BUDDHA SERIES

(Unit Wise Solved Question & Answers)

Course – B.Tech (ASH)

College – Buddha Institute of Technology

(AKTU CODE-525)

Department: Applied Science and Humanities

Subject: Programming for Problem Solving (BCS-101/201)

Faculty Name: Shyam Mohan Singh

Unit – 2

Short Answer type question

1) Differentiate between implicit and explicit type conversion.

(B.TECH-(SEM I) 2022-23)

Answer:-

Implicit type conversion

- The **compiler** provides implicit type conversions when operands are of different data types.
- It is automatically done by the compiler by converting smaller data type into a larger data type.

Explicit type conversion

- Explicit type conversion is done by the user by using (type) operator.
- Before the conversion is performed, a runtime check is done to see if the destination type can hold the source value.

```
int a,c;
float b;
c = (int) a + b
```

2) Write the output of following code:

```
#include <stdio.h>
int main()
{
int a = -10, b = 20;
if(a > 0 && b < 0)
a++;
else if(a < 0 && b < 0)
a--;
else if(a < 0 && b > 0)
b--;
else
b--;
printf("%d\n",a + b);
return 0;
}
```

(B.TECH-(SEM II) 2022-23)

Answer: - 9

3) Write limitations of switch case.

(B.TECH-(SEM I) 2021-22)

Answer:-

- Exclusion of Floating-Point Constants: Switch statements lack the capability to incorporate floating-point constants in both the switch and case segments.
- Limitation on Variable Expressions: In switch statements, the usage of variable expressions within case conditions is not supported.
- Non-Repetition of Constants: It is not possible to employ the same constant in different cases within a switch statement.
- Absence of Relational Expressions: Switch statements do not allow the utilization of relational expressions in defining case conditions.

4) Write advantage of switch case.

(B.TECH-(SEM I) 2021-22)

Answer:-

- The switch statement is easier to read than if-else statements.
- It overcomes the challenges of the "if-else if" statement that makes compilation difficult because of deep nesting. The switch statement has a fixed depth.
- It allows the best-optimized implementation for faster code execution than the "if-else if" statement.
- It is easy to debug and maintain the programs using switch statements.
- The switch statement has faster execution power.

5) Describe syntax and working of ternary operator.

(B.TECH-(SEM I) 2021-22)

Answer:-

The ternary operator, also known as the conditional operator, is a powerful tool in C programming. The ternary operator takes three arguments: *a conditional expression*, *a result if the condition is* True, and *a result if the condition is* False. The ternary operator is represented by the "? :" symbol and can be used to simplify and make your code more readable and efficient.

Ternary Operator Syntax

Here's the syntax for a ternary operator in C:

condition ? true_val : false_val

6) Define all Arithmetic operators.

(B.TECH-(SEM I) 2020-21)

Answer:-

Operator	Name of the Operator	Arithmetic Operation	Syntax
+	Addition	Add two operands.	x **+** y
–	Subtraction	Subtract the second operand from the first operand.	x **–** y
*****	Multiplication	Multiply two operands.	x ***** y
/	Division	Divide the first operand by the second operand.	x ***/*** y
%	Modulus	Calculate the remainder when the first operand is divided by the second operand.	x **%** y

Long Answer type question

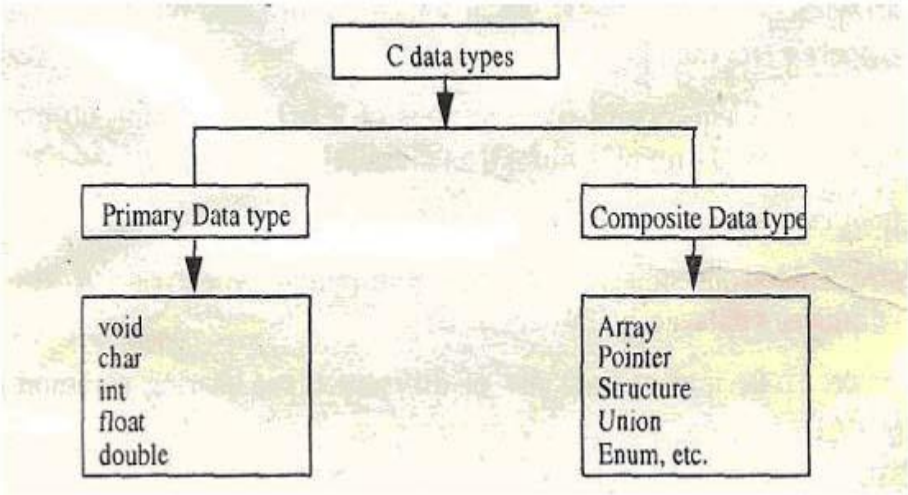
1) **Define Data Types in C. Discuss basic data types in terms of memory occupied, format specifier and range.**

(B.TECH-(SEM I) 2022-23, (SEM II)-2022-23)

Answer:-

C language is rich in its data types. C supports following classes of data types.

1. Primary/Fundamental data types
2. Derived data types/composite data types



All c compiler support five fundamental data types, namely integer (int), character (char), floating point (float), double-precision floating point (double) and void.

1. **Integer types:** integers are whole numbers with a range of values supported by a particular machine. The size of an integer that can be stored depends on the computer. if we use a 16 bit word length , the size of an integer value is limited to the range -32768 to +32767 (-2^{15} to $2^{15}-1$).

2. **Floating point types:** Floating point numbers are stored in 32 bits, with 6 digits of precision. Floating point numbers are defined in c by the keyword float .when the accuracy provided by a float number is not sufficient; the type double can be used to define the number.

3. **Double – precision floating point:** A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. Remember that

double type represents the same data type that float represents, but with greater precision. To extend the precision further, we may use long double which which uses 80 bits.

4. **Character types:** A single character can be defined as character (char) type data. Characters are usually stored in 8 bits (one byte) of internal storage. The qualifier signed or unsigned may be explicitly applied to char. While unsigned chars have values between 0 to 255, signed chars have values from -128 to 127.

5. **Void types:** the void type has no values. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

Size and range of data types

Data types	Range	Bytes	format
Signed char	-128 to 127	1	%c
Unsigned char	0 to 255	1	%c
Short signed int	-32768 to32767	2	%hd
Short unsigned int	0 to 65535	2	%hu
signed int	-32768 to32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483648	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4E38 to +3.4E38	4	%f
double	-1.7E308 to +1.7E308	8	%lf
long double	-1.7E4932 to +1.7E4932	10	%Lf

2) Explain various bitwise operators in C Language with help of an example. When precedence of two operators in an expression is same, how associativity helps in identifying which operator will be evaluated first. Illustrate it with an example.

(B.TECH-(SEM I) 2022-23, (SEM II)-2022-23)

Answer:-

In C, the following 6 operators are bitwise operators (also known as bit operators as they work at the bit-level). They are used to perform bitwise operations in C.

1. The & (bitwise AND) in C or C++ takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
2. The | (bitwise OR) in C or C++ takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
3. The ^ (bitwise XOR) in C or C++ takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.

4. The << (left shift) in C or C++ takes two numbers, the left shifts the bits of the first operand, and the second operand decides the number of places to shift.
5. The >> (right shift) in C or C++ takes two numbers, right shifts the bits of the first operand, and the second operand decides the number of places to shift.
6. The ~ (bitwise NOT) in C or C++ takes one number and inverts all bits of it.

Operator Precedence in C

Operator precedence determines which operation is performed first in an expression with more than one operator with different precedence.
Let's try to evaluate the following expression,

10 + 20 * 30

The expression contains two operators, + (plus)**, and** * (multiply). **According to the given table, the** * **has higher precedence than** + **so, the first evaluation will be**

10 + (20 * 30)

After evaluating the higher precedence operator, the expression is

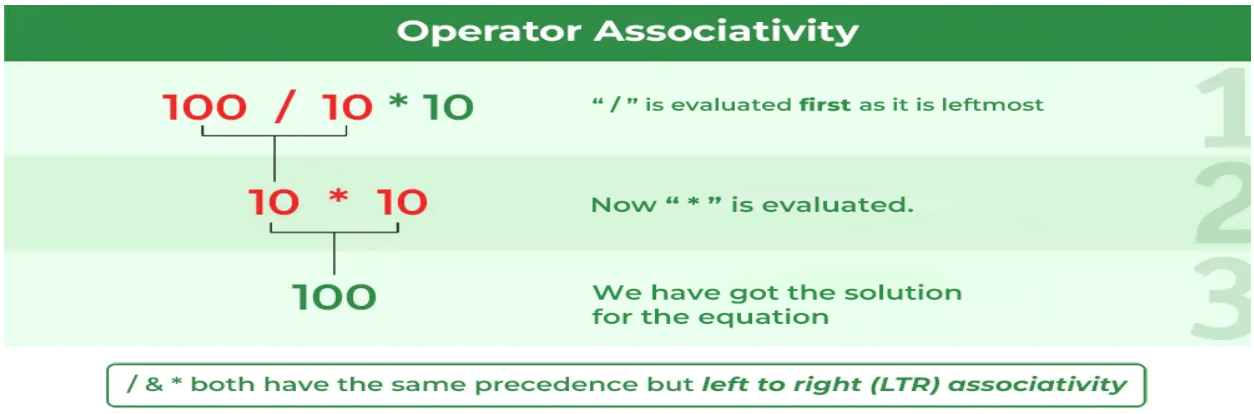
10 + 600

Now, the + operator will be evaluated.

610

Operator Associativity

Operator **associativity** is used when two operators of the same precedence appear in an expression. Associativity can be either from Left to Right or Right to Left.



Precedence	Operator	Description	Associativity
1	()	Parentheses (function call)	Left-to-Right

Precedence	Operator	Description	Associativity
	[]	Array Subscript (Square Brackets)	
	.	Dot Operator	
	->	Structure Pointer Operator	
	++ , —	Postfix increment, decrement	
	++ / —	Prefix increment, decrement	
	+ / –	Unary plus, minus	
	! , ~	Logical NOT, Bitwise complement	
2	**(type)**	Cast Operator	Right-to-Left
	*****	Dereference Operator	
	&	Addressof Operator	
	sizeof	Determine size in bytes	
3	***,/,%**	Multiplication, division, modulus	Left-to-Right
4	**+/-**	Addition, subtraction	Left-to-Right
5	**<< , >>**	Bitwise shift left, Bitwise shift right	Left-to-Right

Precedence	Operator	Description	Associativity
6	< , <=	Relational less than, less than or equal to	Left-to-Right
	> , >=	Relational greater than, greater than or equal to	
7	== , !=	Relational is equal to, is not equal to	Left-to-Right

3) Illustrate various types of storage classes in C-Language with suitable example.

(B.TECH-(SEM I) 2022-23, (SEM II)-2022-23)

Answer:-

A variable name identifies some physical location within the computer where the string of bits representing the variable's value is stored. There are basically two kinds of locations in a computer where such a value may be kept— Memory and CPU registers. Moreover, a variable's storage class tells us:

(a) Where the variable would be stored.

(b) What will be the initial value of the variable, if initial value is not specifically assigned.(i.e. the default initial value).

(c) What is the scope of the variable; i.e. in which functions the value of the variable would be available.

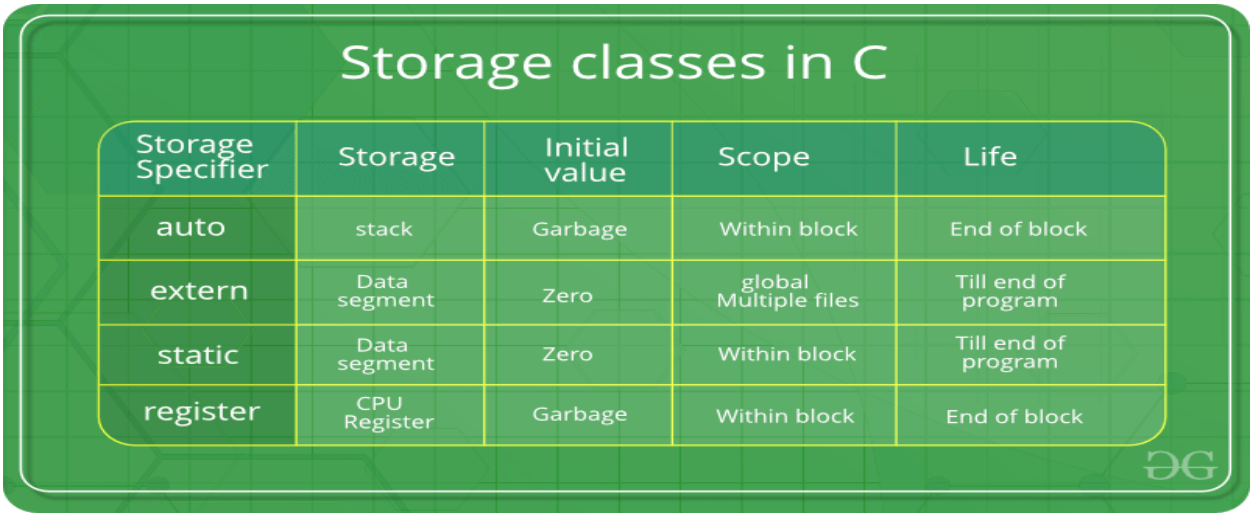
(d) What is the life of the variable; i.e. how long would the variable exist.

1. **Automatic Storage Class-**it is local variable known only to the function in which it is declared. (default is auto).

2. Register Storage Class- local variable which is stored in the register.

3. Static Storage Class- local variable which exists and retains its value even after the control is transferred to calling function.

4. External Storage Class- global variable known to all function in the file.



Automatic Variable

```
void main( )
{
auto int i = 1 ;
{
auto int i = 3 ;
printf ( "\n%d ", i ) ;
}
printf ( "%d", i ) ;
}
}
```

The output of the above program is: 31

External Variable

```
int i ;
void main( )
{
printf ( "\ni = %d", i ) ;
increment( ) ;
increment( ) ;
decrement( ) ;
decrement( ) ;
```

Static Variable

```
void main( )
{
 increment( ) ;
increment( ) ;
increment( ) ;
}

increment( )
{
static int i = 1 ;
 printf ( "%d\n", i ) ;
i=i+1;
}
```

Output:123

Register Variable

```
void main( )
{
register int i ;
for ( i = 1 ; i <= 10 ; i++ )
printf ( "\n%d", i ) ;
}
```

```
}
increment( )
{
i = i + 1 ;
printf ( "\non incrementing i = %d", i ) ;
}
decrement( )
{
i = i - 1 ;
printf ( "\non decrementing i = %d", i ) ;
}
```

The output would be:

```
i = 0
on incrementing i = 1
on incrementing i = 2
on decrementing i = 1
on decrementing i = 0
```

4) Illustrate the concept of type conversion and type casting with program?

(B.TECH-(SEM II) 2021-22)

Answer:-

Type casting is the process in which the compiler automatically converts one data type in a program to another one. Type conversion is another name for type casting. For instance, if a programmer wants to store a long variable value into some simple integer in a program, then they can type cast this long into the int. Thus, the method of type casting lets users convert the values present in any data type into another data type with the help of the cast operator, like:

(type_name) expression

Let us take a look at the following example to understand how we can utilise the cast operator for dividing an integer variable with another one by performing it as an operation of floating-point type:

```
#include <stdio.h>
main() {
int total = 17, values = 5;
double average;
average = (double) total / values;
printf("The average of all the values available with us is : %f\n", average );
}
```

The compilation and execution of the code mentioned here would generate the following output of the program:

The average of all the values available with us is : 3.400000

You must note here that the precedence of the cast operator is much higher than that of the division operator. Thus, the program would first convert the value of *total* into the *double* type. After this, it will finally divide this value with the help of count yielding of the double value.

As a matter of fact, the process of type conversion or type casting can be very implicit. It means that the compiler is capable of performing it automatically. Conversely, we can specify the data type explicitly using the cast operator. Both the methods are okay, but using the cast operator whenever in need is considered a better programming practice (explicit).

Syntax for Type Casting in C

```
int val_1;
float val_2;
// Body of program
val_2 = (float) val_1; // type casting of the values
```

Types of Type Casting in C

The process of type casting can be performed in two major types in a C program. These are:

- Implicit
- Explicit

C Language Implicit Type Casting

Implementing the implicit type casting is very easy in a program. We use it to convert the data type of any variable without losing the actual meaning that it holds. The implicit type casting occurs automatically.

In simpler words, the implicit type casting performs the conversion without altering any of the values stored in the program's variables.

Follow these points to understand what rules the implicit type casting follows in a C program:

- If we are performing a conversion on two of the different data types in a program, then the conversion of the lower data type to the higher data type will occur automatically.
- If we suppose that we are performing the type casting operation among two different data types like float and int, then the resultant value would be the floating data type (float).

Examples of Implicit Type Casting in C

```
int x = 4;
float a = 12.4, b;
b = a / x;
```

In the example given above, the variable x will get converted to the float data type (float) automatically, and it is comparatively a bigger data type in C. Here, the variable x and the variable a would be equal in terms of their data types. Thus, the value of b would be equal to 12.4/4.0=3.1.

Explicit Type Casting

This process is not at all like the implicit type casting in C, where the conversion of data type occurs automatically. Conversely, in the case of explicit type casting, the programmer needs to force the conversion. In simpler words, one has to perform type casting on the data types on their own.

Syntax:

```
(name_of_data_type) expression
```

Here, name_of_data_type refers to the name of that data type to which we want to convert the available data type in the code. This expression can be a constant, a variable, or even an actual expression in a program.

Examples of Explicit Type Casting in C

```
#include<stdio.h>
int main()
{
```

```
float num = 56.3;
int p = (int)num + 50; // data type casting explicitly
printf("Let us understand Explicit Type Casting in C\n");
printf("The value of the digit used is: %f\n", num);
printf("The value of the variable p is: %d\n",p);
return 0;
}
```

5) Define operator and operand? Discuss about various types of operators used in programming?

(B.TECH-(SEM II) 2021-22)

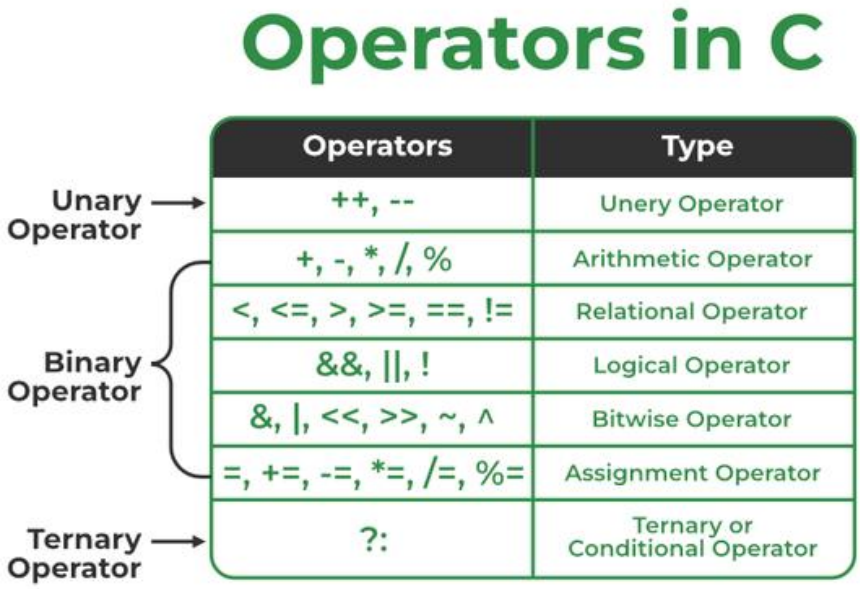
Answer:-

Operators can be defined as the symbols that help us to perform specific mathematical, relational, bitwise, conditional, or logical computations on operands. In other words, we can say that an operator operates the operands. For example, '+' is an operator used for addition, as shown below:

```
c = a + b;
```

Types of Operators in C

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Other Operators



1. **Arithmetic Operations in C**

These operators are used to perform arithmetic/mathematical operations on operands. Examples: (+, -, *, /, %,++,–). Arithmetic operators are of two types:

a) Unary Operators:

Operators that operate or work with a single operand are unary operators. For example: Increment(++) and Decrement(–) Operators

```
int val = 5;
cout<<++val;  // 6
```

b) **Binary Operators:**

Operators that operate or work with two operands are binary operators. For example: Addition(+), Subtraction(-), multiplication(*), Division(/) operators

```
int a = 7;
int b = 2;
cout<<a+b; // 9
```

2. Relational Operators in C

These are used for the comparison of the values of two operands. For example, checking if one operand is equal to the other operand or not, whether an operand is greater than the other operand or not, etc. Some of the relational operators are (==, >= , <=)(See this article for more reference).

```
int a = 3;
int b = 5;
cout<<(a < b);
// operator to check if a is smaller than b
```

3. **Logical Operator in C**

Logical Operators are used to combining two or more conditions/constraints or to complement the evaluation of the original condition in consideration. The result of the operation of a logical operator is a Boolean value either true or false.

For example, the logical AND represented as the '&&' operator in C returns true when both the conditions under consideration are satisfied. Otherwise, it returns false. Therefore, a && b returns true when both a and b are true (i.e. non-zero)(See this article for more reference).

```
cout<<((4 != 5) && (4 < 5));     // true
```

4. **Bitwise Operators in C**

The Bitwise operators are used to perform bit-level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands. Mathematical operations such as addition, subtraction, multiplication, etc. can be performed at the bit level for faster processing. For example, the bitwise AND operator represented as '&' in C takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1(True).

```
int a = 5, b = 9;   // a = 5(00000101), b = 9(00001001)
cout << (a ^ b);   //  00001100
cout <<(~a);       // 11111010
```

5. **Assignment Operators in C**

Assignment operators are used to assign value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type as the variable on the left side otherwise the compiler will raise an error.

Different types of assignment operators are shown below:
a) "="
This is the simplest assignment operator. This operator is used to assign the value on the right to the variable on the left.
Example:
a = 10;
b = 20;
ch = 'y';

6) What is use of break in C? Write a program to develop calculator using case in character format?

(B.TECH-(SEM II) 2021-22)

Answer:-

```
#include <stdio.h>

int main() {

  char op;
  double first, second;
  printf("Enter an operator (+, -, *, /): ");
  scanf("%c", &op);
  printf("Enter two operands: ");
  scanf("%lf %lf", &first, &second);

  switch (op) {
    case '+':
      printf("%.1lf + %.1lf = %.1lf", first, second, first + second);
      break;
    case '-':
      printf("%.1lf - %.1lf = %.1lf", first, second, first - second);
      break;
    case '*':
      printf("%.1lf * %.1lf = %.1lf", first, second, first * second);
      break;
    case '/':
      printf("%.1lf / %.1lf = %.1lf", first, second, first / second);
      break;
    // operator doesn't match any case constant
    default:
      printf("Error! operator is not correct");
  }

  return 0;
}
```

7) **What do you mean by Operands? Discuss the operator precedence and associativity of all the operators.**

(B.TECH-(SEM I) 2020-21)

Answer:-

Operands

In any operation, an operand may any numerical value (known as literals in programming languages), variables, constant, function calling etc on which program makes an operation.

Operator Precedence in C

Operator precedence determines which operation is performed first in an expression with more than one operator with different precedence.
Let's try to evaluate the following expression,

10 + 20 * 30

The expression contains two operators, + (plus)**, and** * (multiply). **According to the given table, the** * **has higher precedence than** + **so, the first evaluation will be**

10 + (20 * 30)

After evaluating the higher precedence operator, the expression is

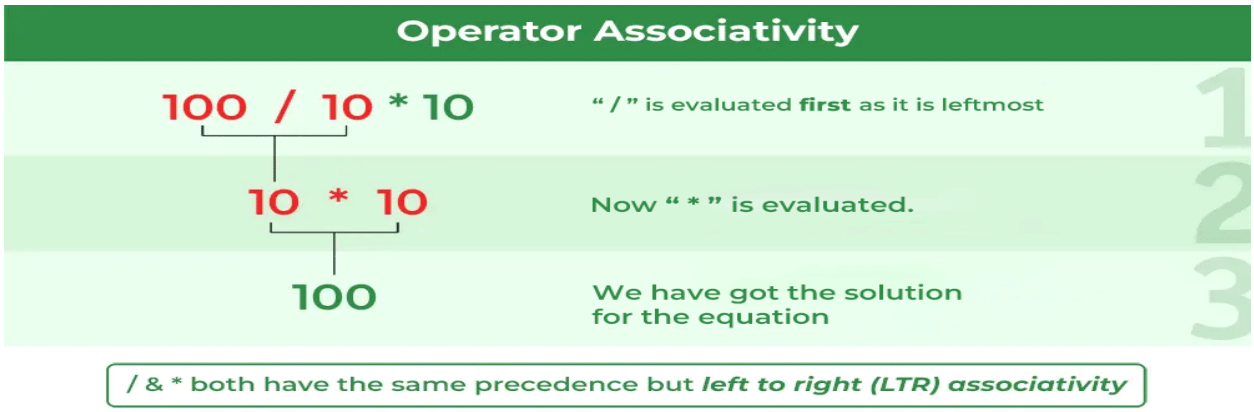
10 + 600

Now, the + operator will be evaluated.

610

Operator Associativity

Operator **associativity** is used when two operators of the same precedence appear in an expression. Associativity can be either from Left to Right or Right to Left.



Precedence	Operator	Description	Associativity
1	()	Parentheses (function call)	Left-to-Right
	[]	Array Subscript (Square Brackets)	
	.	Dot Operator	
	->	Structure Pointer Operator	
	++ , —	Postfix increment, decrement	
2	++ / —	Prefix increment, decrement	Right-to-Left
	+ / –	Unary plus, minus	
	! , ~	Logical NOT, Bitwise complement	
	(type)	Cast Operator	
	*	Dereference Operator	
	&	Addressof Operator	
	sizeof	Determine size in bytes	
3	*,/,%	Multiplication, division, modulus	Left-to-Right
4	+/-	Addition, subtraction	Left-to-Right

Precedence	Operator	Description	Associativity
5	<< , >>	Bitwise shift left, Bitwise shift right	Left-to-Right
6	< , <=	Relational less than, less than or equal to	Left-to-Right
	> , >=	Relational greater than, greater than or equal to	
7	== , !=	Relational is equal to, is not equal to	Left-to-Right

8) Write a program to find out the greatest number out of three numbers.

(B.TECH-(SEM I) 2020-21)

Answer:-

```
#include <stdio.h>

int main() {

double n1, n2, n3;

 printf("Enter three different numbers: ");
scanf("%lf %lf %lf", &n1, &n2, &n3);
  if (n1 >= n2 && n1 >= n3)
 printf("%.2f is the largest number.", n1);

  if (n2 >= n1 && n2 >= n3)
printf("%.2f is the largest number.", n2);


if (n3 >= n1 && n3 >= n2)
```

```
printf("%.2f is the largest number.", n3);

 return 0;
}
```