OBJECT ORIENTED PROGRAMMING

Chapter 1: Concept of Object Oriented Programming

1.1 History & Features of Object Oriented Programming

Need and Requirement of Object Oriented Programming

As software systems grew larger and more complex during the 1970s and 1980s, programmers using traditional procedural languages such as C, FORTRAN and Pascal began to face serious difficulties in managing, maintaining and reusing code. Large programs written using the procedural approach became difficult to debug, extend and modify because data and functions were kept separate from one another. Any change in a data structure often required changes in many functions that used that data, making software maintenance costly and error-prone.

This need for a more reliable, secure and reusable way of writing software led to the development of the Object Oriented Programming (OOP) approach. OOP was introduced to overcome the limitations of procedural programming by combining data and the functions that operate on that data into a single unit called an object. This approach improves data security, reduces code duplication through reusability, and makes large software projects easier to design, understand and maintain.

The main requirements that led to the development of OOP were:

- The need for better data security through data hiding.
- The need to reduce code redundancy through reusability (inheritance).
- The need to model real-world entities more naturally in software.
- The need for easier maintenance and extension of large programs.
- The need to manage growing software complexity in an organised manner.

Procedure Oriented Programming versus Object Oriented Programming

Procedure Oriented Programming (POP) is a programming paradigm in which a program is divided into a set of functions or procedures. The primary focus of POP is on functions, and data is treated as a secondary concern that moves freely between functions. Object Oriented Programming (OOP), on the other hand, focuses on objects, which bind data and the functions that act on that data together, giving data a central and protected role. The table below highlights the major differences between the two approaches.

Basis of Comparison	**Procedure Oriented Programming**	**Object Oriented Programming**
Basic focus	Focus is on functions/procedures	Focus is on data and objects
Program division	Divided into small functions	Divided into objects/classes
Data movement	Data moves freely between functions	Data is bound with related functions
Data security	Little or no data hiding	Data hiding through encapsulation
Overloading	Not supported	Functions and operators can be overloaded

Basis of Comparison	Procedure Oriented Programming	Object Oriented Programming
Reusability	Low; code duplication is common	High, through the concept of inheritance
Approach	Top-down approach	Bottom-up approach
Examples	C, FORTRAN, Pascal, COBOL	C++, Java, Python, C#

Basic Concepts of Object Oriented Programming

Object Oriented Programming is built around a small set of fundamental concepts. Understanding these concepts thoroughly is essential before studying any object oriented language, since all OOP languages are designed to support them in one form or another.

- Class: A class is a user-defined blueprint or template that defines the data members (attributes) and member functions (behaviour) common to all objects of that type. A class does not occupy memory by itself; it is only a description.
- Object: An object is a real instance of a class, created in memory at run time. Each object has its own copy of the data members defined in the class and can access the class's member functions.
- Data Abstraction: Abstraction means representing only the essential features of an entity while hiding the internal implementation details from the user. It allows a programmer to focus on what an object does rather than how it does it.
- Encapsulation: Encapsulation is the wrapping up of data and the functions that operate on that data into a single unit, i.e., a class. It ensures that data cannot be accessed directly from outside the class, providing data security.
- Inheritance: Inheritance is the mechanism by which a new class (derived/child class) acquires the properties and behaviour of an existing class (base/parent class). It promotes code reusability and establishes a hierarchical relationship between classes.
- Polymorphism: Polymorphism means "many forms". It is the ability of a function, object or operator to behave differently in different contexts, commonly achieved through function overloading, operator overloading and function overriding.
- Dynamic Binding: Also called late binding, this refers to linking a function call to its actual code at run time rather than at compile time, allowing more flexible program behaviour.
- Message Passing: Objects communicate with one another by sending and receiving information, known as message passing. A message essentially requests an object to invoke one of its member functions.

Object Oriented Languages

An object oriented language is a programming language that fully supports all the major features of Object Oriented Programming, namely encapsulation, inheritance, polymorphism and dynamic binding. Such languages allow programmers to create classes, derive new classes from existing ones, and override or overload functions.

- C++
- Java

- Python
- C#
- Smalltalk

These languages are considered "pure" or "true" object oriented languages because they support inheritance and polymorphism along with encapsulation, which are essential for building reusable and extensible software.

Object Based Languages

An object based language supports the concept of encapsulation and objects, but it does not support inheritance and/or polymorphism, which are essential features of a fully object oriented language. Such languages allow data and functions to be bound together as objects, but they do not allow one object type to derive properties from another.

- Ada
- Visual Basic (early versions)
- JavaScript (in its basic object model)

Because object based languages lack the inheritance mechanism, they are considered a step below fully object oriented languages, even though they share the idea of encapsulating data with related operations.

1.2 Beginning with C++

Concepts and Structure of a C++ Program

C++ is an object oriented extension of the C programming language, developed by Bjarne Stroustrup at Bell Laboratories in the early 1980s. It retains almost all the features of C while adding support for classes, objects, inheritance, polymorphism and other OOP concepts. A typical C++ program is organised into a well-defined structure, which is generally followed in the order given below.

1. Documentation Section: Contains comments describing the purpose of the program, the author's name, and the date of creation. Comments are written using // for a single line or /* ... */ for multiple lines.
2. Preprocessor Directive / Link Section: Contains header file inclusions such as #include <iostream.h> or #include <iostream>, which allow the program to use predefined library functions and objects.
3. Definition Section: Used to define symbolic constants, typically using #define or the const keyword.
4. Global Declaration Section: Declares global variables and function prototypes that can be used throughout the program, along with the declaration of any classes used.
5. main() Function Section: Every C++ program must contain exactly one main() function, which is the entry point of program execution. Execution always begins from the first statement inside main().
6. Sub-program / Function Definition Section: Contains the definitions of user-defined functions, which may appear before or after main(), or may be placed in separate files.

A simple structural skeleton of a C++ program is shown below:

```
#include <iostream.h>      // Preprocessor directive

void main()                // main function
```

```
{
    cout << "Hello, World!";   // statement
}
```

C++ also introduces several new concepts not present in C, including classes and objects, constructors and destructors, function overloading, operator overloading, inheritance, virtual functions and exception handling, all of which together support the full object oriented programming paradigm.

Insertion and Extraction Operators

C++ handles input and output through a stream-based mechanism rather than the format-string-based functions used in C (such as printf() and scanf()). A stream is simply a sequence of bytes flowing from a source to a destination. To perform stream input and output, C++ provides two special operators: the insertion operator and the extraction operator.

Insertion Operator (<<)

The insertion operator, written as <<, is used to send or "insert" data into an output stream, most commonly the standard output stream cout, so that it can be displayed on the screen. It is called the insertion operator because it inserts the value of a variable or a literal into the stream.

- Syntax: cout << expression;
- Example: cout << "Result = " << result;
- Multiple values can be inserted in a single statement by chaining several << operators together.

Extraction Operator (>>)

The extraction operator, written as >>, is used to "extract" or read data from an input stream, most commonly the standard input stream cin, and store it into a variable. It is called the extraction operator because it takes a value out of the stream and assigns it to a variable.

- Syntax: cin >> variable;
- Example: cin >> marks;
- Multiple variables can be read in one statement by chaining several >> operators, such as cin >> a >> b;

Both << and >> are actually overloaded operators; in C, they are ordinarily used as bitwise shift operators, but C++ overloads them for the ostream and istream classes so that they can perform stream insertion and extraction respectively.

Objects of the Input and Output Stream Classes

C++ provides a hierarchy of predefined classes to manage input and output operations, defined mainly in the header file iostream.h (or <iostream> in standard C++). The two most important of these are the istream class and the ostream class.

- istream Class: This class handles input operations from the standard input device (usually the keyboard). The predefined object cin is an object of the istream class, and it is used along with the extraction operator (>>) to read values into variables.

- ostream Class: This class handles output operations to the standard output device (usually the monitor). The predefined object cout is an object of the ostream class, and it is used along with the insertion operator (<<) to display values and messages on the screen.
- iostream Class: This class is derived from both istream and ostream, and it supports both input and output operations together.

In addition to cin and cout, C++ provides two more predefined stream objects: cerr, an object of the ostream class used for displaying error messages (unbuffered), and clog, another object of the ostream class used for logging messages (buffered). All four objects (cin, cout, cerr and clog) are automatically created when a C++ program that includes iostream.h begins execution.

Uses of the iostream.h Header File

The header file iostream.h (input-output stream header) is one of the most frequently used header files in C++ programming. It must be included at the beginning of a program using the #include directive whenever the program needs to perform any console input or output operation.

- It declares the standard stream objects cin, cout, cerr and clog, along with the classes istream, ostream and iostream that these objects belong to.
- It provides the definitions required to use the insertion operator (<<) and the extraction operator (>>) for performing output and input operations respectively.
- It enables formatted input and output through manipulators such as endl, setw(), setprecision() and others, which control spacing, width and precision of displayed values.
- It forms the foundation of console-based interaction in almost every C++ program, since nearly all beginner-level programs need to display output or accept input from the user.

Without including iostream.h (or its standard equivalent), a C++ program cannot use cin, cout or the stream operators, and the compiler will report an error stating that these identifiers are undeclared.

Multiple Choice Questions (MCQs)

Answer the following multiple choice questions. Each question carries one mark. The correct answers are provided in the answer key at the end.

1. Object Oriented Programming was developed mainly to overcome the limitations of:

(a) Machine language programming
(b) Procedure Oriented Programming
(c) Assembly language programming
(d) Markup languages

2. In Object Oriented Programming, data and functions are bound together in a unit called:

(a) Function
(b) Module
(c) Class/Object
(d) Procedure

3. Which of the following is NOT a basic concept of OOP?

(a) Encapsulation
(b) Inheritance

(c) Polymorphism

(d) Compilation

4. The mechanism by which a new class acquires the properties of an existing class is called:

(a) Encapsulation

(b) Inheritance

(c) Abstraction

(d) Overloading

5. "Many forms" of a single function or operator is referred to as:

(a) Polymorphism

(b) Encapsulation

(c) Binding

(d) Abstraction

6. Which approach does Procedure Oriented Programming primarily follow?

(a) Bottom-up approach

(b) Top-down approach

(c) Object-based approach

(d) Random approach

7. Data hiding in OOP is primarily achieved through:

(a) Inheritance

(b) Polymorphism

(c) Encapsulation

(d) Message passing

8. Which of the following is an example of an object oriented language?

(a) C

(b) Pascal

(c) C++

(d) FORTRAN

9. A language that supports objects and encapsulation but NOT inheritance is called:

(a) Object oriented language

(b) Object based language

(c) Procedural language

(d) Machine level language

10. Which of these is considered an object based language?

(a) Java

(b) C++

(c) Ada

(d) Python

11. C++ was developed by:

(a) Dennis Ritchie

(b) James Gosling

(c) Bjarne Stroustrup

(d) Guido van Rossum

12. In every C++ program, execution always begins from:

(a) The first function defined

(b) The main() function

(c) The class definition

(d) The header file

13. Which operator is known as the insertion operator in C++?

(a) >>

(b) <<

(c) ::

(d) ->

14. The predefined object used along with the extraction operator to read input is:

(a) cout

(b) cerr

(c) cin

(d) clog

15. Which header file must be included in a C++ program to use cin and cout?

(a) stdio.h

(b) conio.h

(c) iostream.h

(d) math.h

Answer Key

1. (b) 2. (c) 3. (d) 4. (b) 5. (a) 6. (b) 7. (c) 8. (c)
9. (b) 10. (c) 11. (c) 12. (b) 13. (b) 14. (c) 15. (c)

Broad / Descriptive Questions

Answer the following questions in detail. These questions are suitable for long-answer type assessment and viva-voce preparation.

1. Explain the need and requirement for Object Oriented Programming over traditional procedural programming, with suitable justification.

2. Differentiate between Procedure Oriented Programming and Object Oriented Programming with the help of a comparative table.

3. Describe the basic concepts of Object Oriented Programming with suitable examples for each concept.

4. What are object oriented languages? List any four such languages and explain why they are considered object oriented.

5. What is meant by an object based language? How does it differ from an object oriented language? Give examples.

6. Explain the general structure of a C++ program with the help of a suitable example, describing each section briefly.

7. What are insertion and extraction operators in C++? Explain their syntax and usage with examples.

8. Describe the istream and ostream classes in C++ and explain the role of the predefined objects cin and cout.

9. Explain the uses of the iostream.h header file in a C++ program.

10. With the help of a program, illustrate how input and output operations are performed in C++ using cin and cout.