1. Programming Language:

➢ What is Programming Language?

⇒ A programming language is a computer language used by programmers (developers) to communicate with computers. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a particular task.

2. Introduction to C Language:

➢ What is C Language?

⇒ The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

⇒ C programming is considered the basis for other programming languages, which is why it is known as the mother language because most of the compilers, JVMs, Kernels, etc. are written in C language, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

⇒ C language is a system programming language too because it can be used to do low-level programming. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, the Linux kernel is written in C.

➢ Features of C Language:

⇒ C is a widely used language. It provides many features that are given below.

- Simple
- Machine Independent or Portable
- Mid-level programming language
- structured programming language

- Rich Library
- Memory Management
- Fast Speed
- Pointers

➢ First C Program:

⇒ To write the first c program, open the C console and write the following code:

```c
#include <stdio.h> // Header Line

int main() // Start of main
{
    // Scope of the main function
    printf("Hello C Language");
    return 0;

} // End of main
```

⇒ # (pound sign) - preprocessor directives
⇒ **include** → Add to the filename in the current file **<filename>** **/** "**filename**"

⇒ **stdio.h** → Standard input/output header file.
→ **stdio** - File name (Standard Input Output)
→ **.h** - Extensions (Header File)

⇒ **int** → datatype

⇒ **main()** → By in-built function
→ Program executes here.

⇒ **{ }** → Scope / Branch / Block / Body

⇒ **printf()** → Print Output to the user screen

⇒ **return 0;** → The main function must be return

2. Basic Syntax:

➢ **Escape Sequence:**

➔ **“\n”** → New line

➔ **“\t“** → Tab line

➢ **Comments:**

⇒ Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

- Single Line Comments
- Multi-Line Comments

1. **Single Line Comments:**

   ⇒ Single-line comments are represented by a double slash ‘ **//** ‘.

2. **Multi-Line Comments:**

   ⇒ Multi-Line comments are represented by a slash asterisk ‘ ‘. It can occupy many lines of code.

3. Data Types:

⇒ A data type specifies the type of data that a variable can store such as integer, floating, character, etc.

➢ **Basic Data Type:**

Value	Name	Datatype	Format Specifier
0, 1, 2, 3,.....,9	Integer	**int**	%d
12.25, 3.66, 9.55,...	Decimal	**float**	%f

a, b, c, A, B, C,....,z, Z	Character	**char**	%c
Skill	Word	**string**	%s
	Empty	**void**	

4. Variables:

⇒ A variable is the name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

➢ **Rules for defining variables:**

- ➔ A variable can have alphabets, digits, and underscore.
- ➔ A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- ➔ No whitespace is allowed within the variable name.
- ➔ A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Syntex:

- ➔ datatype variableName = value; **(declared and initialization variables)**
- ➔ datatype variableName; **(declared variables)**

Example:

- ➔ int a = 20;
- ➔ int b;

➢ **Types of Variables in C:**

1. local variable
2. global variable
3. static variable

5. Operators:

- ➢ **Operator:**
  ⇒ Operators are the special kinds of symbols that are used to perform any specific task mathematically and logically.

Example:

- ➔ a + b // + - is operators

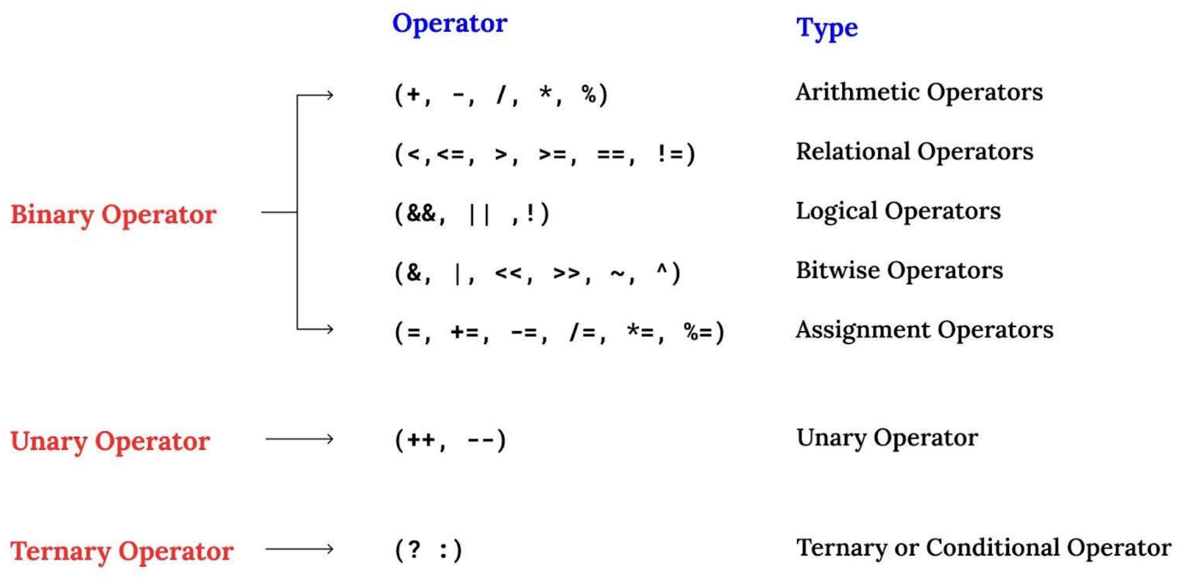
- ➢ **Operands:**
  ⇒ In any operation, an operand may have any numerical value, variables, constants, function calling etc on which the program makes an operation.

Example:

- ➔ a + b // a and b are operands

- ➢ **Type of Operators:**

  - ➔ Arithmetic Operators
  - ➔ Relational Operators
  - ➔ Logical Operators
  - ➔ Assignment Operator
  - ➔ Bitwise Operators
  - ➔ Unary operator
  - ➔ Ternary or Conditional Operator



➤ Arithmetic Operators:

⇒ The arithmetic operators help in performing mathematical functions. Arithmetic Operators in C allow a user to construct various formulas and mathematical equations.

Let us assume that variable A has a value of 5, and variable B has a value of 10. Then:

Operator	Description	Example
+	Addition of two operands	var = A + B = 15
-	Subtraction of the second variable from the first one	var = A – B = -5
/	Division of the numerator by the de-numerator	var = B / A = 2
*****	Multiplication of both the operands	var = A * B = 50
%	The modulus operator and the remainder after the division of an integer	var = B % A = 0

Example:

```
printf("Addition is %d\n", a+b );            // 15
printf("Subtraction is %d\n", a-b );         // -5
printf(" Division is %d\n", b/a );          // 2
printf("multiplication is %d\n", a*b );     // 50
printf("Reminder is %d\n", b%a );          // 0
```

➢ **Relational Operators:**

⇒ These types of operators check the relationship between two of the available operands. In case the relation happens to be true, then it returns 1. But in case this relation turns out to be false, then it returns the value 0. We use the relational operators in loops and in cases of decision-making.

⇒ relational operators are used in decision-making and loops.

Operator	Meaning of Operator
==	Equal to
>	Greater than
<	Less than
!=	Not equal to
>=	Greater than or equal to
<=	Less than or equal to

Example:

```
int p = 5, q = 5, r = 10;

printf("%d == %d is %d \n", p, r, p == r);   // 5 == 10 is
0
printf("%d == %d is %d \n", p, q, p == q);   // 5 == 5 is 1
printf("%d > %d is %d \n", p, r, p > r);     // 5 > 10 is 0
printf("%d > %d is %d \n", p, q, p > q);     // 5 > 5 is 0
printf(“%d < %d is %d \n”, p, r, p < r);      // 5 < 10 is 1
printf(“%d < %d is %d \n”, p, q, p < q);      // 5 < 5 is 0
printf(“%d != %d is %d \n”, p, r, p != r);   // 5 != 10 is
1
printf(“%d != %d is %d \n”, p, q, p != q);   // 5 != 5 is 0
printf(“%d >= %d is %d \n”, p, r, p >= r);   // 5 >= 10 is
0
printf(“%d >= %d is %d \n”, p, q, p >= q);   // 5 >= 5 is 1
printf(“%d <= %d is %d \n”, p, r, p <= r);   // 5 <= 10 is
1
printf(“%d <= %d is %d \n”, p, q, p <= q);   // 5 <= 5 is 1
```

➢ Logical Operators:

⇒ We use logical operators to perform various logical operations on any set of given expressions. The logical operators in C are used for combining multiple constraints/ conditions or for complementing the evaluation of any original condition that is under consideration.

➢ **We have three major logical operators in the C language:**

- Logical NOT (!)
- Logical OR (||)
- Logical AND (&&)

➢ **Here is the truth table for the AND(&&) operator when you are working with two operands:**

First Operands	Second Operands	Result
True	True	True
True	False	False
False	True	False
False	False	False

➢ **Here is the truth table for the OR(||) operator when you are working with two operands:**

First Operands	Second Operands	Result
True	True	True
True	False	True
False	True	True
False	False	False

➢ **Here is the truth table for the NOT(!) operator:**

Second Operands	Result
True	False
False	True

Example:

```
int a = 5, b = 5, c = 10, result;

result = (a == b) && (c > b);
printf("(a == b) && (c > b) is %d \n", result);  // (a == b) &&
(c > b) is 1

result = (a == b) && (c < b);
printf("(a == b) && (c < b) is %d \n", result);  // (a == b) &&
(c < b) is 0

result = (a == b) || (c < b);
printf("(a == b) || (c < b) is %d \n", result);  // (a == b) ||
(c < b) is 1

result = (a != b) || (c < b);
printf("(a != b) || (c < b) is %d \n", result);  // (a != b) ||
(c < b) is 0

result = !(a != b);
printf("!(a != b) is %d \n", result);      // (a != b) is 1

result = !(a == b);
printf("!(a == b) is %d \n", result);      // (a == b) is 0
```

➢ Assignment Operator:

It is an operator which assigns value from right to left operand. We can perform various arithmetic operations before assigning values to the operand.

For example, a = b;
Here, the value of b is set to variable a.

```
a += b; //a = a + b
```

Here, the addition will be performed on a and b operands and the result will be stored in the left side operand, which is a.

```
a -= b; // a = a - b
```

➢ **Unary operator:**
➢ **Ternary or Conditional Operator:**

➢ **Unary Operators:**

⇒ These are the type of operators that act upon just a single operand for producing a new value. All the unary operators have equal precedence, and their associativity is from right to left. Combining the unary operator with an operand gives us the unary expression.

- Increment (++) Operator
- Decrement (--) Operator

➢ **Increment Operator:**

⇒ We use this operator to increment the overall value of any given variable by a value of 1. We can perform increments using two major ways:

- Prefix increment ++a
- Postfix Increment a++

⇒ **Prefix Increment**

⇒ The operator in this method precedes the given operand (Example, ++p). Thus, the value of the operand gets altered *before* we finally use it.

For instance,

```
int p = 1;

int q = ++p; // q = 2
```

⇒ **Postfix Increment**

⇒ The operator in this method follows the given operand (Example, p++). Thus, the value of the available operand gets altered after we use it.

```
int p = 1;

int q = p++; // q = 1

int r = p; // r = 2
```

➢ **Decrement Operator:**

⇒ The prefix decrement operator decrements the current value of the available variable immediately, and only then this value is used in an expression. In simpler words, the value of a variable first gets decremented and then used inside any expression in the case of a decrement operation..

➢ **Postfix Decrement Operators:**

⇒ The postfix decrement operator allows the usage of the current value of a variable in an expression and then decrements its value by 1. In simpler words, the value of a variable is first used inside the provided expression, and then its value gets decremented by 1.

⇒ **Syntax:**

```
B = --A;
```

➢ **Prefix Decrement Operators:**

⇒ The prefix decrement operator decrements the current value of the available variable immediately, and only then this value is used in an expression. In simpler words, the value of a variable first gets decremented and then used inside any expression in the case of a decrement operation

⇒ **Syntax:**

```
B = A--;
```

➢ **Ternary or Conditional Operator:**

⇒ The programmers utilise the ternary operator in case of decision making when longer conditional statements like if and else exist. In simpler words, when we use an operator on three variables or operands, it is known as a Ternary Operator.

⇒ **Syntax:**

Condition? value_if_true: value_if_false;

This very statement evaluates to a value_if_true only when the condition is met. Otherwise, it evaluates to a value_if_false.

6. Control Flow:

- ➢ **Conditional Statements: (if-else, switch)**
- ➢ **Looping Statements: (for, while, do-while)**
- ➢ **Jump Statements: (break, continue, goto)**

➢ **if Statement in C Language:**

⇒ The execution of the statements available inside and *if* block will only happen when the condition defined by this statement turns out to be true. In case the available condition is false, the compiler will skip the available statement that gets enclosed in the body of the if statement.

⇒ **Syntax:**

```
if(condition)

{

 // A block of statements for the C program
```

```
       // The execution of these statements will only happen
when this       condition turns to be true

       }
```

➤ **else Statement in C Language:**

Use the `else` statement to specify a block of code to be executed if the condition is `false`.

⇒ **Syntax:**

```
      if (condition) {

      // block of code to be executed if the condition is true

      } else {

      // block of code to be executed if the condition is false

      }
```

➤ **if-else Statement in C Language:**

⇒ When we use the if… else statement, there occurs an execution of two different types of statements in a program. First, if the available condition in the program is true, then there will be an execution of the first statement. The execution of the second condition will only occur if the condition available to us is false.

⇒ **Syntax:**

```
     if(expression){

     //code to be executed if the condition is true

     }else{

     //code to be executed if the condition is false
```

```
      }
```

➤ ***nested if*** **Statement in C Language:**

⇒ The nested if statement is fairly useful in case a user needs to evaluate multiple types of conditions. Here, the *if* block is present to define a condition (or an expression). The resultant statement will get executed on the basis of the result of this expression

⇒ **Syntax:**

```
if (condition x)

{ Statement x; }

else if(condition y)

{ Statement y; }

else

Statement z;
```

➤ **else-if ladder Statement in C Language:**

⇒ In this statement, the execution of an array of instructions occurs only when the available condition is correct. The verification of the next condition occurs when this first condition is incorrect. In case all of the specifications fail even after the verification, then there will be an execution of the default block statements. The remainder of the program's ladder is shown below.

⇒ **Syntax:**

```
if(condition1){
 //code to be executed if condition1 is true
 }else if(condition2){
 //code to be executed if the condition is true
 }
 else if(condition3){
 //code to be executed if the condition is true
 }
 ...
 else{
 //code to be executed if all the conditions are false
 }
```

➢ **Loop Control Statement:**

⇒ We use the loop control statements in C language for performing various loop operations until we find the condition given in a program to be true. The control comes out of a loop statement when the condition given to us turns out to be false. Looping is basically a phase in which we repeat the very same process multiple times unless it specifies any specific type of condition.

➢ **Types of Loops in C:**

- while loop
- nested loop
- for loop
- do-while loop

➢ **For Loop :**

⇒ The for loop in C language is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

It also executes the code until the condition is false. In this three parameters are given is

- Initialization
- Condition
- Increment/Decrement

⇒ **Syntax :**

```
for(initialization; condition; increment/decrement)
 {
      //code to be executed

 }
```

⇒ It is used when the number of iterations is known while it is used when the number of iterations is not known.

➢ **while loop :**

⇒ While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

⇒ **Syntax :**

```
while (condition test)

{

     //Statements to be executed repeatedly

    //Increment (++) or Decrement (--) Operation

}
```

➢ do while loop in C :

⇒ The do-while loop is a post-tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

⇒ **Syntax :**

```
     do{
     //code to be executed
     }while(condition);
```