2.1 Operators

I. Arithmetic Operators

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

Operator	Meaning
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division(modulo division)

Example:

```
void main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);

    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
    printf("Remainder when a divided by b = %d \n",c);

}
```

Output:

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

II. Unary Operators

There are several Unary Operators in C. Unary operators take just one operand. We have following Unary Operators in C.

Operator	Meaning
+	unary plus
-	subtraction or unary minus
++	Increment
--	Decrement
&	Address of
!	Logical negation
~	one's complement or bitwise negation

Example:

```
void main()
{
   int a = 5, b = 5;
 a++;
 --b;
 printf("%d",a);
 printf("\n%d",b);
 getch();
}
```

Output:

6
4

Difference between pre increment and post increment:

Pre Increment	Post Increment
First the increment will be done and then the value will be assign to variable.	First value will assign and then the increment will be done.
Here ++a is pre increment	Here a++ is post increment

Eg:
void main()
{
int a =10;
clrscr();
	Eg:
	void main()
	{
	int a =10;
	clrscr();

printf("%d",++a);//pre increment
getch();
}
Output:
11
	printf("$d",a++);//post increment
	printf("\n%d",a);
	getch();
	}
	Output:
	10
	11

III. Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0. Relational operators are used in decision making and loops.

Operator	Meaning	Example
==	Equal to	5 == 3 returns 0
>	Greater than	5 > 3 returns 1
<	Less than	5 < 3 returns 0
!=	Not equal to	5 != 3 returns 1
>=	Greater than or equal to	5 >= 3 returns 1
<=	Less than or equal to	5 <= 3 return 0

Example:

```
void main()
{
   int a = 5, b = 5, c = 10;

   printf("%d == %d = %d \n", a, b, a == b); // true
   printf("%d == %d = %d \n", a, c, a == c); // false

   printf("%d > %d = %d \n", a, b, a > b); //false
   printf("%d > %d = %d \n", a, c, a > c); //false


   printf("%d < %d = %d \n", a, b, a < b); //false
   printf("%d < %d = %d \n", a, c, a < c); //true


   printf("%d != %d = %d \n", a, b, a != b); //false
   printf("%d != %d = %d \n", a, c, a != c); //true
```

```
    printf("%d >= %d = %d \n", a, b, a >= b); //true
    printf("%d >= %d = %d \n", a, c, a >= c); //false


    printf("%d <= %d = %d \n", a, b, a <= b); //true
    printf("%d <= %d = %d \n", a, c, a <= c); //true


}
```

Output:

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1
```

IV. Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are commonly used in decision making in C programming.

Operator	Meaning	Example
&&	Logial AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c == 5) && (d > 5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c == 5) || (d > 5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression ! (c == 5) equals to 0.

Example:

```
void main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) equals to %d \n", result);
```

```
    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) equals to %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) equals to %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) equals to %d \n", result);

    result = !(a != b);
    printf("!(a == b) equals to %d \n", result);

    result = !(a == b);
    printf("!(a == b) equals to %d \n", result);
}
```

Output:

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

V. Assignment Operator

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Meaning	Example
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

VI. Conditional Operator

A conditional operator is a ternary operator, that is, it works on 3 operands.

Conditional Operator Syntax

```
conditionalExpression ? expression1 : expression2
```

The conditional operator works as follows:

- The first expression conditionalExpression is evaluated first. This expression evaluates to 1 if it's true and evaluates to 0 if it's false.
- If conditionalExpression is true, expression1 is evaluated.
- If conditionalExpression is false, expression2 is evaluated.

Example:

```
#include <stdio.h>

void main()
{
   int a , b;

   a = 10;
   printf( "Value of b is %d\n", (a == 1) ? 20: 30 );

   printf( "Value of b is %d\n", (a == 10) ? 20: 30 );
}
```

Output:

```
Value of b is 30
Value of b is 20
```

Operator Precedence: OR Operator Priority

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first

Category	Operator	Associativity	Priority
Postfix	() [] -> . ++ - -	Left to right	Highest 14
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left	13
Multiplicative	* / %	Left to right	12
Additive	+ -	Left to right	11
Shift	<< >>	Left to right	10
Relational	< <= > >=	Left to right	9
Equality	== !=	Left to right	8
Bitwise AND	&	Left to right	7
Bitwise XOR	^	Left to right	6
Bitwise OR	|	Left to right	5
Logical AND	&&	Left to right	4
Logical OR	||	Left to right	3
Conditional	?:	Right to left	2
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left	1

Comma	,	Left to right	0

2.2 Expressions

Expressions are combination of variables, constant and operators.

Expressions are evaluated using an assignment statement of the form:

```
variable = expression
```

In the above syntax, **variable** is any valid C variable name. When the statement like the above form is encountered, the expression is evaluated first and then the value is assigned to the variable on the left hand side. All variables used in the expression must be declared and assigned values before evaluation is attempted. Examples of expressions are:

```
x = a*b-c
y=b/c*a
z=a-b/c+d
```

Expressions are evaluated based on operator precedence and associativity rules when an expression contains more than one operator.

Eg.

A+B, c=d-e, a++, --d, a<b, b>a && c<d

I. Arithmetic Expressions

Arithmetic Expressions are combination of Constant, variable and Arithmetic operators.

Example.

A+B, C-D, X/Y etc…

```
void main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);

    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
```

```
        printf("Remainder when a divided by b = %d \n",c);

    }
```

Output:

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

II. Boolean Expressions

Boolean Expressions are combination of Constant, variable and Logical Operators

Eg.

A<B && A<C, D>C || D>B etc

Example:

```
void main()
{
   int a = 5, b = 5, c = 10, result;

   result = (a == b) && (c > b);
   printf("(a == b) && (c > b) equals to %d \n", result);

   result = (a == b) && (c < b);
   printf("(a == b) && (c < b) equals to %d \n", result);

   result = (a == b) || (c < b);
   printf("(a == b) || (c < b) equals to %d \n", result);

   result = (a != b) || (c < b);
   printf("(a != b) || (c < b) equals to %d \n", result);

   result = !(a != b);
   printf("!(a == b) equals to %d \n", result);

   result = !(a == b);
   printf("!(a == b) equals to %d \n", result);
}
```

Output:

```
(a == b) && (c > b) equals to 1
(a == b) && (c < b) equals to 0
(a == b) || (c < b) equals to 1
(a != b) || (c < b) equals to 0
!(a != b) equals to 1
!(a == b) equals to 0
```

2.3 Evaluation and Assignment of Expressions

Category	Operator	Associativity	Priority
Postfix	() [] -> . ++ - -	Left to right	Highest 14
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left	13
Multiplicative	* / %	Left to right	12
Additive	+ -	Left to right	11
Shift	<< >>	Left to right	10
Relational	< <= > >=	Left to right	9
Equality	== !=	Left to right	8
Bitwise AND	&	Left to right	7
Bitwise XOR	^	Left to right	6
Bitwise OR	|	Left to right	5
Logical AND	&&	Left to right	4
Logical OR	||	Left to right	3
Conditional	?:	Right to left	2
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left	1

Comma	,	Left to right	0

Example:

1. a=10, b=12
   C=a+b
   =22
2. a=10, b=20,c=40
   d=a+b*c
   10+20*40
   10+80
   90
3. a=10,b=20,c=30,d=40
   z=(a+b)-(c+d)
   (30)-(70)
   -40