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.

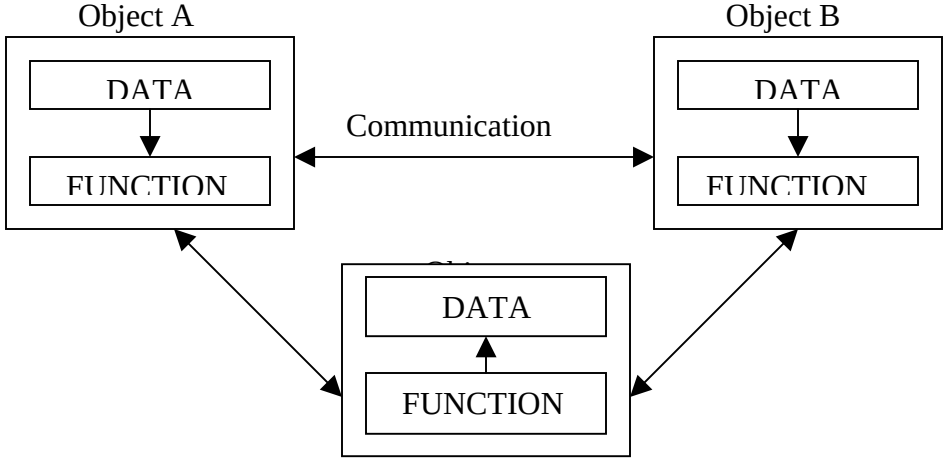
Some Characteristics exhibited by procedure-oriented programming are:

- 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.

1.4 Object Oriented 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 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.1.3. 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 of the 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 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.

Object-oriented programming is the most recent concept among programming paradigms and still means different things to different people.

1.5 Basic Concepts of Object Oriented Programming

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

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

We shall discuss these concepts in some detail in this section.

1.5.1 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. Foe 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 a sufficient to know the type of message accepted, and the type of response returned by the objects. Although different author

represent them differently fig 1.5 shows two notations that are popularly used in object-oriented analysis and design.

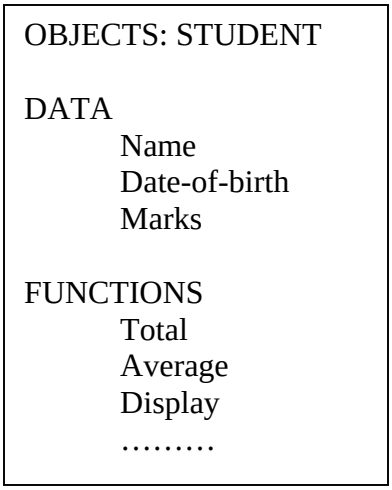


Fig. 1.5 representing an object

1.5.2 Classes

We just mentioned that 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, Mango, Apple and orange members of class fruit. Classes are user-defined that types and behave like the built-in 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.**

1.5.3 Data Abstraction and Encapsulation

The wrapping up of data and function into a single unit (called class) is known as *encapsulation.* Data and 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 direct access by the program is called *data hiding or information hiding*.

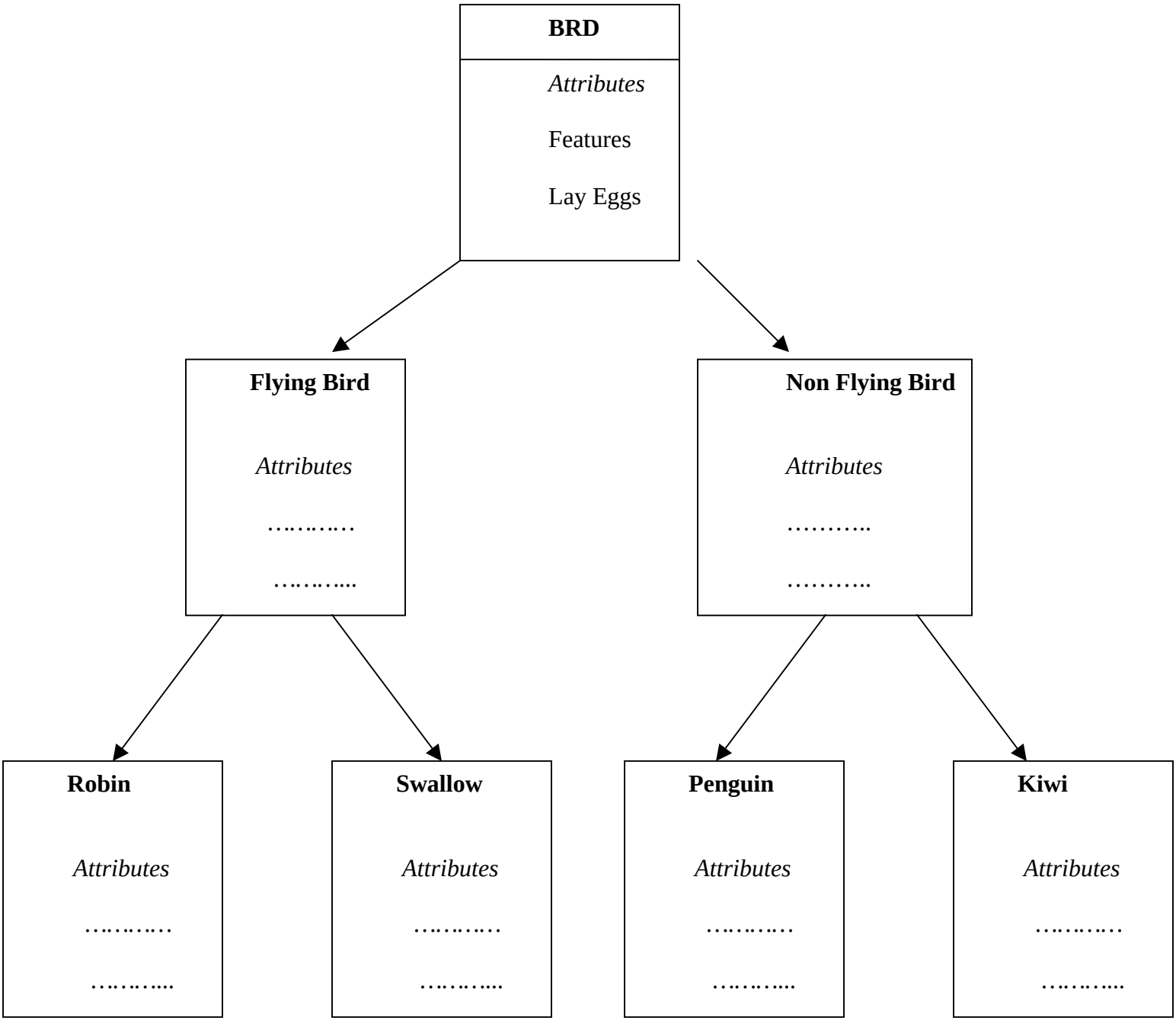
Abstraction refers to the act of representing essential features without including the background details or explanation. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, wait, and cost, and function operate on these attributes. They encapsulate all 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*.

1.5.4 Inheritance

Inheritance is the process by which objects of one class acquired the properties of objects of another classes. It supports the concept of *hierarchical classification.* For example, the bird, 'robin' is a part of class 'flying bird' which is again a part of the class 'bird'. The principal behind this sort of division is that each derived class shares common characteristics with the class from which it is derived as illustrated in fig 1.6.
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 feature of both the classes. The real appeal and power of the inheritance mechanism is that it

Fig. 1.6 Property inheritances



Allows the programmer to reuse a class i.e almost, but not exactly, what he wants, and to tailor the class in such a way that it does not introduced any undesirable side-effects into the rest of classes.

1.5.5 Polymorphism

Polymorphism is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form. An operation may exhibit different behavior is 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*.

Fig. 1.7 illustrates that a single function name can be used to handle different number and different types of argument. This is something similar to a particular word having several different meanings depending upon the context. Using a single function name to perform different type of task is known as *function overloading.*

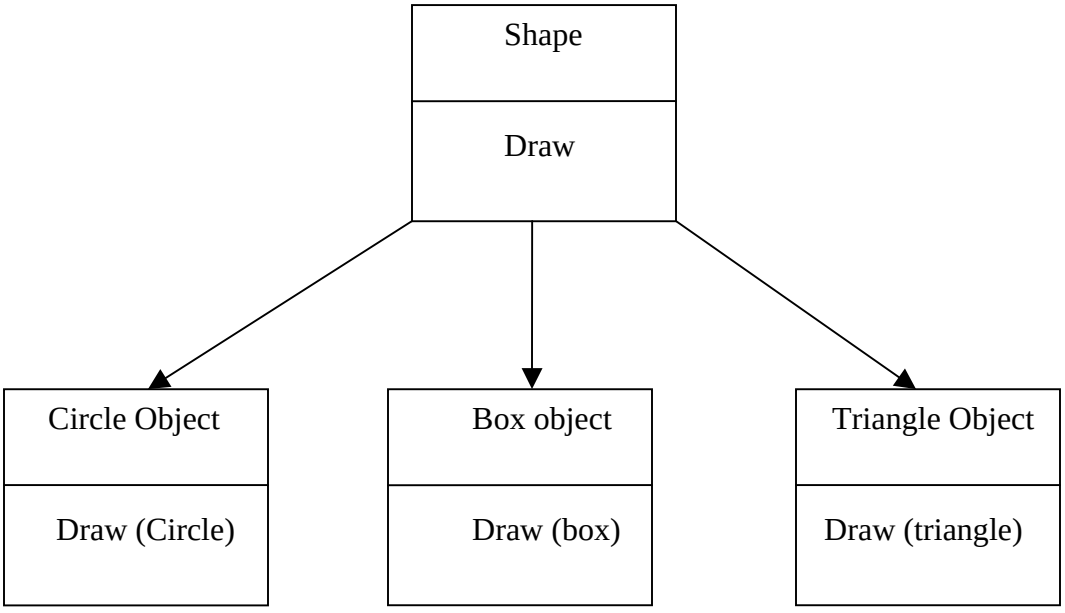


Fig. 1.7 Polymorphism

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 action associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.

1.5.6 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" in fig. 1.7. 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.

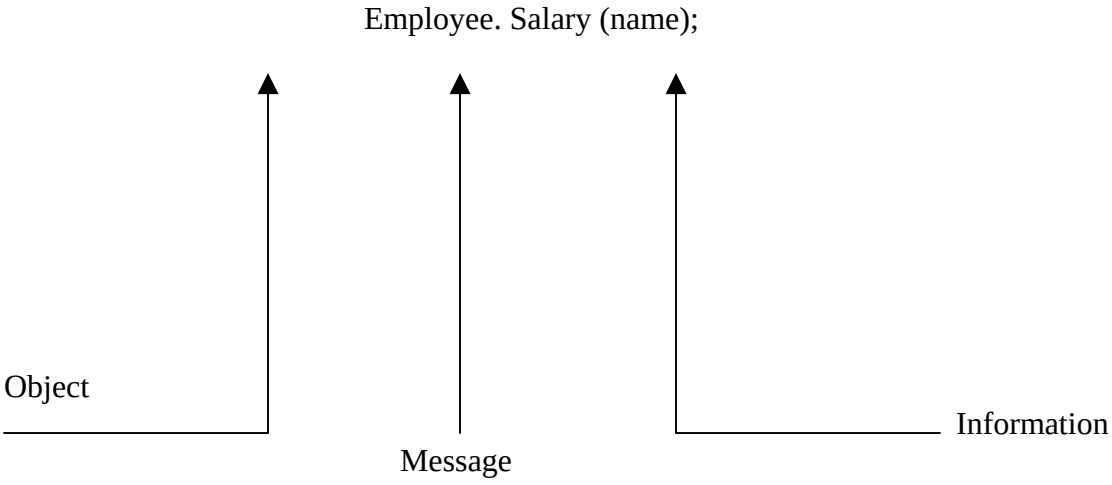
1.5.7 Message Passing

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

1. Creating classes that define object 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 concept 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 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. Example:



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

1.6 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 principal advantages are:

- Through inheritance, we can eliminate redundant code extend the use of existing
- Classes.
- 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.
- The principle of data hiding helps the programmer to build secure program that can not be invaded by code in other parts of a programs.
- It is possible to have multiple instances of an object to co-exist without any interference.
- It is possible to map object in the problem domain to those in the program.
- It is easy to partition the work in a project based on objects.
- The data-centered design approach enables us to capture more detail of a model can implemental form.
- Object-oriented system can be easily upgraded from small to large system.
- Message passing techniques for communication between objects makes to interface descriptions with external systems much simpler.
- Software complexity can be easily managed.

While it is possible to incorporate all these features in an object-oriented system, their importance depends on the type of the project and the preference of the programmer. There are a number of issues that need to be tackled to reap some of the benefits stated above. For instance, object libraries must be available for reuse. The technology is still developing and current product may be superseded quickly. Strict controls and protocols need to be developed if reuse is not to be compromised.

1.7 Object Oriented Language

Object-oriented programming is not the right of any particular languages. Like structured programming, OOP concepts can be implemented using languages such as C and Pascal. However, programming becomes clumsy and may generate confusion when the programs grow large. A language that is specially id designed to support the OOP concepts makes it easier to implement them.

The languages should support several of the OOP concepts to claim that they are object-oriented. Depending upon the features they support, they can be classified into the following two categories:

1. Object-based programming languages, and
2. Object-oriented programming languages.

Object-based programming is the style of programming that primarily supports encapsulation and object identity. Major feature that are required for object based programming are:

- Data encapsulation
- Data hiding and access mechanisms
- Automatic initialization and clear-up of objects
- Operator overloading

Languages that support programming with objects are said to the objects-based programming languages. They do not support inheritance and dynamic binding. Ada is a typical object-based programming language.

Object-oriented programming language incorporates all of object-based programming features along with two additional features, namely, inheritance and dynamic binding. Object-oriented programming can therefore be characterized by the following statements:

Object-based features + inheritance + dynamic binding

1.8 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

The object-oriented paradigm sprang from the language, has matured into design, and has recently moved into analysis. It is believed that the richness of OOP environment will enable the software industry to improve not only the quality of software system but also its productivity. Object-oriented technology is certainly going to change the way the software engineers think, analyze, design and implement future system.

1.9 Introduction of C++

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories in Murray Hill, New Jersey, USA, in the early 1980's. Stroustrup, an admirer of Simula67 and a strong supporter of C, wanted to combine the best of both the languages and create a more powerful language that could support object-oriented programming features and still retain the power and elegance of C. The result was C++. Therefore, C++ is an extension of C with a major addition of the class construct feature of Simula67. Since the class was a major addition to the original C language, Stroustrup initially called the new language 'C with classes'. However, later in 1983, the name was changed to C++. The idea of C++ comes from the C increment operator ++, thereby suggesting that C++ is an augmented version of C.

C+ + is a superset of C. Almost all c programs are also C++ programs. However, there are a few minor differences that will prevent a c program to run under C++ complier. We shall see these differences later as and when they are encountered.

The most important facilities that C++ adds on to C care classes, inheritance, function overloading and operator overloading. These features enable creating of abstract data types, inherit properties from existing data types and support polymorphism, thereby making C++ a truly object-oriented language.

1.9.1 Application of C++

C++ is a versatile language for handling very large programs; it is suitable for virtually any programming task including development of editors, compilers, databases, communication systems and any complex real life applications systems.

- Since C++ allow us to create hierarchy related objects, we can build special object-oriented libraries which can be used later by many programmers.
- While C++ is able to map the real-world problem properly, the C part of C++ gives the language the ability to get closed to the machine-level details.
- C++ programs are easily maintainable and expandable. When a new feature needs to be implemented, it is very easy to add to the existing structure of an object.
- It is expected that C++ will replace C as a general-purpose language in the near future.

1.10 Simple C++ Program

Let us begin with a simple example of a C++ program that prints a string on the screen.

Printing A String

```
#include<iostream>
Using namespace std;
int main()
{
cout<<" c++ is better than c \n";
return 0;
}
```

Program 1.10.1

This simple program demonstrates several C++ features.

1.10.1 Program feature

Like C, the C++ program is a collection of function. The above example contain only one function **main().** As usual execution begins at main(). Every C++ program must have a **main()**. C++ is a free form language. With a few exception, the compiler ignore carriage return and white spaces. Like C, the C++ statements terminate with semicolons.

1.10.2 Comments

C++ introduces a new comment symbol // (double slash). Comment start with a double slash symbol and terminate at the end of the line. A comment may start anywhere in the line, and whatever follows till the end of the line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multiline comments can be written as follows:

```
// This is an example of
// C++ program to illustrate
// some of its features
```

The C comment symbols /*,*/ are still valid and are more suitable for multiline comments. The following comment is allowed:

```
/* This is an example of
   C++ program to illustrate
    some of its features
*/
```

1.10.3 Output operator

The only statement in program 1.10.1 is an output statement. The statement

Cout<<”C++ is better than C.”;

Causes the string in quotation marks to be displayed on the screen. This statement introduces two new C++ features, cout and <<. The identifier cout(pronounced as C out) is a predefined object that represents the standard output stream in C++. Here, the standard output stream represents the screen. It is also possible to redirect the output to other output devices. The operator << is called the insertion or put to operator.

1.10.4 The iostream File

We have used the following #include directive in the program:

#include <iostream>

The #include directive instructs the compiler to include the contents of the file enclosed within angular brackets into the source file. The header file **iostream.h** should be included at the beginning of all programs that use input/output statements.

1.10.5 Namespace

Namespace is a new concept introduced by the ANSI C++ standards committee. This defines a scope for the identifiers that are used in a program. For using the identifier defined in the **namespace** scope we must include the using directive, like

Using namespace std;

Here, std is the namespace where ANSI C++ standard class libraries are defined. All ANSI C++ programs must include this directive. This will bring all the identifiers defined in std to the current global scope. **Using** and **namespace** are the new keyword of C++.

1.10.6 Return Type of main()

In C++, main () returns an integer value to the operating system. Therefore, every main () in C++ should end with a return (0) statement; otherwise a warning an error might occur. Since main () returns an integer type for main () is explicitly specified as **int.** Note that the default return type for all function in C++ is **int.** The following main without type and return will run with a warning:

```
main ()
{
  ……..
  …….
}
```

1.11 More C++ Statements

Let us consider a slightly more complex C++ program. Assume that we should like to read two numbers from the keyboard and display their average on the screen. C++ statements to accomplish this is shown in program 1.11.1

AVERAGE OF TWO NUMBERS

```
#include<iostream.h> // include header file

Using namespace std;

Int main()

{

        Float number1, number2,sum, average;
        Cin >> number1;       // Read Numbers
        Cin >> number2;       // from keyboard
        Sum = number1 + number2;
        Average = sum/2;
        Cout << ”Sum = “ << sum << “\n”;
        Cout << “Average = “ << average << “\n”;

        Return 0;

}       //end of example
```

The output would be:

```
Enter two numbers: 6.5  7.5
Sum = 14
Average = 7
```

Program 1.11.1

1.11.1 Variables

The program uses four variables number1, number2, sum and average. They are declared as type float by the statement.

```
float number1, number2, sum, average;
```

All variable must be declared before they are used in the program.

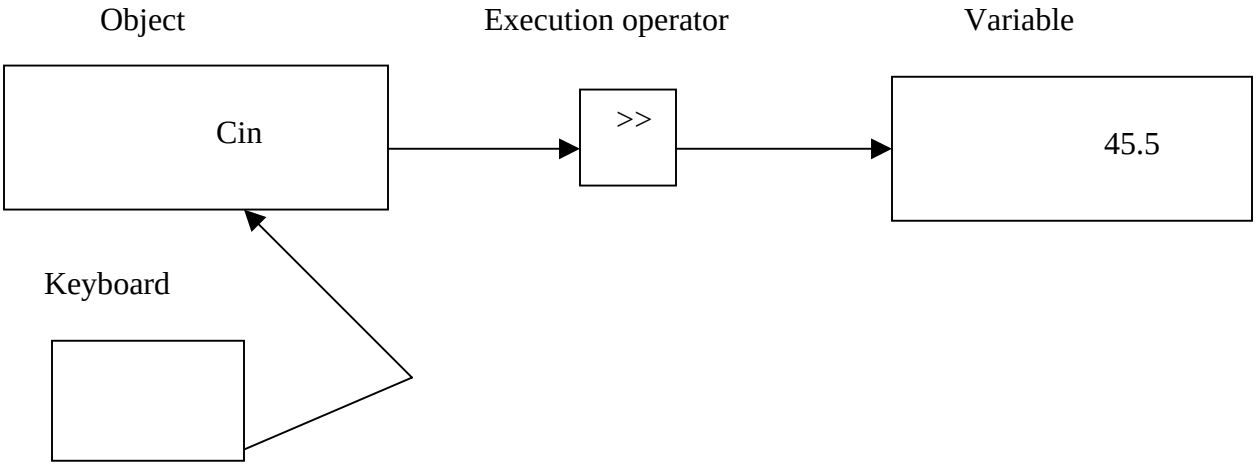
1.11.2 Input Operator

The statement

cin >> number1;

Is an input statement and causes the program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier cin (pronounced ‘C in’) is a predefined object in C++ that corresponds to the standard input stream. Here, this stream represents the keyboard.

The operator >> is known as extraction or get from operator. It extracts (or takes) the value from the keyboard and assigns it to the variable on its right fig 1.8. This corresponds to a familiar scanf() operation. Like <<, the operator >> can also be overloaded.



1.8 Input using extraction operator

1.11.3 Cascading of I/O Operators

We have used the insertion operator << repeatedly in the last two statements for printing results.

The statement

Cout << “Sum = “ << sum << “\n”;

First sends the string “Sum = “ to cout and then sends the value of sum. Finally, it sends the newline character so that the next output will be in the new line. The multiple use of << in one statement is called cascading. When cascading an output operator, we should ensure necessary blank spaces between different items. Using the cascading technique, the last two statements can be combined as follows:

```
Cout << “Sum = “ << sum << “\n”
     << “Average = “ << average << “\n”;
```

This is one statement but provides two line of output. If you want only one line of output, the statement will be:

```
Cout << "Sum = " << sum << ","
     << "Average = " << average << "\n";
```

The output will be:

```
Sum  = 14, average = 7
```

We can also cascade input iperator >> as shown below:

```
Cin >> number1 >> number2;
```

The values are assigned from left to right. That is, if we key in two values, say, 10 and 20, then 10 will be assigned to munber1 and 20 to number2.

1.12 An Example with Class

- One of the major features of C++ is classes. They provide a method of binding together data and functions which operate on them. Like structures in C, classes are user-defined data types.

Program 1.12.1 shows the use of class in a C++ program.

USE OF CLASS

```
#include<iostream.h> // include header file

using namespace std;
class person
{

        char name[30];
        Int age;

        public:
            void getdata(void);
            void display(void);
};
void person :: getdata(void)
{
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter age: ";
        cin >> age;
```

```
}
Void person : : display(void)
{
        cout << "\nNameame: " << name;
        cout << "\nAge: " << age;
}

Int main()
{
        person p;
        p.getdata();
        p.display();

        Return 0;

}       //end of example
```

PROGRAM 1.12.1

The output of program is:

```
Enter Name: Ravinder
Enter age:30
Name:Ravinder
Age: 30
```

The program define **person** as a new data of type class. The class person includes two basic data type items and two function to operate on that data. These functions are called **member function**. The main program uses **person** to declare variables of its type. As pointed out earlier, class variables are known as objects. Here, p is an object of type **person**. Class object are used to invoke the function defined in that class.

1.13 Structure of C++ Program

As it can be seen from program 1.12.1, a typical C++ program would contain four sections as shown in fig. 1.9. This section may be placed in separate code files and then compiled independently or jointly.

It is a common practice to organize a program into three separate files. The class declarations are placed in a header file and the definitions of member functions go into another file. This approach enables the programmer to separate the abstract specification of the interface from the implementation details (member function definition).

Finally, the main program that uses the class is places in a third file which "includes: the previous two files as well as any other file required.

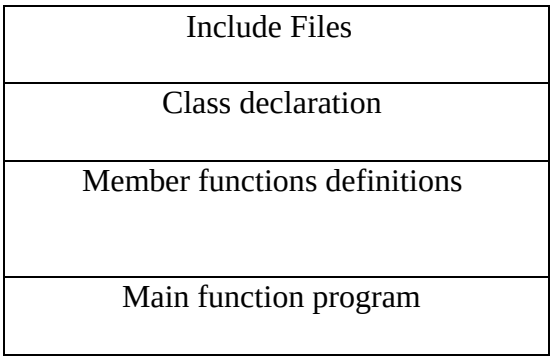
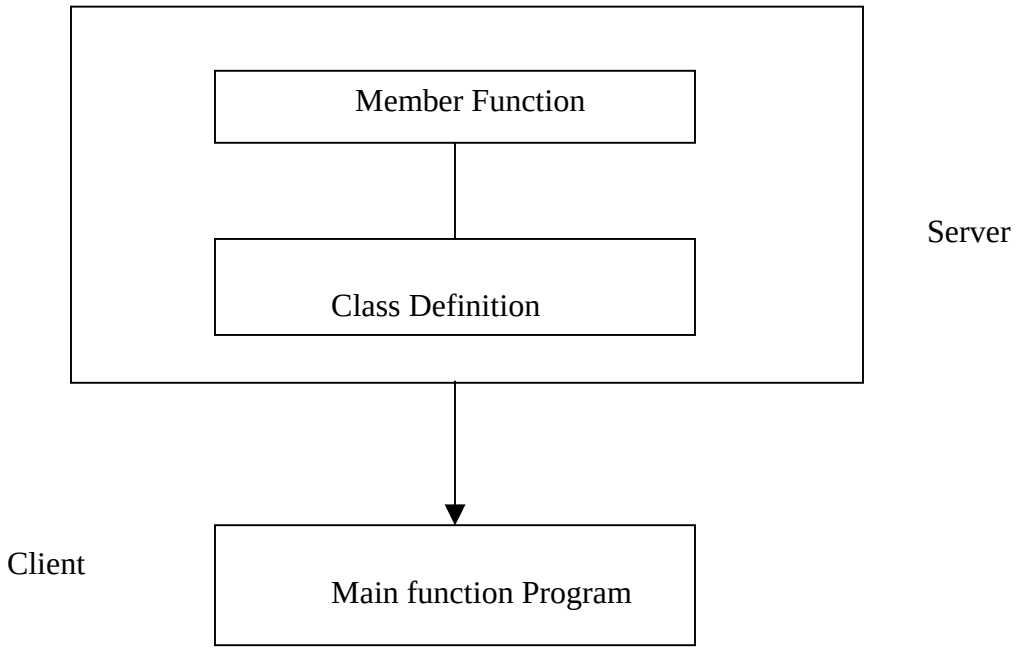


Fig 1.9 Structure of a C++ program

This approach is based on the concept of client-server model as shown in fig. 1.10. The class definition including the member functions constitute the server that provides services to the main program known as client. The client uses the server through the public interface of the class.

Fig. 1.10 *The client-server model*



1.14 Creating the Source File

Like C programs can be created using any text editor. Foe example, on the UNIX, we can use vi or ed text editor for creating using any text editor for creating and editing the source code. On the DOS system, we can use endlin or any other editor available or a word processor system under non-document mode.

Some systems such as Turboc C++ provide an integrated environment for developing and editing programs

The file name should have a proper file extension to indicate that it is a C++ implementations use extensions such as .c,.C, .cc, .cpp and .cxx. Turboc C++ and Borland C++ use .c for C programs and .cpp(C plus plus) for C++ programs. Zortech C++ system use .cxx while UNIX AT&T version uses .C (capital C) and .cc. The operating system manuals should be consulted to determine the proper file name extension to be used.

1.15 Compiling and Linking

The process of compiling and linking again depends upon the operating system. A few popular systems are discussed in this section.

Unix AT&T C++

This process of implementation of a C++ program under UNIX is similar to that of a C program. We should use the "cc" (uppercase) command to compile the program. Remember, we use lowercase "cc" for compiling C programs. The command

CC example.C

At the UNIX prompt would compile the C++ program source code contained in the file **example.C**. The compiler would produce an object file **example.o** and then automatically link with the library functions to produce an executable file. The default executable filename is **a. out**.

A program spread over multiple files can be compiled as follows:

CC file1.C file2.o

The statement compiles only the file **file1.C** and links it with the previously compiled **file2.o** file. This is useful when only one of the files needs to be modified. The files that are not modified need not be compiled again.

Turbo C++ and Borland C++

Turbo C++ and Borland C++ provide an integrated program development environment under MS DOS. They provide a built-in editor and a menu bar includes options such as File, Edit, Compile and Run.

We can create and save the source files under the **File option,** and edit them under the **Edit option.** We can then compile the program under the **compile** option and execute it under the **Run option.** The **Run option** can be used without compiling the source code.

Summary

- Software technology has evolved through a series of phases during the last five decades.
- POP follows top-down approach where problem is viewed as sequence of task to be performed and functions are written for implementing these tasks.

- POP has two major drawbacks:
- Data can move freely around the program.
- It does not model very well the real-world problems.
- OOP was inventing to overcome the drawbacks of POP. It follows down -up approach.
- In OOP, problem is considered as a collection of objects and objects are instance of classes.
- Data abstraction refers to putting together essential features without including background details.
- Inheritance is the process by which objects of one class acquire properties of objects of another class.
- Polymorphism means one name, multiple forms. It allows us to have more than one function with the same name in a program.
- Dynamic binding means that the code associated with a given procedure is not known until the time of the run time.
- Message passing involves specifying the name of the object, the name of the function and the information to be sent.
- C++ is a superset of C language.
- C++ ads a number of features such as objects, inheritance, function overloading and operator overloading to C.
- C++ supports interactive input and output features and introduces anew comment symbol // that can be used for single line comment.
- Like C programs, execution of all C++ program begins at main() function.

Keywords:

- Assembly Language
- Bottom up Programming
- C++
- Classes
- Data Abstraction
- Data Encapsulation
- Data Hiding
- Data Member
- Dynamic Binding
- Early Binding
- Function overloading
- Functions
- Global Data
- Hierarchical Classification
- Inheritance
- Late Binding
- #include
- Main( )
- Local data
- Machine Language
- Member Function
- Message Passing
- Methods
- Modular Programming
- Multiple Inheritances
- Object Based Programming
- Objective C
- Object Oriented Language
- Object Oriented Programming
- Objects
- Operator Overloading
- Polymorphism
- Procedure Oriented Programming
- Reusability
- Top down Programming
- Extraction Operator

- Cascading
- Namespace
- Class
- Object
- Operator overloading
- Comments
- Output operator
- cout
- edlin
- return ()
- Float
- Get from Operator
- Input operator
- Turbo c++
- iostream
- int
- using
- iostream.h
- windows
- Keyboard

Questions

1. What are the major issues facing the software industry today?
2. What is POP? Discuss its features.
3. Describe how data are shared by functions in procedure-oriented programs?
4. What is OOP? What are the difference between POP and OOP?
5. How are data and functions organized in an object-oriented program?
6. What are the unique advantages of an object-oriented programming paradigm?
7. Distinguish between the following terms:
   (a) Object and classes
   (b) Data abstraction and data encapsulation
   (c) Inheritance and polymorphism
   (d) Dynamic binding and message passing
8. Describe inheritance as applied to OOP.
9. What do you mean by dynamic binding? How it is useful in OOP?
10. What is the use of preprocessor directive #include<iostream>?
11. How does a main () function in c++ differ from main () in c?
12. Describe the major parts of a c++ program.
13. Write a program to read two numbers from the keyboard and display the larger value on the screen.
14. Write a program to input an integer value from keyboard and display on screen "WELL DONE" that many times.

References:

1. Object –Oriented –Programming in C++ by E Balagurusamy.
2. Object –Oriented –Programming with ANSI & Turbo C++ by Ashok N. Kamthane.
3. OO Programming in C++ by Robert Lafore, Galgotia Publications Pvt. Ltd.
4. Mastering C++ By K R Venugopal, Rajkumar Buyya, T Ravishankar.
5. Object Oriented Programming and C++ By R. Rajaram.
6. Object –Oriented –Programming in C++ by Robert Lafore.