**“Coding C Language”**

**ASSIGNMENT**

**III**

**Prepared by:** Srijal Dangol **Submitted To:** Ashim Sir

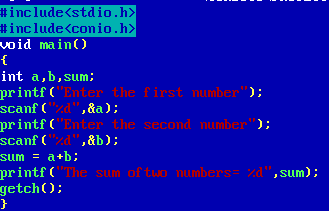
**Roll Number:** 17 **Subject:** C Programming

**Shift:** Morning

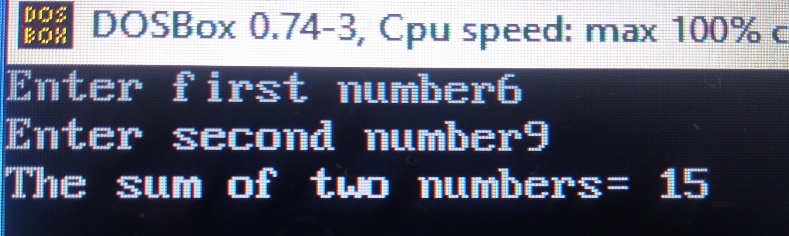
**BscCSIT079**

1. **WAP to display sum of two numbers.**

**Code:**

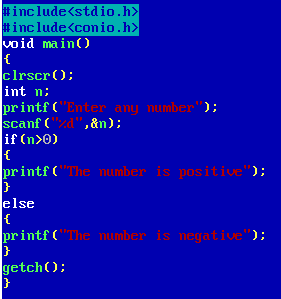
****

**Output:**

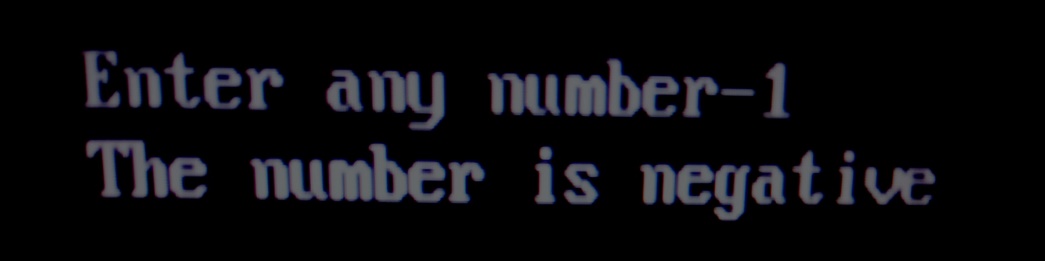
****

1. **WAP to display if the number is positive or negative.**

**Code:**

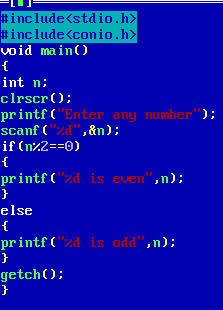
****

**Output:**

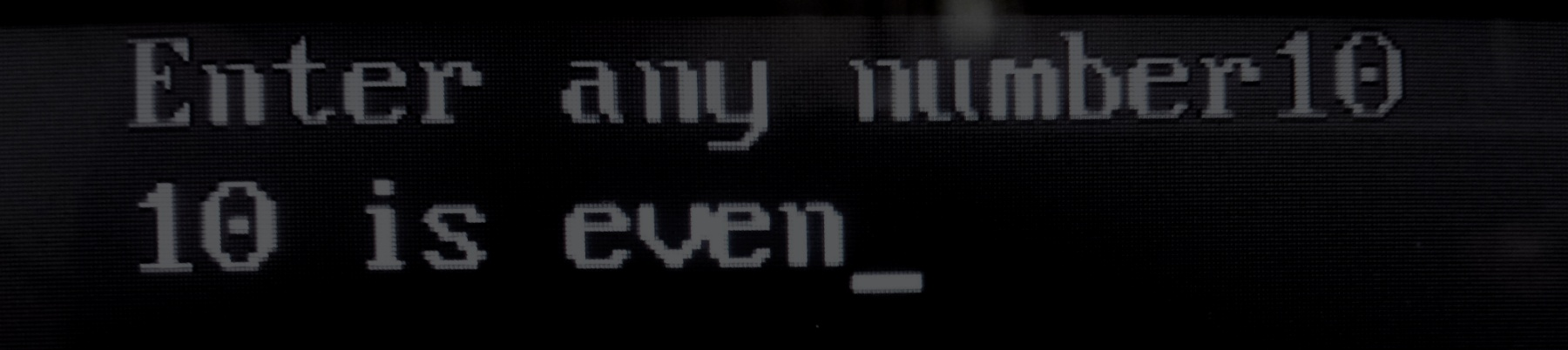
****

1. **WAP to display whether the number is even or odd.**

**Code:**

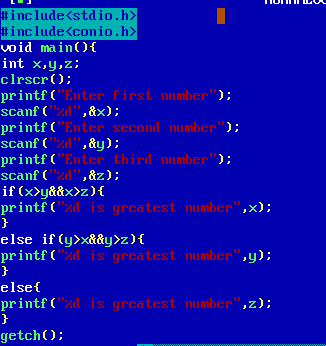
****

**Output:**



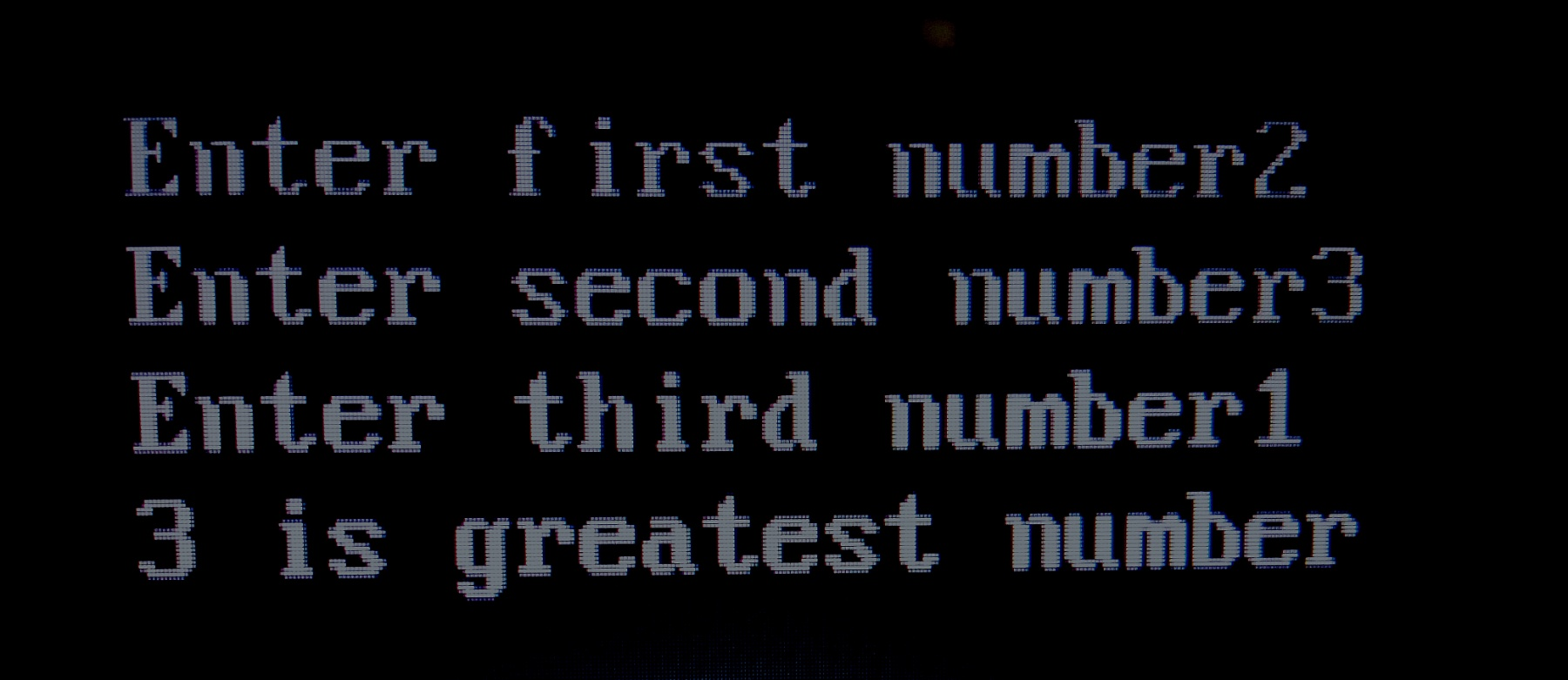
1. **WAP to read three numbers and print the greatest number.**

**Code:**

****

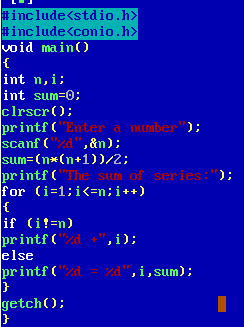
****

**Output:**

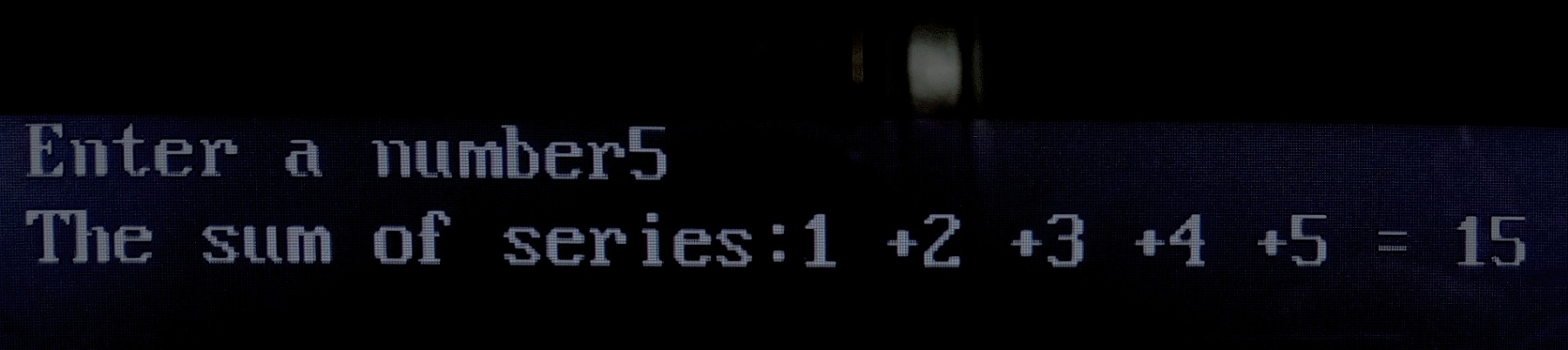


1. **WAP to find the sum of the series 1+2+3+4….. up to entered n numbers.**

**Code:**

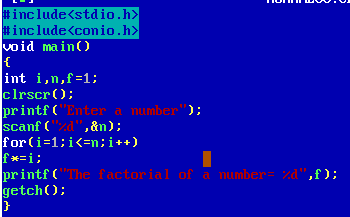
****

**Output:**

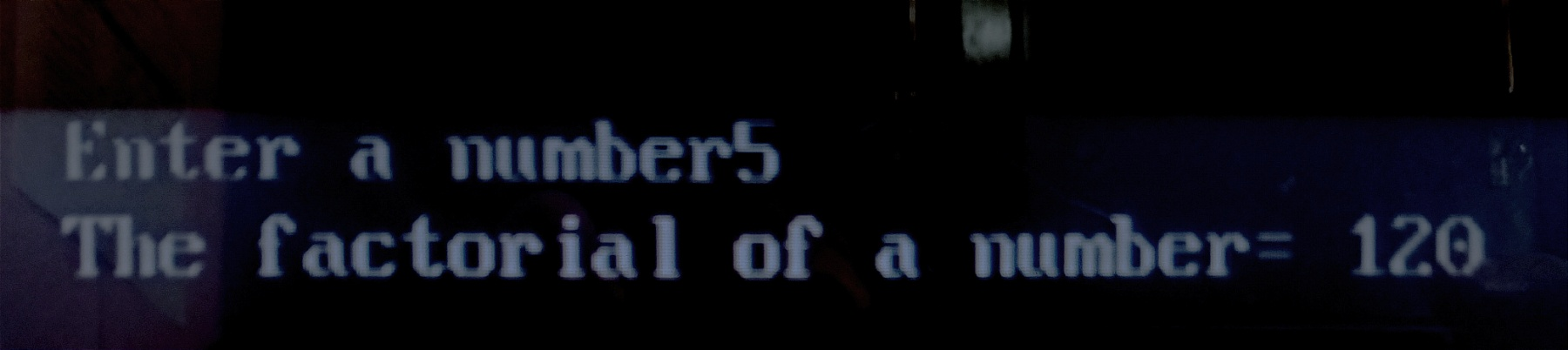


1. **WAP to display factorial of a given number N.**

**Code:**

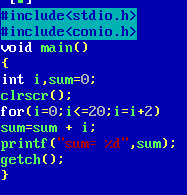
****

**Output:**

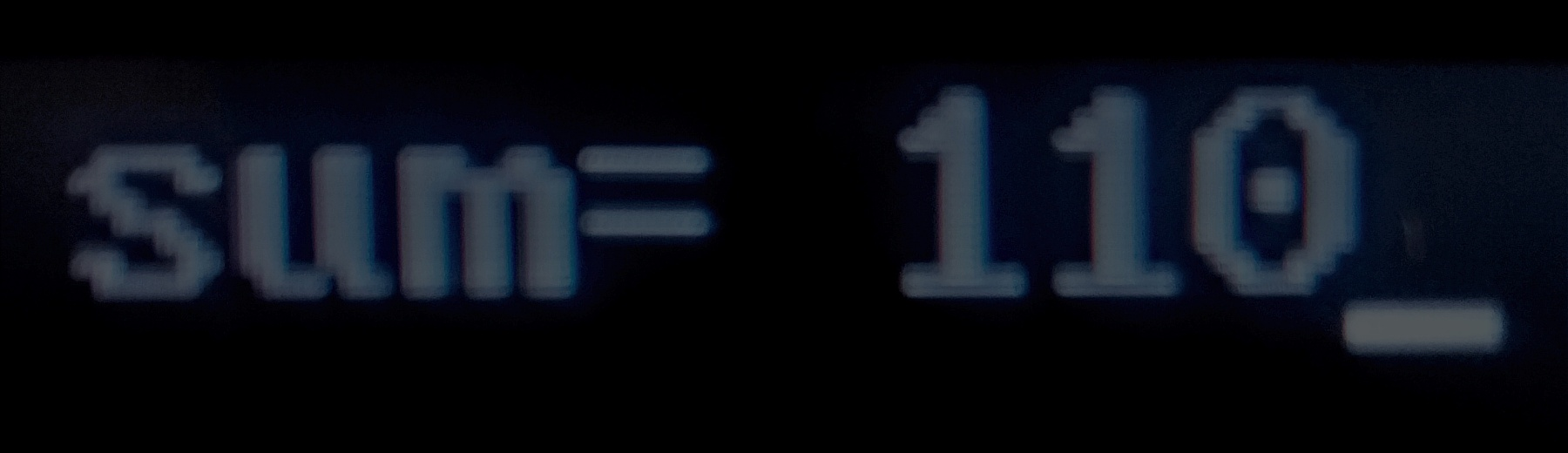


1. **WAP to read first 20 number and display only sum of even numbers.**

**Code:**

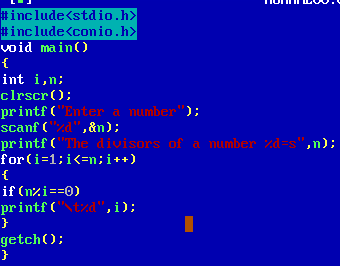
****

**Output:**

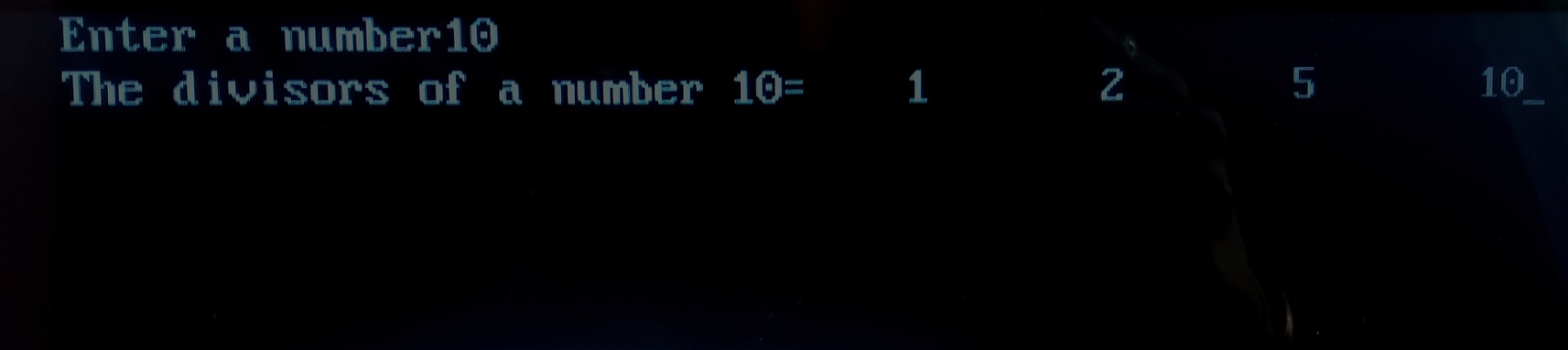


1. **WAP to read a number n and display all of its divisors.**

**Code:**

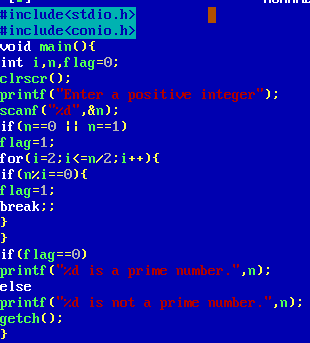
****

**Output:**

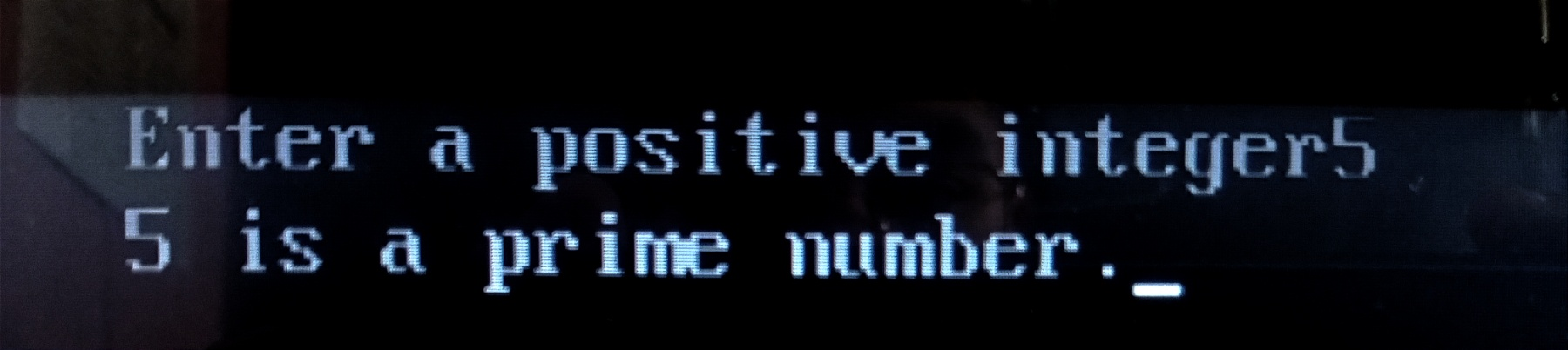


1. **WAP which displays if a number is prime or not.**

**Code:**

****

**Output:**



1. **What do you mean by Precedence and Associativity Of Operators in C. Explain with some examples.**

* The precedence of operators in C dictates the order in which the operators will be evolved in an expression. Associativity, on the other hand, defines the order in which the operators of the same precedence will be evaluated in an expression. Also, associativity can occur from either right to left or left to right.

**Use of the Operator Precedence and Associativity in C**

Precedence and Associativity are two of the characters that are used for operators in an expression for determining the order of sub-expressions when they do not have brackets.

**Operator Precedence**

Operator precedence helps us determine which of the operators in an expression must be evaluated first in case the expression consists of more than a single operator.

*Example,*

50 – 2 \* 15 is going to yield 20. It is because it gets evaluated as 50 – (2 \* 15), and not as (50 – 2) \* 15. The reason here is that subtraction (-) has lower precedence as compared to multiplication (\*).

**Operator Associativity**

We use associativity when two or more than two operators with the same precedence are present in the same expression.

*Example,*

The precedence of Division and Multiplication arithmetic operators is the same. So, let’s say we have an expression with us which is 6 \* 3 / 20. The evaluation of this expression would be (6 \* 3) / 20 because the associativity will be left to right for both the operators – multiplication and division. In a similar case, the calculation of 40 / 4 \* 5 would be (40 / 4) \* 5 because the associativity would be from right to left here as well.

Here is how the operator precedence and associativity work in the C language:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description of Operator** | **Associativity** |
| . | Direct member selection | Left to right |
| -> | Indirect member selection | Left to right |
| [] | Array element reference | Left to right |
| () | Functional call | Left to right |
| ~ | Bitwise(1’s) complement | Right to left |
| ! | Logical negation | Right to left |
| – | Unary minus | Right to left |
| + | Unary plus | Right to left |
| — | Decrement | Right to left |
| ++ | Increment | Right to left |
| \* | Pointer reference | Right to left |
| & | Dereference (Address) | Right to left |
| (type) | Typecast (conversion) | Right to left |
| sizeof | Returns the size of an object | Right to left |
| % | Remainder | Left to right |
| / | Divide | Left to right |
| \* | Multiply | Left to right |
| – | Binary minus (subtraction) | Left to right |
| + | Binary plus (Addition) | Left to right |
| >> | Right shift | Left to right |
| << | Left shift | Left to right |
| > | Greater than | Left to right |
| < | Less than | Left to right |
| >= | Greater than or equal | Left to right |
| <= | Less than or equal | Left to right |
| == | Equal to | Left to right |
| != | Not equal to | Left to right |
| ^ | Bitwise exclusive OR | Left to right |
| & | Bitwise AND | Left to right |
| || | Logical OR | Left to right |
| | | Bitwise OR | Left to right |
| ?: | Conditional Operator | Right to left |
| && | Logical AND | Left to right |
| , | Separator of expressions | Left to right |
| = | Simple assignment | Right to left |
| /= | Assign quotient | Right to left |
| \*= | Assign product | Right to left |
| %= | Assign remainder | Right to left |
| -= | Assign difference | Right to left |
| += | Assign sum | Right to left |
| |= | Assign bitwise OR | Right to left |
| ^= | Assign bitwise XOR | Right to left |
| &= | Assign bitwise AND | Right to left |
| >>= | Assign right shift | Right to left |
| <<= | Assign left shift | Right to left |