Assignment 3.1

1) Differentiate between unary, and binary operators with examples.

→

Unary operators	Binary Operators
- Unary operators perform an action with a single operand.	- Binary operators perform actions with two operands.
- Unary operators have one operand/ value that they work with. Ex: x++ (x is operand).	- Binary Operators need two operands. Ex: y * 7 (operands are y and 7).
- They are pre-increment and post increment (++)	- They are mathematical operators and relational operators
- A unary operator typically appears with its operand in following format: Operator Operand ++ A	- A binary operator appears with its operands in following format: Operand1 Operator Operand2 A + B
- Example of unary operator, prefix (increment ++a), (decrement --a) postfix (increment a++, decrement a--)	- Example of binary Operator, Arithmetic (+, -, *, /, %) Relational (<, <=, >, >=, ==, !=) Logical (&&, ||, !0) Assigment (=, +=, -=, *=, /=)

2) Categorize operators based on their functionality and provide examples for each category.

→ Operators in C++ can be categorized based on their functionality.

1) Arithmetic Operators :
   - Addition (+)
   - Subtraction (-)
   - Multiplication (*)
   - Division (/)
   - Modulus (%)

2) Relational Operators :
   - Equal to (==)
   - Not equal to (!=)
   - Greater than (>)
   - Less than (<)
   - Greater than or equal to (>=)
   - Less than or equal to (<=)

3) Logical operators :
   - Logical AND (&&)
   - Logical OR (||)
   - Logical Not (!)

4) Assignment Operators :
   - Assignment (=)
   - Addition assignment (+=)
   - Subtraction assignment (-=)
   - Multiplication assignment (*=)

- Division assignment (/=)
- Modulus assignment (%=)

5) Increment and Decrement Operators:
- Increment (++)
- Decrement (--)

6) Bitwise Operators:
- Bitwise AND (&)
- Bitwise OR (|)
- Bitwise XOR (^)
- Bitwise NOT (~)
- Left shift (<<)
- Right shift (>>)

7) Ternary Operator:
- Conditional Operator

8) Member Access Operators.
- Dot operators (.)
- Arrow operator (->)

9) Other Operators:
- Comma (,)
- sizeof
- Cast Operator (type)

3) What is type casting and why it is necessary in c++?

→ Type conversion in c++ language, also known as type casting, refers to the process of converting a value from one data type to another.

M T W T F S S

- Types of Type conversion

1) Implicit type conversion : Implicit type conversion is performed by the compiler automatically during compilation.

```
int num1 = 10;
float num2 = 5.5;
float ans = num1 + num2;
//num1 is implicitly converted to float before addition
```

2) Explicit type conversion : Explicit type conversion, or type casting, is done by the programmer explicitly using casting operators. Type casting is performed using casting operators like (type).

```
int num1 = 10;
float num2 = 5.5;
int ans = num1 + (int)num2;
//num2 is explicitly case to int before addition
```

4) Differente between implicit and explicit type casting.

→ implicit type casting is performed automatically by the compiler, while explicit type casting requires explicit instruction from the programmer using

casting operators. Implicit casting is safer and generally results in fewer errors, while explicit casting provides more control over the conversion process but requires careful handling to avoid potential issues such as data loss or unexpected behavior.

5) Explain Operator precendence by solving any expression with appropriate steps.

→ Operator precedence refers to the rules that determine the order in which operators are evaluated in an expression. Operators with higher precedence are evaluated before operators with lower precedence. To illustrate Operator precedence, let's solve the expression '5 * 2 + 10 / 2 - 3'.

1. Multiplication: Evaluate the multiplication operation '5 * 2'. Result: '10'.
2. Division: Evaluate the division Operation '10 / 2'. Result: '5'
3. Addition: Evaluate the addition '5 + 5' Result: '10'
4. Subtraction: Evaluate subtraction operation '10 - 3' Result: '7'

So, the final result of the expression '5 * 2 + 10 / 2 - 3' is 7.