C LANGUAGE

BASIC THEORY

**DATA TYPES:**

A data type is a classification of data that determines the size and layout of the variable in memory, as well as the set of operations that can be performed on it.

The data types in C can be classified as follows:

**Types:**

**Primitive Data Types:**

Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.

**User Defined Data Types:**

The user-defined data types are defined by the user himself.

**Derived Types** :

The data types that are derived from the primitive or built-in data types are referred to as Derived Data Types.

**VARIABLE:**

A variable is a named storage location that holds a value of a specific type. Variables are used to store and manipulate data in a program

**CONDITIONAL STATEMENT :**

**[If Statement]:**

The If statement in C is used to execute a block of code based on a specified condition.

The syntax of If :

If (condition) {

// code to be executed if the condition is true }

**[If-else Statement]:**

The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the specified condition (test expression).

If the given condition is true, then the code inside the If block is executed, otherwise the code inside the else block is executed.

**Syntax of if-else:**

if (condition) {

// code executed when the condition is true

}

else {

// code executed when the condition is false

}

**[NESTED IF -ELSE]:**

A nested if in C is an if statement that is the target of another if statement.

Nested if statements mean an if statement inside another if statement.

Yes, C allow us to nested if statements within if statements, i.e, we can place an if statement inside another if statement.

**Syntax of Nested if-else:**

if (condition1)

{

// Executes when condition1 is true

if (condition\_2)

{

// statement 1

}

else

{

// Statement 2

}

}

else {

if (condition\_3)

{

// statement 3

}

else

{

// Statement 4

}

}

**[LADDER IF-ELSE]:**

The If else if statements are used when the user has to decide among multiple options.

The C if statements are executed from the top down.

As soon as one of the conditions controlling the If is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed.

If none of the conditions is true, then the final else statement will be executed. If-else-If ladder is similar to the switch statement.

**Syntax of if-else-if Ladder:**

if (condition)

statement;

else if (condition)

statement;

.

.

else

statement;

**LOOP:**

Loops are responsible for performing repetitive tasks using a short code block that executes until the condition holds true. In this article, we will learn about for loop in C.

There are two types of loop :

1] Entry control loop -> for, while

2] Exit control loop -> do-while

**For loop :**

The for loop in C Language provides a functionality/feature to repeat a set of statements a defined number of times. The for loop is in itself a form of an entry-controlled loop.

Unlike the while loop and do…while loop, the for loop contains the initialization, condition, and updating statements as part of its syntax.

It is mainly used to traverse arrays, vectors, and other data structures.

Syntax of for Loop:

for(initialization; check/test expression; updation)

{

// body consisting of multiple statements

}

**While loop :**

The while Loop is an entry-controlled loop in C programming language. This loop can be used to iterate a part of code while the given condition remains true.

Syntax:

while (test expression)

{

// body consisting of multiple statements

}

**DO-WHILE LOOP**:

The do-while loop is similar to the while loop with one important difference.

The body of do-while loop is executed at least once. Only then, the test expression is evaluated.

The syntax of the do-while loop is:

do {

// the body of the loop

}

while (test Expression);

**LOOP CONTROL STATEMENT**:

**BREAK STATEMENT :**

The break in C is a loop control statement that is used to terminate the loop.

As soon as the break statement is encountered from within a loop, the loop iterations stop there and control returns from the loop immediately to the first statement after the loop.

Syntax:

break;

**CONTINUE STATEMENT :**

C continue statement is a loop control statement that forces the program control to execute the next iteration of the loop.

As a result, the code inside the loop following the continue statement will be skipped and the next iteration of the loop will begin.

**Syntax:**

continue;

**ARRAY:**

**SINGLE DIMENSIONAL ARRAY :**

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To create an array, define the data type (like int) and specify the name of the array followed by square brackets [].

To insert values to it, use a comma-separated list, inside curly braces:

**SYNTAX :**

Int my Numbers[4] = {25, 50, 75, 100};

**MULTI DIMENSIONAL ARRAY:**

A multidimensional array is basically an array of arrays.

Arrays can have any number of dimensions. In this chapter, we will introduce the most common; two-dimensional arrays (2D).

**SYNTAX:**

int matrix[2][3] = { {1, 4, 2}, {3, 6, 8} };

**STRING :**

Strings are used for storing text/characters.

For example, "Hello World" is a string of characters.

Unlike many other programming languages, C does not have a String type to easily create string variables. Instead, you must use the char type and create an array of characters to make a string in C:

char greetings[] = "Hello World!";

**File Management:**

File handling in C is the process in which we create, open, read, write, and close operations on a file.

C language provides different functions such as fopen(), fwrite(), fread(), fseek(), fprintf(), etc.

to perform input, output, and many different C file operations in our program.

**Types :**

1 write

2 read

3 appned

**use:**

open

content

close

**FUNCTIONS :**

Function is a block of code that we can use it again and again .

**Syntax:**

return\_type name\_of\_the\_function (parameter\_1, parameter\_2);

**FUNCTION TYPES :**

1) user defined :

2) built in :

**1)user defined :**

user defined devided into 4 types :

a)function without parameters and without return type.

b)function with parameters and without return type.

c)function without parameters and with return type.

d)function with parameters and with return type.

2)**built in :**

**FUNCTION USE** :

1)Declaration

2)Defination

3)Calling

**STRUCTURE:**

We have to declare structure in C before using it in our program.

In structure declaration, we specify its member variables along with their datatype.

We can use the struct keyword to declare the structure in C using the following syntax:

**Syntax:**

struct structure\_name {

data\_type member\_name1;

data\_type member\_name1;

....

....

};

**UNION:**

We need to define a variable of the union type to start using union members.

There are two methods using which we can define a union variable.

With Union Declaration

After Union Declaration

**SYNTAX:**

1. Defining Union Variable with Declaration

union union\_name {

datatype member1;

datatype member2;

...

} var1, var2, ...;

**RECURRSION:**

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve.