Rajmani Kumar,
Lecturer, Dept. of BCA
S.U.College, Hilsa (Nalanda)
Patliputra University, Patna

BCA-2nd Year **Paper-III**

INPUT-OUTPUT IN C

When we are saying **Input** that means we feed some data into program. This can be given in the form of file or from command line. C programming language provides a set of built-in functions to read given input and feed it to the program as per requirement.

When we are saying **Output** that means to display some data on screen, printer or in any file. C programming language provides a set of built-in functions to output the data on the computer screen.

Functions *printf()* and *scanf()* are the most commonly used to display out and take input respectively. Let us consider an example:

```
#include <stdio.h> //This is needed to run printf() function.
int main()
{
printf("C Programming"); //displays the content inside quotation return 0;
}
```

Output:

```
C Programming
```

Explanation:

- Every program starts from *main()* function.
- *printf()* is a library function to display output which only works if *#include<stdio.h>*is included at the beginning.
- Here, *stdio.h* is a header file (standard input output header file) and *#include* is command to paste the code from the header file when necessary. When compiler encounters *printf()*function and doesn't find *stdio.h* header file, compiler shows error.
- *return 0*; indicates the successful execution of the program.

Input- Output of integers in C

```
#include<stdio.h>
int main()
{
int c=5;
printf("Number=%d",c);
return 0;
}
```

Output

```
Number=5
```

Inside quotation of *printf()* there, is a conversion format string "%d" (for integer). If this conversion format string matches with remaining argument, i.e, *c* in this case, value of *c* is displayed.

```
#include<stdio.h>
int main()
{ int c;
printf("Enter a number\n"); scanf("%d",&c); printf("Number=%d",c);
return 0;
}
```

Output

```
Enter a number
4
Number=4
```

The *scanf()* function is used to take input from user. In this program, the user is asked an input and value is stored in variable *c*. Note the '&' sign before *c*. &c denotes the address of *c* and value is stored in that address.

<u>Input- Output of floats in C</u>

```
#include <stdio.h>
 int main()
{
float a;
printf("Enter value: "); scanf("%f",&a);
printf("Value=%f",a); //%f is used for floats instead of %d return 0;
}
```

Output

```
Enter value: 23.45
Value=23.450000
```

Conversion format string "%f" is used for floats to take input and to display floating value of a variable.

<u>Input - Output of characters and ASCII code</u>

```
#include <stdio.h>
int main()
{
char var1;
printf("Enter character: "); scanf("%c",&var1); printf("You entered %c.",var1);
return 0;
}
```

Output

```
Enter character: g
You entered g.
```

Conversion format string "%c" is used in case of characters.

ASCII code

When character is typed in the above program, the character itself is not recorded a numeric value (ASCII value) is stored. And when we displayed that value by using "%c", that character is displayed.

```
#include <stdio.h>
int main()
{
char var1;
printf("Enter character: "); scanf("%c",&var1);
printf("You entered %c.\n",var1);
/* \n prints the next line(performs work of enter). */ printf("ASCII value of
%d",var1);
return 0;
}
```

Output:

```
Enter character: g
103
```

When, *'g'* is entered, ASCII value 103 is stored instead of *g*.

You can display character if you know ASCII code only. This is shown by following example.

```
#include <stdio.h>
 int main()
{
int var1=69;
printf("Character of ASCII value 69: %c",var1); return 0;
}
```

Output

```
Character of ASCII value 69: E
```

The ASCII value of 'A' is 65, 'B' is 66 and so on to 'Z' is 90. Similarly ASCII value of 'a' is 97, 'b' is 98 and so on to 'z' is 122.

FORMATTED INPUT-OUTPUT

Data can be entered & displayed in a particular format. Through format specifications, better presentation of results can be obtained.

Variations in Output for integer & floats:

```
#include<stdio.h>
int main()
{
printf("Case 1:%6d\n",9876);
/* Prints the number right justified within 6 columns */ printf("Case 2:%3d\n",9876);
/* Prints the number to be right justified to 3 columns but, there are 4 digits so number is not right justified */
printf("Case 3:%.2f\n",987.6543);
/* Prints the number rounded to two decimal places */ printf("Case 4:%.f\n",987.6543);
/* Prints the number rounded to 0 decimal place, i.e, rounded to integer */ printf("Case 5:%e\n",987.6543);
/* Prints the number in exponential notation (scientific notation) */ return 0;
}
```

Output

```
Case 1: 9876
Case 2:9876
Case 3:987.65
Case 4:988
Case 5:9.876543e+002
```

Variations in Input for integer and floats:

```
#include <stdio.h> int main()
{
int a,b;
float c,d;
printf("Enter two intgers: "); /*Two integers can be taken from user at once as below*/
scanf("%d%d",&a,&b);
printf("Enter intger and floating point numbers: ");
/*Integer and floating point number can be taken at once from user as below*/ scanf("%d%f",&a,&c);
return 0;
}
```

Similarly, any number of inputs can be taken at once from user.

EXERCISE:

1. To print out a and b given below, which of the following printf() statement will you use?

```
#include<stdio.h> float a=3.14;
double b=3.14;
```

A. printf("%f %lf", a, b);

B. printf("%Lf %f", a, b);

C. printf("%Lf %Lf", a, b);

D. printf("%f %Lf", a, b);

2. To scan a and b given below, which of the following scanf() statement will you use?

```
#include<stdio.h>
 float a;
double b;
```

A. scanf("%f %f", &a, &b);

B. scanf("%Lf %Lf", &a, &b);

C. scanf("%f %Lf", &a, &b);

D. scanf("%f %lf", &a, &b);

3. For a typical program, the input is taken using.

A. scanf

B. Files

C. Command-line

D. None of the mentioned

4. What is the output of this C code?

```
#include <stdio.h>
int main()
{ int i = 10, j = 2;
printf("%d\n", printf("%d %d ", i, j));
}
```

A. Compile time error

B. 10 2 4

C. 10 2 2

D. 10 2 5

5. What is the output of this C code?

```
#include <stdio.h>
int main()
{
int i = 10, j = 3; printf("%d %d %d", i, j);
}
```

A. Compile time error

B. 10 3

C. 10 3 some garbage value

D. Undefined behavior

6. What is the output of this C code?

```
#include <stdio.h>
int main()
{ int i = 10, j = 3, k = 3;
printf("%d %d ", i, j, k);
}
```

A. Compile time error

B. 10 3 3

C. 10 3

D. 10 3 somegarbage value

7. The syntax to print a % using printf statement can be done by.

A. %

B. %

C. ‘%’

D. %%

8. What is the output of this C code?

```
#include <stdio.h> int main()
{ int n;
scanf("%d", n);
printf("%d\n", n);
return 0;
}
```

A. Compilation error

B. Undefined behavior

C. Whatever user types

D. Depends on the standard

9. What is the output of this C code?

```
#include <stdio.h>
int main()
{ short int i;
scanf("%hd", &i);
printf("%hd", i);
return 0;
}
```

A. Compilation error

B. Undefined behavior

C. Whatever user types

D. None of the mentioned

10. In a call to printf() function the format specifier %b can be used to print binary equivalent of an integer.

A. True

B. False

11. Point out the error in the program?

```
#include<stdio.h>
int main()
{
char ch;
int i;
scanf("%c", &i);
scanf("%d", &ch);
printf("%c %d", ch, i);
return 0;
}
```

A. Error: suspicious char to in conversion in scanf()

B. Error: we may not get input for second scanf() statement

C. No error

D. None of above

12. Which of the following is NOT a delimiter for an input in scanf?
A. Enter
B. Space
C. Tab
D. None of the mentioned

Libraries in C allow you to re-use important functions without the need for extra lines of code. In this lesson, you'll learn about the most common standard libraries used in C. Working code examples will be provided.

Standard Libraries

Imagine that you want to display output to the screen, but each time you want to display output you have to re-write 1500 lines of special code to make that happen. This is very inefficient and wasteful - not to mention the 1500 opportunities for bugs in your program!

Thankfully, the C programming language comes with a huge library of functions that you can use in your code, without having to re-write everything. You only need to include the library at the top of your code. When you use a command such as printf (to print to the screen), you're actually using a function that is in a standard C library.

Libraries are included in your code by referencing the **header file**. Think of it as the link to the library. To add the header file to your code, add the following at the top of your program (before any other statements):

```
#include <filename.h>
```

Note that there's no semicolon at the end of this line. One of the most common libraries used is the stdio library, which includes functions to write output to the screen and collect input from the user:

```
#include <stdio.h>
int main(void) {
/* Rest of program */
}
```

Now, let's take a look at some of the libraries, starting with stdio.h.

Stdio.h: Standard Input/Output

First, here's an example of some **stdio.h** code:

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

The standard input and output library is **stdio.h**, and you will find that you include this library in almost every program you write. It allows printing to the screen and collecting input from the user. The functions you will use the most include:

- **printf()** is output to the screen
- **scanf()** is read input from the screen
- **getchar()** is return characters typed on screen
- **putchar()** is output a single character to the screen
- **fopen()** is open a file, and
- **fclose()** is close a file

There are many times when we need to get user input from the screen and output information to the screen. Here is code that asks a user for their login name and then displays it on screen:

```c
#include <stdio.h>
int main(void) {
  char login[50];
  printf("Enter Name: ");
  scanf("%s", login);
  printf("\nWelcome %s\n", login);
}
```

When the code is compiled and executed, the output is as follows:

```
Enter Name: Jane
Welcome Jane
```

String.h: String Functions

First off, here's an example of **string.h**:

```c
#include <string.h>
```

The functions in **string.h** are useful for manipulating strings and characters:

- **strcat()**: concatenates (joins) one string to another string
- **strcpy()**: copies a string's content to another string, and
- **strcmp()**: compares the contents of two strings

For example, the code here concatenates the user's first name and last name into a single string:

```c
#include <stdio.h>
#include <string.h>
int main(void) {
  char fname[30] = "Jane";
  char lname[30] = "Doe";
  char full_name[61];
  strcat(full_name, fname);
  strcat(full_name, " " );
  strcat(full_name, lname);
  printf("%s ", full_name);
}
```

Output of this program is:

```
Jane Doe
```

Math.h: Math Functions

Now, here's an example of a math.h function:

```c
#include <math.h>
```

The functions in **math.h** usually require double data type as a parameter. That is, if you're trying to find the square root of 5.325, you'll need to write C code to calculate the square root on that number!

- **floor(double x)**: Returns the next-lowest value to x (or x, if x is an integer)

- **ceil(double x)**: Returns the smallest value greater than or equal to x
- **exp(double x)**: Returns the value of a number raised to the xth power
- **sqrt(double x)**: Square root of x

C supports all math functions, and we have only covered a few here. The following code shows an example of using the floor and ceiling functions in C:

```
double age = 15.3445;
/* Floor */
printf("%f", floor(age));
/* Ceiling */
printf("%f", ceil(age));
```

The output will display 15 and 16, respectively. The closest bottom integer to 15.3445 is 15; the next-highest is 16. This is useful in situations where you would need to work with a whole number for age, but you still need to store the value as is. For life insurance, you may charge rates based on either the floor or ceiling of a person's age.

Math.h also includes functions for trigonometry, fractions, and advanced mathematics.

Time.h:Date & Time Functions

Here’s an example of some time and date, or time.h functions:

```
#include <time.h>
```

There are a few date and time functions that are useful for your code. For example, if you need to write long files for a program, you can date and time-stamp that data by accessing the time library.