3.1 Iterative/Repetition/Loop Control Structures-2

**💻Iterative/Repetition/Loop Control Structures Key Points**

**📚Topics**

This module will cover the following topics:

1. The while loop control structure
2. The do while loop control structure
3. The for loop control structure

**📀Iterative/Repetition/Loops Control Structure**

* Enable programs to perform statements repeatedly as long as a condition remains true.
* Loops can execute a block of code as long as a specified condition is reached.
* Loops are handy because they save time, reduce errors, and they make code more readable.

**📀Types of Iterative/Repetition/Loop Control Structures**

* The while loop
* The do while loop
* The for loop

### ****The While Loop Control Structure****

#### ****📀Types of Iterative/Repetition/Loop Control Structure****

## **(a) C++ While Loop**

* Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
* The while condition loops through  a block of code as long as a specified condition is TRUE.
* The while loop runs a block of code until a condition is no longer true. If the condition is never true, the code is never executed.
* The minimum output of while loop is **zero or more.**

### Syntax

while (condition) {  
*// code block to be executed*  
}

### ****The Do While Loop Control Structure****

#### ****📀 Types of Iterative/Repetition/Loops Control Structure****

## **(b) C++ Do While Loop**

* Repeats a statement or group of statements while a given condition is true. It tests the condition at the end of the loop body.
* The do while loop while execute the code block once, before checking if the condition is TRUE, then it will repeat the as long as the condition is TRUE.
* The do while loop runs a block of code at least once, even if the tested condition is never true.

### Syntax

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

### ****The For Loop Control Structure****

#### ****📀 Types of Iterative/Repetition/Loops Control Structure****

## **(c) C++ for Loop**

* Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
* If you know exactly how many times you want to loop through a block of code, used for loop instead of a while loop.
* A for loop is designed using initialization, test and action sections. These sections make it possible to create a counter variable, test its value, and change its value all within the for statement that creates the loop.

### Syntax

for(initialization; condition; formula)  
 {  
*// code block to be executed*  
}

Module 3 Summary -2

**📔Module 3 Summary or Quick Review**

* **The Iterative/Repetition/Loops Control Structure  enable programs to perform statements repeatedly as long as a condition remains true.**
* **The while loop repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.**
* **The do while loop repeats a statement or group of statements while a given condition is true. It tests the condition at the end of the loop body.**
* **The for loop executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.**

4.1 Function-2

**💻 Functions**

**📚Topics**

This module will cover the following topics:

1. What is Functions?
2. Declaring and Defining Functions
3. Void Functions
4. Returning Values from Functions
5. Function Call by Value
6. Function Call by Reference

**📀C++ Functions**

* A function is a block of code which only runs when it is called.
* You can pass data, known as parameters, into a function.
* Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.
* A function is a group of statements that together perform a task. Every C++ program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
* You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.
* The C standard library provides numerous built-in functions that your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.
* A function can also be referred as a method or a sub-routine or a procedure, etc.

**📀 Declaring and Defining Functions**

* A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.
* The parameter list identifies each parameter and its type, separated by commas.
* Here's a declaration for a function that  determine a rectangle's area using length and width parameters:

                  int findArea (int length, int width);

**The three parts of the declaration are the following:**

* The return type, int
* The name, findArea
* The type and name of two parameters, an**int** named **length** and an **int** named **width**

4.1.1 Void Functions

**💻 Void Functions**

**📀Void Functions**

* **Void** functions are functions that do not return a value.
* A common application where a void function is used in printing the result of calculations to the screen.  The calculations might be performed elsewhere, but the results would be printed using the void function.
* If a function is declared with a **void**, it means there is**no return value**, it just does something.
* **Here is an example of simple void function declaration:**

         void displayMessage();

         void addTwoNumbers(int, int);

4.1.2 Returning Values from Functions

**💻 Returning Values from Functions**

**📀 Returning Values from Functions**

* If you want the function to return a value, you can a data type ( such as**int, string**, etc.) instead of void, and use the **return** keyword inside the function.
* **Here is an example of returning values from function declaration:**

               int  addTwoNumbers(int, int);

4.1.3 Function Call by Value

**💻 Function - Call by Value**

**📀Function Call by Value**

* This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

4.1.4 Function Call by Reference

**💻 Function - Call by Reference**

**📀Function Call by Reference**

* This method copies the address of an argument into the formal parameter.
* Inside the function, the address is used to access the actual argument used in the call.
* This means that changes made to the parameter affect the argument.
* The address of the argument is passed to the function. Return more than one values.
* Any changes made to the formal parameter will affect the actual parameter
* The reference uses **&** symbol.

Module 4 Summary -2

**📔Module 4 Summary or Quick Review**

* A function is a block of code which only runs when it is called.
* A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.
* Voidfunctions are functions that do not return a value.
* Function  Call by Value method copies the actual value of an argument into the formal parameter of the function.
* Function  Call by Value changes made to the formal parameter does not affect the actual parameter
* Function  Call by Reference method copies the address of an argument into the formal parameter.
* Function  Call by Reference uses & symbol
* Function  Call by Reference - any changes made to the formal parameter will affect the actual parameter

### ****Arrays****

#### ****📀C ++ Arrays****

* Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
* C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type.
* An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
* A group of consecutive memory locations of the same name and type, structures of related data items.
* All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.
* Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

#### ****📀Declaring Arrays using Single-Dimensional and Multi-Dimensional(Two-Dimensional)****

* To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store.

**1)  Single-Dimensional Arrays Syntax:**

  arrayType ArrayName[arraySize];

* This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare a 5-element array called balance of type integer, use this statement

### ****Single-Dimensional Arrays****

#### ****📀Declaring  Single-dimensional Arrays****

* To declare an array, define the variable type, specify the name of the array followed by **square brackets** and specify the number of elements it should store.

Syntax:  type ArrayName[arraySize];

* This is called a single-dimension array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C++ data type. For example, to declare a 10-element array called balance of type double, use this statement

### ****Two-Dimensional Arrays****

#### ****📀 Two-Dimensional Arrays****

* The simplest form of the multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size x,y, you would write something as follows:

#### ****📀Declaring Two-Dimensional Arrays****

Syntax:

type arrayName[x][y];

* Where **type** can be any valid C++ data type and **arrayName** will be a valid C++ identifier.
* A two-dimensional array can be think as a table, which will have x number of rows and y number of columns. A 2-dimensional array**num**, which contains three rows and three columns can be shown as below.

  INSERT PICTURE OF 3X3 ARRAY

* Thus, every element in array a is identified by an element name of the form **a[ i ][ j ]**, where a is the name of the array, and i and j are the subscripts that uniquely identify each element in a.

#### ****📀Initializing Two-Dimensional Arrays****

* Two-dimensional arrays may be initialized by specifying bracketed values for each row.

Example:

int num[2][2] = {{1,1},{2,2}};

#### ****📀Accessing Two-Dimensional Array Elements****

* An element in 2-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.

For example  int val = num[2][1];