EXERCISE 3

Content:

I. ASCII value in C

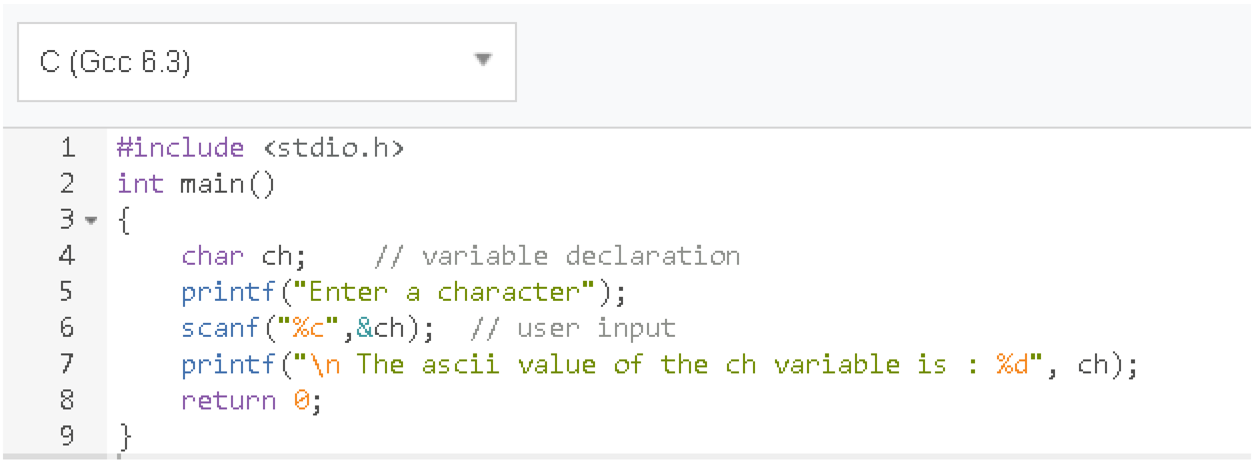
The full form of ASCII is the American Standard Code for information interchange. It is a character encoding scheme used for electronics communication. Each character or a special character is represented by some ASCII code, and each ascii code occupies 7 bits in memory.

In C programming language, a character variable does not contain a character value itself rather the ascii value of the character variable. The ascii value represents the character variable in numbers, and each character variable is assigned with some number range from 0 to 127. For example, the ascii value of 'A' is 65.

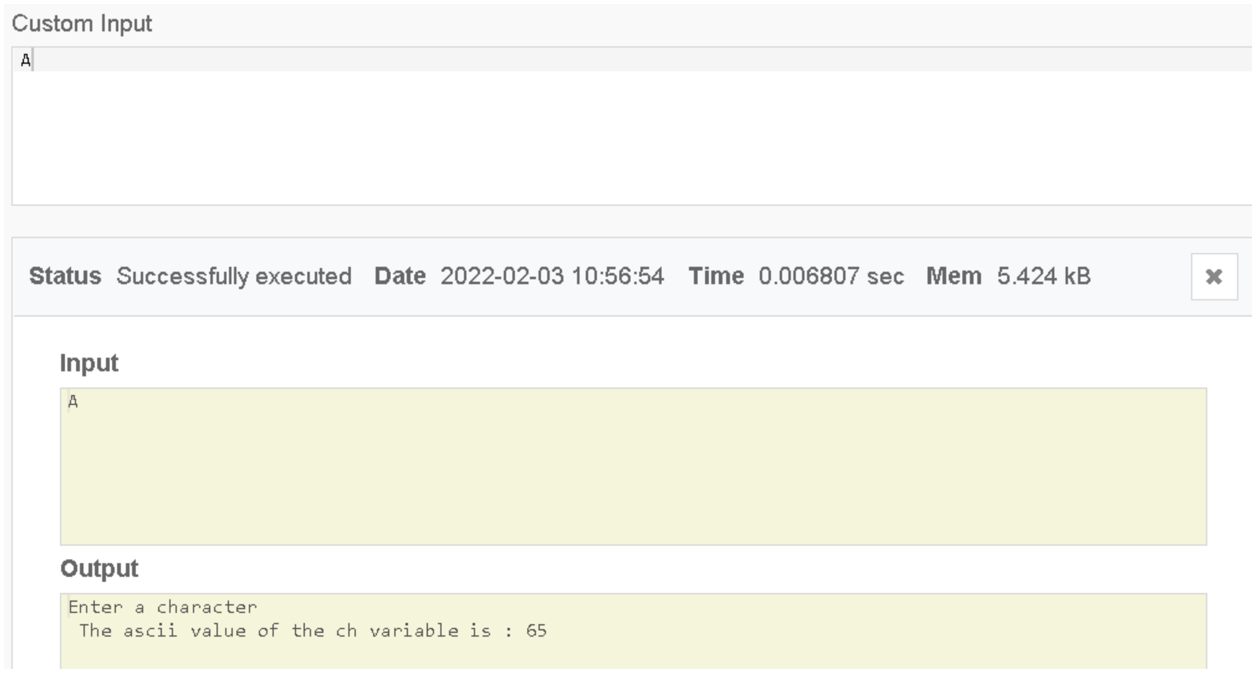
In the above example, we assign 'A' to the character variable whose ascii value is 65, so 65 will be stored in the character variable rather than 'A'.

Let's understand through an example.

LAB: Create a program which will display the ascii value of the character variable.



```c
#include <stdio.h>
int main()
{
    char ch;    // variable declaration
    printf("Enter a character");
    scanf("%c",&ch);  // user input
    printf("\n The ascii value of the ch variable is : %d", ch);
    return 0;
}
```


Custom Input

```
A
```

Status Successfully executed Date 2022-02-03 10:56:54 Time 0.006807 sec Mem 5.424 kB

Input

```
A
```

Output

```
Enter a character
 The ascii value of the ch variable is : 65
```

The first you will give the character input (in the example it is “A”), and the input will get stored in the 'ch' variable. If we print the value of the 'ch' variable by using %c format specifier, then it will display 'A' because we have given the character input as 'A', and if we use the %d format specifier then its ascii value will be displayed, i.e., 65.

II. Constants in C

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

There are different types of constants in C programming.

List of Constants in C

onstant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.

Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c in javatpoint" etc.

2 ways to define constant in C

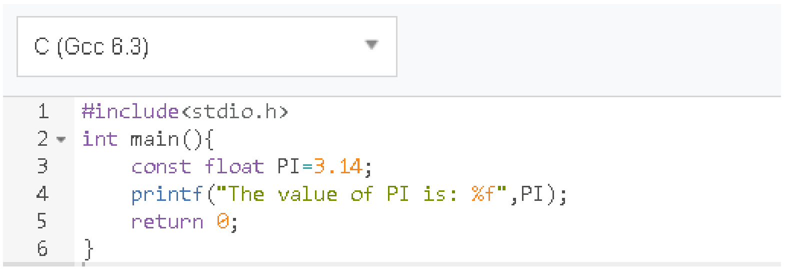
There are two ways to define constant in C programming – const keyword and #define preprocessor.

- **C const keyword**

The const keyword is used to define constant in C programming.

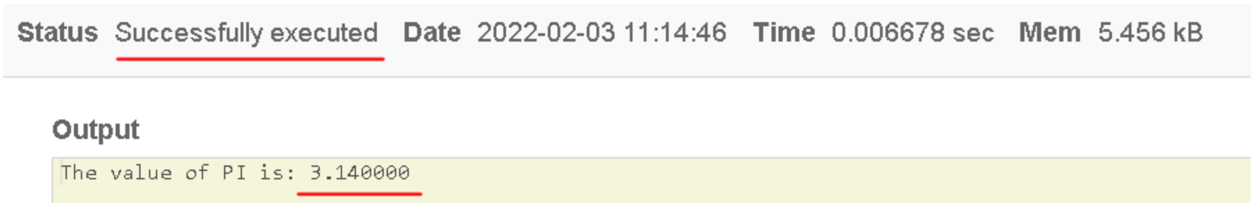
```c
const float PI=3.14;
```

The value of PI variable can't be changed.



```c
#include<stdio.h>
int main(){
    const float PI=3.14;
    printf("The value of PI is: %f",PI);
    return 0;
}
```

Output:

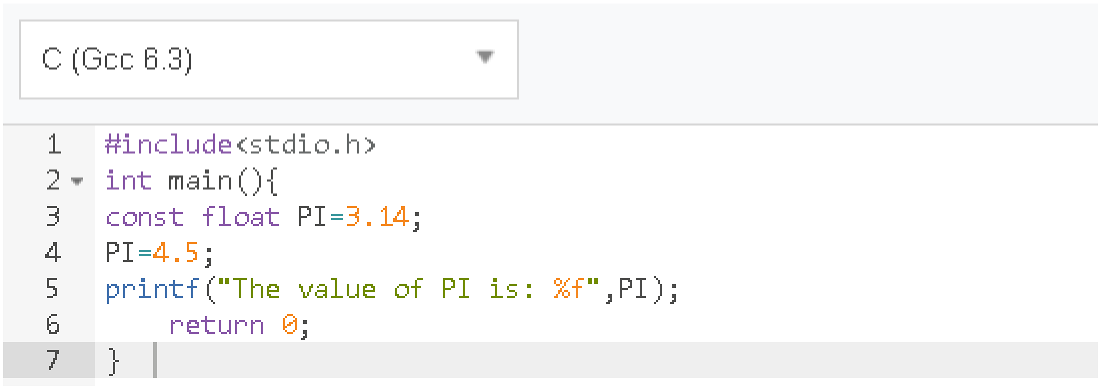


Status Successfully executed Date 2022-02-03 11:14:46 Time 0.006678 sec Mem 5.456 kB

Output

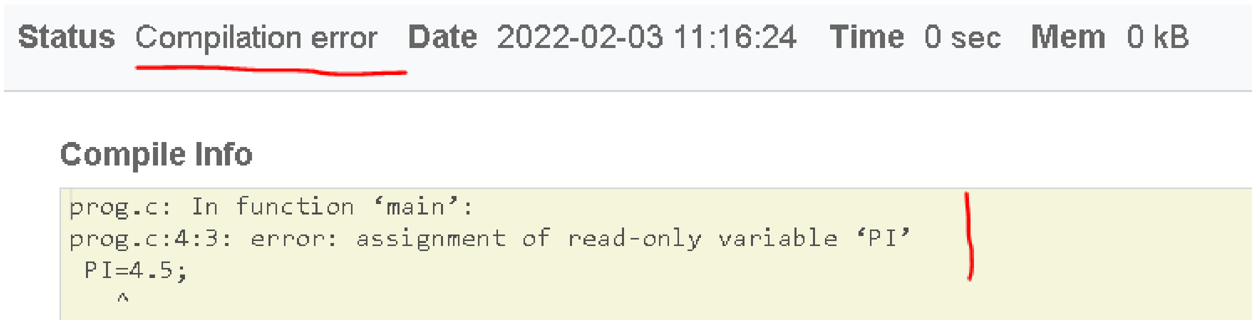
```
The value of PI is: 3.140000
```

If you try to change the value of PI, it will render compile time error.



```c
#include<stdio.h>
int main(){
const float PI=3.14;
PI=4.5;
printf("The value of PI is: %f",PI);
    return 0;
}
```

Output:


Status Compilation error Date 2022-02-03 11:16:24 Time 0 sec Mem 0 kB

Compile Info

```
prog.c: In function ‘main’:
prog.c:4:3: error: assignment of read-only variable ‘PI’
 PI=4.5;
   ^
```

- **C #define preprocessor**

The #define preprocessor is also used to define constant. We will learn about #define preprocessor directive later.

III. Literals in C

Literals are the constant values assigned to the constant variables. We can say that the literals represent the fixed values that cannot be modified. It also contains memory but does not have references as variables. For example, const int =10; is a constant integer expression in which 10 is an integer literal.

1. Types of literals

There are four types of literals that exist in C programming: Integer literal; Float literal; Character literal and String literal.

- **Integer literal**

It is a numeric literal that represents only integer type values. It represents the value neither in fractional nor exponential part.

It can be specified in the following three ways:

Decimal number (base 10)

It is defined by representing the digits between 0 to 9. For example, 45, 67, etc.

Octal number (base 8)

It is defined as a number in which 0 is followed by digits such as 0,1,2,3,4,5,6,7. For example, 012, 034, 055, etc.

Hexadecimal number (base 16)

It is defined as a number in which 0x or 0X is followed by the hexadecimal digits (i.e., digits from 0 to 9, alphabetical characters from (a-z) or (A-Z)).

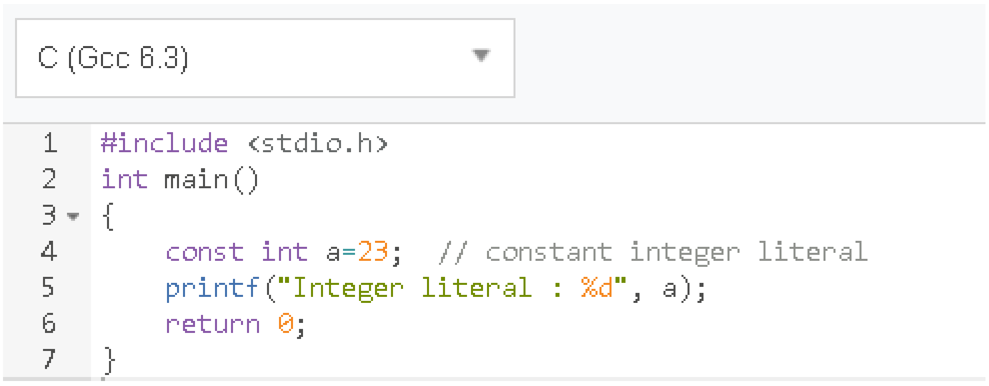
An integer literal is suffixed by following two sign qualifiers:

L or l: It is a size qualifier that specifies the size of the integer type as long.

U or u: It is a sign qualifier that represents the type of the integer as unsigned. An unsigned qualifier contains only positive values.

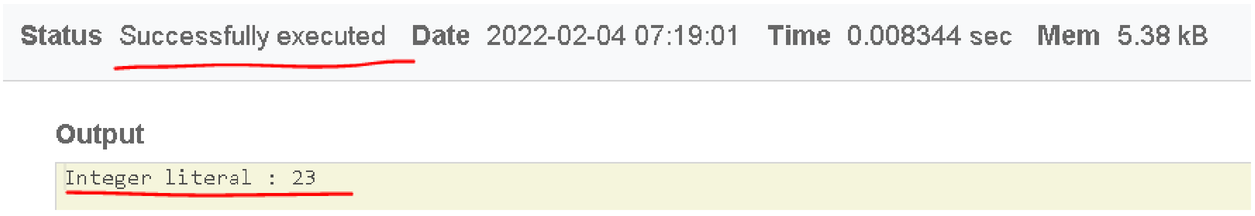
Note: The order of the qualifier is not considered, i.e., both lu and ul are the same.

LAB: Simple example of integer literal.



```c
#include <stdio.h>
int main()
{
    const int a=23;  // constant integer literal
    printf("Integer literal : %d", a);
    return 0;
}
```

Result/Output:



```
Status  Successfully executed   Date  2022-02-04 07:19:01   Time  0.008344 sec   Mem  5.38 kB

Output
Integer literal : 23
```

- **Float literal**

It is a literal that contains only floating-point values or real numbers. These real numbers contain the number of parts such as integer part, real part, exponential part, and fractional part. The floating-point literal must be specified either in decimal or in exponential form. Let's understand these forms in brief.

Decimal form

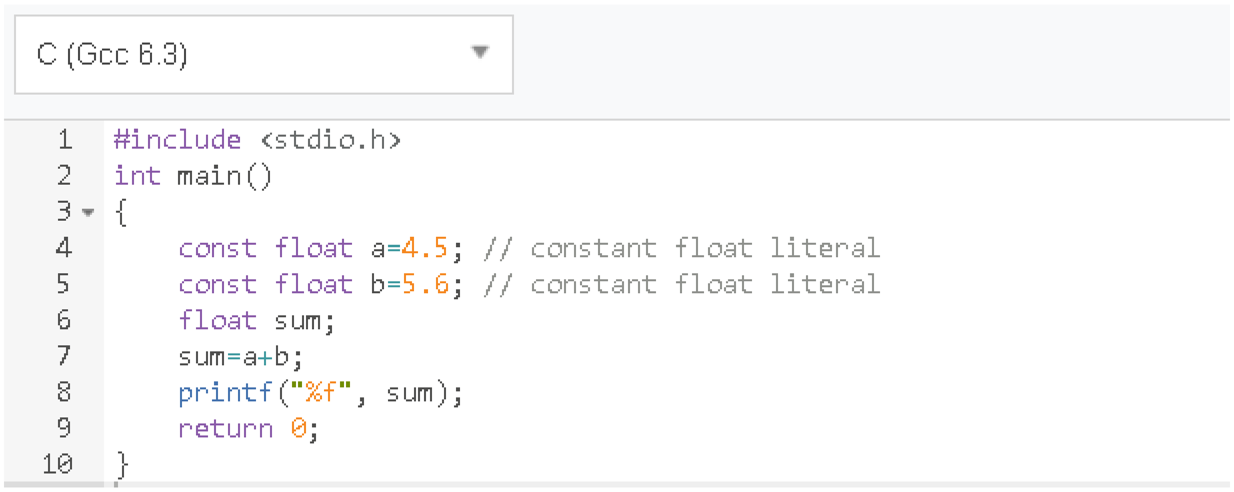
The decimal form must contain either decimal point, exponential part, or both. If it does not contain either of these, then the compiler will throw an error.

The decimal notation can be prefixed either by '+' or '-' symbol that specifies the positive and negative numbers.

Examples of float literal in decimal form are:

1.2, +9.0, -4.5

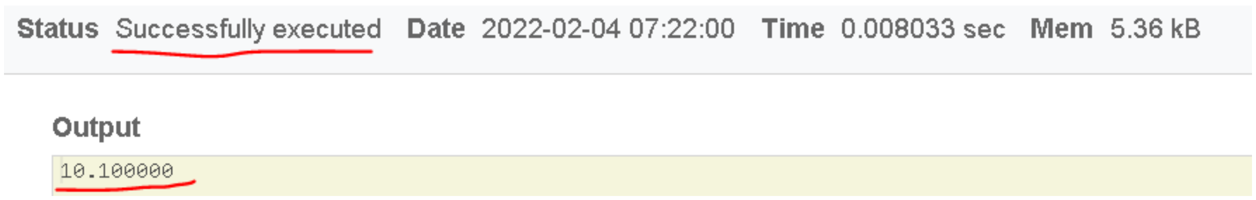
LAB: Simple example of float literal in decimal form.



C (Gcc 6.3)

```c
#include <stdio.h>
int main()
{
    const float a=4.5; // constant float literal
    const float b=5.6; // constant float literal
    float sum;
    sum=a+b;
    printf("%f", sum);
    return 0;
}
```

Result/Output:



Status Successfully executed Date 2022-02-04 07:22:00 Time 0.008033 sec Mem 5.36 kB

Output

```
10.100000
```

Exponential form

The exponential form is useful when we want to represent the number, which is having a big magnitude. It contains two parts, i.e., mantissa and exponent. For example, the number is 2340000000000, and it can be expressed as 2.34e12 in an exponential form.

Syntax of float literal in exponential form:

```
[+/-] <Mantissa> <e/E> [+/-] <Exponent>
```

Examples of real literal in exponential notation are:

```
+1e23, -9e2, +2e-25
```

Rules for creating an exponential notation:

The following are the rules for creating a float literal in exponential notation:

- In exponential notation, the mantissa can be specified either in decimal or fractional form.
- An exponent can be written in both uppercase and lowercase, i.e., e and E.
- We can use both the signs, i.e., positive and negative, before the mantissa and exponent.
- Spaces are not allowed

- **Character literal**

A character literal contains a single character enclosed within single quotes. If multiple characters are assigned to the variable, then we need to create a character array. If we try to store more than one character in a variable, then the warning of a multi-character character constant will be generated.

Representation of character literal

A character literal can be represented in the following ways:

- It can be represented by specifying a single character within single quotes. For example, 'a', 'b', etc.
- We can specify the escape sequence character within single quotes to represent a character literal. For example, '\n', '\a', '\b'.
- We can also use the ASCII in integer to represent a character literal. For example, the ascii value of 65 is 'A'.
- The octal and hexadecimal notation can be used as an escape sequence to represent a character literal. For example, '\023', '\0x12'.

- **String literal**

A string literal represents multiple characters enclosed within double-quotes. It contains an additional character, i.e., '\0' (null character), which gets automatically inserted. This null character specifies the termination of the string. We can use the '+' symbol to concatenate two strings.

For example,

```
String1= "My name is";
```

```
String2= "Zeki";
```

To concatenate the above two strings, we use '+' operator, as shown in the below statement:

"My name is " + "Zeki"= My name is Zeki

<u>Note:</u> If we represent a single character, i.e., 'b', then this character will occupy a single byte as it is a character literal. And, if we represent the character within double quotes "b" then it will occupy more bytes as it is a string literal.

IV. Tokens in C

Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

1. Classification of tokens in C

Tokens in C language can be divided into the following categories:

- **Keywords in C**

Keywords in C can be defined as the pre-defined or the reserved words having its own importance, and each keyword has its own functionality. Since keywords are the pre-defined words used by the compiler, so they cannot be used as the variable names. If the keywords are used as the variable names, it means that we are assigning a different meaning to the keyword, which is not allowed. C language supports 32 keywords given below:

auto	**double**	**int**	**struct**
break	**else**	**long**	**switch**

case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

- **Identifiers in C**

Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords. Rules for constructing identifiers in C are given below:

- 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.
- **Strings in C**

Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

```
char a[10] = "cplanguage"; // The compiler allocates the 10 bytes to the
'a' array.
```

```
char a[] = "cplanguage"; // The compiler allocates the memory at the run time.

char a[10] = {'c','p','l','a','n','g','u','a','g','e','\0'}; // String is represented in the form of characters.
```

- **Operators in C**

Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

<u>Unary Operator</u>

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

<u>Binary Operator</u>

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

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

- **Constants in C**

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

- Using const keyword
- Using #define pre-processor

<u>Types of constants in C:</u>

Constant	Example

Integer constant	**10, 11, 34, etc.**
Floating-point constant	**45.6, 67.8, 11.2, etc.**
Octal constant	**011, 088, 022, etc.**
Hexadecimal constant	**0x1a, 0x4b, 0x6b, etc.**
Character constant	**'a', 'b', 'c', etc.**
String constant	**"java", "c++", ".net", etc.**

Special characters in C

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.

V. C Boolean

In C, Boolean is a data type that contains two types of values, i.e., 0 and 1. Basically, the bool type value represents two types of behavior, either true or false. Here, '0' represents false value, while '1' represents true value.

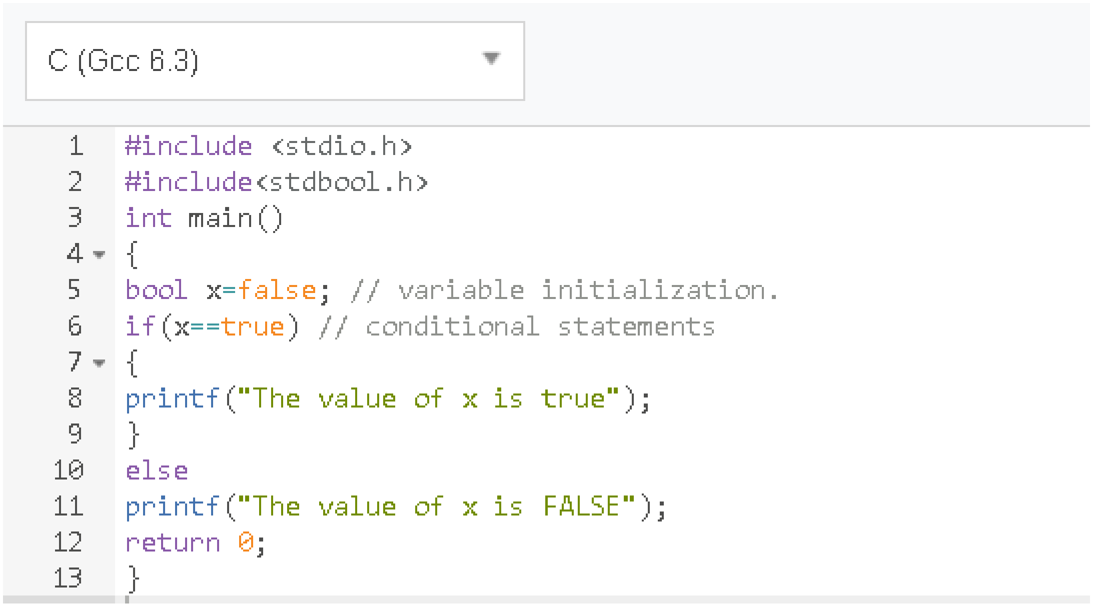
In C Boolean, '0' is stored as 0, and another integer is stored as 1. We do not require to use any header file to use the Boolean data type in C++, but in C, we have to use the header file, i.e., stdbool.h. If we do not use the header file, then the program will not compile.

Syntax

```
bool variable_name;
```

In the above syntax, bool is the data type of the variable, and variable_name is the name of the variable.

LAB: Let's understand through an example.

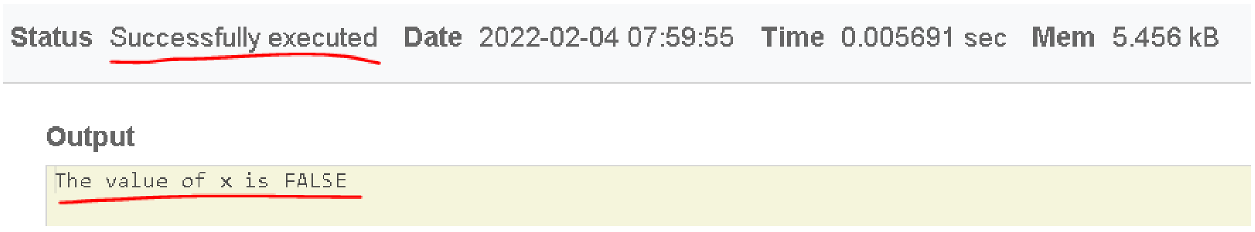


C (Gcc 6.3)

```
#include <stdio.h>
#include<stdbool.h>
int main()
{
bool x=false; // variable initialization.
if(x==true) // conditional statements
{
printf("The value of x is true");
}
else
printf("The value of x is FALSE");
return 0;
}
```

In the above code, we have used <stdbool.h> header file so that we can use the bool type variable in our program. After the declaration of the header file, we create the bool type variable 'x' and assigns a 'false' value to it. Then, we add the conditional statements, i.e., if..else, to determine whether the value of 'x' is true or not.

Result/Output:



Status Successfully executed Date 2022-02-04 07:59:55 Time 0.005691 sec Mem 5.456 kB

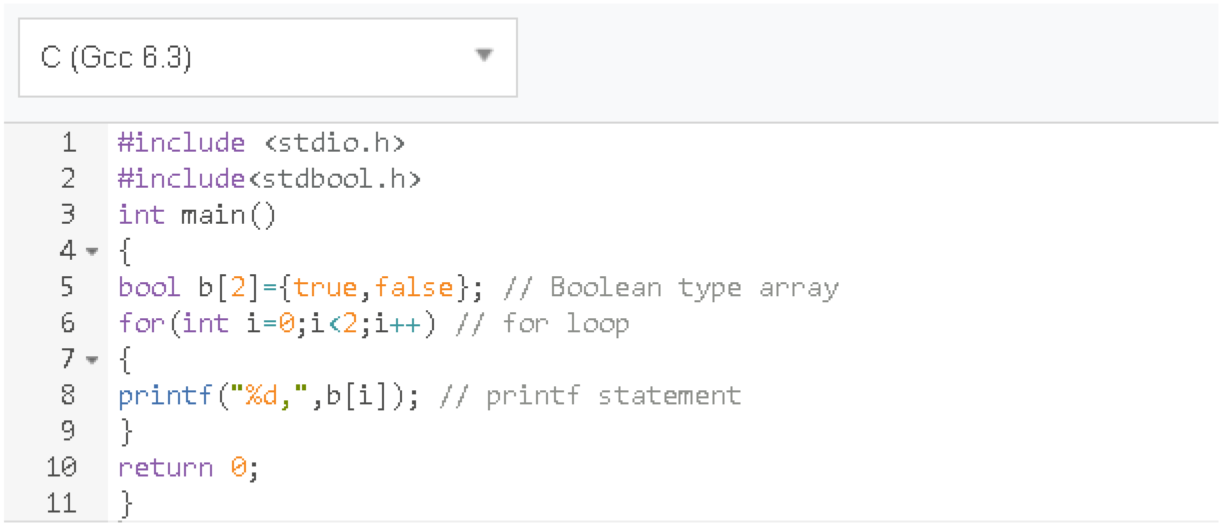
Output

```
The value of x is FALSE
```

Boolean Array

Now, we create a bool type array. The Boolean array can contain either true or false value, and the values of the array can be accessed with the help of indexing.

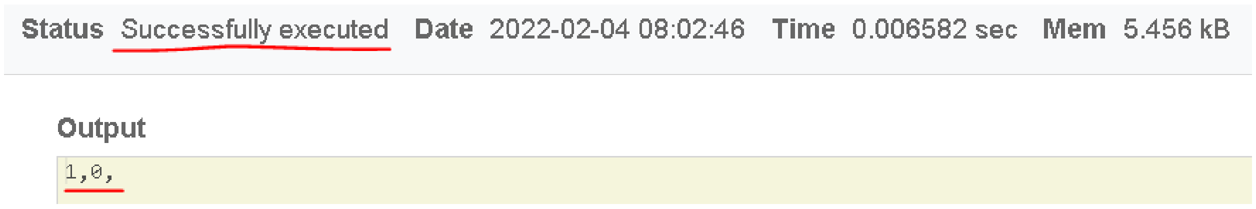
LAB: Let's understand this scenario through an example.


C (Gcc 6.3)

```c
#include <stdio.h>
#include<stdbool.h>
int main()
{
bool b[2]={true,false}; // Boolean type array
for(int i=0;i<2;i++) // for loop
{
printf("%d,",b[i]); // printf statement
}
return 0;
}
```

In the above code, we have declared a Boolean type array containing two values, i.e., true and false.

Result/Output:


Status Successfully executed Date 2022-02-04 08:02:46 Time 0.006582 sec Mem 5.456 kB

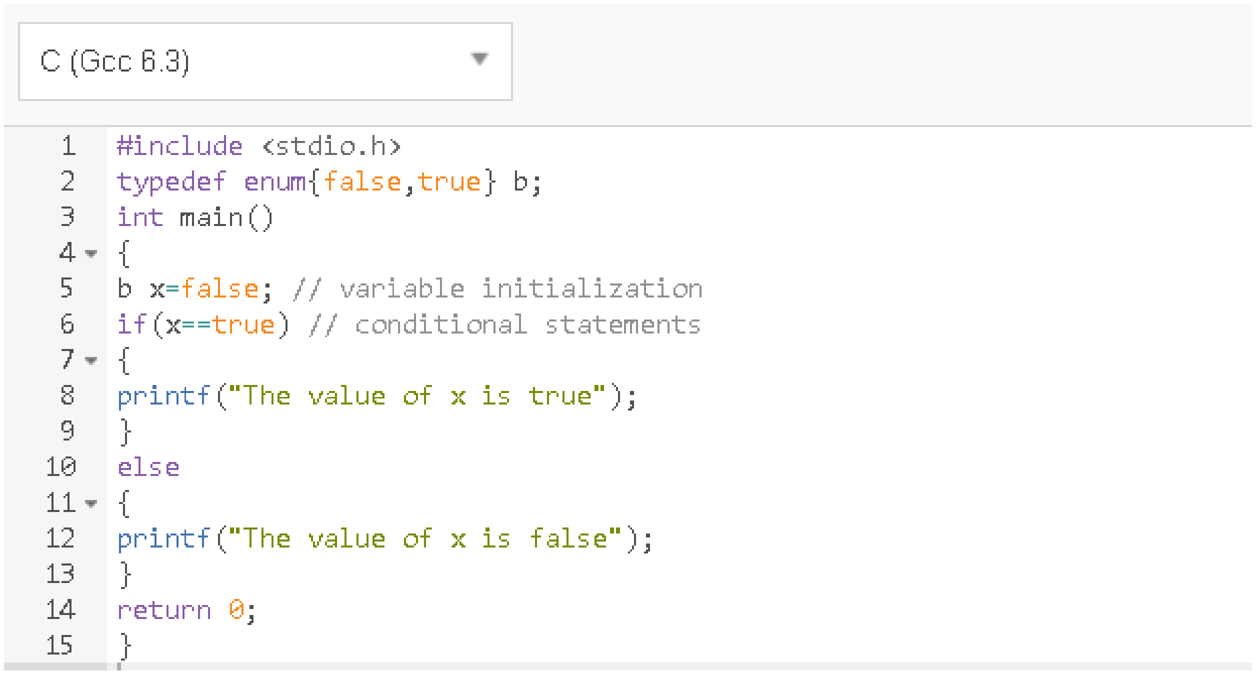
Output

```
1,0,
```

typedef

There is another way of using Boolean value, i.e., typedef. Basically, typedef is a keyword in C language, which is used to assign the name to the already existing datatype.

LAB: Let's see a simple example of typedef.



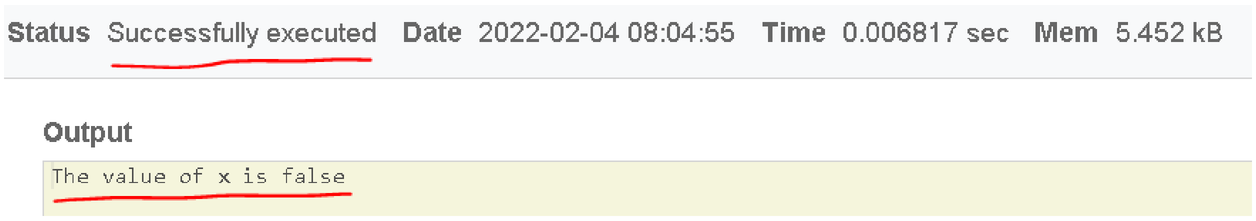
```
#include <stdio.h>
typedef enum{false,true} b;
int main()
{
b x=false; // variable initialization
if(x==true) // conditional statements
{
printf("The value of x is true");
}
else
{
printf("The value of x is false");
}
return 0;
}
```

In the above code, we use the Boolean values, i.e., true and false, but we have not used the bool type. We use the Boolean values by creating a new name of the 'bool' type. In order to achieve this, the typedef keyword is used in the program.

```
typedef enum{false,true} b;
```

The above statement creates a new name for the 'bool' type, i.e., 'b' as 'b' can contain either true or false value. We use the 'b' type in our program and create the 'x' variable of type 'b'.

Result/Output:



```
Status  Successfully executed   Date  2022-02-04 08:04:55   Time  0.006817 sec   Mem  5.452 kB
```

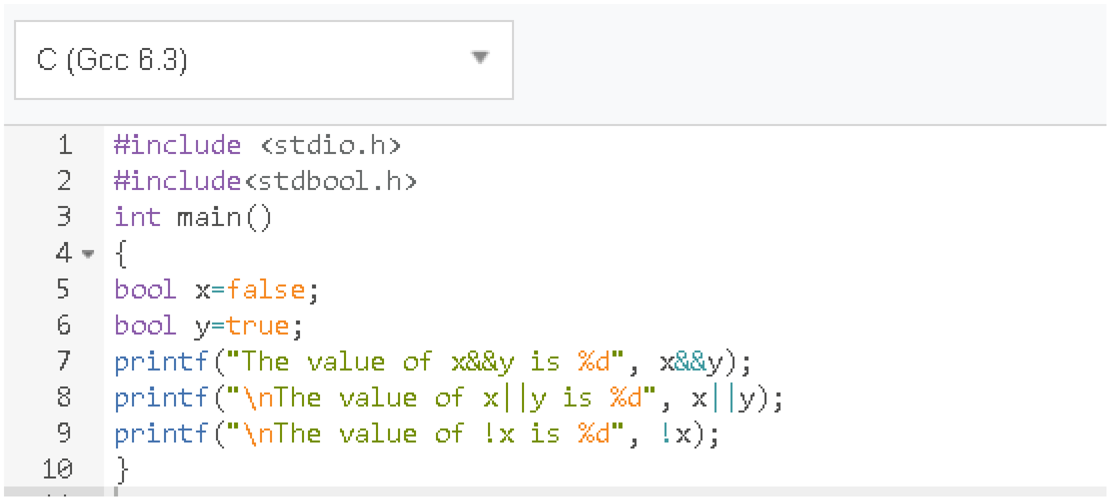
```
The value of x is false
```

Boolean with Logical Operators

The Boolean type value is associated with logical operators. There are three types of logical operators in the C language:

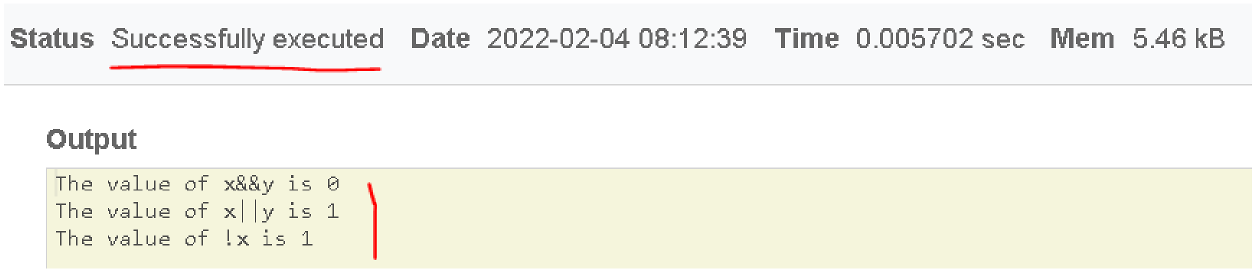
- &&(AND Operator): It is a logical operator that takes two operands. If the value of both the operands are true, then this operator returns true otherwise false
- ||(OR Operator): It is a logical operator that takes two operands. If the value of both the operands is false, then it returns false otherwise true.
- !(NOT Operator): It is a NOT operator that takes one operand. If the value of the operand is false, then it returns true, and if the value of the operand is true, then it returns false.

LAB: Let's understand through an example.



```c
#include <stdio.h>
#include<stdbool.h>
int main()
{
bool x=false;
bool y=true;
printf("The value of x&&y is %d", x&&y);
printf("\nThe value of x||y is %d", x||y);
printf("\nThe value of !x is %d", !x);
}
```

Result/Output:



Status Successfully executed **Date** 2022-02-04 08:12:39 **Time** 0.005702 sec **Mem** 5.46 kB

Output

```
The value of x&&y is 0
The value of x||y is 1
The value of !x is 1
```

VI. Static in C

Static is a keyword used in C programming language. It can be used with both variables and functions, i.e., we can declare a static variable and static function as well. An ordinary variable is limited to the scope in which it is defined, while the scope of the static variable is throughout the program.

Static keyword can be used in the following situations:

- **Static global variable**

When a global variable is declared with a static keyword, then it is known as a static global variable. It is declared at the top of the program, and its visibility is throughout the program.

- **Static function**

When a function is declared with a static keyword known as a static function. Its lifetime is throughout the program.

- **Static local variable**

When a local variable is declared with a static keyword, then it is known as a static local variable. The memory of a static local variable is valid throughout the program, but the scope of visibility of a variable is the same as the automatic local variables. However, when the function modifies the static local variable during the first function call, then this modified value will be available for the next function call also.

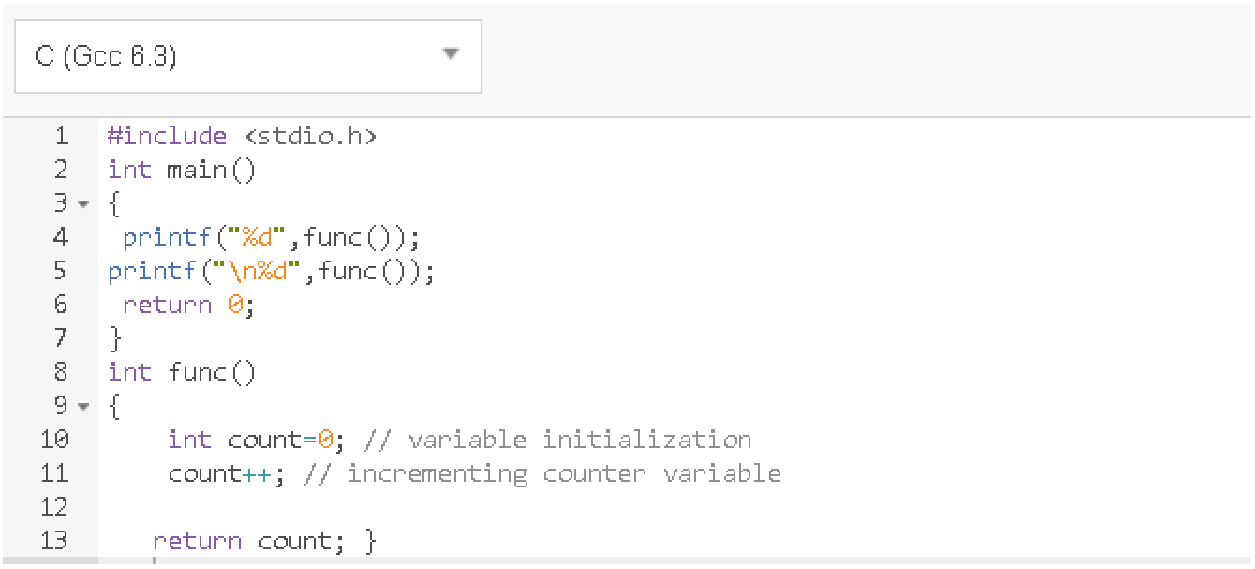
- **Static member variables**

When the member variables are declared with a static keyword in a class, then it is known as static member variables. They can be accessed by all the instances of a class, not with a specific instance.

- **Static method**

The member function of a class declared with a static keyword is known as a static method. It is accessible by all the instances of a class, not with a specific instance.

LAB: Let's understand through an example.

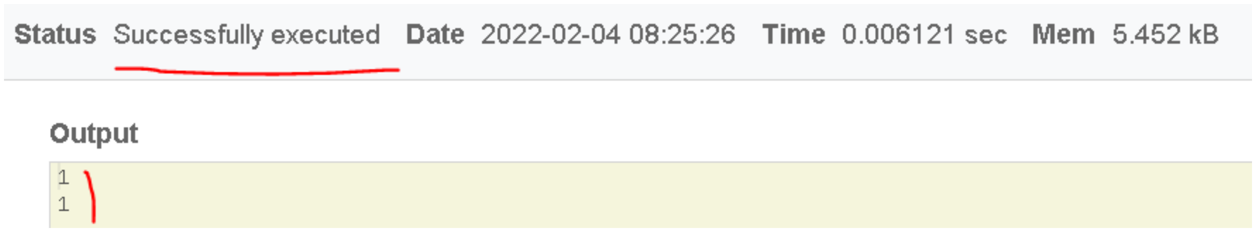


```c
#include <stdio.h>
int main()
{
 printf("%d",func());
printf("\n%d",func());
 return 0;
}
int func()
{
    int count=0; // variable initialization
    count++; // incrementing counter variable

   return count; }
```

In the above code, the func() function is called. In func(), count variable gets updated. As soon as the function completes its execution, the memory of the count variable will be removed. If we do not want to remove the count from memory, then we need to use the count variable as static. If we declare the variable as static, then the variable will not be removed from the memory even when the function completes its execution.

Result/Output:



Status Successfully executed **Date** 2022-02-04 08:25:26 **Time** 0.006121 sec **Mem** 5.452 kB

Output

```
1
1
```

Static variable

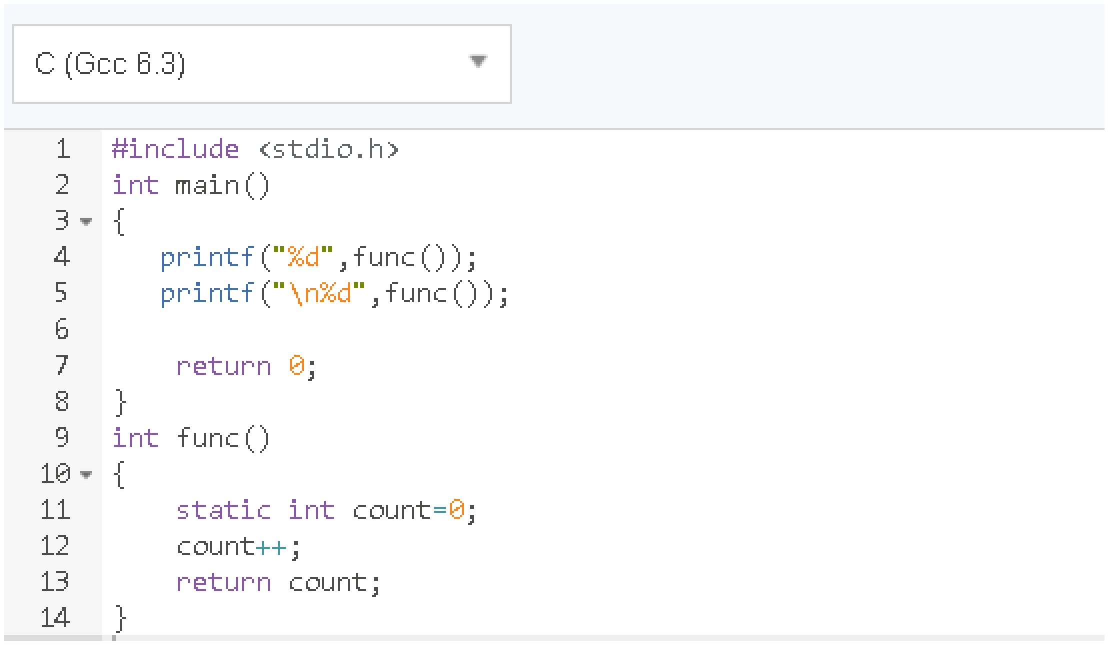
A static variable is a variable that persists its value across the various function calls.

Syntax.

The syntax of a static variable is given below:

```
static data_type variable_name;
```

LAB: Let's look at a simple example of static variable.



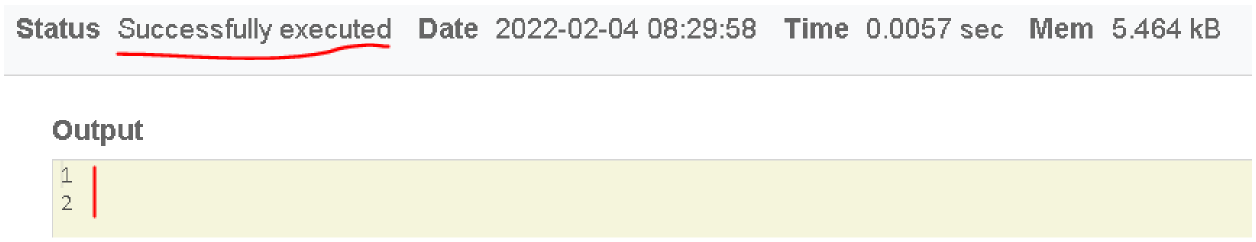
C (Gcc 6.3)

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

     return 0;
}
int func()
{
     static int count=0;
     count++;
     return count;
}
```

In the above code, we have declared the count variable as static. When the func() is called, the value of count gets updated to 1, and during the next function call, the value of the count variable becomes 2. Therefore, we can say that the value of the static variable persists within the function call.

Result/Output:



```
Status  Successfully executed   Date  2022-02-04 08:29:58   Time  0.0057 sec   Mem  5.464 kB
```

Output

```
1
2
```

Static Function

As we know that non-static functions are global by default means that the function can be accessed outside the file also, but if we declare the function as static, then it limits the function scope. The static function can be accessed within a file only.

The static function would look like as:

```
static void func()
{
    printf("Hello World!");
}
```

<u>Differences b/w static and global variable</u>

Global variables are the variables that are declared outside the function. These global variables exist at the beginning of the program, and its scope remains till the end of the program. It can be accessed outside the program also.

Static variables are limited to the source file in which they are defined, i.e., they are not accessible by the other source files.

Both the static and global variables have static initialization. Here, static initialization means if we do not assign any value to the variable then by default, 0 value will be assigned to the variable.

<u>Differences b/w static local and static global variable</u>

Static global variable

If the variable declared with a static keyword outside the function, then it is known as a static global variable. It is accessible throughout the program.

Static local variable

The variable with a static keyword is declared inside a function is known as a static local variable. The scope of the static local variable will be the same as the automatic local variables, but its memory will be available throughout the program execution. When the function modifies the value of the static local variable during one function call, then it will remain the same even during the next function call.

<u>Properties of a static variable</u>

The following are the properties of a static variable:

- The memory of a static variable is allocated within a static variable.
- Its memory is available throughout the program, but the scope will remain the same as the automatic local variables. Its value will persist across the various function calls.
- If we do not assign any value to the variable, then the default value will be 0.
- A global static variable cannot be accessed outside the program, while a global variable can be accessed by other source files.