Expressions

➲ An expression is a sequence of operands and operators that reduces to a single value. Expressions can be simpler or complex.

- ✓ An operator is a syntactical token that requires an action to be taken.
- ✓ An operand is an object on which an operation is performed.

➲ We can divide simple expressions into six categories based on the number of operands, relative positions of the operand and operator

1. Primary Expressions
2. Unary Expressions
3. Binary Expressions
4. Ternary Expressions
5. Assignment Expressions
6. Comma Expressions

1. Primary Expressions

➲ An expression with only one operand but without any operator is called primary expression. The various types of primary expressions are:

(i)	Names	Ex: int a
(ii)	Constants	Ex: 5, 10.5
(iii)	Parenthesized expressions	Ex: (2+3*(4-2))

2. Unary Expressions

➲ An expression with only one operand and one operator is called unary expression. The unary expression act on a single operand to produce a value. The various types of unary expressions are:

(i)	Unary minus expression	Ex: -5
(ii)	Unary plus expression	Ex: +5
(iii)	Prefix expression	Ex: ++i
(iv)	Postfix expression	Ex: i++

3. Binary Expressions

➲ An expression containing two operands and an operator is called binary expression. A binary operator acts on two operands to produce a value. The various types of binary expressions are:

(i)	Multiplicative expressions	Ex: 2*3 , 6/2
(ii)	Additive expressions	Ex: a-b, a + b etc

(iii)	Relational expressions	Ex: a>b, a<b etc
(iv)	Logical expressions	Ex: a && b, a || b
(v)	Bitwise expressions	Ex: a &b , a | b

4. **Ternary Expressions**

➲ An expression containing three operands and two operators is called ternary expression. Here, the two operators act on three operands.

Example:a ?b:c // a,b and c are operands and **?** and**:**are operators

5. **Assignment Expressions**

➲ A statement with assignment operator is called assignment statement. The assignment statements are often referred to as assignment expressions

Example1: a=10 **Example 2:**a = b + c

6. **Comma Expressions**

➲ A set of statements separated by commas are evaluated from left to right one after the other. Such statements are called statements with comma operator.

Example: a=10, b=20, c=30;

Type Conversion

➲ The process of converting the data from one data type to another data type is called Type conversion.

➲ Type conversion is of two types

(i) Implicit type conversion

(ii) Explicit type conversion

1. **Implicit Conversion**

➲ C permits mixing of constants and variables of different types in an expression. C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance. The automatic conversion is known as implicit conversion.

➲ If the Operands are of different types, the lower data type is automatically converted to higher type before the operation proceeds.

For example,

- If one operand type is same as other operand type, no conversion takes place and type of result remains same as the operands i.eint + int = int, float + float = float etc.
- If one operand type is int and other operand type is float, the operand with type int is promoted to float.

Examples:

int + int = int	int + float = float
5 + 3 = 8	5 + 3.5 = 8.5

2. Explicit type conversion

- Implicit type conversion is possible only if the data types of two operands are different. In some situation where the data types of two operands are same, still conversion is required, and then we have to go for explicit type conversion.

  Example,

  We want to find the ratio of girls to boys in the college. The ratio is given by

  Ratio = no_of_girls / no_of_boys.

- Here, number of girls and number of boys will be of type integer. So, the compiler will not do any implicit type conversion because, both operands are of the same data type and hence the result will be of type integer. But, to get the ratio which is a floating point number, we are forced to convert one of the operands to float so that the result is also float. This is the place where we require explicit type conversion.
- ***Definition:*** If the Operands are of the same data type, no conversion takes place by the compiler. Sometimes, the type conversion is required to get the desired results. In such case, the programmer can instruct the compiler to change the type of the operand from one data type to another data type. *This forcible conversion from one data type to another data type is called explicit type conversion.*

  The syntax is: (type) expression

  Example:

  (int) 9.43: The data conversion from higher data type to lower data type is possible using explicit type conversion. The following example gives the explicit type conversion.

1	x=(int) 7.5	7.5 is converted to integer by truncation
2	a=(int)21.3 / (int) 4.5	Evaluated as 21/4 and the result would be 5.
3	b=(double) sum/n	Division is done in floating point mode
4	y=(int)(a+b)	The result a + b is converted to integer
5	z=(int)a +b	a is converted to integer and then added to b

Now let us see "What is type casting? Explain with example?"

Definition: C allows programmers to perform typecasting by placing the type name in parentheses and placing this in front of the value.

For instance

```
main()
{
float a;
a = (float)5 / 3;
}
```

Gives result as 1.666666 . This is because the integer 5 is converted to floating point value before division and the operation between float and integer results in float.

Important points to be remembered:

- **Precedence[Priority]**

  It is the rule that specifies the order in which certain operations need to be performed in an expression. For a given expression containing more than two operators, it determines which operations should be calculated first.

- **Associativity**

  If all the operators in an expression have equal priority then the direction or order chosen left to right or right to left to evaluate an expression is called associativity.

__BODMAS Rule__ can be used to solve the expressions,

Where- __B__: Brackets __O__: Operators __D__: Division __M__: Multiplication __A__: Addition __S__: Subtraction

- **The Precedence (Hierarchy) of Operator**

Operator Category	**Operators in Order of Precedence (Highest to Lowest)**	**Associativity**
Innermost brackets/Array elements reference	(), [], {}	Left to Right(L->R)
Unary Operators	++, --, sizeof(), ~, +, -	Right to Left(R->L)
Member Access	-> or *	L->R
Arithmetic Operators	*, /, %	L->R
Arithmetic Operators	-, +	L->R
Shift Operators	<<, >>	L->R
Relational Operators	<, <=, >, >=	L->R

Equality Operators	==, !=	L->R
Bitwise AND	&	L->R
Bitwise XOR	^	L->R
Bitwise OR	|	L->R
Logical AND	&&	L->R
Logical OR	||	L->R
Conditional Operator	?:	R->L
Assignment Operator	=, +=, -=,*=, /=, %=	R->L
Comma Operator	,	L->R

➲ **Evaluation of Expressions involving all the operators**

The rules to be followed while evaluating any expressions are shown below.

- ❖ Replace the variables if any by their values.
- ❖ Evaluate the expressions inside the parentheses
- ❖ Rewrite the expressions with increment or decrement operator as shown below:
  - a) Place the pre-increment or pre-decrement expressions before the expression being evaluated and replace each of them with corresponding variable.
  - b) Place the post-increment or post- decrement expressions after the expression being evaluated and replace each of them with corresponding values.
- ❖ Evaluate each of the expression based on precedence and associativity.

➲ **Program to demonstrate the use of Bitwise operators:**

```
#include <stdio.h>
        main()
        {
                unsignedint a = 60;             /* 60 = 0011 1100 */
                unsignedint b = 13;             /* 13 = 0000 1101 */
                int c = 0;
                c = a & b;                       /* 12 = 0000 1100 */
                printf("Line 1 - Value of c is %d\n", c );
                c = a | b;                      /* 61 = 0011 1101 */
                printf("Line 2 - Value of c is %d\n", c );
                c = a ^ b;                      /* 49 = 0011 0001 */
                printf("Line 3 - Value of c is %d\n", c );
                c = ~a;                         /*-61 = 1100 0011 */
                printf("Line 4 - Value of c is %d\n", c );
                c = a << 2;                     /* 240 = 1111 0000 */
                printf("Line 5 - Value of c is %d\n", c );
                c = a >> 2;                    /* 15 = 0000 1111 */
                printf("Line 6 - Value of c is %d\n", c );
        }
```

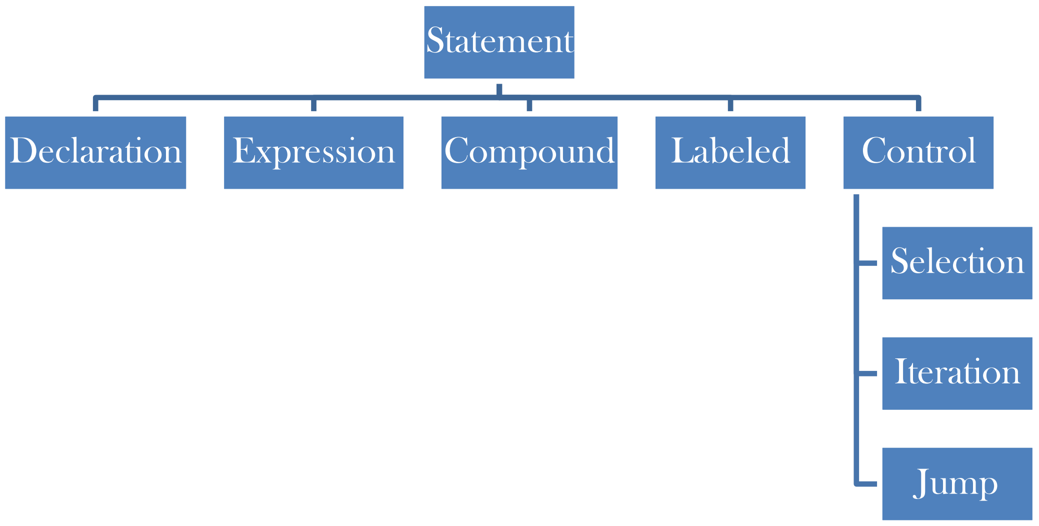
When you compile and execute the program, it produces the following result −

Output:

```
        Line 1 - Value of c is 12
        Line 2 - Value of c is 61
        Line 3 - Value of c is 49
        Line 4 - Value of c is -61
        Line 5 - Value of c is 240
```

Program Statement

- Statement is a syntactic construction that performs an action when a program is executed. All C program statements are terminated with a semi-colon (;)



- **Declaration** is a program statement that communicates with the compiler about the name and types of the data needed for program execution.

  Example:inta, b;

- **Expression** statement contains no more that expression i.e.., only the sequence of operands with operators.

  Example: C = A+B*D;

- **Compound** statements are the sequence of statements that may be treated as a single statement in the constructions larger statements. Always compound statements will be enclosed with in { and }.
- **Labeled** statements can be used to mark any statement so that control may be transferred to that statement.
- **Control** statement is a statement whose execution results is a choice being made on condition or path of execution. It can also be defined as the statement determines the flow of control of the program.
  - **Selection statements** allow the programmer to select a particular execution path from a set of one or more alternatives.
  - **Iteration statements** are used to execute a group of one or more statements repeatedly.
  - **Jump statements** cause an unconditional jump to some other place in the program.