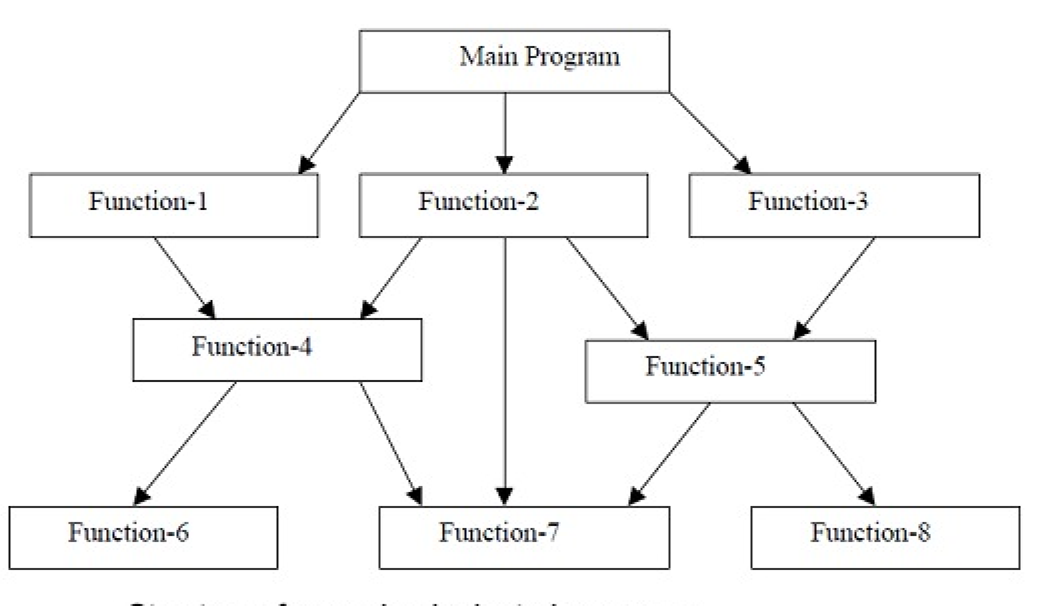
Procedure-Oriented Programming Vs. OOP

In the procedure oriented approach, the problem is viewed as the sequence of things to be done such as reading, calculating and printing such as C, Pascal, fortran etc. The primary focus is on functions. A typical structure for procedural programming is shown in fig below. The technique of hierarchical decomposition has been used to specify the tasks to be completed for solving a problem.



Structure of procedural oriented programs

Procedure oriented programming basically consists of writing a list of instructions for the computer to follow and organizing these instructions into groups known as functions. We normally use flowcharts to organize these actions and represent the flow of control from one action to another.

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 an opportunity for bugs to creep in.

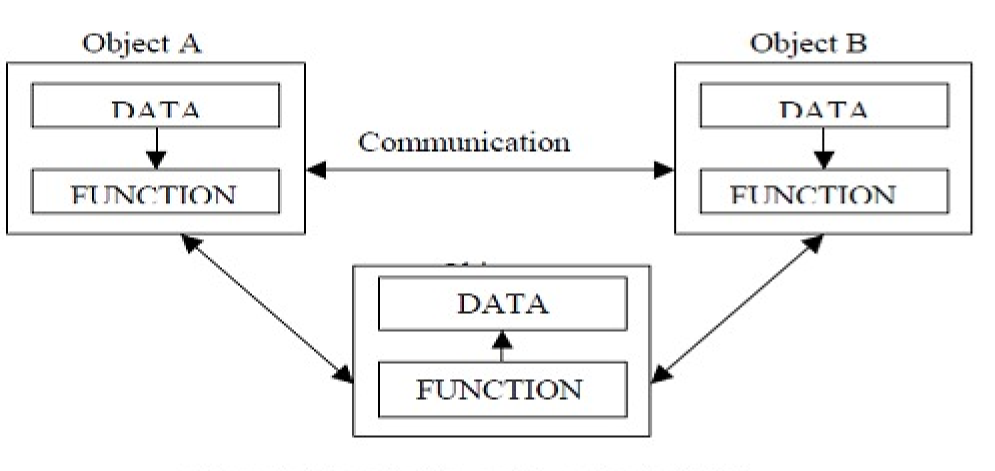
Another serious drawback with the procedural approach is that we do not model real world problems very well. This is because functions are action-oriented and do not really corresponding to the element of the problem.

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 (OOP)
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 function that operate on it, and protects it from accidental modification from outside function. OOP allows decomposition of a problem into a number of entities called objects and then builds data and function around these objects.

The organization of data and function in object-oriented programs is shown in fig below. The data of an object can be accessed only by the function associated with that object. However, function of one object can access the function of
other objects.



Organization of data and function in OOP

Some Characteristics of Object Oriented Programming

- 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 ties together in the data structure.
- Data is hidden and cannot be accessed by external function.
- Objects may communicate with each other through function.
- New data and functions can be easily added whenever necessary.
- Follows bottom up approach in program design.

To support the principles of object-oriented programming, all OOP languages, including C++, have the following characteristics

- Objects
- Classes
- Data abstraction and encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing
  Let's examine each.

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 term 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 to object in a program, then the customer object may send a message to the count object requesting for the bank balance. Each object contain data, and code to manipulate data. Objects can interact without having to know 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
As mentioned above the objects contain 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 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 similar types. For examples, guava, mango and orange are members of the class fruit. Classes are user-defined that types and behave like the builtin types of a programming language. The syntax used to create an object is not different then the syntax used to create an integer object in C. If fruit has been defines as a class, then the statement Fruit Mango; will create an object mango belonging to the class fruit.

Data Abstraction and Encapsulation
Encapsulation is a programming mechanism that binds together code and the data
it manipulates, and that keeps both safe from outside interference and misuse. In an object-oriented language, code and data can be bound together in such a way that a self-contained black box is created. Within the box are all necessary data and code. When code and data are linked together in this fashion, an object is created.

In other words, an object is the device that supports encapsulation. Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, wait, and cost, and function operate on these attributes. They encapsulate all the essential properties of the object that are to be created. The attributes are some time called data members because they hold information. The functions that operate on these data are sometimes called methods or member function.

Polymorphism
Polymorphism (from Greek meaning "many forms") is the quality that allows one interface to access a general class of actions. A simple example of polymorphism is found in the steering wheel of an automobile. The steering wheel (the interface) is the same no matter what type of actual steering mechanism is used. That is, the steering
wheel works the same whether your car has manual steering, power steering, or rackand-pinion steering. Thus, turning the steering wheel left causes the car to go left no matter what type of steering is used. The benefit of the uniform interface is, of course, that once you know how to operate the steering wheel, you can drive any type of car.
The same principle can also apply to programming.

For example, consider a stack (which is a first-in, last- out list). You might have a program that requires three different types of stacks. One stack is used for integer values, one for floating-point values, and one for characters. In this case, the algorithm that implements each stack is the same, even though the data being stored differs.
In a non–object-oriented language, you would be required to create three different sets of stack routines, with each set using different names. However, because of polymorphism, in C++ you can create one general set of stack routines that works for all three situations. This way, once you know how to use one stack, you can use them all.

Inheritance
Inheritance is the process by which one object can acquire the properties of another object. This is important because it supports the concept of hierarchical classification. If you think about it, most knowledge is made manageable by hierarchical (that is, topdown) classifications. For example, a Red Delicious apple is part of the classification apple, which in turn is part of the fruit class, which is under the larger class food. That is,
the food class possesses certain qualities (edible, nutritious, and so on) which also, logically, apply to its subclass, fruit. In addition to these qualities, the fruit class has specific characteristics (juicy, sweet, and so on) that distinguish it from other food. The apple class defines those qualities specific to an apple (grows on trees, not tropical, and
so on). A Red Delicious apple would, in turn, inherit all the qualities of all preceding classes and would define only those qualities that make it unique. Without the use of hierarchies, each object would have to explicitly define all of its characteristics.

Using inheritance, an object need only define those qualities that make it unique within its class. It can inherit its general attributes from its parent. Thus, it is the inheritance mechanism that makes it possible for one object to be a specific instance of a more general case.

Dynamic Binding
Binding refers to the linking of a procedure call to the code to be executed in response to the call. Dynamic 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 polymorphism and inheritance. A function call associated with a polymorphic reference depends on the dynamic type of that
reference. Consider the procedure "draw" 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. Objects communicate with one another by sending and receiving information. A Message for an object is a request for execution of a procedure, and therefore will invoke a function (procedure) in the receiving object that generates the desired results. Message passing involves specifying the name of object, the name of the function (message) and the information to be sent.

Application of OOP
OOP has become one of the programming buzzwords today. There appears to be a great deal of excitement and interest among software engineers in using OOP. Applications of OOP are beginning to gain importance in many areas.

The most popular application of object-oriented programming, up to now, has been in the area of user interface design such as window. Hundreds of windowing systems have been developed, using the OOP techniques.
Real-business system are often much more complex and contain many more objects with complicated attributes and method. OOP is useful in these types of application because it can simplify a complex problem.

The promising areas of application of OOP include:

- Real-time system
- Simulation and modeling
- Object-oriented data bases
- Hypertext, Hypermedia, and expertext
- AI and expert systems
- Neural networks and parallel programming
- Decision support and office automation systems
- CIM/CAM/CAD systems