Lecture 1

19CSE201 Advanced Programming

Course Outcome

	Course Outcome	Bloom’s Taxonomy Level
CO 1	Understand the static object oriented programming concepts and thereby to understand a given program.	L3
CO 2	Understand the dynamic object oriented programming concepts and thereby to understand a given program.	L3
CO 3	Implement ADT in static and dynamic object oriented paradigm.	L3
CO 4	Analyze the similarities, differences and code efficiency among object oriented programming languages.	L4
CO 5	Develop computer programs that implement suitable algorithms for given problem scenario and applications.	L4

Syllabus

Unit 1

- Overview of Object Oriented Paradigm, Programming in C++: Objects as a group of variables, Classes as a named group of methods and data, Morphing from structures to classes, Input and Output, Access Specifiers, Member functions: Accessor, Mutator and Auxiliary, Constructors and Destructors, New and Delete Operators, Overloading, Inheritance: Handling Access and Specialization through Overriding, Polymorphism: Virtual Functions, Abstract Class and Virtual Function Tables.

Unit 2

- Revisiting Pointers: Pointers to Pointers, Pointers and String Array, Void Pointers and Function Pointers, Standard Template Library, Implementation of Stack, Queue, Hash Table and Linked Lists with STL. Basic Python: Multi paradigm language, Data Types and Variables, Indentation, Input and Output statements, Lists and Strings, Deep and Shallow Copy, Tuples and Dictionaries, Set and Frozen Sets, Control Statements and Loops, Iterators and Iterable, Functions, Recursion and Parameter Passing, Namespaces and Variable Scope, Exception Handling.

Unit 3

- Object Oriented Concepts in Python: Class, Instance Attributes, Getters, Setters, Inheritance, Multiple Inheritance, Magic Methods and Operator Overloading, Class Creation, Slots, Meta Classes and Abstract Classes, Implementation of Stack, Queue, Hash Table and Linked Lists.

Books

- ***Text Books***
- *Stroustrup B. Programming: principles and practice using C++. Second edition, Addison Wesley; 2014.*
- *Charles R. Severance. Python for Everybody: Exploring Data Using Python 3, Charles Severance; 2016.*
- ***Reference(s)***
- *Guttag J. Introduction to Computation and Programming Using Python: With Application to Understanding Data. Second Edition. MIT Press; 2016.*
- *Gaddis T. Starting Out with Python. Third Edition, Pearson; 2014.*
- *Lambert KA. Fundamentals of Python: first programs. Second Edition, Cengage Learning; 2018.*
- *Downey AB. Think Python: How to Think Like a Computer Scientist. O'Reilly Media; 2012.*

Course Plan

Lecture No(s)	Topics	Key-words	Objectives
1	Introduction to different paradigms of Programming languages	Paradigms of Programming Languages.	To introduce students to the different paradigms of Programming Languages.
2	Why object-oriented programming?	OOP, Abstraction, Objects, Encapsulation, Class	To introduce students to different terminologies in OOPS
3	Key concept in OOP and the object oriented technology Mindset	Inheritance, Generalization, Message passing between objects,Polymorphism,interface,Modularity, Reusability	To introduce students to key concepts in OOPs and compare it with other programming languages
4	Anatomy of C++	Anatomy of C++, Input and Output statements, Hello World	Transition from C to C++
5	C++ Basics	Assignment, initialization, customizing input and output Selection operations, simple programs, Array and loop	Understanding C++ syntax
6	Introduction to object-oriented concepts Class definition, Object Instantiation	Types, instances, attributes, behaviors, Class definition- declaring members, creating instances of a class	Beginning with Object oriented programming
7	Accessing member of the class, Member function definition, Access modifiers	Member selection operator Member function definition- inline and implicit inline functions Access modifiers: private, protected, public	Understanding how to access data members of a class Defining member function of a class To understand data hiding and accessibility of class members

Lecture No(s)	Topics	Key-words	Objectives
8	Standard input output member function in a class	Accessor, Mutator and Auxiliary function	To understand how to define input and output functions for a class
9	Morphing from Structure to class, Types of constructors and destructors	Morphing from struct to class Parameter constructor, default constructor, copy constructor, destructor	To understand the difference between procedural and object-oriented programming To understand various constructors and its use.
10	Relationships among Classes	Introduction to Inheritance, type of Inheritance	To understand inheritance relationships between classes, including public, protected, and private inheritance.
11	Types of Inheritance	Single, Multiple, Multilevel Inheritance	To understand the different Type of Inheritance
12	Behaviors of modifier in inheritance	Data member, Member function, class	To understand the dependency relationship between classes to show that one class can use the objects of another class.
13	Dynamic memory allocation	New, Delete operator	To understand how to handle dynamic memory operations in C++
14	Polymorphism	Function overloading, operator overloading	To understand the how functions and operators can be overloaded

Lecture No(s)	Topics	Key-words	Objectives
15	Function overriding	Inheritance	To understand function overriding
16	Virtual Function	Virtual Table, Virtual Function	To understand virtual tables and how they help the system decide which virtual function to use during run time.
17	Abstract Class	Abstract Class , pure Virtual Function	To Understand abstract classes and their use
18	Exception Handling	Try-catch block, throw statement	To understand exception handling in a function using three different patterns that use try-catch block and throw statements.
	Periodical Test 1		

Lecture No(s)	Topics	Key-words	Objectives
19	Pointers	Introduction to pointer, Access variables.	To understand how to retrieve and change data in a data variable through the pointer variable.
20	Pointer Type and pointer variable	Pointer Type, Pointer Variables	To understand the pointer type and allows us to use pointer variables.
21	Basic Python	Multiparadigm language, Data Types & Variables, Input and Output statements,	To understand basic syntax of Python
22	Basic Python	Control Statements and Loops	To understand various control statements
23	Basic Python	Lists and Strings	To understand Lists and string concepts
24	Specialized python operators	Deep and Shallow Copy	To understand various collections in python
25	Specialized python operators	Tuples, Dictionaries	To understand Tuples and Dictionaries concepts
26	Specialized python operators	Set and Frozen Sets, Iterators and Iterable	To understand unordered and unindexed python objects

Lecture No(s)	Topics	Key-words	Objectives
27	Functions	Functions, Passing Parameters, Namespaces & Scope of variables	To understand functions, namespaces
28	Exception Handling	Exception Handling	To understand exception handling
29	Object Oriented Concepts in Python	Class, Objects, Instance Attributes	To understand the basic class and objects
30	Object Oriented Concepts in Python	Instance Method	To understand instance methods
31	Data Encapsulation	Getters, Setters	To understand data encapsulation
32	Inheritance	Inheritance, type of Inheritance,	To understand inheritance relationships between classes, including public, protected, and private inheritance.
33	Type of Inheritance	Single, Multiple, Multilevel Inheritance	To understand the different Type of Inheritance
34	Magic Methods and Operator Overloading	Magic Methods, Operator Overloading	To understand Magic Methods and Operator Overloading
35	Slots, Meta Classes	Slots, Meta Classes	To understand Slots and Meta Classes
36	Abstract Classes	Abstract Classes	To understand Abstract Classes
	Periodical Test 2		

Lecture No(s)	Topics	Key-words	Objectives
37	Templates	Function Templates, Class Templates	To understand templates for generic programming
38	Abstract Data Type (ADT)-Stack	Stack ADT	To understand ADT for Stack
39	Abstract Data Type (ADT)-Queue	Queue ADT	To understand ADT for Queue
40	Abstract Data Type (ADT)-Hash Table	Hash Table ADT	To understand ADT for Hash Table
41	Abstract Data Type (ADT)-Linked List	Linked List ADT	To understand ADT for Linked List

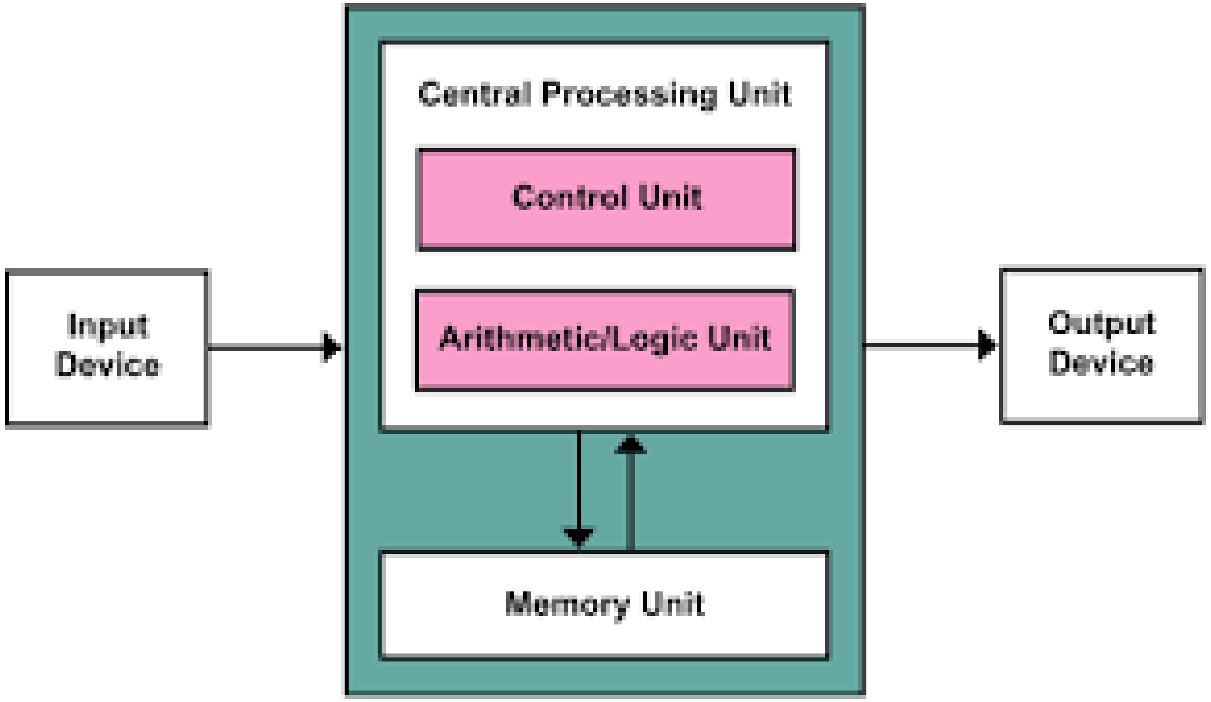
Lecture No(s)	Topics	Key-words	Objectives
42	Abstract Data Type (ADT)-Stack	Stack ADT	To understand ADT for Stack
43	Abstract Data Type (ADT)-Queue	Queue ADT	To understand ADT for Queue
44	Abstract Data Type (ADT)-Hash Table	Hash Table ADT	To understand ADT for Hash Table
45	Abstract Data Type (ADT)-Linked List	Linked List ADT	To understand ADT for Linked List
	End Semester		

Evaluation

Plan of Evaluation

Component	Events	Weightage	Max Marks
Internals:70%	Quiz-1 and Quiz-2	5+5	10
	Periodical Test-1	10	50
	Periodical Test-2	10	50
	Lab Evaluations (one test based on C++, Project based on Python)	15+15	30
	Lab Programs	10	30
External :30%	End Semester Exam	30	100

Introduction to computers



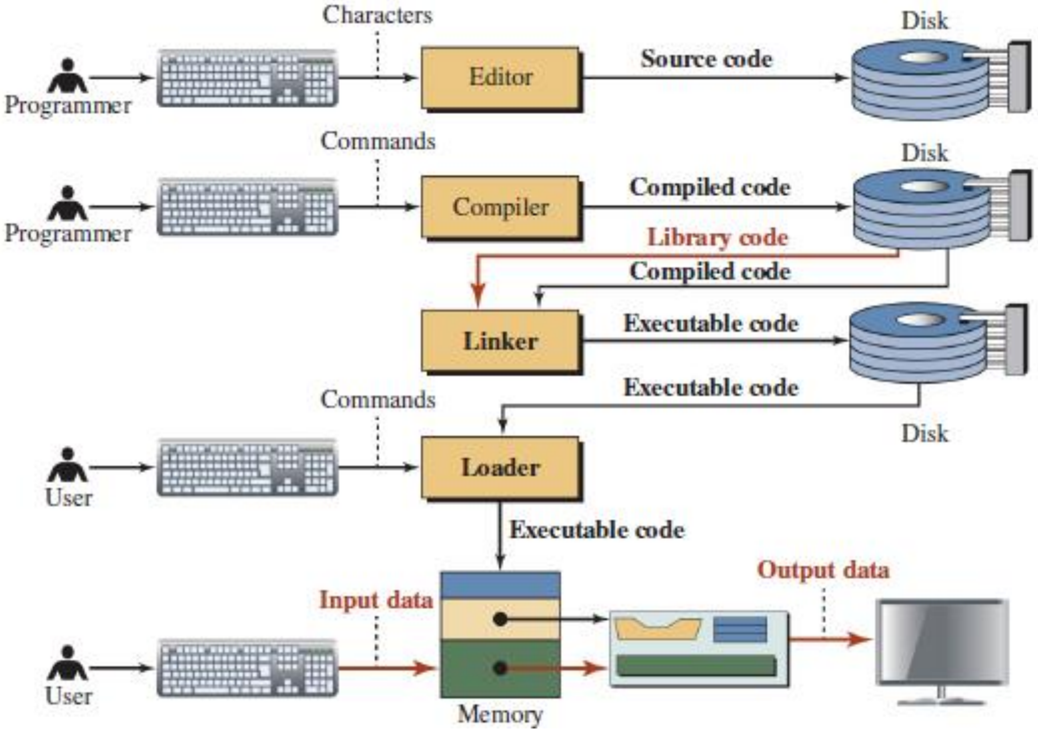
- Software
  - System Software- eg: compiler, interpreter, OS
  - Application software- eg: aums, irctc, library management system
- Hardware

Computer Languages

- **Machine Languages/Low level languages**
  - **0s and 1s**
  - **Only machine can understand this language**
- **Symbolic Languages/Assembly Languages**
  - **Converter Assembler is used to convert to low level languages**
- **High Level Languages**
  - **Compiler/Interpreter used to convert to low level language**
  - **Eg: C, C++, Java**

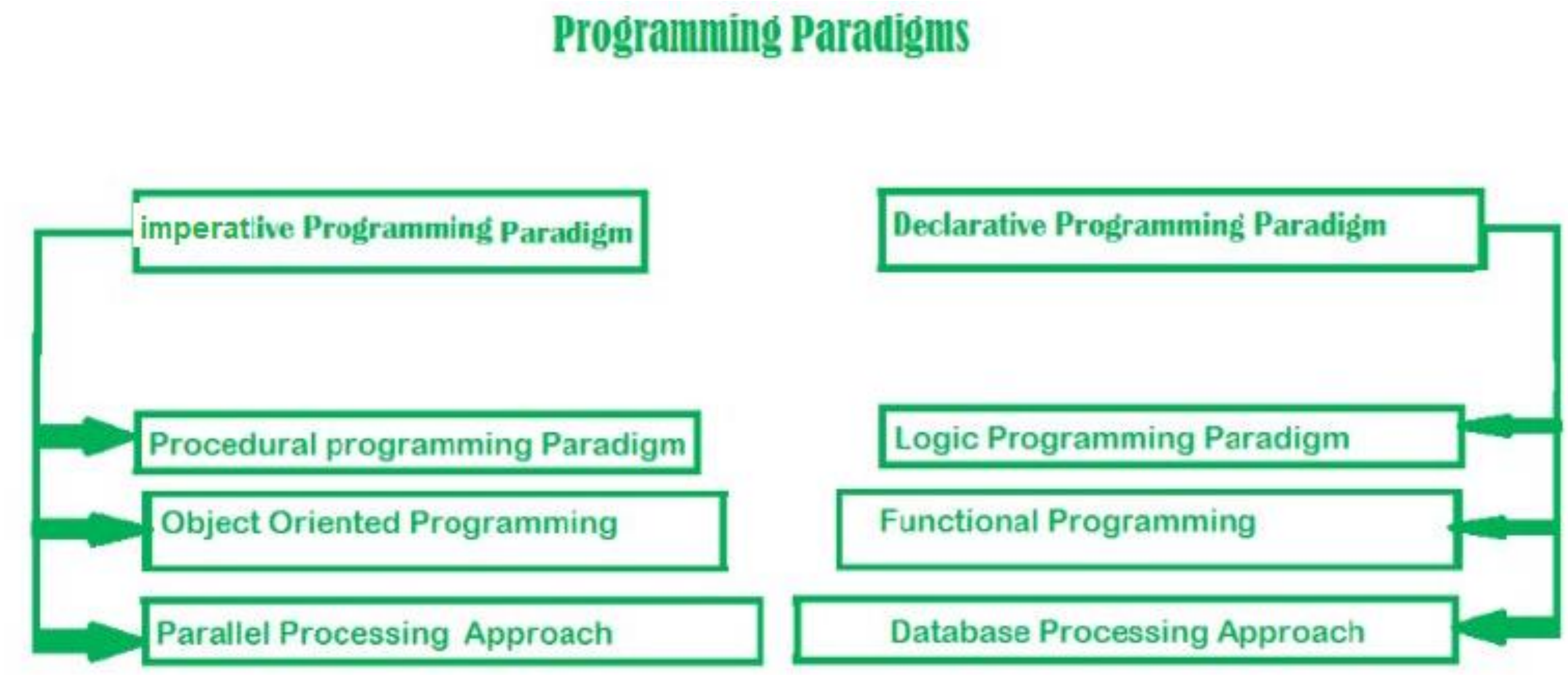
How to Develop a program?

- Write and edit the program.
- Compile the program.
- Link the program with the required library modules (normally done automatically).
- Execute the program. From our point of view, executing the program is one step.
- From the computer point of view, however, it is two sub steps: load the program and run the program.



Different paradigms of Programming languages

A **programming paradigm** is a style, or “way,” of **programming**


Programming Paradigms
imperative Programming Paradigm
Procedural programming Paradigm
Object Oriented Programming
Parallel Processing Approach
Declarative Programming Paradigm
Logic Programming Paradigm
Functional Programming
Database Processing Approach

- Unlike declarative **programming**, which describes "what" a **program** should accomplish, **imperative programming** explicitly tells the computer "how" to accomplish it

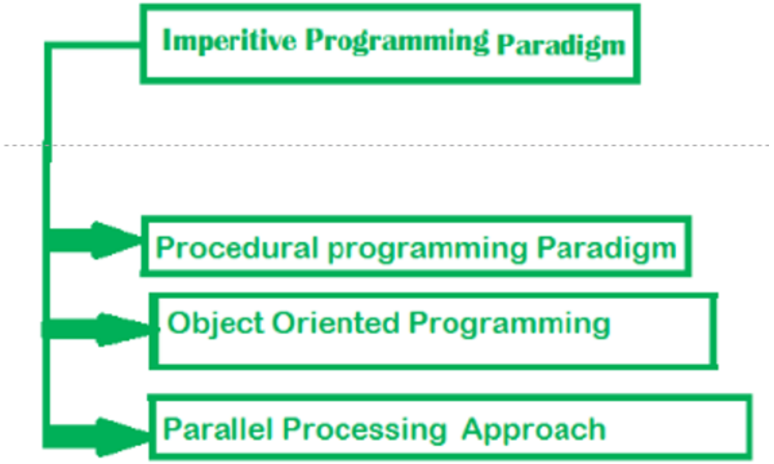
Imperative programming paradigm

- Oldest programming paradigm.
- **Imperative programming** is a **paradigm** of computer **programming** in which the **program** describes a sequence of steps that change the state of the computer.
- **Advantage:**
  - Very simple to implement
  - It contains loops, variables etc.

Procedural programming paradigm

- It has the ability to reuse the code and it was boon at that time when it was in use because of its reusability.

Pascal


Imperitive Programming Paradigm

Procedural programming Paradigm

Object Oriented Programming

Parallel Processing Approach

Java

Object oriented programming

- The program is written as a collection of classes and object which are meant for communication.
- **Advantages:**
- Data security
- Inheritance
- Code reusability
- Flexible and abstraction is also present

Java
python
Ruby
SIMULA

Parallel processing approach

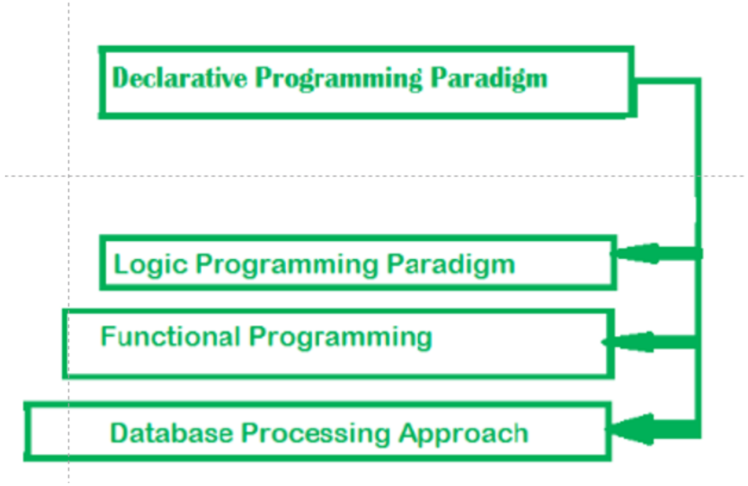
- Parallel processing is the processing of program instructions by dividing them among multiple processors.
- A parallel processing system posses many number of processors with the objective of running a program in less time by dividing them.

Examples

NESL (one of the oldest one) and C/C++ has library functions

Declarative programming paradigm

- It is divided as Logic, Functional, Database.
- It is a style of building programs that expresses logic of computation without talking about its control flow.
- Imperative (how to do) and declarative (what to do)


Declarative Programming Paradigm
Logic Programming Paradigm
Functional Programming
Database Processing Approach

Logic programming paradigms

- In logical programming the main emphasize is on knowledge base and the problem.
- It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc.
- Any program written in a logic programming language is a set of sentences in logical form, expressing facts and rules about some problem domain.

Functional programming paradigms

- The functional programming paradigms has its roots in mathematics and it is language independent
- The key principal of this paradigms is the execution of series of mathematical functions.

Database/Data driven programming approach

- This programming methodology is based on data and its movement.
- Program statements are defined by data rather than hard-coding a series of steps.

Other categories

- Structured programming Language
- Unstructured programming Language

Structured programming Language

- Enforces logical structure on the program to make it more efficient and easier to understand and modify
- Supports several loop constructs
- Use of goto is discouraged
- Eg: C, C++, Java, Ada

Unstructured programming Language

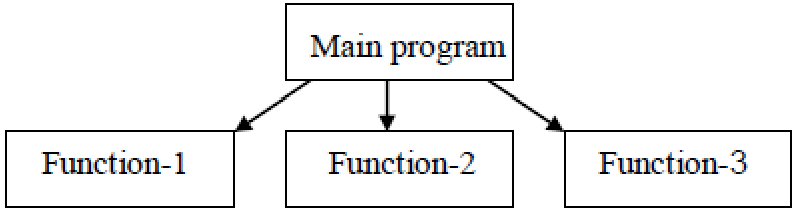
- Wont follow any particular structure
- Old programming languages
- We cant use it for complex programming
- Eg: BASIC, COBOL, FORTRAN

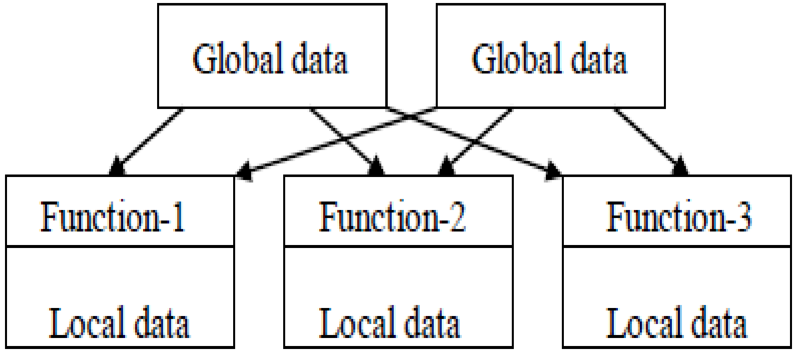
Focus is on

- Procedural programming Language
- Object oriented programming Language

Procedure Oriented Programming

- Focus is on functions
- Emphasis on doing things(algorithms)
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data
- Data move openly from one function to another
- Employ top- down approach in program design
- Eg: C, Cobol, Fortran



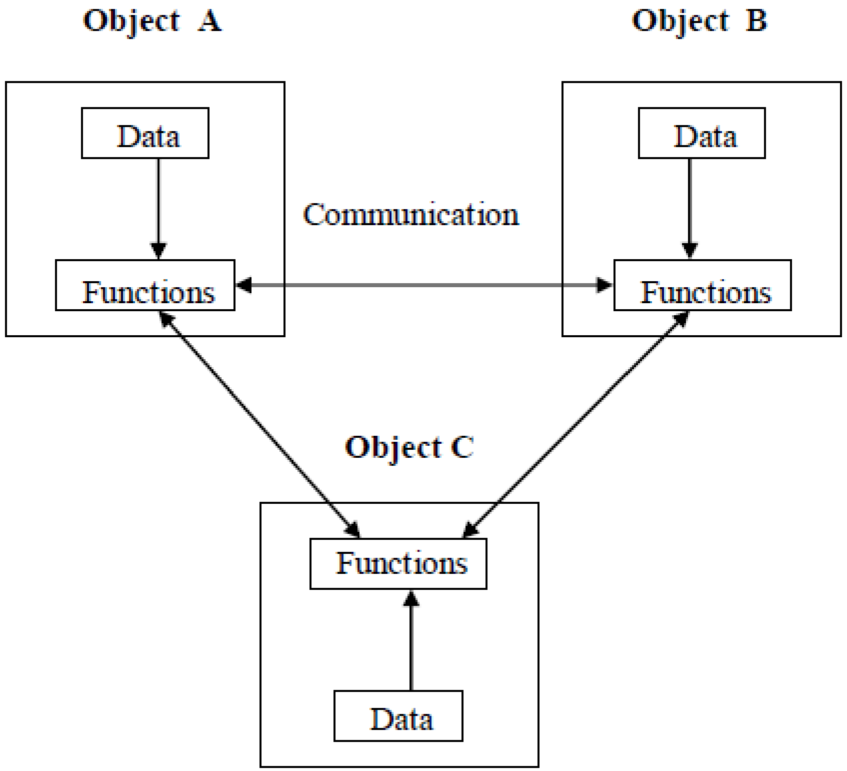


Drawback of Procedure Oriented Programming

- Cant model real world problems very well
- Important data is placed as global. So all functions can access and change that data.
- So its difficult to find which data is accessed by which function.
- Data move openly from one function to another

Object Oriented Programming

- It wont allow data to move freely around the system
- It ties data and functions together and protects it from accidental modifications from outside.
- Program is divided into entities and objects and builds data and functions around these objects.
- Bottom up approach
- More emphasis is on data.
- Data is hidden and can’t be accessed by external function
- Objects communicate with each other through functions.
- Can model real world problems


Object A
Object B
Data
Data
Communication
Functions
Functions
Object C
Functions
Data