CHAPTER

MANAGING INPUT AND OUTPUT FUNCTIONS

Reading data, processing it, and writing the processed data known as information or result of a program, are the essential functions of a program. **C** provides a number of macros and functions to enable the programmer to effectively carry out input and output operations.

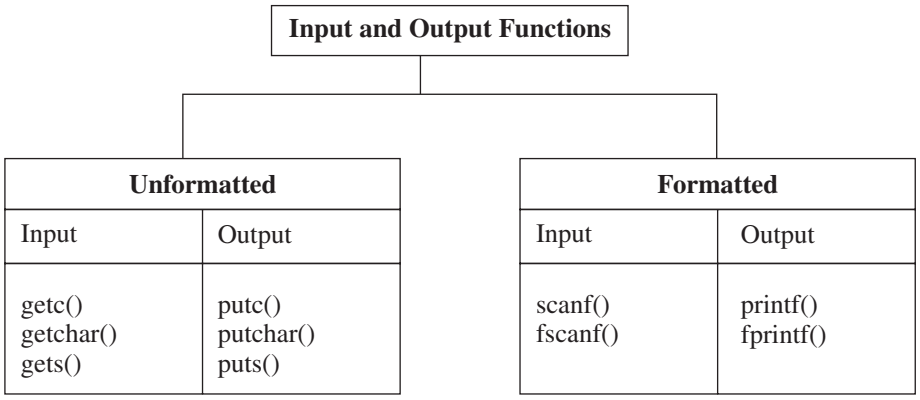
The data is entered by the user through the standard input device (i.e., keyboard) and processed data is displayed on the standard output device (i.e., monitor).

9.1 INPUT & OUTPUT FUNCTIONS_

The input and output functions are used to provide input to the program and display the result to the user respectively. These functions are available in the header file.

There are two types of input/output functions. They are:

1. Formatted I/O Functions
2. Unformatted I/O Functions



The formatted I/O functions allow programmers to specify the type of data and the way in which it should be read in or written out. On the other hand, unformatted I/O functions do not specify the type of data and the way it should be read or written.

Most of the input/output functions are provided as part of the C standard library which is accessed through the header file called stdio.h. The stdio.h is an abbreviation for standard input-output header file. This header file is included in the program using the preprocessor directive, #include<stdio.h>. It tells compiler to place contents of header file in the program. After compilation, the contents of the header file become a part of the source code.

9.2 FORMATTED INPUT FUNCTION

9.2.1 scanf() function

Input data can be entered using the scanf() function which is available in the standard input & output header file.0

The general syntax for the scanf() function is as follows:

scanf("control string",address of the variable);

The arguments for scanf function consists of a control string (in quotes) and address(es) of memory location(s) where the values of input variable(s) should be stored. Like printf function, the scanf function uses format specifiers preceded by a % sign to specify what type of data is to be read.

Format Specifier	Type of data item read
%c	Single Character
%d	Signed decimal integer
%e	Floating point number
%f	Floating point number
%g	Floating point number
%h	Short integer
%i	Integer in either decimal, octal or hexadecimal format
%o	Octal string
%s	Character string
%u	Unsigned decimal integer
%x	Hexadecimal integer

EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
    char name[10];
    int rollno;
    float pr_marks;
    scanf("%s%d%f",name,&rollno,&pr_marks);
}
```

where, & is the address of operator which fetches the address of the variables.

Rules to be followed:

1. The control string must be enclosed within double quotation.
2. The variables can be separated by using comma.
3. It should be terminated by a semicolon.
4. No empty space should be given between two control strings.
5. All addresses must be separated by commas.
6. For every data item to be read, there must be a format specifier corresponding to its data type.
7. The format specifiers are matched from left to right.
8. Multiple format specifiers are allowed in a control string. In such cases, they may be contiguous or separated by blank spaces.
9. Each format specifier must be preceded by % sign.
10. Each variable in the address list must be preceded by an ampersand except string variable.

1. Integer Input: Like printf function, scanf function also uses modifiers to specify the field width of the input variable. If the input number has more digits than the specified field width then extra digits are not stored. For example,

```
int a;
scanf("%d",&a);
```

2. Floating Point Input: The format specifier for floating point number is %f. It reads both, number specified using decimal notation and using exponential notation.

```
float avg;
scanf("%f",&avg);
```

3. Character Input: Single character is read using %c format specifier in scanf function. A sequence of characters is read using %wc or %ws specifier. **w** indicates field width and it is optional.

```
char op;
scanf( "%c", &op);
```

9.2.2 fscanf() function

This function is used to read the data from a file.

EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
    FILE *fp;
    char buff[255];
    fp = fopen("/tmp/test.txt", "r");
    fscanf(fp, "%s", buff);
    printf("1 : %s\n", buff );
}
```

9.3 FORMATTED OUTPUT FUNCTION

9.3.1 printf() function

printf() function is used to display the output or result on the output device. This function can display any combination of numerical values, single characters and strings.

The general syntax is as follows:

Printf("control string", variable name);

Example:

```
int a=10;
printf("%d",a);
```

The printf statement consists of two parts: the function name and the function arguments, enclosed in parentheses. The arguments for printf function consist of a control string (in quotes) and a print list (the variable or variables whose values to be displayed).

The %d in the printf statement is known as placeholder or conversion character or format code or format specifier. At the time of display, the value of the specified variable is substituted at the place of the placeholder.

The printf function uses different placeholders to display different types of data.

Rules for writing printf() function:

1. The variable and the control string should match.
2. The control string must be enclosed within double quotation.
3. The data items must be included in the print list and they must be separated by comma.
4. For every data item to be displayed, there must be a placeholder corresponding to its data type.
5. Print list is not enclosed within double quotes.
6. The format string and the print list must be separated by comma.

SAMPLE PROGRAM USING printf() FUNCTIONS

```
#include<stdio.h>
void main()
{
     int a;
     printf("\nEnter the value for a');
     scanf("%d",&a);
     printf("\n The value entered is %d",a);
}
```

Output

```
Enter the value for a 7
The value entered is 7
```

Formatting Integer Output

The integer numbers are displayed on the monitor using %d placeholder. Modifiers can be added to placeholders to specify the minimum field width and left justification. All outputs are right justified by default. It can be left justified by putting a minus sign directly after the %.

In the below examples, if the number width in digits is less than the minimum field width, the empty spaces are filled with spaces.

Print Statement	Output(assume x=1234)
printf("%d",x);	1 2 3 4
printf("%6d",x);	Right Justified ␣ ␣ 1 2 3 4
printf("%-6d",x);	Left Justified 1 2 3 4 ␣ ␣

However, if the number width is greater than the minimum field width, the number is printed in full, overriding the minimum field width specification.

It is also possible to pad the empty places with zeros. To pad zeros, a zero is to be placed before the minimum field width specifier.

Print Statement	Output(assume x=1234)
printf("%3d",x);	1 2 3 4 Overriding the minimum field width
printf("%06d",x);	0 0 1 2 3 4 Padding with zeros

Formatting Floating Point Output

printf Statement	Output(assyme y=3456.12065)
printf("%f",y);	Default precision is 6 3 4 5 6 · 1 2 0 6 5 0 6 decimal places
printf("%7.2f",y);	7 characters wide 3 4 5 6 · 1 2 2 decimal places
printf("%9.2f",y);	9 characters wide 3 4 5 6 · 1 2 Right Justified 2 decimal places
printf("%-9.2f",y);	Left Justified 3 4 5 6 · 1 2

In case of floating point numbers, placeholders can accept modifiers that specify the minimum field width, precision and left justification. To add a modifier, a decimal point is placed followed by the precision after the field width specifier. For e and f formats, the precision modifier determines the number of decimal places to be displayed. For example, %8.4f will display a number at least 8 characters wide including decimal point with four decimal places.

Formatting String Output

When the precision modifier is applied to strings, the number preceding period (decimal point) specifies the minimum field width and the number following the period specifies the number of characters of the string to be displayed. For example %8.5s will display a string that will be at least eight characters long; however only first five characters from the string are displayed with remaining blank characters.

Important points related to formatting string display:

- ❖ When minimum field width is specified without negative sign, display is right justified.
- ❖ When only minimum field width is specified and it is less than the actual string length, the minimum field width is overridden and the complete string is displayed.
- ❖ Negative sign before the minimum field width specifies the left justified display.

EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
      printf(":%s:\n", "Hello, world!");
      printf(":%15s:\n", "Hello, world!");
      printf(":%.10s:\n", "Hello, world!");
      printf(":%-10s:\n", "Hello, world!");
      printf(":%-15s:\n", "Hello, world!");
      printf(":%.15s:\n", "Hello, world!");
      printf(":%15.10s:\n", "Hello, world!");
      printf(":%-15.10s:\n", "Hello, world!");
}
```

Output

```
:Hello, world!:
:Hello, world!:
:Hello, wor:
:Hello, world!:
:Hello, world!:
:Hello, world!:
:Hello, wor:
:Hello, wor:
```

- ❖ The printf(":%s:\n", "Hello, world!"); statement prints the string (nothing special happens.)
- ❖ The printf(":%15s:\n", "Hello, world!"); statement prints the string, but print 15 characters. If the string is smaller, the "empty" positions will be filled with "whitespace."
- ❖ The printf(":%.10s:\n", "Hello, world!"); statement prints the string, but print only 10 characters of the string.
- ❖ The printf(":%-10s:\n", "Hello, world!"); statement prints the string, but prints at least 10 characters. If the string is smaller "whitespace" is added at the end. (See next example.)

- The printf(":%-15s:\n", "Hello, world!"); statement prints the string, but prints at least 15 characters. The string in this case is shorter than the defined 15 character, thus "whitespace" is added at the end (defined by the minus sign.)
- The printf(":%.15s:\n", "Hello, world!"); statement prints the string, but print only 15 characters of the string. In this case the string is shorter than 15, thus the whole string is printed.
- The printf(":%15.10s:\n", "Hello, world!"); statement prints the string, but print 15 characters.
- If the string is smaller, the "empty" positions will be filled with "whitespace." But it will only print a maximum of 10 characters, thus only part of new string (old string plus the whitespace positions) is printed.
- The printf(":%-15.10s:\n", "Hello, world!"); statement prints the string, but it does the exact same thing as the previous statement, except the "whitespace" which is added at the end.

Enhancing the Readability of Output

The readability of the output is enhanced using two specifiers.

- \t: It is a 'tab' character which inserts four/eight blank spaces between the two fields.
- \n: It is newline character which displays the message or variable values following it on the next line.

The backslash symbol (\) is known as escape character and '\t' and '\n' are called escape sequences. There are few more escape sequences.

Escape Sequences	Function
\n	**Newline:** Displays the message or variable values following it on the next line
\t	**Tab:** Moves cursor to the next tab
\b	**Backspace:** Moves cursor one position to the left of its current position
\f	**Form Feed:** Advances the computer stationary attached to the printer to the top of next page.
\'	**Single Quote:** Displays single quote
\\	**Backslash:** Displays backslash
\"	**Double Quote:** Displays double quote
\r	**Carriage Return:** Takes the cursor to the beginning of the line in which it is currently placed.
\a	**Alert:** Alerts the user by sounding the speaker inside the computer.

9.3.2 fprintf() Function

This function is used to write data into a file.

EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
FILE *fp;
fp = fopen("/tmp/test.txt", "w+");
fprintf(fp, "This is testing for fprintf...\n");
fputs("This is testing for fputs...\n", fp);
fclose(fp);
}
```

9.4 UNFORMATTED INPUT AND OUTPUT FUNCTIONS

Some of the unformatted input and output functions are given below:

Input	Output
getc()	putc()
getchar()	putchar()
gets()	puts()

9.4.1 Unformatted Input Functions (Single Character Input Function)

getc(): It is used to get a single character from the standard input to a character variable. The getc() function is often used in file processing.

Syntax

character variable = getc();

Example

```
char c;
c = getc();
```

getchar(): It reads a single character from the standard input device. This function does not require any arguments, though a pair of empty parentheses must follow the word getchar as syntax. It returns a single character from a standard input device and it can be assigned to predeclared character variable.

Syntax

character variable = getchar();

Example

```
char x;
x= getchar();
```

The difference between getc() and getchar() is getc() can read from any input stream, but getchar() reads from standard input. So getchar() is equivalent to getc(stdin).

String Input Function

gets(): It is used to read the string (string is a group of character) from the standard input device and stores them at the address pointed by its argument. It reads the string of characters until the Enter or Return Key is pressed. A null terminator is pressed at the end.

Syntax

gets(char type of array variable);

Example

```
char c[10];
gets(c);
```

gets() function does not perform any boundary checks on the array. Thus it is possible for the user to enter more characters than the array can hold.

9.4.2 Unformatted Output Functions

Single Character Output Function

The unformatted output function such as putchar() and puts() are concerned with displaying a single character or string of characters on the display monitor. They are included in the stdio.h header. Therefore, the C program that uses these functions should exclusively have the #include<stdio.h> statement as a preprocessor directive.

putc(): It is used to display a single character in a character variable to standard output device. The putc() function is often used in file processing.

Syntax

putc(character variable);

Example

```
char c;
putc(c);
```

putchar(): It displays one character at a time on the standard output device. The character to be displayed is of type char.

Syntax

putchar(ch_var);

where, ch_var is a previously declared character variable.

EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
      char x='A';
      putchar(x);
}
```

Output

```
A
```

String Output Function

puts(): The puts function displays a string of characters on the monitor followed by a newline.

Syntax

puts(string);

where, string is a sequence of characters or previously declared array of characters.

EXAMPLE PROGRAM

```
#include<stdio.h>
main()
{
char str[40]=" hai welcome"
      puts("sequence of characters");
      puts(str);
      }
```

Output

```
Sequence of characters
hai welcome
```

Sample program for single input & output functions getchar() & putchar()

```
#include<stdio.h>
main()
{
char x;
printf("Enter any alphabet either in Lower or Uppercase….");
x=getchar();
if(islower(x))
putchar(toupper(x));
else
putchar(tolower(x));
}
```

Output

```
1. Enter any alphabet either in Lower or Uppercase……s
S
2. Enter any alphabet either in Lower or Uppercase……M
m
```

Sample program for gets() & puts() functions

```
#include<stdio.h>
main()
{
char x[40];
puts("Enter String….");
gets(x);
puts("Print the name….");
puts(x);
}
```

Output

```
Enter String…… welcome
Print the name……welcome
```