POINTER TO MEMBERS;

It is possible to take the address of a member of a class and assign it to a pointer. The address of a member can be obtained by applying the operator & to a “fully qualified” class member name.

A class member pointer can be declared using the operator **::** * with the class name.

For Example:

```
class A
{
private:
        int m;
public:
        void show( );
};
```

We can define a pointer to the member m as follows :

```
int A :: * ip = & A :: m
```

The ip pointer created thus acts like a class member in that it must be invoked with a class object. In the above statement. The phrase A **::** * means “pointer - to - member of a class” . The phrase & A **::** m means the “ Address of the m member of a class”

The following statement is not valid :

```
int   *ip=&m ; // invalid
```

This is because m is not simply an int type data. It has meaning only when it is associated with the class to which it belongs. The scope operator must be applied to both the pointer and the member.

The pointer ip can now be used to access the m inside the member function (or friend function).

Let us assume that “a” is an object of “ A” declared in a member function . We can access "m" using the pointer ip as follows.

```
cout<< a . * ip;
cout<< a.m;
ap=&a;
cout<< ap-> * ip;
cout<<ap->a;
```

The deferencing operator ->* is used as to accept a member when we use pointers to both the object and the member. The dereferencing operator. .* is used when the object itself is used with the member pointer. Note that * ip is used like a member name.

We can also design pointers to member functions which ,then can be invoked using the deferencing operator in the main as shown below.

```
(object-name.* pointer-to-member function)
(pointer-to -object -> * pointer-to-member function)
```

The precedence of ( ) is higher than that of .* and ->* , so the parenthesis are necessary.

P.T.C

DEREFERENCING OPERATOR:

```
        #include<iostream.h>
                class M
                {
                        int x;
                        int y;
                public:
                        void set_xy(int a,int b)
                                {
                                        x=a;
                                        y=b;
                                }
                friend int sum(M);
                };

                int sum (M m)
                {
                        int M :: * px= &M :: x; //pointer to member x





                                int M :: * py- & m ::y;//pointer to y
                                M * pm=&m;
                                int s=m.* px + pm->py;
                                return(s);
                        }
                int main ( )
                        {
                        M m;
                        void(M::*pf)(int,int)=&M::set-xy;//pointer to function set-xy (n*pf)( 10,20);
                        //invokes set-xy
                        cout<<"sum=:"<<sum(n)<<cncil;
                        n *op=&n; //point to object n
                        ( op->* pf)(30,40); // invokes set-xy
                        cout<<"sum="<<sum(n)<<end 1 ;
                        return(0);
                        }
        output:
                sum= 30
                sum=70
```

CONSTRUCTOR:

A constructor is a special member function whose task is to initialize the objects of its class . It is special because its name is the same as the class name. The constructor is invoked when ever an object of its associated class is created. It is called constructor because it construct the values of data members of the class.

A constructor is declared and defined as follows:

```
//'class with a constructor
        class integer
        {
                int m,n:
        public:
                integer! void);//constructor declared
                ------------
                ------------
        };
integer :: integer(void)
        {
                m=0;
                n=0;
        }
```

When a class contains a constructor like the one defined above it is guaranteed that an object created by the class will be initialized automatically.

For example:-

```
Integer int1;   //object int 1 created
```

This declaration not only creates the object int1 of type integer but also initializes its data members m and n to zero.

A constructor that accept no parameter is called the default constructor. The default constructor for class A is A :: A( ). If no such constructor is defined, then the compiler supplies a default constructor .

Therefore a statement such as :-

```
A  a  ;//invokes the default constructor of the compiler of     the
   compiler to create the object   "a" ;
```

Invokes the default constructor of the compiler to create the object a.

The constructor functions have some characteristics:-

- They should be declared in the public section .
- They are invoked automatically when the objects are created.
- They don't have return types, not even void and therefore they cannot return values.
- They cannot be inherited , though a derived class can call

P.T.C

the base class constructor .

- Like other C++ function , they can have default arguments,
- Constructor can't be virtual.
- An object with a constructor can't be used as a member of union.

Example of default constructor:

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

        class abc
        {
        private:
                char nm[];
        public:
                abc ( )
                {
                        cout<<”enter your name:”;
                        cin>>nm;
                }
                void display( )





                        {
                                cout<<nm;
                        }

                        };

                        int main( )
                        {
                        clrscr( );
                        abc d;
                        d.display( );
                        getch( );
                        return(0);
                        }
```

PARAMETERIZED CONSTRUCTOR:-

the constructors that can take arguments are called parameterized constructors. Using parameterized constructor we can initialize the various data elements of different objects with different values when they are created.

Example:-

```
class integer
{
        int m,n;
public:
        integer( int x, int y);
                --------
                ---------
        };
```

P.T.C

```
integer:: integer (int x, int y)
                              {
                                        m=x;n=y;
                              }
```

the argument can be passed to the constructor by calling the constructor implicitly.

```
integer int 1 = integer(0,100); // explicit call
integer int 1(0,100);     //implicite call
```

CLASS WITH CONSTRUCTOR:-

```
#include<iostream.h>
class integer
{
          int m,n;
public:
          integer(int,int);
          void display(void)




          {
                    cout<<"m=:"<<m ;
                    cout<<"n="<<n;
          }
};
          integer :: integer( int x,int y) // constructor defined
                    {
                    m=x;
                    n=y;
                    }
          int main( )
          {
                    integer int 1(0, 100);    // implicit call
                    integer int2=integer(25,75);
                    cout<<" \nobject1 "<<endl;
                              int1.display( );
                              cout<<" \n object2 "<<endl;
                              int2.display( );
          }
```

output:

```
          object 1
          m=0
          n=100
          object2
          m=25
          n=25
```

P.T.O

Example:-

```
        #include<iostream.h>
        #include<conio.h>
        class abc
        {
        private:
                char nm [30];
                int age;
        public:
                abc ( ){ }// default
                abc ( char x[], int y);
                void get( )
                {
                cout<<"enter your name:";
                cin>>nm;
                cout<<" enter your age:";
                cin>>age;
                }
        void display( )
        {
                cout<<nm«endl;
                cout«age;
        }
    };
        abc : : abc(char x[], int y)
                {
                        strcpy(nm,x);
                        age=y;
                }
    void main( )
    {
    abc 1;
    abc m=abc("computer",20000);
    l.get();
    l.dispalay( );
    m.display ( );
    getch( );
    }
```

<u>OVERLOADED CONSTRUCTOR:-</u>

```
#include<iostream.h>
#include<conio.h>
        class sum
        {
        private;
                int a;
                int b;
                int c;
                float d;
                double e;
        public:
                sum ( )
{
cout<<”enter a;”;
cin>>a;
cout<<”enter b;”;
cin>>b;
cout<<”sum= “<<a+b<<endl;
        }
        sum(int a,int b);
        sum(int a, float d,double c);
        };
        sum :: sum(int x,int y)
                {
                        a=x;
                        b=y;
                }
        sum :: sum(int p, float q ,double r)
                {
                        a=p;
                        d=q;
                        e=r;
                }
        void main( )
        {
        clrscr( );
        sum 1;
        sum m=sum(20,50);
        sum n= sum(3,3.2,4.55);
        getch( );
        }
```

```
        output:
        enter a : 3
        enter b : 8
        sum=11
        sum=70
        sum=10.75
```

COPY CONSTRUCTOR:

A copy constructor is used to declare and initialize an object from another object.

Example:-

the statement

```
integer 12(11);
```

would define the object 12 and at the same time initialize it to the values of 11.

Another form of this statement is : integer 12=11;

The process of initialization through a copy constructor is known as copy initialization.

Example:-

```
#incliide<iostream.h>
        class code
        {
                int id;
```

```
                    public
                              code ( ) { } //constructor
                              code (int a) { id=a; } //constructor
                              code(code &x)
                              {
                                        Id=x.id;
                              }
                              void display( )
                              {
                              cout<<id;
                              }
                    };
          int main( )
          {
          code A(100);
          code B(A);
          code C=A;
          code D;
          D=A;
          cout<<" \n id of A :"; A.display( );
          cout<<" \nid of B :"; B.display( );
          cout<<" \n id of C:"; C.display( );
          cout<<" \n id of D:"; D.display( );
          }
```

output :-

```
          id of A:100
          id of B:100
          id of C:100
          id of D:100
```

<u>DYNAMIC CONSTRUCTOR:-</u>

The constructors can also be used to allocate memory while creating objects . This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size, thus resulting in the saving of memory.

Allocate of memory to objects at the time of their construction is known as dynamic constructors of objects. The memory is allocated with the help of new operator.

Example:-

```
#include<iostream.h>
#include<string.h>
class string
{
          char *name;




                    int    length;
          public:
                    string ( )
```

```
        {
        length=0;
        name= new char [length+1]; /* one extra for \0 */
        }
string( char *s) //constructor 2
        {
        length=strlen(s);
        name=new char [length+1];
        strcpy(name,s);
        }
void display(void)
{
        cout<<name<<endl;
}
void join(string &a .string &b)
        {
        length=a. length +b . length;
        delete name;
        name=new char[length+l]; /* dynamic allocation */
        strcpy(name,a.name);
        strcat(name,b.name);
        }
        };
int main( )
{
char * first = “Joseph” ;
string name1(first),name2(“louis”),naine3( “LaGrange”),sl,s2;
sl.join(name1,name2);
s2.join(s1,name3);
namel.display( );
name2.display( );
name3.display( );
s1.display( );
s2.display( );
}
output :-
        Joseph
        Louis
        language
        Joseph Louis
        Joseph Louis Language
```

DESTRUCTOR:-

A destructor, us the name implies is used to destroy the objects that have been created by a constructor. Like a constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde.

For Example:-

```
~ integer( ) { }
```

A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program to clean up storage that is no longer accessible. It is a good practice to declare destructor in a program since it releases memory space for future use.

Delete is used to free memory which is created by new.

Example:-

```
matrix : : ~ matrix( )
{
        for(int i=0; i<11;i++)
                delete p[i];
                delete p;
}
```

IMPLEMENTED OF DESTRUCTORS:-

```
#include<iostream.h>
        int count=0;
        class alpha
        {
        public:
                alpha( )
                {
                count ++;
                cout<<"\n no of object created :"<<endl;
                }
                ~alpha( )
                {
                        cout<<"\n no of object destroyed :" <<endl;
                        coutnt--;
                }
        };



int main( )
{

cout<<" \n \n enter main \n:";
alpha A1,A2,A3,A4;
{
        cout<<" \n enter block 1 :\n";
```