Module-2:

CLASS:-

Class is a group of objects that share common properties and relationships .In C++, a class is a new data type that contains member variables and member functions that operates on the variables. A class is defined with the keyword class. It allows the data to be hidden, if necessary from external use. When we defining a class, we are creating a new abstract data type that can be treated like any other built in data type.

Generally a class specification has two parts:-

a) Class declaration
b) Class function definition

the class declaration describes the type and scope of its members. The class function definition describes how the class functions are implemented.

Syntax:-

```
class class-name
    {
    private:
            variable declarations;
            function declaration ;
    public:
            variable  declarations;
            function declaration;
            };
```

The members that have been declared as private can be accessed only from with in the class. On the other hand , public members can be accessed from outside the class also. The data hiding is the key feature of oops. The use of keywords private is optional by default, the members of a class are private.

The variables declared inside the class are known as data members and the functions are known as members mid the functions. Only the member functions can have access to the private data members and private functions. However, the public members can be accessed from the outside the class. The binding of data and functions together into a single class type variable is referred to as encapsulation.

Syntax:-

```
class item
{
        int member;
        float cost;
public:
        void getldata (int a ,float b);
        void putdata (void);
```

The class item contains two data members and two function members, the data members are private by default while both the functions are public by declaration. The function getdata() can be used to assign values to the member variables member and cost, and putdata() for displaying their values . These functions provide the only access to the data members from outside the class.

CREATING OBJECTS:

Once a class has been declared we can create variables of that type by using the class name.

Example:

```
item x;
```

creates a variables x of type item. In C++, the class variables are known as objects. Therefore x is called an object of type item.

item x, y ,z also possible.

```
class item
{
-----------
-----------
-----------
}x ,y ,z;
```

would create the objects x ,y ,z of type item.

ACCESSING CLASS MEMBER:

The private data of a class can be accessed only through the member functions of that class. The main() cannot contains statements that the access number and cost directly.

Syntax:

```
object name.function-name(actual arguments);
```

Example:- x. getdata(100,75.5);

It assigns value 100 to number, and 75.5 to cost of the object x by implementing the getdata() function .

similarly the statement

x. putdata ( ); //would display the values of data members.

x. number = 100 is illegal .Although x is an object of the type item to which number belongs , the number can be accessed only through a member function and not by the object directly.

Example:

```
class xyz
{
        Int x;
        Int y;
public:
        int z;
};
---------
----------
xyz p;
p. x =0;        error . x is private
p, z=10;        ok ,z is public
```

DEFINING MEMBER FUNCTION:

Member can be defined in two places

- Outside the class definition
- Inside the class function

OUTSIDE THE CLASS DEFINAT1ON;

Member function that are declared inside a class have to be defined separately outside the class.Their definition are very much like the normal functions.

An important difference between a member function and a normal function is that a member function incorporates a membership.Identify label in the header. The 'label' tells the compiler which class the function belongs to.

Syntax:

```
return type class-name::function-name(argument  declaration )
                {
                function-body
                }
```

The member ship label class-name :: tells the compiler that the function function - name belongs to the class class-name . That is the scope of the function is restricted to the class-name specified in the header line. The :: symbol is called scope resolution operator.

Example:

```
void item :: getdata (int a , float b )
        {
        number=a;
        cost=b;
        }
void item :: putdata ( void)
        {
        cout<<"number=:"<<number<<endl;
        cout<<"cost="<<cost<<endl;
        }
```

The member function have some special characteristics that are often used in the program development.

- Several different classes can use the same function name. The "membership label" will resolve their scope, member functions can access the private data of the class .A non member function can't do so.
- A member function can call another member function directly, without using the dot operator.

P.T.C

INSIDE THE CLASS DEF1NATION:

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class .

```
                Example:
                        class item
                        {
                                Intnumber;
                                float cost;
                        public:
                                void getdata (int a ,float b);
                                void putdata(void)
                                {
                                        cout<<number<<endl; cout<<cost<<endl;
                                }
                                };
```

A C++ PROGRAM WITH CLASS:

```
# include< iostream. h>
  class item
          {
                  int number;
                  float cost;
          public:
                  void getdata ( int a , float b);
                  void putdala ( void)
                          {
                          cout<<"number:"<<number<<endl;
                          cout<<"cost :"<<cost<<endl;
                          }
                          };
  void item :: getdata (int a , float b)
                  {
                  number=a;
                  cost=b;
                  }
          main ( )
          {
                  item x;
                  cout<<"\nobjectx"<<endl;
                  x. getdata( 100,299.95);
                  x .putdata();
                  item y;
                  cout<<"\n object y"<<endl;
                  y. getdata(200,175.5);
                  y. putdata();
                  }
```

Output:

```
object x
number   100
```

```
cost=299.950012
object -4
cost=175.5
```

Q.

Write a simple program using class in C++ to input subject mark and prints it.

ans:

```
class marks
{
        private :
                int ml,m2;
                public:
                void getdata();
                void displaydata();
};
void marks: :getdata()
{
        cout<<"enter 1st subject mark:";
        cin>>ml;
        cout<<"enter 2nd subject mark:";
        cin>>m2;
}
void marks: :displaydata()
{
        cout<<"Ist subject mark:"<<ml<<endl ;
        cout<<"2nd subject mark:"<<m2;
}
void main()
{
clrscr();
marks x;
x.getdata();
x.displaydata();

}
```

P.T.C

NESTING OF MEMBER FUNCTION;

A member function can be called by using its name inside another member function of the same class. This is known as nesting of member functions.

```
#include <iostream.h>
class set
{
        int m,n;
        public:
                void input(void);
                void display (void);
                void largest(void);
};
int set::largest (void)
{
        if(m>n)
                return m;
        else
                return n;
}
void set::input(void)
{
        cout<<”input values of m and n:”;
        cin>>m>>n;
}
void set::display(void)
{
cout<<”largestvalue=”<<largest()<<”\n”;
}
void main()
{
        set A;
        A.input( );
        A.display( );
}
```

output:

```
Input values of m and n:

3017

largest value= 30
```

Private member functions:

Although it is a normal practice to place all the data items in a private section and all the functions in public, some situations may require contain functions to be hidden from the outside calls. Tasks such as deleting an account in a customer file or providing increment to and employee are events of serious consequences and therefore the functions handling such tasks should have restricted access. We can place these functions in the private section.

A private member function can only be called by another function that is a member of its class. Even an object can not invoke a private function using the dot operator.

```
Class sample
{
        int m;
        void read (void);
        void write (void);
};
```

if si is an object of sample, then

```
s.read();
```

is illegal. How ever the function read() can be called by the function update ( ) to update the value of m.

```
                        void sample :: update(void)
                                {
                                read( );
                                }
```

```
        #include<iostream.h>
        class part
        {
private:
        int   modelnum,partnum;
        float cost;
public:
        void setpart ( int mn, int pn ,float c)
                {
                        modelmim=mn;
                        partnum=pn;
                        cost=e;
                }
        void showpart ( )
                {
                Cout<<endl<<"model:"<<modelnum<<end1;
                Cout<<"num:"<< partnum <<endl
                Cout<<"cost:"<<"$"<cost;
                }
                };
                void main()
                {
                part  pl,p2;
                p1.setpart(644,73,217.55);
                p2.setpart(567,89,789.55);
                pl.showpart();
                pl.showpart();
                }
```

```
output:- model:644
num:73
cost: $217550003
model: 567
num:89
cost: $759.549988
```

```
#indude<iostream.h>
        class distance
        {
        private:
                int feet;
                float inches;
        public:
                void setdist ( int ft, float in)
                        {
                                feet=ft;
                                inches=in;
                        }
                void    getdist()
                {
                cout<<"enter feet:";
                cin>>feet;
                cout<<"enter  inches:";
                cin>>inches;
                }
                void showdist()
                {
                cout<< feet<<"_"inches«endl;
                }
                };
                void main( )
                {
                distance dl,d2;
                d1.setdist(1 1,6.25);
                d2.getdata();
                cout<<endl<<"dist:"<<d 1 .showdist();
                cout<<"\n"<<"dist2:";
                d2.showdist();
                }
```

output:-

```
enter feet: 12

enter inches: 6.25

dist 1:"11'- 6.1.5"

dist 2:   12'- 6.25"
```

ARRAY WITH CLASSES:

```
                    #include<iostream.h>
                    #include<conio.h>
                    class employee
                    {
                    private:
                              char name[20];
                              int age,sal;
                    public:
                              void getdata();
                              void putdata();
                    };
void employee : : getdata ()
          {
                    cout<<"enter name :";
                    cin>>name;
                    cout<<"enter age :";
                    cin>>age;
                    cout<<"enter salary:";
                    cin>>sal;
                    return(0);
          }
          void employee : : putdata ( )
          {
          cout<<name <<endl;
          cout<<age<<endl;
          cout<<sal<<endl;
          return(0);
          }
          int main()
          {
```

P.T.C

```
employee emp[5]:
for( int i=0;i<5;i++)
{
emp[i].getdata();
}
cout<<endl;
for(i=0;i<5;i++)
{
emp[i].putdata();
}
getch();
return(0);
}
```

ARRAY OF OBJECTS:-

```
#include<iostream.h>
#include<conio.h>
        class emp
                {
                private:
                        char name[20];
                        int age,sal;
                public:
                        void    getdata( );
                        void    putdata( );
                                };
void    emp : : getdata( )
                {
                coul<<”enter empname”:   .
                cin>>name;
                cout<<”enter age:”<<endl;
                cin>>age;
                cout<<”enter salun :”;
```

```
                        cin>>sal;
                }
void    emp :: putdata ()
        {
        cout<<"emp name:"<<name<<endl;
        cout<<"emp age:"<<age<<endl;
        cout<<"emp salary:"<<sal;
        }



        void    main()
        {
        emp     foreman[5];
        emp     engineer[5];
        for(int i=0;i<5;i ++)
        {
        cout<<" for foreman:";
        foreman[i] . getdata();
        }
        cout<<endl;
        for(i=0;i<5;i++)
        {
        Foreman[i].putdata(); .
        }
        for(int i=0;i<5;i ++)
        {
        cout<<" for engineer:";
        ingineer[i].getdata();
        }
        for(i=0;i<5;i++)
        {
        ingineer[i].putdata();
        }
        getch();
        return(0);
        }
```

P.T.C

REPLACE AND SORT USING CLASS:-

```
                #include<iostream.h>
                #include<constream.h>
                        class sort
                                {
                        private:
                                int nm[30];
                        public;
                                void getdata();
                                void putdata();
                        }:
        void   sort :: getdata()
                    {
                        int i,j,k;
                        cout<<"enter 10 nos:" ;
                        for(i=0;i<10;i++)
                                {
                                cin>>nm[i];
                                }
                        for(i=0;i<9;i++)
                                {
                                for(j=i+l:j<10:j++)
                                {
                                        if(nm[i]>nm[j])
                                                {
                                                     k=nm[i];
                                                     nm[i]=nm[j];
                                                     nm[j]=k;
                                                }
                                }


        void sort :: putdata()
                {
                int k;
                for(k=0;k<10;k++)
                {
                        cout<<num [k] <<endl ;
```

P.T.C

```
                }
                }
                int main()
                {
                clrscr();
                sort   s;
                s.getdata();
                s.putdata();
                return(0);
}
```

ARRAY OF MEMBERS:

```
                #include<iostream.h>

                #include<constream.h>

                        const   int m=50;

                        class items

                        {

                                int    item_code[m];

                                float item_price[m];

                                int count;

                        public:

                                void cnt(void) { count=0;}

                                void get_item(void);

                                void display_sum(void);

                                void remove(void);

                                void display _item(void);

                        };





                        void items :: get_item (void)
                                {
                                        cout<<"enter itemcode:";
                                        cin>> item_code[code];
                                        cout<<"enter item cost:";
                                        cin>>item_price[count];
                                        count ++ ;
                                }

                        void items :: display _sum(void)
                                {
                                        float sum=0;
                                        for( int i=0;i<count;i++)
                                                {
```

```
                                        sum=sum+item_price[i];
                                        }
                    cout<< “\n total value:”<<sum<<endl;
          }
int main ( )
{
          items order;
          order.cnt();
          int x;
                    do
                    {
                    cout<<”\nyou can do the following:”;
                    cout<<”enter appropriate no:”;
                    cout<<endl<<” 1 :add an item’’;
                    cout<<endl<<”2: display total value :”;
                    cout<<endl<<”3 : display an item”;
                    cout<<endl<<”4 :display all item:”;
                    cout<<endl<<”5 : quit:”;
                    cout<<endl<<endl<<”what is your option:”;
                    cin>>x;

                    switch(x)
                    {
                    case 1: order.get_item(); break;
                    case 2: order.display_sum(); break;
                    cose 3: order.remove(); break;
                    case 4: order.display_item();break;
                    case 5: break;
                    default : cout<<”error in input; try again”;
                  }
      } while(x!=5);
}
```

P.T.C

STATIC DATA MEMBER:

A data member of a class can be qualified as static . The properties of a static member variable are similar to that of a static variable. A static member variable has contain special characteristics.

Variable has contain special characteristics:-

1) when the first object of its class is created.No other initialization is permitted.
2) Only one copy of that member is created for the entire class and is shared by , no matter how many objects are created.
3) It is visible only with in the class but its life time is the entire program. Static variables are normally used to maintain values common to the entire class. For example a static data member can be used as a counter that records the occurrence of all the objects.

```
int item :: count; // definition of static data member
```

Note that the type and scope of each static member variable must be defined outside the class definition .This is necessary because the static data members are stored separately rather than as a part of an object.

Example :-

```
#include<iostream.h>
        class   item
        {
                        //count is static
                int number;
        public:
                void getdata(int a)
                 {
                        number=a;
                        count++;
                 }
                void getcount(void)
                 {
                        cout<<"count:";
                        cout<<count<<endl;
                 }
        };
   int item :: count ; //count defined
        int main( )
        {
        item a,b,c;
        a.get_count( );
        b.get_count( );
        c.get_count( ):
        a.getdata( ):
        b.getdata( );
```

```
                    c.getdata( );
                    cout«"after reading data : "«endl;
                              a.get_count( );
                              b.gel_count( );
                              c.get count( );
          return(0);
          }
```

The output would be

```
          count:0
          count:0
          count:0
After reading data
          count: 3
          [redacted]
          [redacted]
```

The static Variable count is initialized to Zero when the objects created . The count is incremented whenever the data is read into an object. Since the data is read into objects three times the variable count is incremented three times. Because there is [redacted]ll t[redacted].

<u>STATIC MEMBER FUNCTIONS:-</u>

A member function that is declared static has following properties :-

1. A static function can have [redacted]e [redacted]
2. A static member function can be called using the class name as follows:-
   [redacted]

Example:-

```
          #include<iostream.h>
                    class test
                    {
                              int code;
                              [redacted] // static member variable
                    public:
                              void set(void)
                              {
                              code=++count;
                              }
                              void showcode(void)
                              {
                              cout<<"object member : "<<code<<end;
                              }
                    [redacted]
                              { cout<<"count="<<count<<endl; }
                              };
int test:: count;
int main()
{
```

```
test t1,t2;
t1.setcode( );
t2.setcode( );
test :: showcount ( );
test t3;
t3.setcode( );
[redacted]
t1.showcode( );
t2.showcode( );
t3.showcode( );
return(0);
```

output:-
```
count : 2
   count: 3
object number 1
object number 2
object number 3
```

OBJECTS AS FUNCTION ARGUMENTS

Like any other data type, an object may be used as A function argument. This can cone in two ways

1. [redacted]ction.
2. [redacted]ction

The first method is calle[redacted]. Since a copy of the object is passed to the function, any change made to the object inside the function [redacted].
The second method is called [redacted] . When an address of the object is passed, the called function works directly on the actual object used in the call. This means that [redacted].The pass by reference method is more efficient since it requires to pass only the address of the object and not the entire object.

Example:-
```
#include<iostream.h>
        class time
        {
                int hours;
                int minutes;
        public:
                void gettime(int h, int m)
                {
                                hours=h;
                                minutes=m;
                }
        void puttime(void)
                {
                cout<< hours<<"hours and:";

                cout<<minutes<<"minutes:"<<end;
                }
```

P.T.O

```
                void sum( time ,time);
        };
void time :: sum (time t1,time t2)      .
        {
        minutes=t1.minutes + t2.minutes;
        hours=minutes%60;
        minutes=minutes%60;
        hours=hours+t 1.hours+t2.hours;
        }

        int main()
        {
        time T1,T2,T3;
        T1.gettime(2,45);
        T2.gettime(3,30);
        T3.sum(T1,T2);
        cout<<"T1=";
        T1.puttime( );
        cout<<"T2=";

        cout<<"T3=";

        return(0);
        }
```

FRIENDLY FUNCTIONS:-

We know private members can not be accessed from outside the class. That is a non - member function can't have an access to the private data of a class. However there could be a case where two classes manager and scientist, have been defined we should like to use a function income-tax to operate on the objects of both these classes.

In such situations, c++ allows the common function lo be made friendly with both the classes , there by following the function to have access to the private data of these classes .Such a function need not be a member of any of these classes.

To make an outside function "friendly" to a class, we have to simply declare this function as a friend of the classes as shown below :

```
class ABC
{
---------
---------
public:
        --------
        ----------
              void xyz(void);
};
```

The function declaration should be preceded by the keyword friend , The function is defined else where in the program like a normal C ++ function . The function definition does not use their the keyword friend or the scope operator **::** . The functions that are declared with the keyword friend are known as friend functions. A function can be declared as a friend in any no of classes. A friend function, as though not a member function ,

A friend function processes certain special characteristics:

a. It is not in the scope of the class to which it has been declared as friend.
b. Since it is not in the scope of the class, it cannot be called using the object of that class. It can be invoked like a member function without the help of any object.
c. Unlike member functions.

Example:

```
#include<iostream.h>
        class sample
        {
                int a;
                int b;
        public:
                void setvalue( ) { a=25;b=40;}
                      float mean( sample s);
        }
        float   mean (sample s)
                {
                        return (float(s.a+s.b)/2.0);
                }
int  main ( )
    {
```

P.T.C

```
                                        sample x;
                                        x . setvalue( );
                                        cout<<"mean value="<<mean(x)<<endl;
                                        return(0);

                                        }

                    output:
                    mean value : 32.5
```

A function friendly to two classes

```
                    #include<iostream.h>
                          class abc;
                          class xyz
                          {
                                int x;
                          public:
                                void setvalue(int x) { x-= I; }
                                friend void max (xyz,abc);
                          };
                          class abc
                          {
                                int a;
                          public:
                                void setvalue( int i) {a=i; }
                                friend void max(xyz,abc);
                          };


                          void  max( xyz m, abc n)
                          {
                                if(m . x >= n.a)
                                      cout<<m.x;
                                else
                                      cout<< n.a;
                          }

                          int main( )
                          {
                          abc j;
                          j . setvalue( 10);
                          xyz   s;
                          s.setvalue(20);
                          max( s , j );
                          return(0);
                          }
```

SWAPPING PRIVATE DATA OF CLASSES:

```
              #include<iostream.h>

                          class class-2;
                          class class-1
                          {
```

```
          int value 1;
public:
          void indata( int a) { value=a; }
          void display(void) { cout<<value<<endl; }
          friend   void exchange ( class-1 &, class-2 &);
};

class class-2
{
          int value2;
public:
          void indata( int a) { value2=a; }
          void display(void) { cout<<value2<<endl; }
          friend void exchange(class-l & , class-2 &);
          };
void exchange ( class-1 &x, class-2 &y)
          {
                    int temp=x. value 1;
                    x. value I=y.valuo2;
                    y.value2=temp;
          }

          int main( )
          {
          class-1 c1;
          class-2 c2;
          c1.indata(l00);
          c2.indata(200);
          cout<<"values before exchange:"<<endl;
          c1.display( );
          c2.display( );
          exchange (c1,c2);
          cout<<"values after  exchange :"<< endl;
          c1. display ( );
          c2. display ( );
          return(0);
          }
```

output:

```
          values before exchange
                    100
                    200
          values after exchange
                    200
                    100
```

P.T.C

PROGRAM FOR ILLUSTRATING THE USE OF FRIEND FUNCTION:

```
        #include< iostream.h>
        class account1;
        class  account2
        {
        private:
                int balance;
        public:
        account2( ) { balance=567; }
        void showacc2( )
        {
        cout<<"balanceinaccount2 is:"<<balance<<endl;
friend int transfer (account2 &acc2, account1 &acc1,int amount);
        };
class acount1
{
private:
        int balance;
public:
        account1 ( ) { balance=345; }


        void showacc1 ( )
        {
                cout<<"balance in account1 :"<<balance<<endl;
        }
friend int transfer (account2 &acc2, account1 &acc1 ,int amount);
        };

int transfer ( account2 &acc2, account1 & acc1, int amount)
        {
                if(amount <=accl . bvalance)
                        {
                        acc2. balance + = amount;
                        acc1 .balance - = amount;
                        }
                else
                        return(0);
        }
int main()
{
account1    aa;
account2    bb;




        cout << "balance in the accounts before transfer:" ;
        aa . showacc1( );
        bb . showacc2( );
        cout << "amt transferred from account1 to account2 is:";
        cout<<transfer ( bb,aa,100)<<endl;
```

P.T.C

```
                    cout<< " balance in the accounts after the transfer:";
                    aa . showacc 1 ( );
                    bb. showacc 2( );
                    return(0);
}
```

output:

```
balance in the accounts before transfer
        balance in account 1 is 345
        balance in account2  is 567
and transferred from account! to account2 is 100
        balance in  account 1 is 245
        balance in   account2 is 667
```

P.T.C

RETURNING OBJECTS:

```
# include< iostream,h>
        class complex
        {
                float x;
                float y;
        public:
                void input( float real , float imag)
                        {
                                x=real;
                                y=imag;
                        }
                friend  complex sum( complex , complex);
                        void    show ( complex );
        };
complex   sum ( complex c1, complex c2)
                {
                complex c3;
                c3.x=c1.x+c2.x;
                c3.y=c1.y+c2.y;
                return c3;}



        void complex   :: show ( complex c)
        {
        cout<<c.x<<" +j "<<c.y<<endl;
         }

        int  main( )
        {
        complex a, b,c;
        a.input(3.1,5.65);
        b.input(2.75,1.2);
        c=sum(a,b);
        cout <<" a="; a.show(a);
        cout <<" b= "; b.show(b);
        cout <<" c=" ; c.show(c);
        return(0);
        }
```

output:

```
a =3.1 + j 5.65
b= 2.75+ j 1.2
c= 5.55 + j 6.85
```

P.T.C