Module #3 Introduction to OOPS Programming

1. Introduction to C++

THEORY EXERCISE:

1. What are the key differences between Procedural Programming and ObjectOrientedProgramming (OOP)?

- **Approach:** POP focuses on functions and procedures, while OOP focuses on objects and classes.
- **Data Handling:** POP has global data accessible by all functions, whereas OOP encapsulates data inside objects.
- **Modularity:** POP is less modular, making code harder to manage, while OOP promotes modularity with classes.
- **Reusability:** OOP supports reusability through inheritance, which POP lacks.

2. List and explain the main advantages of OOP over POP.

- **Encapsulation: P**rotects data by keeping it private.
- **Modularity:** Code is organized into objects, making it more manageable.
- **Reusability:** Inheritance allows code reuse, reducing redundancy.
- **Maintainability:** Easier to modify and maintain due to modular structure.
- **Abstraction:** Hides complexity, showing only essential features.

3. Explain the steps involved in setting up a C++ development environment.

1. **Install IDE:** Download and install IDE like Dev C++, Code::Blocks, or Visual Studio.
2. **Install Compiler:** Ensure a C++ compiler (like GCC) is installed (often bundled with IDEs).
3. **Configure IDE:** Set compiler path and project settings if needed.
4. **Create a Project:** Open IDE, create a new project or file, and write code.
5. **Compile and Run:** Compile the code to check for errors, then run the program.

4. What are the main input/output operations in C++? Provide examples.

Input: Use cin to take user input

```
int age;
std::cin >> age;
```

Output: Use cout to display output.

```
std::cout << "Your age is: " << age << std::endl;
```

2. Variables, Data Types, and Operators

THEORY EXERCISE:

1. What are the different data types available in C++? Explain with examples.

- **Integer Types:** Store whole numbers.

1. int age = 20; // Stores integers
2. short s = 100; // Short integer
3. long l = 100000L; // Long integer

- **Floating-point Types:** Store decimal numbers.

1. float height = 5.9f; // Single precision
2. double pi = 3.14159; // Double precision

- **Character Type:** Holds a single character.

1. char grade = 'A'; // Stores one character

- **Boolean Type:** Holds true or false.

1. bool isStudent = true; // Stores true/false

- **String Type (from <string>):** Holds sequences of characters.

1. #include <string>
2. std::string name = "Udit"; // Stores text

- **Void Type:** Represents no value, mainly used for functions.

1. void displayMessage(); // Function returning no value

- **Derived Types:**

Array: Collection of elements of the same type

1. int numbers[5] = {1, 2, 3, 4, 5};

Pointer: Holds memory address

1. int x = 10;
2. int* ptr = &x; // Pointer to x

Reference: Alias for another variable

1. int y = 100;
2. int &ref = y; // ref is a reference to y

2. Explain the difference between implicit and explicit type conversion in C++.

Implicit Conversion: Automatically done by the compiler when converting smaller data types to larger ones.

```
int a = 10;
double b = a;  // Implicit conversion (int to double)
```

Explicit Conversion: Manually done using type casting.

```
double pi = 3.14159;
int intPi = (int)pi;  // Explicit conversion (double to int)
```

3. What are the different types of operators in C++? Provide examples of each.

- **Arithmetic: +, -, *, /, %**

```
int sum = 5 + 3;
```

- **Relational: ==, !=, >, <, >=, <=cpp**

```
cout << (5 > 3);  // true (1)
```

- **Logical: &&, ||, !cpp**

```
cout << (5 > 3 && 3 > 2);  // true (1)
```

- **Bitwise: &, |, ^, ~, <<, >>cpp**

```
cout << (5 & 3);  // Bitwise AND
```

- **Assignment: =, +=, -=, *=, /=cpp**

```
int x = 5;
x += 3;  // x = 8
```

- **Unary: ++, --, -cpp**

```
x++;
```

- **Ternary: condition ? true_value : false_valuecpp**

```
int max = (5 > 3) ? 5 : 3;
```

4. Explain the purpose and use of constants and literals in C++.

Constants: Fixed values that cannot be changed

const double PI = 3.14159;

Literals: Direct values assigned to variables.

- Integer: int age = 20;
- Float: double pi = 3.14159;
- Char: char grade = 'A';
- String: string name = "Udit";
- Boolean: bool isStudent = true;

Constants ensure data integrity, and literals represent fixed values in code.

3. Control Flow Statements

1. What are conditional statements in C++? Explain the if-else and switch statements.

Conditional statements in C++ control the flow of execution based on conditions.

- **if-else:** Executes a block of code if the condition is true; otherwise, executes the "else" block.

```
if (x > 0) {
    cout << "Positive";
} else {
    cout << "Negative";
}
```

- **switch:** Selects one of many blocks to execute based on the value of an expression

```
switch (choice) {
    case 1: cout << "One"; break;
    case 2: cout << "Two"; break;
    default: cout << "Other";
}
```

2. What is the difference between for, while, and do-while loops in C++?

Loop Types:

- **for:** Used when the number of iterations is known.

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

- **while:** Repeats as long as the condition is true.

```
int i = 0; while (i < 5) { cout << i++; }
```

- **do-while:** Executes at least once before checking the condition.

```
int i = 0; do { cout << i++; } while (i < 5);
```

3. How are break and continue statements used in loops? Provide examples.

- **break:** Exits the loop immediately.

code :

```
for (int i = 0; i < 5; i++) {
  if (i == 3) break;  // Stops when i is 3
  cout << i << " ";
}
```

- **continue:** Skips the remaining code and moves to the next iteration.

code :

```
for (int i = 0; i < 5; i++) {
   if (i == 2) continue;  // Skips 2
   cout << i << " ";
}
```

4. Explain nested control structures with an example.

Nested Control Structures:

Control structures (loops or conditionals) inside other control structures.

Example:

```
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= i; j++) {
        cout << "* ";
    }
    cout << endl;
}
// Output:// * // * * // * * *
```

4. Functions and Scope

1. What is a function in C++? Explain the concept of function declaration, definition, and calling.

Function in C++:

A function is a block of code that performs a specific task.

- **Declaration:** Tells the compiler about the function’s name, return type, and parameters (done before main()).

```
int add(int, int);  // Declaration
```

- **Definition:** Contains the actual code of the function.

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

- **Calling:** Invokes the function to perform its task.

```
int result = add(5, 10);  // Calling
```

2. What is the scope of variables in C++? Differentiate between local and global scope.

Scope of Variables:

- **Local Scope:** Variables declared inside a function/block are accessible only within that function/block.

code

```
void func() { int x = 10; }  // x is local to func()
```

- **Global Scope:** Variables declared outside all functions are accessible throughout the program.

code

```
int x = 100;  // Global variablevoid func() {
cout << x; }  // Access global x
```

3. Explain recursion in C++ with an example.

Recursion in C++:
A function that calls itself to solve smaller subproblems.

Example: Factorial Calculation

code

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

4. What are function prototypes in C++? Why are they used?

Function Prototypes:

A function prototype is a declaration of a function before its use, informing the compiler about its signature.

- **Purpose:** Ensures the function is recognized even if defined later, supporting modular programming.
- **Example:**

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

5. Arrays and Strings

1. What are arrays in C++? Explain the difference between single-dimensional and multi-dimensional arrays

Arrays are collections of elements of the same data type stored in contiguous memory locations.

Arrays in C++:

- **Single-dimensional:** Stores elements in a single row.

code

```
int arr[5] = {1, 2, 3, 4, 5};
```

- **Multi-dimensional:** Stores elements in a grid (rows and columns).

code

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

2. Explain string handling in C++ with examples.

String Handling in C++:

- **Using char arrays:**

```
char str[20] = "Hello";
cout << str;
```

- **Using string class:**

```
#include <string>
string s = "World";
cout << s;
```

3. How are arrays initialized in C++? Provide examples of both 1D and 2D arrays.

Array Initialization:

- **1D Array:**

```
int arr[5] = {1, 2, 3, 4, 5};
```

- **2D Array:**

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

4. Explain string operations and functions in C++.

- **Length:**

```
string s = "Hello";
cout << s.length();
```

- **Concatenation:**

```
string a = "Hi ", b = "there!";
cout << a + b;
```

- **Substring:**

```
string s = "HelloWorld";
cout << s.substr(0, 5);  // Output: Hello
```

- **Compare:**

```
string s1 = "abc", s2 = "xyz";
if (s1 == s2) cout << "Equal"; else cout << "Not Equal";
```

6. Introduction to Object-Oriented Programming

1. Explain the key concepts of Object-Oriented Programming (OOP).

Key Concepts of OOP:

- **Encapsulation**: Wrapping data and methods into a single unit (class).
- **Inheritance**: Deriving new classes from existing ones.
- **Polymorphism**: Same function behaves differently based on context.
- **Abstraction**: Hiding complex details and showing only essentials.

2. What are classes and objects in C++? Provide an example.

Classes and Objects:

- **Class:** Blueprint for creating objects.
- **Object:** Instance of a class.

code :

```
class Car {
public:
  string brand;
  void showBrand() { cout << brand; }
};
Car myCar;
myCar.brand = "Toyota";
myCar.showBrand();
```

3. What is inheritance in C++? Explain with an example.

Inheritance:

One class acquires properties of another.

code:

```
class Animal {
public:
    void sound() { cout << "Animal Sound";
}
};
class Dog : public Animal {};  // Dog
inherits Animal
Dog d;
d.sound();  // Inherited method
```

4. What is encapsulation in C++? How is it achieved in classes?

Encapsulation:
Protects data by keeping it private and accessing it through public methods.

code :

```
class BankAccount {
private:
   int balance;
public:
   void setBalance(int b) { balance = b; }
   int getBalance() { return balance; }
};
```

// ALL LAB EXERCISES ARE DONE IN DEV C++

THE END

THANK YOU !!