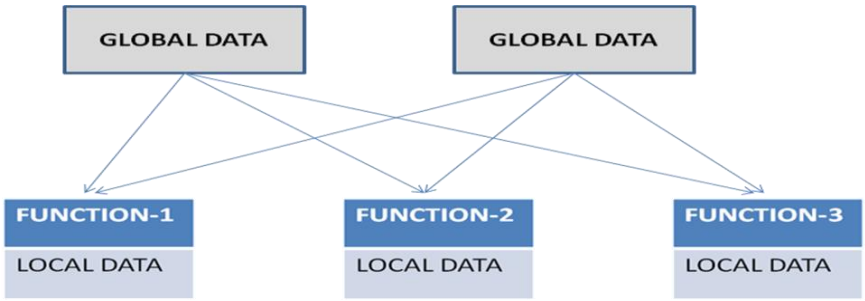
PROCEDURE ORIENTED PROGRAMMING

Conventional programming, using high level languages such as COBOL, FORTRAN and C, is commonly known as *procedure-oriented programming* (POP). In the procedure-oriented approach, the problem is viewed as a sequence of things to be done such as reading, calculating and printing. A number of functions are written to accomplish these tasks. The primary focus is on functions. The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.

Procedure-oriented programming basically consists of writing a list of instructions (or actions) for the computer to follow, and organizing these instructions into groups known as *functions.* We normally use a flowchart to organize these actions and represent the flow of control from one action to another. While we concentrate on the development of functions, very little attention is given to the data that are being used by various functions. What happens to the data? How are they affected by the functions that are work on them?

In a multi-function program, many important data items are placed as *global* so that they may be accessed by all the functions. Each function may have its own *local data.*



Global data are more vulnerable to an inadvertent change by a function. In a large program it is very difficult to identify what data is used by which function. In case we need to revise an external data structure, we also need to revise all functions that access the data. This provides and opportunity for bugs to creep in.

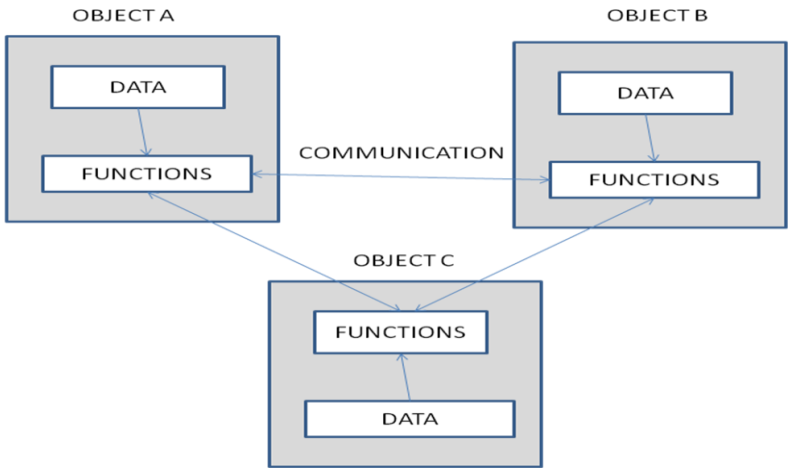
Another serious drawback with the procedural approach is that it does not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the elements of the problem.

Some characteristics exhibited by procedure-oriented programming are:

- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach in program design.

OBJECT ORIENTED PROGRAMMING PARADIGM

The major motivating factor in the invention of object oriented approach is to remove some of the flaws encountered in the procedural approach. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it, and protects it from accidental modification from outside functions. OOP allows decomposition of a problem into a number of entities called objects and then builds data and functions around these objects. The organization of data and functions in object-oriented programs is shown in the figure below. The data of an object can be accessed only by functions associated with that object. However, functions of one object can access the function of other objects.



Some of the striking features of object oriented programming are:

- Emphasis is on data rather than procedure.
- Programs are divided into what are known as objects.
- Data structures are designed such that they characterize the objects.
- Functions that operate on the data of an object are tied together in the data structure.
- Data is hidden and cannot be accessed by external functions
- Objects may communicate with each other through functions.
- New data and functions can be easily added whenever necessary.
- Follows bottom-up approach in program design.

BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING

It is necessary to understand some of the concepts used extensively in object oriented programming. These includes:

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message passing

OBJECTS:

Objects are the **basic run-time entities in an object-oriented system**. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user-defined data such as vectors, time and lists. Programming problem is analyzed in terms of objects and the nature of communication between them. Program objects should be chosen such that they match closely with the real-world objects. Objects take up space in the memory and have an associated address like a record in Pascal, or a structure in C.

When a program is executed, the objects interact by sending messages to one another. For example, if "customer" and "account" are two objects in a program, then the customer object may send a message to the account object requesting for bank balance. **Each object contains data and code to manipulate the data.** Objects can interact without having to know the details of each other's data or code. It is sufficient to know the type of message accepted, and the type of response returned by the objects.

CLASSES:

Object contains data and code to manipulate that data. The entire set of data and code of an object can be made a user-defined data type with the help of a *class.* In fact, objects are variables of the type *class.* Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated with the data of type class with which they are created. A class is thus a collection of objects of similar type. For example, mango, apple and orange are members of the class fruit. Classes are user-defined data types and behave like the built-in types of a programming language. The syntax used to create an object is no different than the syntax used to create an integer object in C. If fruit has been defined as a class, then the statement

```
fruit mango;
```

will create an object **mango** belonging to the class **fruit.**

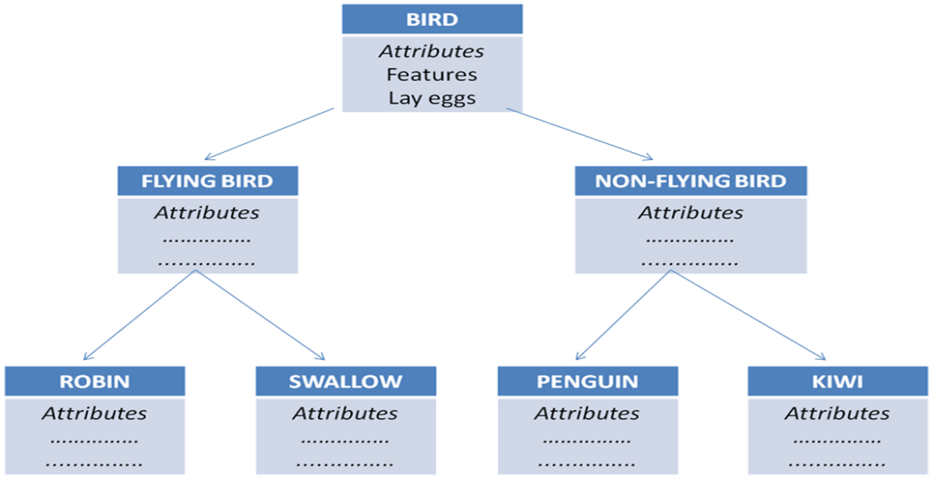
DATA ABSTRACTION AND ENCAPSULATION:

The wrapping up of the data and functions into a single unit (called class) is known as *encapsulation*. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the

interface between the object's data and the program. This insulation of the data from the data from direct access by the program is called *data hiding or information hiding.*
Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract *attributes* such as size, weight and cost, and *functions* to operate on these attributes. They encapsulate all the essential properties of the objects that are to be created. The attributes are sometimes called *data members* because they hold information. The functions that operate on these data re sometimes called *methods or member functions.*

Since the classes use the concept of data abstraction, they are known as ***Abstract Data Types (ADT).***

INHERITANCE:
Inheritance is the process by which objects of one class acquire the properties of objects of another class. It supports the concept of *hierarchical classification*. For example, the bird 'robin' is a part of the class 'flying bird' which is again a part of the class 'bird'. The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived.



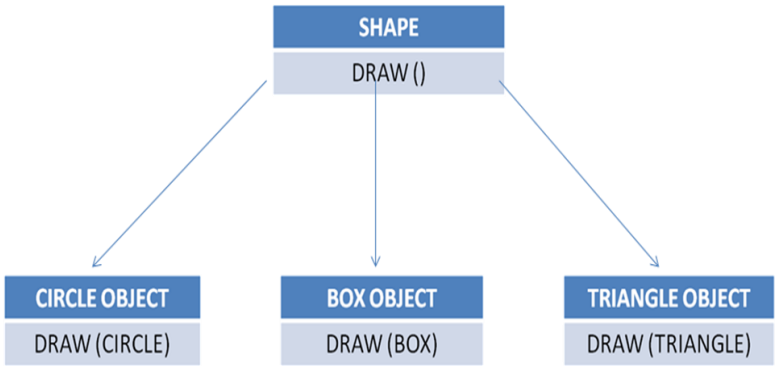
In OOP, the concept of inheritance provides the idea of *reusability.* This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes. The real appeal and power of the inheritance mechanism is that it allows the programmer to reuse a class that is almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduce any undesirable side-effects into the rest of the classes.

Note that each sub-class defines only those features that are unique to it. Without the use of classification, each class would have to explicitly include all of its features.

POLYMORPHISM:

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends upon the types of data used in the operation. For example, consider the operation of addition. For two numbers, the operation will generate a sum. If the operands are strings, then the operation would produce a third string by concatenation. The process of making an operator to exhibit different behaviors in different instances is known as *operator overloading.*

Figure illustrates that a single function name can be used to handle different number and different types of arguments. This is something similar to a particular word having several different meanings on the context. Using a single function name to perform different types of tasks is known as *function overloading.*



Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific actions associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

DYNAMIC BINDING:

Binding refers to the linking of a procedure call to the code to be executed in response to the call. *Dynamic binding (ALSO KNOWN AS LATE BINDING)* means that the code associated with a given procedure call is not known until the time of the call at run-time. It is associated with the polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that reference.

Consider the procedure “draw” in above diagram. By inheritance, every object will have this procedure. Its algorithm is, however, unique to each object and so the draw procedure will be redefined in each class that defines the object. At run-time, the code matching the object under current reference will be called.

MESSAGE PASSING

An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, therefore, involves the following basic steps:

1. Creating classes that define objects and their behavior,
2. Creating objects from class definitions, and
3. Establishing communication among objects.

Objects communicate with one another by sending and receiving information much the same way as people pass messages to one another. The concepts of message passing makes it easier to talk about building systems that directly model or simulate their real-world counterparts.

A message for an object is a request fro execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired result. *Message passing* involves specifying the name of the object, the name of the function (message) and the information to be sent.

Objects have a life cycle. They can be created and destroyed. Communication with an object is feasible as long as it is alive.

BENEFITS OF OOP:

OOP offers several benefits to both the program designer and the user. Object orientation contributes to the solution of many problems associated with the development and quality of software products. The new technology promises greater programmer productivity, better quality of software and lesser maintenance cost. The principle advantages are:

1. Through inheritance, we can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
3. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
4. It is possible to have multiple instances of an object to co-exist without any interference.
5. It is possible to map objects in the problem domain to those in the program.
6. It is easy to partition the work in a project based on objects.
7. Message passing techniques for communication between objects makes the interface descriptions with the external systems much simpler.
8. Software complexity can be easily managed.