******

3/20/2022

**COMSATS University Islamabad**

**Abbottabad Campus**

PROGRAMMING FUNDAMENTALS

ASSIGNMENT 01

NAME:

SYED SHAH HUSSAIN BADSHAH

( FA21-BSE-172)

SUBMITTED TO:

SIR YASHER ALI

Table of Contents

[**OPERATORS IN C LANGUAGE:** 2](#_Toc99481740)

[RELATIONAL OPERATORS: 2](#_Toc99481741)

[ EQUAL TO OPERATOR (==): 2](#_Toc99481742)

[ NOT EQUAL TO OPERATOR (!=): 2](#_Toc99481743)

[ LESS THAN OPERATOR (<): 2](#_Toc99481744)

[ GREATER THAN OPERATOR (>): 3](#_Toc99481745)

[ LESS THAN EQUAL TO OPERATOR (<=): 3](#_Toc99481746)

[ GREATER THAN EQUAL TO OPERATOR (>=): 3](#_Toc99481747)

[LOGICAL OPERATORS: 6](#_Toc99481758)

[ LOGICAL AND: 6](#_Toc99481759)

[ LOGICAL OR: 6](#_Toc99481760)

[ LOGICAL NOT: 6](#_Toc99481761)

[ BITWISE AND OPERATOR (&): 7](#_Toc99481762)

[ BITWISE OR OPERATOR (|): 7](#_Toc99481763)

[ BITWISE XOR OPERATOR(^): 8](#_Toc99481764)

[ BITWISE COMPLEMENT(~): 8](#_Toc99481765)

[ BITWISE SHIFT OPERATORS(>>,<<): 8](#_Toc99481766)

[UNARY OPERATORS: 9](#_Toc99481767)

[ UNARY MINUS (-): 9](#_Toc99481768)

[ UNARY PLUS (+): 10](#_Toc99481769)

[UNARY INCREMENT OPERATOR: 11](#_Toc99481770)

[ PRE INCREMENT: 11](#_Toc99481771)

[ POST INCREMENT: 11](#_Toc99481772)

[UNARY DECREMENT OPERATOR: 12](#_Toc99481773)

[ PRE DECREMENT: 12](#_Toc99481774)

[ POST DECREMENT: 13](#_Toc99481775)

[UNARY SIZE OF () OPERATOR: 13](#_Toc99481776)

[LOGICAL NOT (!): 14](#_Toc99481777)

[ADDRESS OF OPERATOR (&): 15](#_Toc99481778)

[ARITHMITICAL OPERATORS: 16](#_Toc99481779)

[INPUT: 18](#_Toc99481780)

[OUTPUT: 18](#_Toc99481781)

[ Printf() OUTPUT STATEMENT: 18](#_Toc99481782)

[ Scanf() INPUT STATEMENT: 19](#_Toc99481783)

[ CHAR: 20](#_Toc99481784)

[ INT: 20](#_Toc99481785)

[ FLOAT: 20](#_Toc99481786)

[ DOUBLE: 20](#_Toc99481787)

[EXAMPLE: 21](#_Toc99481788)

***QUESTION 02:***

# **OPERATORS IN C LANGUAGE:**

* Unary operators.
* Bitwise operators.
* Logical operators.
* Relational operators.

RELATIONAL OPERATORS:

The relational operators checks the relationship between two operands.

* EQUAL TO OPERATOR (==):

It is used to compare both operands and returns 1 if both are equal or the same, and 0 represents the operands that are not equal.

* NOT EQUAL TO OPERATOR (!=):

The Not Equal To Operator is the opposite of the Equal To Operator and is represented as the (!=) operator. The Not Equal To Operator compares two operands and returns 1 if both operands are not the same; otherwise, it returns 0.

* LESS THAN OPERATOR (<):

It is used to check whether the value of the left operand is less than the right operand, and if the statement is true, the operator is known as the Less than Operator.

* GREATER THAN OPERATOR (>):

The operator checks the value of the left operand is greater than the right operand, and if the statement is true, the operator is said to be the Greater Than Operator.

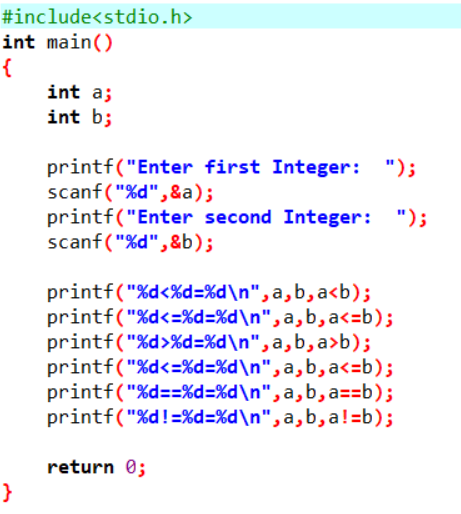
* LESS THAN EQUAL TO OPERATOR (<=):

The operator checks whether the value of the left operand is less than or equal to the right operand, and if the statement is true, the operator is said to be the Less than Equal To Operator.

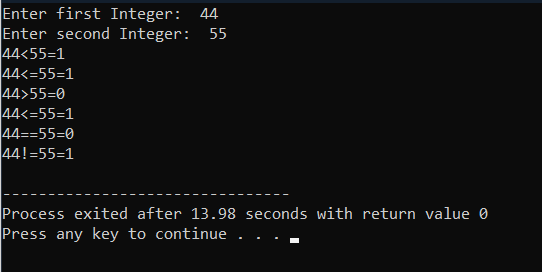
* GREATER THAN EQUAL TO OPERATOR (>=):

The operator checks whether the left operand's value is greater than or equal to the right operand. If the statement is true, the operator is said to be the Greater than Equal to Operator.

EXAMPLE:



OUTPUT:



COMPOUND ASSIGNEMENT OPERATORS:

Following are the compound assignment operators.

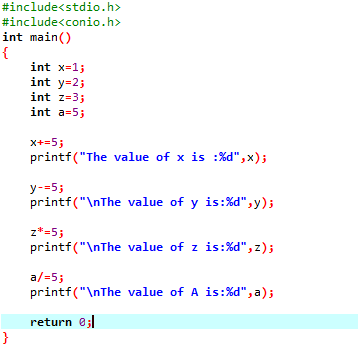
• +=

• -=

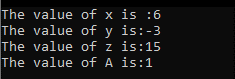
• \*=

• /=

For Example:



Output:



LOGICAL OPERATORS:

There are three logical operators:

* Logical AND.
* Logical OR.
* Logical NOT.
* LOGICAL AND:

If both the operands are non-zero, then the condition becomes true.

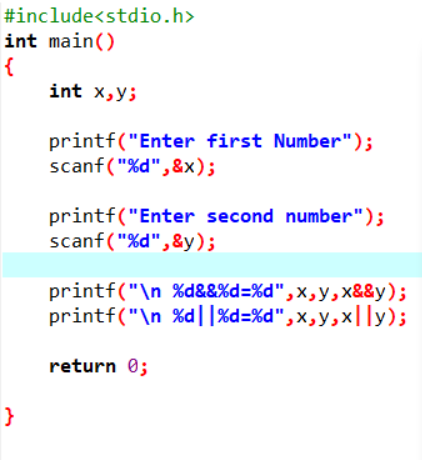
* LOGICAL OR:

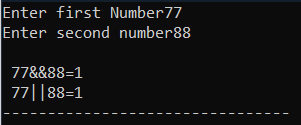
If any of the two operands is non-zero, then the condition becomes true.

* LOGICAL NOT:

It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

EXAMPLE:





BITWISE OPERATORS:

* BITWISE AND OPERATOR (&):

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

* BITWISE OR OPERATOR (|):

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

* BITWISE XOR OPERATOR(^):

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

* BITWISE COMPLEMENT(~):

Bitwise compliment operator is an unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by ~.

* BITWISE SHIFT OPERATORS(>>,<<):

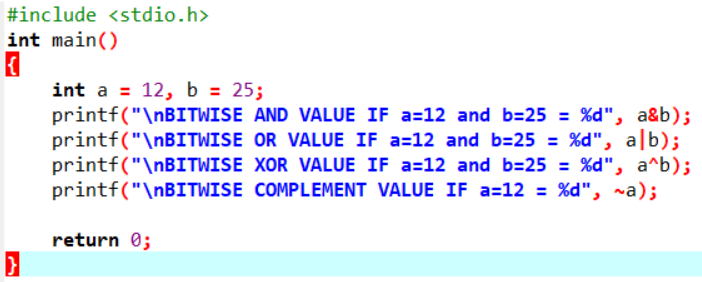
There are two shift operators in C programming:

* Right shift operator
* Left shift operator.

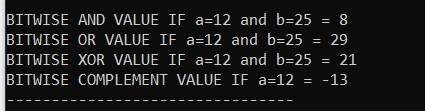
Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by >>.

Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is <<.

EXAMPLE:



OUTPUT:



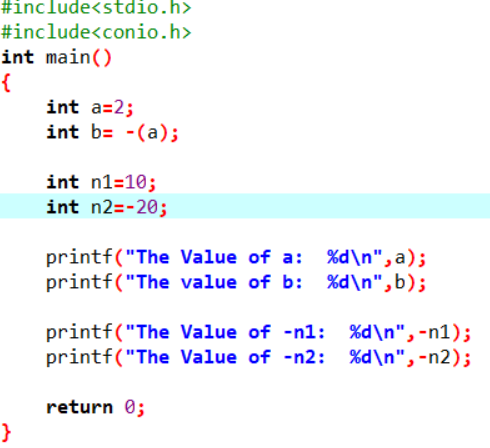
# UNARY OPERATORS:

Unary operators is an operator used to operate single operand to return a new value. In other words, it is an operator have equal value of an operand or expression value. In unary operator operators have equal priority from left to right side associativity.

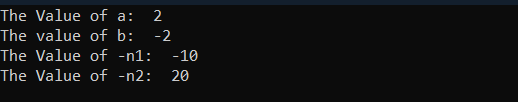
# UNARY MINUS (-):

The minus operator changes the sign of its arguments. A positive number becomes negative and a negative number becomes positive.

EXMAPLE :



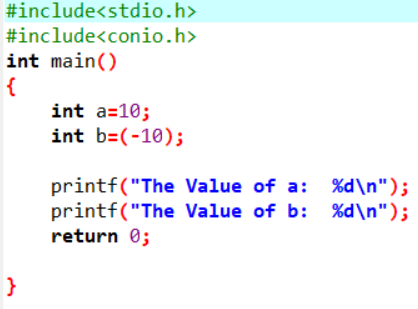
OUTPUT:



# UNARY PLUS (+):

The unary plus operator is represented as “+” symbol and it does not change to the operand value.

EXAMPLE :



OUTPUT:



# UNARY INCREMENT OPERATOR:

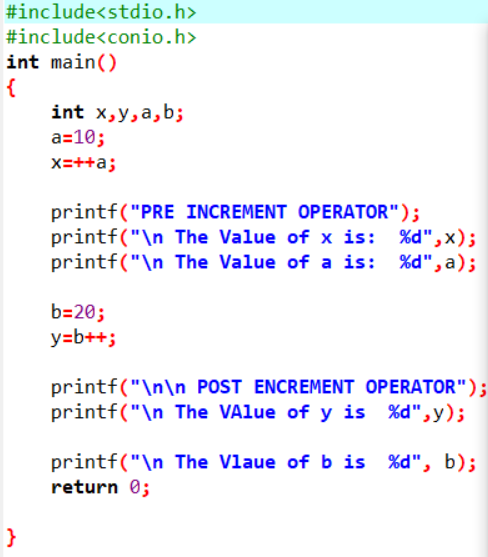
# PRE INCREMENT:

This operator is represented by (++a) and is also known as pre increments operator. Which means value is increment by 1 before using operand to the expression.

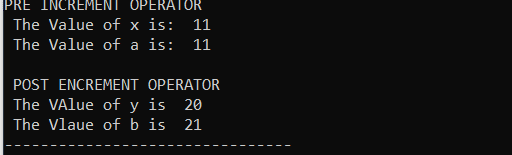
# POST INCREMENT:

The operator is represented by (a++) and is known as post increment operator which means the value of a is incremented by 1 after assigning the original value to the expression.

EXAMPLE:



OUTPUT:



# UNARY DECREMENT OPERATOR:

It is used to decrement the value of the variable by 1. The decrement can be done in two ways:

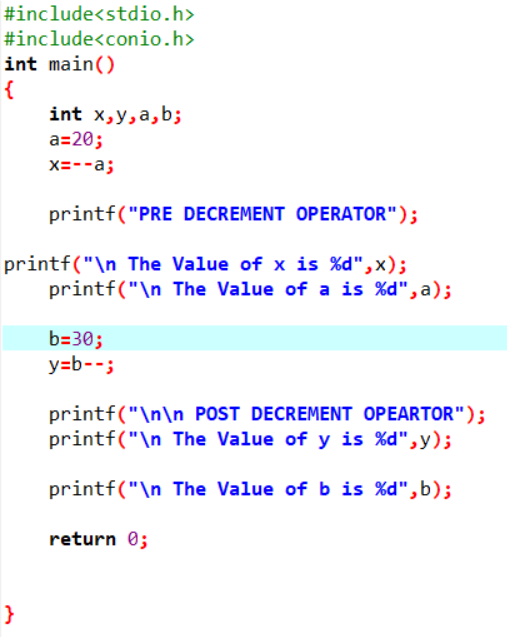
# PRE DECREMENT:

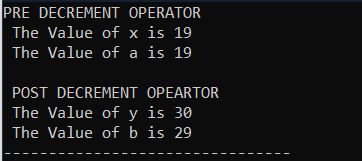
In this method the operator precedes (--a). The value of operand will be altered before it is used.

# POST DECREMENT:

In this method the operator follows the operand (a--). The value of operand will be altered after it is used.

EXAMPLE :

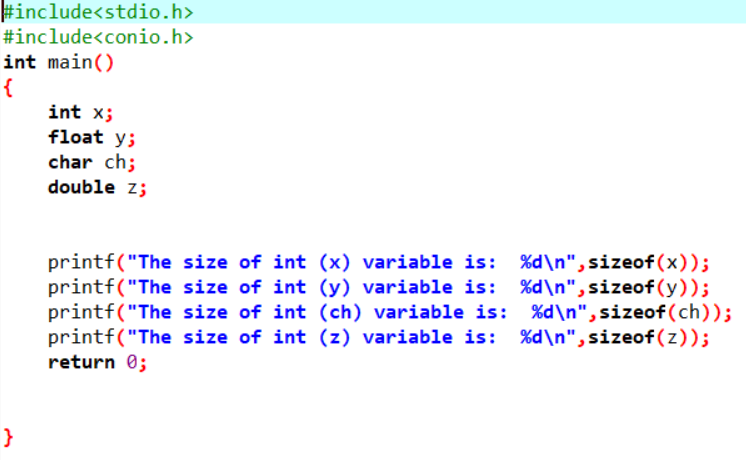
OUTPUT:



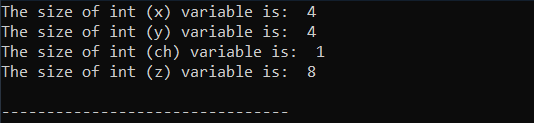
# UNARY SIZE OF () OPERATOR:

This operator returns the size of its operand, in bytes. The sizeof operator always precedes its operand.

EXAMPLE:



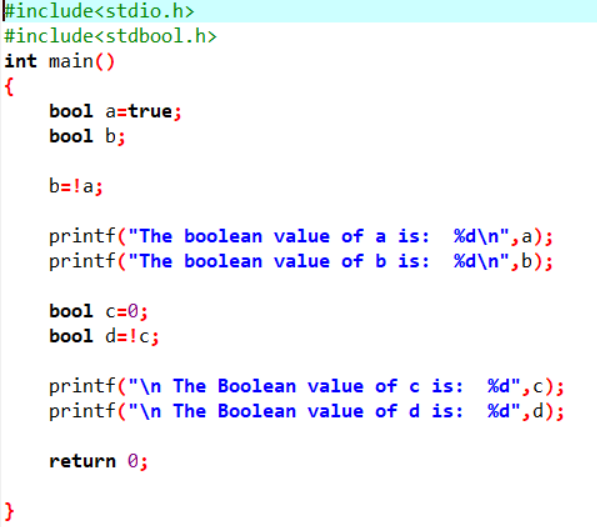
OUTPUT:



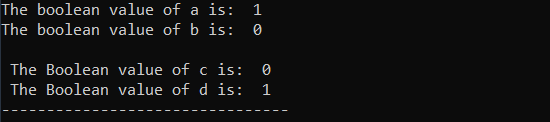
# LOGICAL NOT (!):

It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

EXAMPLE;



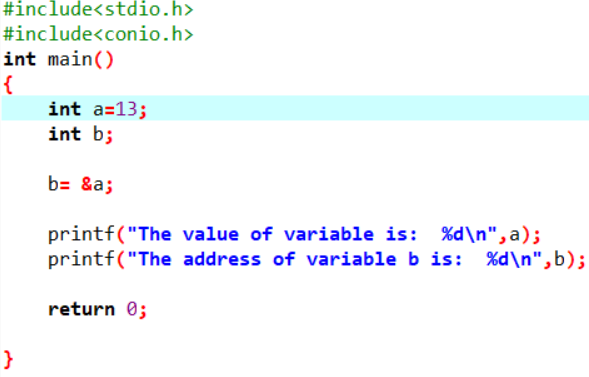
OUTPUT:



# ADDRESS OF OPERATOR (&):

It gives an address of a variable. It is used to return the memory address of a variable. These addresses returned by the address-of operator are known as pointers because they “point” to the variable in memory.

EXAMPLE:



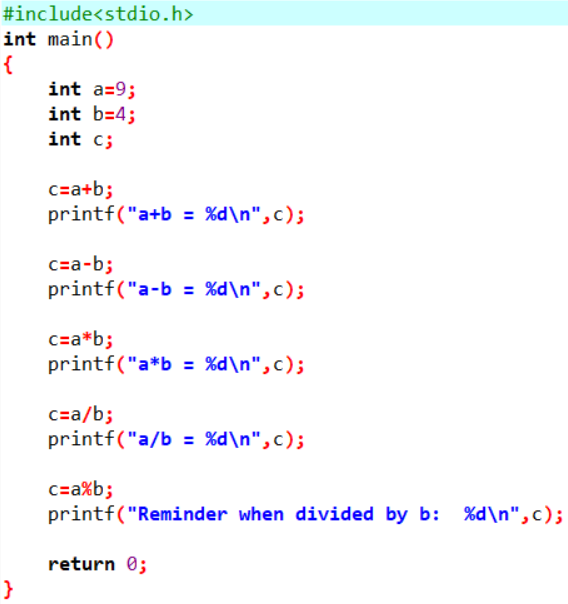
OUTPUT:



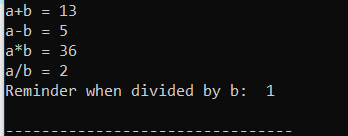
# ARITHMITICAL OPERATORS:

An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

| **Operator** | **Meaning of Operator** |
| --- | --- |
| + | addition or unary plus |
| - | subtraction or unary minus |
| \* | multiplication |
| / | Division |
| % | remainder after division (modulo division) |
| EXAMPLE: |  |



OUTPUT:



***QUESTION 03:***

# INPUT:

Input means to provide the program with some data to be used in the program.

# OUTPUT:

Output means to display the data on the screen.

The C programming language provides standard library functions to read any given input and to display data on the console.  
The functions used for standard input and output are present in the stdio.h header file. Hence to use the functions we need to include the stdio.h header file in our program.

#include<stdio.h>

Following are the functions used for standard input and output:

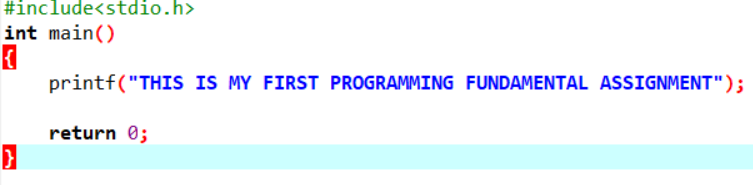
# Printf() OUTPUT STATEMENT:

printf() function shows output.

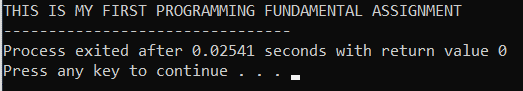
The printf() function is the most used function in c language. This function is defined in the stdio.h header file and is used to show output on the console.

This function is used to print a simple text sentence or value of any variable which can be of int, char, float, or any other datatype.

EXAMPLE;



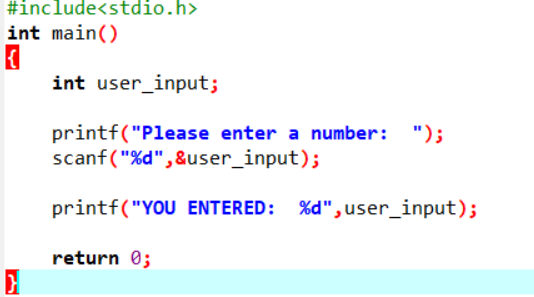
Output:

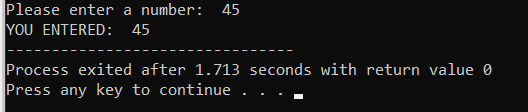


# Scanf() INPUT STATEMENT:

Scanf() function takes input:

If we have to take an integer value input from the user we have to define an integer variable and then use a scanf() function.





In the above code example, we have used %d format specifier to inform the scanf() function that user input will be of type integer.

***QUESTION 01:***

Each variable in C has an associated data type. Each data type requires different amounts of memory and has some specific operations which can be performed over it. Let us briefly describe them one by one:

# CHAR:

The most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

# INT:

As the name suggests, an int variable is used to store an integer.

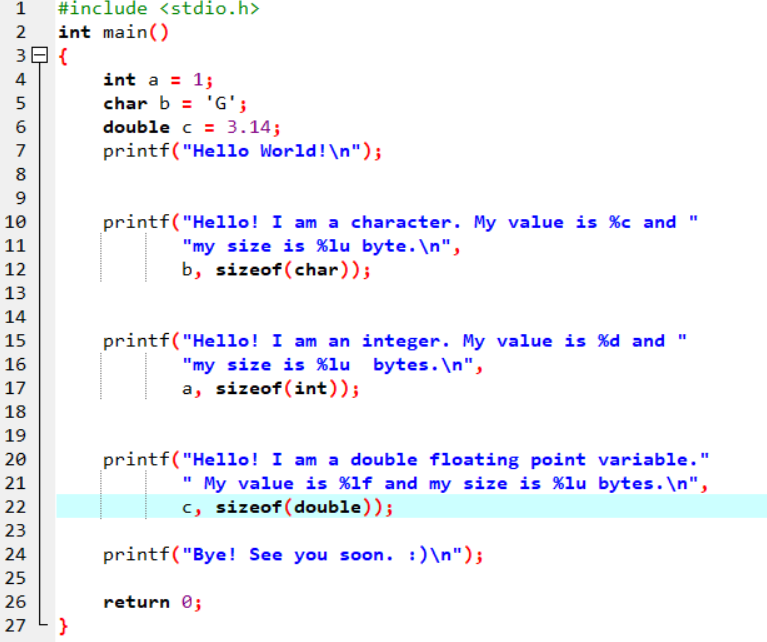
# FLOAT:

It is used to store decimal numbers (numbers with floating point value) with single precision.

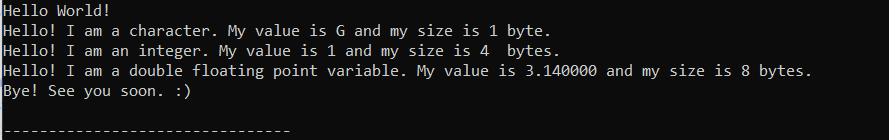
# DOUBLE:

It is used to store decimal numbers (numbers with floating point value) with double precision.

# EXAMPLE:



OUTPUT:



|  |  |
| --- | --- |
| **C Data types / storage Size** | **Range** |
| char / 1 | –127 to 127 |
| int / 2 | –32,767 to 32,767 |
| float / 4 | 1E–37 to 1E+37 with six digits of precision |
| double / 8 | 1E–37 to 1E+37 with ten digits of precision |
| long double / 10 | 1E–37 to 1E+37 with ten digits of precision |
| long int / 4 | –2,147,483,647 to 2,147,483,647 |
| short int / 2 | –32,767 to 32,767 |
| unsigned short int / 2 | 0 to 65,535 |
| signed short int / 2 | –32,767 to 32,767 |
| long long int / 8 | –(2power(63) –1) to 2(power)63 –1 |
| signed long int / 4 | –2,147,483,647 to 2,147,483,647 |
| unsigned long int / 4 | 0 to 4,294,967,295 |
| unsigned long long int / 8 | 2(power)64 –1 |

THE END:)