UTTARANCHAL UNIVERSITY

(Established vide Uttaranchal University Act, 2012, Uttarakhand Act No. 11 of 2013)
Premnagar-248007, Dehradun, Uttarakhand, INDIA

ASSIGNMENT COVER PAGE

Name of Student:	Rajesh Kayal
Batch:	July 23 Batch
Program:	Master of Computer Applications (OL)
Subject & Code:	OOPS using C++
Semester:	1st
Learner ID:	2316010607

NECESSARY INSTRUCTIONS

1. Cover Page must be filled in Capital Letters. All Fields of the Form are compulsory to be filled.
2. The assignment should be written / computer typed on A4 size paper and it should be neat and clearly readable.
3. The cover page should be stapled at the front of each and every assignment.
4. Incomplete Assignments will not be accepted.

Object-Oriented Programming Vs Procedure-Oriented Programming

In this assignment, we will explore the key differences between Object Oriented Programming (OOP) and Procedure Oriented Programming (POP) using C++ as the programming language.

Introduction

Object-Oriented Programming (OOP) and Procedure-Oriented Programming (POP) are two fundamental paradigms that shape the landscape of software development. Both approaches offer distinct methodologies for structuring and organizing code, each with its own set of principles, concepts, and advantages. Grasping the nuances that differentiate these paradigms is essential for developers to make informed decisions regarding the design and implementation of their software solutions. In this comprehensive overview, we'll delve into the key distinctions between Object-Oriented Programming and Procedure-Oriented Programming, providing a clear understanding of their respective strengths and applications.

☐ Object-Oriented Programming (OOP):

- OOP focuses on representing objects and their interactions.
- It uses classes and objects to encapsulate data and behaviour.
- OOP provides the concepts of inheritance, polymorphism, and encapsulation.
- Code reuse and modularity are emphasised in OOP.

Objects

In OOP, real-world entities are modelled as objects. These objects encapsulate both data and the procedures that operate on that data.

Classes

Classes act as blueprints or templates for creating objects. They define the structure and behaviour that objects of the class will exhibit.

Instantiation

Objects are instances of classes, created by instantiating the class. This instantiation process involves allocating memory and initializing the object.

Attributes and Methods

Objects have attributes (data members) that represent their state and methods (functions) that define their behaviour. This encapsulation ensures data integrity and modularity.

Encapsulation

Encapsulation is a key principle in Object-Oriented Programming (OOP) that involves bundling data and the methods that operate on that data into a single unit, known as a class. The primary characteristics of encapsulation include:

- **Data Hiding:**

  Encapsulation hides the internal details and implementation of an object from the outside world. The internal state of an object is not directly accessible.

- **Access Control:**

  Access to the internal components (attributes and methods) of a class is controlled through access modifiers like public, private, and protected.

- **Modularity:**

  Encapsulation promotes modularity by organizing code into self-contained classes.

- **Security and Data Integrity:**

  By hiding the internal details, encapsulation provides a level of security for the data. Unauthorized access and manipulation of data are minimized, enhancing data integrity.

Polymorphism

Polymorphism is a core principle in Object-Oriented Programming (OOP) that allows objects of different types to be treated as objects of a common type. There are two main types of polymorphism: compile-time (or static) polymorphism and runtime (or dynamic) polymorphism. The primary characteristics of polymorphism include

☐ **Compile-time Polymorphism:**

- Also known as static polymorphism, this occurs during compile-time.
- Achieved through function overloading, where multiple functions with the same name but different parameter lists coexist in the same scope.

☐ **Runtime Polymorphism:**

- Also known as dynamic polymorphism, this occurs during runtime.
- Achieved through function overriding, where a base class defines a method, and a derived class provides a specific implementation of that method

Inheritance

Inheritance is a fundamental concept in Object-Oriented Programming (OOP) that allows a new class (subclass or derived class) to inherit attributes and behaviours from an existing class (base class or superclass). The primary characteristics of inheritance include

☐ **Code Reusability:**

- Inheritance promotes code reuse by allowing a subclass to inherit the properties and methods of a superclass.
- Common functionalities can be defined in a base class, and subclasses can reuse or extend those functionalities

☐ **Hierarchy of Classes:**

- Inheritance establishes a hierarchy of classes, with the base class at the top and subclasses below.
- Subclasses inherit both the public and protected members of the base class.

☐ **Overriding:**

Subclasses have the option to override (redefine) methods inherited from the base class. This allows customization of behaviour while maintaining a common interface.

Design Patterns

In addition to encapsulation, inheritance, and polymorphism, OOP leverages design patterns. These are reusable solutions to common problems encountered in software design. Examples include the Singleton pattern for ensuring a class has only one instance and the Factory pattern for creating objects.

Exception Handling

OOP excels in exception handling through mechanisms like try-catch blocks. This allows developers to gracefully handle errors and exceptions, enhancing the robustness of the code.

User Interface

OOP is prevalent in UI development, where frameworks like Qt and JavaFX utilize concepts such as classes and objects to model UI components. This provides a more modular and extensible approach to building graphical interfaces.

Database Integration

OOP principles extend to database design and integration, enabling the creation of object-relational mappings (ORMs) for seamless interaction between application code and databases.

☐ Procedure-Oriented Programming (POP):

- POP is generally more lightweight in terms of performance.
- Direct function calls and simpler data structures can result in faster execution.
- Well-suited for performance-critical applications with minimal abstraction.

Procedure-oriented programming (POP) is a programming paradigm that organizes a program as a sequence of procedures, also known as routines, functions, or subroutines. In POP, a program is divided into smaller, manageable parts called procedures, each performing a specific task. These procedures are executed sequentially, and data is typically shared among them through parameters or global variables. The focus in POP is on performing operations through a series of well-defined procedures.

Procedures/Functions:

In Procedure-Oriented Programming (POP), functions or procedures play a crucial role in achieving modularity, reusability, and maintainability of code. Here's how functions are typically used in POP.

Modularity:

Functions allow breaking down a program into smaller, manageable units. Each function is responsible for a specific task or operation, promoting a modular design.

Reusability:

Once a function is defined, it can be called from different parts of the program. This promotes code reuse, as the same functionality can be invoked multiple times without rewriting the code.

Readability:

Functions enhance the readability of the code by encapsulating specific operations. Instead of having a monolithic code structure, the program is organized into a set of functions, making it easier to understand and maintain.

Maintenance:

Changes or updates to a specific functionality can be made within the corresponding function, without affecting the rest of the program. This simplifies maintenance and reduces the risk of introducing errors.

Data-Centric Nature

In Procedure-Oriented Programming (POP), the data-centric nature emphasizes the importance of data and the procedures that operate on that data. Unlike Object-Oriented Programming (OOP), where the focus is on encapsulating data and behaviour into objects, POP tends to centre around procedures that manipulate and process data. Here are key points regarding the data-centric nature of POP:

Global Data and Procedures:

In POP, data is often declared globally or passed as parameters to procedures. Procedures act as operations on the data, and the program's structure is determined by the sequence of procedures that manipulate the shared data.

Limited Encapsulation:

While procedures provide a level of encapsulation by bundling related operations together, the emphasis is more on the procedures themselves rather than on creating self-contained, encapsulated units with their own internal state (as in OOP).

Data-Driven Flow:

The flow of control in a POP program is often driven by the data. Procedures are designed to process and transform the data, and the program's logic is structured around the sequence of data manipulations.

Absence of Objects in POP:

- POP Perspective: POP revolves around procedures/functions manipulating data sequentially.
- Data Handling: Data is often global or passed between procedures as parameters.
- Encapsulation: Limited encapsulation within procedures; data may be shared globally.
- Decomposition: Programs decomposed into step-by-step procedures.
- Inheritance/Polymorphism: Not present; reusability achieved through functions.
- Global Data Sharing: Common; data shared among procedures.

Contrast with Object-Oriented Programming (OOP):

- OOP Perspective: Organized around objects encapsulating data and behaviour.
- Encapsulation: Strong encapsulation within objects, promoting data security.
- Decomposition: Programs decomposed into classes and objects.
- Inheritance: Hierarchy of classes for code reuse and relationships.
- Polymorphism: Supported, enabling flexibility with a common interface.
- Global Data Sharing: Reduced reliance; data encapsulated within objects.

Concurrency and Multithreading

While POP can handle concurrent tasks, it often lacks built-in mechanisms for managing multithreading complexities. Direct manipulation of shared data may lead to challenges related to synchronization and thread safety.

Testing and Debugging

In POP, unit testing can be less straightforward, and the absence of features like dependency injection might make it harder to isolate components for testing. Debugging might require a more linear and manual approach.

Tooling and IDEs

POP doesn't enjoy the same level of support from Integrated Development Environments (IDEs) as OOP. IDEs designed for OOP languages provide features like code completion, refactoring tools, and advanced debugging capabilities.

Industry Use Cases

POP is often found in legacy systems or scenarios where simplicity and direct control over procedures are prioritized. Small-scale projects with minimal complexity may benefit from the straightforward nature of POP.

Differences Between Object-Oriented Programming and Procedure-Oriented Programming:

Aspect	Object-Oriented Programming (OOP)	Procedure-Oriented Programming (POP)
Approach to Problem-Solving	Models real-world entities as objects.	Focuses on procedures/functions for sequential operations.
	Emphasizes abstraction and encapsulation.	Break down problems into procedures; procedural approach.
Code Reusability	Achieved through inheritance.	Relies on functions and procedures for code reuse.
	Supports polymorphism for versatile code use.	Code reuse by calling existing procedures.
Readability & Maintainability	Improved readability due to encapsulation.	Readable for sequential operations; linear structure.
Real-world Analogy	Car manufacturing plant: Each car type is a class.	Cooking recipe: Each step is a procedure and the ingredients as data.

Performance Considerations: OOP vs. POP

Aspect	Object-Oriented Programming (OOP)	Procedure-Oriented Programming (POP)
Execution Speed	Slight overhead due to dynamic dispatch.	Generally faster due to direct procedural calls.
Memory Usage	Additional memory overhead for vtables and metadata.	Simpler memory model, reducing overhead.
Suitability for Performance-Critical Applications	Suitable when design extensibility is a priority, even with minor speed impact.	Preferable in performance-critical scenarios prioritizing memory efficiency and maximum speed.
Scenarios Favoring OOP	Large and complex systems emphasizing maintainability.	Collaborative development environments require modular design.
Scenarios Favoring POP	Embedded systems with resource constraints, prioritizing minimal memory usage.	Performance-critical algorithms require direct control over data and execution flow.

Conclusion

In conclusion, C++ with Object-Oriented Programming (OOP) offers a structured and efficient approach to software development. The principles of encapsulation, inheritance, and polymorphism enhance code organization, readability, and reusability. By modelling real-world entities as objects, C++ OOP provides a powerful paradigm for building adaptable and maintainable software solutions.

_____