Example 4.1

The program in Fig.4.1 shows the use of **getchar** function in an interactive environment.

The program displays a question of YES/NO type to the user and reads the user's response in a single character (Y or N). If the response is Y, it outputs the message

My name is BUSY BEE

otherwise, outputs.

You are good for nothing

Note there is one line space between the input text and output message.

READING A ***CHARACTER*** *FROM KEYBOARD*

Program

```
#include   <stdio.h>

main()
{
   char answer;

   printf("Would you like to know my name?\n");
   printf("Type Y for YES and N for NO:  ");

   answer = getchar();  /* .... Reading a character...*/

   if(answer == 'Y' || answer == 'y')
     printf("\n\nMy name is BUSY BEE\n");
   else
     printf("\n\nYou are good for nothing\n");
}
```

Output

```
Would you like to know my name?
Type Y for YES and N for NO:  Y

My name is BUSY BEE

Would you like to know my name?
Type Y for YES and N for NO:  n
```

```
You are good for nothing
```

Fig.4.1 *Use of* ***getchar*** *function*

Example 4.2

The program of Fig.4.2 requests the user to enter a character and displays a message on the screen telling the user whether the character is an alphabet or digit, or any other special character.

This program receives a character from the keyboard and tests whether it is a letter or digit and prints out a message accordingly. These tests are done with the help of the following functions:

isalpha(character)
isdigit(character)

For example, **isalpha** assumes a value non-zero (TRUE) if the argument **character** contains an alphabet; otherwise it assumes 0 (FALSE). Similar is the case with the function **isdigit**.

TESTING CHARACTER TYPE

Program:

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

main()
{
    char character;
    printf("Press any key\n");

    character = getchar();

    if (isalpha(character) > 0)
       printf("The character is a letter.");

    else
       if (isdigit (character) > 0)
          printf("The character is a digit.");

       else
          printf("The character is not alphanumeric.");
}
```

Output

```
Press any key
h
The character is a letter.

Press any key
5
```

```
The character is a digit.

Press any key
*
The character is not alphanumeric.
```

Fig.4.2 Program to test the ***characte****r type*

Example 4.3

A program that reads a character from keyboard and then prints it in reverse case is given in Fig.4.3. That is, if the input is upper case, the output will be lower case and vice versa.

The program uses three new functions: **islower, toupper**, and **tolower**. The function **islower** is a conditional function and takes the value TRUE if the argument is a lower case alphabet; otherwise takes the value FALSE. The function **toupper** converts the lower case argument into an upper case alphabet while the function **tolower** does the reverse.

WRITING A **CHARACTER** TO THE SCREEN

Program

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

main()
{
    char alphabet;

    printf("Enter an alphabet");
    putchar('\n');   /* move to next line */
    alphabet = getchar();

    if (islower(alphabet))
       putchar(toupper(alphabet));
    else
       putchar(tolower(alphabet));
}
```

Output

```
Enter an alphabet
a
A
Enter an alphabet
Q
q
Enter an alphabet
z
Z
```

Fig.4.3 *Reading and writing of alphabets in reverse case*

Example 4.4

Various input formatting options for reading integers are experimented in the program shown in Fig. 4.4.

The first **scanf** requests input data for three integer values **a, b,** and **c,** and accordingly three values 1, 2, and 3 are keyed in. Because of the specification %*d the value 2 has been skipped and 3 is assigned to the variable **b**. Notice that since no data is available for c, it contains garbage.

The second **scanf** specifies the format %2d and %4d for the variables **x** and **y** respectively. Whenever we specify field width for reading integer numbers, the input numbers should not contain more digits that the specified size. Otherwise, the extra digits on the right-hand side will be truncated and assigned to the next variable in the list. Thus, the second **scanf** has truncated the four digit number 6789 and assigned 67 to **x** and 89 to **y**. The value 4321 has been assigned to the first variable in the immediately following **scanf** statement.

READING **INTEGER** NUMBERS

Program:

```
main()
{
    int a,b,c,x,y,z;
    int p,q,r;

    printf("Enter three integer numbers\n");
    scanf("%d %*d %d",&a,&b,&c);
    printf("%d %d %d \n\n",a,b,c);

    printf("Enter two 4-digit numbers\n");
    scanf("%2d %4d",&x,&y);
    printf("%d %d\n\n", x,y);

    printf("Enter two integers\n");
    scanf("%d %d", &a,&x);
    printf("%d %d \n\n",a,x);

    printf("Enter a nine digit number\n");
    scanf("%3d %4d %3d",&p,&q,&r);
    printf("%d %d %d \n\n",p,q,r);

    printf("Enter two three digit numbers\n");
    scanf("%d %d",&x,&y);
    printf("%d %d",x,y);
}
```

Output

```
Enter three integer numbers
1 2 3
1 3 -3577

Enter two 4-digit numbers
6789  4321
67 89

Enter two integers
44  66
4321 44

Enter a nine-digit number
123456789
66 1234 567

Enter two three-digit numbers
123  456
89 123
```

Fig.4.4 *Reading integers using* **scanf**

Example 4.5

Reading of real numbers (in both decimal point and exponential notation) is illustrated in Fig.4.5.

READING OF **REAL** NUMBERS

Program:

```
main()
{
    float x,y;
    double p,q;

    printf("Values of x and y:");
    scanf("%f %e", &x, &y);
    printf("\n");
    printf("x = %f\ny = %f\n\n", x, y);

    printf("Values of p and q:");
    scanf("%lf %lf", &p, &q);
    printf("\np = %lf\nq = %e",p,q);
    printf("\n\np = %.12lf\np = %.12e", p,q);
}
```

Output

```
Values of x and y:12.3456  17.5e-2

x = 12.345600
y = 0.175000

Values of p and q:4.142857142857  18.5678901234567890

p = 4.142857142857
q = 1.856789012346e+001
```

Fig.4.5 *Reading of **real** numbers*

Example 4.6

Reading of strings using **%wc** and **%ws** is illustrated in Fig.4.6.

The program in Fig.4.6 illustrates the use of various field specifications for reading strings. When we use **%wc** for reading a string, the system will wait until the w^{th} character is keyed in.

Note that the specification **%s** terminates reading at the encounter of a blank space. Therefore, **name2** has read only the first part of "New York" and the second part is automatically assigned to **name3**. However, during the second run, the string "New-

York" is correctly assigned to **name2**.

READING STRINGS

Program

```
main()
{
    int no;
    char name1[15], name2[15], name3[15];

    printf("Enter serial number and name one\n");
    scanf("%d %15c", &no, name1);
    printf("%d %15s\n\n", no, name1);

    printf("Enter serial number and name two\n");
    scanf("%d %s", &no, name2);
    printf("%d %15s\n\n", no, name2);

    printf("Enter serial number and name three\n");
    scanf("%d %15s", &no, name3);
    printf("%d %15s\n\n", no, name3);
}
```

Output

```
Enter serial number and name one
1 123456789012345
1 123456789012345r

Enter serial number and name two
2 New York
2             New

Enter serial number and name three
2            York

Enter serial number and name one
1 123456789012
1 123456789012   r

Enter serial number and name two
2 New-York
2 New-York
Enter serial number and name three
3 London
3 London
```

Fig. 4.6 Reading of strings

Example 4.7

The program in Fig. 4.7 illustrates the function of %[ ] specification.

ILLUSTRATION OF %[ ] SPECIFICATION

Program-A

```
main()
{
    char address[80];
    printf("Enter address\n");
    scanf("%[a-z’]", address);
    printf("%-80s\n\n", address);
}
```

Output

```
Enter address
new delhi 110002
new delhi
```

ILLUSTRATION OF %[^] SPECIFICATION

Program-B

```
main()
{
    char address[80];

    printf("Enter address\n");
    scanf("%[^\n]", address);
    printf("%-80s", address);
}
```

Output

```
Enter address
New Delhi 110 002
New Delhi 110 002
```

Fig.4.7 *Illustration of conversion specification%[] for strings*

Example 4.8

The program presented in Fig.4.8 illustrates the testing for correctness of reading of data by **scanf** function.

The function **scanf** is expected to read three items of data and therefore, when the values for all the three variables are read correctly, the program prints out their values. During the third run, the second item does not match with the type of variable and therefore the reading is terminated and the error message is printed. Same is the case with the fourth run.

In the last run, although data items do not match the variables, no error message has been printed. When we attempt to read a real number for an **int** variable, the integer part is assigned to the variable, and the truncated decimal part is assigned to the next variable. Note that the character `2' is assigned to the character variable c.

TESTING FOR CORRECTNESS OF INPUT DATA

Program

```
main()
{
     int a;
     float b;
     char c;

     printf("Enter values of a, b and c\n");

     if (scanf("%d %f %c", &a, &b, &c) == 3)
        printf("a = %d  b = %f   c = %c\n" , a, b, c);
     else
        printf("Error in input.\n");
}
```

Output

```
Enter values of a, b and c
12  3.45  A
a = 12  b = 3.450000   c = A

Enter values of a, b and c
23  78  9
a = 23  b = 78.000000   c = 9

Enter values of a, b and c
8  A  5.25

Error in input.
Enter values of a, b and c
Y  12  67

Error in input.
Enter values of a, b and c
15.75  23  X

a = 15  b = 0.750000   c = 2
```

Fig.4.8 *Detection of errors in* ***scanf*** *input*

Example 4.9

The program in Fig.4.9 illustrates the output of integer numbers under various formats.

PRINTING OF **INTEGER** NUMBERS

Program:

```
main()
{
    int m = 12345;
    long n = 987654;

    printf("%d\n",m);
    printf("%10d\n",m);
    printf("%010d\n",m);
    printf("%-10d\n",m);
    printf("%10ld\n",n);
    printf("%10ld\n",-n);
}
```

Output

```
12345
     12345
0000012345
12345
    987654
   -987654
```

Fig.4.9 *Formatted output of integers*

Example 4.10
All the options of printing a real number are illustrated in Fig.4.10.

PRINTING OF **REAL NUMBERS**

***Program*:**

```
main()
{
    float y = 98.7654;

    printf("%7.4f\n", y);
    printf("%f\n", y);
    printf("%7.2f\n", y);
    printf("%-7.2f\n", y);
    printf("%07.2f\n", y);
    printf("%*.*f", 7, 2, y);
```

```
        printf("\n");
        printf("%10.2e\n", y);
        printf("%12.4e\n", -y);
        printf("%-10.2e\n", y);
        printf("%e\n", y);
    }
```

Output

```
98.7654
98.765404
98.77
98.77
0098.77
98.77
9.88e+001
-9.8765e+001
9.88e+001
9.876540e+001
```

Fig.4.10 *Formatted output of **real** numbers*

Example 4.11

Printing of characters and strings is illustrated in Fig.4.11.

PRINTING OF CHARACTERS AND STRINGS

Program

```
    main()
    {
        char x = 'A';
        static char  name[20] = "ANIL KUMAR GUPTA";

        printf("OUTPUT OF CHARACTERS\n\n");
        printf("%c\n%3c\n%5c\n", x,x,x);
        printf("%3c\n%c\n", x,x);
        printf("\n");

        printf("OUTPUT OF STRINGS\n\n");
        printf("%s\n", name);
        printf("%20s\n", name);
        printf("%20.10s\n", name);
        printf("%.5s\n", name);
        printf("%-20.10s\n", name);
        printf("%5s\n", name);
    }
```

Output

```
OUTPUT OF CHARACTERS

 A
   A
     A
   A
 A

OUTPUT OF STRINGS

ANIL KUMAR GUPTA
    ANIL KUMAR GUPTA
          ANIL KUMAR
ANIL
ANIL KUMAR
ANIL KUMAR GUPTA
```

Fig.4.11 *Printing of characters and strings*