Object Oriented analysis and design:-

This model based on pattern of classes and objects. In which class associated with behaviors (Methods) and attributes (data Members).Such model must be satisfied following three properties.

P → Polymorphism
I → Inheritance
E → Encapsulation

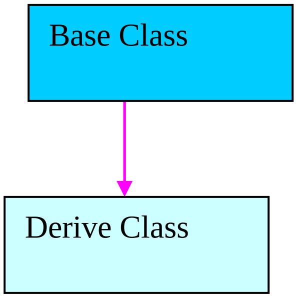
Polymorphism:-

A single function may perform various types of tasks depending upon arguments.

Example:-
draw (Line)
draw (Circle)

Inheritance:-

It means reusability. That is property of base class is going to access by its derive class.



Encapsulation:-

It is an abstraction process of binding and hiding properties. In encapsulation a set of policies (what) and mechanism (How) boths are **hide** and **bind.**
Encapsulation is the method of combining the data and functions inside a class. This hides the data from being accessed from outside a class directly, only through the functions inside the class is able to access the information.

Concept of Abstraction:-

This is also known as "Data Abstraction", as it gives a clear separation between properties of data type and the associated implementation details. There are two types, they are "function abstraction" and "data abstraction". Functions that can be used without knowing how its implemented is function abstraction. Data abstraction is using data without knowing how the data is stored.

Tasks /Functions/Characteristics of Object Oriented:-

1 Classes must be identified.
2 A class hierarchy must be specified.
3 Object to object relationships should be represented.
4 Object behavior must be modeled.
5 Modular facility for solving complex problem.
6 It provides reusability features.
7 It is robust technique for design object.

Format of class Declaration in C++:-

```
class <class_Name>
{
private :
        Data_members;
        Member_functions;
public:
        Data_members;
        Member_functions;
protected:
        Data_members;
         Member_functions;
};
```

Where :-

Private, public and protected are called **visibility mode**.

Object-oriented systems analysis make these promises:-

- Reduced maintenance.
- The primary goal of object-oriented development is the assurance that the system will enjoy a longer life while having far smaller maintenance costs Real-world modeling.
- Objects are organized into classes of objects, and, objects are associated with behaviors. The model is based on objects rather than on data and processing.
- Object-oriented systems promise to be far more reliable than traditional systems, primarily because new behaviors can be built from existing objects.
- High code reusability.
- When a new object is created, it will automatically inherit the data attributes and characteristics of the class from which it was spawned.

Object Modeling Technique (OMT):-Most Important 15 Marks

This modeling technique consist of following three models

- Object model specifies what it happens to.
- Dynamic Model specifies when it happens.
- Functional model specifies what happens.

- **Object Model.**

This model describes the structure of objects in a system, their relationships to other objects, their attributes and their operations. It provides essential framework into which the dynamic and functional models can be placed. The main objective of this model is to capture those concepts from the real world that are important to an application.

- **Dynamic Model.**

This model describes those aspects of a system concerned with time and sequence of operations. Events that mark changes, sequence of events, status that define the context of events. This model is represented graphically with state diagrams.

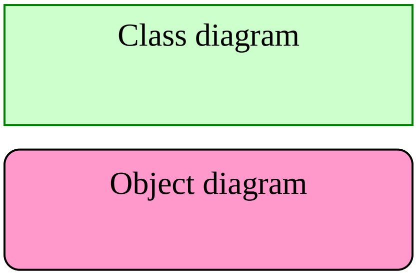
- **Functional Model.**

  This model describes those aspects of a system concerned with transformations of values--- (Functions, mapping, constraints).

Relationships among Model:-

Each model describes one aspects of the system but contains references to the other models. The object model describes data structure that the dynamic and functional models operate on.

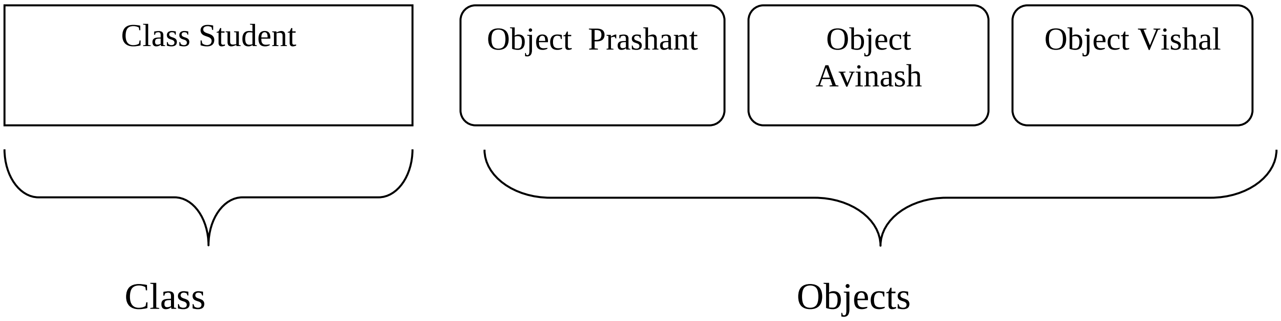
Class and Object diagram:-



Class diagram

Object diagram

Example:-



Class Student

Object Prashant

Object Avinash

Object Vishal

Class

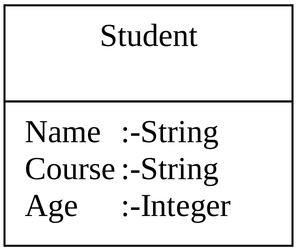
Objects

Attributes of Object:-

An attribute is data value held by the objects in a class.

Example:-

Class With attributes:-



Student

Name :-String
Course :-String
Age :-Integer

Links and Association:-

Links:-

It is a physical or conceptual connection between object instances.

Example:-

Avinash, Shiv prakash, Azmal and Raju all study in ICSM.

Association:-

It describes a group of links with common structure and common sementics.

Example:-

A person works for Infosys Company.

Object With attributes:-

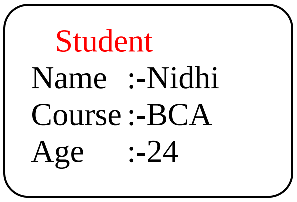
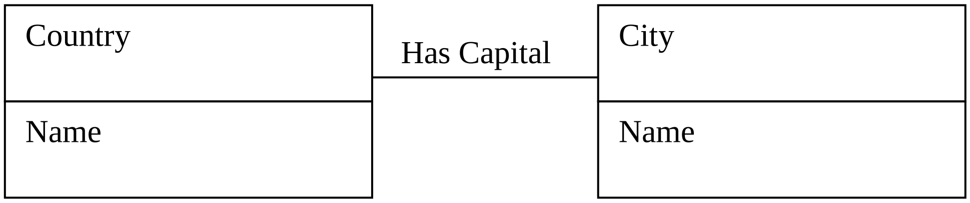


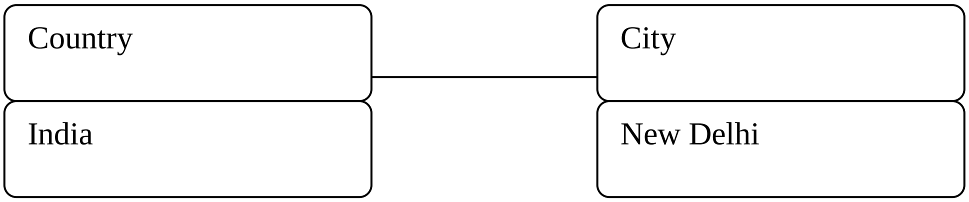
Diagram for Associations and Links:-

1 to 1 Association and links:-

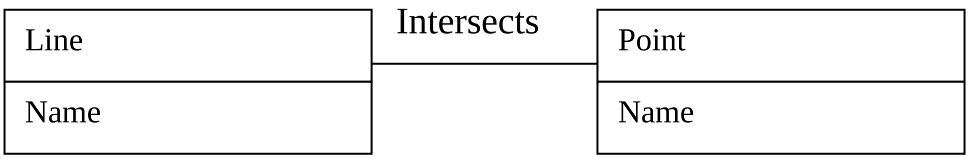
Association



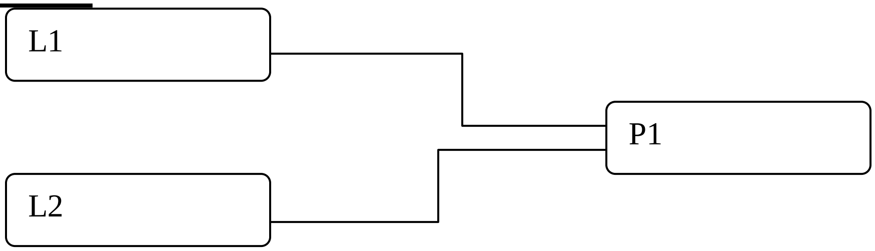
Link



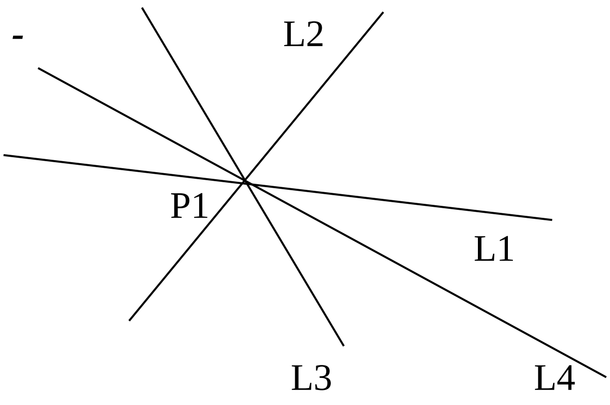
Many to many Association and links:-



Association

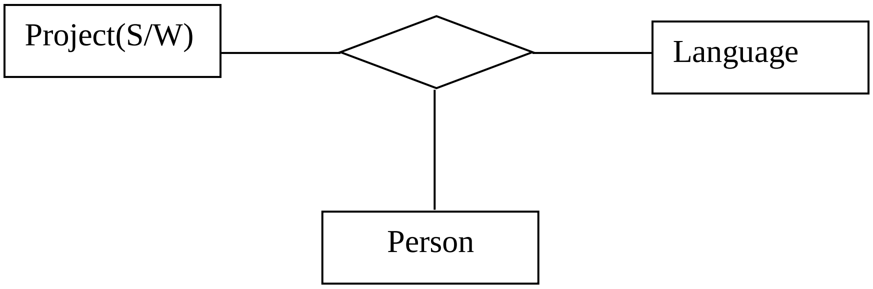


Links: -

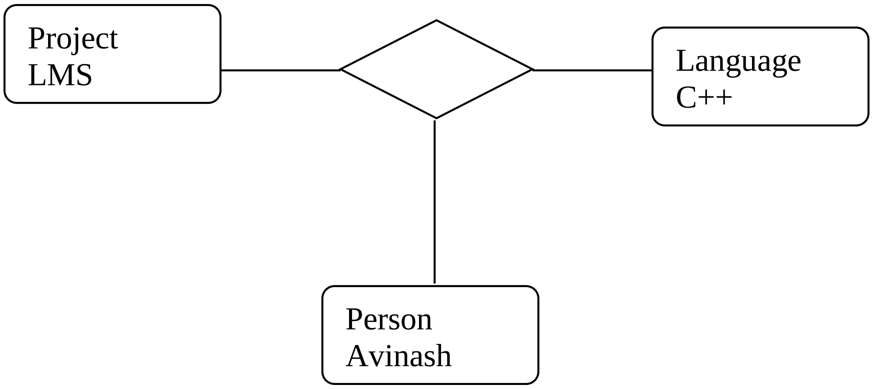


Ternary Associations and Links:-

Association:-



Links:-



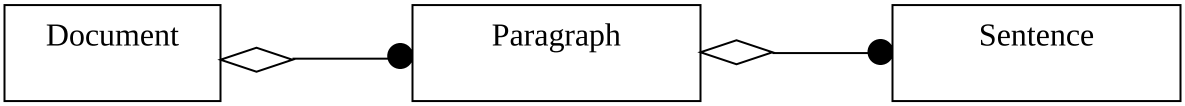
Aggregation:- **Important Questions**

It is strong form of association .The aggregate object made of components. Or It is **a part of** relationship. Aggregation relates instances. It is sometimes called an “AND relationship”

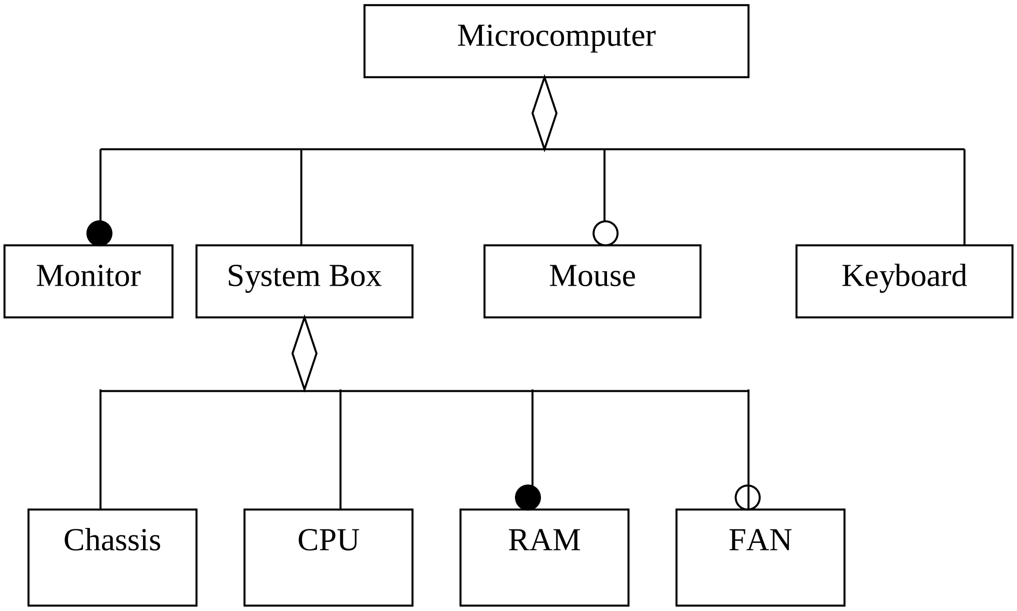
Generalization:-

It relates classes and structure description of an object.
It is sometimes called an “Or relationship”

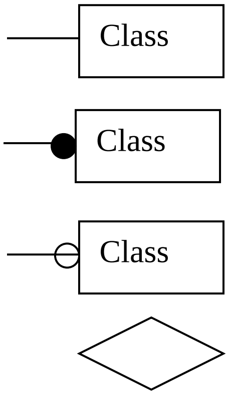
Example1 of Aggregation:-



Example 2 of Aggregation:-



Note:-



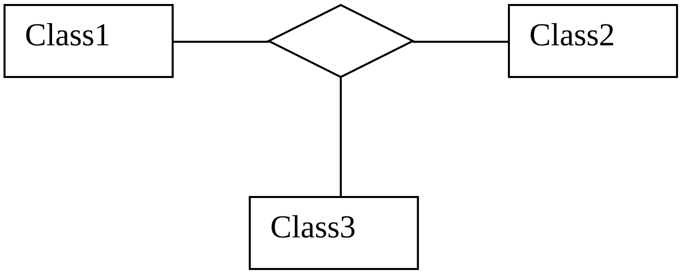
Exactly one

Many (Zero or More)

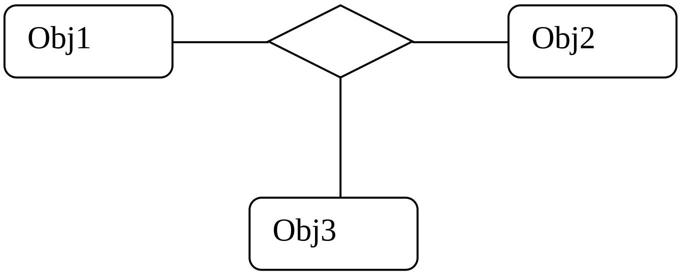
Optional (Zero or One)

Aggregation

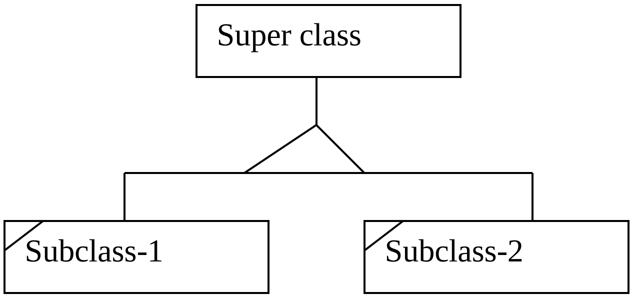
Ternary Association



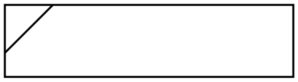
Ternary Link



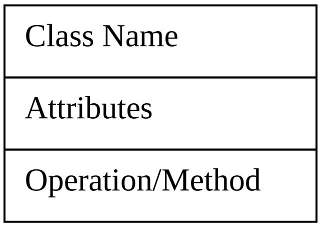
Generalization (Inheritance):-



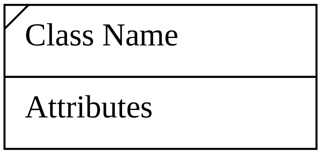
Derived Class:-



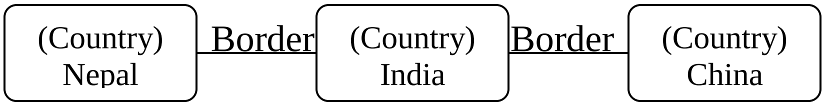
Class Attributes and Class Operation:-



Derived Attributes:-



Question:-1 Prepare a class diagram from the instance(Object).



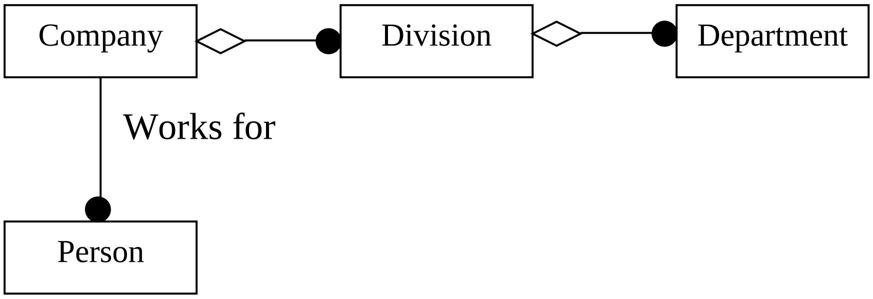
Advanced Object Modeling:- Important

Discussion about aggregation, inheritance, metadata and constraints.

Aggregation:-

It is strong form of association .The aggregate object made of components. Or It is **a part of** relationship. Aggregation relates instances. It is sometimes called an "AND relationship"

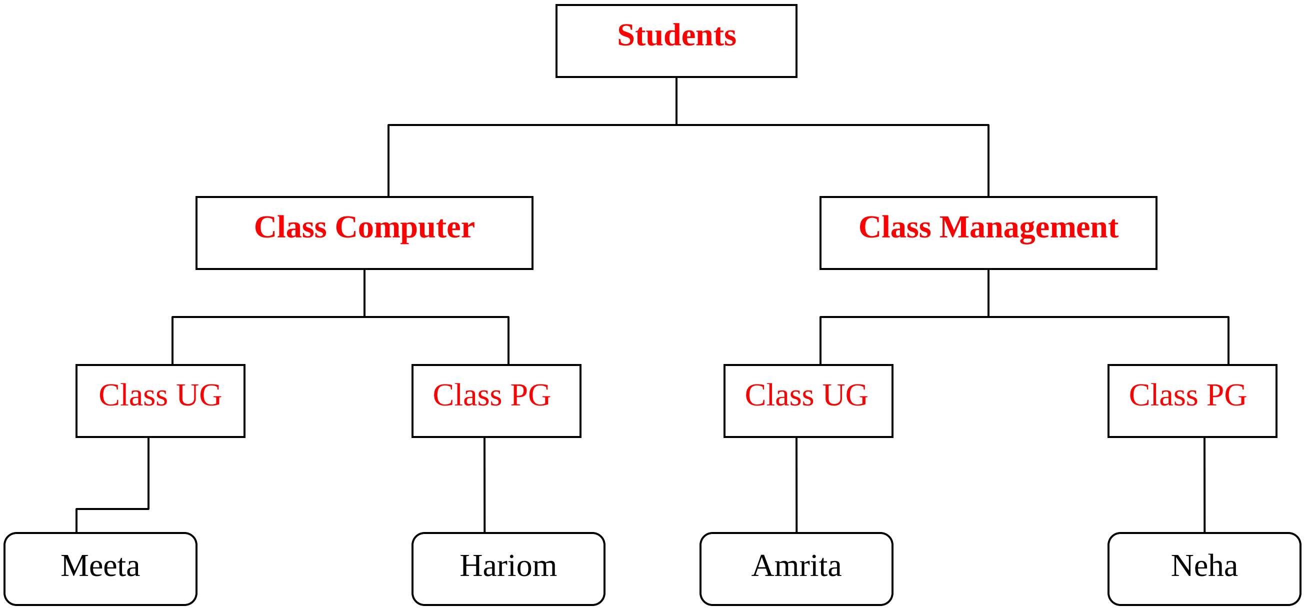
Example:-



Generalization:-

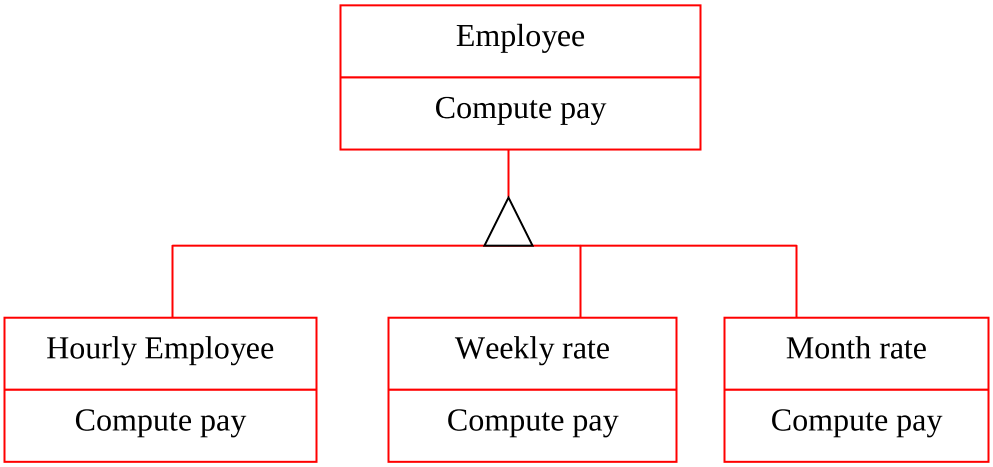
It relates classes and a way of structuring the description of a single object. With generalization, an object is simultaneously an instance of the superclass.A generalization tree is composed of classes that describe an object.

Example:-



Abstract classes:-

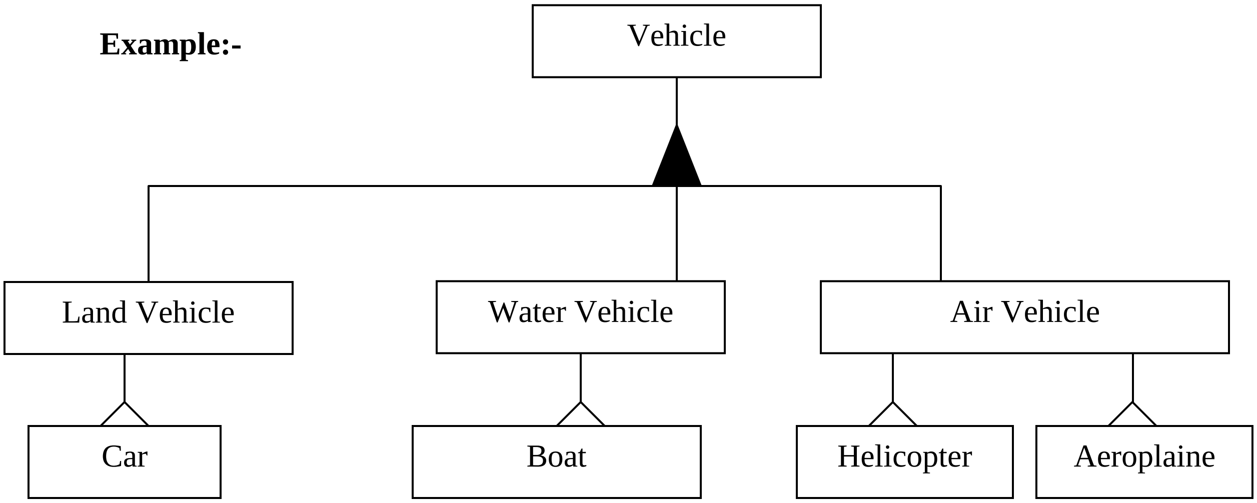
It is a class that has no direct instances but whose descendent classes have direct instances.



Multiple Inheritances:-

It permits a class to have more than one superclass and to inherit features from all parents.

Example:-



Metadata:-

Metadata is data that describes other data, the definition of a class is metadata. Models are inherently metadata, since they describe the things being models (rather than being the things).

Example:-1

Dictionary.
Parts catalogs.
Blue prints.
Etc.

Example:-2

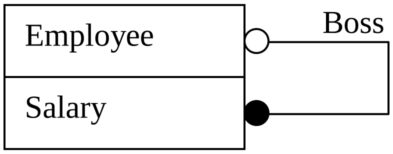
```
Class person
        {
          …
          …
          …
        }
Inherit from person
Class male
        {
        …
        …
        …
        }
Inherit from person
Class female
        {
        …
        …
        …
        }

Inherit from person
Class grandperson
        {
        …
        …
        …
        }
```

Constraints/Validation Rules:-

It is functional relationships between entities of an object model. The term entities includes objects, classes, attributes, links and associations. A constraints restricts the values that entities can assume.

Example:-

Employee includes name, father name, job, salary, department etc.



(Salary<= salary of boss)

Dynamic Model:-

It describes control structure of object modeling. In this model David Harel follow drawing structure state diagrams using nested contours to show structure. There are following two situations happen in dynamic model.

- ✓ **Events:-**An event is something that happens at a point in time, Such as a flight no 2345 departure from Babatpur to Bombay.

  BabatupurAirport (VNS) →Flight No 2345→ (State1) →Santacruz Airport (Bombay) → (State-2)

- ✓ **States:-**A state is an abstraction of the attribute values and links of an object.Sets of values grouped together into a state according to properties that affect the gross behavior of the object.

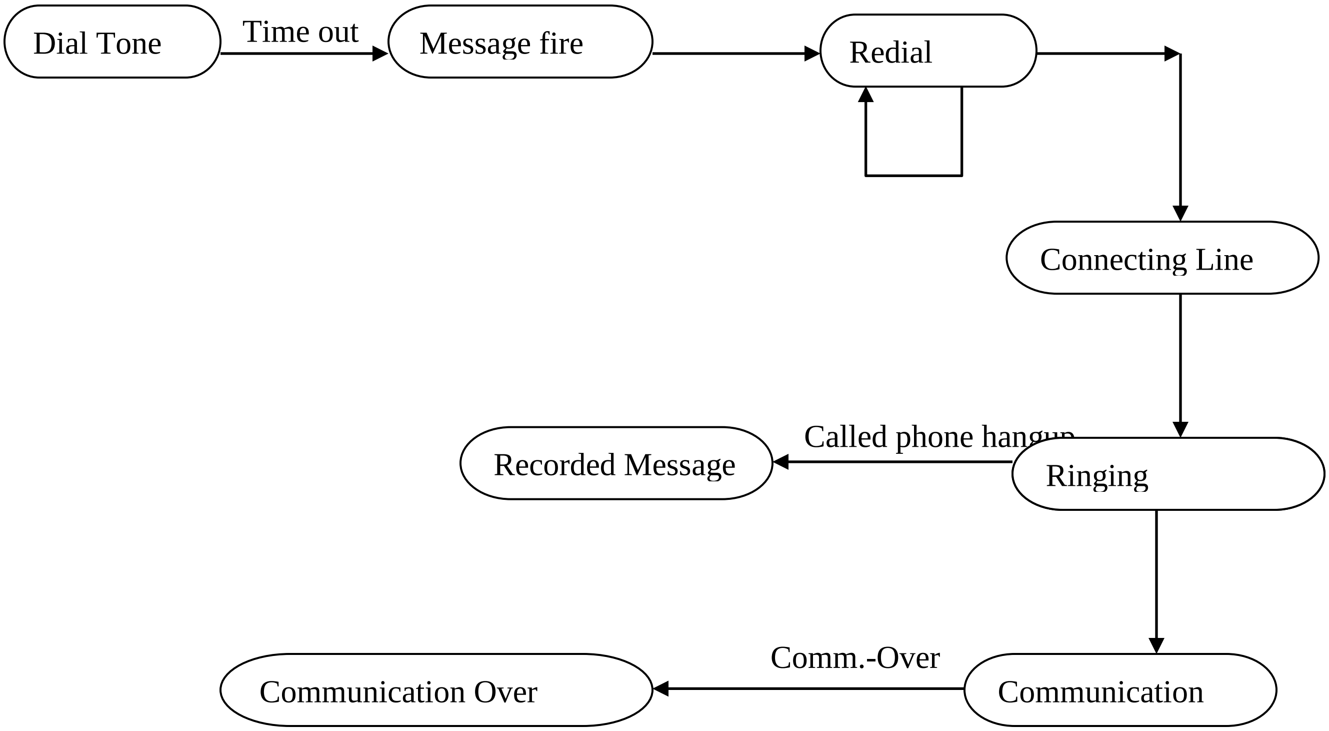
  Example:-
  BabatupurAirport (VNS) → (State-1)

  Santacruz Airport (Bombay) → (State-2)

Note:-

A state diagram relates events and states

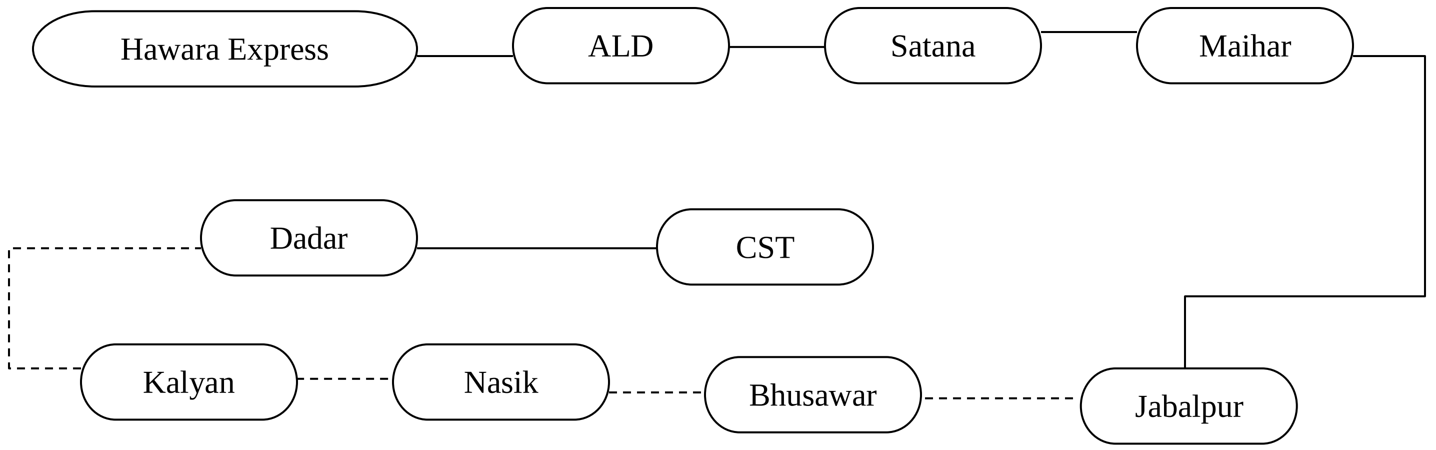
Example of state diagram of Phone line:-



Event-State Conditions:-

A condition is a Boolean function of object values Such as "Temperature is below freezing". This condition is valid over an interval of time.
Conditions can be used as guards on transitions.

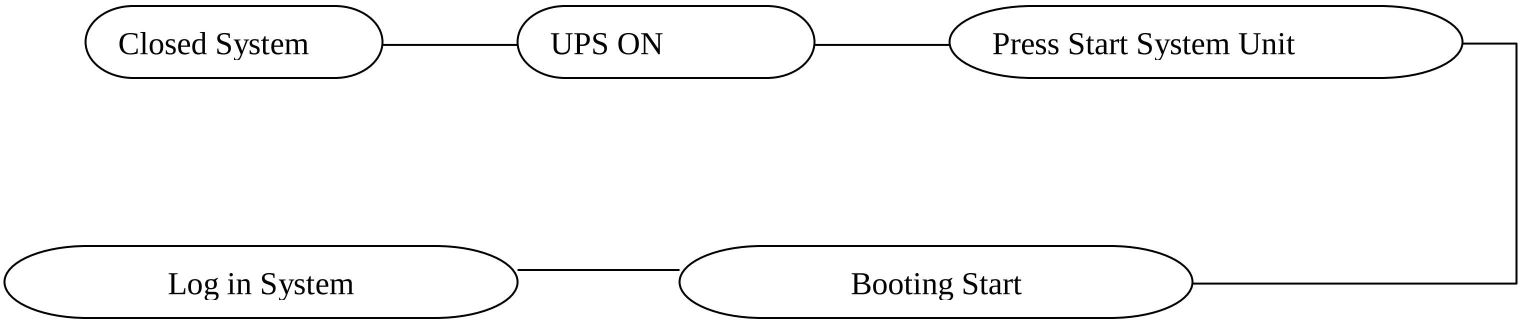
State diagram of guarded Transition:-



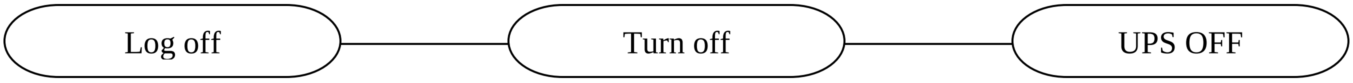
Advanced dynamic modeling concepts:-

In this model we will describe about Entry and Exit actions. Actions can be associated with entering or exiting a state.

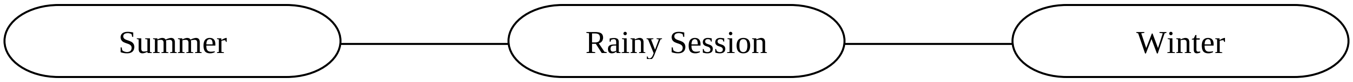
Action for Entry In PC:-



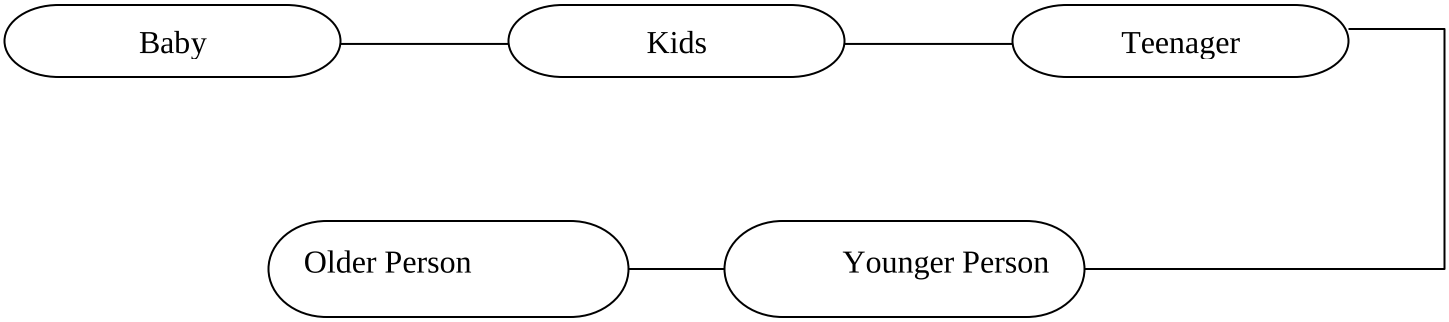
Action for Exit PC:-



Automatic transition diagram of season:-



Automatic transition diagram of person:-



Functional Modeling:-

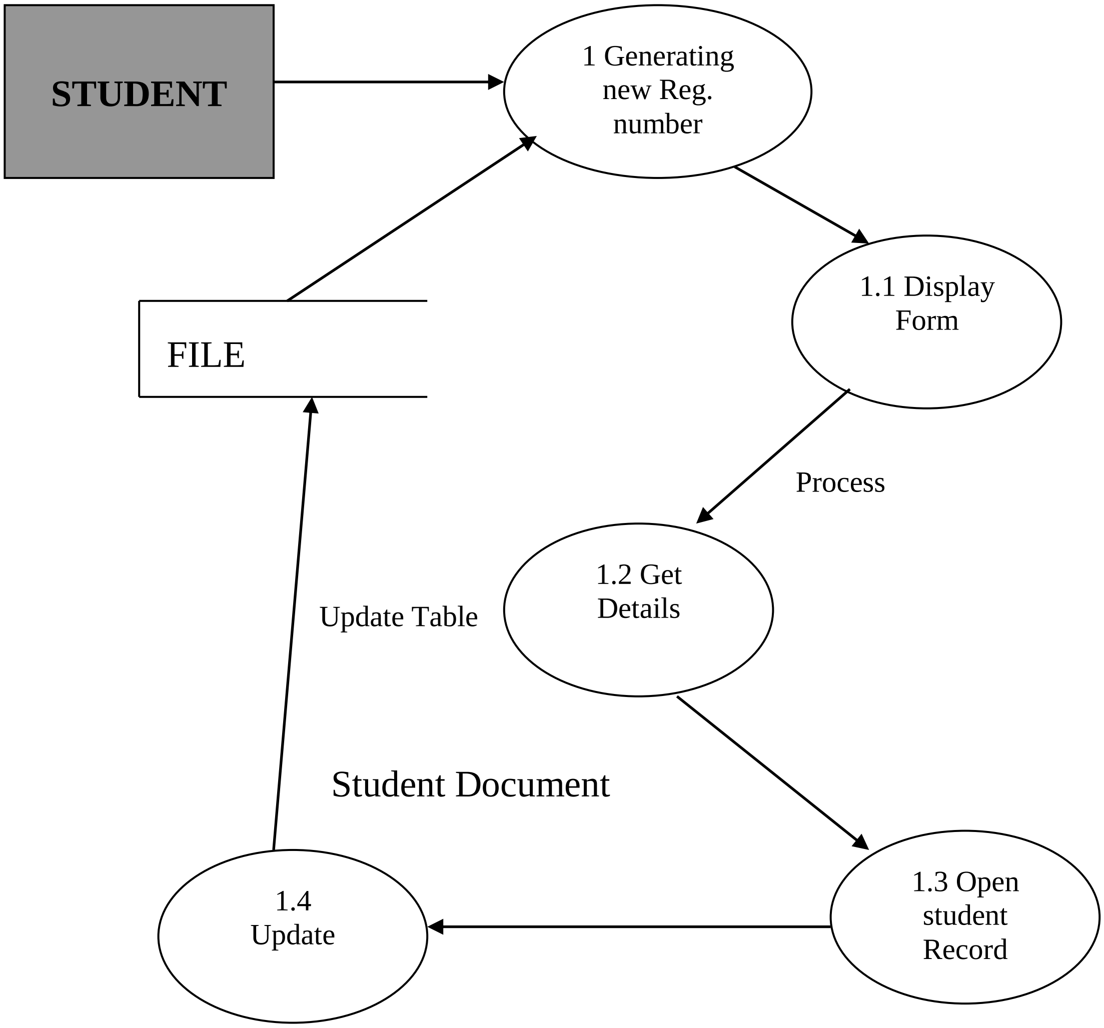
It describes computation within a system. It is third leg of modeling tripod.

Function model specifies what happens.
Dynamic Model specifies when it happens.
Object model specifies what it happens to.

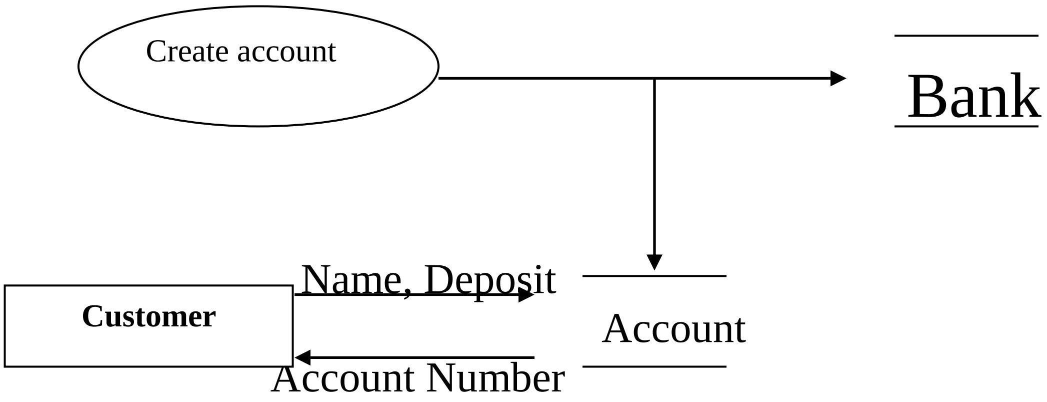
Example:-1

DFD→it shows the functional relationships of the values computed by a system. It include input values, Output values and internal data.It is a graph showing a flow of data values from source in object through process that transform them to their destinations in other objects.
DFD describe various level it may be LEVEL-0,LEVEL-1, LEVEL-2, LEVEL-3…

DATA FLOW DIAGRAM OPENING A NEW REG.NO



Example:-2 Creation of new Account:-



Methodology Preview Of OMT:-

OMT as a software engineering methodology:-

It is a process for the organized production of s/w, using a collection of predefined techniques and notational conventions. It is represented a series of steps.

OMT methodology consists of several phases:-

1. Analysis:-It concerned with understanding and modeling the application and the domain within which it operates.
2. System Design:-The overall architecture of the system is determined during system design.
3. Object design:-these models are elaborated, refined and then optimized to produce a practical design. During object design there is a shift in emphasis from application concepts towards computer concepts.
4. Summarizes the OMT methodology:-In this phase OMT methodology compare with other s/w engineering methodologies

   Note:-S/w is gradually made robust through incremental improvements to the specifications, design and implementation.

Detail of analysis phase:-

A:-Requirements statements includes following elements

- ✓ Problem scope.
- ✓ What is needed.
- ✓ Application Context
- ✓ Assumptions
- ✓ Performance needs

B:-Design & Implementation

- ✓ General approach
- ✓ Algorithms
- ✓ Data structure
- ✓ Architectures
- ✓ Optimizations(Achieving the best)

Object Modeling/Object Design

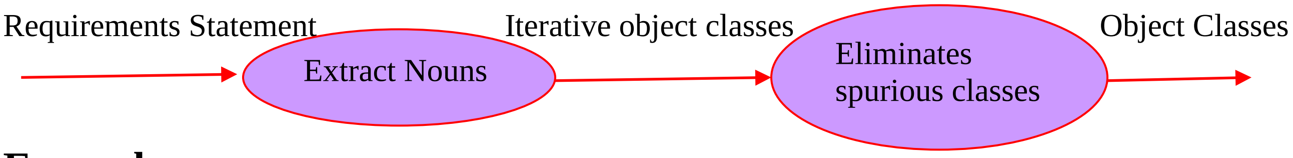
It shows the static data structure of the real world system and organizes it into workable pieces. It describes real world object classes and their relationships to each other.

Following steps are performed in constructing an object model.

- ✓ Identify object and classes
- ✓ Prepare a data dictionary
- ✓ Identify associations between objects
- ✓ Identify attributes of objects and links
- ✓ Organize and simplify object classes using inheritance
- ✓ Verify that access paths exist for likely queries

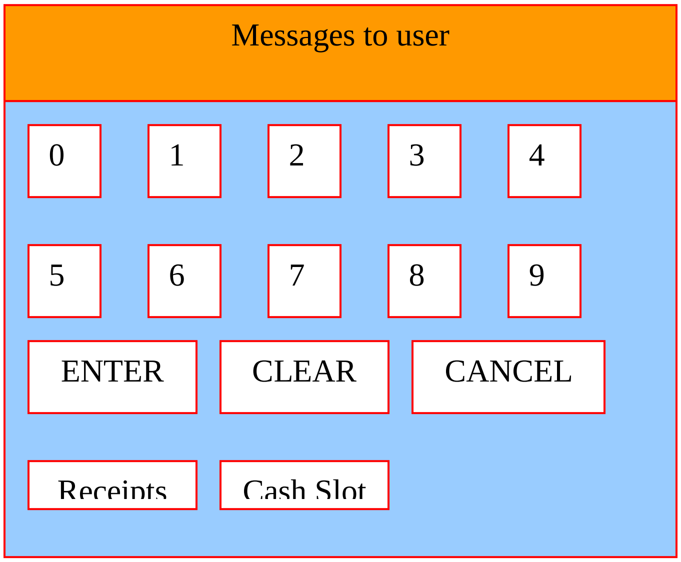
- ✓ Iterate and refine model
- ✓ Group classes into modules

Identify object and classes



Example:-

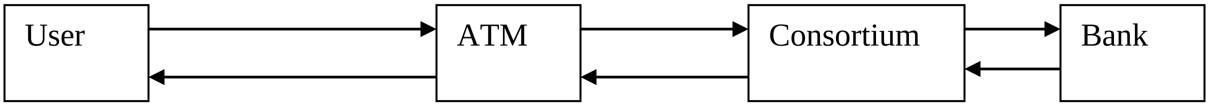
ATM Interface Formate:-



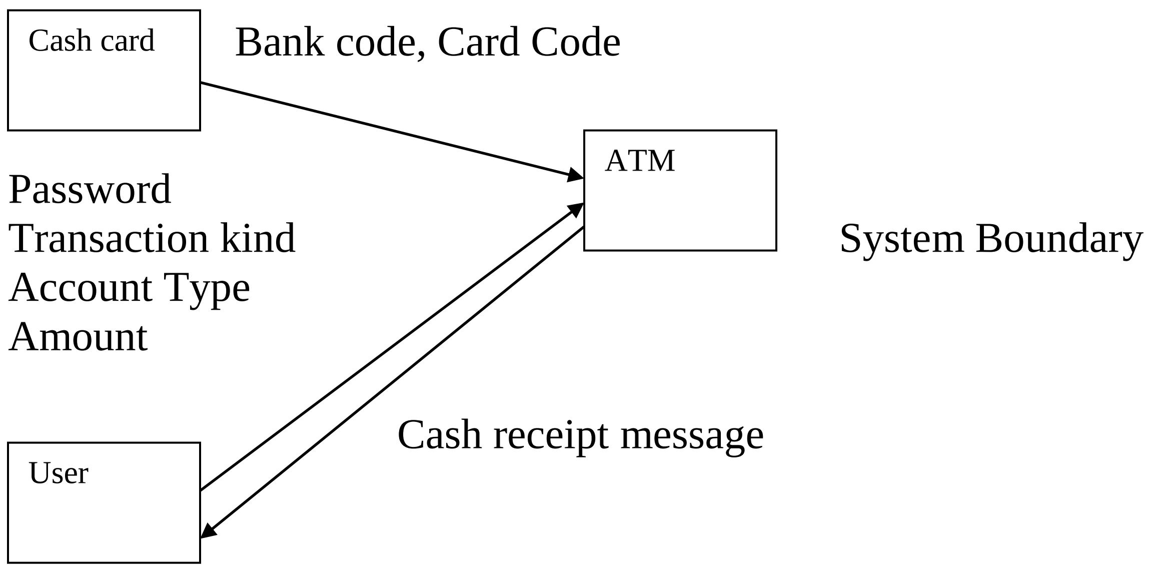
Explanation:-

Examine the scenarios to identify all external events.It includes all signals, inputs, decision, intrrupts, transition and actions to or from users or external devices. Internal computation steps are not events except for decision points that interact with external world.

Event Flow Diagram:-



Input And Output for ATM:-



Introduction of C++:-

It is Object Oriented Programming language that provide the programmer the ability to create class hierarchies, instantiate co-operative objects collectively working on a problem to produce the solution and send messages between objects to process themselves.The power of object oriented languages is that the programmer can create moduler, reusable code and as a result, formulate a program by composition and modification of the existing modules. This language is pointed out by Tim Rentsch.

Any OOPs languages must be satisfied following three necessary and essential conditions.

P → **Polymorphism.**
I → **Inheritance.**
E → **Encapsulation.**
A → **Abstraction**

What is run-time error, logical error and syntax error:-

Syntax error - The errors which are traced by the compiler during compilation, due to wrong grammar for the language used in the program, are called syntax errors.
For example, cin<<a; // instead of extraction operator insertion operator is used.
Run time Error - The errors encountered during execution of the program, due to unexpected input or output are called run-time error.
For example - a=n/0; // division by zero
Logical Error - These errors are encountered when the program does not give the desired output, due to wrong logic of the program.
For example : remainder = a+b // instead of using % operator + operator is used.

Programming elements Of C++:-

A:-Data Types:-

1. **Simple data Types**
   1.1. Integer data types
   1.2. Real Data type/Floating data Types.
   1.3. Character data types.
2. **Structured data types**
   2.1. Arrays.
   2.2. Strings.
   2.3. Structures.
   2.4. Unions.
3. **Enumerated data Types**:-It is a user define type, with values ranging over a finite set of identifiers called enumeration constants. Example:-1 enum color {red, blue, green}; Color m; The value of m exist red, blue, green. Example:-2 enum flag {true, false}; flag p; the value of p exist either true or false.
4. **Pointer Data type:-** It is a variable which point address of variable.
5. **void data Type:-**There is no any return type.

Tables of Integer data types:-

Type	Size	Minimum value	Maximum Value
short int/int	2 Byte	-32768	32767
long int	4 Byte	-2147483648	2147483647

Tables of Real data types:-

Type	Size	Minimum value	Maximum Value
float	4 Byte	$3.4*10^{-38}$	$3.4*10^{+38}$
double	8 Byte	$1.7*10^{-308}$	$1.7*10^{+308}$
long double	10 Bytes	$3.4*10^{-4932}$	$1.1*10^{+4932}$

Character data types:-

It is enclosed into single quote. ASCII range of characters exist between 0 to 255.
Example: - ‘a’, ‘6’, ‘M’, ‘ ’;

B:-Operators:-

Arithmetic Operators

+,-,*,/,%(Remainder or modulo operator)

Relational Operators

>, <,>=, <=,! = (Not equal), = = (equal).

Logical Operators

&&(And),||(Or),!(Not).

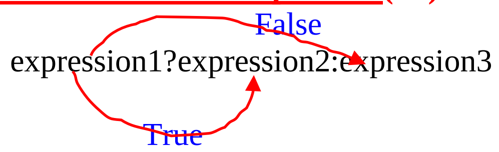
Bitwise operator

&	Bitwise and
|	Bitwise or
^	Bitwise exclusive or
~	Bitwise compliment Operator
<<	Bitwise left shift
>>	Bitwise Right shift

Increment & decrement Operator

++m	preincrement by one
m++	Post increment by one
--m	predecrement by one
m--	Post decrement by one

Ternary Operator/Conditional Operator (? :) :-



False

expression1?expression2:expression3

True

Indirection Operator (*) :-

*p →It point address of variable p.

Assignment operator:-

= example:- m=a+b

Compound assignment operators:-

▪	+=	a+=expr	Example a=a+6;	or	a+=6;
▪	-=	a-=expr	Example b=b-4;	or	b-=4;
▪	*=	a*=expr	Example b=b*4;	or	b*=4;
▪	/=	a/=expr	Example b=b/4;	or	b/=4;
▪	%=	a%=expr	Example b=b%4;	or	b%=4;
▪	&=	a&=expr			
▪	^=	a^=expr			
▪	<<=	a<<=expr			
▪	>>=	a>=expr			

How to write program in C++:-

Example Based on above Operators:-

Example 1:-Bitwise operator

```
#include<iostream.h>
void main()
{
int a=5,b=6,c1,c2,c3,c4,c5,c6;
c1=a&b;
c2=a|b;
c3=a^b;
c4=~a;
c5=a<<2;
c6=b>>2;
cout<<"\nValue of c1="<<c1<<endl;          //cout→Console out
cout<<"\nValue of c2="<<c2<<endl;
cout<<"\nValue of c3="<<c3<<endl;
cout<<"\nValue of c4="<<c4<<endl;
cout<<"\nValue of c5="<<c5<<endl;
cout<<"\nValue of c6="<<c6<<endl;
}
```

Example 2:-Arithmetic compound expression

```
#include<iostream.h>
void main()
{
int a=5,b=6;
cout<<"\nValue of a="<<a<<endl;
a+=7;
cout<<"\nValue of a="<<a<<endl;
cout<<"\nValue of b="<<b<<endl;
b*=7;
cout<<"\nValue of b="<<b<<endl;

}
```

Example 3:-Increment and decrement

```
#include<iostream.h>
void main()
{
int a=5,b=6;
cout<<"\nValue of a="<<a<<endl;
cout<<"\nValue of b="<<b<<endl;
a=++a + b++;
cout<<"\nValue of a="<<a<<endl;
cout<<"\nValue of b="<<b<<endl;
b=--a + b--;
cout<<"\nValue of a="<<a<<endl;
cout<<"\nValue of b="<<b<<endl;
}
```

Example 4:-Ternary operator

```
#include<iostream.h>
void main()
{
 int a,b,c1;
cout<<"\nValue of a="<<endl;     //>> get operator or extraction
cin>>a;                          //console input
cout<<"\nValue of b="<<endl;     //<< insertion or put to operator
cin>>b;
c1=a>b?a*a:b*b;
cout<<"\nC1="<<c1<<endl;
}
```

Example 5:-sizeof() operator

```
#include<iostream.h>
void main()
{
```

```
cout<<"\nSize of int="<<sizeof(int)<<endl;
cout<<"\nSize of float="<<sizeof(float)<<endl;
cout<<"\nSize of double="<<sizeof(double)<<endl;
cout<<"\nSize of char="<<sizeof(char)<<endl;
cout<<"\nSize of long int="<<sizeof(long int)<<endl;
cout<<"\nSize of long double="<<sizeof(long double)<<endl;
}
```

Example6:-Conversion of Celsius into foreignheight

```
# include<iostram.h>
main()
{
float centigrade,foreingn_height;
cout<<”\nEnter Tempereature in Centigrate=”<<endl;
cin>>centigrade;
foreingn_height=(9* centigrade)/5.0 +32;
cout<<”\nTempereature in Foreignheight=”<<foreingn_height<<endl;
}
```

Control Statement:-

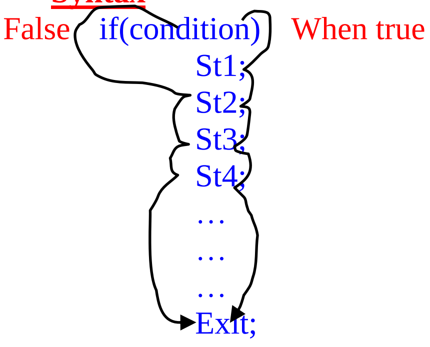
‘C++’ language provides facilities for controlling the order of execution of the statements, which is referred to as flow control statements/control statements.

There are following three categories of flow control statements.

- Decision Control Statements
  - if statement
  - if-else statement
  - nested if-else statement
  - else –if construct statement
  - switch case statement
- Looping Control Statement
  - while loop
  - do-while loop
  - for loop
- Jumping Control Statement
  - goto
  - break
  - continue

if statement:-

Syntax



```
False  if(condition)   When true
             St1;
             St2;
             St3;
             St4;
             …
             …
             …
             Exit;
```

Example7:- (To clear concept of if control statement)

```
# include<iostream.h>
main()
{
int n;
cout<<"\nEnter value of ="<<endl;
cin>>n;
if(n>=10)
cout<<"\nStatement1"<<endl;
cout<<"\nStatement2"<<endl;
cout<<"\nStatement3"<<endl;
cout<<"\nStatement4"<<endl;
cout<<"\nExit_Statement"<<endl;
}
```

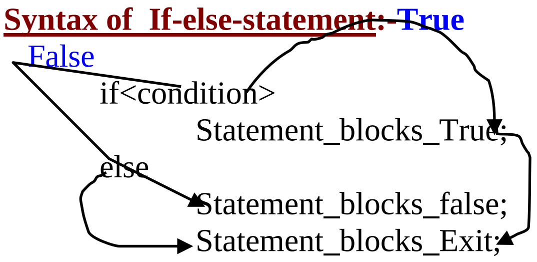
Example8:- (To check year is leap or not leap year)

```
# include<iostream.h>
main()
{
int n,year;
cout<<"\nEnter value of Year="<<endl;
cin>>year;
n=year%4;
if(n==0)
cout<<"\nLeap Year"<<endl;
  if(n!=0)
cout<<"\nNot Leap Year"<<endl;
}
```

Example9:- (To check profit ,loss or none)

```
        #include<iostream.h>
        main()
        {
        float sale,purchase,m;
        cout<<"\nEnter Purchase cost="<<endl;
        cin>>purchase;
        cout<<"\nEnter sale cost="<<endl;
        cin>>sale;
        m= sale- purchase;
        if(m==0)
        cout<<"\neither Profit nor Loss"<<endl;
        if(m>0)
        cout<<"\n Profit="<<m<<endl;
        if(m<0)
        cout<<"\n Loss="<<m<<endl;
        }
```

Syntax of If-else-statement:-



```
            if<condition>
                        Statement_blocks_True;
            else
                        Statement_blocks_false;
                        Statement_blocks_Exit;
```

Example10:- (Program of leap year by using if-else statement)

```
# include<iostream.h>
main()
{
int n,year;
cout<<"\nEnter value of Year="<<endl;
cin>>year;
n=year%4;
if(n==0)
cout<<"\nLeap Year"<<endl;
else
cout<<"\nNot Leap Year"<<endl;
}
```

Syntax of Nested If-else-statement:-

```
if<condition1>
            Statement_blocks1;
    else
            if<condition2>
               Statement_blocks2;
             else
                 if<condition3>
                        Statement_blocks3;
                else
                        if<condition4>
                         Statement_blocks4;
                         …
                         …
                        …
                        else
                                  Statement_blocks_Exit;
```

Example11:- (Program of largest of three numbers)

```
#include<iostream.h>
#include<conio.h>
main()
{
 int a,b,c;
 cout<<"\nEnter First number="<<endl;
 cin>>a;
 cout<<"\nEnter Second number="<<endl;
 cin>>b;
 cout<<"\nEnter Third number="<<endl;
 cin>>c;
cout<<"A="<<a<<"\t"<<"B="<<b<<"\t"<<"C="<<c<<endl;
if(a==b && b==c )
```

```
 cout<<"\nAll Numbers are equal"<<endl;
 else
if(a>b && b>c|| a>c && c>b )
cout<<"\nA is the largest numbers"<<endl;
 else
 if(b>c && c>a|| b>a && a>c )
 cout<<"\nB is the largest numbers"<<endl;
 else
 if(a==b && a>c )
 cout<<"\nA and B Both larger than C"<<endl;
else
 if(a==b && a<c )
 cout<<"\nA and B Both Smaller than C"<<endl;
else
 if(a==c && a>b )
 cout<<"\nA and C Both Larger than B"<<endl;
else
 if(a==c && a<b )
 cout<<"\nA and C Both Smaller than B"<<endl;
else
 if(b==c && a<b )
 cout<<"\nB and C Both Smaller than A"<<endl;
else
 if(b==c && a>b )
 cout<<"\nB and C Both Larger than A"<<endl;
else
  cout<<"\n"<<"C is the Largest Number"<<endl;
}
```

Example13:-(Grading System)

```
#include<iostream.h>
#include<math.h>
main()
{
 int h,e,m,p,c,tot;
 float per;
 cout<<"\nEnter Marks Obtained In Hindi="<<endl;
 cin>>h;
 cout<<"\nEnter Marks Obtained In English="<<endl;
 cin>>e;
  cout<<"\nEnter Marks Obtained In Maths="<<endl;
 cin>>m;
  cout<<"\nEnter Marks Obtained In Physics="<<endl;
 cin>>p;
 cout<<"\nEnter Marks Obtained In Chemistry="<<endl;
 cin>>c;
cout<<"\n--------------------------------------------------------------------------------"<<endl;

cout<<"\n\nHindi="<<h<<"tEnglish="<<e<<"\tMaths="<<m<<"\t"<<"\tPhysics"<<p<<"\tChemistry="
<<c<<endl;
cout<<"\n--------------------------------------------------------------------------------"<<endl;
 tot=h+e+m+p+c;
 per=tot/5.0;
 if(per>=85 &&per<=100 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
            cout<<"\n\n"<<"A Grade & Passed"<<"\t Tot="<<tot<<"Per="<<per<<endl;
            else
```

```
                if(per>=75 &&per<=100 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
                cout<<"\n\n"<<"B Grade & Passed"<<"\t Tot="<<tot<<"Per="<<per<<endl;
                 else
                if(per>=65 &&per<=100 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
                  cout<<"\n\n"<<"C Grade & Passed"<<"\t Tot="<<tot<<"Per="<<per<<endl;
                 else
                if(per>=45 &&per<=100 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
                cout<<"\n\n"<<"D Grade & Passed"<<"\t Tot="<<tot<<"Per="<<per<<endl;
                 else
                if(per>=33 &&per<=100 && h>=33 && e>=33 && m>=33 && p>=33 &&c>=33)
                cout<<"\n\n"<<"E Grade & Passed"<<"\t Tot="<<tot<<"Per="<<per<<endl;
                 else

                cout<<"\n\n"<<"F Grade & Failed"<<"\t Tot="<<tot<<"Per="<<per<<endl;
        cout<<"\n----------------------------------------------------------------------------------------"<<endl;
}
```

Syntax of -else-if-Construct Statement/Ladder Statement:-

It is used for making solving of choice based problem.

```
        …
        …
        …
   else
        if<condition>
                Statement_Blocks
                …
                …
                …
```

Example14:-

```
#include<iostream.h>
#include<math.h>
main()
{
int ch,n,year;
float a,b,s1,p1,d1,sub;
cout<<"\n\n\n\t\t---------------------------------------------"<<endl;
cout<<"\n\t\tChoice Based Arithmetic,Year & Number Checking\n"<<endl;
cout<<"\n\t\t---------------------------------------------"<<endl;
cout<<"\n\t\t\t1:-ADDITION"<<endl;
cout<<"\n\t\t\t2:-SUBSTRACTION"<<endl;
cout<<"\n\t\t\t3:-PRODUCT"<<endl;
cout<<"\n\t\t\t4:-DIVISION"<<endl;
cout<<"\n\t\t\t5:-YEAR CHECKING"<<endl;
cout<<"\n\t\t\t6:-NUMBER CHECKING"<<endl;
cout<<"\n\t\t---------------------------------------------"<<endl;
cout<<"\n\t\t---------------------------------------------"<<endl;
cout<<"\n\t\tENTER YOUR CHOICE="<<endl;
cin>>ch;
cout<<"\n\t\t---------------------------------------------"<<endl;
cout<<"\n\t\tENTER ANY TWO NUMBERS FOR ARITHMETIC OPERATIONS="<<endl;
cin>>a>>b;
cout<<"\n\t\t---------------------------------------------"<<endl;
cout<<"\n\t\t---------------------------------------------"<<endl;
if(ch==1)
{
s1=a+b;
cout<<"\n\t\tSUM="<<s1<<endl;
}
else
if(ch==2)
{
sub=a-b;
cout<<"\n\t\tDIFFERENCE="<<sub<<endl;
}
else
if(ch==3)
{
  p1=a*b;
  cout<<"\n\t\tPRODUCT="<<p1<<endl;
}
else
if(ch==4)
{
d1=a/b;
cout<<"\n\t\tDIVISION="<<d1<<endl;
}
else
if(ch==5)
{
cout<<"\n\t\tENTER YEAR="<<endl;
cin>>year;
if(year%4==0)
cout<<"\n\t\tLEAP YEAR="<<year<<endl;
else
cout<<"\n\t\tNOT LEAP YEAR="<<year<<endl;
}
else
if(ch==6)
{
cout<<"\n\t\tENTER NUMBER="<<endl;
cin>>n;
if(n%2==0)
cout<<"\n\t\tEVEN NUMBER="<<n<<endl;
else
cout<<"\n\t\tODD NUMBER="<<n<<endl;
}
else
cout<<"\n\t\tWRONG CHOICE AGAIN ENTER...!"<<endl;
}
```

Syntax switch-case statement :-

It is also used for solving choice based problems. It is an alternative of else-if construct statement.

Syntax:-

```
        switch(expression)
        {
        case <value1>:
        Statement_Blocks_1;
        break;
        case <value2>:
        Statement_Blocks_2;
```

```
            break;
            case <value3>:
            Statement_Blocks_3;
            break;
            …
            …
            …
            default:
            Statement_Blocks_False;
            }
            Exit_Statement;
```

Example15:-

```
#include<iostream.h>
#include<math.h>
main()
{
int day_code;

cout<<"\n\n\n\t\t-----------------------------------------------"<<endl;
cout<<"\n\t\tCONVERSION OF DAY CODE INTO DAY"<<endl;
cout<<"\n\t\t-----------------------------------------------"<<endl;
cout<<"\n\t\tENTER DAY CODE="<<endl;
cin>>day_code;
 switch(day_code)
 {
 case 1:
            cout<<"\n\n\t\tSUNDAY"<<endl;
            break;
 case 2:
            cout<<"\n\n\t\tMONDAY"<<endl;
             break;
 case 3:
            cout<<"\n\n\t\tTUESDAY"<<endl;
             break;
 case 4:
             cout<<"\n\n\t\tWEDNESDAY"<<endl;
             break;
 case 5:
            cout<<"\n\n\t\tTHURSDAY"<<endl;
             break;
 case 6:
            cout<<"\n\n\t\tFRIDAY"<<endl;
             break;
 case 7:
            cout<<"\n\n\t\tSATURDAY"<<endl;
             break;
 default :
             cout<<"\n\n\t\t\tWrong day Code"<<endl;
 }
}
```

Looping Control Statement/Iteration Statements/Repetition Statement:-

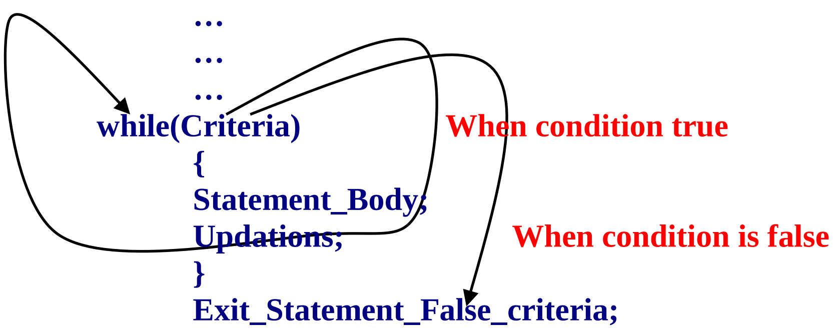
It allows the execution of some set of statements repeatedly till either for a known number of times or till certain conditions are met. There are following three types of looping statements.

- while loop
- do-while loop
- for loop

while loop:-

It executes looping body when condition is true.

Syntax of While loop:-



```
…
…
…
while(Criteria)
{
Statement_Body;
Updations;
}
Exit_Statement_False_criteria;
```

Example16:-

```
#include<iostream.h>
#include<conio.h>
void main()
{
int n,i=1;
clrscr();
cout<<"\n\n\nEnter Any Positive integer="<<endl;
cin>>n;
while(i<=n)
{
cout<<i<<"\t";
i++;
}
cout<<"\n\nThank You"<<endl;
getch();
}
```

Example17 (Series of even and Odd):-

```
#include<iostream.h>
#include<conio.h>
void main()
{
int n,i=1,j=2;
clrscr();
cout<<"\n\n\nEnter Any Positive integer="<<endl;
cin>>n;
cout<<"\nSeries of Even Numbers=\n"<<endl;
while(j<=n)
{
```

```
cout<<j<<"\t";
j=j+2;
}
cout<<"\nSeries of Odd Numbers=\n";
while(i<=n)
{
cout<<i<<"\t";
i=i+2;
}
cout<<"\n\nThank You"<<endl;
getch();
}
```

Example18 (Power of n^n):-

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
void main()
{
long int n,i=1,m1;
clrscr();
cout<<"\n\n\nEnter Any Positive integer="<<endl;
cin>>n;
cout<<"\nSeries of Power of Same Number"<<endl;
while(i<=n)
{
m1=pow(i,i);
cout<<i<<"\t"<<"="<<m1<<endl;
i++;
}
getch();
}
```

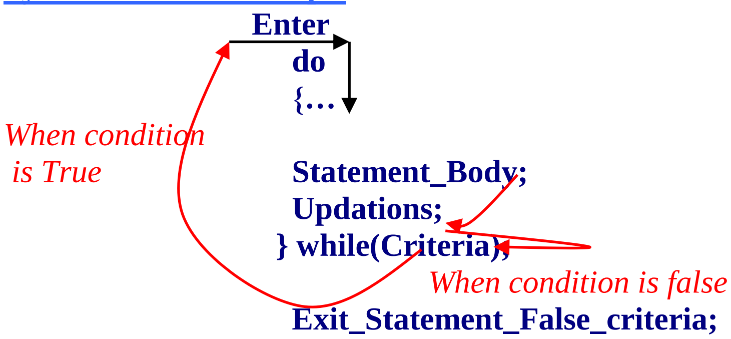
Example19:-Series of factorial numbers

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
main()
{
long int n,i=1,fact=1;
clrscr();
cout<<"\n\n\nEnter Any Positive integer ="<<endl;
cin>>n;
while(i<=n)
{
fact=fact*i;
cout<<"\nFactorial of Number="<<i<<"="<<"\t"<<fact<<endl;
i++;
}
getch();
}
```

do-while loop:-

It executes looping body at least one time when condition is false. and further execute when condition is true.

Syntax of do-while loop:-



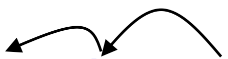
```
       Enter
          do
          {…

          Statement_Body;
          Updations;
        } while(Criteria);

          Exit_Statement_False_criteria;
```

Example 20:-

Example Generating a Series of Fibonacci Numbers :-

0 1 1 2 3 5 8 13 21 34 55 …

Logic:-



a	b	s=1+b	
a	b	s=a+b	
0	1	0	Let
1	0	1	
0	1	1	
1	1	2	
1	2	3	
2	3	5	
3	5	8	
5	8	13	
8	13	21	
.	.	.	
.	.	.	

```
#include<iostream.h>
#include<conio.h>
void main()
{
int i=0,a=0,b=1,s=0,n;
clrscr();
cout<<"\n\nEnter Value of n="<<endl;
cin>>n;
do
{
 cout<<s<<"\t";
 a=b;
 b=s;
 s=a+b;
 i++;
 }
while(i<=n);
 getch();
}
```

Program21:-

Example Generating a Series of Armstrong Numbers :-

1 153 370 371 407 . . .

Logic:-

$1^3=1$

$153=1^3+5^3+3^3=153$

$370=3^3+7^3+0^3=370$

$371=3^3+7^3+1^3=371$

$370=3^3+7^3+0^3=370$

$407=4^3+0^3+7^3=407$

…

…

…

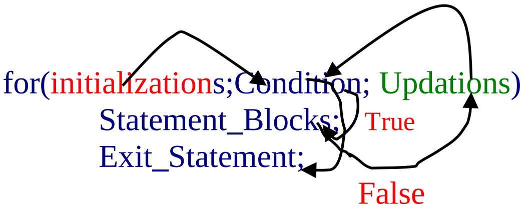
Program22:-

```
#include<iostream.h>
#include<conio.h>
main()
{
 int n=0,m=0,d=0,s=0,a=1,k;
clrscr();
cout<<"\n\nEnter Value of k="<<endl;
 cin>>k;
 while(a<=k)
 {
n=a;
s=0;
while(n>0)
{
m=n%10;
n=n/10;
s=s+m*m*m;
}
if(s==a)
cout<<"Series of Armstrong Numbers="<<a<<endl;
a++;
}
getch();
}
```

Definition of for Loop:-

A loop construct found in many procedural languages which repeatedly executes some instructions while a condition is true. The for loop is an alternative way of writing a while loop that is convenient because the loop control logic is collected in a single place.

Syntax of for loop:-



```
for(initializations;Condition; Updations)
        Statement_Blocks;  True
        Exit_Statement;
                          False
```

Initializations :-There are many initializations by using commas.

Updations :-There are many Updations by using commas.

Condition :-Only one Condition will be defined.

Semicolan :-There are two semicolons must be inside for loop.

Program23:-

```
#include<iostream.h>
#include<conio.h>
void main()
{
 int n,i,s;
clrscr();
 cout<<"\nEnter Value of n="<<endl;
 cin>>n;
 for(i=1,s=0;i<=n;i++)
 {
cout<<i<<"\t";
 s=s+i;
 }
 cout<<"\n\nSum="<<s<<endl;
 getch();
}
```

Program24:-

```
#include<iostream.h>
#include<conio.h>
void main()
{
 int n,i,s,a,b;
clrscr();
 cout<<"\nEnter Value of n="<<endl;
 cin>>n;
 for(i=1,a=0,b=1,s=0;i<=n;a=b,b=s,s=a+b,i++)
  cout<<s<<"\t";
 getch();
}
```

Program25:- (Factorial Series)

```
#include<iostream.h>
#include<conio.h>
void main()
{
 int n,i,fact;
clrscr();
 cout<<"\nEnter Value of n="<<endl;
 cin>>n;
 for(i=1,fact=1;i<=n;i++, fact=fact*i)
  cout<<"Factorial Of Value="<<i<<"\t="<<fact<<endl;
getch();
}
```

Definition of nested for loop:-

A *nested loop* is a loop within a loop, an inner loop within the body of an outer one. How this works is that the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again. This repeats until the outer loop finishes. Of course, a *break* within either the inner or outer loop would interrupt this process.

Syntax of nested for loop:-

```
    for(initializations1;Condition1; Updations1)
     {
            ….
            ….
        for(initializations2;Condition2; Updations2)
        {
                ….
                ….
            for(initializations3;Condition3; Updations3)
            {
                    ….
                    ….
                for(initializations4;Condition4; Updations4)
                    {
                        ….
                        ….
                        ….


                    }
            }
        }
    }
```

Program26:- (Pattern based Program)

```
#include<iostream.h>
#include<conio.h>
main()
{
int i,j,n;
clrscr();
cout<<"\nEnter Value of n=\n\n\n"<<endl;
cin>>n;
for(i=1;i<=n;i++)
{
for(j=i;j<=n;j++)
cout<<i;
cout<<"\n";
}
for(i=1;i<=n;i++)
{
for(j=i;j<=n;j++)
cout<<j;
cout<<"\n";
}
for(i=1;i<=n;i++)
{
for(j=i;j<=n;j++)
cout<<"*";
cout<<"\n";
}
getch();
}
```

Program 27(Series of Prime Numbers=”):-

```
#include<iostream.h>
#include<conio.h>
main()
{
int i,j,n,f;
clrscr();
cout<<"\nEnter Value of n=\n\n\n"<<endl;
cin>>n;
cout<<"\nSeries of Prime Numbers\n\n\n"<<endl;
for(i=2;i<n;i++)
{
 for(f=0,j=2;j<i;j++)
 {
  if(i%j==0)
  {
            f=1;
   break;
            }
            }
            if(f==0)
            {
             cout<<"\t"<<i;
            }
 }
getch();
}
```

Program28(Series of ASCII=”):-

```
#include<iostream.h>
#include<conio.h>
 #include<string.h>
main()
{
int i;
clrscr();
cout<<"\nSeries of ASCII Codes\n"<<endl;
for(i=0;i<=255;i++)
cout<<i<<"\t"<<"="<<char(i)<<endl;
getch();
}
```

The break statement

break leaves a loop, even if the condition for its end is not fulfilled. It can be used to end an infinite loop, or to force it to end before its natural end.

Syntax:-   break;

Example:-

```
            #include <iostream.h>
void main ()
            {
            int n;
            cout<<"\nEnter Value of n"<<endl;
```

```
        cin>>n;
        for (; n>0; n--)
         {
          cout << n << "\t ";
         if (n==5)
         {
        cout << "\n\n\nThank You Koamal-Chanda-Ankita!";
         break;
        }
         }
        cout << "\n\n\nWhen Condition is false!";
        }
```

Program 29 (Program for checking for prime or not):-

```
#include<iostream.h>
#include<conio.h>
        main()
        {
        int n,i;
        clrscr();
        cout<<"\nEnter any number="<<endl;
        cin>>n;
         for(i=2;i<n;i++)
         {
          if(n%i==0)
          {
          cout<<"\nNot prime"<<endl;
          break;
          }
          }
          if(i==n)
         cout<<"Prime Number"<<endl;
        getch();
        }
```

(c) The continue statement:-

The continue statement causes the program to skip the rest of the loop in the current iteration, as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

Syntax:-

```
        continue;
```

Example:-

```
        #include <iostream.h>
        void main ()
        {
        int n;
        cout<<"\nEnter Value of n"<<endl;
        cin>>n;
          for (n; n>0; n--)
         {
          if (n==5) continue;
         cout << n << ", ";
        }
        cout << "liftoff!\n";
        }
```

The goto statement

goto allows to make an absolute jump to another point in the program. This unconditional jump ignores nesting levels, and does not cause any automatic stack unwinding. Therefore, it is a feature to use with care, and preferably within the same block of statements, especially in the presence of local variables.

Definition of label:-

The destination point is identified by a label, which is then used as an argument for the goto statement. A label is made of a valid identifier followed by a colon (:).

Syntax of goto Control:-

goto <label_name>**;**

Syntax of label Control:-

<label_name>**:**

Function:- Full question

It is subprogram which is used for performing some well defined specific task.
Function may or may not consist of arguments.Arguments enclosed within parenthesis.

Or

A function is a set of program statements that can be processed independently. A function can be invoked which behaves as though its code is inserted at the point of the function call.The communication between caller(Calling function) and callee(called function) takes place through parameter.

Example:-

f(x)	Function with single argument/Parameter x.
f()	Function with no argument/Parameter.
f(x1,x2,x3,…)	Function with multiple arguments/Parameters.

Advantage of Function:-

- Modular programming.
- Redunction in the amount of work and development time.
- Program and function debugging is easier.
- Reduction in the size of program due to code reusability.
- Library of functions can be implemented by combining well designed, tested and proven functions.

Types of functions:-

- ✓ Built In function/System defined functions
- ✓ User Defined Functions
    - A function without argument and no return value.
    - A function without argument and return value.
    - A function with argument and no return value.
    - A function with argument and return value.
    - A function call by Address
    - A function pass by reference
    - Recursive function/Calling itself function.

Built In function/System defined functions:-

Example:-

pow(m,n)	m^n	
log(m)	m>0	
sqrt(x)	x>=0	
ln(x)	x>0	Natural log to the base 2<e<3.
strlen(string)		For measuring length of string.
strrev(string)		For reversing of string.
gets(string)		Accept a string from standard input device.
puts(string)		This function outputs a string constant or a string variable to the standard output device.
getchar()		It return a character that has been recently typed.

getche() It also return a character that has been recently typed.The typed character is echoed to the computer screen.

getch() This function too returns a character that has been recently typed.But neither the user is required to type enter key after entering character nor the typed character echoed to the computer screen.

putchar() This function output a character constant or a character variable to the standard output device.

main()
clrscr()
etc.

Function Components:-

- Function declaration or prototype.
- Function parameter(Formal Parameter).
- Combination of function declaration and its definition.
- Function definition(function declarator and function body).
- Return statement.
- Function call.

A function without argument and no return value.

Program 30

```
    #include<iostream.h>
    #include<conio.h>
    void sum()
    {
    int a,b,s;
    cout<<"\nEnter Any Two numbers="<<endl;
    cin>>a>>b;
    s=a+b;
    cout<<"\n Sum="<<s<<endl;
    }
void product()
{
int a1,b1,p;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a1>>b1;
p=a1*b1;
cout<<"\n Product="<<p<<endl;
}
void main()
{
sum();
product();
getch();
}
```

A function without argument and return value.

Program 31

```
#include<iostream.h>
#include<conio.h>
int sum()
{
int a,b,s;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a>>b;
s=a+b;
return s;
}
int product()
{
int a1,b1,p;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a1>>b1;
p=a1*b1;
return p;
}
void main()
{
int m1,n1;
m1=sum();
cout<<"\n Sum="<<m1<<endl;
n1=product();
cout<<"\n Product="<<n1<<endl;
getch();
}
```

A function with argument and no return value.

Program 32

```
#include<iostream.h>
#include<conio.h>
void product(int a1,int b1)
{
int p;
p=a1*b1;
cout<<"\n Product="<<p<<endl;
}
void main()
{
int a,b;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a>>b;
product(a,b);
getch();
}
```

A function with argument and return value.

Program 33

```
#include<iostream.h>
#include<conio.h>
int product(int a1,int b1)
{
int p;
p=a1*b1;
return p;
}
void main()
{
int a,b,m1;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a>>b;
m1=product(a,b);
cout<<"\n Product="<<m1<<endl;
getch();
}
```

A function call by Address:-

Program 34

```
#include<iostream.h>
#include<conio.h>
int swap(int *a1,int *b1)
{
int p;
p=*a1;
*a1=*b1;
*b1=p;
}
void main()
{
int a,b,m1;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a>>b;
cout<<"\n Value Before swapping="<<"a="<<a<<"\tb="<<b<<endl;

cout<<"\n Value Before swapping="<<"a="<<a<<"\tb="<<b<<endl;
swap(&a,&b);
cout<<"\n Value after swapping="<<"a="<<a<<"\tb="<<b<<endl;
getch();
}
```

A function pass by reference:-

Program 35

```
#include<iostream.h>
#include<conio.h>
int swap(int &a1,int &b1)
{
int p;
p=a1;
a1=b1;
b1=p;
}
void main()
{
int a,b,m1;
cout<<"\nEnter Any Two numbers="<<endl;
cin>>a>>b;
cout<<"\n Value Before swapping="<<"a="<<a<<"\tb="<<b<<endl;
swap(a,b);
cout<<"\n Value after swapping="<<"a="<<a<<"\tb="<<b<<endl;
getch();
}
```

Recursive function/Calling itself function.

A function that contains a function call to itself or a function call to a second function which eventually calls the first function is known as recursive function.

Condition for recursion:-

- Each time a function calls itself it must be nearer ,in some sense to a solution.
- There must be a decision criterion for stopping the process or computation.

Example:-

Recursive function for factorial:-

```
fact(n)=1              if n=0
fact(n)=n*fact(n-1)    if n>0
```

Program 36

```
#include<iostream.h>
#include<conio.h>
long int fact(long int n1)
{
if(n1==0)
return 1;
else
 return n1*fact(n1-1);
}
 void main()
{
long int n,m;
clrscr();
cout<<"\nEnter value of n="<<endl;
cin>>n;
m=fact(n);
cout<<"\n\nFactorial of number="<<n<<"="<<m<<endl;
getch();
}
```

Program 37(Series of Factorial numbers)

```
#include<iostream.h>
#include<conio.h>
long int fact(long int n1)
{
if(n1==0)
return 1;
else
 return n1*fact(n1-1);
}
 void main()
{
long int n,m;
int i;
clrscr();
cout<<"\nEnter value of n="<<endl;
cin>>n;
for(i=1;i<=n;i++)
{
```

```
m=fact(i);
cout<<"\n\nFactorial of number="<<i<<"="<<m<<endl;
}
getch();
}
```

Program 38(Series of Fibonaccie numbers)

Recursive function for Fibonaccie series:-

```
        fib(n)=0                  if      n=0
        fib(n)=1                  if      n=1
        fib(n)=fib(n-1)+fib(n-2)  if      n>1
```

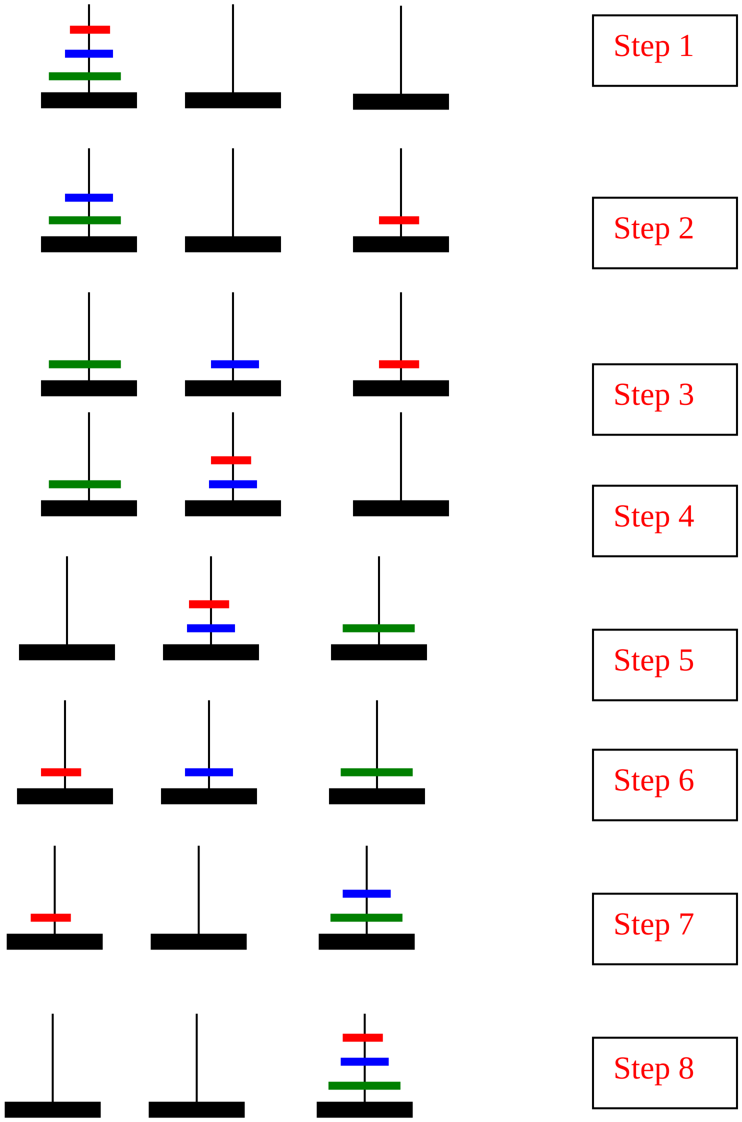
```
#include<iostream.h>
#include<conio.h>
 fib(long int n1)
{
if(n1==0 || n1==1)
  return 1;
else
  return fib(n1-1)+fib(n1-2);
}

void main()
{
long int n,m,i;
clrscr();
cout<<"\nEnter value of n="<<endl;
cin>>n;
cout<<"\nFibonaccie series=\n\n"<<endl;
for(i=0;i<=n;i++)
{
m=fib(i);
cout<<m<<"\t";
}
getch();
}
```

Tower of Hanoi Using Recursion Method:-

Tower of Hanoi is a historical problem, which can be easily expressed using recursion. There are n disks of decreasing size stacked on one needle, and two other empty needles, It is required to stack all disks onto a second needle in the decreasing order of size. Third needle can be used, as temporary storage.The movement of disks must confirm to the following rules.

- Only one disk may be moved at a time.
- A disk can be moved from any needle to any other.
- At no time, A larger disks rests upon a smaller one.

Let number of disks n=3



Example:-39

```
#include<iostream.h>
void hanoi(int n1,char left,char mid,char right)
{
 if(n1!=0)
 {
  hanoi(n1-1,left,right,mid);
  cout<<"Move Disk="<<n1<<"\t From\t"<<left<<"\t To\t"<<right<<endl;
  hanoi(n1-1,mid,left,right);
  }
  }
void main()
{
 int n;
 char Source='S',Inter='I',Dest='D';
 void hanoi(int,char,char,char);
 cout<<"\nEnter Number of Disc="<<endl;
 cin>>n;
 cout<<"Tower of Hanoi Problem="<<n<<"-Disks"<<endl;
 hanoi(n,Source,Inter,Dest);
}
```

Inline Function:-

An inline function definition is similar to an ordinary function except that the keyword inline precedes the function definition.The significant feature of inline functions is:There is no explicit function call and body is substituted at the point of inline function call,thereby,the run-time overhead for function linkage mechanism is reduced.

Syntax:-

```
Inline <return_type> function_name(return type)
            {
                        …..
                        ….
                        ….

            }
```

Example:-40 Product of any two numbers using inline keywords

```
#include<iostream.h>
inline f1(int a1,int b1)
{
 int c1;
 c1=a1*b1;
 return c1;
  }
void main()
{
 int a,b,n;
 cout<<"\nEnter any two Number ="<<endl;
 cin>>a>>b;
 n=f1(a,b);
  cout<<"\nProduct of any two integers="<<n<<endl;
}
```

Function Overloading:- Important

Assigning one or more function body to the same name is known as function overloading or function name overloading.

Example:-41

```
#include<iostream.h>
void f1(int a1,int b1)
{
             int c1;
             c1=a1+b1;
            cout<<"\nSum="<<c1<<endl;
  }
  void f1(float m1,float m2)
{
             float c2;
             c2=m1/m2;
            cout<<"\nDivision="<<c2<<endl;
  }
void main()
{
             int a,b;
             float m,n;
             cout<<"\nEnter any two integer Numbers ="<<endl;
             cin>>a>>b;
            f1(a,b);
            cout<<"\nEnter any two float Numbers ="<<endl;
             cin>>m>>n;
             f1(m,n);
}
```

Function Templates:- Important

C++ allows to create a single function possessing the capabilities of several functions. Which differ only in the data types.Such function is known as function templates or generic function.

Syntax:-

```
            template<class T1,class T2…>
            Return_Type function_name( Parameter of type T1,T2…)
            {
            ….
            ….
            ….
            }
```

Example:-42

```
  template <class T1>
  void swap(T1 &a,T1 &b)
  {
  T1 t1;
  t1= a;
  a=b;
  b=t1;
  }
void main()
{
 int a,b;
 float m,n;
 char p,q ;
 cout<<"\nEnter any two integer Numbers ="<<endl;
 cin>>a>>b;
cout<<"\nInteger before swaping="<<"a="<<a<<"\t"<<"b="<<b<<endl;
swap(a,b);
cout<<"\nInteger after swaping="<<"a="<<a<<"\t"<<"b="<<b<<endl;
cout<<"\nEnter any two float Numbers ="<<endl;
 cin>>m>>n;
 cout<<"\nFloat before swaping="<<"m="<<m<<"\t"<<"n="<<n<<endl;
swap(m,n);
cout<<"\nFloat after swaping="<<"m="<<m<<"\t"<<"n="<<n<<endl;
cout<<"\nEnter any two Characters ="<<endl;
 cin>>p>>q;
 cout<<"\nCharacter before swaping="<<"p="<<p<<"\t"<<"q="<<q<<endl;
swap(p,q) ;
cout<<"\nCharacter after swaping="<<"p="<<p<<"\t"<<"q="<<q<<endl;
}
```

Array:-

Collection of similar data types element is called array.

Types of array:-

1:-Single dimensional array.
2:-Double Dimensional Array.

Syntax:-

Single dimensional array

<Data_types> <Array_Name>[Size];

Example:-

int m[10]={4,7,1,9,2,2,2,11,22,33};

m[0]	m[1]	m[2]	m[3]	m[4]	m[5]	m[6]	m[7]	m[8]	m[9]
4	7	1	9	2	2	2	11	22	33

Double dimensional array

<Data_types> <Array_Name>[Size1][Size2];
Or
<Data_types> <Array_Name>[Row][Column];

int m[3][3] = {{2,5,9},{8,9,4},{8,6,5}};

Row Column

$$\begin{pmatrix} m[0][0] & m[0][1] & m[0][2] \\ m[1][0] & m[1][1] & m[1][2] \\ m[2][0] & m[2][1] & m[2][2] \end{pmatrix}$$

$$\begin{pmatrix} 2 & 5 & 9 \\ 8 & 9 & 4 \\ 8 & 6 & 5 \end{pmatrix}$$

Example 43 of Single Dimensional Array:-

```
#include<iostream.h>
```

```
#include<conio.h>
void main()
{
 int m[12],i;
 clrscr();
cout<<"\nEnter elements of Single Dimensional arrays="<<endl;
  for(i=0;i<=11;i++)
  cin>>m[i];
  cout<<"\nElements of Single dimensional arrays="<<endl;
  for(i=0;i<=11;i++)
  cout<<m[i]<<"\t";
  getch();
 }
```

Example 44 of Double Dimensional Array:-

```
#include<iostream.h>
#include<conio.h>
void main()
{
 int m1[3][4],i,j;
clrscr();
cout<<"\nEnter elements of matrix="<<endl;
 for(i=0;i<=2;i++)
for(j=0;j<=3;j++)
cin>>m1[i][j];
cout<<"\nElements of matrix="<<endl;
 for(i=0;i<=2;i++)
{
for(j=0;j<=3;j++)
cout<<m1[i][j]<<"\t";
cout<<"\n";
}
}
```

Sorting:-

It is a technique for ordering elements either ascending order or descending order.

- Selection Sort.
- Bubble Sort.
- Insertion Sort.
- Heap Sort.
- Quick Sort.
- Radix Sort/Bucket Sort.
- Merge Sort.

Selection Sort:

Sort the following unordered elements in either ascending order or descending order.

44, 33,55,22,11

```
#include<iostream.h>
main()
{
int a[10],i,j;
clrscr();
cout<<"\nEnter Unordered Elements of Arrays= "<<endl;
for(i=0;i<=9;i++)
cin>>a[i];
cout<<"\nUnOrdered Elements of Arrays=\n "<<endl;
for(i=0;i<=9;i++)
cout<<a[i]<<"\t";
cout<<"\nOrdered Elements of Arrays=\n "<<endl;
for(i=0;i<=9;i++)
{
for(j=i+1;j<=9;j++)
{
 if(a[i]>a[j])
 {
  int t;
  t=a[i];
  a[i]=a[j];
  a[j]=t;
 }
 }
 }
for(i=0;i<=9;i++)
cout<<a[i]<<"\t";
}
```

Bubble Sort:

```
#include<iostream.h>
#include<conio.h>
main()
{
int a[10],i,j;
clrscr();
cout<<"\nEnter Unordered Elements of Arrays= "<<endl;
for(i=0;i<=9;i++)
cin>>a[i];
cout<<"\nUnOrdered Elements of Arrays=\n "<<endl;
for(i=0;i<=9;i++)
cout<<a[i]<<"\t";
cout<<"\nOrdered Elements of Arrays=\n "<<endl;
for(i=0;i<=9;i++)
{
for(j=0;j<=9-i;j++)
{
 if(a[j]>a[j+1])
 {
  int t;
  t=a[j];
  a[j]=a[j+1];
  a[j+1]=t;
 }
 }
 }
for(i=0;i<=9;i++)
cout<<a[i]<<"\t";
getch();
}
```

Insertion Sort:

```
#include<iostream.h>
#include<conio.h>
main()
```

```
{
int a[10],i,j,k,t;
clrscr();
cout<<"\nEnter Unordered Elements of Arrays= "<<endl;
for(i=0;i<=9;i++)
cin>>a[i];
cout<<"\nUnOrdered Elements of Arrays=\n "<<endl;
for(i=0;i<=9;i++)
cout<<a[i]<<"\t";
cout<<"\nOrdered Elements of Arrays=\n "<<endl;
for(i=0;i<=9;i++)
{
t=a[i];
for(j=0;j<i;j++)
{
 if(t<a[j])
 {
  for(k=i;k>=j;k--)
  a[k]=a[k-1];
  a[j]=t;
  break;
 }
 }
 }
for(i=0;i<=9;i++)
cout<<a[i]<<"\t";
getch();
}
```

Quick Sort:-

```
#include<iostream.h>
int part(int low,int high,int *a)
{
 int i,h=high,l=low,p,t;  //p==pivot
p=a[low];
 while(low<high)
{
while(a[l]<p)
{
l++;
}
while(a[h]>p)
{
 h--;
}
if(l<h)
{
 t=a[l];
a[l]=a[h];
a[h]=t;
 }
else
{
 t=p;
 p=a[l];
a[l]=t;
 break;
}
}
 return h;
}
void quick(int l,int h,int *a)
{
  int index,i;
  if(l<h)
  {
  index=part(l,h,a);
  quick(l,index-1,a);
  quick(index+1,h,a);
  }
}
int main()
{
int a[10],n,l,h,i;
cout<<"Enter number of elements:";
cin>>n;
cout<<"Enter the elements (Use Space As A Separator):";
for(i=0;i<n;i++)
cin>>a[i];
cout<<"\nInitial Array:\n";
for(i=0;i<n;i++)
{
cout<<a[i]<<"\t";
}
h=n-1;
l=0;
quick(l,h,a);
cout<<"\nAfter Sorting:\n";
for(i=0;i<n;i++)
{
cout<<a[i]<<"\t";
}
return 0;
}
```

Heap Sort:-

```
#include <iostream.h>
void max_heapify(int *a, int i, int n)
{
int j, temp;
 temp = a[i];
 j = 2*i;
while (j <= n)
{
 if (j < n && a[j+1] > a[j])
j = j+1;
 if (temp > a[j])
break;
 else if (temp <= a[j])
{
a[j/2] = a[j];
j = 2*j;
 }
 }
a[j/2] = temp;
 return;
}
void heapsort(int *a, int n)
{
 int i, temp;
 for (i = n; i >= 2; i--)
{
 temp = a[i];
a[i] = a[1];
a[1] = temp;
max_heapify(a, 1, i - 1);
}
}
void build_maxheap(int *a, int n)
{
 int i;
 for(i = n/2; i >= 1; i--)
{
max_heapify(a, i, n);
}
}
int main()
{
    int n, i, x;
   cout<<"enter no of elements of array\n";
    cin>>n;
    int a[20];
    for (i = 1; i <= n; i++)
{
    cout<<"enter element"<<(i)<<endl;
    cin>>a[i];
    }
 build_maxheap(a,n);
    heapsort(a, n);
    cout<<"sorted output\n";
    for (i = 1; i <= n; i++)
   {
  cout<<a[i]<<endl;
    }
}
```

Pointer:-   Most Important

It is variable which point the address of variable.Ponter variable represented by using indirection operator (*).

What Are Pointers?

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before you can use it to store any variable address. The general form of a pointer variable declaration is:

```
type *variable_name;
```

Syntax:-

<Data_type> <variable1>,<variable2>,<variable3>…,*ptr1, *ptr2,*ptr3…;

Where :-

ptr1=address of variable1 (&variable1);

```
ptr2=address of variable2 (&variable2);
ptr3=address of variable3 (&variable3);
…
…
…
*ptr1=Value at address variable1;
*ptr2=Value at address variable2;
*ptr3=Value at address variable3;
…
…
…
```

Example:-

```
int a=12,b=16,*p1,*p2;
p1=&a;
p2=&b;
*p1=value at address a;
*p2=value at address b;
```

Program based on OOPs

Format of class Declaration in C++:-

```
    class <class_Name>
    {
    private :
        Data_members        //Attributes;
        Member_functions    //Behaviours//Methods;
    public :
        Data_members        //Attributes;
        Member_functions    //Behaviours//Methods;
    protected :
        Data_members        //Attributes;
        Member_functions    //Behaviours//Methods;
    };
```

Where :-

private, public and protected are called **visibility mode**.

Example:-1

```
#include<iostream.h>
 #include<conio.h>
    class base
    {
    private :
    void f1()
        {
        int a=10,b=12,c1;
        c1=a+b;
    public:
    void f2()
        {
        int a1=10,b1=12,c2;
        c2=a1*b1;
         cout<<"\n Product of two numbers="<<c2<<endl;
        cout<<"\n I am member of Visibility mode public"<<endl;
        }
    protected:
    void f3()
        {
        int a2=10,b2=12,c3;
        c3=a2-b2;
         cout<<"\n Difference of two numbers="<<c3<<endl;
        cout<<"\n I am member of Visibility mode protected"<<endl;
        }
    };
    void main()
    {
     base obj1;
     //obj1.f1();
     obj1.f2();
    // obj1.f3();
    }
```

Example:-2

```
#include<iostream.h>
#include<conio.h>
class base
{
private :
void f1()
        {
        cout<<"\n I am member of Visibility mode private"<<endl;
        }
public:
void f2()
        {
        f1();
        cout<<"\n I am member of Visibility mode public"<<endl;
        f3();
        }
protected:
        void f3()
        {
        cout<<"\n I am member of Visibility mode protected"<<endl;
        }
};
void main()
{
 base obj1;
 obj1.f2();
}
```

Example:-3 Function using arguments in class based program

```
#include<iostream.h>
#include<conio.h>
class base
{
private :
        int x,y;
public:
void f2(int a1,int b1)
        {
        int c1;
          x=a1;
          y=b1;
          c1=x*y;
        cout<<"\n Product of two numbers="<<c1<<endl;
          }
protected:
        void f3()
        {
        cout<<"\n I am member of Visibility mode base  protected"<<endl;
        }
};
void main()
{
 base obj1;
int a,b;
cout<<"\nEnter Value of a and b="<<endl;
```

```
        cin>>a>>b;
        obj1.f2(a,b);
        }
```

Example:-4 Function using arguments in class based program with return keyword

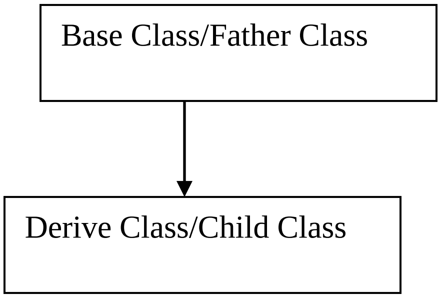
```
#include<iostream.h>
#include<conio.h>
class base
{
private :
        int x,y;
public:
        int f2(int a1,int b1)
                {
                int c1;
                  x=a1;
                  y=b1;
                  c1=x+y;
                  return c1;
                  }
protected:
        void f3()
                {
                cout<<"\n I am member of Visibility mode base  protected"<<endl;
                }
        };

        void main()
        {
         base obj1;
        int a,b,m1;
        cout<<"\nEnter Value of a and b="<<endl;
        cin>>a>>b;
          m1=obj1.f2(a,b);
        cout<<"\n Sum of two numbers="<<m1<<endl;
        }
```

Inheritence:- (Important)

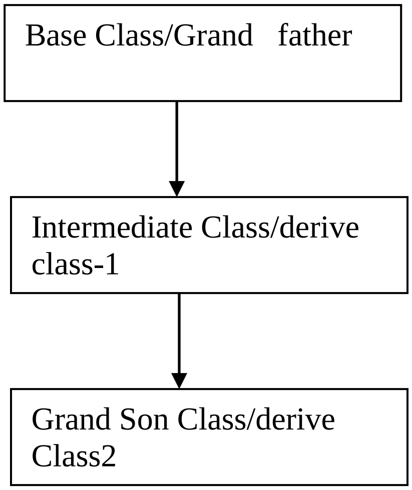
It means reusability.That is property of base class is going to access by its derive class. Whose property is going access is called as base class and in which the property is access is called as a derive class.

Types of Inheritence:-

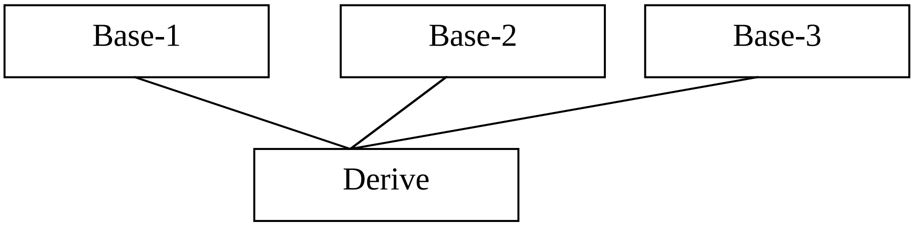
- **Single Inheritence**



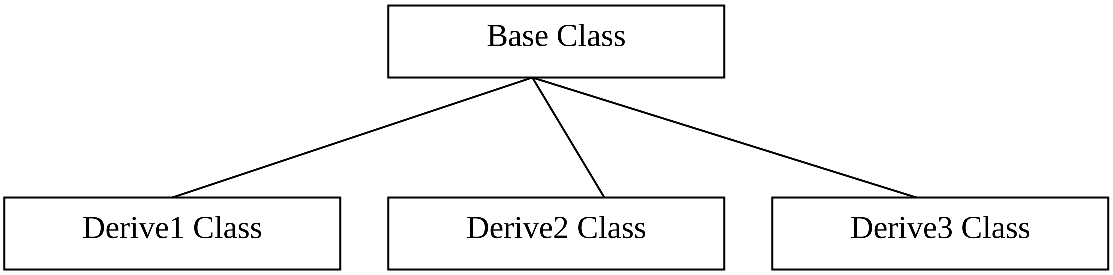
- **Multi level Inheritence**


Base Class/Grand father
Intermediate Class/derive class-1
Grand Son Class/derive Class2

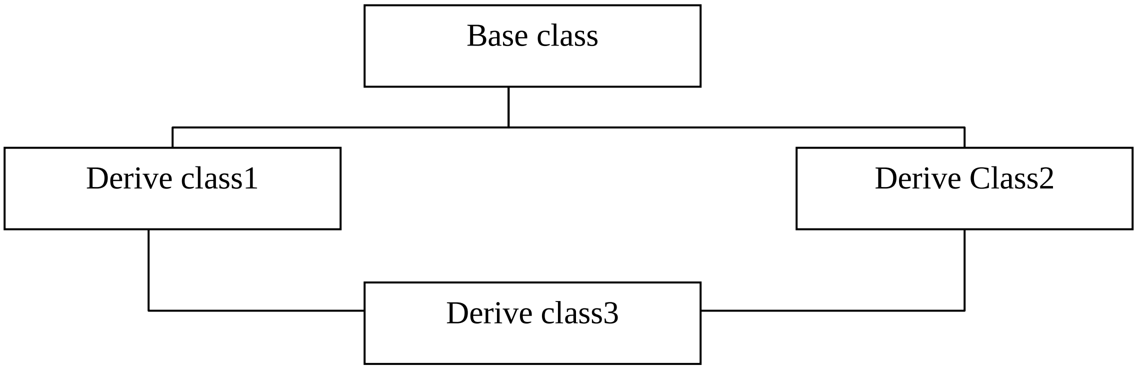
- **Multiple Inheritence**



- **Hierarchical Inheritence**



- **Hybrid Inheritence**



Syntax:-

```
class <new class name> :Visibility Mode <Base Class> or <Old Class>
        or
        Derive class
        {
        ……………
        ……………
        ……………
        }
```

Visibility Mode :-Visibility mode of derive class (Only public & private).

- **Single Inheritence**

```
#include<iostream.h>
 #include<conio.h>
class base
{
private :
void f1()
        {
        cout<<"\n I am member of Visibility mode private"<<endl;
        }
public:
void f2()
        {
        cout<<"\n I am member of Visibility mode public"<<endl;
        }
protected:
void f3()
        {
        cout<<"\n I am member of Visibility mode protected"<<endl;
        }
};
class derive : public base
{
private :
void f4()
        {
        cout<<"\n I am member of Visibility mode private in derive class"<<endl;
        }
public:
void f5()
        {
        cout<<"\n I am member of Visibility mode public in derive class"<<endl;
        }
protected:
void f6()
        {
        cout<<"\n I am member of Visibility mode protected in derive class"<<endl;
        }
};
void main()
{
 base obj1;
 derive obj2;
 obj1.f2();
 obj2.f2();
 obj2.f5();
}
```

Note:-

- If VM of derive class is private then public of base class is become private.As the result the object of base class can access the members of own.
- But object of derive class can not access members of the public of the base class directly.
- Protected member of base class can be inherit by derive class, can be access by public its own class but can be access directly by the main.

Multi level Inheritence

```
#include<iostream.h>
#include<conio.h>
        class base
        {
        private :
        void f1()
                {
                cout<<"\n I am member of Visibility mode private"<<endl;
                }
        public:
        void f2()
                {
                cout<<"\n I am member of Visibility mode public"<<endl;
                }
        protected:
        void f3()
                {
                cout<<"\n I am member of Visibility mode protected"<<endl;
                }
        };
        class derive1 : public base
        {
        private :
        void f4()
                {
                cout<<"\n I am member of Visibility mode private in derive1 class"<<endl;
                }
        public:
        void f5()
                {
                cout<<"\n I am member of Visibility mode public in derive1 class"<<endl;
                }
        protected:
        void f6()
                {
                cout<<"\n I am member of Visibility mode protected in derive1 class"<<endl;
                }
        };

        class derive2 : public derive1
        {
        private :
        void f7()
                {
                cout<<"\n I am member of Visibility mode private in derive2 class"<<endl;
                }
        public:
        void f8()
                {
                cout<<"\n I am member of Visibility mode public in derive2 class"<<endl;
                }
        protected:
        void f9()
                {
```

```
                cout<<"\n I am member of Visibility mode protected in derive2 class"<<endl;
                }
        };
        void main()
        {
         base obj1;
         derive1 obj2;
         derive2 obj3;
         obj1.f2();
         obj2.f5();
         obj3.f8();
         obj2.f2();
         obj3.f2();
         obj3.f5();
        }
```

Multiple Inheritence

```
#include<iostream.h>
 #include<conio.h>
        class base1
        {
        private :
        void f1()
                {
                cout<<"\n I am member of Visibility mode private base1"<<endl;
                }
        public:
        void f2()
                {
                cout<<"\n I am member of Visibility mode public base1"<<endl;
                }
        protected:
        void f3()
                {
                cout<<"\n I am member of Visibility mode protected base1"<<endl;
                }
        };
        class base2
        {
        private :
        void f4()
                {
                 cout<<"\n I am member of Visibility mode private in base2 class"<<endl;
                }
        public:
        void f5()
                {
                cout<<"\n I am member of Visibility mode public in base2 class"<<endl;
                }
        protected:
        void f6()
                {
                cout<<"\n I am member of Visibility mode protected in base2 class"<<endl;
                }
        };
        class derive1 : public base1,public base2
        {
        private :
        void f7()
                {
                cout<<"\n I am member of Visibility mode private in derive1 class"<<endl;
                }
        public:
        void f8()
                {
                cout<<"\n I am member of Visibility mode public in derive1 class"<<endl;
                }
        protected:
        void f9()
                {
                cout<<"\n I am member of Visibility mode protected in derive1 class"<<endl;
                }
        };
        void main()
        {
         base1 obj1;
         base2 obj2;
         derive1 obj3;
         obj1.f2();
         obj2.f5();
         obj3.f8();
         obj3.f2();
         obj3.f5();
         }
```

Hierarchical Inheritence

```
#include<iostream.h>
 #include<conio.h>
        class base
        {
        private :
        void f1()
                {
                cout<<"\n I am member of Visibility mode private base"<<endl;
                }
        public:
        void f2()
                {
                cout<<"\n I am member of Visibility mode public base"<<endl;
                }
        protected:
        void f3()
                {
                cout<<"\n I am member of Visibility mode protected base"<<endl;
                }
        };
        class derive1 : public base
        {
        private :
        void f4()
                {
```

```
        cout<<"\n I am member of Visibility mode private in derive1 class"<<endl;
        }
public:
void f5()
        {
        cout<<"\n I am member of Visibility mode public in derive1 class"<<endl;
        }
protected:
void f6()
        {
        cout<<"\n I am member of Visibility mode protected in derive1 class"<<endl;
        }
};

class derive2 : public base
{
private :
void f7()
        {
        cout<<"\n I am member of Visibility mode private in derive2 class"<<endl;
        }
public:
void f8()
        {
        cout<<"\n I am member of Visibility mode public in derive2 class"<<endl;
        }
protected:
void f9()
        {
        cout<<"\n I am member of Visibility mode protected in derive2 class"<<endl;
        }
};
void main()
{
  base obj1;
  derive1 obj2;
  derive2 obj3;
  obj1.f2();
  obj2.f5();
  obj2.f2();
  obj3.f2();
  obj3.f8();
 }
```

Virtual function:-Gauranteed Question

With the same function name is used with the same prototype in base class & derive class such type of function is created as virtual function.

Syntax:-

```
virtual <return_Type> function (arguments)
{
Procedure;
};
```

```
#include<iostream.h>
```

```
#include<conio.h>
        class base
        {
        private :
        public:
        virtual void f1()
                {
                cout<<"\n I am member of Visibility mode public base"<<endl;
                }
        protected:
        };
        class derive1 : public base
        {
        private :
        public:
        void f1()
                {
                cout<<"\n I am member of Visibility mode public in derive1 class"<<endl;
                }
        protected:
        };
        void main()
        {
         base obj1;
         derive1 obj2;
         base *ptr1;
         ptr1=&obj1;
         ptr1->f1();
         derive1 *ptr2;
         ptr2=&obj2;
         ptr2->f1();
        }
```

Example of passing arguments in Virtual function:-

```
        #include<iostream.h>
         #include<conio.h>
        class base
        {
        private :
        public:
        virtual void f1(int a1,int b1)
                {
                int c1;
                c1=a1+b1;
                cout<<"\n Sum of Two numbers="<<c1;
                cout<<"\n I am member of Visibility mode public base"<<endl;
                }
        protected:
        };
        class derive1 : public base
        {
        private :
        public:
        void f1(int a1,int b1)
                {
                int c2;
                c2=a1*b1;
                cout<<"\n Product of Two numbers="<<c2;
                cout<<"\n I am member of Visibility mode public in derive1 class"<<endl;
                }
        protected:
        };
        void main()
        {
          base obj1;
          derive1 obj2;
          int a,b;
          cout<<"\nEnter Value of a and b="<<endl;
          cin>>a>>b;
          base *ptr1;
          ptr1=&obj1;
          ptr1->f1(a,b);
          derive1 *ptr2;
          ptr2=&obj2;
          ptr2->f1(a,b);
         }
```

Friend Functions:-Most important

It is useful in following situations

- Function operating on objects of two different classes. This is ideal situation where the friend function can be used to bridge two classes.
- Friend functions can be used to increase the versatility of overloaded operators.
- Sometimes,a friend function allows a more obvious syntax for calling a function, rather than what a member function can do.

Syntax:-

friend <return_type> function(object type argument1,object type argument2…)

```
#include<iostream.h>
#include<conio.h>
class base2;
class base3;
class base1
{
private :
int x;
public:
         void f1(int x1)
        {
        x=x1;
        }
        friend int sum(base1 obj1,base2 obj2,base3 obj3);
        friend int pro(base1 obj1,base2 obj2,base3 obj3);
        friend int dif(base1 obj1,base2 obj2,base3 obj3);

};
class base2
{
private :
        int p;
```

```
public:
	 void f2(int p1)
	{
	p=p1;
	}
	friend int sum(base1 obj1,base2 obj2,base3 obj3);
	friend int pro(base1 obj1,base2 obj2,base3 obj3);
	friend int dif(base1 obj1,base2 obj2,base3 obj3);

};
class base3
{
private :
	int m;
public:
	void f3(int m1)
	{
	m=m1;
	}
	friend int sum(base1 obj1,base2 obj2,base3 obj3);
	friend int pro(base1 obj1,base2 obj2,base3 obj3);
	friend int dif(base1 obj1,base2 obj2,base3 obj3);

};

int sum(base1 obj1,base2 obj2,base3 obj3)
{
 int s1;
 s1=obj1.x + obj2.p+obj3.m;
 return s1;
}
int pro(base1 obj1,base2 obj2,base3 obj3)
{
 int z1;
 z1=obj1.x * obj2.p * obj3.m;
 return z1;
}

int dif(base1 obj1,base2 obj2,base3 obj3)
{
 int z2;
 z2=obj1.x - obj2.p - obj3.m;
 return z2;
}

void main()
{
 int r,s,d,z,g,q;
 base1 obj1;
 base2 obj2;
  base3 obj3;
 cout<<"\nEnter any three digits="<<endl;
 cin>>r>>s>>d;
 obj1.f1(r);
 obj2.f2(s);
 obj3.f3(d);
 z=sum(obj1,obj2,obj3);
 g=pro(obj1,obj2,obj3);
q=dif(obj1,obj2,obj3);
 cout<<"\n\nSum of three integers="<<z<<endl;
 cout<<"\n\nProduct of three integers="<<g<<endl;
cout<<"\n\nDifference of three integers="<<q<<endl;

 }
```

Friend Class:- Gauranteed Question

All the member functions of one class can be friend functions of another class.

Syntax:-

friend class <class_name>;

```
#include<iostream.h>
class base1
{
private :
int x,y;
public:
void f1(int x1,int y1)
{
x=x1;
y=y1;
}
friend class base2;
};
class base2
{
private :
public:
int f2(base1 obj1)
{
return obj1.x+obj1.y;
}
};
void main()
{
int a,b;
cout<<"\nEnter Value of a and b="<<endl;
cin>>a>>b;
base1 obj1;
base2 obj2;
obj1.f1(a,b);
cout<<"\nValue of x="<<obj2.f2(obj1)<<endl;
}
```

Constructor & Destructor:-

Constructor:-

It is used to initialize an object.

Feature:-

- Name of the class & name of the function should be same.
- Any constructor will not return any value
- But constructor can call the values.
- Constructor which called the value, It is called parameterized constructor.

```
class name (arguments)
{
        Procedure;
}
```

Example:-1 constructor passing no parameter

```
#include<iostream.h>
#include<conio.h>
class base1
{

public:
         base1()
         {
         cout<<"\n I Am first Constructor"<<endl;
         }
};
void main()
{
base1 obj1;
 }
```

Example:-2 constructor using technique of scope resolution ::

```
#include<iostream.h>
#include<conio.h>
class base1
{
private :

public:
         base1();
};
base1:: base1()
        {
         cout<<"\n I Am first Constructor"<<endl;
         }

 class base2
{
private :

public:
         base2();
};
base2:: base2()
        {
         cout<<"\n I Am second Constructor"<<endl;
         }

void main()
{
        base1 obj1;
        base2 obj2;
 }
```

Example:-3 constructor using parameter

```
#include<iostream.h>
#include<conio.h>
class base1
{
private :
        int x,y;
public:
         base1(int x1,int y1)
         {
               int z1;
               x=x1;
               y=y1;
               z1=x+y;
               cout<<"\nSum of two numbers="<<z1<<endl;
         }
};
void main()
{
  base1 obj1(7,8);
 }
```

Example:-4 constructor using parameter passing values from keyboard

```
#include<iostream.h>
#include<conio.h>
class base1
{
private :
        int x,y;
public:
         base1(int x1,int y1)
         {
               int z1;
               x=x1;
               y=y1;
               z1=x+y;
       cout<<"\nSum of two numbers="<<z1<<endl;
         }
};
void main()
{
       int a,b;
       cout<<"\nEnter Value of a and b="<<endl;
       cin>>a>>b;
       base1 obj1(a,b);
 }
```

Example:-5 constructor overloading

```
#include<iostream.h>
#include<conio.h>
class base1
{
private :
        int x,y;
        float m,n;
public:
```

```
         base1(int x1,int y1)
         {
                int z1;
                x=x1;
                y=y1;
                z1=x+y;
        cout<<"\nSum of two numbers="<<z1<<endl;
         }
         base1(float m1,float n1)
         {
                float p;
                m=m1;
                n=n1;
                p=m/n;
        cout<<"\nDivision of two numbers="<<p<<endl;
         }
};
void main()
{
        int a,b;
        float s,t;
        cout<<"\nEnter two integer Value of a and b="<<endl;
        cin>>a>>b;
        base1 obj1(a,b);
        cout<<"\nEnter two float Value of s and t="<<endl;
        cin>>s>>t;
        base1 obj2(s,t);
 }
```

Destructor:-

It is used to destroy the object which is created by constructor. Destructor neither call nor return the value.

Example:-

```
#include<iostream.h>
#include<conio.h>
class base1
{
public:
  base1();
 ~base1();
};
base1:: base1()
         {
         cout<<"\n I Am first Constructor"<<endl;
         }
        base1:: ~base1()
         {
         cout<<"\n  I Am Destructor of above constructor"<<endl;
         }
void main()
        {
        base1 obj1;
         }
```

Overriding Method:-

In this technique of programming defining a method in the subclass that has the same name, same arguments and same return type as a method in the subclass.Then, when the method is called, the method defined in subclass is invoked and executed instead of one in the super class. This is known as overriding.

Example:-1

```
#include<iostream.h>
class base
{
 public:
 void show()
 {
  cout << "\nAccessing member Base class";
 }
};
class derive:public base
{
 public:
 void show()
 {
  cout << "\nAccessing member of Derived Class";
 }
};
void main()
{
 base obj1;
derive obj2;
obj1.show();
obj2.show();
}
```

Example:-2

```
#include<iostream.h>
class base
{
int x,y;
 public:
 void show(int a1,int b1)
 {
int c1;
x=a1;
y=b1;
c1=x+y;
  cout << "\nAccessing member Base class";
cout << "\nsum of two numbers="<<c1;
 }
};
class derive:public base
{
int x,y;
 public:
 void show(int a1,int b1)
 {
int c2;
x=a1;
y=b1;
c2=x*y;
 cout << "\nProduct of two numbers="<<c2;
  cout << "\nAccessing member of Derived Class";
 }
};
void main()
{
 base obj1;
derive obj2;
int a,b;
cout<<"\nEnter value of a and b=";
cin>>a>>b;
obj1.show(a,b);
obj2.show(a,b);
}
```

Example of OOPs Languages:-

C++ Language

In the early 1980's, also at Bell Laboratories, another programming language was created which was based upon the C language. This new language was developed by Bjarne Stroustrup and was called C++. Stroustrup states that the purpose of C++ is to make writing good programs easier and more pleasant for the individual programmer. When he designed C++, he added OOP (Object Oriented Programming) features to C without significantly changing the C component. Thus C++ is a "relative" (called a superset) of C, meaning that any valid C program is also a valid C++ program.

Python Language:- Guido van Rossum

Python is a widely used general-purpose, high-level programming language. Python is a general purpose programming language created in the late 1980s, and named after Monty Python, that's used by thousands of people to do things from testing microchips at Intel, to powering Instagram, to building video games with the Play Game library .

Beta Language:-

BETA is a pure object-oriented language originating within the "Scandinavian School" in object-orientation where the first object-oriented language **Simula** was developed. Among its notable features, it introduced nested classes, and unified classes with procedures into so called patterns.

Here are the current beta features:-

Media Viewer

Improve your multimedia viewing experience with this new tool.

Visual Editor

Visual Editor makes it easy to edit pages without having to learn wiki code.

Visual Editor Formula

VisualEditor for creating and editing mathematical formula like algebra or equations on pages you're working on.

Typography Refresh

This beta feature makes text more readable, accessible and consistent.

C# Language:-

C# is a hybrid of C and C++, it is a Microsoft programming language developed to compete with Sun's Java language. C# is an object-oriented programming language used with XML-based Web services on the .NET platform and designed for improving productivity in the development of Web applications.

The following are additional C# resources:-

- For a good general introduction to the language.
- For detailed information about specific aspects of the C# language.