Operators :

i) Arithmetic operators   + * - / %

ii) Relational operators != == < > <= >=

iii) Logical operators && ||

iv) Bitwise operators & | ^

v) Function calling operators ()

vi) Memory dereferencing operators &(address of) *(value at)

vii) Association/Assignment operator c=a+b

viii) Conditional / Tertiary operator ?:

comma ,operators can be used for separation

[] array operator

Arithmatic operators

+ - * / %(modulus)

int a=50, b=6,c;

c=a+b

c=a-b

c=a*b

c=a/b // 8.33 (8) division operator can be used to find quotient

c=a%b // it is used to find the remainder

c=a%b=50%6= 2

```
#include<stdio.h>
//#include<conio.h>
void main()
{
int a, b,c;
printf("Enter value for a and b\n");
scanf("%d %d", &a,&b);
c=a+b;
printf("\naddition of %d and %d is=%d", a,b,c);
printf("\nSubstraction of %d and %d is =%d",a,b,a-b);
printf("\nMultiplication of %d and %d is =%d",a,b,a*b);
printf("\ndivision of %d and %d is =%d",a,b,a/b);
printf("\nRemainder of %d and %d is =%d",a,b,a%b);
//getch();
}
```

```
#include <stdio.h>
int main()
{
int a, b,c;
printf("Enter value for a and b\n");
scanf("%d %d", &a,&b);
c=a+b;
printf("\naddition of %d and %d is=%d", a,b,c);
c=a-b;
printf("\nSubstraction of %d and %d is =%d",a,b,c);
c=a*b;
printf("\nMultiplication of %d and %d is =%d",a,b,c);
c=a/b;
printf("\ndivision of %d and %d is =%d",a,b,c);
c=a%b;
printf("\nRemainder of %d and %d is =%d",a,b,c);
return 0;
}
```

Relational Operators(if , if else, while, for)

!= not equal to

== comparison

< less than

> greater than

<= less than or equal to

>= greater than or equal to

float per;

int age;

age>=18 vote

per>=75 distinction

per>=60 and per<75 first class

per>=50 && per<60 second class

Logical operators (when we have to merge/join more than one condition)

These operators are used with relational operators

&& ||

eg :

per=89;

per>=60 && per<75 first class

per>=50 && per<60 second class

eg. swara, ch   //vowels

ch=='a' || ch=='e'||ch=='i'||ch=='o'||ch=='u’

And operator :

a & b

T & T is T other wise all are false i.e

T & F is F , F & T is F, F & F is F

Or operator :

a | b

F | F is F other wise all are True i.e

T | T is T , F | T is T, T | F is T

Bitwise operators & | ^ << >>

Int a=10, b=8; in binary 10 means 00001010 and 8 means 00001000

00001010 (10) // eg. A<<2 ~~10~~001010 => 00101000

&

00001000 (8) // eg. B >> 2 000011 ~~1~~ ~~0~~ => 00000011

00001000 (8)

1010 | 1000 = 1010(10)

^1010 =0101 (4)

^1000= 0111(7)

a << 1 = 00010100 // from left required no of bits(1) will be skipped and that no of bits(1) will be inserted on right side as 0

b>>2 = 00000010 // from right required no of bits(2) will be skipped and that no of bits(2) will be inserted on right side as 0

printf(" And operator a & b = %d\n", a & b); //8

printf("Or operator a | b =%d\n", a | b);

printf(" Negation operator ^a = %d\n", ^a);

printf(" Negation operator ^b = %d\n", ^b);

printf(" Left shift operator a<<1 = %d\n", a<<1);

printf(“ Right Shift operator b>>2 = %d\n”, b>>2);

```
#include<stdio.h>
int main()
{
   int x = 19;
   printf ("x << 1 = %d\n", x << 1);
   printf ("x >> 1 = %d\n", x >> 1);
   return 0;
}
```

Output :

```
38

9
```

Associative operator   a=a+b; i.e. a=+b

Increment and Decrement operator (4 marks) ++ --

Increment Operators :

Whenever , we have to increment the value of variable by 1 only that time we can use this increment operator.

Eg.   i = 6 , i=i+1 or i++

Increment : Post incrementation(i++) , Pre incrementation(++i)

i=10;

printf(“ value of i is =%d”, i++); //10 first assign then increment

printf(“\nI is=%d”, i); //11

printf(“ value of i is =%d”, ++i); // 12 first increment then assign

Decrement Operators :

Whenever , we have to decrement the value of variable by 1 only that time we can use this decrement operator.

Eg.   i = 6 , i=i-1 or i- -

Decrement : Post decrementation(i--) , Pre decrementation(--i)

i=20;

printf(“ value of i is =%d”, i--); // 20

printf(“\nI is=%d”, i); //19

printf(“ value of i is =%d”, --i); //18

We have use of these operators in loops mostly.

*How to use the **Precedence and Associativity** of the operators smartly is one of the important part of C programming.*

Precedence talks about the priority among the different operators, which to consider first. Like arithmetic operators have higher priority than assignment operators and so on. When we have more than one operators in a single statement then this precedence comes into the picture as the result may vary greatly.
For eg – 2 + 3 * 4 – 1 == 2+ (3 * 4) – 1 = 16 => This happens because precedence of multiplication operator is higher than the other two.

Associativity comes into picture when we have operators of the same precedence. It doesn’t talk about which to pick first rather says the order of evaluation.

See the table,

OPERATOR	TYPE	ASSOCIAVITY
( ) [ ] . ->		left-to-right
++ -- + - ! ~ (*type*) * & sizeof	Unary Operator	right-to-left
* / %	Arithmetic Operator	left-to-right
+ -	Arithmetic Operator	left-to-right
<< >>	Shift Operator	left-to-right
< <= > >=	Relational Operator	left-to-right
== !=	Relational Operator	left-to-right
&	Bitwise AND Operator	left-to-right
^	Bitwise EX-OR Operator	left-to-right
|	Bitwise OR Operator	left-to-right
&&	Logical AND Operator	left-to-right
||	Logical OR Operator	left-to-right
? :	Ternary Conditional Operator	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Operator	right-to-left
,	Comma	left-to-right

Precedence and Associativity Table

What is the output of below program?

```
#include<stdio.h>
int main()
{
int a = 5, b = 10;
int c;
c = a * 2 + b/2;
printf("\n output = %d", c);
return 0;
}
```

In the above example, two operators (* and /) has the same precedence, in that situation we need to check the precedence and associativity table. According to table * and / has the associativity from the left to right multiplication will be evaluated first.

```
c = a* 2 + b / 2 ;
c = 5 * 2 + 10 / 2 ;
c = 10 + 10 / 2 ;
c = 10 + 5;
c = 15;
```