Ch-4

MANAGING INPUT & OUTPUT OPERATIONS

* Each 'C' program that uses a standard I/O statement must include the following:

```
#include <stdio.h>   (Standard I/O Header file)
```

This instruction tells the computer to search for a file named stdio.h and place its contents at this point in the program. The contents of the header file then become part of the source code when it is compiled.

Reading a Character

* The simplest I/O operation is to read a character from keyboard and write it onto the screen
* Reading a single character is done through the function getchar().

Syntax:

```
variable_name = getchar();
```

Eg:

```
char name;
name = getchar();
```

This will assign character 'H' to the variable 'name' when we press 'H' on the keyboard.

* The same thing can be done through scanf function.

Program Example

```
/* Use of getchar() function */
#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();
    if (answer == 'Y' || answer == 'y')
    printf("\n My name is \n");
    else
    printf("\n No \n");
}
```

Writing a Character - putchar()

* It is analogus to getchar() for writing characters one at a time on the screen

Syntax:

```
putchar(variable-name);
```

Eg:
```
answer = 'Y';
putchar (answer);
```

will display the character 'Y' on the screen.

Eg: `putchar('\n');`

will cause the cursor on the screen to move to the beginning of the next line.

NOTE: Character functions contained in the header File ctype.h and thus the following header file must be included in the program:

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

Eg:
```
isalpha(character)   // non-zero if character is alphabet
isdigit(character)   // non-zero if character is a number
```

/* Program that reads a character from keyboard and then prints in reverse case */

```
#include <stdio.h>
#include <ctype.h>
main()
{
    char alphabet;
    printf("Enter an alphabet");
    putchar('\n');
    alphabet = getchar();
    if (islower(alphabet));
      putchar(toupper(alphabet));
    else
      putchar(tolower(alphabet));
}
```

O/P:

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

Formatted Input

* Formatted input refers to an input data that has been arranged in a particular format

Eg: 15.27 123 John

Such data has to be read according to the data type. This is possible through scanf function

Syntax:

```
scanf("control string", arg1, arg2, ..., argn);
```

where control string: specifies the field format in which data is to be entered

arg1, arg2, ..., argn: address of locations where the data is stored

Inputting Integer Numbers

For reading an integer number,

```
%wd
```

w: integer that specifies the field width of the no. to be read

EX: scanf("%2d %5d", &num1, &num2);

Data: 50, 31426

Suppose the input is as follows:

31426 50

num1 = 31

num2 = 426

Inputting Real Numbers

* scanf reads real numbers with %f for decimal point notation and exponential notation

Eg: scanf ("%f %f %f", &x, &y, &z);

with the input data

475.89 43.21E-1 678

x = 475.89

y = 4.321

z = 678.0

* for double, the specification should be %lf.

Reading mixed Data Types

* It is possible to input mixed data using scanf.
* In such case, it should be ensured that the input data items match the control specification in order and type.

Eg: scanf("%d %c %f %s", &count, &code, &ratio, name);

will read the data:

15 P 1.575 coffee

Points to Remember While Using Scanf

P86(B)

NOTE:

* scanf() is not capable of receiving multi-word strings into a single variable. Eg: "Good Morning" would be read as only Good..
* The moment a blank is typed after a word, scanf() assumes that the name being entered has ended.
* The solution to this problem is to use gets() function.
* It terminates when an Enter key is hit. So, spaces and tabs are acceptable as part of input string.

Formatted Output

* The general form of printf statement is:

```
printf("control string", arg1, arg2, ..., argn);
```

* The control string consists of three types of items:
  - Characters that will be printed on the screen
  - Format specifications that define the o/p format
  - Escape sequence characters such as \n, \t and \b.

Output of Integer Numbers

Format Specification for printing an integer:

```
%wd
```

where w specifies the minimum width for the o/p.

* If a number is greater than 'w', it will be printed in full
* The number is written right-justified and leading blanks will appear as necessary
* It is possible to make the printing left-justified by placing a minus sign after the '%' symbol
* It is also possible to pad with zeros the leading blanks by placing a '0' before 'w'

Format	Output
Eg: printf("%d", 9876)	9 8 7 6
printf("%6d", 9876)	_ _ 9 8 7 6
printf("%2d", 9876)	9 8 7 6
printf("%-6d", 9876)	9 8 7 6 _ _
printf("%06d", 9876)	0 0 9 8 7 6

Output of Real Numbers

* Format Specification

%w.pf

(Rounded to 'p' decimal places printed right-justified in 'w' columns)

w: minimum no. of positions that are to be used for display of value

p: no. of digits to be displayed after the decimal point (precision)

* The default precision is 6 decimal places

eg.

* Negative nos. will be printed with the minus sign.

* * real nos. in exponential notation is displayed with the following specification:

mantissa exponent

% w.pe

* Padding with leading zeros and printing with left justification is possible by introducing '0' or '-' before 'w'.

Eg: y = 98.7654

Format	Output
printf("%7.4f", y)	9 8 . 7 6 5 4
printf("%7.2f", y)	_ _ 9 8 . 7 7
printf("%-7.2f", y)	9 8 . 7 7 _ _
printf("%.f", y)	9 8 . 7 6 5 4
printf("%10.2e", y) 98.7654 ≃ 9.88 x 10^1	_ _ 9 . 8 8 e + 0 1
printf("%11.4e", -y) 98.7654 ≃ 9.8765 x 10^1	- 9 . 8 7 6 5 e + 0 1
printf("%-10.2e", y) 9.88 x 10^1	9 . 8 8 e + 0 1 _ _
printf("%e", y)	9 . 8 7 6 5 4 e + 0 1

Printing of a single character

* A single character can be displayed using the format:

```
%wc
```

* The character will be displayed right-justified in the field of 'w' columns
* Default value of w is 1.
* The display is made left-justified by placing a minus sign before the integer w.

Printing of strings

* Format specification for outputting strings:

```
%w.ps
```

where w: Field width for display
p: it specifies only the first p characters of the string are to be displayed

* Display is right justified.

Ex: String = "NEW DELHI 110001" (16 characters including blanks)

Specification

Output

Specification	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
%0s, %.5s	N	E	W		D	E	L	H	I		1	1	0	0	0	1				
%.20s				N	E	W		D	E	L	H	I		1	1	0	0	0	1	
%20.10s										N	E	W		D	E	L	H	I		
%.5s	N	E	W		D															
%-20.10s	N	E	W		D	E	L	H	I											