);
```

```
  printf("%s\n", dup_str);
  free(dup_str);
        return 0;
}
```

9.函数名: stricmp
功　能: 以大小写不敏感方式比较两个串
用　法: int stricmp(char *str1, char *str2);
程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int main(void)
{
     char *buf1 = "BBB", *buf2 = "bbb";
     int ptr;
     ptr = stricmp(buf2, buf1);
     if (ptr > 0)
       printf("buffer 2 is greater than buffer 1\n");
     if (ptr < 0)
       printf("buffer 2 is less than buffer 1\n");
     if (ptr == 0)
       printf("buffer 2 equals buffer 1\n");
     return 0;
}
```

10.函数名: strerror
功　能: 返回指向错误信息字符串的指针
用　法: char *strerror(int errnum);
程序例:

```
#i nclude <stdio.h>
#i nclude <errno.h>
int main(void)
{
  char *buffer;
  buffer = strerror(errno);
  printf("Error: %s\n", buffer);
  return 0;
}
```

11.函数名: strcmpi
功　能: 将一个串与另一个比较, 不管大小写
用　法: int strcmpi(char *str1, char *str2);
程序例:

```
#i nclude <string.h>
```

```
#i nclude <stdio.h>
int main(void)
{
    char *buf1 = "BBB", *buf2 = "bbb";
    int ptr;
    ptr = strcmpi(buf2, buf1);
    if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");
    return 0; }
```

12.函数名: strncmp
功　能: 串比较
用　法: int strncmp(char *str1, char *str2, int maxlen);
程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int    main(void)
{
    char *buf1 = "aaabbb", *buf2 = "bbbccc", *buf3 = "ccc";
    int ptr;
    ptr = strncmp(buf2,buf1,3);
    if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
    else
      printf("buffer 2 is less than buffer 1\n");
    ptr = strncmp(buf2,buf3,3);

    if (ptr > 0)
      printf("buffer 2 is greater than buffer 3\n");
    else
      printf("buffer 2 is less than buffer 3\n");

    return(0);
    }
```

13.函数名: strncmpi
功　能: 把串中的一部分与另一串中的一部分比较, 不管大小写
用　法: int strncmpi(char *str1, char *str2);
程序例:

```
#i nclude <string.h>
```

```
#i nclude <stdio.h>
int main(void)
{
    char *buf1 = "BBBccc", *buf2 = "bbbccc";
    int ptr;
    ptr = strncmpi(buf2,buf1,3);
    if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
    if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");
    return 0;
    }
```

14.函数名: strncpy
功　能: 串拷贝
用　法: char *strncpy(char *destin, char *source, int maxlen);
程序例:

```
#i nclude <stdio.h>
#i nclude <string.h>
int main(void)
{
    char string[10];
    char *str1 = "abcdefghi";
    strncpy(string, str1, 3);
    string[3] = '\0';
    printf("%s\n", string);
    return 0;
}
```

15.函数名: strnicmp
功　能: 不注重大小写地比较两个串
用　法: int strnicmp(char *str1, char *str2, unsigned maxlen);
程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int main(void)
{
    char *buf1 = "BBBccc", *buf2 = "bbbccc";
    int ptr;
    ptr = strnicmp(buf2, buf1, 3);
    if (ptr > 0)
      printf("buffer 2 is greater than buffer 1\n");
```

```
    if (ptr < 0)
      printf("buffer 2 is less than buffer 1\n");
    if (ptr == 0)
      printf("buffer 2 equals buffer 1\n");
    return 0;
}
```

16.函数名: strnset
功　能: 将一个串中的所有字符都设为指定字符
用　法: char *strnset(char *str, char ch, unsigned n);
程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
int main(void)
{
    char *string = "abcdefghijklmnopqrstuvwxyz";
    char letter = 'x';
    printf("string before strnset: %s\n", string);
    strnset(string, letter, 13);
    printf("string after    strnset: %s\n", string);
    return 0;
 }
```

17.函数名: strpbrk
功　能: 在串中查找给定字符集中的字符
用　法: char *strpbrk(char *str1, char *str2);
程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
int main(void)
{
  char *string1 = "abcdefghijklmnopqrstuvwxyz";
  char *string2 = "onm";
  char *ptr;
      ptr = strpbrk(string1, string2);
    if (ptr)
      printf("strpbrk found first character: %c\n", *ptr);
    else
      printf("strpbrk didn't find character in set\n");
    return 0;
}
```

18.函数名: strrchr
功　能: 在串中查找指定字符的最后一个出现

用　法: char *strrchr(char *str, char c);
程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int main(void)
{
  char string[15];
  char *ptr, c = 'r';
       strcpy(string, "This is a string");
  ptr = strrchr(string, c);
  if (ptr)
  printf("The character %c is at position: %d\n", c, ptr-string);
  else
  printf("The character was not found\n");
  return 0;
}
```

19.函数名: strrev
功　能: 串倒转
用　法: char *strrev(char *str);
程序例:

```
#i nclude <string.h>
#i nclude <stdio.h>
int main(void)
{
     char *forward = "string";
     printf("Before strrev(): %s\n", forward);
     strrev(forward);
     printf("After strrev():    %s\n", forward);
     return 0;
  }
```

20.函数名: strset
功　能: 将一个串中的所有字符都设为指定字符
用　法: char *strset(char *str, char c);
程序例:

```
#i nclude <stdio.h>
#i nclude <string.h>
int main(void) {
  char string[10] = "123456789";
  char symbol = 'c';
     printf("Before strset(): %s\n", string);
     strset(string, symbol);
     printf("After strset():    %s\n", string);
```

```
    return 0;
}
```

21.函数名: strspn
功　能: 在串中查找指定字符集的子集的第一次出现
用　法: int strspn(char *str1, char *str2);
程序例:

```
#i nclude <stdio.h>
#i nclude <string.h>
#i nclude <alloc.h>
int main(void)
{
    char *string1 = "1234567890";
    char *string2 = "123DC8";
    int length;
    length = strspn(string1, string2);
    printf("Character where strings differ is at position %d\n", length);
    return 0;
}
```

22.函数名: strstr
功　能: 在串中查找指定字符串的第一次出现
用　法: char *strstr(char *str1, char *str2);
程序例:

```
#i nclude <stdio.h>
#i nclude <string.h>
int main(void)
{
    char *str1 = "Borland International", *str2 = "nation", *ptr;
    ptr = strstr(str1, str2);
    printf("The substring is: %s\n", ptr);
    return 0; }
```

23.函数名: strtod
功　能: 将字符串转换为 double 型值
用　法: double strtod(char *str, char **endptr);
程序例:

```
#i nclude <stdio.h>
#i nclude <stdlib.h>
int main(void)
{
    char input[80], *endptr;
    double ;
    printf("Enter a floating point number:");
```

```
    gets(input);
    = strtod(input, &endptr);
    printf("The string is %s the number is %lf\n", input, );
    return 0;
}
```

24.函数名: strtok
功　能: 查找由在第二个串中指定的分界符分隔开的单词
用　法: char *strtok(char *str1, char *str2);
程序例:

```
＃i nclude <string.h>
＃i nclude <stdio.h>
int main(void)
{
  char input[16] = "abc,d";
  char *p;
    /* strtok places a NULL terminator       in front of the token, if found */
    p = strtok(input, ",");
    if (p)
      printf("%s\n", p);
    /* A second call to strtok using a NULL
    as the first parameter returns a pointer
    to the character following the token
    */
    p = strtok(NULL, ",");
    if (p)
    printf("%s\n", p);
    return 0;
}
```

25.函数名: strtol
功　能: 将串转换为长整数
用　法: long strtol(char *str, char **endptr, int base);
程序例:

```
＃i nclude <stdlib.h>
＃i nclude <stdio.h>
int main(void)
{
      char *string = "87654321", *endptr;
  long lnumber;
    /* strtol converts string to long integer    */
    lnumber = strtol(string, &endptr, 10);
    printf("string = %s    long = %ld\n", string, lnumber);
    return 0;
```

}
函数名: strupr
功　能: 将串中的小写字母转换为大写字母
用　法: char *strupr(char *str);
程序例:

```
＃i nclude <stdio.h>
＃i nclude <string.h>
int main(void)
{
    char *string = "abcdefghijklmnopqrstuvwxyz", *ptr;
    /* converts string to upper case characters */
    ptr = strupr(string);        printf("%s\n", ptr);
    return 0;
}
```

26.函数名: swab
功　能: 交换字节
用　法: void swab (char *from, char *to, int nbytes);
程序例:

```
＃i nclude <stdlib.h>
＃i nclude <stdio.h>
＃i nclude <string.h>
char source[15] = "rFna koBlrna d"; char target[15];
int main(void) {
  swab(source, target, strlen(source));
  printf("This is target: %s\n", target);
  return 0;
}
```

C 语言字符串操作[转]
2009-03-04 11:06
本章集中讨论字符串操作，包括拷贝字符串，拷贝字符串的一部分，比较字符串，字符串右对齐，删去字符串前后的空格，转换字符串，等等。C 语言提供了许多用来处理字符串的标准库函数，本章将介绍其中的一部分函数。

在编写 C 程序时，经常要用到处理字符串的技巧，本章提供的例子将帮助你快速学会一些常用函数的使用方法，其中的许多例子还能有效地帮助你节省编写程序的时间。

6．1 串拷贝(strcpy)和内存拷贝(memcpy)有什么不同?它们适合于在哪种情况下使用?

strcpy()函数只能拷贝字符串。strcpy()函数将源字符串的每个字节拷贝到目录字符串中，当遇到字符串末尾的 null 字符(\0)时，它会删去该字符，并结束拷贝。

memcpy()函数可以拷贝任意类型的数据。因为并不是所有的数据都以 null 字符结束，所以你要为 memcpy()函数指定要拷贝的字节数。

在拷贝字符串时，通常都使用 strcpy()函数；在拷贝其它数据(例如结构)时，通常都使

用 memcpy()函数。

以下是一个使用 strcpy()函数和 memcpy()函数的例子：

```
#include <stdio. h>
#include <string. h>
typedef struct cust-str {
    int id ;
    char last_name [20] ;
    char first_name[l5];
} CUSTREC;
void main (void);
void main (void)
{
    char * src_string = "This is the source string" ;
    char dest_string[50];
    CUSTREC src_cust;
    CUSTREC dest_cust;
    printf("Hello! I'm going to copy src_string into dest_string!\n");
    / * Copy src_ string into dest-string. Notice that the destination
      string is the first argument. Notice also that the strcpy()
      function returns a pointer to the destination string. * /
    printf("Done! dest_string is: %s\n" ,
        strcpy(dest_string, src_string)) ;
    printf("Encore! Let's copy one CUSTREC to another. \n") ;
    prinft("I'll copy src_cust into dest_cust. \n");
    / * First, intialize the src_cust data members. * /
    src_cust. id = 1 ;
    strcpy(src_cust. last_name, "Strahan");
    strcpy(src_cust. first_name, "Troy");
    / * Now, Use the memcpy() function to copy the src-cust structure to
        the dest_cust structure. Notice that, just as with strcpy(), the
        destination comes first. * /
    memcpy(&dest_cust, &src_cust, sizeof(CUSTREC));
    printf("Done! I just copied customer number # %d (%s %s). " ,
        dest_cust. id, dest_cust. first_name, dest_cust. last_name) ;
}
```

请参见：

6．6 怎样拷贝字符串的一部分?

6．7 怎样打印字符串的一部分?

6. 2 怎样删去字符串尾部的空格?。

C 语言没有提供可删去字符串尾部空格的标准库函数，但是，编写这样的一个函数是很方便的。请看下例：

```
#include <stdio. h>
```

```
# include <string. h>

void main (void);
char * rtrim(char * );
void main(void)
{
      char * trail_str = "This string has trailing spaces in it";
    / * Show the status of the string before calling the rtrim()
         function. * /
      printf("Before calling rtrim(), trail_str is '%s'\fi" , trail_str);
      print ("and has a length of %d. \n" , strlen (trail_str));
    / * Call the rtrimO function to remove the trailing blanks. * /
      rtrim(trail_str) ;
    / * Show the status of the string
         after calling the rtrim() function. * /
      printf("After calling rttim(), trail_ str is '%s' \n", trail _ str );
      printf ("and has a length of %d. \n" , strlen(trail-str)) ;
}
/ * The rtrim() function removes trailing spaces from a string. * /.

char * rtrim(char * str)
{
      int n = strlen(str)-1; / * Start at the character BEFORE
                                          the null character (\0). * /
      while (n>0) / * Make sure we don't go out of hounds. . . * /
      {
            if ( * (str + n) 1 =' ') / * If we find a nonspace character: * /
            {
                  * (str+n+1) = '\0' ; / * Put the null character at one
                                                character past our current
                                                position. * /
                  break ; / * Break out of the loop. * /
            }
            else / * Otherwise , keep moving backward in the string. * /.
                  n--;
      }
      return str;       /*Return a pointer to the string*/
}
```

在上例中，rtrim()是用户编写的一个函数，它可以删去字符串尾部的空格。函数 rtrim()从字符串中位于 null 字符前的那个字符开始往回检查每 个字符，当遇到第一个不是空格的字符时，就将该字符后面的字符替换为 null 字符。因为在 C 语言中 null 字符是字符串的结束标志，所以函数 rtrim ()的作用实际上就是删去字符串尾部的所有空格。

请参见：

6．3 怎样删去字符串头部的空格?

6．5 怎样将字符串打印成指定长度?

6．3 怎样删去字符串头部的空格?

C 语言没有提供可删去字符串头部空格的标准库函数，但是，编写这样的一个函数是很方便的。请看下例；

```
#include <stdio. h>
#include <string. h>

void main(void);
char * ltrim (char * ) ;
char * rtrim(char * ) ;
void main (void)
{
    char * lead_str = " This string has leading spaces in it. " ;,
    / * Show the status of the string before calling the Itrim()
        function. * /
    printf("Before calling Itrim(), lead-str is '%s'\n", lead_str);
    printf("and has a length of %d. \n" , strlen(lead_str));
    / * Call the Itrim() function to remove the leading blanks. * /.
        Itrim(lead_str);
    / * Show the status of the string
        after calling the Itrim() function. * /
    prinft("After calling Itrim(), lead_str is '%s'\n", lead_str);
    print("and has a length of %d. \n" , strlen(lead-str)) ;
}
/ * The Itrim() function removes leading spaces from a string. * /

char * ltrim(char * str)
{
    strrev(str) ; / * Call strrevO to reverse the string. * /
    rtrim(str)). /* Call rtrimO to remvoe the "trailing" spaces. * /
    strrev(str); / * Restore the string's original order. * /
    return str ; / * Return a pointer to the string. * /.
}
/ * The rtrim() function removes trailing spaces from a string. * /

char* rtrim(char* str)
{
    int n = strlen (str)-l ; / * Start at the character BEFORE
    the null character (\0). * /
    while (n>0) / * Make sure we don't go out of bounds... * /.
    {
```

```
        if ( * (str+n) ! =' ') If we find a nonspace character: * /
        {
            * (str+n + 1) = '\0' ; / * Put the null character at one
            character past our current
            position. * /
            break;j / * Break out of the loop. * /
        }
        else / * Otherwise, keep moving backward in the string. * /
            n --;
    }
    return str;         /* Return a pointer tO the string.    */
}
```

在上例中，删去字符串头部空格的工作是由用户编写的 ltrim()函数完成的，该函数调用了·6．2 的例子中的 rtrim()函数和标准 C 库函数 strrev()。ltrim()函数首先调用 strrev()函数将字符串颠倒一次，然后调用 rtrim()函数删去字符串尾部的空格，最后调用 strrev()函数将字符串再颠倒一次，其结果实际上就是删去原字符串头部的空格。

请参见：

6．2 怎样删去字符串尾部的空格?

6．5 怎样将字符串打印成指定长度?

6．4 怎样使字符串右对齐?

C 语言没有提供可使字符串右对齐的标准库函数，但是，编写这样的一个函数是很方便的。请看下例：

```
#include <stdio. h>
#include <string. h>
#include <malloc. h>

void main (void);
char * r just (char * ) ;
char * rtrim(char * );
void main (void)
{
    char * rjust_str = "This string is not righ-justified. " ;
    / * Show the status of the string before calling the rjust()
        function. * /
    printf("Before calling rjust(), rjust_str is ' %s'\n. " , rjust_str);
    / * Call the rjustO function to right-justify this string. * /
    rjust(rjust_str) ;
    / * Show the status of the string
        after calling the rjust() function. * /
    printf ("After calling rjust() , rjust_str is ' %s'\n. " , rjust_str) ;
}
```

```
/ * The rjust() function right-justifies a string. * /
char * r just (char * str)
{
    int n = strlen(str); / * Save the original length of the string. * /
    char* dup_str;
    dup_str = strdup(str); / * Make an exact duplicate of the string. * /
    rtrim(dup_str); /* Trim off the trailing spaces. */
    / * Call sprintf () to do a virtual "printf" back into the original
        string. By passing sprintf () the length of the original string,
        we force the output to be the same size as the original, and by
        default the sprintf() right-justifies the output. The sprintf()
        function fills the beginning of the string with spaces to make
        it the same size as the original string. * /

    sprintf(str, "%*. * s", n, n, dup_str);

    free(dup-str) ; / * Free the memory taken by
    the duplicated string. * /
    return str;\ / * Return a pointer to the string. * /
}

/ * The rtrim() function removes trailing spaces from a string. * /
char * rtrim(char * str)
{
    int n = strlen(str)-l; / * Start at the character BEFORE the null
                                    character (\0). * /
    while (n>0) / * Make sure we don't go out of bounds... * /
    {
        if ( * (str+n) ! = ' ') / * If we find a nonspace character: * /
        {
            * (str + n + 1) = '\0';( / * Put the null character at one
                                        character past our current
                                        position. * /
             break; / * Break out of the loop. * /
        }
        else / * Otherwise, keep moving backward in the string. * /
            n—;
    }
    return str ; / * Return a pointer to the string. * /
}
```

在上例中，使字符串右对齐的工作是由用户编写的 rjust()函数完成的，该函数调用了 6．2 的例子中的 rtrim()函数和几个标准函数。rjust()函数的工作过程如下所示：

(1) 将原字符串的长度存到变量 n 中。这一步是不可缺少的，因为输出字符串和原字符

串的长度必须相同。

(2) 调用标准 C 库函数 strdup()，将原字符串复制到 dup_str 中。原字符串需要有一份拷贝，因为经过右对齐处理的字符串要写到原字符串中。

(3) 调用 rtrim()函数，删去 dup_str 尾部的空格。

(4) 调用标准 C 库函数 sprinf()，将 dup_str 写到原字符串中。由于原字符串的长度(存在 n 中)被传递给 sprintf()函数，所以迫使输出字符 串的长度和原字符串相同。因为 sprintf()函数缺省使输出字符串右对齐，因此输出字符串的头部将被加入空格，以使它和原字符串长度相同，其效果实际 上就是使原字符串右对齐。

(5)调用标准库函数 free()，释放由 strdup()函数分配给 dup_str 的动态内存。

请参见：

6．5 怎样将字符串打印成指定长度?

6．5 怎样将字符串打印成指定长度?

如果要按表格形式打印一组字符串，你就需要将字符串打印成指定长度。利用 printf()函数可以很方便地实现这一点，请看下例:

```
# include <stdio. h>
char * data[25] = {
    "REGION", "--Q1--", "--Q2--", "--Q3--", "--Q4--",
    "North" , "10090. 50" , "12200. 10" , "26653.12" , "62634. 32" ,
    "South", "21662.37", "95843.23", "23788.23", "48279.28",
    "East", "23889.38", "23789.05", "89432.84", "29874.48",
    "West", "85933.82", "74373.23", "78457.23", "28799.84" };
void main (void) ;
void main (void)
{
    int x;
    fox (x = 0, x<25; x+ + )
    {
        if ((x % 5) == 0&&(x !=0))
        printf("\n");
        printf (" %-10. 10s" , data[x]) ;
    }
}
```

在上例中，字符串数组 char *data[]中包含了某年 4 个地区的销售数据。显然，你会要求按表格形式打印这些数据，而不是一个挨一个地毫无格式地打印这些数据。因此，上例中用下述语句来打印这些数据：

printf("%-10．10s"，data[x])；

参数"%-10．10s"指示 printf()函数按 10 个字符的长度打印一个字符串。在缺省情况下，printf()函数按右对齐格式打印字符串，但 是，在第一个 10 的前面加上减号(-)后，prinft()函数，就会使字符串左对齐。为此，printf()函数会在字符串的尾部加入空格，以使其长度 达到 10 个字符。上例的打印输出非常整洁，类似于一张表格，如下所示：

REGION	--Q1--	--Q2--	--Q3--	--Q4--
North	10090.50	12200.10	26653.12	62634.32
SOuth	2166 2.37	95843.23	23788.23	48279.28
East	23889.38	23789.05	89432.84	29874.48
West	85933.82	74373.23	78457.23	28799.84

请参见：

6．4 怎样使字符串右对齐?

6．6．怎样拷贝字符串的一部分?

利用标准库函数 strncpy()，可以将一字符串的一部分拷贝到另一个字符串中。strncpy()函数有 3 个参数：第一个参数是目录字符串；第二个参 数是源字符串；第三个参数是一个整数，代表要从源字符串拷贝到目标字符串中的字符数。以下是一个用 strncpy()函数拷贝字符串的一部分的例子：

```
# include <stdio. h>
# include <string. h>

void main(void);
void main (void)
{
    char * source_str = "THIS IS THE SOURCE STRING" ;
    char dest_strl[40]= {0}, dest_str2[40]= {0};
    / * Use strncpy() to copy only the first 11 characters. * /
    strncpy(dest_strl, source-str, 11);
    printf("How about that! dest-strl is now: '%s'!!!\n", dest-strl);
    / * Now, use strncpy() to copy only the last 13 characters. * /
    strncpy(dest_strl, source_str + (strlen(source_str)-l3) , 13);
    printf("Whoa! dest_str2 is now: '%s'!!!\n". dest_str2);
}
```

在上例中，第一次调用 strncpy()函数时，它将源字符串的头 11 个字符拷贝到 dest_str1 中，这是一种相当直接的方法，你可能会经常用到。第二次调用 strncpy()函数时，它将源字符串的最后 13 个字符拷贝到 dest_str2 中，其实现过程为：

(1)用 strlen()函数计算出 source_str 字符串的长度，即 strlen(source_str)。

(2)将 source_str 的长度减去 13(13 是将要拷贝的字符数)，得出 source_str 中剩余的字符数，即 pstrlen(source_str)-13。

(3)将 strlen(source_str)-13 和 source_str 的地址相加，得出指向 source_str 中倒数第 13 个字符的地址的指针， 即 source_str+(strlen(source_str)-13)。这个指针就是 strncpy()函数的第二个参数。

(4)在 strncpy()函数的第三个参数中指定要拷贝的字符是 13。

上例的打印输出如下所示：

How about that! dest_str1 is now：'THIS IS THE'!!!
Whoa! dest_str2 is now：'SOURCE STRING'!!!

需要注意的是，在将 source_str 拷贝到 dest_strl 和 dest_str2 之前，dest_strl 和 dest_str2 都要被初始化为 null 字符(\0)。这是因为 strncpy()函数在拷贝字符串时不会自动将 null 字符添加到目录字符串后面，因此你必须确保在目标字符串的后面加 上 null 字符，否则会导致打印出一些杂乱无章的字符。

请参见；
6．1 串拷贝(strcpy)和内存拷贝(memcpy)有什么不同?它们适合于在哪种情况下使用?
6．9 怎样打印字符串的一部分?

6．7 怎样将数字转换为字符串?
C 语言提供了几个标准库函数，可以将任意类型(整型、长整型、浮点型等)的数字转换为字符串。以下是用 itoa()函数将整数转换为字符串的一个例子：

```
# include <stdio. h>
# include <stdlib. h>

void main (void);
void main (void)
{
    int num = 100;
    char str[25];
    itoa(num, str, 10);
    printf("The number 'num' is %d and the string 'str' is %s. \n" ,
                                num, str);
}
```

itoa()函数有 3 个参数：第一个参数是要转换的数字，第二个参数是要写入转换结果的目标字符串，第三个参数是转移数字时所用的基数。在上例中，转换基数为 10。

下列函数可以将整数转换为字符串：

函数名	作 用
itoa()	将整型值转换为字符串
itoa()	将长整型值转换为字符串
ultoa()	将无符号长整型值转换为字符串

请注意，上述函数与 ANSI 标准是不兼容的。能将整数转换为字符串而且与 ANSI 标准兼容的方法是使用 sprintf()函数，请看下例：

```
#include<stdio.h>
# include <stdlib. h>
```

```
void main (void);
void main (void)
{
      int num = 100;
      char str[25];
      sprintf(str, " %d" , num);
     printf ("The number 'num' is %d and the string 'str' is %s. \n" ,
                                        num, str);

}
```

在将浮点型数字转换为字符串时，需要使用另外一组函数。以下是用 fcvt()函数将浮点型值转换为字符串的一个例子：

```
# include <stdio. h>
# include <stdlib. h>

void main (void);
void main (void)
{
      double num = 12345.678;
      char * sir;
      int dec_pl, sign, ndigits = 3; /* Keep 3 digits of precision. * /
      str = fcvt(num, ndigits, &dec-pl, &sign); /* Convert the float
                                                          to a string. * /
      printf("Original number; %f\n" , num) ; /* Print the original
                                                          floating-point
                                                              value. * /
      printf ("Converted string; %s\n",str);       /* Print the converted
                                                         string's value. * /
      printf ("Decimal place: %d\n" , dec-pi) ; /* Print the location of
                                                          the decimal point. * /
      printf ("Sign: %d\n" , sign) ;                  /* Print the sign.
                                                          0 = positive,
                                                          1 = negative. * /
}
```

fcvt()函数和 itoa()函数有数大的差别。fcvt()函数有 4 个参数：第一个参数是要转换的浮点型值；第二个参数是转换结果中十进制小数点右侧 的位数；第三个参数是指向一个整数的指针，该整数用来返回转换结果中十进制小数点的位置；第四个参数也是指向一个整数的指针，该整数用来返回转换结果的符 号(0 对应于正值，1 对应于负值)。

需要注意的是，fcvt()函数的转换结果中并不真正包含十进制小数点，为此，fcvt()函数返回在转换结果中十进制小数点应该占据的位置。在上例中， 整型变量 dec_pl 的结果值为

5，因为在转换结果中十进制小数点应该位于第 5 位后面。如果你要求转换结果中包含十进制小数点，你可以使用 gcvt()函 数(见下表)。

下列函数可以将浮点型值转换为字符串：

函数名	作 用
ecvt()	将双精度浮点型值转换为字符串，转换结果中不包含十进制小数点
fcvt()	以指定位数为转换精度，余同 ecvt()
gcvt()	将双精度浮点型值转换为字符串，转换结果中包含十进制小数点

请参见：
6．8 怎样将字符串转换为数字?

6．8 怎样将字符串转换为数字?

C 语言提供了几个标准库函数，可以将字符串转换为任意类型(整型、长整型、浮点型等)的数字。以下是用 atoi()函数将字符串转换为整数的一个例子：

```
# include <stdio. h>
# include <stdlib. h>

void main (void) ;
void main (void)
{
    int num;
    char * str = "100";
    num = atoi(str);
    printf("The string 'str' is %s and the number 'num' is %d. \n",
                        str, num);
}
```

atoi()函数只有一个参数，即要转换为数字的字符串。atoi()函数的返回值就是转换所得的整型值。

下列函数可以将字符串转换为数字：

函数名	作 用
atof()	将字符串转换为双精度浮点型值
atoi()	将字符串转换为整型值
atol()	将字符串转换为长整型值
strtod()	将字符串转换为双精度浮点型值，并报告不能被转换的所有剩余数字
strtol()	将字符串转换为长整值，并报告不能被转换的所有剩余数字

strtoul() 将字符串转换为无符号长整型值，并报告不能被转换的所有剩余数字

将字符串转换为数字时可能会导致溢出，如果你使用的是 strtoul()这样的函数，你就能检查这种溢出错误。请看下例：

```
# include <stdio. h>
# include <stdlib. h>
# include <limits. h>

void main(void);
void main (void)
{
    char* str = "12345678910111213141516171819 20" ;
    unsigned long num;
    char * leftover;
    num = strtoul(str, &leftover, 10);
    printf("Original string: %s\n",str);
    printf("Converted number: %1u\n" , num);
    printf("Leftover characters: %s\n" , leftover);
}
```

在上例中，要转换的字符串太长，超出了无符号长整型值的取值范围，因此，strtoul()函数将返回 ULONG_MAX(4294967295)，并 使。char leftover 指向字符串中导致溢出的那部分字符；同时，strtoul()函数还将全局变量 errno 赋值为 ERANGE，以通知函数的调用者发生了 溢出错误。函数 strtod()和 strtol()处理溢出错误的方式和函数 strtoul()完全相同，你可以从编译程序文档中进一步了解这三个函数的 有关细节。

请参见：

6．7 怎样将数字转换为字符串?

6．9 怎样打印字符串的一部分?

6．6 中讨论了怎样拷贝字符串的一部分，为了打印字符串的一部分，你可以利用 6．6 的例子中的部分技巧，不过你现在要使用的是 printf()函数，而不是 sprintf()函数。请看下例：

```
# include <stdio. h>
# include <stdlib. h>

void main (void);
void main (void)
{
    char * source_str = "THIS IS THE SOURCE STRING" ;
    / * Use printfO to print the first 11 characters of source_str. * /
    printf("First 11 characters: ' %11. lls'\n" , source_str);
    / * Use printf() to print only the
```

```
        last 13 characters of source _str. * /
    printf("Last 13 characters:'%13.13'\n",
                        source_str+(strlen(source_str)-13));
}
```

上例的打印输出如下所示：

First 11 characters: 'THIS IS THE'

Last 13 characters：'SOURCE STRING'

在上例中，第一次调用 printf()函数时，通过指定参数"%11.11s"，迫使 printf()函数只打印 11 个字符的长度，因为源字符串的长度大 于 11 个字符，所以在打印时源字符串将被截掉一部分，只有头 11 个字符被打印出来。第二次调用 printf()函数时，它将源字符串的最后 13 个字符打印 出来，其实现过程为：

(1)用 strlen()函数计算出 source_str 字符串的长度，即 strlen(source_str)。

(2)将 source_str 的长度减去 13(13 是将要打印的字符数)，得出 source_str 中剩余字符数，且 pstrlen(source_str)-13。

(3)将 strlen(source_str)-13 和 source_str 的地址相加，得出指向 source_str 中倒数第 13 个字符的地址的指针； 即 source_str+(strlen(source_str)-13)。这个指针就是 printf()函数的第二个参数。

(4)通过指定参数“％13．13s”，迫使 printf()函数只打印 13 个字符的长度，其结果实际上就是打印源字符串的最后 13 个字符。

请参见：

6．1 串拷贝(strcpy)和内存拷贝(memcpy)有什么不同?它们适合于在哪种情况下使用？

6．6 怎样拷贝字符串的一部分?

6．10 怎样判断两个字符串是否相同?

C 语言提供了几个标准库函数，可以比较两个字符串是否相同。以下是用 strcmp()函数比较字符串的一个例子：

```
#include <stdio. h>
#include <string. h>

void main (void);
void main(void)
{
    char* str_1 = "abc" ; char * str_2 = "abc" ; char* str_3 = "ABC" ;
    if (strcmp(str_1, str_2) == 0)
        printf("str_1 is equal to str_2. \n");
    else
        printf("str_1 is not equal to str_2. \n");
    if (strcmp(str_1, str_3) == 0)
       printf("str_1 is equal to str_3．\n");
    else
        printf("str_1 is not equalto str_3．\n");
}
```

上例的打印输出如下所示：

str_1 is equal to str_2.

str_1 is not equal to str_3.

strcmp()函数有两个参数，即要比较的两个字符串。strcmp()函数对两个字符串进行大小写敏感的(case-sensitiVe)和字典式的(lexicographic)比较，并返回下列值之一：

返 回 值	意 义
<0	第一个字符串小于第二个字符串
0	两个字符串相等
>0	第一个字符串大于第二个字符串

在上例中，当比较 str_1(即“abc”)和 str_2(即“abc”)时，strcmp()函数的返回值为 0。然
而，当比较 str_1(即"abc")和 str_3(即"ABC")时，strcmp()函数返回一个大于 0 的值，因为按 ASCII 顺序字符串“ABC”小于“abc”。

strcmp()函数有许多变体，它们的基本功能是相同的，都是比较两个字符串，但其它地方
稍有差别。下表列出了 C 语言提供的与 strcmp()函数类似的一些函数：

函 数 名	作 用
strcmp()	对两个字符串进行大小写敏感的比较
strcmpi()	对两个字符串进行大小写不敏感的比较
stricmp()	同 strcmpi()
strncmp()	对两个字符串的一部分进行大小写敏感的比较
strnicmp()	对两个字符串的一部分进行大小写不敏感的比较

在前面的例子中，如果用 strcmpi()函数代替 strcmp()函数，则程序将认为字符串“ABC”等于“abc”。

请参见：

6．1 串拷贝(strcpy)和内存拷贝(memcpy)有什么不同?它们适合于在哪种情况下使用?