Module #3: Introduction to Object-Oriented Programming (OOP) in C++

Introduction to C++

LAB EXERCISES

---

1. First C++ Program: Hello World

Program:

```
#include <iostream>

using namespace std;



int main() {

    cout << "Hello, World!" << endl;

    return 0;

}
```

Objective:
Understand the basic structure of a C++ program, including #include, main(), and cout.

---

2. Basic Input/Output

Program:

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

int main() {
    string name;
    int age;

    cout << "Enter your name: ";
    cin >> name;

    cout << "Hello, " << name << "!" << endl;

    cout << "Enter your age: ";
    cin >> age;

    cout << "You are " << age << " years old." << endl;
    return 0;
}
```

Objective:

Practice input/output operations using cin and cout.

---

3. POP vs. OOP Comparison Program

a) Procedural Approach (POP):

```
#include <iostream>

using namespace std;



int main() {

   float length, width;

   cout << "Enter length and width: ";

   cin >> length >> width;

   float area = length * width;

   cout << "Area of rectangle: " << area << endl;

   return 0;

}
```

b) Object-Oriented Approach (OOP):

```
#include <iostream>

using namespace std;
```

```
class Rectangle {

private:

   float length, width;

public:

   void setData(float l, float w) {

      length = l;

      width = w;

   }

   float getArea() {

      return length * width;

   }

};



int main() {

   Rectangle r;

   float l, w;

   cout << "Enter length and width: ";
```

```
    cin >> l >> w;
    r.setData(l, w);
    cout << "Area of rectangle: " << r.getArea() << endl;
    return 0;
}
```

Objective:
Highlight the difference between Procedural Programming and Object-Oriented Programming.

---

4. Setting Up Development Environment

Program:

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

int main() {
    int a, b;
    cout << "Enter two numbers: ";
    cin >> a >> b;
    cout << "Sum = " << a + b << endl;
    return 0;
}
```

Objective:
Understand how to install, configure, and run programs in an IDE (e.g., CodeBlocks, Dev C++).

THEORY EXERCISES

1. **Differences Between POP and OOP:**
   - POP: Functions and logic are written sequentially.
   - OOP: Data and functions are bundled into objects.
2. **Advantages of OOP:**
   - Encapsulation
   - Reusability through inheritance
   - Abstraction
   - Better code organization
3. **Setting Up a C++ Development Environment:**
   - Install IDE (e.g., CodeBlocks)
   - Set compiler path
   - Create and run new projects
4. **Input/Output Operations:**

- cin for input
  Example: cin >> name;

- cout for output
  Example: cout << "Hello";

Variables, Data Types, and Operators

LAB EXERCISES

1. Variables and Constants

Program:

```
#include <iostream>

using namespace std;



int main() {

    int age = 25;

    float salary = 55000.50;

    const float PI = 3.14159;
    cout << "Age: " << age << "\n";

    cout << "Salary: " << salary << "\n";

    cout << "PI: " << PI << "\n";



    return 0;

}
```

Objective:
Understand the difference between variables and constants.

---

2. Type Conversion

Program:

```
#include <iostream>

using namespace std;



int main() {

    int a = 5;

    float b = 2.5;



    // Implicit conversion
    float result = a + b;

    cout << "Implicit: " << result << endl;



    // Explicit conversion

    int result2 = a + (int)b;

    cout << "Explicit: " << result2 << endl;



    return 0;

}
```

Objective:
Practice type casting in C++.

---

3. Operator Demonstration

Program:

```
#include <iostream>

using namespace std;



int main() {

    int a = 5, b = 2;
    // Arithmetic

    cout << "Addition: " << a + b << endl;



    // Relational

    cout << "Is a > b? " << (a > b) << endl;



    // Logical

    cout << "Logical AND: " << ((a > 0) && (b > 0)) << endl;



    // Bitwise

    cout << "Bitwise AND: " << (a & b) << endl;



    return 0;

}
```

Objective:
Understand different types of operators in C++.

THEORY EXERCISES

1. **Data Types in C++:**
    - int, float, char, double, string, bool
2. **Implicit vs. Explicit Type Conversion:**
    - Implicit: Done automatically by compiler
    - Explicit: Done manually by programmer
3. **Types of Operators:**
    - Arithmetic: +, -, *, /
    - Relational: ==, !=, >, <
    - Logical: &&, ||, !
    - Bitwise: &, |, ^, ~
4. **Constants and Literals:**
    - Constants: const int MAX = 100;
    - Literals: Fixed values like 10, 3.14, 'A'

Control Flow Statements

LAB EXERCISES

---

1. Grade Calculator

Program:

```
#include <iostream>

using namespace std;



int main() {

    int marks;

    cout << "Enter marks: ";

    cin >> marks;



    if (marks >= 90) cout << "Grade: A";

    else if (marks >= 75) cout << "Grade: B";

    else if (marks >= 60) cout << "Grade: C";

    else cout << "Grade: F";



    return 0;

}
```

2. Number Guessing Game

Program:

```
#include <iostream>

using namespace std;



int main() {

    int secret = 42, guess;

    while (true) {

        cout << "Guess the number (1-100): ";

        cin >> guess;



        if (guess == secret) {

            cout << "Correct!";

            break;

        } else if (guess < secret) {

            cout << "Too low!" << endl;

        } else {

            cout << "Too high!" << endl;
        }
    }
    return 0;
}
```

---

3. Multiplication Table

Program:

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

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

    for (int i = 1; i <= 10; i++) {
        cout << num << " x " << i << " = " << num * i << endl;
    }
    return 0;

}
```

---

4. Star Triangle

Program:

```
#include <iostream>

using namespace std;



int main() {

    int rows;

    cout << "Enter number of rows: ";

    cin >> rows;



    for (int i = 1; i <= rows; i++) {

        for (int j = 1; j <= i; j++) {

            cout << "*";

        }

        cout << endl;
```

```
   }

   return 0;

}
```

THEORY EXERCISES

1. **Conditional Statements:**
   - if, if-else, switch
2. **Loop Types:**
   - for: Fixed iteration
   - while: Condition checked before loop
   - do-while: Condition checked after loop
3. **Break and Continue:**
   - break: exits loop
   - continue: skips to next iteration
4. **Nested Control Structures:**
   Example: Nested for loop for pattern printing.

Functions and Scope

LAB EXERCISES

---

1. Simple Calculator Using Functions

Program:

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

int add(int a, int b) { return a + b; }
int sub(int a, int b) { return a - b; }
int mul(int a, int b) { return a * b; }
float divi(int a, int
```

2. Factorial Calculation Using Recursion

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

int factorial(int n) {
    if (n == 0) return 1;
    else return n * factorial(n - 1);
}



int main() {

    int num;

    cout << "Enter a number: ";

    cin >> num;

    cout << "Factorial of " << num << " is " << factorial(num);

    return 0;

}
```

🎯 **Objective:** Understand recursion in functions.

---

3. Variable Scope

```
#include <iostream>

using namespace std;



int globalVar = 100; // Global variable



void showScope() {
    int localVar = 50; // Local variable

    cout << "Local Variable: " << localVar << endl;

    cout << "Global Variable inside function: " << globalVar << endl;

}



int main() {

    showScope();

    cout << "Global Variable in main: " << globalVar << endl;

    return 0;

}
```

🎯 **Objective:** Reinforce the concept of variable scope.

📝 **THEORY EXERCISES**

1. **What is a function in C++?**
   A function is a block of code that performs a specific task.

   - **Declaration:** int sum(int a, int b);

   - **Definition:**

   - int sum(int a, int b) {

   -     return a + b;

   - }

   - **Calling:** sum(3, 4);

2. **Scope of Variables in C++:**

   - **Local Scope:** Declared inside functions or blocks, accessible only there.

   - **Global Scope:** Declared outside all functions, accessible from any function.

3. **Explain Recursion:**
   A function calling itself.
   Example:

4. int factorial(int n) {

5.   if(n == 0) return 1;

6.   return n * factorial(n - 1);

7. }

8. **Function Prototypes in C++:**
   A declaration of a function before its use.
   Used to inform the compiler about the function’s name and parameters.
   Example:

9. int sum(int, int); // Prototype

❖ **Arrays and Strings**

💻 **LAB EXERCISES**

---

1. Array Sum and Average

```
#include <iostream>

using namespace std;



int main() {

    int arr[5], sum = 0;

    for(int i = 0; i < 5; i++) {

        cout << "Enter element " << i+1 << ": ";

        cin >> arr[i];

        sum += arr[i];

    }

    cout << "Sum = " << sum << endl;

    cout << "Average = " << sum / 5.0 << endl;

    return 0;

}
```

🎯 **Objective:** Understand basic array manipulation.

---

2. Matrix Addition

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


int main() {
    int a[2][2], b[2][2], c[2][2];


    cout << "Enter 4 elements of first 2x2 matrix:\n";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            cin >> a[i][j];


    cout << "Enter 4 elements of second 2x2 matrix:\n";
    for (int i = 0; i < 2; i++)
        for (int j = 0; j < 2; j++)
            cin >> b[i][j];


    cout << "Resultant matrix:\n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            c[i][j] = a[i][j] + b[i][j];
            cout << c[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

🎯 **Objective:** Practice multi-dimensional arrays.

---

3. String Palindrome Check

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


int main() {
    string str, rev = "";
    cout << "Enter a string: ";
    cin >> str;


    for(int i = str.length() - 1; i >= 0; i--)

        rev += str[i];


    if(str == rev)

        cout << "Palindrome!";

    else

        cout << "Not a palindrome!";

    return 0;

}
```

🎯 **Objective:** Practice string operations.

📝 **THEORY EXERCISES**

1. **What are arrays?**
   Arrays store multiple elements of the same type.

   - **1D:** int arr[5];

   - **2D:** int matrix[2][3];

2. **String Handling:**
   Use #include <string>
   Example:

3. string name = "Alice";

4. cout << name.length();

5. **Array Initialization:**

   - 1D: int arr[3] = {1, 2, 3};

   - 2D: int matrix[2][2] = {{1, 2}, {3, 4}};

6. **String Functions:**

   - length(), substr(), compare(), find(), append()

🔷 **Introduction to Object-Oriented Programming**

💻 **LAB EXERCISES**

---

1. Class for a Simple Calculator

#include <iostream>

using namespace std;

```
class Calculator {
public:
    int add(int a, int b) { return a + b; }
    int sub(int a, int b) { return a - b; }
    int mul(int a, int b) { return a * b; }
    float div(float a, float b) { return a / b; }
};


int main() {
    Calculator calc;
    int a = 10, b = 5;
    cout << "Add: " << calc.add(a, b) << endl;
    cout << "Sub: " << calc.sub(a, b) << endl;
    cout << "Mul: " << calc.mul(a, b) << endl;
    cout << "Div: " << calc.div(a, b) << endl;
    return 0;
}
```

🎯 **Objective:** Introduce basic class structure.

2. Class for Bank Account

```
#include <iostream>

using namespace std;



class BankAccount {

private:

    float balance;



public:

    BankAccount() { balance = 0; }



    void deposit(float amount) {

        balance += amount;

    }



    void withdraw(float amount) {

        if (amount <= balance)
```

```
            balance -= amount;
        else
            cout << "Insufficient balance!" << endl;
    }

    void display() {
        cout << "Current Balance: " << balance << endl;
    }
};

int main() {
    BankAccount account;
    account.deposit(1000);
    account.withdraw(500);
    account.display();
    return 0;
}
```

Objectives Understand encapsulation in classes.

3. Inheritance Example

```
#include <iostream>

using namespace std;



class Person {

public:

   string name;

   void getName() {

      cout << "Enter name: ";

      cin >> name;

   }

   void showName() {

      cout << "Name: " << name << endl;

   }

};



class Student : public Person {
```

```
public:

    void showRole() {

        cout << "I am a student." << endl;

    }

};



class Teacher : public Person {

public:

    void showRole() {

        cout << "I am a teacher." << endl;

    }

};



int main() {

    Student s;

    s.getName();

    s.showName();

    s.showRole();
    Teacher t;

    t.getName();

    t.showName();

    t.showRole();

    return 0;

}
```

Objective: Learn the concept of inheritance.

THEORY EXERCISES

1. **Key OOP Concepts:**
   - **Encapsulation**
   - **Abstraction**
   - **Inheritance**
   - **Polymorphism**
2. **Classes and Objects:**
   - Class: Blueprint
   - Object: Instance of class
     Example:

3. class Car { public: void start() { } };

4. Car myCar; myCar.start();

5. **Inheritance:**
   Enables code reuse.

6. class A { };

7. class B : public A { };

8. **Encapsulation:**
   Wrapping data & functions in a class and hiding implementation details using private access.