* Features of object oriented programming

Object - An object consists of name state and behaviour

Object

name
State
behaviour

Student
Char name [50] int Roll no.; Char branch; getdata (); display ();

Name :- It is an unique identity for representing any object of the real world.

State - It is representation of attributes by internal data structure.

Behaviour - It is a set of allowed operation, function and methods.

i) Object - Objects are the basic run time entities in an object oriented system.

ii) Class - Class is a collection of similar type of objects.
- Class is an user defined datatype.
- Objects are the variable of type class.
- Once a class has been defined, We can create n number of object belonging to that class.

Syntax to create an object is:

```
Classname . objectname
```

>> → extraction operator cin >>

<< → insertion operator cout <<

cout << "Hello";

scanf ("%d", &x);

printf ("%d", x);

#include <iostream>
using namespace std;
we before writing any code in C++

- Write a program to find out multiplication of two object without using multiplication operator in C++.

```
#include <iostream>
using namespace std;
int main()
    int x, y, i;
    cin >> x, y;
for (i=0; i <= y; i++)
    { z = x + z; }
cout << "Product of a & b is " << z
    return 0; }
```

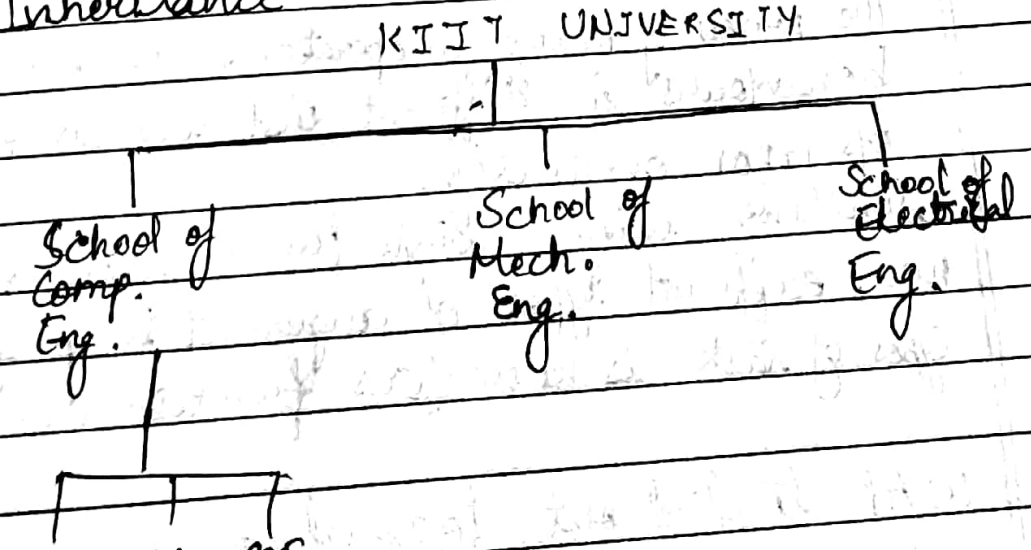
(3) Abstraction
The act of representing essential features without including the background details. It is of two types -
(i) Data abstraction
(ii) Control abstraction

(i) Data - Hiding the details about the data
(ii) Control - Hiding the implementation details.

Encapsulation
Wrapping up of data & function into a single unit (class) is called as Encapsulation.

Data hiding - The insulation from direct access of the program is called as data hiding.
Encapsulation & data hiding are the way to implement data abstraction.

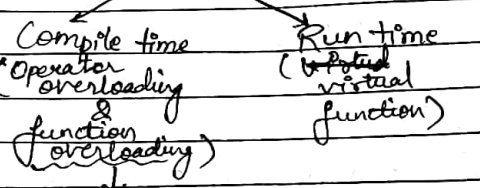
(4) Inheritance -



This is the property by which object of one class acquires the property of object of another class.
In OOP, the concept of inheritence provides the idea of reusability. This means that we can add additional

features to an existing class without modifying it. This can be done by deriving new class from an existing class.
The new the class will have the combined features for both the class. ~~This is the~~

5. Polymorphism




```
int Add(int, int);
float Add(int, float);
float Add (float, float);
float Add( int, float, float);
//function overloading doesn't depends on return type or any parameters.
```

Polymorphism means the ability to take more than one call. The
The process of making an operator to exhibit different behaviours in different instances is known as operator overloading.
Similarly using a single function name & different no., different types of argument to perform different types of task is known as function overloading.

Q WAP to find out factorial of a given no. using recursion.

```
#include <iostream>
int main()
{
int x;
```

```
fact(int);
```

```
cout << " Enter no. ";
cin >> x ;
cout << fact (x);
return 0;
}
int fact (int a);
{
if(a>1) a = a * (a-1)
return(a*fact(a));
return(fact(x));
}
```

1. WAP in C to input & display the details of some employees (structure)
2. WAP in C++ to check whether a no. is prime or composite using function.
3. WAP in C++ to check whether a no. is perfect or not.
4. WAP in C++ whether a no. is armstrong or not.

1

```
#include <stdio.h>
struct details
{
char name[30];
int age;
} e[2];
int main ()
{
int i;
for (i=1; i<3; i++)
{ scanf ("%s %d", e[i].name, & e[i].age);
}
for (i=1; i<3; i++)
{ printf ("name - %s age - %d \n", e[i].name,
e[i].age);
}
return 0;
}
```

Formal argument → Name of the calling function call

Dynamic Binding

- It is the process of linking a func. called to the actual code of the function at run time.
- Dynamic binding means that the code associated with a given procedure is not known until the time of the call at run time.
- The concept of dynamic binding is implemented with the help of inheritance and run time polymorphism (virtual function)

Message Passing

An object of the class can communicate with other objects by sending & recieving information. Message passing involves specifying name of object, name of function (message) & the information to be sent.

```
Classname obj;
obj.getdata(2,3.5);
```

obj → name, getdata → message, (2,3.5) → information

Reference variable

Call by value

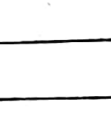
```
main()
{   add(int, int);
    int a=2, b=3;
    add(a,b)
}
add(int p, int q)
{   cout << p+q;
}
```

Call by address

```
main()   { add (int*, int*)
    int a=2,b=3;
    add (&a, &b)
}
add (int *p, int *q)
{   cout << *p + *q ;
}
```

Reference variable

```
int a;
a=10;
int &b=a
```



Syntax →

datatype & referencevarname = Existing variable

→ formanal variable → Ref. var.

Call by Reference

```
main()
{   add (int &, int &);
    int a=2, b=3;
    add (a, b);
}
add (int &p, int &q)
{   cout << p+q
}
```

Q WAP to represent call by value, address & reference to increment a variable 5 by 1.

```
#include<iostream.h>
main()  using namespace std
{
f1(int); f2(int*) ; f3(int&);
int s = 5;
f1(s); f2(&s) ; f3(s);
}
f1(int p)
{
++*p;
cout << p;
}
f2(int *p)
{
++p;
cout <<*p;
}
f3(int &p)
{++p; cout << p;
}
```

Default argument

WAP to perform multiplication of three nos. using default arguments for atleast two nos..

```
#include<iostream>
int mult(int a, int b, int c)
{ return (a*b*c);
}
int main()
{ int mult(int a ,int b=20, int c=2);
cout << mult(10,30) << mult(30,20)
                    << mult(20);
return 0;
}
```

Inline function

Inline functions are expanded in a line where it is called. The inline keyword sends a request not a command to the compiler to expand it in a line.

→ The inline function should have almost one or two lines of code.

Scope Resolution operator (::)

used to perform addition of local & global variable with same name.

```
#include <iostream>
using namespace std;
int m=10;
int main()
{ int m=20;
  cout << ::m + m;
  return 0;
}
```

O/p → 30

Memory management operator

In C++ new operator is used for dynamic memory allocation & delete operation is used to free the allocated memory.

The general syntax of new & delete operator

```
datatype pointer = new datatype(value);
datatype pointer = new datatype [value];
delete p;
delete [] p;
```

Q) WAP to dynamically create & initialise one integer variable, char var & float var.

```
#include <iostream>
using namespace std;
int main()
int *p = new int (20);
char *q = new char('A');
float *r = new float (4.0);

cout << *p << *q << *r;
return 0;
}
```

Q) To initialize a float array using new operator, display the content of the array and delete the allocated memory.

```
#include <iostream>
using namespace std;
int main()
int *p = new float [5];
int i;
for(i=0; i<5; i++)
cin >> *(p+i);

for(i=0; i<5; i++)
{ cout << *(p+i); }
delete [5]p;

return 0;
}
```

To declare & define MEMBER CLASS

```
Class classname
{
  Private :
    data member
    Member function
  Public :
    data member
    Member funct-
};
```

```
Class A
{ Private :
    int b;
    float c;
  Public :
    void input ();
    void output ();
    { cout << b << c;
    }
};
```

The general syntax to define a function outside the class

```
return type :: fun( Arg)
  {
  }
```

```
void A :: input ()
  { cin >> b >> c;
  }

int main()
{ A :: ob;
  ob.input();
  ob.output();
  return 0;
}
```

```
void main()
{
    Employee e[70];
    for( int i=0; i<70; i++)
    {   e[i].input();
    }
    for( int i=0; i<70; i++)
    {   e[i].input();
    }
    return 0;
}
```

datatype pointer
datatype po

delete.p;
delete.[

```
# Class ABC
{
  int x;
  Public:
    int y;
    void input();
    void show();
};
void ABC::input()
{   x=5;
}
void ABC::show()
{ cout<<x<<y;
}
int main()
{   ABC ob;
    ob.input();
    Ob.y = 10;
    ob.show();
    return 0;
}
```

Private acess Specifier

```
Class ABC
{ int x;
  int input();
  Public:
   void show()
    { cout << input();
    }
}
int ABC::input()
{ cin >> x;
   return x;
}

int main()
{ ABC ob;
  Ob.show();
   return 0;
}
```

Q) WAP to find out largest b/w two nos. using three member function -

(i) Input
(ii) calculation of largest
(iii) display output

Define largest func. under private access specifier.

```
Class largest
{ int x, y;
  Public:
void input()
  { cin >> x >> y; }
void output()
```

{ cout << calculation(); }

Private:

int calculation();

};

int largest :: calculation()
{ if (x > y)
return x;
else if (x < y)
return y;
}

int main()
{ largest ob;
ob. input();
ob. output();
return 0;
}

Nested member function

Class ABC
{ int x;
Public:
void show();
void input();
};

int ABC :: show()
{ input(); cout << x; }

int ABC :: input()
{ cin >> x; }

int main()
{ ABC ob;
ob. show();
return 0;
}

Inline function in Class

```
Class ABC
{  int x;
Public:
inline void input ();
};
Inline void ABC :: input()
   {  cin >>x;
      cout <<x;
   }
int main()
{  ABC a;
a. input ();
return 0;
}
```

Q WAP to implement private, nested & inline func. in class.

```
Class ABC
   int x;
void Input ();
   Public:  int y
    void output ();
{
      input ();
};
inline void ABC :: input ()
   {     cin >> x >>y;
         cout <<x;
   }
```

```
int main ()
{ ABC a;
  a.output ();
  return 0;
}
```

Lab 24/4

Static Data member

Memory allocation of objects

Memory space for objects is allocated when they are declared and not when the class is specified this statement is partly true.
Actually the member func.s are created & placed in the memory space only once when they are defined as a part of class specification.
Since all the objects belonging to that class use the same member func.s, no separate space is allocated for member functions when the objects are created.
Only space for member variable is allocated separately for each object belonging to that class.

Static Data member

```
Class A
{ Static int c; // declaration
  int x;
Public:
void input ()
{ cin >> x;
  c++;
}
```

Datatype pointer = datatype pointer

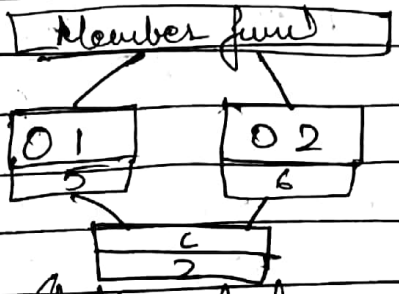
delete. p;
delete. [5];

```
void show()
{ cout<<x;
  cout<<c;
};
int A::c; // definition
main()
{ A o1,o2;
o1.input(); // 5
o2.input(); // 6
o1.show();
o2.show();
}
```

if c = 10
then o/p is
5 12
6 12

o/p

5	2
6	2



Characteristics of static data member

→ It is initialized to 0 when the first object of its class is created no other initialization is permitted.

→ Only one copy of that static member is created for the entire class & is shared by all objects of that class.

→ It is visible only within the class but its lifetime is entire program.

Static datamembers are normally used to maintain values common to the entire class.

The type & scope of each static datamember must be defined outside the class definition this is necessary because the static data members are stored separately rather than as part of an object.

Q) To count the no. of objects created for a class using static datamember.

```
Class A
{   Static int count;
    Public:
    void B()
     {count++;
     };        void show () { cout << c; }
main ()
{
   A Ob1, Ob2, Ob3;
   Ob1. B();
    Ob2. B();
    Ob3. B();
                show();
}
```

Static Member function (SMF)

→ SMF can have access to only other static members, functions & variables declared in that class.

→ A SMF can be called using the colon syntax
i.e class name :: funct ();

Static Private & Public member function

```
Class count
    {   Static int c;
Public:
Static void input ()
{  c++; cout << c; }
};          int count :: c = 10;
```

Datatype pointer =
datatype pointer

delete. P;
delete. [P]

```
int main()
{
    count :: input ();
    return 0;
}
```

private Static member function

```
Class count
{   Static int c;
  static void input ()
  {  c++;
  }
  Public:
  static void show ()
  { input ();
    cout << c;
  }

  int count :: c = 10;
  int main ()
  {
    count :: show ();
    return 0;
  }
```

24/7/19

Assignment

~~SOAP to count the no. using static member function.~~

1. To cal. the area of rectangle using class & object. (datamember , member func.)
2. Overload the area method in ~~Shape~~ shape class to calculate area of circle, rectangle & triangle. // Outside the class

3. Input
Final
higher

3. Input details of n no. of students (branch, name, cgpa) & display them in tabular format & finally display the name of the student having highest cgpa.

Q) WAP to declare a local variable, global & static, public data member with the same name, initialize those data values and display that without using member function.

```
#include <iostream>
using namespace std;
int x=5;
class A
{
public:
  static int x;
};
  int A::x=10;
  main()
  { int x=6;
    cout<<x<<::x<<A::x;
  }
```

Q) How to pass object as an argument.

```
class A
{int c;
public:
  void input()
  { cin>>c;
  }
  void sum(A, A);
};
void A::sum(A o1, A o2)
{ cout<<o1.c+o2.c;
}
main()
{ A ob1, ob2;
  ob1.input(); //ob1.c
  ob2.input(); //ob2.c
  ob1.sum(ob1,ob2);
  ob2.sum(ob1,ob2);
}
```

datatype pointer = datatype pointer

delete p;
delete [5];

Q) Pass argument using call by reference & call by address.

Call by address

```
Class A
{
  int c;
  Public:
  void input()
  { cin >> c;
  }
  void sum(A*, A*);
  };
void A :: sum(A*O1, A*O2)
  { cout << *O1.c + *O2.c;
  }
main()
{
  A Ob1, Ob2;
  Ob1.input();
  Ob2.input();
  Ob1 sum(&Ob1, &Ob2)
```

call by reference

```
Class A
{ int c;
Public:
  void input()
  { cin >> c;
  }
  void sum(A&, A&);
  };
void A:: sum(A&O1, A&O2)
  { cout << O1.c + O2.c;
  }
```

```
main()
{   A Ob1, Ob2;
    Ob1.input();
    Ob2.input();
    Ob1.sin(&Ob1, &Ob1);
}
```

Friend function

```
Class A
{ int c;
  Public:
    void input()
    { cin >>c;
    }
  friend void sum(A&, A&);
  };
void sum(A O1, A O2)
{  cout << O1.c + O2.c;
}
main()
{  A Ob1, Ob2;
   Ob1.input(); // Ob1.c
   OB2.input(); //Ob2.c
      Sum(Ob1, Ob2);
   }
```

C++ has a mechanism by which a non member function can access the private member of a class to acheive this we should declare a function using the keyword friend inside the class definition there is no scope restriction for the friend func. hence they can be called without using object name & dot operator.

Datatype pointer
datatype po

delete.p;
delete.[

→ Unlike member func. of a class, friend func. can't access the private datamember of the class directly it uses object name & dot operator to access the private data member.

→ By default friendship is not shared

→ Use of friend func. is rarely done because it violates the rule of encapsulation & data hiding.

→ The friend func. can be declared under private or public access specifier without changing its meaning.

Q)

```
Class B;
Class A;
{ int C;
Public:
void input()
{ cin>>C;
}
friend void sum(A, B);
};
Class B
{ int d;
Public:
void input()
{ cin >>d;
}
friend void sum(A, B);
};
void (A O1, B O2)
{ cout << O1.C + O2.B.d;
}
```

Constructor

→ The constructor constructs the object.
→ When an object is created the compiler automatically executes the constructor function.
→ It is optional to declare a constructor if the programmer doesn't define the constructor the compiler executes implicit constructor

Characteristics of a constructor :-

Constructor is a special member func. whose name is same as the classname.
→ constructor should not have any return type nor even void.
Constructor func. is executed when an object is created.
The main func. of the constructor is to initialize the data object and allocate appropriate memory to the object
Constructor can have default argument & can be overloaded.
The constructor without any argument is called as default constructor.

Types of Constructor

(i) Default Constructor
(ii) Parameterized constructor
(iii) Copy constructor
(iv) Dynamic Constructor

Eg. of Default argument

```
Class A
int x;
{ public:
  A();
};
A::A()
{ cout << "Constructor is called";
    x = 2;
    cout << x;
}
main()
{ A obB;
}
```

Parameterized constructor

```
Class A
int x;
{ public:
   A(int);
};
A::A(int P)
{ cout << "cons. is called";
   x = P;
  cout << x;
}
main()
{ A ob(2);   // Implicit call
  A Ob1 = A(3);   // Explicit call
}
```

Datatype pointer = new datatype pointer;

delete p;
delete [];

1) WAP to declare a class with two 2 dim data type int & char. Initialize the DM using constructor.

```
Class A
int x;
char b;
{ Public
   A ();
   A(int,char);
};
   A :: A()
  { cout << " cons. is called";
    x = 2 ; b = 2;
   cout << x << b;
  }
   A :: A(int x, char p)
      {cout << " cons. with arg. is called";
x = p ; b = p;
      cout << b << x;
    }
main()
{       A Ob;
        A Ob1 = A(2,3);
}
```

Example of Constructor overloading

Declare a class with 3 dm of type int, float & char. Define 3 constructors with 3 arg., 2 arg., & 0 arg. resp. to initialize the dm of class & display the corresponding values.

```
Class A
{
 int x;
 float y;
 Char z;
  A (int, float char);
  A (int, float);
  A ( );
};
A::A(int a; float b; char z)
{
   x= a; y = b; z = c;
   cout << " constructor 1 is called" << x << y << z;
}
A::A(int a; float b)
  { x=a; y=b; z='l';
   cout << "2 " << x << y << z;
  }
A::A()
  { x = 1; y = 1.1; z = 'q';
   cout << "0" << x << y << z;
  }
main()
{
A ob = A(1, 1.1, 'q');
 A ob1 = A(1, 2.1);
 A ob2;
}
```

datatype pointer = datatype pointer

delete p;
delete [5];

Constructor with default argument

```
Class D
{
int a;
float b;
Public:
D(int p=10, float q=2.5)
  { a=p; b=q;
  cout << a <<b;
}
main()
{ A obj(6, 5.5);
  A obj2(3);
  A obj3;
```

1) WAP to find out power of a given no. using default argument.

```
#include<math.h>
Class A
{ int x,y;
 Public:
 A(int P=2; int q=3)
   { x=P;
     y=q;
   cout << x << y;
}
void call()
{ int a;
  a = pow(x, y);
  cout << a;
};
```

main()
{ A Ob
A
ob. cal
3

Lab 4

As

Define a cl
int type
→ float
the d
using
one
detai

Clas
{
P

Scanned by CamScanner

```
main()
{ A ob(5, 6);
  ob.cal();
}
```

lab 4

Assignment-

Define a class bank with private DM account no. → int type, account holder name → char array, balance → float type, min balance → static & float type. Initialize one dat for two objects in constructor & then using a friend func. Transfer some amount from one account to another account & display the details after the transfer.

Copy Constructor

```
Class C
{ int K;
Public:
  C()
{  K=5;
}
  C(int P)
{  K=P;
}
C(C&J)
{ K=J.K;
}
void show()
{  cout<<K;
}
};
```

```
main()
{ C ob1;
  ob1.show(); //5
C ob2(ob1);
  ob2.show(); //5
C ob3(7);
ob3.show(); //7
C ob4=ob3; //call copy constru.
ob4.show();
  C ob5;
ob5=ob3; //won't call copy constructor
ob5.show();
```

Datatype pointer = datatype pointer

delete p;
delete [5];

* It is possible to pass reference of object to the constructor such declaration is called as copy constructor.
* Whenever a constructor is called a copy of an object is created

- Dynamic Constructor

```
int *p = new int (5);
      OR
int *p;
p = new int;
*p = 5;
```

Q) WAP to allocate memory to an integer variable using dynamic constructor

```
Class D
{
  int *p;
  Public:
  D()
  { p = new int;
    *p = 6;
  }
  D(int k)
  { p = new int;
    *p = k;
  }
  int display()
  { return (*p);
  } ~D() { delete p; }
};
```

```
main ()
{
    D ob1;
cout << ob1.display ();      //6

    D ob2(8);
    cout << ob2.display();   //8
}
```

Destructor

The general syntax of a destructor is
~D() { delete (*p)}

Destructor is a member function which destroys the object. Destructor name name is same as class name with '~' sign.

General Syntax → ~classname ();

A Destructor function is called when the object goes out of the scope automatically by the compiler. Destructor does not take any argument and does not return any value.

There can only be one destructor in a class.

If we do not write our own destructor in class, compiler calls the default destructor. Default destructor works fine unless we have dynamically allocated memory or pointer in a class.

→ When a class contains a pointer to memory allocated in class, we should write a destructor func. to release the memory.

8) WAP to create a class to initialize a string using dynamic constructor.

```
Class A
{
char *s; int size;
Public: A(char *); // constructor
        {
        s = new char [size+1];
        strcpy(s, c);
        }
void display()
{ cout << s;
}
~A(); // Destructor
};
A :: A(char *c)
{
  size = strlen(c);
  s = new char [size+1];
```

```
Char st
{ char *s; int size;
Public : st(char *); // Constr
   void display()
   { cout << s; }
   ~st(); // Destr.
};   st :: st(char *c)
     {
       size = strlen(c);
     s = new char [size+1];
       strcpy(s, c);
     }
```

→ We can
keyword
→ Only da
member
→ The
called
valu
→ The
mem

```
st::~st()
{ delete [s];
}
main()
{ st obj("XYZ");
  Obj.display();
}
```

Constant Object

→ We can create & use constant objects using const keyword before object declaration.

→ Only constructor can initialize the data member variables of constant object.

→ The data members of constant objects are also called as read only data member because their values can't be changed.

→ The constant object can only access constant member function.

```
Class C
{ int K;
Public:
 C(int P)
{ K=P;
 }
void show() const
 {
  cout<<K;
 }
};
```

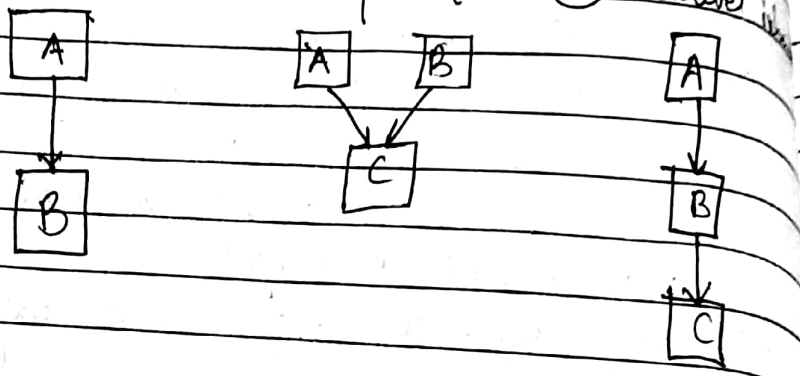
```
main()
{ const C ob(5);
    ob.show();
}
```

Inheritance

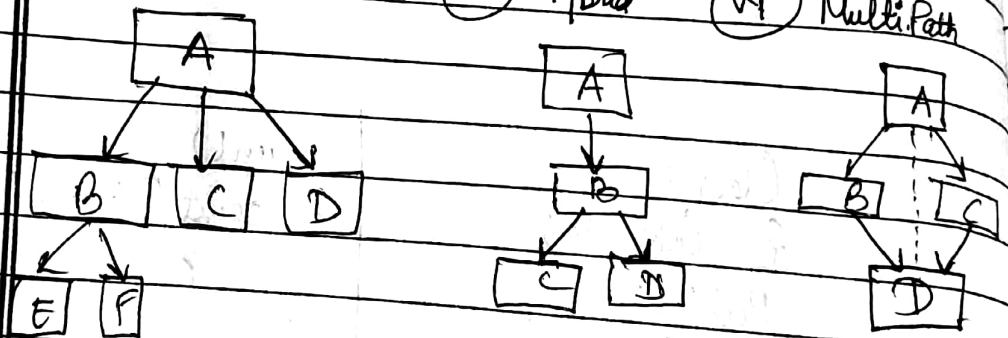
→ The procedure of creating a new class from an existing class is termed as inheritance.
→ The existing class is called as the base class and the new class is called as the derived class.

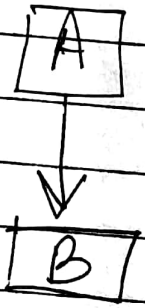
Types of Inheritance

(I) Single inheritance (II) Multiple inheritance (III) Multilevel inheritance



(IV) Hierarchical (V) Hybrid (VI) Multi Path





Syntax :-

```
Class A
{
};
Class B :- Public A
{
};
```

Base class visibility	Derived class visibility		
	Public mode	Private mode	Protected mode
Private	Not inherited	Not inherited	Not inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

Q) WAP to create a class A with a public data member of type integer. Derive a class B in public mode from class A. Initialize the data of class A & B using derive class object.

```
Class A
{
Public : int a;
};
Class B: Public A
{ Public: int b;
};

main ()
{
    B ob;
ob.a = 5;
ob.b = 6;
cout << ob.a << ob.b;
}

Class A
{ Protected:
    int x;
};
```

```
Class B : Public A
{
    int y;
Public:
    void input()
    { cin>>x>>y;
      cout<<x<<y;
    }
};
main()
{
  B ob;
  ob.input();
}
```

```
Class A
{ Private:
    int x;
Public: A()
    { x=5; cout<<x;
    }
};
Class B : Public A
{ Private:
    int y;
Public: B()
    {
      y=6; cout<<y;
    }
};
main()
{
  B ob;
}
```

O	P
5	6

Base class initialized first

```
Class A
{  Public:
       int x;
};
Class B: Private A
{   Private:
      cnt y;
    Public:
      B()
      {   x = 4;
          y = 6;
        cout << y << x;
      }
};
main ()
{   B ob;
```

Private Constructor & Destructor

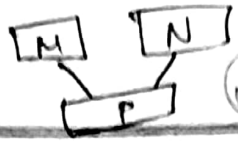
```
Class A
{ int x;
  A()
  { x=5;
  cout << "constructor" << x ; }
}
  ~A()
  cout << " Destructor";
  }
Public :
  void disp ()
    {  this->A::A ();
       this-> A::~A ();
    }
};
```

```
main ()
{
  A *ob;
  ob -> disp ();
}
```

Q

```
Class A
{ Protected:
    int a;
};
Class B : Public A
{ Protected:
    int b;
};
Class C : Public B
{ Protected:
    int c;
Public: void input()
{ cin >> c >> b >> a;
cout << a << b << c; };

main ()
{ C ob;
  ob input ();
}
```

Q WAP to create a class student with dm name & roll no. & a member func. input(). to take the input & add a display func. Derive a class exam with DM Mark1, Mark2 and if member func. get data ito take the input & data to display mark Derive a class result with DM total & a member func. to calculate the total marks of a particular student



Create a class M with a data x of type int, create another class N with a data y of type int. Derive a class P from both M & N through the member func. of P initialize the data of class M & N and display the corresponding values.

```
Class M:
{ Protected:
  int x;
};
Clas N:
{ Protected:
  int y;
  };
Class P:: Public M, Public N
{ Public:
   void input ()
   {
    cin >> x >> y;
    cout << x << y;
   }
main ()
P ob;
 ob. input ();
```

```
Class M:
{ Protected : int x;
Public: void input ()
   { cin >> x; }
  };
Class N
{ Protected : int y;
Public : void input ()
     { cin >> y; }
  };
```

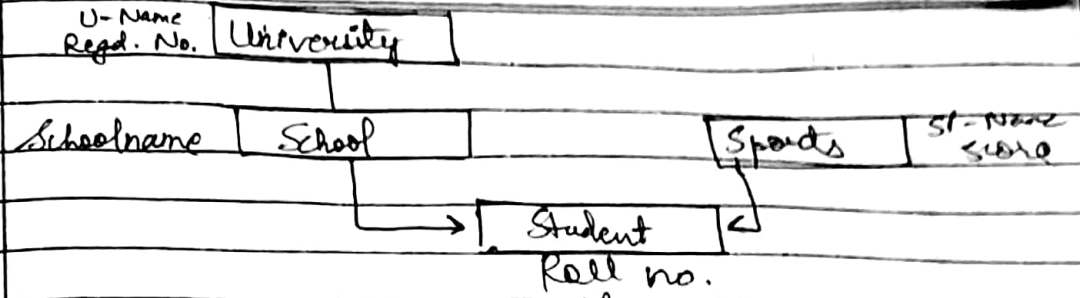
```
Class P:: Public M, Public N
{ int z;
  Public : void input()
           {
             cin >> z;
  M :: input();
  N :: input(); cout << x << y << z;
           }
};

main()
  P ob;
  ob. input();
```

Ambiguity Resolution in

7/Aug.

Q1 Define class human with public data member name and age and public member function read human() and display Human() to read and display details of human. Publicly inherit a class Student from Human. The Student class will contain additional data member roll, branch & mark and member func. readStudent() & displayStudent() to read and display the details of Student. Write a main func. to read and display the details of a student.

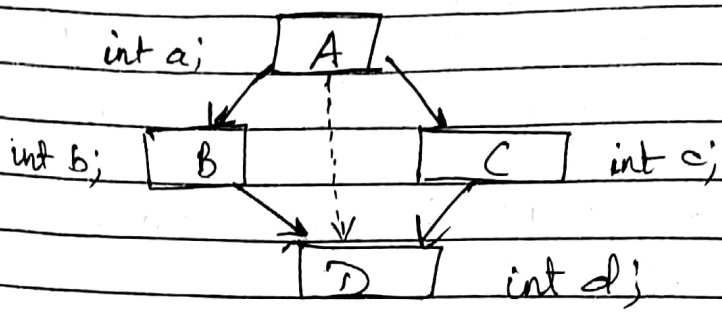
Q2 Define a class staff with DM staff_ID & LEVEL and mf. to read and display details of staff. Design another class Teacher with dm TEACHER_ID & subject and mf to read & display the details of Teacher. Design another class coordinator that inherits the TEACHER & STAFF with own dm PROGRAM and mf to read & display details. Write a main func. to implement above classes




```
class University
protected:
    char uname[];
    int regdno;
};
class school : Public university
Protected :
    char sname [];
Public:
    void input2 ()
    { cin >> sname;
};
class sports
{ Protected :
    char spname [];
    int score ;
};
class Student : Public school, Public sports
{ Protected:
    int Rollno.;
    char name [];
void input ()
{   cin >> rollno. >> name >> spname >> score;
    cin >> sname >> uname >> regdno;
cout << rollno. << name << spname << score << sname;
    cout << uname << regdno.;
};
```

```
int main()
{
student ob;
  ob.input();
return;}
```

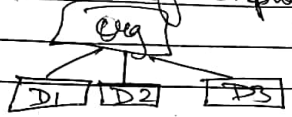
Multipath Inheritance (One Virtual base class)




```
Class A
{  protected:
   int a;
};
  Class B: Public virtual A
  {  protected:
       int b;
  };
Class C: virtual Public A
  {  Protected:
       int c;
  };
Class D : Public B, Public C
  {
    Protected:
    int d;
    Public:
Void input()
  { cin >>a>>b>>c>>d;
    cout <<a<<b<<c<<d;
  };
```

```
int main()
{
    D ob;
    ob.input();
    return 0;
}
```

Q) Create a class organisation with dm orgname & orgregdno. Derive three classes department 1, department 2, & department 3, with dm dept. name, id & no. of employees working in that dept.




```
Class org
{ Protected:
    char orgname[];
    int orgregdno;
};
```

```
Public : void input1()
{ cin >> orgname >> orgregdno;
  cout << orgname << orgregdno; }
```

```
Class dept1 : Public org
{
Protected:
char deptname[];
  int id;
  int n;
Public:
  void input2()
  { cin >> deptname >> id >> n;
    input1();
    cout << deptname << id << n; }
  };
char dept2: Public org
{ Protected: chardeptname[]; intid; int n;
Public: void input3()
{ cin >> deptname >> id >> n;
    input1();
    cout << deptname << id << n; } };
```

```
class dept3: public org
{ protected:
  char deptname[];
  int id;
  int n;
  Public:
  void input4()
  { cin >> deptname >> id >> n;
    input1()
    cout << deptname << id << n;
    }
};

int main()
dept1 ob1;
dept2 ob2;
dept3 ob3;
      ob1.input2();
      ob2.input3();
      ob3.input4();
return 0; }
```

→ As long as no base class construction takes any argument, the derived class did not have a constructor func.

→ If any base class constructor contains 1 or more argument then it is compulsary for the derived class to have constructor and pass argument to the base class.

→ While applying inheritance we usually create obj of the derived class that help make it sense for the derived class to pass arguments to the base class constructor.

→ When both the derived and base classes contains constructor, the base class constructor is executed first then the derived class constructor is executed.

In case of multiple Inheritance, the base classes are constructed in order in which they appear in the declaration of the derived class.

```
Class A : Public B                    B(), A()
{

};
```

	Order
Class A : Public B, Public C	B(), C(), A()
Class A : Public C, Public B	C(), B(), A()
Class A : Public B, virtual Public C	C(), B(), A()

Syntax of Deriving Constructor (Parametrized)

```
Derive ( Arg1, Arg2, Arg3) : Base(Arg1), Base1(Arg2)
{

};
```

Q WAP to create a class A with Dm x and through parametrized constructor initialize the dm. Create another class B with float data member y and initialize y with a parametrized constructor class B. Derive a class C from class A and B with a dm m & n of type. Through parametrized constructor initialize the value of m & n.

```
class A
{
protected:
int x;
A(int a)
{
x = a;
}};
class B
{
protected:
float y;
B(float b)
{
y = b;
}
};
Class C : Public A, Public B
{
int m, n;
Public:
C(int j, float k, int s, int t) :A(j), B(k)
{
m = s;
n = t;
}
void disp ()
{
cout << x << y << m << n;
}
};
int main ()
{ C ob(2, 3.5, 4, 5);
ob.display();
return 0;
}
```

9) WAP to create a class α with a data member int x. Create another class β with that dm pkg. Initialize p with the same value that is passed from the main fun. and initialize q with a value more than three of the marginal value. Derive the class γ with base classes α and β with dm s & r of type int.

```
Class α
{
Protected:
int x;
α (int a)
{
  x=a;
}
};
Class β
{
Protected:
float p,q;
β (float b, float c) : q(c+3)
{
   p=b;
}
}
Class γ: public α, public β
{ int S, R;
Public:
γ( int m, float o, float p, int t; int u)
                : α(m), β(o,p)
{ S=t;
   R=u;
}
```

```
void show()
 {
   cout << x, P, O, S, R;
 }
main()
{  x ob(1, 2, 3, 4, 5);
   ob.show()
   }
```

Nested Class

```
Class A
{
Public:
int x;
A()  { x=5; }
 ClassB
 {
  Public:
   Int y;
 B()
 { y=6; }
 void show()
  { A ob;
  cout << ob.x << y;
   }
 };
};
main()
{
  B A; B b; → // b obj. of class B.
  b.show();
}
```

8) WAP to define a class to take the input for a string using dynamic constructor join two strings of that class using a mf.

```
Class A
{
  char *s;
  int size;
  Public :
  st (char *c)
  {
    size = strlen(c);
    s = new char[size+1];
    strcpy(s,c);
  }
  st()
  { size=0;
    name = new char[length+1];
  }
  void display()
  { cout << name; }
  void join (st &a, st &b);
};
void st :: join (st &a, st &b)
{ length = a.length + b.length;
  delete s;
  s = new char[size+1];
  strcpy(s, strcat(a.s, b.s));
}
```

```
st()
{
  char *c
  cin >> c;
  size = strlen(c);
  s = new char[size+1];
  strcpy(s, c);
}
```

```
main()
{
  st s1("KIIT"), s2("Uni");
   st ob;
  ob.join(s1, s2);
  s1.display();
  s2.display();
   ob1.display();
}
```

Application of OOP
Benefits of OOP
Advantage & disadvantage of OOP

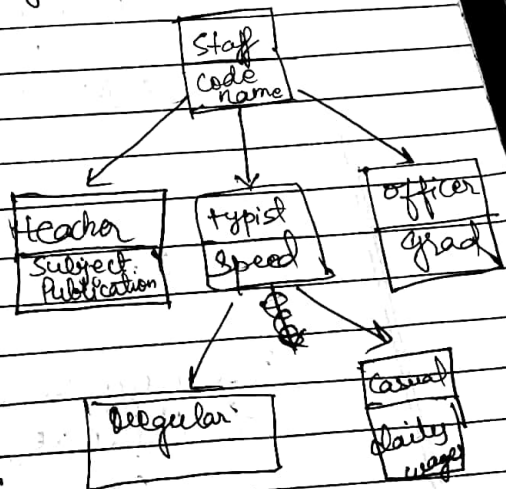
Q WAP to count the no. of times constructors and destructors are called using static dm.

```
Class A
static int C;
  A()
  { cout << "constructor is called";
    c++;
  }
  ~A()
  { cout << "\n" "Destructor is called";
    c--;
  }
};  A::C = 0;
main()
{  A ob;
cout << ob.C;
}
```

Abstract Class

An abstract class is one that is not used to create objects.
An abstract class is designed only to act as a base class (to be inherited by other classes). It is a designed concept in program development and provides a base upon which other classes may be built.

OOAP.

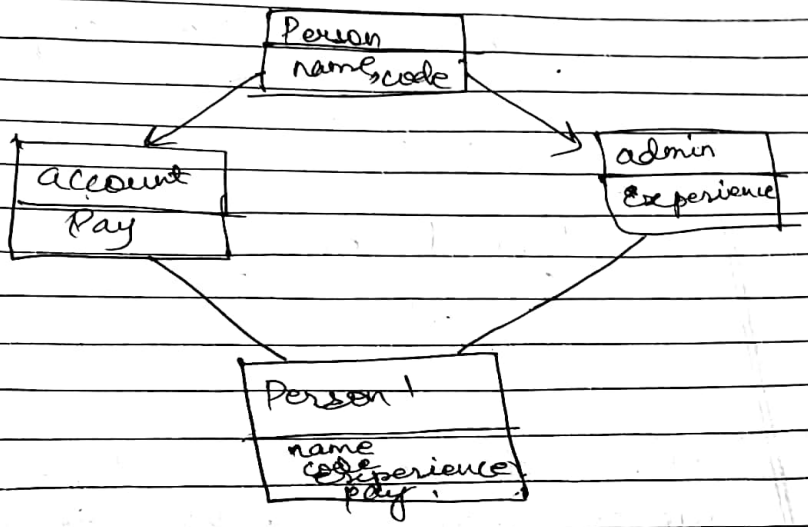



```
#include<iostream>
using namespace std;
Class staff
{ Public:
char code name
Public:
void input ()
{ cin >> code name;
cout<< code name;
}
};
Class teacher : Public staff
{ char subject;
  char Publication;
  void input ()
  { cin >> subject >> Publication;
  cout<< subject << Publication
  input ();
  }
};
```

```
Class typist : Public staff
{ Protected:
    int speed;
Public: void input 2();
    { cin >> speed;
      input ();
    }
};

Class officer : Public staff
{ Protected:
    int grade;
Public: void input 3()
    { cin >> grade;
      input ();
    }
};

Class regular: Public typist
{ Public:
    void input 2();
    { cin >> speed;
      cout << speed;
input(); }
};
Class casual : Public typist
{ Protected:
    int daily wages;
  Public:
    input ();
    void input 4();
    { cin >> speed;
      cout << speed;
    }
```

```
int main()
{  teacher a ; regular b ; casual c; offica d;
a.input1()
d.input3()
b.input2()
c.input4()
return 0;
}
```

Q2




```
Class Person
{ Protected:
  Char name;
  int code;
  Public:
    void input()
    { cin>>name>>code;
      cout<<name<<code;
    }
};
Class accout: public virtual person
{ Protected: int pay;
  void input1()
{ cin>>pay;
```

```
Class admin: Public virtual Person
{ Protected:
    float experience;
};

Class Person1: Public account, Public admin
{ Public: void input();
    cin >> name >> code >> experience >> pay;
    cout << name << code << experience << pay;
};

int main()
{
    Person1 ob;
    ob.input();
    return 0;
}
```

3) WAP to create two classes DK & DM to store the value of distance. Class DK stores distances in km & m. Class DM stores distances in m & cm. Use a friend func. to perform addition of two distances stored in DK and DM. The display should be in the format of km & m or m & cm.

```
Class DM;
Class DK
{ Private: int km, m;
  Public:
    void input1()
    { cin >> km >> m;
      cout << km << "km" << m << "m";
    } friend void sum (DK, DM);
};
```

```
Class DM
{ protected :
    int m, cm;
  Public :
    void input2()
    { cin >> m >> cm;
      cout << m << "m" << cm << "cm";
    }
    Friend void sum (DK, DM);
};

void sum (DK O1, DM O2)
  { int a, b;
a = O1. Km * 1000;
b = O1. m * 100;
  cout << a + O2. m << "m" << b + O2. cm <<
}
int main()
{ DK O1;
  DM O2;
  O1. input1();
  O2. input2();
  sum (O1, O2);
  return 0;
}
```

8)
```
Class A
{
Public:
int a;
A()
{
  a=10;
}
};
Class B : Public A
{ Public:
  int b;
  B()
  {
    b=20;
  }
 ~B()
  {
  cout<<a<<b;
  }
};
int main()
{
  B();
return 0;
}
```

⇒ execution

O/P ⇒ 10 20

8)
```
class A
{
  int x;
};
Class B : A
{
  int y;
};
  main()
  {
    B b;
  }
```

Ans ⇒ Class A is an abstract class as object of class B is created and B A can't be used because it is private

9)
```
class A
{
 Public:
 int a;
 Private:
  int b;
 Public:
  A()
  {
  a=10; b=20; cout<<a<<b;
  }
};
int main
{ A A;
 return 0;
}
```

⇒ created ob → 10 20