INHERITANCE

- The procedure of creating a new class from one or more existing classes is called inheritance.
- The program can defined new member variables and function in the derived class .
- The base class remain unchanged.
- An object of derived class can access members of base as well as derived class but reverse is not possible.
- The term reusability means reuse of properties of base class in the derived class.

7.1Types of Inheritance

7.1.1 Single Inheritance: It is the inheritance hierarchy wherein one derived class inherits from one base class.

Program- Write a program to show single inheritance.

```
#include<iostream.h>
#include<conio.h>
class student
{
    public:
    int rno;
    //float per;
    char name[20];
    void getdata()
    {
            cout<<"Enter RollNo :- \t";
            cin>>rno;
            cout<<"Enter Name :- \t";
            cin>>name;

    }
};
class marks : public student
{
public:
    int m1,m2,m3,tot;
    float per;
    void getmarks()
    {
            getdata();
            cout<<"Enter Marks 1 :- \t";
            cin>>m1;
            cout<<"Enter Marks 2 :- \t";
            cin>>m2;
            cout<<"Enter Marks 2 :- \t";
            cin>>m3;
    }
    void display()
```

```
    {
            getmarks();
            cout<<"Roll Not \t Name \t Marks1 \t marks2 \t Marks3 \t Total \t Percentage";
            cout<<rno<<"\t"<<name<<"\t"<<m1<<"\t"<<m2<<"\t"<<m3<<"\t"<<tot<<"\t"<<per;
    }
};
void main()
{
    student std;
    clrscr();
    std.getmarks();
    std.display();
    getch();
}
```

7.1.2 Multiple Inheritance: It is the inheritance hierarchy wherein one derived class inherits from multiple base class(es)

Program - Write a program calculates the area and perimeter of an rectangle but, to perform this program, multiple inheritance is used.

```
#include <iostream>
using namespace std;
class Area
{
 public:
   float area_calc(float l,float b)
   {
      return l*b;
   }
};

class Perimeter
{
 public:
   float peri_calc(float l,float b)
   {
      return 2*(l+b);
   }
};

/* Rectangle class is derived from classes Area and Perimeter. */
class Rectangle : private Area, private Perimeter
{
   private:
      float length, breadth;
   public:
     Rectangle() : length(0.0), breadth(0.0) { }
```

```
    void get_data( )
    {
       cout<<"Enter length: ";
       cin>>length;
       cout<<"Enter breadth: ";
       cin>>breadth;
    }

    float area_calc()
    {
    /* Calls area_calc() of class Area and returns it. */

       return Area::area_calc(length,breadth);
    }

    float peri_calc()
    {
    /* Calls peri_calc() function of class Perimeter and returns it. */

       return Perimeter::peri_calc(length,breadth);
    }
};

int main()
{
   Rectangle r;
   r.get_data();
   cout<<"Area = "<<r.area_calc();
   cout<<"\nPerimeter = "<<r.peri_calc();
   return 0;
}
```

7.1.3 Hierarchical Inheritance: It is the inheritance hierarchy wherein multiple subclasses inherit from one base class.

Program- Write a program to implement hierarchical inheritance.

```
#include<iostream.h>
#include<conio.h>
class A
{
   public:
   int a,b;
   void getnumber()
   {
   cout<<"\n\nEnter Number :::\t";
   cin>>a;
   }
```

```
};

class B : public A
{
   public:
   void square()
   {
   getnumber();
   cout<<"\n\n\tSquare of the number :::\t"<<(a*a);

   }
};

class C :public A
{
   public:
   void cube()
   {
   getnumber(); //Call Base class property
   cout<<"\n\n\tCube of the number :::\t"<<(a*a*a);

   }
};

int main()
{
clrscr();

B b1;
b1.square();
C c1;
c1.cube();

getch();
}
```

7.1.4 Multilevel Inheritance: It is the inheritance hierarchy wherein subclass acts as a base class for other classes.

Program – Write program to implement multilevel inheritance.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class student    // base class
{
private:
```

```
  int rl;
  char nm[20];
public:
   void read();
   void display();
};
class marks : public student // dervivec from student
{
protected:
  int s1;
  int s2;
  int s3;
public:
   void getmarks();
   void putmarks();
};
class result : public marks // derived  from marks
{
private:
  int t;
  float p;
  char div[10];
public:
   void process();
   void printresult();
};
   void student::read()
   {
    cout<<"enter Roll no and Name "<<endl;
    cin>>rl>>nm;
   }
   void student:: display()
   {
   cout <<"Roll NO:"<<rl<<endl;
   cout<<"name : "<<nm<<endl;
   }
   void marks ::getmarks()
   {
   cout<<"enter three subject marks  "<<endl;
    cin>>s1>>s2>>s3;
   }
   void marks ::putmarks()
   {
    cout <<"subject 1:"<<s1<<endl;
    cout<<" subject 2 : "<<s2<<endl;
```

```
 cout <<"subject 3:"<<s3<<endl;
}

void result::process()
{
 t= s1+s2+s3;
 p = t/3.0;
 p>=60?strcpy(div,"first"):p>=50?strcpy(div, "second"): strcpy(div,"third");
}

void result::printresult()
{
 cout<<"total = "<<t<<endl;
 cout<<"per  = "<<p<<endl;
 cout<<"div = "<<div<<endl;
}
void main()
{
 result x;
 clrscr();
 x.read();
 x.getmarks();
 x.process();
 x.display();
 x.putmarks();
 x.printresult();
 getch();
}
```

7.1.5 Hybrid Inheritance: The inheritance hierarchy that reflects any legal combination of other four types of inheritance.

Program- Write a program to implement hybrid inheritance

```
#include<iostream.h>
#include<conio.h>
class arithmetic
{
protected:
int num1, num2;
public:
void getdata()
{
cout<<"For Addition:";
cout<<"\nEnter the first number: ";
cin>>num1;
cout<<"\nEnter the second number: ";
```

```
cin>>num2;
}
};
class plus:public arithmetic
{
protected:
int sum;
public:
void add()
{
sum=num1+num2;
}
};
class minus
{
protected:
int n1,n2,diff;
public:
void sub()
{
cout<<"\nFor Subtraction:";
cout<<"\nEnter the first number: ";
cin>>n1;
cout<<"\nEnter the second number: ";
cin>>n2;
diff=n1-n2;
}
};
class result:public plus, public minus
{
public:
void display()
{
cout<<"\nSum of "<<num1<<" and "<<num2<<"= "<<sum;
cout<<"\nDifference of "<<n1<<" and "<<n2<<"= "<<diff;
}
};
void main()
{
clrscr();
result z;
z.getdata();
z.add();
z.sub();
z.display();
```

```
getch();
}
```

7.1.6 Multipath Inheritance :

When a class is derived from two or more classes that are derived from same base class . such type of inheritance is known as multipath inheritance.

Program- Write a program to implement multipath inheritance

```
#include<iostream.h>
#include<conio.h>
class student
{
protected:
    int roll;
    char nm[30],branch[20],sem[10];
};
class exam:virtual public student
{
protected:
    int m1,m2,m3,m4,m5,m6;
};
class sport:virtual public student
{
protected:
    char gd[10],sp[10];
};
class result:public exam,sport
{
protected:
    int total;
public:
    void in()
    {
    cout<<"enter name,roll,branch,semester & 6 subject mark:";
    cin>>nm>>roll>>branch>>sem>>m1>>m2>>m3>>m4>>m5>>m6;
    total=(m1+m2+m3+m4+m5+m6);
    cout<<"enter sports name & grade:";
    cin>>gd>>sp;
    }
    void out()
    {
    cout<<"\n name="<<nm<<"\n roll="<<roll<<"\n branch="<<branch<<"\n semester="<<sem;
    cout<<"\n total="<<total;
    cout<<"\n sports="<<sp<<"\n grade="<<gd;
    }
};
void main()
{
```

```
clrscr();
result r;
r.in();
r.out();
getch();
}
```

7.2 Container class

- Declaring object of one class as data member in another class is known as deligation.
- A class that has objects of another class as its data member is known as container class
- This kind of relationship is known as has-a-relationship or containership.

Program- Write a program to implement container class

```
#include<iostream.h>
#include<conio.h>
class door
{
public:
        int dc,dn;
        void in()
        {
        cout<<"enter no.of doors:";
        cin>>dn;
        dc=2000*dn;
        cout<<dc;
        }
};
class windo
{
public:
        int wc,wn;
        void in()
        {
        cout<<"enter no.of windows:";
        cin>>wn;
        wc=1500*wn;
        cout<<wc;
        }
};
class house
{
public:
        door d;
        windo w;
        long int hc,rn,rc;
        house()
```

```
        {
        d.in();
        w.in();
        cout<<"enter the no. of rooms and price of room:";
        cin>>rn>>rc;
        hc=d.dc+w.wc+rn*rc;
        cout<<"price="<<hc;
        }
        ~house()
        {
        }
};
void main()
{
clrscr();
house h;
getch();
}
```

7.3 Abstract class

- When a class is not used for creating objects, it is called as abstract class.
- Abstract class can act only as base class.

7.4 Common constructor

- When constructor of derived class is used to initialize the data members of base class as well as derived class, known as common constructor.

7.5 Pointer & Inheritance

```
#include<iostream.h>
#include<conio.h>
class A
{
Protected:
int x,y;
public:
    A()
    {
    x=1; y=2;
    }
};
class B:private A
{
Public:
    Int z;
    B()
    {
    Z=3;
```

```
    }
};
void main()
{
clrscr();
B b;
int *p;
obj.in();
p=&b.z;
cout<<(unsigned)p<<*p;
p--;
cout<<(unsigned)p<<*p;
p--;
cout<<(unsigned)p<<*p;
}
```

7.6 Overloading Member function

```
#include<iostream.h>
#include<conio.h>
class B
{
public:
    void show()
    {
    cout<<"in base class";
    }
};
class D:public B
{
public:
    void show()
    {
    cout<<"in derived class";
    }
};
void main()
{
clrscr();
B b;
D d;
b.show();
d.show();
d.B::show();
getch();
}
```

OPERATOR OVERLOADING

8.1 Introduction

- The capability to relate the existing operator with a member function and use the resulting operator with objects of its class as its operands is called operator overloading.

  Program – Write a program to perform addition of two objects

```
#include<iostream.h>
#include<conio.h>
class number
{
public:
int x,y;
number()
{        }
number(int j,int k)
{
x=j;
y=k;
}
numberoperator +(number D)
{
number T;
T.x=x+D.x;
T.y=y+D.y;
return T;
}
void show()
{
cout<<"x="<<x<<"y="<<y;
}
};
void main()
{
clrscr();
number A(2,3),B(y,5),c;
A.show();
B.show();
C=A+B;
C.show();
getch();
}
```

8.2 Overloading Unary operator

/* Write a program to overload unary operator */

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

class complex
{
    int a,b,c;
   public:
      complex(){}
      void getvalue()
     {
            cout<<"Enter the Two Numbers:";
            cin>>a>>b;
     }



  void operator++()
     {
            a=++a;
            b=++b;
     }

     void operator--()
     {
            a=--a;
            b=--b;
      }

      void display()
      {
            cout<<a<<"+\t"<<b<<"i"<<endl;
       }
};

void main()
{
    clrscr();
    complex obj;
    obj.getvalue();
    obj++;
    cout<<"Increment Complex Number\n";
    obj.display();
    obj--;
    cout<<"Decrement Complex Number\n";
    obj.display();
    getch();
}
```

8.3 Overloading binary operator using friend function

```
#include<iostream.h>
#include<conio.h>
class num
{
private:
    int a,b,c,d;
public:
    void in()
    {
    cout<<"enter a value of a,b,c,d:";
    cin>>a>>b>>c>>d;
    }
    void show();
    friend num operator *(int,num);
};
void num::show()
{
cout<<a<<b<<c<<d;
}
num operator *(int a,num t)
{
num tmp;
tmp.a=a*t.a;
tmp.b=a*t.b;
tmp.c=a*t.c;
tmp.d=a*t.d;
return(tmp);
}
void main()
{
clrscr();
num x,z;
x.in();
x.show();
z=3*x;
z.show();
}
```

8.4 Overloading increment & decrement operator

```
#include<iostream.h>
#include<conio.h>
class number
{
float x;
public:
```

```
    number(float k)
    {
    x=k;
    }
    void operator ++(int)
    {
    x++;
    }void operator--()
    {
    --x;
    }
    void show()
    {
    cout<<x;
    }
};
void main()
{
clrscr();
number N(2,3);
N.show();
N++;
N.show();
--N;
N.show();
getch();
}
```

8.5 Overloading special operator

```
#include<iostream.h>
#include<conio.h>
class num
{
int a[3];
public:
    num (int i,int j, int k)
    {
    a[0]=i;
    a[1]=j;
    a[2]=k;
    }
    int operator [ ](int i)
    {
    return a[i];
    }
```

```
};
void main()
{
clrscr();
num ob(1,2,3);
cout<<ob[1];
cout<<ob[2];
cout<<ob[3];
getch();
}
```

8.6 Overloading function call operator

```
#include<iostream.h>
#include<conio.h>
class loc
{
int longitude,latitude;
public:
    loc() { }
    loc(int lg,int lt)
    {
    longitude=lg;
    latitude= lt;
    }
    void show()
    {
    cout<<longitude<<latitude;
    }
    loc operator() (int i,int j);
};
loc loc::operator()(int i,int j)
{
longitude= i;
latitude=j;
return *this;
}
void main()
{
clrscr();
loc ob1(10,20);
ob1.show();
ob1(7,8);
ob1.show();
}
```

8.7 Overloading class member access operator

```
#include<iostream.h>
#include<conio.h>
class num
{
public:
    int i;
    num *operator->()
    {
    return this;
    }
};
void main()
{
num ob;
ob->i=10;
cout<<ob.i<<  <<ob->i;
}
```

8.8 Overloading comma operator

```
/* Write a program to overload comma operator*/
#include<iostream.h>
#include<conio.h>
class loc
{
int longitude,latitude;
public:
    loc() { };
    loc(int lg,int lt)
    {
    longitude=lg;
    latitude=lt;
    }
    void show()
    {
    cout<<"\n longitude="<<longitude<<"latitude="<<latitude;
    }
    loc operator +(loc op2);
    loc operator,(loc op2);
};
loc loc :: operator,(loc op2)
{
loc temp;
temp.longitude=op2.longitude;
temp.latitude=op2.latitude;
return temp;
```

```
}
loc loc::operator +(loc op2)
{
loc temp;
temp.longitude=op2.longitude+longitude;
temp.latitude=op2.latitude+latitude;
return temp;
}
void main()
{
clrscr();
loc ob1(10,20),ob2(5,30),ob3(1,1);
ob1.show();
ob2.show();
ob3.show();
ob1=(ob1,ob2+ob2,ob3);
ob1.show();
getch();
}
```

8.9 Overloading stream operator

```
#include<iostream.h>
#include<conio.h>
class mat
{
public:
    int r,c,a[30][30],i,j;
    void in()
    {
    cout<<"\nEnter row & column size";
    cin>>r>>c;
    cout<<"\nEnter"<<r*c<<" elements of matrix:";
    for(i=0;i<r;i++)
    {
    for(j=0;j<c;j++)
    cin>>a[i][j];
    }
    }

    friend ostream & operator << (ostream &,mat &);
    friend istream & operator >> (istream &,mat &);

};
istream & operator>>(istream &in, mat &m)
{
cout<<"\nEnter row & column size:";
```

```
cin>>m.r>>m.c;
cout<<"\nEnter "<<m.r*m.c<<" elements :";
   for (int i = 0; i < m.r; ++i)
   {
     for (int j = 0; j < m.c; ++j)
        in >> m.a[i][j];
   }
   return in;
}
ostream & operator<<(ostream &out, mat &m)
{
   for (int i = 0; i < m.r; ++i)
   {
     for (int j = 0; j < m.c; ++j)
      out << m.a[i][j] << "  ";
     out << endl;
   }
   return out;
}
void main()
{
clrscr();
mat p,q;
p.in();
q.in();
mat M1;
cin>>M1;
cout<< M1;
getch();
}
```