Module 1: Introduction to C++ and Programming Concepts

1. Introduction to Programming

What is Programming?

Programming is the process of creating a set of instructions that tell a computer how to perform a task. These instructions are written in programming languages like C++. Programming involves designing algorithms, writing code, testing, and debugging.

---

2. Basic Syntax and Structure

Every C++ program follows a specific structure which includes:

- **Preprocessor Directives**: Instructions for the compiler to include necessary libraries.
- **The main() Function**: The entry point of every C++ program.
- **Input and Output**: Using cin and cout to interact with the user.

Basic Program Example:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, World!" << endl;  // Display output
    return 0;
}
```

Explanation:

- #include <iostream>: Includes the input/output stream.
- using namespace std;: Allows us to use cout and cin without prefixing them with std::.
- cout: Outputs data to the console.
- endl: Inserts a newline character.

---

3. Comments in C++

Single-line comment

```
// This is a single-line comment
```

Multi-line comment

```
/* This is a
   multi-line comment */
```

Program Example with Comments:

```
#include <iostream>
using namespace std;

int main() {
    // Output message to the console
    cout << "Hello, World!" << endl;  // Single-line comment
    /* This program demonstrates the use of comments in C++ */
    return 0;
}
```

---

4. Data Types and Variables

Variables in C++ are used to store data of different types, such as int, float, char, etc.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int age = 25;           // Integer variable
    float height = 5.9;     // Floating-point variable
    char grade = 'A';       // Character variable
    bool isStudent = true;  // Boolean variable

    cout << "Age: " << age << endl;
    cout << "Height: " << height << endl;
    cout << "Grade: " << grade << endl;
    cout << "Is Student: " << isStudent << endl;

    return 0;
}
```

---

5. Constants

Constants are variables whose values cannot be changed during the execution of the program.

Example:

```
#include <iostream>
using namespace std;

int main() {
    const float PI = 3.14159;  // Constant declaration
    float radius = 5.0;
    float area = PI * radius * radius;

    cout << "Area of the circle: " << area << endl;
    return 0;
}
```

---

6. Input and Output (cin, cout)

cin is used to take input from the user, and cout is used to display output.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int age;
    cout << "Enter your age: ";  // Prompt the user
    cin >> age;  // Read input from the user
    cout << "Your age is: " << age << endl;  // Display the entered value
    return 0;
}
```

Module 2: Control Flow and Decision Making

1. Operators in C++

Operators are used to perform operations on variables and values.

Arithmetic Operators Example:

```
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 3;
    cout << "Sum: " << a + b << endl;
    cout << "Difference: " << a - b << endl;
    cout << "Product: " << a * b << endl;
    cout << "Quotient: " << a / b << endl;
    cout << "Remainder: " << a % b << endl;
    return 0;
}
```

Relational and Logical Operators Example:

```
cpp
Copy code
#include <iostream>
using namespace std;

int main() {
    int x = 5, y = 10;

    if (x < y && y > 0) {
        cout << "Both conditions are true." << endl;
    }
    if (x != y) {
        cout << "x and y are not equal." << endl;
    }
    return 0;
}
```

---

2. Conditional Statements

Conditional statements control the flow of the program based on certain conditions.

if, else if, else Example:

```
#include <iostream>
using namespace std;

int main() {
    int number;
    cout << "Enter a number: ";
    cin >> number;

    if (number > 0) {
        cout << "The number is positive." << endl;
    } else if (number == 0) {
        cout << "The number is zero." << endl;
    } else {
        cout << "The number is negative." << endl;
    }
    return 0;
}
```

switch-case Example:

```
#include <iostream>
using namespace std;

int main() {
    int day;
    cout << "Enter a day (1-7): ";
    cin >> day;

    switch(day) {
        case 1: cout << "Monday" << endl; break;
        case 2: cout << "Tuesday" << endl; break;
        case 3: cout << "Wednesday" << endl; break;
        case 4: cout << "Thursday" << endl; break;
        case 5: cout << "Friday" << endl; break;
        case 6: cout << "Saturday" << endl; break;
        case 7: cout << "Sunday" << endl; break;
        default: cout << "Invalid day!" << endl;
    }
    return 0;
}
```

---

3. Loops and Iteration

Loops allow us to repeat a block of code multiple times.

for Loop Example:

```
#include <iostream>
using namespace std;
```

```
int main() {
    for (int i = 1; i <= 5; i++) {
        cout << "Number: " << i << endl;
    }
    return 0;
}
```

while Loop Example:

cpp
Copy code

```
#include <iostream>
using namespace std;

int main() {
    int i = 1;
    while (i <= 5) {
        cout << "Number: " << i << endl;
        i++;
    }
    return 0;
}
```

Module 3: Functions

1. Function Definition and Declaration

A function is a block of code designed to perform a specific task.

Example:

```
#include <iostream>
using namespace std;

// Function declaration
int add(int a, int b);

int main() {
    int result = add(5, 10);  // Function call
    cout << "Sum: " << result << endl;
    return 0;
}

// Function definition
```

```
int add(int a, int b) {
    return a + b;
}
```

---

2. Function Parameters (Pass by Value and Pass by Reference)

Pass by Value Example:

```
#include <iostream>
using namespace std;

void add(int x) {
    x = x + 5;  // This only changes the local copy of x
    cout << "Inside function: " << x << endl;
}

int main() {
    int x = 10;
    add(x);  // Pass by value
    cout << "Outside function: " << x << endl;  // Original x remains unchanged
    return 0;
}
```

Pass by Reference Example:

```
#include <iostream>
using namespace std;

void add(int &x) {  // Pass by reference
    x = x + 5;  // This changes the actual x
    cout << "Inside function: " << x << endl;
}

int main() {
    int x = 10;
    add(x);  // Pass by reference
    cout << "Outside function: " << x << endl;  // x is now changed
    return 0;
}
```

Module 4: Arrays and Strings

1. Declaration and Initialization of Arrays

Arrays are used to store multiple values in a single variable.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int arr[5] = {1, 2, 3, 4, 5};  // Initializing array

    for (int i = 0; i < 5; i++) {
        cout << "Element " << i + 1 << ": " << arr[i] << endl;  // Accessing array
elements
    }
    return 0;
}
```

---

2. Multi-Dimensional Arrays

Multi-dimensional arrays allow us to store data in a grid or matrix format.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};
```

Module 4: Arrays and Strings (Continued)

2. Multi-Dimensional Arrays (Continued)

Example:

```
#include <iostream>
using namespace std;

int main() {
    int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} };  // Initializing a 2x3 matrix

    // Display the matrix
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            cout << matrix[i][j] << " ";  // Accessing matrix elements
        }
        cout << endl;  // Move to the next line after each row
    }
    return 0;
}
```

Explanation:

- matrix[2][3]: Declares a 2D array with 2 rows and 3 columns.
- Nested loops are used to iterate through each element and display it.

Module 5: Strings

1. Introduction to Strings

Strings in C++ are objects of the string class in the Standard Library. They are used to represent sequences of characters.

Example:

```
#include <iostream>
#include <string>  // Include the string library
using namespace std;

int main() {
    string str = "Hello, World!";  // String initialization

    cout << "String: " << str << endl;  // Display the string
    cout << "Length: " << str.length() << endl;  // Display the length of the string

    // Accessing individual characters
    for (int i = 0; i < str.length(); i++) {
        cout << "Character at position " << i << ": " << str[i] << endl;
    }

    return 0;
}
```

Explanation:

- #include <string>: Includes the string library.

- str.length(): Returns the number of characters in the string.
- str[i]: Accesses individual characters of the string.

---

2. String Operations

Strings support various operations such as concatenation, comparison, and finding substrings.

Example:

```
#include <iostream>
#include <string>
using namespace std;

int main() {
    string str1 = "Hello";
    string str2 = "World";
    string result;

    // Concatenation
    result = str1 + " " + str2;
    cout << "Concatenated String: " << result << endl;

    // Comparison
    if (str1 == str2) {
        cout << "Strings are equal." << endl;
    } else {
        cout << "Strings are not equal." << endl;
    }

    // Substring
    string sub = result.substr(6, 5);  // Get substring starting at index 6 with length 5
    cout << "Substring: " << sub << endl;

    return 0;
}
```

Explanation:

- str1 + " " + str2: Concatenates two strings.
- result.substr(6, 5): Extracts a substring starting from index 6 with length 5.

Module 6: Advanced Topics

1. Pointers

Pointers are variables that store memory addresses. They are used to directly access and manipulate memory.

Example:

```
#include <iostream>
using namespace std;

int main() {
    int num = 10;
    int *ptr = &num;  // Pointer to integer

    cout << "Value of num: " << num << endl;
    cout << "Address of num: " << &num << endl;
    cout << "Value of ptr: " << ptr << endl;  // Address stored in pointer
    cout << "Value at ptr: " << *ptr << endl;  // Dereferencing pointer to get value

    return 0;
}
```

Explanation:

- int *ptr = &num;: ptr is a pointer that stores the address of num.
- *ptr: Dereferences the pointer to access the value stored at the address.

---

2. Classes and Objects

Classes in C++ are used to define new data types that can encapsulate data and functions.

Example:

```
#include <iostream>
using namespace std;

class Rectangle {
public:
    int width, height;
    void setValues(int w, int h) {
        width = w;
        height = h;
    }

    int area() {
        return width * height;
    }
};

int main() {
    Rectangle rect;
    rect.setValues(5, 10);  // Set width and height
    cout << "Area of Rectangle: " << rect.area() << endl;  // Calculate and display area

    return 0;
}
```

Explanation:

- class Rectangle: Defines a class with data members and member functions.
- setValues(): Sets the dimensions of the rectangle.
- area(): Calculates the area of the rectangle.