EXERCISE 2

Content:

1. printf scanf
2. Variables in C
3. Data Types in C
4. Keywords in C
5. C Identifiers
6. C Operators
7. C Comments
8. C Format Specifier
9. C Escape Sequence

I. printf scanf

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function

The printf() function is used for output. It prints the given statement to the console.

The syntax of printf() function is given below:

```
printf("format string",argument_list);
```

The format string can be %d (integer), %c (character), %s (string), %f (float) etc.

scanf() function

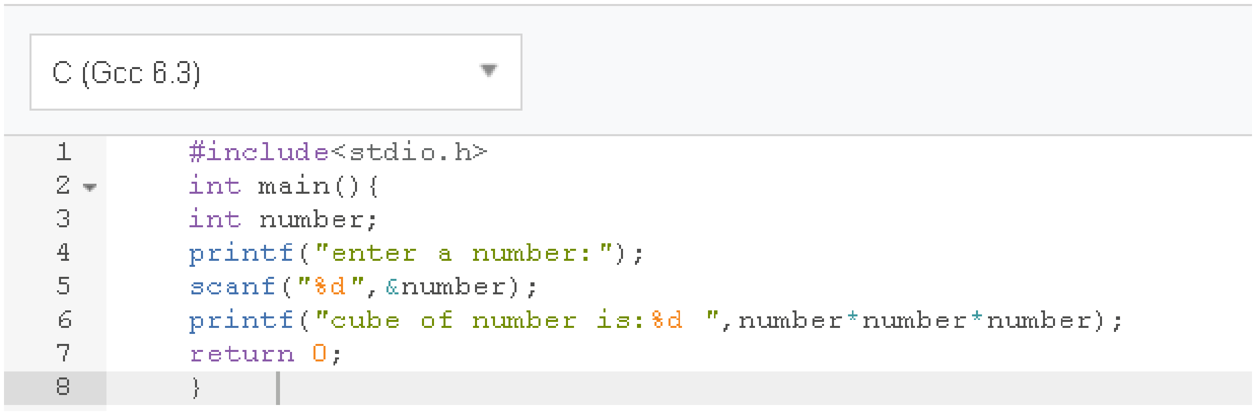
The scanf() function is used for input. It reads the input data from the console.

```
scanf("format string",argument_list);
```

LAB: Program to print cube of given number

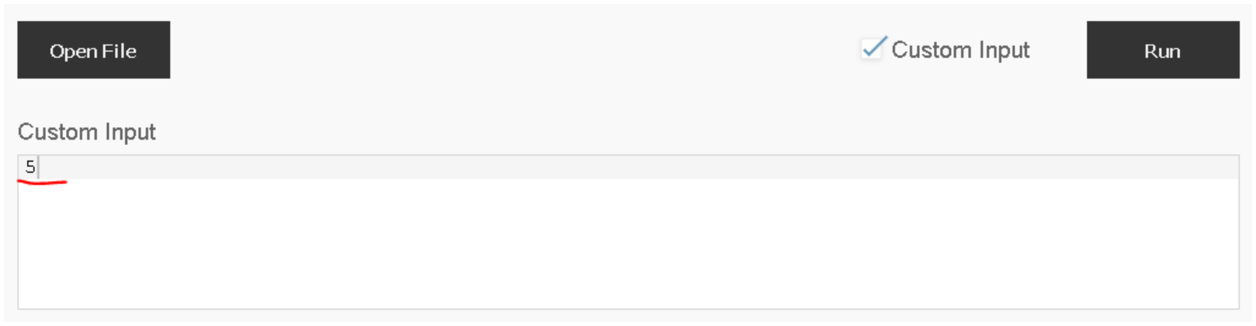
Let's see a simple example of C language that gets input from the user and prints the cube of the given number.

Step1: Type the following code in the CodeChef IDE:



```c
#include<stdio.h>
int main(){
int number;
printf("enter a number:");
scanf("%d",&number);
printf("cube of number is:%d ",number*number*number);
return 0;
}
```

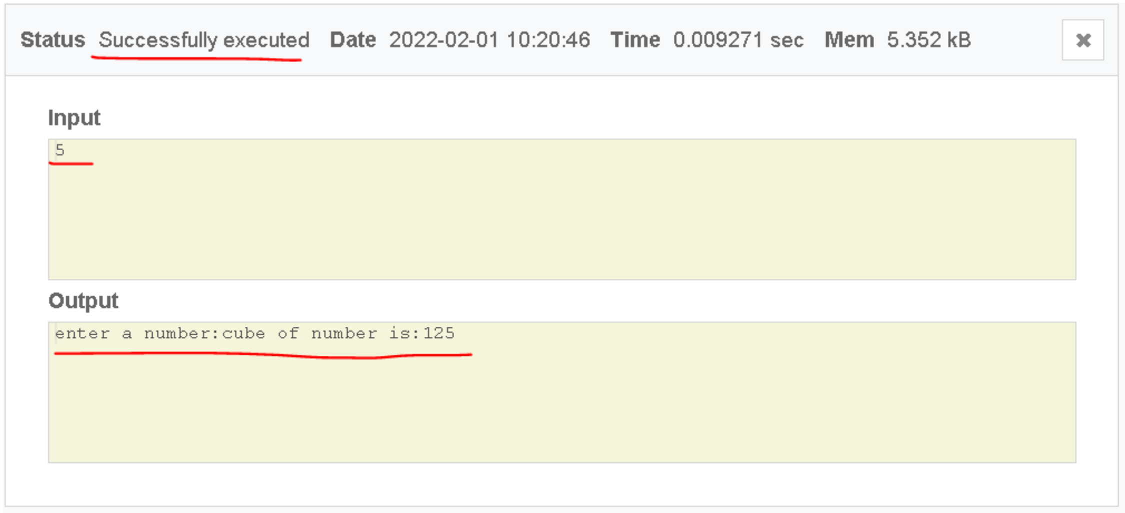
Step 2: Enter your number (for example 5) in Custom input:



```
5
```

Step 3: Run the program by Run button.

Step 4: Check the result:



Status Successfully executed **Date** 2022-02-01 10:20:46 **Time** 0.009271 sec **Mem** 5.352 kB

Input

```
5
```

Output

```
enter a number:cube of number is:125
```

If you enter number 5 the result has to be 125, because 5*5 is 25 and 25*5 is 125 (5*5*5=125).

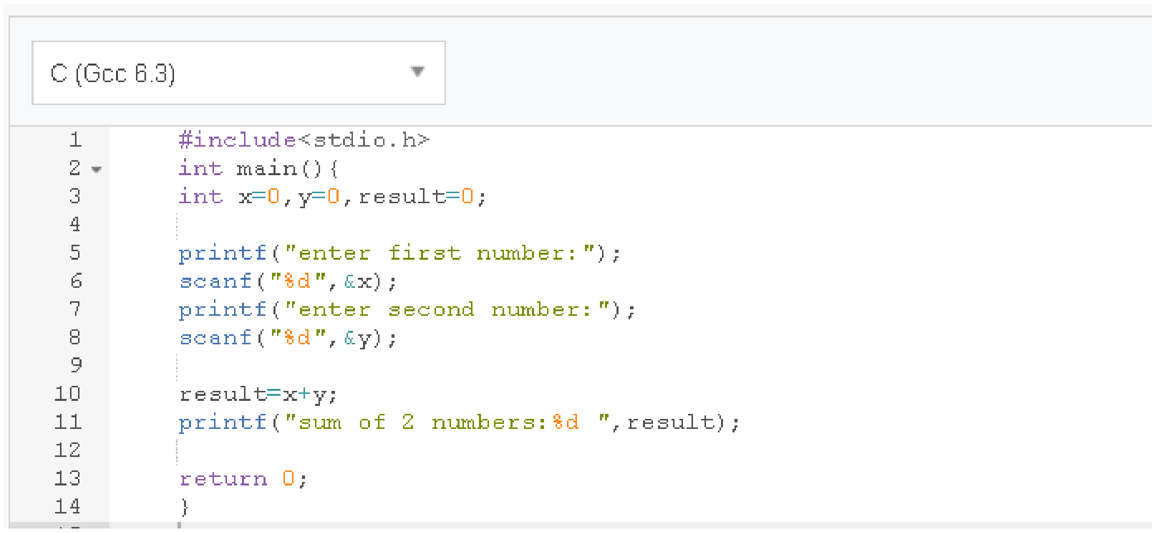
The **scanf("%d",&number)** statement reads integer number from the console (Custom input) and stores the given value in number variable.

The **printf("cube of number is:%d ",number*number*number)** statement prints the cube of number on the console (Output).

LAB: Program to print sum of 2 numbers

Let's see a simple example of input and output in C language that prints addition of 2 numbers.

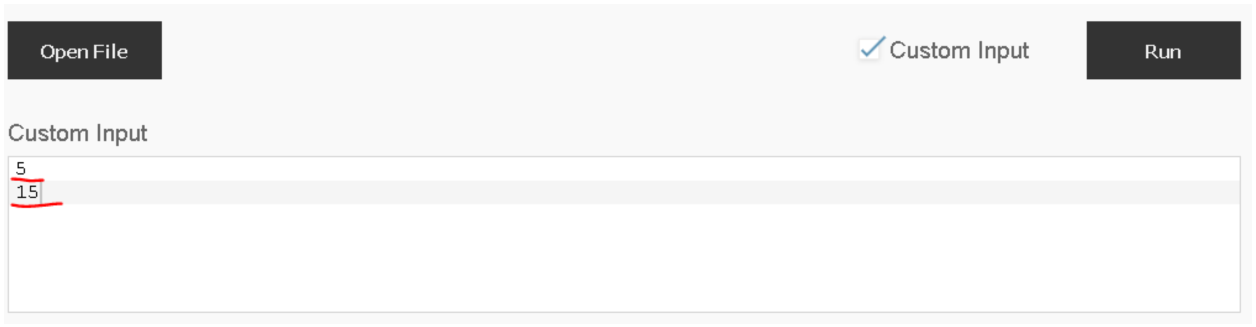
Step 1: Type the following code in the CodeChef IDE:



C (Gcc 6.3)

```
#include<stdio.h>
int main(){
int x=0,y=0,result=0;

printf("enter first number:");
scanf("%d",&x);
printf("enter second number:");
scanf("%d",&y);

result=x+y;
printf("sum of 2 numbers:%d ",result);

return 0;
}
```

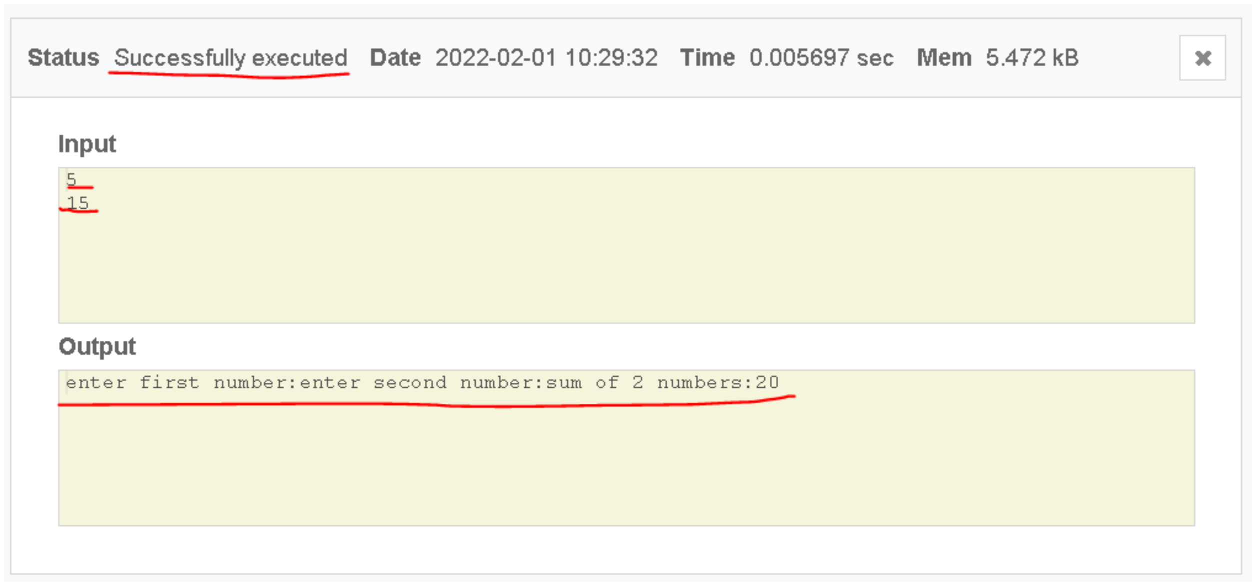
Step 2: Enter your 2 numbers (for example 5 and 15) in Custom input:



```
5
15
```

Step 3: Run the program by Run button.

Step 4: Check the result:


Status Successfully executed Date 2022-02-01 10:29:32 Time 0.005697 sec Mem 5.472 kB

Input

```
5
15
```

Output

```
enter first number:enter second number:sum of 2 numbers:20
```

If you enter 5 for first number and 15 for second number the result has to be 20, because 5 + 15 = 20.

II. Variables in C

A variable is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

```
type variable_list;
```

The example of declaring the variable is given below:

```
int a;
float b;
char c;
```

Here, **a, b, c** are variables. The **int, float, char** are the data types.

We can also provide values while declaring the variables as given below:

```
int a=10, b=20; //declaring 2 variable of integer type
float f=20.8;
char c='A';
```

1. Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Valid variable names:

```
int a;
int _ab;
int a30;
```

Invalid variable names:

```
int 2;
int a b;
int long;
```

2. Types of Variables in C

There are many types of variables in c: local variable; global variable; static variable; automatic variable; external variable

- **Local Variable**

A variable that is declared inside the function or block is called a local variable.

It must be declared at the start of the block.

```
void function1(){
int x=10; //local variable
}
```

You must have to initialize the local variable before it is used.

- **Global Variable**

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

```
int value=20; //global variable
void function1(){
int x=10; //local variable
}
```

- **Static Variable**

A variable that is declared with the static keyword is called static variable.

It retains its value between multiple function calls.

```
void function1(){
int x=10; //local variable
static int y=10; //static variable
x=x+1;
y=y+1;
printf("%d,%d",x,y);

}
```

If you call this function many times, the local variable will print the same value for each function call, e.g, 11,11,11 and so on. But the static variable will print the incremented value in each function call, e.g. 11, 12, 13 and so on.

- **Automatic Variable**

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using auto keyword.

```
void main(){

int x=10; //local variable (also automatic)

auto int y=20; //automatic variable

}
```

- **External Variable**

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use extern keyword.

myfile.h

```
extern int x=10; //external variable (also global)
```

program1.c

```
#include "myfile.h"

#include <stdio.h>

void printValue(){

    printf("Global variable: %d", global_variable);

}
```

III. Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

There are the following data types in C language:

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

- **Basic Data Types**

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given according to 32-bit architecture.

Data Types	Memory Size	Range
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535

short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	

IV. Keywords in C

A keyword is a reserved word. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below:

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

We will learn about all the C language keywords later.

V. C Identifiers

C identifiers represent the name in the C program, for example, variables, functions, arrays, structures, unions, labels, etc. An identifier can be composed of letters such as uppercase, lowercase letters, underscore, digits, but the starting letter should be either an alphabet or an underscore. If the identifier is not used in the external linkage, then it is called as an internal identifier. If the identifier is used in the external linkage, then it is called as an external identifier.

We can say that an identifier is a collection of alphanumeric characters that begins either with an alphabetical character or an underscore, which are used to represent various programming elements such as variables, functions, arrays, structures, unions, labels, etc. There are 52 alphabetical characters (uppercase and lowercase), underscore character, and ten numerical digits (0-9) that represent the identifiers. There is a total of 63 alphanumerical characters that represent the identifiers.

1. Rules for constructing C identifiers

The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.

- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

<u>Example of valid identifiers:</u>

```
total, sum, average, _m _, sum_1, etc.
```

<u>Example of invalid identifiers:</u>

```
2sum (starts with a numerical digit)

int (reserved word)

char (reserved word)

m+n (special character, i.e., '+')
```

2. Types of identifiers

- **Internal Identifier**

If the identifier is not used in the external linkage, then it is known as an internal identifier. The internal identifiers can be local variables.

- **External Identifier**

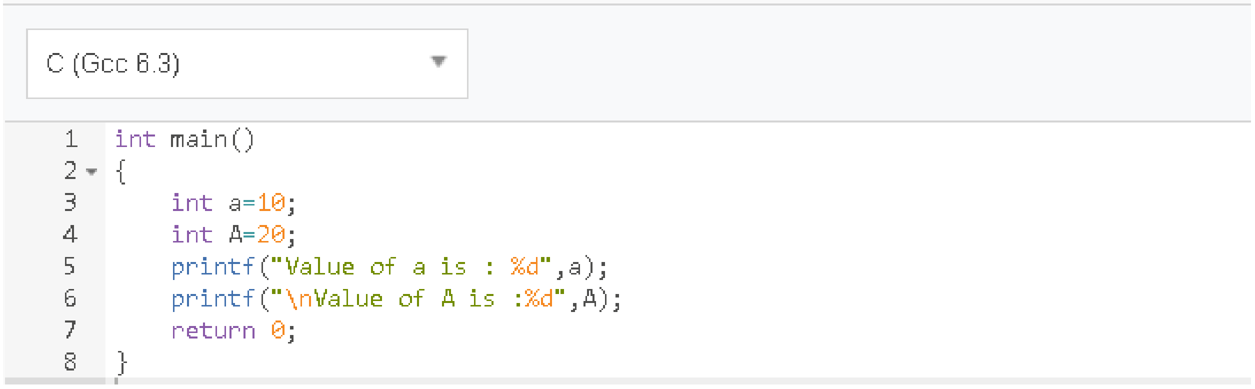
If the identifier is used in the external linkage, then it is known as an external identifier. The external identifiers can be function names, global variables.

3. Differences between Keyword and Identifier

Keyword	Identifier
Keyword is a pre-defined word.	The identifier is a user-defined word
It must be written in a lowercase letter.	It can be written in both lowercase and uppercase letters.
Its meaning is pre-defined in the c compiler.	Its meaning is not defined in the c compiler.
It is a combination of alphabetical characters.	It is a combination of alphanumeric characters.
It does not contain the underscore character.	It can contain the underscore character.

LAB: Let's understand through an example.

Step 1: Type the following code in the CodeChef IDE:

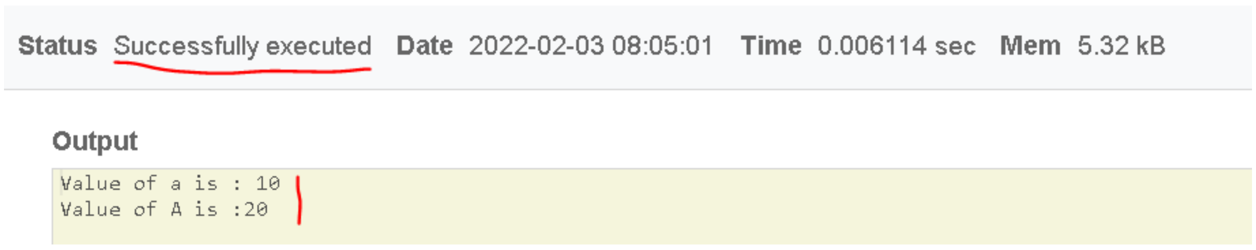


C (Gcc 6.3)

```
int main()
{
    int a=10;
    int A=20;
    printf("Value of a is : %d",a);
    printf("\nValue of A is :%d",A);
    return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



Status Successfully executed Date 2022-02-03 08:05:01 Time 0.006114 sec Mem 5.32 kB

Output

```
Value of a is : 10
Value of A is :20
```

The above output shows that the values of both the variables, 'a' and 'A' are different. Therefore, we conclude that the identifiers are case sensitive.

VI. C Operators

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language:

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Ternary or Conditional Operators
- Assignment Operator
- Misc Operator

Precedence of Operators in C

The precedence of operator species that which operator will be evaluated first and next. The associativity specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

```
int value=10+20*10;
```

The **value** variable will contain 210 because * (multiplicative operator) is evaluated before + (additive operator): 10 + (20 * 10) = 10 + 200 = 210

The precedence and associativity of C operators is given below:

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

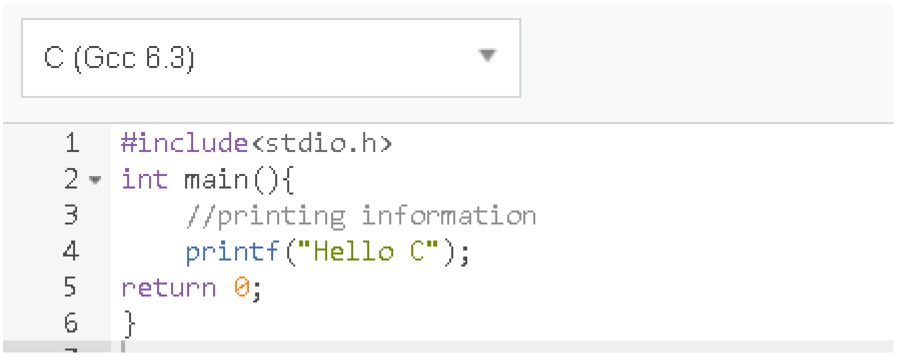
VII. C Comments

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments

Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

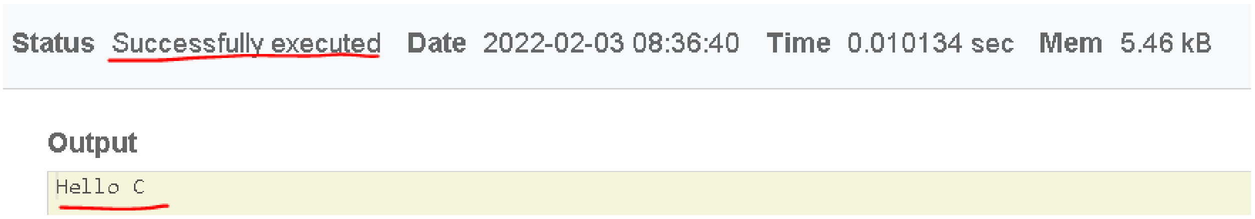
Step 1: Type the following code in the CodeChef IDE:



```c
#include<stdio.h>
int main(){
    //printing information
    printf("Hello C");
return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



```
Status  Successfully executed   Date  2022-02-03 08:36:40   Time  0.010134 sec   Mem  5.46 kB

Output
Hello C
```

Even you can place the comment after the statement. For example:

```c
printf("Hello C"); //printing information
```

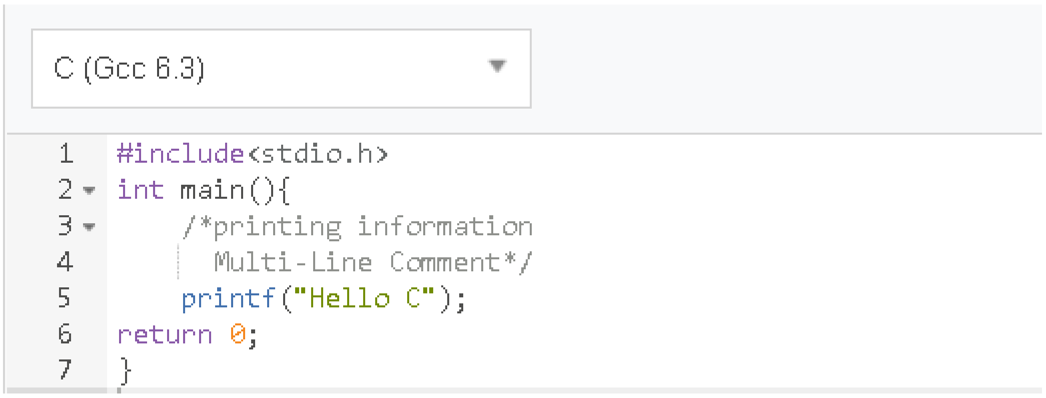
2. Mult Line Comments

Multi-Line comments are represented by slash asterisk * ... *\. It can occupy many lines of code, but it can't be nested. Syntax:

```c
/*
code
to be commented
*/
```

Let's see an example of a multi-Line comment in C.

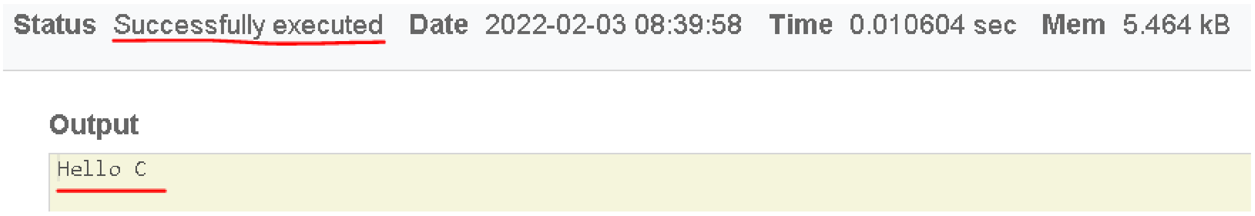
Step 1: Type the following code in the CodeChef IDE:



```c
#include<stdio.h>
int main(){
    /*printing information
      Multi-Line Comment*/
    printf("Hello C");
return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



```
Status  Successfully executed   Date  2022-02-03 08:39:58   Time  0.010604 sec   Mem  5.464 kB

Output
Hello C
```

VIII. C Format Specifier

The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.

The commonly used format specifiers in printf() function are:

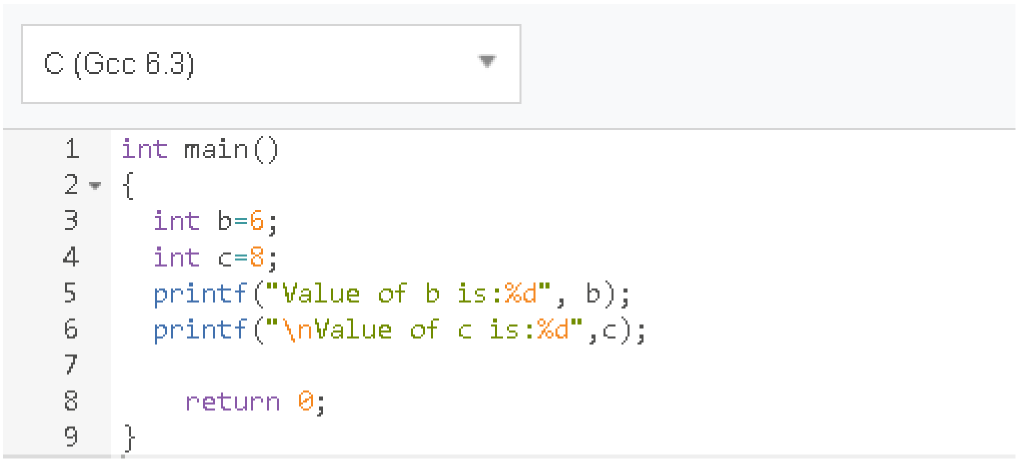
Format specifier	Description
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.

%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

1. **Let's understand the format specifiers in detail through an example.**

LAB: We are printing the integer value of b and c by using the %d specifier.

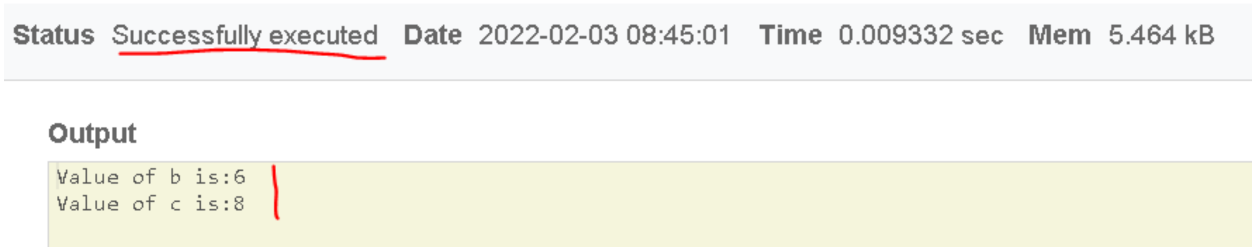
Step 1: Type the following code in the CodeChef IDE:



```
int main()
{
  int b=6;
  int c=8;
  printf("Value of b is:%d", b);
  printf("\nValue of c is:%d",c);

    return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



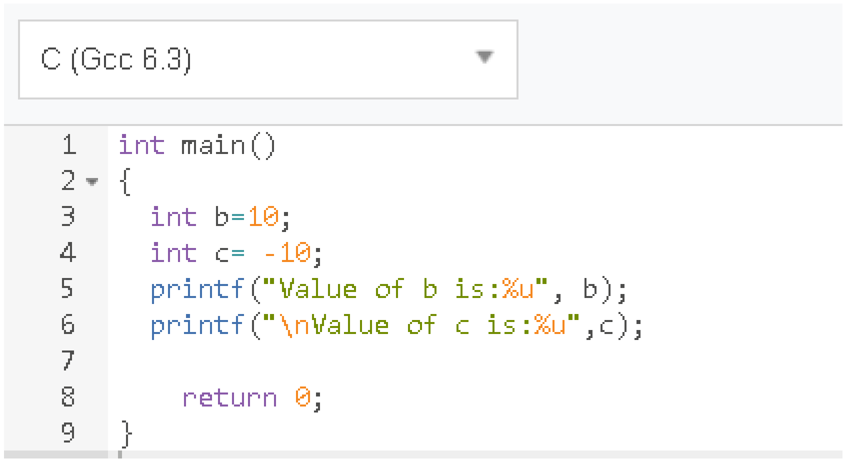
Status Successfully executed Date 2022-02-03 08:45:01 Time 0.009332 sec Mem 5.464 kB

Output

```
Value of b is:6
Value of c is:8
```

LAB: We are displaying the value of b and c by using an unsigned format specifier, i.e., %u. The value of b is positive, so %u specifier prints the exact value of b, but it does not print the value of c as c contains the negative value.

Step 1: Type the following code in the CodeChef IDE:

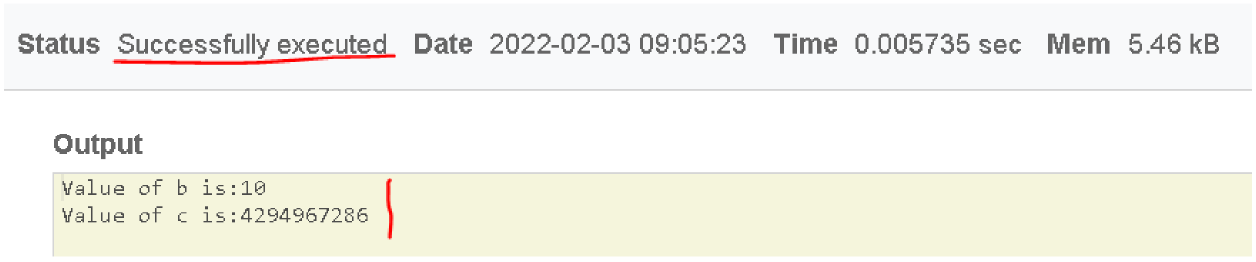


C (Gcc 6.3)

```c
int main()
{
  int b=10;
  int c= -10;
  printf("Value of b is:%u", b);
  printf("\nValue of c is:%u",c);

    return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



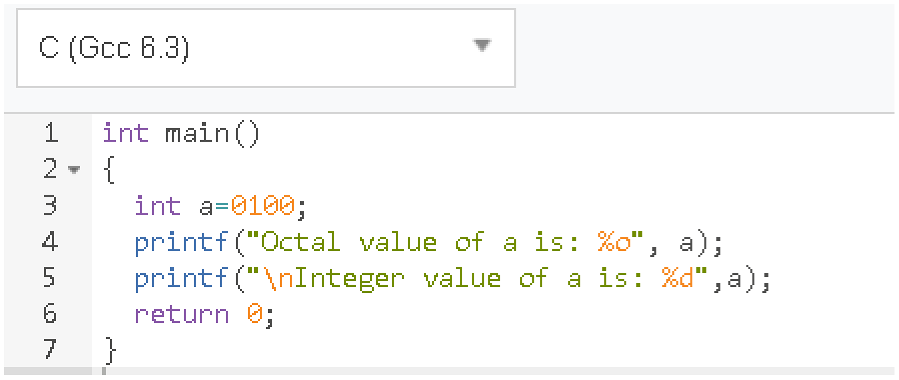
Status Successfully executed Date 2022-02-03 09:05:23 Time 0.005735 sec Mem 5.46 kB

Output

```
Value of b is:10
Value of c is:4294967286
```

LAB: We are displaying the octal value and integer value of a.

Step 1: Type the following code in the CodeChef IDE:

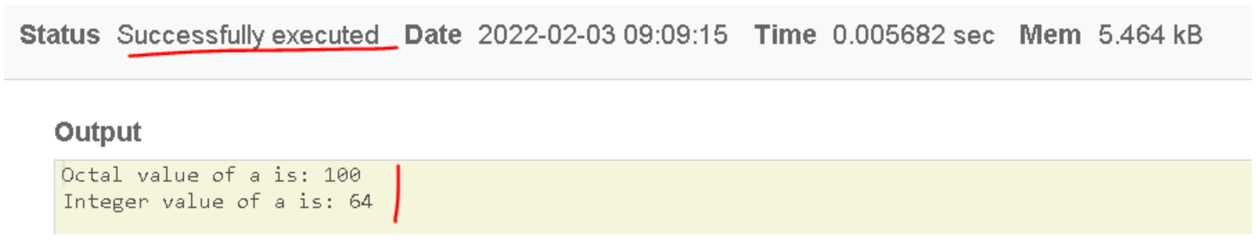


C (Gcc 6.3)

```c
int main()
{
  int a=0100;
  printf("Octal value of a is: %o", a);
  printf("\nInteger value of a is: %d",a);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



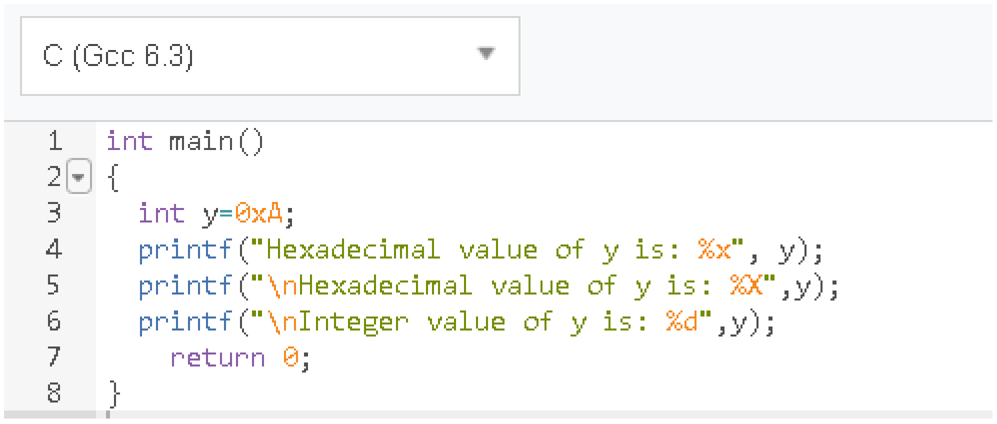
Status Successfully executed Date 2022-02-03 09:09:15 Time 0.005682 sec Mem 5.464 kB

Output

```
Octal value of a is: 100
Integer value of a is: 64
```

LAB: y contains the hexadecimal value 'A'. We display the hexadecimal value of y in two formats. We use %x and %X to print the hexadecimal value where %x displays the value in small letters, i.e., 'a' and %X displays the value in a capital letter, i.e., 'A'.

Step 1: Type the following code in the CodeChef IDE:

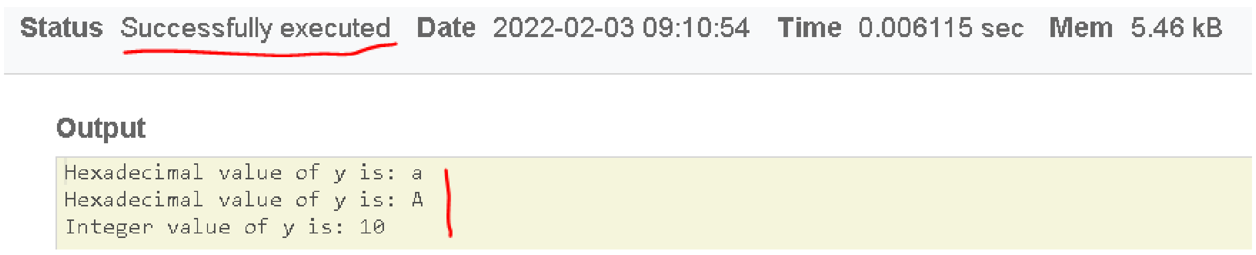


C (Gcc 6.3)

```c
int main()
{
  int y=0xA;
  printf("Hexadecimal value of y is: %x", y);
  printf("\nHexadecimal value of y is: %X",y);
  printf("\nInteger value of y is: %d",y);
    return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:

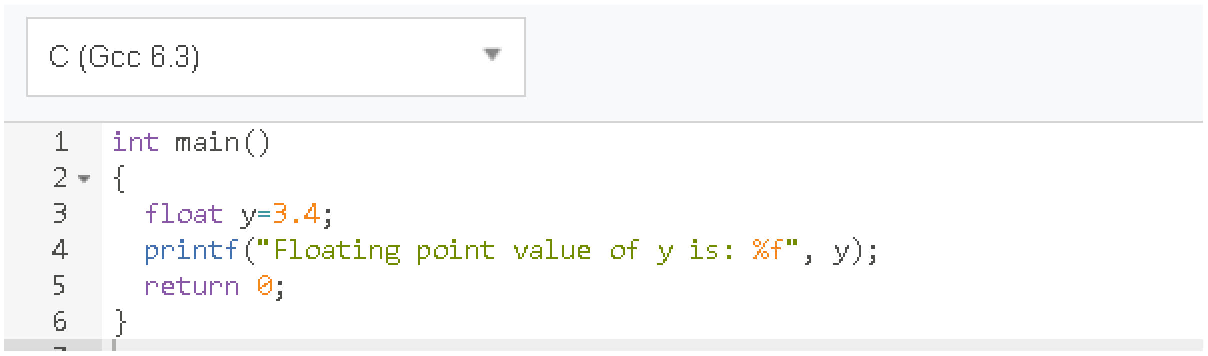

Status Successfully executed Date 2022-02-03 09:10:54 Time 0.006115 sec Mem 5.46 kB

Output

```
Hexadecimal value of y is: a
Hexadecimal value of y is: A
Integer value of y is: 10
```

LAB: Prints the floating value of y.

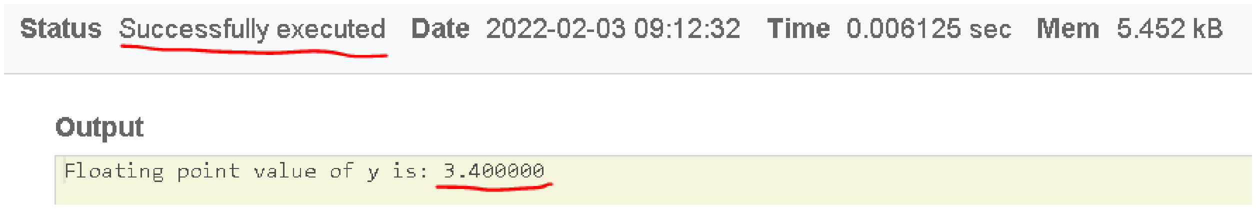
Step 1: Type the following code in the CodeChef IDE:


C (Gcc 6.3)

```
int main()
{
  float y=3.4;
  printf("Floating point value of y is: %f", y);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:

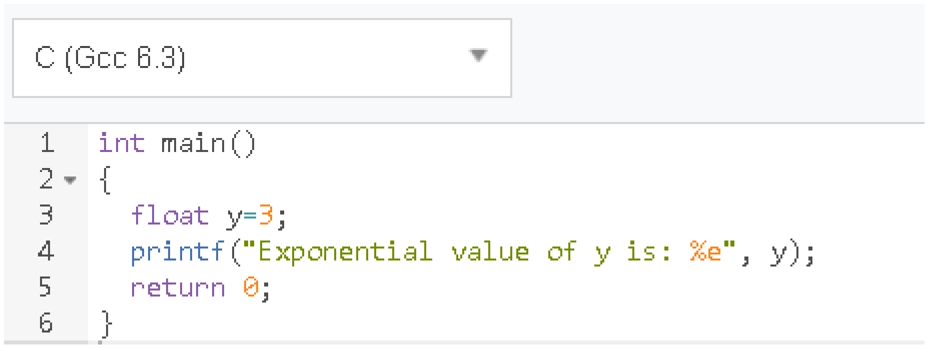

Status Successfully executed Date 2022-02-03 09:12:32 Time 0.006125 sec Mem 5.452 kB

Output

```
Floating point value of y is: 3.400000
```

LAB: %e

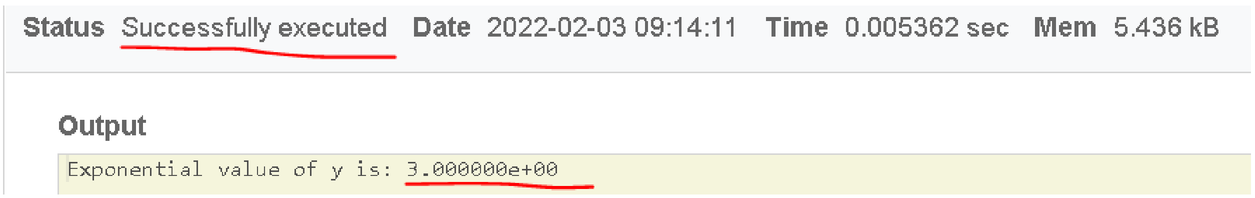
Step 1: Type the following code in the CodeChef IDE:


C (Gcc 6.3)

```
int main()
{
  float y=3;
  printf("Exponential value of y is: %e", y);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



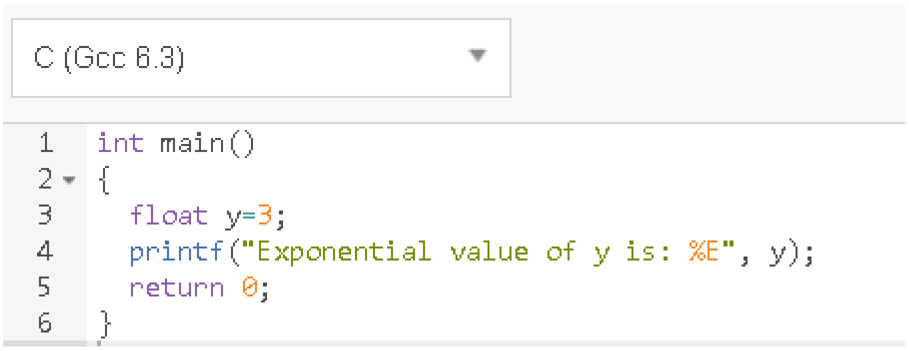
```
Status Successfully executed   Date 2022-02-03 09:14:11   Time 0.005362 sec   Mem 5.436 kB
```

Output

```
Exponential value of y is: 3.000000e+00
```

LAB: %E

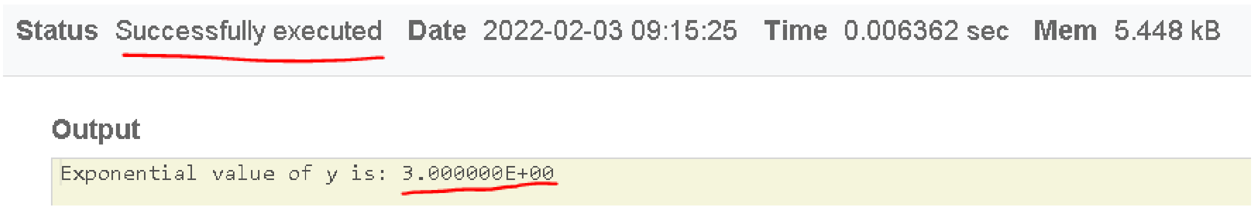
Step 1: Type the following code in the CodeChef IDE:



```c
int main()
{
  float y=3;
  printf("Exponential value of y is: %E", y);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



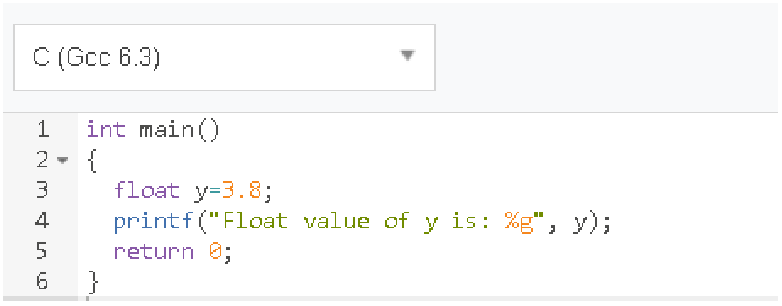
```
Status Successfully executed   Date 2022-02-03 09:15:25   Time 0.006362 sec   Mem 5.448 kB
```

Output

```
Exponential value of y is: 3.000000E+00
```

LAB: %g

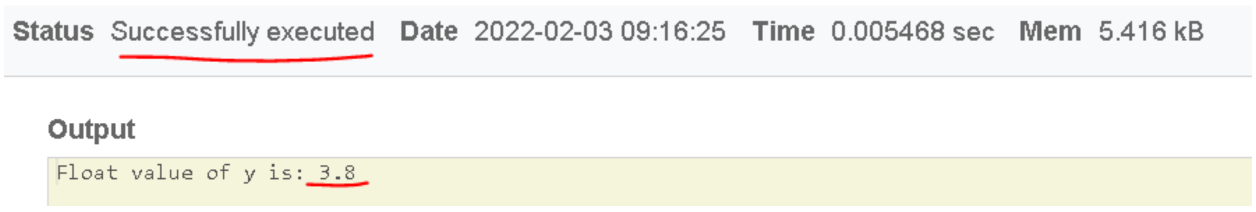
Step 1: Type the following code in the CodeChef IDE:



```c
int main()
{
  float y=3.8;
  printf("Float value of y is: %g", y);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



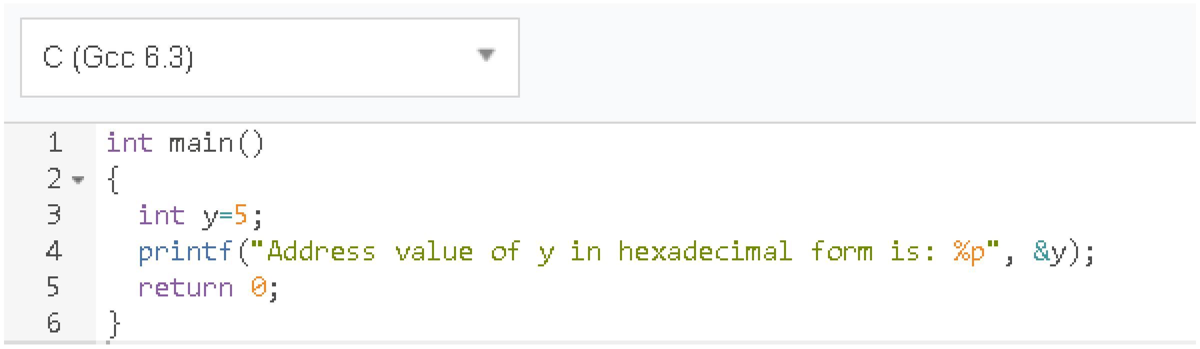
```
Status  Successfully executed   Date  2022-02-03 09:16:25   Time  0.005468 sec   Mem  5.416 kB
```

Output

```
Float value of y is: 3.8
```

LAB: %p

Step 1: Type the following code in the CodeChef IDE:

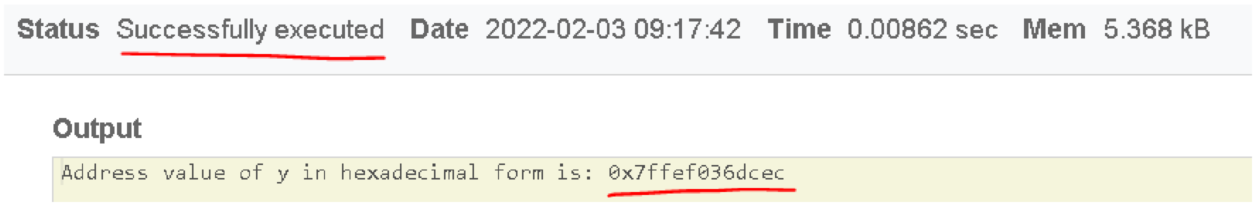


C (Gcc 6.3)

```c
int main()
{
  int y=5;
  printf("Address value of y in hexadecimal form is: %p", &y);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



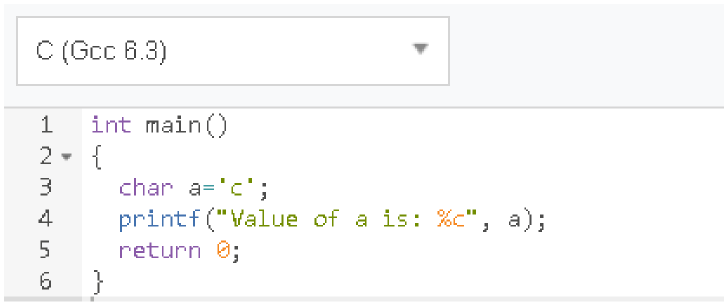
```
Status  Successfully executed   Date  2022-02-03 09:17:42   Time  0.00862 sec   Mem  5.368 kB
```

Output

```
Address value of y in hexadecimal form is: 0x7ffef036dcec
```

LAB: %c

Step 1: Type the following code in the CodeChef IDE:

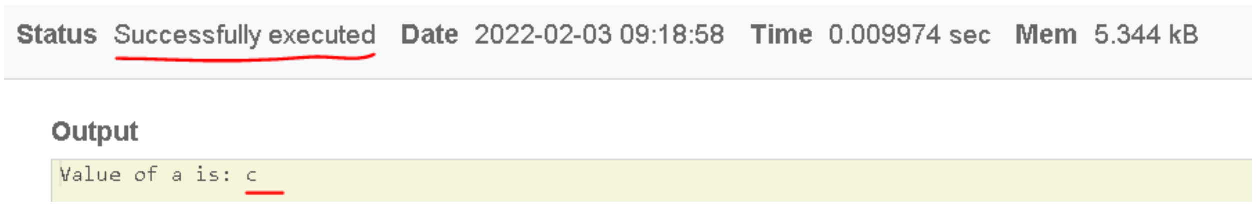


C (Gcc 6.3)

```c
int main()
{
  char a='c';
  printf("Value of a is: %c", a);
  return 0;
}
```

Step 2: Run the program by Run button.

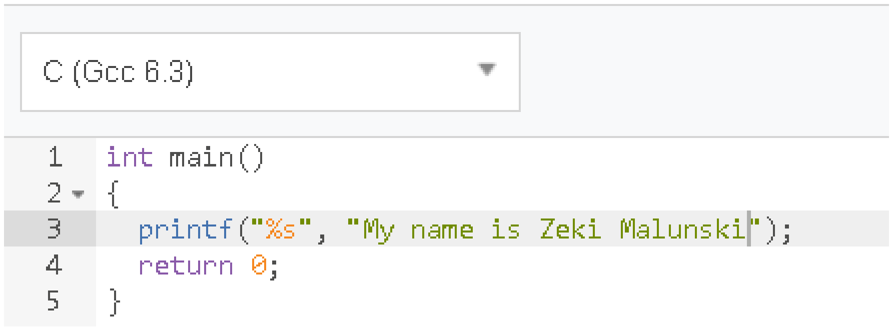
Step 3: Check the result:



```
Status  Successfully executed   Date  2022-02-03 09:18:58   Time  0.009974 sec   Mem  5.344 kB

Output
Value of a is: c
```

LAB: %s

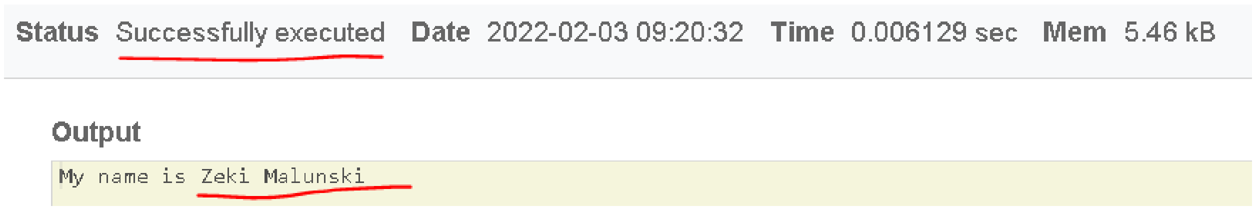
Step 1: Type the following code in the CodeChef IDE:



```c
int main()
{
  printf("%s", "My name is Zeki Malunski");
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



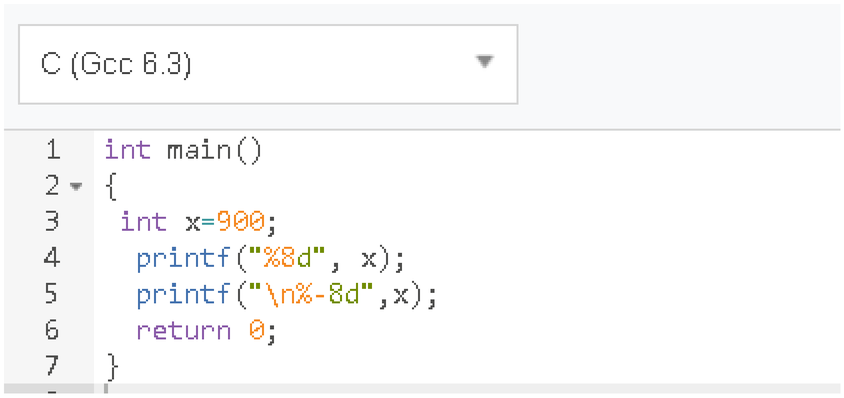
```
Status  Successfully executed   Date  2022-02-03 09:20:32   Time  0.006129 sec   Mem  5.46 kB

Output
My name is Zeki Malunski
```

2. Minimum Field Width Specifier

Suppose we want to display an output that occupies a minimum number of spaces on the screen. You can achieve this by displaying an integer number after the percent sign of the format specifier.

LAB: %8d specifier displays the value after 8 spaces while %-8d specifier will make a value left-aligned.

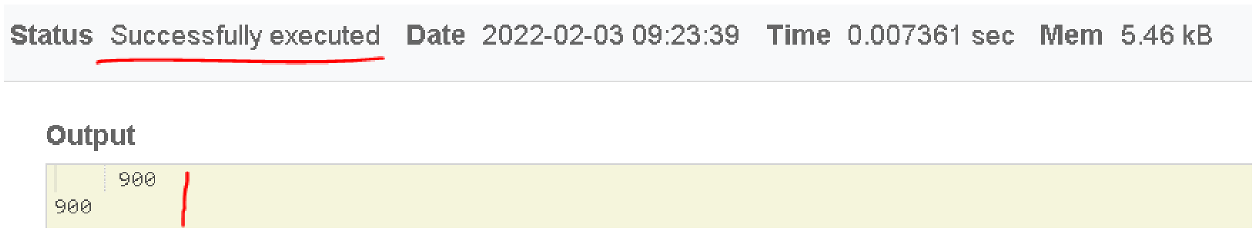
Step 1: Type the following code in the CodeChef IDE:


C (Gcc 6.3)

```c
int main()
{
 int x=900;
   printf("%8d", x);
   printf("\n%-8d",x);
   return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:

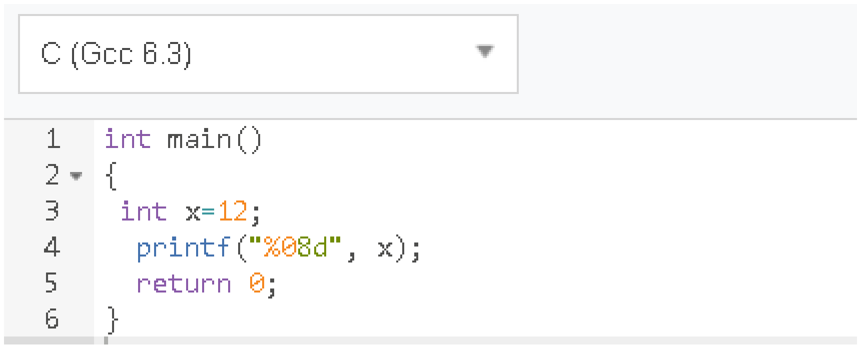

Status Successfully executed **Date** 2022-02-03 09:23:39 **Time** 0.007361 sec **Mem** 5.46 kB

Output

```
     900
900
```

LAB: We will see how to fill the empty spaces. %08d means that the empty space is filled with zeroes.

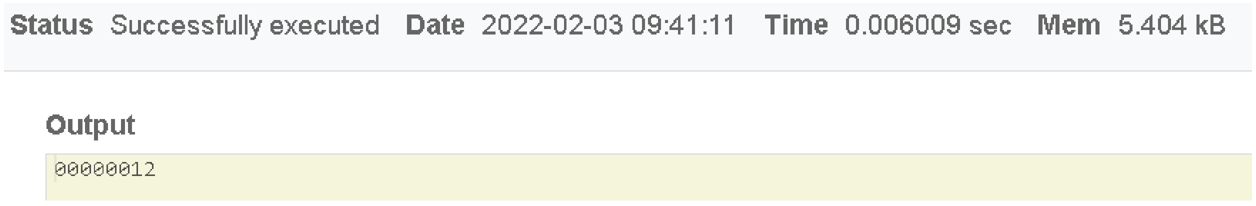
Step 1: Type the following code in the CodeChef IDE:


C (Gcc 6.3)

```c
int main()
{
 int x=12;
   printf("%08d", x);
   return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:


Status Successfully executed **Date** 2022-02-03 09:41:11 **Time** 0.006009 sec **Mem** 5.404 kB

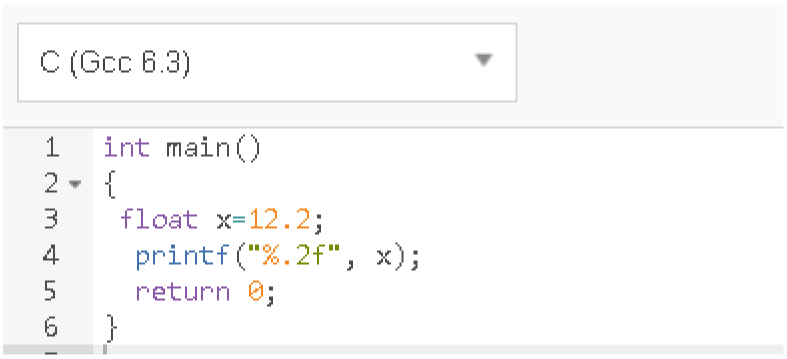
Output

```
00000012
```

3. Specifying Precision

We can specify the precision by using '.' (Dot) operator which is followed by integer and format specifier.

LAB: Step 1: Type the following code in the CodeChef IDE:

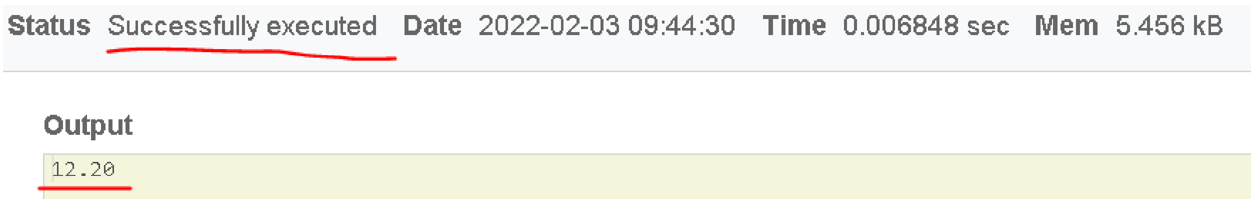


C (Gcc 6.3)

```c
int main()
{
 float x=12.2;
  printf("%.2f", x);
  return 0;
}
```

Step 2: Run the program by Run button.

Step 3: Check the result:



```
Status  Successfully executed   Date  2022-02-03 09:44:30   Time  0.006848 sec   Mem  5.456 kB

Output
12.20
```

IX. C Escape Sequence

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

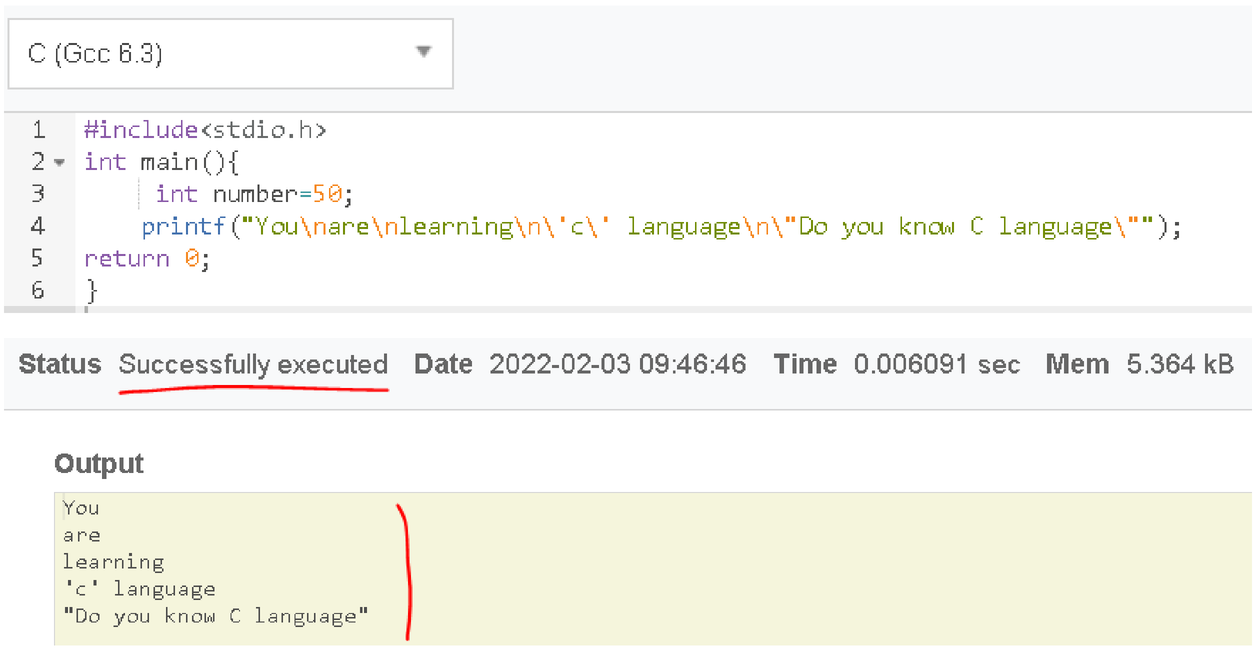
It is composed of two or more characters starting with backslash \. For example: \n represents new line.

List of Escape Sequences in C

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace

\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

Escape Sequence Example



```c
#include<stdio.h>
int main(){
     int number=50;
     printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\"");
return 0;
}
```

Status Successfully executed **Date** 2022-02-03 09:46:46 **Time** 0.006091 sec **Mem** 5.364 kB

Output

```
You
are
learning
'c' language
"Do you know C language"
```