

## 4 Basics of object oriented programming and procedure oriented programming

### 4.1 Procedure-Oriented Programming

In the procedure oriented approach, the problem is viewed as the sequence of things to be done such as reading, calculating and printing. The primary focus is on functions. A typical structure for procedural programming is shown in figure below. The technique is to divide a larger problem into smaller problems.

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.

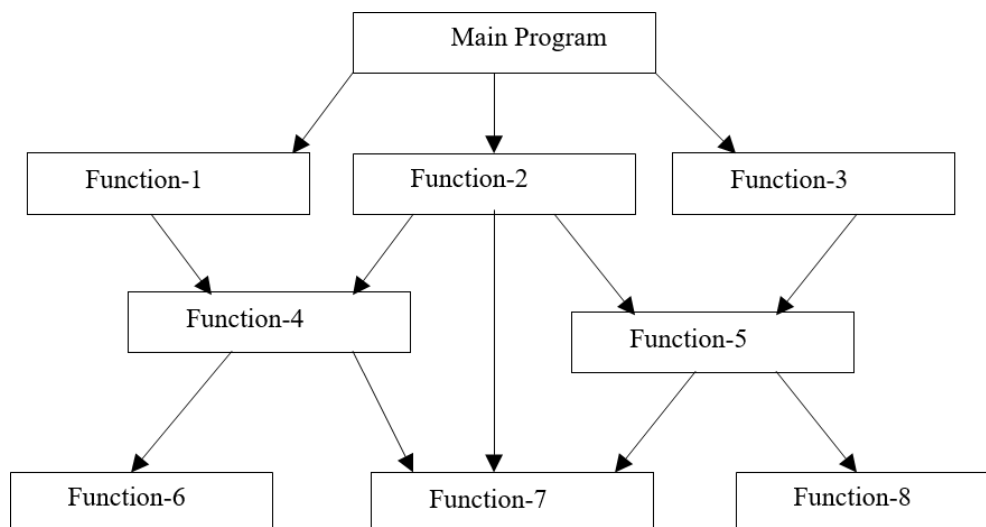


Fig. 1.2 Typical structure of procedural oriented programs

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:

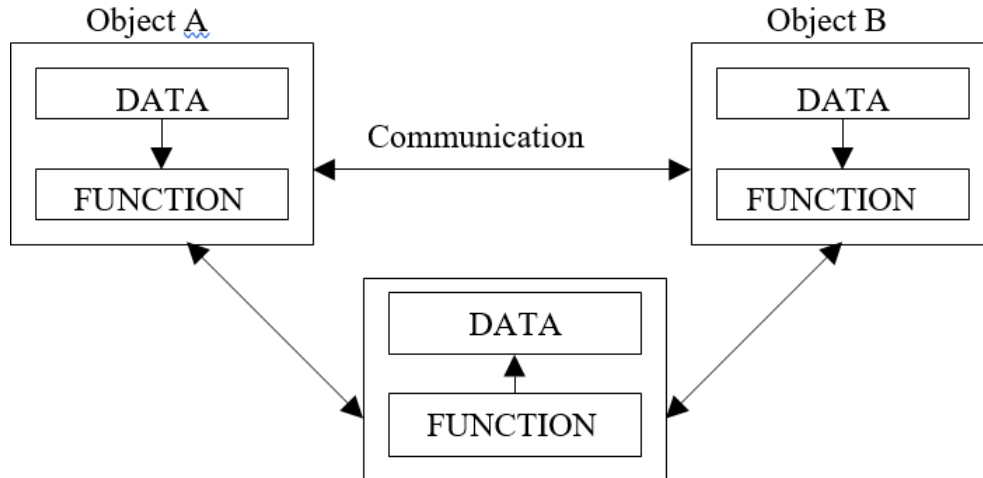
- Emphasis is on doing things (algorithms).
- Large programs are divided into smaller programs known as functions.
- Most of the functions share global data.
- Data move openly around the system from function to function.
- Functions transform data from one form to another.
- Employs top-down approach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

### 4.2 Object Oriented Programming

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 tied 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.

#### **4.2.1 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.

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. 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.

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 account object requesting for the bank balance. Each object contain data, and code to manipulate data.

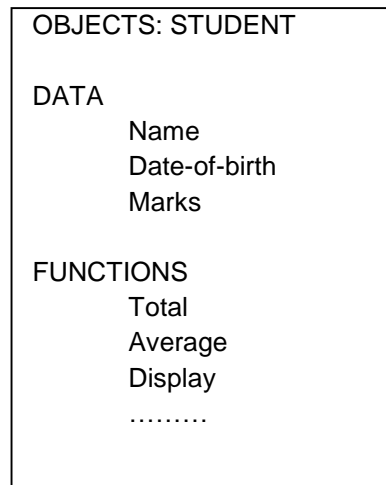


Fig. 1.5 representing an object

### Classes

We just mentioned that objects contain data, and code to manipulate that data. The entire set of data and code of an object is defined with the help of class. Once a class has been defined, we can create any number of objects belonging to that class. Hence objects are variables of the type 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. 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

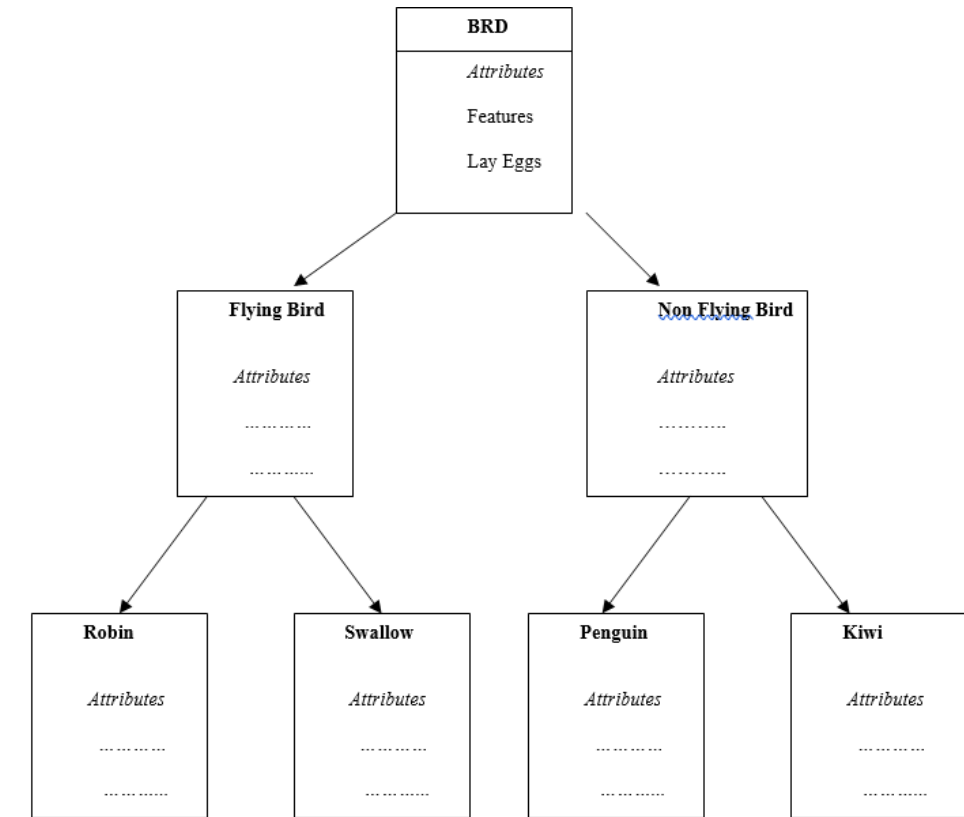
The wrapping up of data and function into a single unit (called class) is known as *encapsulation*. Data and encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

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. Like you don't need to understand working of a car to drive.

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*.

## Inheritance

*Inheritance* is the process by which objects of one class acquired the properties of objects of another classes. 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 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.

## Polymorphism

*Polymorphism* is another important OOP concept. Polymorphism, a Greek term, means the ability to take more than on form. An example you will study is function 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*.

## Message Passing

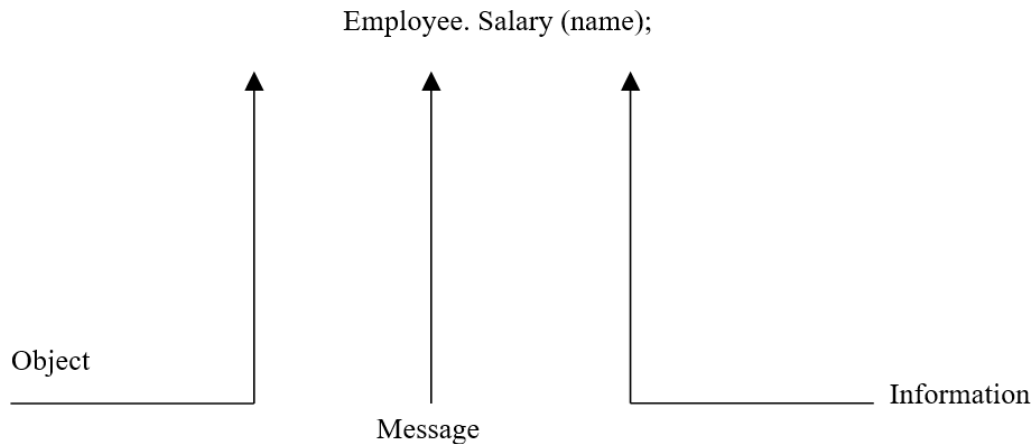
An object-oriented program consists of a set of objects that communicate with each other. The process of programming in an object-oriented language, involves the following basic steps:

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.

#### 4.3 Benefits of OOP

OOP offers several benefits to both the program designer and the user. Object-Oriented 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.

Applications of OOP

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

### **5 Introduction to C++**

C++ is an object-oriented programming language. It was developed by Bjarne Stroustrup at AT&T Bell Laboratories 1980's.

The most important facilities that C++ adds compared to other programming languages are classes, inheritance, function overloading.

#### **5.1 Structure of C++ Program**

A typical C++ program would contain four sections as shown in figure below. The first section is include section where header files needed for the program is placed. Next section classes are declared if object are needed in the program and function prototypes or declarations. Next section will have definitions of member functions of classes. Next section will have main function which joins together all these sections. Last section will have function definitions.

These sections can be placed in different files or in the same file.

Include Files
---------------

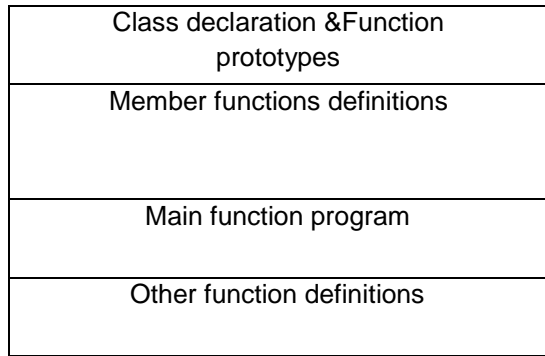
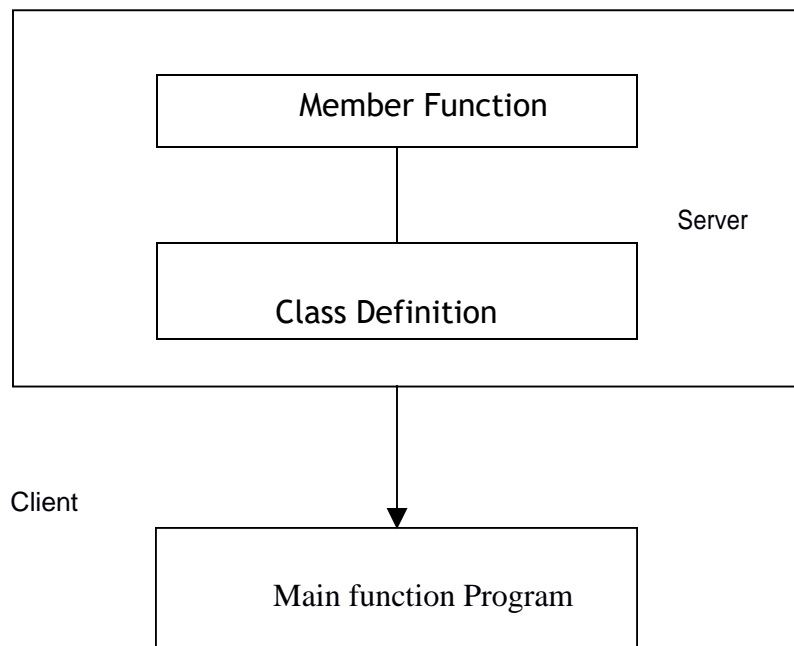


Fig 1.9 Structure of a C++ program

This approach is based on the concept of client-server model, where client asks for the service of the server like your browser and a website, as shown in figure below. 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



## 5.2 Simple C++ Program

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

## Printing A String

```
#incl
ude<i
ostre
am>
Usin
g
name
space
std;
int
main(
)
{
cout<<"C
++ is the
best ";
return 0;
}
```

```
Pro
gra
m
1.10
.1
Out
put
C++ is the best
```

This simple program demonstrates several C++ features.

### Program feature

Every C++ program must a main() function. Also the execution of every C++ program begins at main()(even if we have other functions above it). C++ statements(simple lines) terminate with semicolons ';'.

### Comments

Comments are special lines which will not be executed. They are used to help the reader of a program understand it better.

The double slash comment is basically a single line comment, you can write comment that is one line long.

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



If you want to write comments that have more than one line use /\*,\*/:

```
/*  
T  
h  
i  
s  
i  
s  
a  
n  
e  
x  
a  
m  
p  
l  
e  
o  
f  
C  
+  
+  
p  
r  
o  
g  
r  
a  
m  
t  
o  
i  
l  
l  
u  
s  
t  
r  
a  
t  
e  
s  
o  
m  
e  
o  
f
```

```
    i  
    t  
    s  
    f  
    e  
    a  
    t  
    u  
    r  
    e  
    s  
    *
```

```
 /  
  
 O  
  
 u  
  
 t  
  
 p  
  
 u  
  
 t  
  
 o  
  
 p  
  
 e  
  
 r  
  
 a  
  
 t  
  
 o  
  
 r
```

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(monitor) 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.

#### The `iostream` File

We have used the following `#include` directive in the program: `#include <iostream>`

The `#include` directive instructs the compiler(the software which converts your program into an exe and also checks for any mistakes or errors in your code) to include the contents of the file enclosed within angular brackets into the source file. The header file **`iostream`** should be included at the beginning of all programs that use input/output statements that is to print or read from keyboard.

#### 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++.

#### 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**.

## Introduction to class and Object

### 4.1.1 DEFINITION AND DECLARATION OF A CLASS

A class in C++ combines related data and functions together. It makes a data type which is used for creating objects of this type.

Classes represent real world entities that have both data type properties (characteristics) and associated operations

(behaviour) like a bank account which has properties like account number, balance and functions like withdraw amount or deposit amount.

The syntax of a class definition is shown below :

```
Class name_of_class
{
private : variable declaration; // data member
        Function declaration; // Member
Function (Method) protected: Variable
declaration;
        Function
declaration; public
        : variable
declaration;
        Function declaration;
};
```

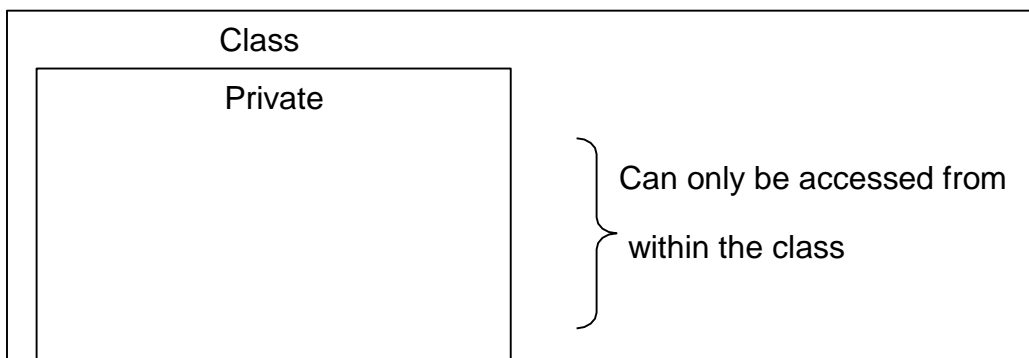
Here, the keyword class specifies that we are using a new data type and is followed by the class name.

The body of the class has two keywords namely :

(i) *private*

(ii) *public*

In C++, the keywords **private** and **public** are called access specifiers. The data hiding concept in C++ is achieved by using the keyword **private**. Private data and functions can only be accessed from within the class itself. Public data and functions are accessible outside the class also. This is shown below :



data members  
and  
member functions

data members  
and  
member functions

Data hiding not mean the security technique used for protecting computer databases.

The security measure is used to protect unauthorized users from performing any operation (read/write or modify) on the data.

The data declared under **Private** section are hidden and safe from accidental manipulation. Though the user can use the private data but not by accident.

The functions that operate on the data are generally **public** so that they can be accessed from outside the class but this is not a rule that we must follow.

#### 4.1.2 Data Members

Data members are variables that you create in a normal program. Here while we define members we also specify its protection level(public, private). Unlike a normal variable data members are usually accessed within a class.

#### 4.1.3 MEMBER FUNCTION DEFINITION

The class specification can be done in two part :

- (i) **Class definition.** It describes both data members and member functions.
- (ii) **Class method definitions.** It describes how certain class member functions are coded.

We have already seen the class definition syntax as

well as an example. In C++, the member functions can

be coded in two ways :

*(a) Inside class definition*

*(b) Outside class definition using scope resolution operator (::)*

The code of the function is same in both the cases, but the function header is different as explained below :

##### 4.1.3.1 Inside Class Definition:

When a member function is defined inside a class, we do not require to place a membership label along with the function name. We use only small functions inside the class definition and such functions are known as **inline** functions.

In case of inline function the compiler inserts the code of the body of the function at the place where it is invoked (called) and in doing so the program execution is faster but memory penalty is there.

#### 4.1.3.2 Outside Class Definition Using Scope Resolution Operator (::) :

In this case the function's full name (qualified\_name) is written as shown:

```
Name_of_the_class :: function_name
```

The syntax for a member function definition outside the class definition

is :

```
return_type name_of_the_class::function_name(argument list)
{
    body of function
}
```

Here the operator::known as scope resolution operator helps in defining the member function outside the class.

#### 4.1.4 DECLARATION OF OBJECTS AS INSTANCES OF A CLASS

The objects of a class are declared after the class definition. One must remember that a class definition does not define any objects of its type, but it defines the properties of a class. For utilizing the defined class, we need variables of the class type. For example,

```
Complex comnum; //object declaration
```

will create two objects **ob1** complex class type. As mentioned earlier, in C++ the variables of a class are known as objects. These are declared like a simple variable i.e., like fundamental data types.

#### 4.1.5 Accessing Class Members

The private data of a class can be accessed only through the member functions of that class. The main() cannot contain statements that access private directly. The public members of a class can be accessed in the following syntax(for a function),

```
Object_name.function_name (arguments);
```

For example, the function call statement  
complexnum. display();

Example program #include<iostream>

```
using namespace std;
class complex
{
    int real;
    int imag;
public:
    complex()
    {
        real=0;
        imag=0;
    }

    void display()
    {
        cout<<real<<"+"<<imag<<"i"<<endl;
    }

};

int main()
{
    complex cnum;
    cnum.display();
    return 0;
}
```

#### **4.2 Constructor**

- A constructor is a special member function whose task is to initialize the object of a class.
- Its name is same as the class name.
- A constructor does not have a return type.



- A constructor is called or invoked when the object of its associated class is created.
- It is called constructor because it constructs the values of data members of the class.
- There three types of constructor:
  - (i) Default Constructor
  - (ii) Parameterized Constructor

#### 4.2.1 Default Constructor

The constructor which has no arguments is known as default constructor.

#### Demonstration of default Constructor.

Use the same program as above

#### 4.2.2 Parameterized constructor

The constructor which takes some argument is known as parameterized constructor.

```
#include<iostream>
using namespace std;
class complex
{
    int real;
    int imag;
public:
    complex(int r, int i)
    {
        real=r;
        imag=i;
    }
    void display()
    {
        cout<<real<<"+"<<imag<<"i"<<endl;
    }
};
int main()
{
    complex cnum(10,5);
    cnum.display();
    return 0;
}
```

A parameterized constructor can be called:

- (i) **Implicitly:** account jimsaccount(1000);
- (ii) **Explicitly :** account samsaccount=account(5000);

#### Destructor

- A destructor is used to destroy the objects that have been created by a constructor.

- Like constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde.

eg: `~integer () {}`

- A destructor never takes any argument nor does it return any value.
- It will be invoked automatically upon exit from the program – or block or function as the case may be – to clean up storage that is no longer accessible.
- It is a good practice to declare destructors in a program since it releases memory space for further use.
- Whenever **new** is used to allocate memory in the constructor, we should use **delete** to free that memory.
-



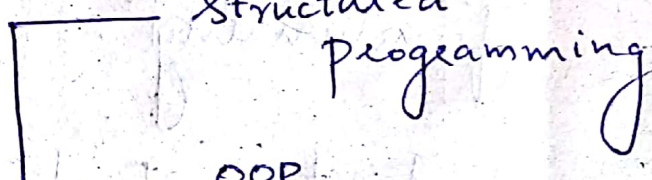
6/01/18

## MODULE - I

Translator - it is used to translate one form of language into another.

3 different types language

- Machine language
- Assembly language
- High level language

High level language 

Structured programming → Main program is divided into different function or procedure or sub program

Data hiding property not possible in C

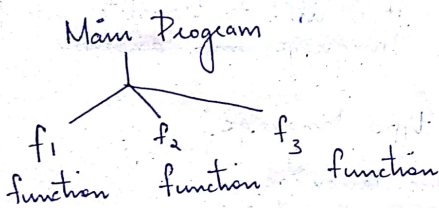
Eg: C, FORTRAN

Security is not consider more here.

Here data is not much considered, more

importance to function.

\* Top-bottom approach.



### OOP

- more importance to the data not on functionality.

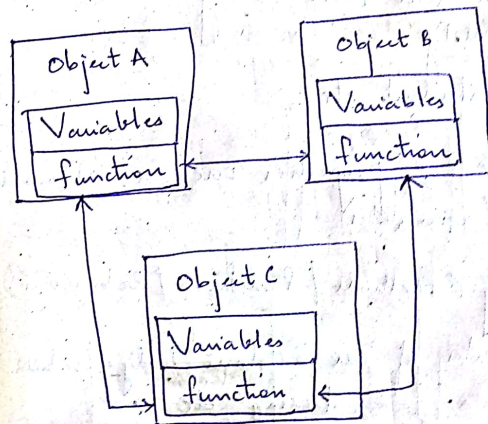
- objects → Features/Attribute  
          ↳ functionalities

- Bottom - Top approach

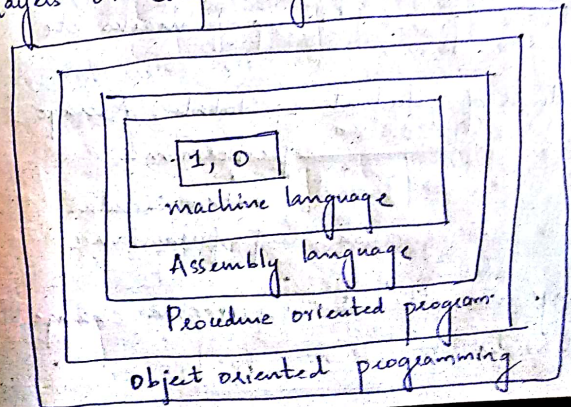
- Data hiding property is possible here

- Security is concerned more.

Object [ Variables / data  
          ] functions.



### layers of Computer System



## Concept of OOPs

1) Object : It is a piece of code which represents a real life thing or real life entity.

Eg: Car, Student etc

An object has 2 Properties one in attributes (features / variables)

Functionality (Methods / Functions / Behaviours)

Eg 1: Attributes of car - Colour, body, wheel steering etc

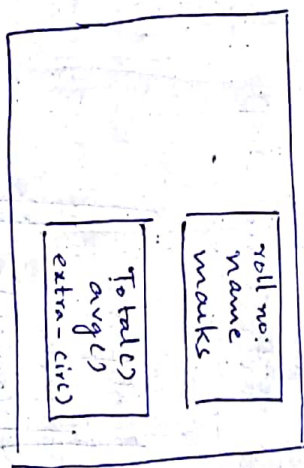
Eg 2: Attributes of student - name, Roll no, marks etc

Methods of student - Total, Average,

Performance,  
extra circular  
Performance

A change in attribute result in change in behaviour

2) Class : It is specification of object.  
Class student

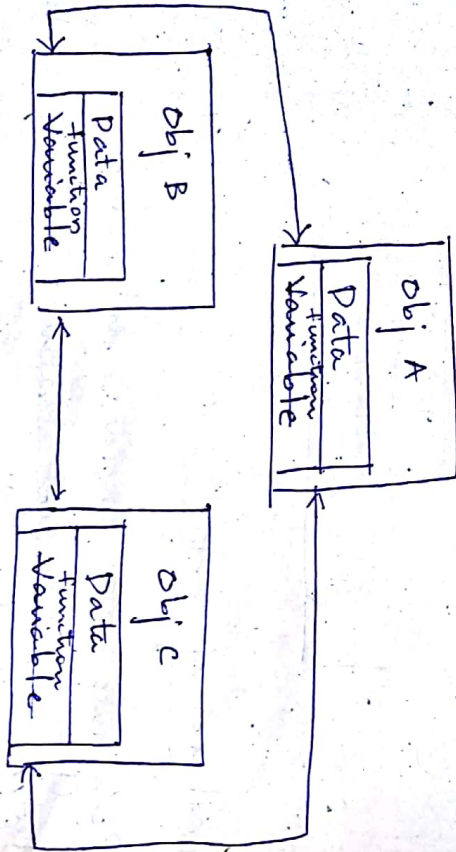


## Features of OOP

- \* Programs are divided into objects
- \* Data is emphasised rather than procedure
- \* Data structure are designed such that they characterise the objects
- \* Data structure → arrangement of data in the memory
- \* Data in system and cannot accessed by its external function
- \* Objects communicate through functions
- \* New data & functions are added easily whenever necessary

\* Follows bottom-up approach in program design.

Organisation data in OOP

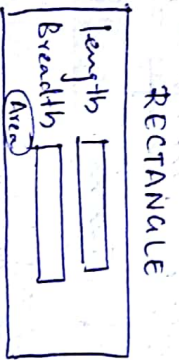


C++ → Biome Strukturrup

8/01/18

3) Data encapsulation

It is an OOP concept that combines / binds data with function that manipulate data

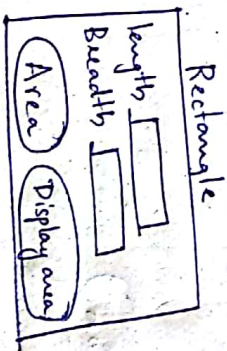


⇒ Data abstraction & hiding

Information hiding

Data hiding → It is OOP concept that hide data from user

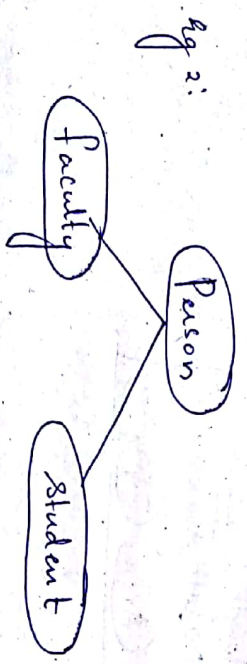
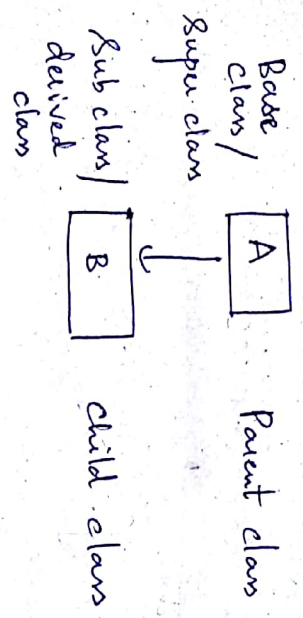
As in the above eg the data, length & breadth are not accessible to the outside world only the function area can access it or only those function which are wrapped into rectangle class, eg display area as shown below



⇒ Data abstraction : It refers to the act of representing essential features without including the background details of explanation.

As in the above eg the calculation of the area above eg (Background details) is not shown to the users  
 → Background details → characters should not be entered, range should be specify → in the code

4) Substantive: It is the process by which the objects of one class acquire the properties of objects of another class. ie Define a class in terms of another class



eg 2:

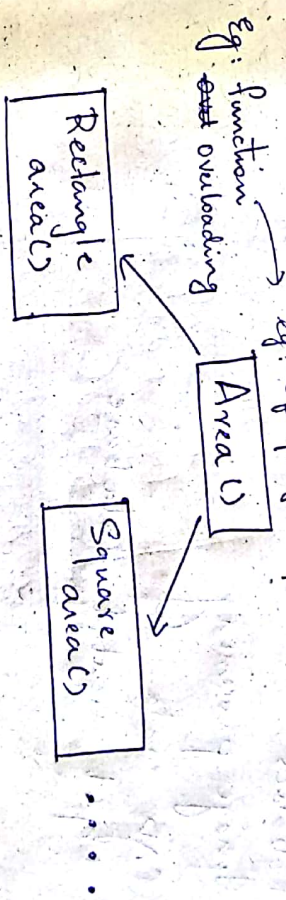
5) Poly morphism: It means the ability to take more than one form. An operator may exhibit diff behaviours  
 Eg: Addition operator  
 In this the case of no: it will generate a sum. In the case of string it will generate

the operator will produce a 3rd string by concatenation

Eg -  $2 + 3 = 5$   
 2) "Hello" + "World" = HelloWorld  
 } operator overloading

6) Dynamic binding  
 → At the run time, data and functions are bind

combined together.  
 Eg: of polymorphism



7) Message passing: Communication between 2 objects is called

message passing  
 Message passing specified by

- i) name of objects
- ii) name of the function (message)
- iii) information to be send sent



Steps

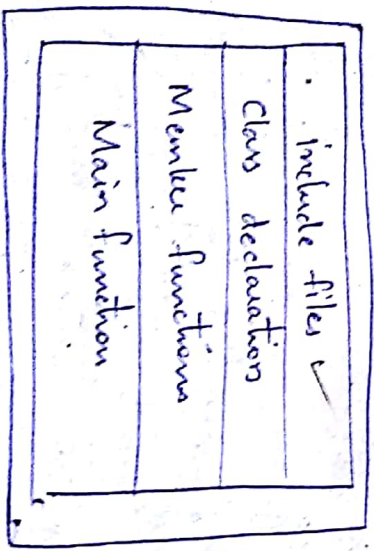
- i) create class
- ii) create object
- iii) establish communication between objects

Benefits Application

- Artificial intelligence
- ~~Simulation~~ Simulation of signals
- Decision making
- Robotics
- Image processing
- CAD
- object oriented database

Structure of C++ Programming

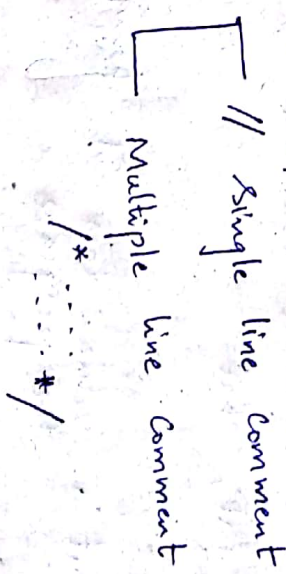
Program features



Program features

1) iostream: it is header file which execute the i/p, o/p operations (cout, cin)

2) Comments



3) Data types

a) character type — 'A', '2' → 1 Byte — 8 bits

b) integer type → whole no: -1, 0, 2, ...

↳ long int

Eg: char c;

int roll no; — 2 Bytes — 16 bits

c) floating point → 6 bit precision

Eg: float marks; (4 bytes)

Double → (8 bytes)

Keywords — int, char  
It cannot be used as ~~variable~~ as declaring as the variable

Identifiers  $\rightarrow$  Variable  
 naming / assigning values  
 to unknown variables

Eg: int, char

Constant  $\rightarrow$  value does not change through out the program

const pi = 3.14

Range

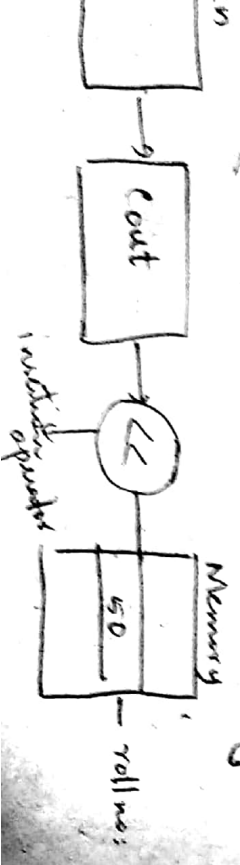
Integer data type -  $2^{-15}$  to  $2^{15}-1$

Character data type -  $2^7$  to  $2^7-1 = -128$  to  $127$

float data type = \_\_\_\_\_

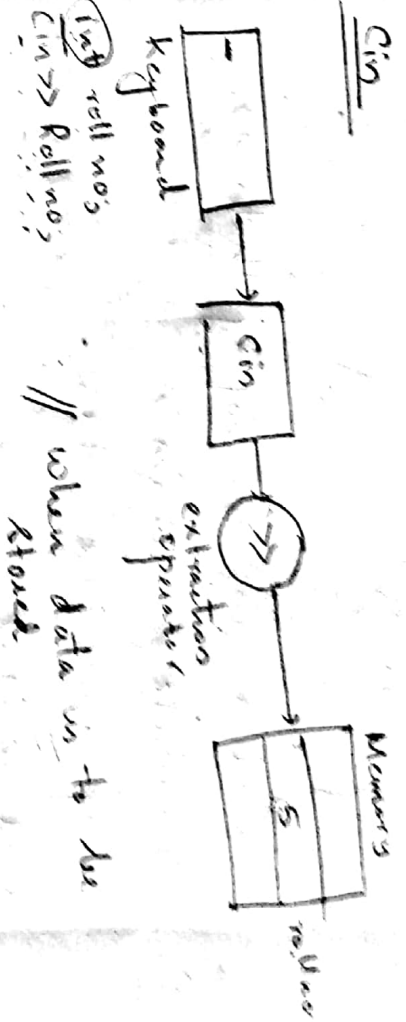
// display a message Hello world

cin - console i/p  
 cout  $\rightarrow$  console o/p  $\rightarrow$  display values from the memory



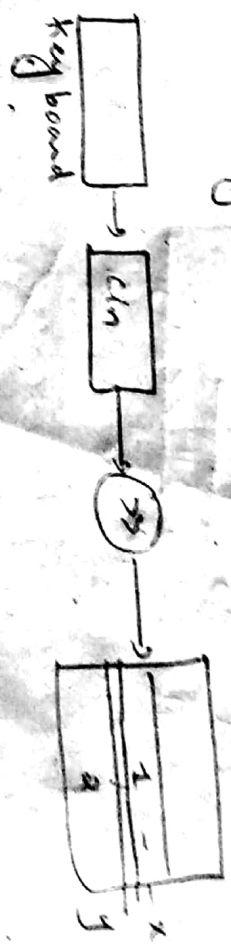
Cascading of o/p operator

cout << "Roll no is" << roll no's



Cascading of i/p operator

cin >> x >> y



Program (ANSI C++)

# include < iostream >  
 using namespace std;  
 void main (c)  
 {  
 cout << " Hello world " ;  
 }

class  $\rightarrow$  identifier  
 i/p o/p operators  
 scope in defined

3) Sum of two no:s

# include <iostream>

using namespace std;

void main ()

{ int a, b;

cout << " enter 2 no: " <<

cin >> a >> b;

cout << " sum of the two no: " <<

}

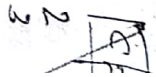
16/11/18

Class and objects

Specifying a class

- 1) class declaration - type & scope of it
- 2) class Function definition
  - how function are implemented

General form of class specification



class class-name

{ Private:

Variable declarations;  
Function declarations;

Public:

Variable declarations;  
Function declarations;

}

Note: The members of a class are collectively called class members - data members (Variables) & member function

Class members are divided into 2 sections

- Private
- Public

Private: These members are accessible only within the class.

Public: These members are accessible outside the class.

By default, members of a class are private

eg 1: class xyz

{ int x;

public:  
int z;

}

// Here the members are divided into sections - private & public x in private (by default) & z in public.

Example 2:

class Rectangle

{ int length, breadth;

public:

void getData (int l, int b);

void putData (void);

}

Syntax eg: class name object name;

eg Rectangle r1;

## Creating objects

Objects are instances of class

Syntax for creating objects

```
Classname objectname;
```

Eg: Rectangle r1;

OR

```
class Rectangle
```

```
{ int length, breadth;
```

```
public:
```

```
void getData ( int l, int b);
```

```
void putData ( void);
```

```
} r1, r2;
```

// r1, r2 are the objects of class Rectangle

## Example

```
class xyz
```

```
{ int x;
```

```
// private member
```

```
public:  
int z;
```

```
}  
=====  
xyz p;
```

```
p.z = 10;
```

altering the z variable (operator)