POINTERS TO MEMBER

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.

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)<<cnc
        il;  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
```