Unit 1: Concept of Object Oriented Programming (OOP)

1.1 Programming Languages and Software Crisis

Programming Languages: Tools used by developers to write instructions that a computer can execute. Examples include C, C++, Java, Python, etc.

- **Low-Level Languages:** Machine language and assembly language.
- **High-Level Languages:** Procedural (C, Pascal), functional (LISP, Haskell), object-oriented (C++, Java), scripting (Python, JavaScript), and more.

Software Crisis: Refers to the difficulties in writing correct, understandable, and verifiable computer programs. Issues included increasing complexity, costs, and delays in software projects.

Challenges included:

- High costs and long development times.
- Poor quality and reliability.
- Difficulty in managing large projects.
- Maintenance issues.

1.2 Procedure Vs Object-Oriented Programming Language

C++ was devised by Bjarne Stroustrup in early 1980's (1983 A.D.) at Bell Laboratories. It is an extension of C by adding some enhancements specially addition of class into C language. So, it is also called as superset of C. Initial it was called as C with class. The main difference between C and C++ is that C++ is an object oriented while C is function or procedure oriented. Object oriented programming paradigm is focused on writing programs that are more readable and maintainable. It also helps the reuse of code by packaging a group of similar objects or using the concept of component programming model. It helps thinking in a logical way by using the concept of real world concept of objects, inheritance and polymorphism. It should be noted that there are also some drawbacks of such features. For example, using polymorphism in a program can slow down the performance of that program. On the other hand, functional and procedural programming focus primarily on the action and events, and the programming model focus on the logical assertions that trigger execution of program code.

- **Procedural Programming:** Focuses on functions or procedures to operate on data. Examples include C and Pascal.

- **Key Concepts:** Functions, top-down design, and flowcharts.
- Focuses on functions or procedures.

- Follows a top-down approach.
- Data is separate from functions.
- Examples: C, Pascal, COBOL, FORTRAN.

- **Object-Oriented Programming (OOP):** Focuses on objects that encapsulate data and behavior.

- **Key Concepts:** Classes, objects, inheritance, polymorphism, encapsulation, and abstraction.
- Focuses on objects which encapsulate data and functions.
- Follows a bottom-up approach.
- Promotes data hiding and abstraction.
- Examples: C++, Java, Python.

1.3 Features of Object-Oriented Programming

1. **Class and Object:** Basic units of OOP. A class defines a type, and an object is an instance of a class.
2. **Encapsulation:** Bundling data with the methods that operate on that data. Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines. The wrapping up of data and functions into a single unit is called Encapsulation.
3. **Abstraction:** Hiding complex implementation details and showing only the necessary features. Hiding internal details and showing functionality is known as abstraction.
4. **Inheritance:** Mechanism by which one class inherits properties and behaviors from another. It is the process by which objects of one class acquire the properties of another class. It supports the concept of hierarchical classification. In OOL, the concept of inheritance provides idea of reusability.
5. **Polymorphism:** Ability to take many forms. Methods to operate on objects of different classes through a common interface.

1.4 Popular Object-Oriented Programming Languages and Features

C++:

- Combines procedural and object-oriented features.
- Supports operator overloading, multiple inheritance.

Java:

- Purely object-oriented.
- Platform-independent due to Java Virtual Machine (JVM).
- Automatic garbage collection.

Python:

- Supports multiple programming paradigms.
- Dynamic typing and memory management.
- Easy to read and write.

1.5 Advantages and Disadvantages of OOP

- Benefits of using OOP:

OOP offers several benefits to both the program designer and the user. Object orientation programming promises greater programmer productivity, better quality of software and lesser maintenance cost. The principle advantages are:

• Through inheritance, we can eliminate redundant code & extend the use of existing classes.
• Reusability saves the development time and helps in higher productivity.
• It is possible to map objects in the problem domain to those in the program.
• It is easy to partition the work in a project based on objects.
• Object oriented systems can be easily upgraded from small to large system. • Software complexity can be easily managed

- **Advantages:**
  - Modularity: Easier to manage and understand.
  - Reusability: Code can be reused through inheritance. Reusability through inheritance.
  - Maintainability: Easier to update and modify. Improved software maintainability.
  - Scalability: Better handling of complex and large systems.
- **Disadvantages:**
  - Complexity: Can be more complex than procedural programming.
  - Performance: Sometimes slower due to abstraction. Slower execution compared to procedural programming in some cases.
  - Learning Curve: Steeper for beginners.

1.6 Introduction of C++ and Compilers

C++: An extension of C with object-oriented features. Used for system/software development.

- Developed by Bjarne Stroustrup in 1983.
- Extension of the C language with object-oriented features.
- Commonly used for system/software development, game programming.

Compilers: Translate C++ code into executable programs. Examples include GCC, Clang, and Microsoft Visual C++.

- Translate source code into machine code.
- Examples: GCC (GNU Compiler Collection), Clang, Microsoft Visual C++.

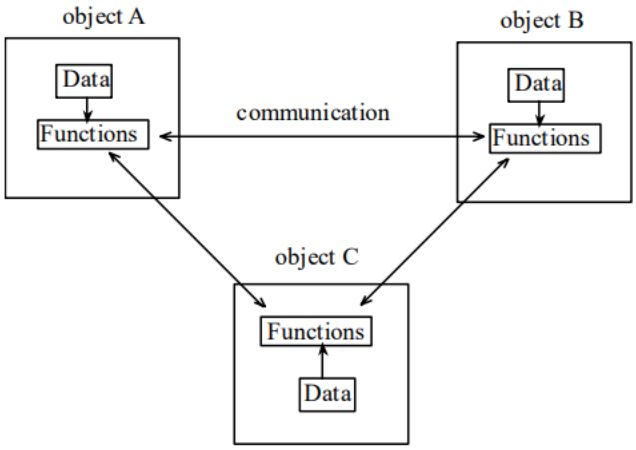

Fig: Organization of data & functions in OOP

The data of an object can be accessed only by the function associated with that object. However, functions of one object can access the function of other objects.

1.7 Programming Structure in C++

1. **Header Files:** Libraries to include.
2. **Main Function:** Entry point of the program.
3. **Classes and Objects:** Fundamental building blocks in OOP.
4. **Member Functions:** Methods within a class.
5. **Access Specifiers:** Public, private, and protected sections.

Structure:

- **Include Directives:** `#include <iostream>`
- **Namespace:** `using namespace std;`
- **Main Function:** `int main() { // code }`
- **Statements and Expressions**

Example:

```
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```

C++ Program

Before starting the abcd of C++ language, you need to learn how to write, compile and run the first C++ program.
To write the first C++ program, open the C++ console and write the following code:

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

int main() {
  int x;
  cout << "Type a number: "; // Type a number and press enter
  cin >> x; // Get user input from the keyboard
  cout << "Your number is: " << x;
  return 0;
}
```

Omitting Namespace

The `using namespace std` line can be omitted and replaced with the `std` keyword, followed by the `::` operator for some objects:

```
#include <iostream>
int main() {
  std::cout << "Hello World!";
  return 0;
}
```

```
Addition of two numbers in C++ program
#include<iostream>//library
//using namespace std;//namspace
int main()//main function
{
    int a,b,sum;//declaration
    std::cout<<"enter two numbers"<<std::endl;
    std::cin>>a>>b;//input from user
```

```
    sum=a+b;//process
    std::cout<<"sum is "<<sum;//print on the console

    return 0; //does not return null value
}
```

1.8 Comparison of C and C++

C:

- Procedural language.
- Functions and variables are the main building blocks.No built-in support for OOP: Does not support classes and objects
- Supports functions and pointers.

C++:

- Supports both procedural and object-oriented programming.
- Has features of classes, objects, inheritance, and polymorphism.
- Supports features like function overloading, function overloading, templates, and exceptions.

1.9 Additional Data Types, Token in C++

- **Data Types:**
    - Fundamental: int, char, float, double, void.
    - Derived: arrays, pointers, references.
    - User-defined: classes, structures, unions, enumerations.

- **Tokens:**
    - Keywords: Reserved words with special meaning(e.g., `int`, `class`, `return`)..
    - Identifiers: Names given to variables, functions, etc.
    - Constants: Fixed values.
    - Operators: Symbols that perform operations(e.g., `+`, `*`, `==`)..
    - Separators/ Punctuators: Symbols that organize code like semicolons, braces, commas(e.g., `{}`, `;`).

1.10 Insertion and Extraction Operators

Insertion Operator (<<): Used to output data to the console.
cout << "Hello, World!";

- Used to output data to streams (e.g., `std::cout`).
- Example: `std::cout << "Hello, World!" << std::endl;`

Extraction Operator (>>): Used to input data from the console.

int x;

cin >> x;

- Used to input data from streams (e.g., `std::cin`).
- Example: `std::cin >> userInput;`