1. INTRODUCTION TO PROGRAMMING PARADIGMS & OOP CONCEPTS

Title: Introduction to Programming Paradigms and Object-Oriented Programming

Introduction: Programming paradigms define the style and structure of writing programs. Traditional programming focuses on functions and logic, while Object Oriented Programming (OOP) focuses on real-world modeling using objects and classes. C++ is a powerful language that supports OOP and allows programmers to build modular, secure, and reusable software.

Learning Objectives: After completing this module, learners will be able to,

- Understand programming paradigms
- Define class and object
- Differentiate between structure and class
- Understand data members and member functions
- Explain key characteristics of OOP

Main Content

- **Programming Paradigms:** Procedural, Object-Oriented, Generic
- **Class:** Blueprint of an object
- **Object:** Instance of a class
- **Objects as variables** of class data type
- **Data Members:** Variables inside a class
- **Member Functions:** Functions inside a class
- **Access Specifiers:**
  - `public` – accessible everywhere
  - `private` – accessible only inside the class

Structure vs Class

Structure	Class
Members are public by default	Members are private by default
Used in C	Used in C++
No data hiding	Supports data hiding

Characteristics of OOP

- **Encapsulation:** Binding data and functions together
- **Data Hiding:** Restricting access to data
- **Data Security:** Controlled access using access specifiers

Example:

```
class Student
        {
                private:
                        int roll;
                public:
                         void setRoll(int r) { roll = r; }
                        int getRoll() { return roll; }
        };
```

Activity: Identify class, object, data member, and member function from the above example.

Assessment

1. Define object and class
2. Difference between structure and class

Summary: OOP helps in building secure and reusable programs by modelling real-world entities using classes and objects.

2. BASICS OF C++ PROGRAMMING

Title: Structure and Basics of C++ Programs
Introduction: C++ programs follow a specific structure. Understanding program flow, object creation, constructors, and destructors is essential for effective programming.
Learning Objectives: Learners will be able to,

- Write basic C++ programs
- Create classes and objects
- Understand constructors and destructors
- Use `cin` and `cout`

Main Content

Structure of C++ Program

- Header files
- `main()` function

- Class definition
- Object creation

Creating Objects

```
Student s1;
```

Constructors

- Special function with the same name as the class
- Initialises objects automatically

Destructor

- Cleans the memory when the object is destroyed

Input/Output

- `cin` – input
- `cout` – output

Example:

```
class Test
  {
     Public:
          Test() { cout << "Constructor"; }
          ~Test() { cout << "Destructor"; }
  };
```

Activity: Write a program using a constructor and a destructor.
Assessment: What is the role of the constructor?
Summary: Constructors and destructors manage object lifecycle, while `cin` and `cout` help in user interaction.

3. ADVANCED CLASS FEATURES

Title: Advanced Concepts in Classes
Introduction: Advanced class features improve safety, efficiency, and design quality in C++ programs.
Learning Objectives: Students will learn,

- Friend functions and classes
- Static members
- `this` pointer

- Dynamic memory allocation

Main Content

- **Friend Function:** Access private data
- **Static Members:** Shared among all objects
- **this Pointer:** Refers to the current object
- **Dynamic Objects:** `new` and `delete`
- **Constant Objects:** Prevent modification
- **Composition:** Class inside another class

Example:

```
class Sample
  {
    static int count;
  };
```

Activity: Explain why static members are useful.
Assessment: What is the danger of returning a reference to private data?
Summary: Advanced class features enhance flexibility and memory control.

4. OPERATOR OVERLOADING & TEMPLATES

Title: Operator Overloading and Generic Programming
Introduction: Operator overloading allows user-defined meaning for operators, while templates enable generic programming.
Learning Objectives: Learners will,

- Overload operators
- Differentiate member and friend operator functions
- Understand templates

Main Content

- **Operator Overloading Rules**
- Cannot create new operators
- Syntax must be correct
- Stream operators `<<` and `>>`
- Unary and binary operators
- Type conversion

Templates

```
template <class T>
T add(T a, T b) {
    return a + b;
}
```

Activity: Create a template function for multiplication.
Assessment: Why are some operators not overloadable?
Summary: Operator overloading improves readability, and templates promote code reuse.

5. INHERITANCE & POLYMORPHISM

Title: Inheritance, Polymorphism and Virtual Functions
Introduction: Inheritance allows code reuse, while polymorphism enables dynamic behaviour.
Learning Objectives: Students will

- Understand inheritance types
- Use virtual functions
- Understand abstract classes

Main Content

- **Inheritance:** IS-A relationship
- **Types:** Single, Multiple, Multilevel
- **Access Modes:** Public, Private, Protected
- **Virtual Functions:** Runtime binding
- **Abstract Class:** Contains pure virtual function
- **Virtual Destructor:** Proper memory cleanup

Example

```
class Base
 {
     public:
          virtual void show() = 0;
 };
```

Activity: Explain a real-life example of inheritance.
Assessment: What is dynamic binding?

Summary: Inheritance and polymorphism enable flexible and extensible program design.

6. FILE HANDLING, UML & OOA/OOD

Title: File Handling and Object-Oriented Analysis & Design
Introduction: Large systems require planning and documentation using UML and OOA/OOD methods.
Learning Objectives: Learners will:

- Perform file operations
- Understand UML diagrams
- Learn OMT methodology

Main Content

- **File Handling:** Read/write using streams
- **UML Diagrams:** Class, Object, Interaction
- **OOA:** Problem analysis
- **OOD:** Solution design
- **OMT (Rumbaugh):** Object, Dynamic, Functional models
- **Case Studies:** Real-world system modelling

Example:

```
ofstream file("data.txt");
file << "C++ File Handling";
```

Activity: Draw a class diagram for the Library System.
Assessment: Differentiate OOA and OOD.
Summary: Proper analysis and design ensure scalable and maintainable software systems.