Example 8.1

Write a program to read a series of words from a terminal using **scanf** function

The program shown in Fig.8.1 reads four words and displays them on the screen. Note that the string 'Oxford Road' is treated as *two words* while the string 'Oxford-Road' as *one word.*

READING A SERIES OF WORDS USING **scanf** FUNCTION

Program

```
main( )
{
    char word1[40], word2[40], word3[40], word4[40];

    printf("Enter text : \n");
    scanf("%s %s", word1, word2);
    scanf("%s", word3);
    scanf("%s", word4);

    printf("\n");
    printf("word1 = %s\nword2 = %s\n", word1, word2);
    printf("word3 = %s\nword4 = %s\n", word3, word4);
}
```

Output

```
Enter text :
Oxford Road, London M17ED

word1 = Oxford
word2 = Road,
word3 = London
word4 = M17ED

Enter text :
Oxford-Road, London-M17ED United Kingdom
word1 = Oxford-Road
word2 = London-M17ED
word3 = United
word4 = Kingdom
```

Fig.8.1 Reading a series of words using ***scanf***

Example 8.2

Write a program to read a line of text containing a series of words from the terminal.

The program shown in Fig.8.2 can read a line of text (upto a maximum of 80 characters) into the string **line** using **getchar** function. Every time a character is read, it is assigned to its location in the string **line** and then tested for *newline* character. When the *newline* character is read (signalling the end of line), the reading loop is terminated and the *newline* character is replaced by the null character to indicate the end of character string.

When the loop is exited, the value of the index **c** is one number higher than the last character position in the string (since it has been incremented after assigning the new character to the string). Therefore the index value **c-1** gives the position where the *null* character is to be stored.

PROGRAM TO **READ A LINE OF TEXT** FROM TERMINAL

Program

```
#include  <stdio.h>

main( )
{
    char  line[81], character;
    int   c;

    c = 0;

    printf("Enter text. Press <Return> at end\n");
    do
    {
        character = getchar();
        line[c]   = character;
        c++;
    }
    while(character != '\n');

    c = c - 1;
    line[c] = '\0';
    printf("\n%s\n", line);
}
```

Output

```
Enter text. Press <Return> at end
Programming in C is interesting.

Programming in C is interesting.

Enter text. Press <Return> at end
National Centre for Expert Systems, Hyderabad.

National Centre for Expert Systems, Hyderabad.
```

Fig.8.2 *Program to* ***read a line of text*** *from terminal*

Example 8.3

Write a program to copy one string into another and count the number of characters copied.

The program is shown in Fig.8.3. We use a **for** loop to copy the characters contained inside **string2** into the **string1**. The loop is terminated when the *null* character is reached. Note that we are again assigning a null character to the **string1.**

COPYING ONE **STRING** INTO **ANOTHER**

Program

```
main( )
{
    char  string1[80], string2[80];
    int   i;

    printf("Enter a string \n");
    printf("?");

    scanf("%s", string2);

    for( i=0 ; string2[i] != '\0'; i++)
       string1[i] = string2[i];

    string1[i] = '\0';

    printf("\n");
    printf("%s\n", string1);
    printf("Number of characters = %d\n", i );
}
```

Output

```
Enter a string
?Manchester

Manchester
Number of characters = 10

Enter a string
?Westminster

Westminster
Number of characters = 11
```

Fig.8.3 *Copying one string into another*

Example 8.4

Write a program to store the string "United Kingdom" in the array **country** and display the string under various format specifications.

The program and its output are shown in Fig.8.4. The output illustrates the following features of the **%s** specifications.

1. When the field width is less than the length of the string, the entire string is printed.
2. The integer value on the right side of the decimal point specifies the number of characters to be printed.
3. When the number of characters to be printed is specified as zero, nothing is printed.
4. The minus sign in the specification causes the string to be printed left-justified.
5. The specification % **.**ns prints the first n characters of the string

```
WRITING STRINGS USING %s FORMAT
```

Program

```
main()
{
    char  country[15] = "United Kingdom";

    printf("\n\n");
    printf("*123456789012345*\n");
    printf(" --------------- \n");

    printf("%15s\n", country);
    printf("%5s\n", country);
    printf("%15.6s\n", country);
    printf("%-15.6s\n", country);
    printf("%15.0s\n", country);
    printf("%.3s\n", country);
    printf("%s\n", country);
    printf("---------------- \n");
}
```

Output

```
*123456789012345*
------------------
  United Kingdom
 United Kingdom
          United
 United

 Uni
 United Kingdom
------------------
```

Fig.8.4 *Writing strings using %s format*

Example 8.5

Write a program using **for loop** to print the following output.

```
C
CP
CPr
CPro
.....
.....
CProgramming
____________
CProgramming
.....
.....
CPro
CPr
CP
C
```

The outputs of the program in Fig.8.5, for variable specifications **%12.*s, %.*s,** and **%*.1s** are shown in Fig.8.6, which further illustrates the variable field width and the precision specifications.

PRINTING **SEQUENCES OF CHARACTERS**

Program

```
main()
{
    int  c, d;
    char  string[] = "CProgramming";

    printf("\n\n");
    printf("--------------\n");
    for( c = 0 ; c <= 11 ; c++ )
    {
       d = c + 1;
       printf("|%-12.*s|\n", d, string);
    }
    printf("|------------|\n");

    for( c = 11 ; c >= 0 ; c-- )
    {
       d = c + 1;
       printf("|%-12.*s|\n", d, string);
    }
    printf("--------------\n");
}
```

Output

```
C
CP
CPr
CPro
CProg
CProgr
CProgra
CProgram
CProgramm
CProgrammi
CProgrammin
CProgramming
```

```
CProgramming
CProgrammin
CProgrammi
CProgramm
CProgram
CProgra
CProgr
CProg
CPro
CPr
CP
C
```

Fig.8.5 *Illustration of variable field specifications*

Example 8.6

Write a program which would print the alphabet set a to z and A to Z in decimal and character form.

The program is shown in Fig.8.7. In ASCII character set, the decimal numbers 65 to 90 represent uppercase alphabets and 97 to 122 represent lowercase alphabets. The values from 91 to 96 are excluded using an **if** statement in the **for** loop.

PRINTING ALPHABET SET IN DECIMAL AND CHARACTER FORM

Program

```
main()
{
    char  c;

    printf("\n\n");
    for( c = 65 ; c <= 122 ; c = c + 1 )
    {
       if( c > 90  &&  c < 97 )
         continue;
```

```
            printf("|%4d - %c ", c, c);
        }
        printf("|\n");
    }
```

Output

```
| 65 - A |  66 - B |  67 - C |  68 - D |  69 - E |  70 - F
| 71 - G |  72 - H |  73 - I |  74 - J |  75 - K |  76 - L
| 77 - M |  78 - N |  79 - O |  80 - P |  81 - Q |  82 - R
| 83 - S |  84 - T |  85 - U |  86 - V |  87 - W |  88 - X
| 89 - Y |  90 - Z |  97 - a |  98 - b |  99 - c | 100 - d
| 101 - e | 102 - f | 103 - g | 104 - h | 105 - i | 106 - j
| 107 - k | 108 - l | 109 - m | 110 - n | 111 - o | 112 - p
| 113 - q | 114 - r | 115 - s | 116 - t | 117 - u | 118 - v
| 119 - w | 120 - x | 121 - y | 122 - z |
```

Fig.8.7 *Printing of the alphabet set in decimal and character form*

Example 8.7

The names of employees of an organization are stored in three arrays, namely **first_name**, **second_name,** and **last_name**. Write a program to concatenate the three parts into one string to be called **name**.

The program is given in Fig.8.8. Three **for** loops are used to copy the three strings. In the first loop, the characters contained in the **first_name** are copied into the variable **name** until the *null* character is reached. The *null* character is not copied; instead it is replaced by a *space* by the assignment statement

name[i] = ' ' **;**

Similarly, the **second_name** is copied into **name**, starting from the column just after the space created by the above statement. This is achieved by the assignment statement

name[i+j+1] = second_name[j];

If **first_name** contains 4 characters, then the value of i at this point will be 4 and therefore the first character from **second_name** will be placed in the *fifth cell* of **name.** Note that we have stored a space in the *fourth cell*.

In the same way, the statement

name[i+j+k+2] = last_name[k];

is used to copy the characters from **last_name** into the proper locations of **name.**
At the end, we place a null character to terminate the concatenated string **name**. In this example, it is important to note the use of the expressions **i+j+1** and **i+j+k+2.**

CONCATENATION OF STRINGS

Program

```
main()
{
    int  i, j, k ;
    char   first_name[10] = {"VISWANATH"}  ;
    char  second_name[10] = {"PRATAP"} ;
    char    last_name[10] = {"SINGH"} ;
    char   name[30] ;

  /* Copy first_name into name */

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

  /* End first_name with a space */

    name[i] = ' ' ;

  /* Copy second_name into name */

    for( j = 0 ; second_name[j] != '\0' ; j++ )
       name[i+j+1] = second_name[j] ;

  /* End second_name with a space */

    name[i+j+1] = ' ' ;

  /* Copy last_name into name */

    for( k = 0 ; last_name[k] != '\0'; k++ )
       name[i+j+k+2] = last_name[k] ;

  /* End name with a null character */

    name[i+j+k+2] = '\0' ;

    printf("\n\n") ;
    printf("%s\n", name) ;
}
```

Output

```
            VISWANATH PRATAP SINGH
```

Fig.8.8 *Concatenation of strings*

Example 8.8

s1, s2, and **s3** are three string variables. Write a program to read two string constants into **s1** and **s2** and compare whether they are equal or not. If they are not, join them together. Then copy the contents of **s1** to the variable **s3**. At the end, the program should print the contents of all the three variables and their lengths.

The program is shown in Fig.8.9. During the first run, the input strings are "New" and "York". These strings are compared by the statement

```
x = strcmp(s1, s2);
```

Since they are not equal, they are joined together and copied into **s3** using the statement

```
strcpy(s3, s1);
```

The program outputs all the three strings with their lengths.

During the second run, the two strings **s1** and **s2** are equal, and therefore, they are not joined together. In this case all the three strings contain the same string constant "London".

ILLUSTRATIONS OF STRING-HANDLING FUNCTIONS

Program

```
#include    <string.h>

main()
{   char  s1[20], s2[20], s3[20];
    int   x, l1, l2, l3;

    printf("\n\nEnter two string constants \n");
    printf("?");
    scanf("%s %s", s1, s2);

 /* comparing s1 and s2 */

    x = strcmp(s1, s2);
    if(x != 0)
    {   printf("\n\nStrings are not equal \n");
        strcat(s1, s2);   /* joining s1 and s2 */
    }
    else
        printf("\n\nStrings are equal \n");

/* copying s1 to s3

    strcpy(s3, s1);

 /* Finding length of strings */

    l1 = strlen(s1);
    l2 = strlen(s2);
    l3 = strlen(s3);

 /* output */

    printf("\ns1 = %s\t length = %d characters\n", s1, l1);
    printf("s2 = %s\t length = %d characters\n", s2, l2);
    printf("s3 = %s\t length = %d characters\n", s3, l3);

}
```

Output

```
     Enter two string constants
     ? New York
     Strings are not equal
     s1 = NewYork     length = 7 characters
     s2 = York        length = 4 characters
     s3 = NewYork     length = 7 characters

     Enter two string constants
     ? London   London
     Strings are equal
     s1 = London      length = 6 characters
     s2 = London      length = 6 characters
     s3 = London      length = 6 characters
```

Fig.7.9 *Illustration of string handling functions*

Example 8.9

Write a program that would sort a list of names in alphabetical order.

A program to sort the list of strings in alphabetical order is given in Fig.8.10. It employs the method of bubble sorting described in Case Study 1 in the previous chapter.

```
              SORTING OF STRINGS IN ALPHABETICAL ORDER
```

Program

```
   #define ITEMS   5
   #define MAXCHAR 20
   main( )
   {
        char string[ITEMS][MAXCHAR], dummy[MAXCHAR];
         int  i = 0, j = 0;
        /* Reading the list */
        printf ("Enter names of %d items \n ",ITEMS);
        while (i < ITEMS)
             scanf ("%s", string[i++]);
        /* Sorting begins */
        for (i=1; i < ITEMS; i++) /* Outer loop begins */
        {
             for (j=1; j <= ITEMS-i ; j++) /*Inner loop begins*/
         {
              if (strcmp (string[j-1], string[j]) > 0)
              {   /* Exchange of contents */
                  strcpy (dummy, string[j-1]);
                  strcpy (string[j-1], string[j]);
                  strcpy (string[j], dummy );
              }

         } /* Inner loop ends */

    } /* Outer loop ends */

    /* Sorting completed */

    printf ("\nAlphabetical list \n\n");
    for (i=0; i < ITEMS ; i++)
        printf ("%s", string[i]);
}
```

Output

```
Enter names of 5 items
London Manchester Delhi Paris Moscow
Alphabetical list

Delhi
London
Manchester
Moscow
Paris
```

Fig.8.10 *Sorting of strings.*