Input - Output

I/O functions

- For data input and output, C provides a collection of library functions such as:
  - getchar
  - putchar
  - gets
  - Puts
  - scanf
  - printf

- C always treats all input-output data, regardless of where they originate or where they go, as a stream of characters.
- A stream of characters or text stream, is a sequence of characters divided into lines.
- Each line consists of various characters followed by a newline character (\n).
- All input-output functions in C conform to this model.

- The operating system makes the input and output devices available to a C program as if these devices were files.

- The standard input-output devices or the associated files or text streams, are referred as -
  - **stdin -** standard input file, normally connected to the keyboard
  - **stdout -** standard output file, normally connected to the screen/console
  - **stderr -** standard error display device file, normally connected to the screen/console

- In order to be able to use the above mentioned input-output functions in your C program, you must begin each C program with a pre-processor directive to include these standard library functions.

- This can be done with a line entry -

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

- The input-output functions, fall into two categories -
  - formatted read/write functions
  - non-formatted read/write functions.

getchar()

- This is a single character input function. getchar() reads a single character from stdin - the standard input data stream, viz. the file associated with the standard input device, which is normally the keyboard.

Example

```
#include <stdio.h>
main()
{
    char c;
    printf("\nContinue(Y/N)?"); //Display to prompt the user to key in his response
    c = getchar();
    if(c=='Y')
    {
        - do something -
    }
    else
    {
        - do something else -
    }
}
```

- Note that when you input a single character for the C program to read, you must indicate end of data stream or end of input by pressing the return/enter key after entering your response character.
- When you simply press the enter key without inputting any value first, the getchar function will return the symbolic constant EOF (for end of file) which typically stores the value -1, when there is no input.
- The EOF is an integer constant defined in stdio.h header file.

putchar()

- This is a single character output function. putchar() writes a single character to stdout - the standard output data stream, viz. the file associated with the standard output device, which is normally the console.

Example

```
#include <stdio.h>
main()
{
    char c;
    printf("\nContinue(Y/N)?");
    c = getchar();
    putchar(c);
}
```

- Note that in the above example code, you can declare the variable c either as a char or as an int, since in C, a single character is internally represented as it's corresponding numeric ASCII code.

gets()

- The standard library function gets accepts input in the form of a string.
- The character string may even include whitespace characters.
- Each call to gets will read all the characters from the input steam until an end of line character is encountered.
- The end of line character is represented as \n and gets generated when you press the enter key.
- gets assigns the read string to the variable that is passed as its parameters.
- gets assigns NULL when an error occurs.

puts()

- The standard library function puts sends the passed string to stdout.
- After the output, puts sends out a carriage return and a line feed character.
- This takes the cursor to the next line automatically.

example

```
#include <stdio.h>
main()
{
    char name[71]; //String variable declaration
    printf("\nEnter your Name: ");
    gets(name);
    puts(name);
}
```

scanf()

- The scanf function is used to read formatted input data.
- The format in which input data is to be provided is specified by the scanf function itself as it's first parameter.
- The scanf function is written as -

```
scanf(<control string>, &address1, &address2, . . . , &addressn)
```

```
scanf(<control string>, &address1, &address2, . . . , &addressn)
```

- the first parameter <control string> contains a list of format specifiers indicating the format and type of data to be read.
- The remaining parameters - &address1, &address2, ..., &addressn are addresses of the variables where the read data will be stored.
- scanf reads the input data as per the format specifiers and stores them (i.e., assigns them) to the corresponding addresses.
- An & is pre-fixed to the variable name to denote its address.
- Note that there must be the same number of format specifiers and addresses as there are input data.

`scanf("%d %f",&x,&y)`

- the first argument is a string that contains two format specifiers - %d and %f, the first format specifier (%d) is for argument &x and the second one (%f) is for the argument &y.
- So, two pieces of input data will be read - the first piece is treated according to %d and the second one is treated according to %f.
- Format specifiers in the <control string> are always specified for the remaining arguments, in order, from left to right.

- Keep in mind that scanf() treats all white-space characters as de-limiters to separate out one data input from the other.
- So, while entering your data inputs, you can separate them out with a blank space or a tab space character or a newline character.

Syntax of format specifier

- Each format specifier begins with the percentage character (%) followed by a conversion character which indicates the type of the corresponding data item.
- Similar control string is also used with the printf function to be discussed later.
- The meaning of the conversion characters for input/output is given in the next table, which apply to both scanf and printf functions.

Format Specifier	Input Data (scanf)	Output Data (printf)
%c	reads a single character	single character
%d	reads a numeric value as signed int. Treats the input data as a decimal number, i.e. a number in the base 10 number system	signed decimal integer
%i	can read data value provided either as a decimal int, hexadecimal int or octal int	print as a signed decimal integer.
%o	reads data value as an octal number	prints data as an octal integer without leading zero.
%x	reads data value as a hexadecimal number	prints data as a hexadecimal integer without leading **0x**.
%u	reads data value as an unsigned integer	prints as an unsigned integer.
%h	reads data value as a short integer	N/A
%f	reads data as a floating point value without the exponent (i.e of the form [-]dddd.dddd)	prints as a floating point value (without exponent)
%e	reads data as a floating point value. The input data can also have an exponent part (eg: -1.2e+5, which means -1.2 x 10^5)	prints as a floating point value in exponent form
%E	This is same as %e, except that in this case the exponent is represented with a capital E (eg: -1.2E+5)	
%g	reads data as a floating point value in either of the format specified for %f or %e	prints as a floating point value. Depending upon the magnitude of the value, it will be displayed either as e-type or f-type conversion. If the value does not have any fractional part, the trailing decimal point is not printed.
%G	This is same as %g, except that in this case if an exponent is there, it is represented with a capital E.	
%s	reads data as a string of characters. Reading terminates when a whitespace character is encountered. The read string of characters is appended with a null character **'\\0'** at the end.	prints a character string.
%[]	reads data as a string which may even include whitespace characters as well	N/A

Example

```
#include <stdio.h>
main()
{
    char name[40];
    int rollno;
    char grade;
    scanf("%s %d %f", name, &rollno, &grade);
}
```

- A sample data input could be - Ady 12345 A
- The following data input is incorrect, and will give error – Budi Winardi 12345 A

- Note: The variable name being a string (an array of characters) is not preceded by ‘&’, but the numerical variables rollno and grade are preceded by ‘&’.
- This is because, in case of an array type variable, the variable name itself represents its address in the memory.
- Also note that while providing data values they must be separated by whitespace character, blank space, tab space, new line character (enter key), etc. which act as de-limiters of individual data items.

- Avoid using a new line character (i.e. enter key) to de-limit input values as it can create problems when one or more of the data items is a single character.
- For instance, in the example before, if you typed Ady, then pressed enter key, then typed 12345, pressed enter key, and then typed A and pressed enter key, then while name and rollno would get read correctly, grade will take in the \n character (corresponding to the enter key following value 12345) and so A will not get read.

- The format specifier can be additionally pre-fixed with a number to indicate the width of the associated data field.
- See example below:

```
#include <stdio.h>
main()
{
    int x, y, z;
    float u, v, w;
    char a, b, c;
    scanf("%3d %4f %c", &x, &u, &c);
}
```

- If the data items provided are 105 40.36 A, then the variables get values x = 105, u = 40.3, c = 6.
- Note that the character A is ignored.
- This is because the field width for u is 4 which can accommodate 40.3 (the decimal point is also counted).
- The next character in the input steam is character 6, which gets assigned to c and A is left un-read.

printf()

- The printf function is used to output data onto the standard output device. In general, the printf function is written as

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

- where the <control string> refers to a string containing required formatting information as in scanf, and arg1, arg2, ..., argn are individual data variables whose values are to be printed.
- However, unlike scanf, these data variable names are not preceded by the & symbol.
- This is because printf is expected to only output the values of these variables and not their addresses.

Examples

```
#include <stdio.h>
main()
{
    printf("Our first program in C\n");
}
```

- Output: Our first program in C

```
#include <stdio.h>
main()
{
    int n;
    n = 25;
    printf("The value of n = %d\n", n);
    //note the usage of format specifier %d
}
```

- Output: The value of n = 25

```
#include <stdio.h>
main()
{
    int n;
    printf("Give an integer: ");
    scanf("%d",&n); //note usage of & before n
    fflush(stdin); //flushes the standard input buffer
    printf("Integer read is %d\n", n);
    //note the absence of & symbol before variable n
}
```

```
#include <stdio.h>
main()
{
    float x = 2.0;
    int y = 4;
    char c = 'A';
    printf("%f%d%c\n", x, y, c);
}
```

- Output: 2.0000004A

- As you notice, the printf statement prints the data values x, y, and c all sticking to each other, followed by a blank line (due to \n at the end of the control string).
- For clarity of data output, you may separate the format specifiers with blank space or comma (,). So, a better way of writing would be -

```
printf("%f, %d, %c\n", x, y, c); or
printf("%f %d %c\n",x,y,z);
```

```
#include <stdio.h>
main()
{
    float x = 453.7869;
    int y = 243;
    char c = 'B';
    printf("%6.2f %2d %c\n", x, y, c);
}
```

- Output: 453.79 243 B

- Note that due to the precision specification the floating point variable x has been rounded to fit the specified field width.
- Also, note that the field width of 2 specified for y is too small as compared to the specified field width supported for integer data items (which is normally 8 digits).
- When such a precision is specified, it is ignored and the entire integer value is displayed.

- Having developed an elementary understanding of the control string in printf, let us now define the format specifier for printf in full. A full format of the format specifier in printf would be like -

`%[-][width][flags]format` [ ] implies that the specifiers are optional.

where:

% : denotes the beginning of a format specifier

- : indicates that data is to be displayed as left-justified. If the - is missing, data display is right-justified.

width : the width of field or number of spaces to use for displaying

flags : precision of output to be displayed

format : the format specifier itself

```
#include <stdio.h>
main()
{
    float x = 123.456;
    printf("/%f/\n", x);
    printf("/%15f/\n", x);
    printf("/%-15f/\n", x);
    printf("/%2f/\n", x);
    printf("/%.3f/\n", x);
    printf("/%.1f/\n", x);
}
```

Output:

- /123.456000/ when no precision is specified, the default precision is 6 places after decimal
- / 123.456000/ 5 blank spaces introduced before the output
- /123.456000 / 5 blank spaces introduced after the output
- /123.456000/ the width specifier 2 is ignored
- /123.456/ decimal places precision specified is 3
- /123.5/ note the rounding

```
#include <stdio.h>
main()
{
    printf("/%5s/\n", "Hello World!");
    printf("/%15s/\n", "Hello World!");
    printf("/%15.5s/\n", "Hello World!");
}
```

Output:

- /Hello World!/ : width specifier ignored
- / Hello World!/ : right-justified, pre-filled with 3 blank spaces
- / Hello/: only first 5 characters displayed. 10 blank spaces introduced before

```
#include <stdio.h>
main()
{
    char stud_name[21];
    printf("Enter your name: ");
    scanf("%s", stud_name); //Note that a string variable is not prefixed by & symbol
    ffiush(stdin);
    printf("%-20s", stud_name); //A '-' before 20s implies display is left justified
}
```

- In the above example if the name is entered with embedded space then the variable stud_name will be assigned all characters until the space, since space will terminate the input stream.
- So, if input given is - Satyen Bose, stud_name gets Satyen only.

```c
#include <stdio.h>
main()
{
    char in_string[80];
    char out_string[80] = "IIT-Dedicated to the Service of the Nation";

    //Special scanf cases
    printf("Enter a string: ");
    scanf("%[^ \n]s", in_string); //Statement A
    ffiush(stdin);
    getchar(); //get rid of the \n character (enter key) after first string is input
    printf("Enter another string: ");
    scanf("%[ABCEFGHIJKLMNOPQRSTUVWXYZ ]s", in_string); //Statement B
    ffiush(stdin);

    //printf cases
    printf("/%s/", out_string);
    printf("/%20s/", out_string);
    printf("/%10.3s/", out_string);
    printf("/%-10.3s/", out_string);
    printf("/%.3s/", out_string);
}
```

- In the above program, the statement A will allow input of any character except the newline character (\n).
- The ^ symbol before '\n' indicates that '\n' is not allowed - all other characters are allowed.
- The input stream will be terminated as soon as the user presses the enter key.
- Most importantly, the user can also input embedded spaces.

- The statement B indicates a list of only those characters that are allowed.
- It will allow input of capital alphabets and spaces only.
- A preceding ^ in the format specification would have restricted the input to the characters other than specified within the square brackets.

Q&A