

# Chapter 8

## Classes and Objects

### Learning Objectives

**After completing this chapter, you will be able to:**

- *Understand classes.*
- *Understand objects.*
- *Understand member functions.*
- *Understand friend functions.*
- *Understand array of objects.*
- *Understand the concept of passing objects to functions.*
- *Understand the concept of assigning an object.*
- *Understand pointers to objects.*
- *Understand static data members.*
- *Understand static member functions.*
- *Understand local classes.*
- *Understand the concept of this pointer.*
- *Understand inline functions.*

## INTRODUCTION

In this chapter, you will learn about classes and objects, which are the main features of object-oriented programming (OOP). Classes and objects are used for data hiding.

## DEFINING A CLASS

A class is a collection of data members and member functions. These member functions are used to manipulate the data. A class is a user-defined data type, which hides the data members and member functions from external use, if necessary. It works like a protection cover on the data and functions.

### Declaring a Class

A class is declared in the same way as a structure except that a class declaration begins with the keyword **class**. The syntax is as follows:

```
class class_name
{
    access specifier:
        data member;
    access specifier:
        member function;
};
```

In the above syntax, the declaration begins with the keyword **class** followed by a **class\_name**, which is an identifier given by the programmer to specify the name of the class. The body of the class is enclosed within the curly braces {} and is terminated by a semicolon (;), and it contains the following:

- a. Access specifier
- b. Data members
- c. Member functions

### Access Specifier

An access specifier specifies the visibility of a data member or a member function of a class. It should be terminated by a colon (:). The access specifier can be any one of the following three keywords:

#### Private

By default, the members of a class are **private**. These members can only be accessed by the other members of the same class.

#### Public

The **public** members of a class can be accessed by the members of the same class and by the members of the other classes.

**Protected**

The **protected** members of a class can be accessed by the members of the same class, a friend class, and a derived class. The concept of the friend class and derived class will be discussed later in this chapter.

**Data Members**

The data members are variables of any valid data type. These variables are declared within the class.

**Member Functions**

A class contains the following member functions:

- a. Simple function
- b. Constructors
- c. Destructors

**Simple Function**

You have already learned about these functions in the previous chapter. These functions are defined with a return type and a list of arguments, and are used to manipulate the data members of the class.

**Constructors**

This is a special member function of a class. This function has the following properties:

- a. The constructor name is the same as the name of the class.
- b. No value is returned by the constructor, which means it does not contain any return type, or void.
- c. A constructor is used for memory allocation and also for initialization of the members of a class.

**Destructors**

The name of a destructor is the same as the name of the class but it is preceded by a negation operator (~). This is basically used for memory de-allocation.

The following example illustrates the declaration of a class:

```
class customer
{
    char name[20]; //data member1
    int acc_number; //data member2
public:
    void getdata( ); //member function1
    void showdata( ); //member function2
};
```

In the above example, the class **customer** contains two data members and two member functions. The data members in this class are **private** by default because here no access

specifier is used. The member function **getdata()** is used to read the values of the data members from the user and the member function **showdata()** is used to display these values on the screen. In this example, the member functions are only declared but they are not defined. You can also define the body of a function inside the class declaration (this will be discussed later in this chapter).

**Note**

*A good programming skill requires that all the data members should be declared in the private section and all the member functions should be declared in the public section while defining a class.*

## Creating the Objects of a Class

An object is an instance of a class. Each object of a class is uniquely identified by its name or properties.

After the declaration of a class, you can create the objects of that type by using the class name. The syntax for creating an object is as follows:

```
class_name var1;
```

In the above syntax, the **class\_name** represents the name of the class and **var1** represents an object of a class type, which is specified by the **class\_name**.

For example, if you want to create an object of the class **customer**, you will write the following code:

```
customer c1;
```

Here, **customer** is the class name and **c1** is an object of the type **customer**.

You can also create more than one object in a single statement as follows:

```
class_name var1, var2, var3, ....., varN;
```

Here, the object names are separated by a comma operator (,).

You can also create one or more than one object at the time of definition of a class. The syntax is as follows:

```
class class_name
{
    access specifier:
        data member;
    access specifier:
        member function;
}var1, var2, var3;
```

In this syntax, **var1**, **var2**, and **var3** are declared as the objects of class **class\_name**.

For example:

```
class customer
{
    char name[20];
    int acc_number;
public:
    void getdata( );
    void showdata( );
}c1, c2, c3;
```

In the above example, **c1**, **c2**, and **c3** are declared as the objects of the class **customer**.

### Accessing the Members of a Class

You cannot access the data members and member functions of a class directly in the **main( )** function. For doing so, the data member or member function must be preceded by an object name and the dot operator (.). The syntax is as follows:

object\_name . function\_name (list of arguments);

For example, if you want to access the member function **showdata( )** of the class **customer**, you will write the following code:

```
c1.showdata( );
```

You can access the **public** data members of a class directly by using the object name and the dot operator but not the **private** data members.

For example:

```
class demo
{
    int a;
public:
    int b;
};
int main( )
{
    .....;
    demo d1;
    d1.b=20;
    d1.a=10;//error, because a is private
    .....;
    .....;
}
```

In this example, you cannot access the member variable **a** in the **main( )** function with the object **d1** of class **demo**. This is because **a** is declared as a private data member of the class **demo**.

## DEFINING MEMBER FUNCTIONS

You can define the member functions of a class in two ways:

- a. Inside the class definition
- b. Outside the class definition

### Inside the Class Definition

You can define a member function inside a class definition by replacing the function declaration with a function definition. The syntax is as follows:

```
class class_name
{
    access specifier:
        data members:
    access specifier:
        return_type fun_name (list of arguments)
        {
            body of the function;
        }
};
```

For example:

```
class customer
{
    char name[20];
    int acc_number;
public:
    void getdata( )
    {
        cout<<"Enter the name of the customer:"<<endl;
        cin>>name;
        cout<<"Enter the account number:"<<endl;
        cin>>acc_number;
    }
    void showdata()
    {
        cout<<"The name entered by you is:";
        cout<<name<<endl;
```

```
        cout<<"The account number entered by you is:";
        cout<<acc_number<<endl;
    }
};
```

In this example, the functions **getdata()** and **showdata()** are defined inside the class **customer**.

## Outside the Class Definition

You can also define a member function of a class outside the class definition. The member function definition is very much similar to the normal function definition. The function definition and member function definition both have a return type, an argument list, and a function body. But a member function also has an identity label that specifies to the compiler the class to which the function belongs. The syntax for defining a function outside the class definition is as follows:

```
return_type class_name :: fun_name (argument list)
{
    function body
}
```

In the above syntax, the identity label **class\_name ::** specifies to the compiler that the function specified by **fun\_name** belongs to the class specified by **class\_name**. The operator (**::**) used in the syntax is called the scope resolution operator, which specifies the scope of the function. The scope of the function means that the function specified by **fun\_name** is restricted to the class specified by **class\_name**.

For example:

```
class customer
{
    char name[20];
    int acc_number;
public:
    void getdata( );
    void showdata( );
};
void customer::getdata()
{
    cout<<"Enter the name of the customer:"<<endl;
    cin>>name;
    cout<<"Enter the account number:"<<endl;
    cin>>acc_number;
}
void customer::showdata()
{
    cout<<"The name is:";
```

```

        cout<<name<<endl;
        cout<<"The account number is:";
        cout<<acc_number<<endl;
    }

```

The following example illustrates the use of a class.

### Example 1

Write a program to create a class.

The following program will create a class, which contains the member functions that prompt the user to enter the name and account number. It will also display the resultant values on the screen. The line numbers on the right are not a part of the program and are for reference only.

```

//Write a program to illustrate the use of a class                                1
#include<iostream>                                                                2
using namespace std;                                                            3
class customer                                                                    4
{                                                                                  5
    char name[20];                                                                6
    int acc_number;                                                              7
public:                                                                           8
    void getdata( );                                                            9
    void showdata( );                                                         10
};                                                                               11
void customer::getdata( )                                                       12
{                                                                               13
    cout<<"Enter the name of the customer:"<<endl;                            14
    cin>>name;                                                                  15
    cout<<"Enter the account number:"<<endl;                                  16
    cin>>acc_number;                                                            17
}                                                                               18
void customer::showdata( )                                                       19
{                                                                               20
    cout<<"The name is: ";                                                       21
    cout<<name<<endl;                                                           22
    cout<<"The account number is:  ";                                           23
    cout<<acc_number<<endl;                                                     24
}                                                                               25
int main( )                                                                      26
{                                                                               27
    customer c1, c2, c3;                                                        28
    c1.getdata( );                                                              29
    c1.showdata( );                                                            30
    c2.getdata( );                                                              31

```



```
        c2.showdata( );           32
        c3.getdata( );           33
        c3.showdata( );          34
        return 0;                 35
    }                               36
```

### Explanation

Line 4

**class customer**

In this line, **class** is the keyword and **customer** is the name of the class.

Line 5

```
{
```

This line indicates the start of the definition of the class **customer**.

Line 6 and 7

**char name[20];**

**int acc\_number;**

These lines contain two data members of the class **customer**. Here, no access specifier is specified with these data members. So, they are considered as **private** data members and can be accessed only within the same class **customer**.

Lines 8 to 10

**public:**

**void getdata( );**

**void showdata( );**

Lines 9 and 10 contain two member functions of the class **customer**, **getdata( )** and **showdata( )**. Line 8 contains an access specifier **public**, which specifies that both the member functions can be accessed in the same class and also from outside the class **customer**.

Line 11

```
};
```

This line indicates the end of the definition of the class **customer** and is terminated by a semicolon.

Line 12

**void customer::getdata( )**

Here, the member function **getdata( )** is defined outside the class **customer** definition.

Line 19

**void customer::showdata( )**

Here, the member function **showdata( )** is defined outside the class **customer** definition.

Line 28

**customer c1, c2, c3;**

In this line, three objects **c1**, **c2**, and **c3** are declared as **customer** type and each of them has its own copy of data members of the class **customer**.

Line 29

**c1.getdata( );**

In this line, a function call is made to the member function **getdata( )**, which is used to read the value of data members of the class from the end user. Here, the values entered by the user are assigned to the copy of the data members of the object **c1**.

Line 30

**c1.showdata( );**

In this line, a function call is made to the member function **showdata( )**, which is used to display the values of data members of the object **c1** on the screen.

Line 31

**c2.getdata( );**

In this line, a function call is made to the member function **getdata( )**, which is used to read the values of data members of the class from the end user. Here, the values entered by the user are assigned to the copy of the data members of the object **c2**.

Line 32

**c2.showdata();**

In this line, a function call is made to the member function **showdata()**, which is used to display the values of data members of the object **c2** on the screen.

The output of the program is as follows:

Enter the name of the customer:

John

Enter the account number

102

The name is: John

The account number is: 102

Enter the name of the customer:

Smith

Enter the account number

110

The name is: Smith

The account number is: 110

Enter the name of the customer:

Michael

Enter the account number

145

The name is: Michael

The account number is: 145

The output will be displayed on the screen as follows:

```

c:\ "c:\C++\C++_programs\Ch_08\ch08_pro01\... - _ x
Enter the name of the customer:
John
Enter the account number:
102
The name is: John
The account number is: 102
Enter the name of the customer:
Smith
Enter the account number:
110
The name is: Smith
The account number is: 110
Enter the name of the customer:
Michael
Enter the account number:
145
The name is: Michael
The account number is: 145
Press any key to continue
  
```

## MEMORY ALLOCATION FOR OBJECTS

When an object of a class is created, some amount of memory is allocated to it. The amount of memory allocated to an object is equal to the amount of memory required by the data members of that class. This is because each object member of a class has its own copy of the data members. All the objects of a class share the same copy of the member functions of a class, as shown in Figure 8-1.

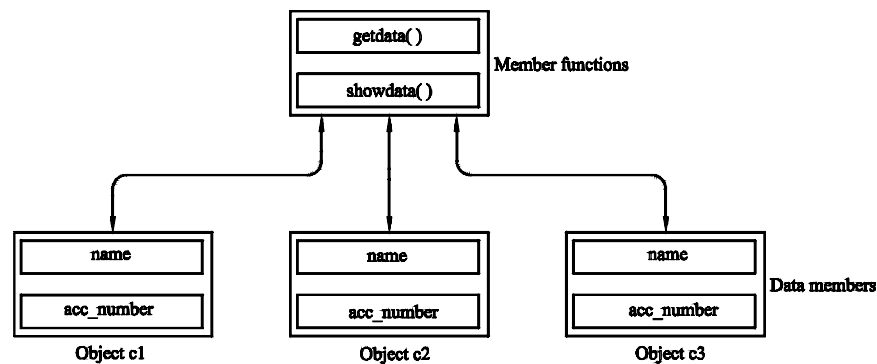


Figure 8-1 Memory allocation

In the above figure, the member functions of a class are shared by all the three objects **c1**, **c2**, and **c3** and each object has its own copy of data members of a class.

## NESTED MEMBER FUNCTIONS

Until now, you have observed that a member function of a class is called with the help of an object of that class and a dot operator. You can also make a call to a member function of a

class by using its name inside another member function of the same class. These type of functions are called nested member functions.

The following example illustrates the use of the nested member functions.

### Example 2

Write a program to find the average of three numbers.

The following program will prompt the user to enter three numbers, calculate their average, and display the resultant value on the screen.

```
//Write a program to calculate the average of three numbers      1
#include<iostream>                                              2
using namespace std;                                           3
class average                                                  4
{                                                              5
    int a, b, c, d;                                           6
    public:                                                    7
        void input( );                                       8
        void show( );                                       9
        int avrg( );                                       10
};                                                            11
void average :: input(void)                                   12
{                                                            13
    cout<<"Enter three values"<<endl;                       14
    cin>>a>>b>>c;                                           15
}                                                            16
int average::avrg(void)                                       17
{                                                            18
    d=(a+b+c)/3;                                           19
    return d;                                              20
}                                                            21
void average::show(void)                                     22
{                                                            23
    cout<<"The average of three numbers is: "<< avrg( )<<endl; 24
}                                                            25
int main( )                                                  26
{                                                            27
    average a1;                                             28
    a1.input( );                                           29
    a1.show( );                                           30
    return 0;                                              31
}                                                            32
```

### Explanation

Lines 22 to 25

```
void average::show(void)
{
```

```
    cout<<"The average of three numbers is:"<< avrg()<<endl;
```

```
}
```

These lines contain the definition of the member function **show()**. In the body of this function, another member function **avrg()** is called, which returns the resultant value after calculating the average of three numbers.

The output of the program is as follows:

Enter three values

26

25

24

The average of three numbers is: 25

## FRIEND FUNCTIONS

You can also access the **private** and **protected** data members of a class by using the **friend** function. The syntax for declaring a **friend** function is as follows:

```
friend ret_type fun_name (argument list)
```

The **friend** function is not a member of a class. A **friend** function declared in the **public** or the **private** section in the definition of a class is not affected like the other member functions of the class. You can define a **friend** function outside the class definition without using the class name and scope resolution operator (**::**). You can also make a call to a **friend** function in the main program without the help of any object. A **friend** function always contains an object as an argument.

The following example illustrates the use of the **friend** function.

### Example 3

Write a program to calculate the average of three numbers.

The following program will prompt the user to enter three numbers, calculate their average, and display the resultant value on the screen.

```
//Write a program to calculate the average of three numbers      1
#include<iostream>                                                2
using namespace std;                                             3
class average                                                    4
{                                                                5
    int a, b, c;                                                 6
public:                                                         7
```

```

        friend int avrg(average a1);           8
        void input();                         9
};                                           10
void average::input(void)                   11
{                                           12
    cout<<"Enter three values"<<endl;        13
    cin>>a>>b>>c;                             14
}                                           15
int avrg(average a1)                       16
{                                           17
    return (a1.a+a1.b+a1.c)/3;              18
}                                           19
int main( )                               20
{                                           21
    average n;                             22
    n.input();                             23
    cout<<"The average of three numbers is: "<<avrg(n)<<endl; 24
    return 0;                             25
}                                           26

```

### Explanation

Line 8

**friend int avrg(average a1);**

In this line, the function **avrg** is declared as a **friend** function and it takes an object **a1** of the class **average** as an argument.

Lines 16 to 19

**int avrg(average a1)**

```

{
    return (a1.a+a1.b+a1.c)/3;
}

```

Lines 16 to 19 contain the definition of the function **avrg**, which is a **friend** of the class **average**. This function is not the member of the class. So, it is defined outside the class definition as a normal function without using the class name and the scope resolution operator.

Line 22

**average n;**

In this line, the variable **n** is declared as an object of type **average**.

Line 23

**n.input();**

In this line, a call is made to the member function **input( )** with the help of an object **n** and a dot operator.

Line 24

```
cout<<avrg(n);
```

In this line, a call is made to the **friend** function **avrg(n)** and an object of the class **average** is passed as an argument.

The output of the program is as follows:

Enter three numbers

26

25

24

The average of three numbers is: 25

---

## ARRAY OF OBJECTS

As you already know that an array is a collection of data elements of the same type. Similarly, an array can also contain a collection of objects that are of the same class type. These types of arrays are known as array of objects. The syntax for declaring an array of objects is the same as for an array of any type.

For example:

```
class demo
{
    int a, b;
    public:
        void input( );
        void show( );
};
int main( )
{
    demo d[3];
    .....;
    .....;
}
```

In the above example, array **d** represents an array of objects that contains three objects **d[0]**, **d[1]**, and **d[2]** of type **demo**.

The following example illustrates the use of an array of objects.

### Example 4

Write a program to create an array of objects.

The following program will prompt the user to enter a name and employee ID, create an array of objects, and display the resultant array on the screen.

```

//Write a program to create an array of objects          1
#include<iostream>                                         2
using namespace std;                                     3
class employee                                           4
{                                                         5
    char name[30];                                        6
    int emp_id;                                          7
    public:                                              8
        void input( );                                  9
        void show( );                                  10
};                                                       11
void employee :: input( )                                12
{                                                         13
    cout<<"Enter the name:"<<endl;                     14
    cin>>name;                                           15
    cout<<"Enter employee ID:"<<endl;                  16
    cin>>emp_id;                                         17
}                                                         18
void employee :: show( )                                19
{                                                         20
    cout<<"The name entered by you is: "<<name<<endl;    21
    cout<<"The employee ID is: "<<emp_id<<endl;         22
}                                                         23
int main( )                                              24
{                                                         25
    employee e[3];//Array of objects                    26
    int i;                                              27
    for (i=0;i<3;i++)                                   28
    {                                                     29
        e[i].input( );                                  30
    }                                                     31
    for(i=0;i<3;i++)                                   32
    {                                                     33
        e[i].show( );
    }
    return 0;
}

```

The output of the program is as follows:

Enter the name:

John

Enter employee ID:

110

Enter the name:

Smith

Enter employee ID:

130

Enter the name:

Michael

Enter employee ID:

125



The name entered by you is: John  
The employee ID is: 110  
The name entered by you is: Smith  
The employee ID is: 130  
The name entered by you is: Michael  
The employee ID is: 125

## PASSING OBJECTS TO FUNCTIONS

An object can be passed to a function as an argument in the same way as the variables of any other data type. In this process, a copy of the object is passed to a function. If any change is made to the given object inside the function, it will not be reflected in the actual object. This process is similar to the concept of pass by value. Another way of passing an object is to pass the starting address of the object to a function. If any change is made to the object inside the function, it will be reflected in the actual object. This process is similar to the concept of pass by reference.

The following example illustrates the use of passing objects to functions.

### Example 5

Write a program to multiply the values of two objects.

The program given next will multiply the values of two objects that are passed as arguments to a function, assign the resultant value to the third object, and display the resultant value on the screen.

```
//Write a program to multiply the values of two objects      1
#include<iostream>                                           2
using namespace std;                                         3
class multiply                                               4
{                                                            5
    int a;                                                    6
public:                                                       7
    void getvalue(int x)                                     8
    {                                                         9
        a=x;                                                 10
    }                                                         11
    void showvalue( )                                       12
    {                                                         13
        cout<<a;                                           14
    }                                                         15
    void mul(multiply m1, multiply m2);                     16
};                                                            17
void multiply :: mul(multiply m1, multiply m2)              18
{                                                            19
    a= m1.a * m2.a;                                         20
}                                                            21
```

```

int main( )                                22
{                                           23
    multiply M1, M2, M3;                    24
    M1.getvalue(10);                        25
    M2.getvalue(20);                        26
    M3.mul(M1, M2);                         27
    cout<<"M1= ";M1.showvalue();cout<<endl; 28
    cout<<"M2= ";M2.showvalue();cout<<endl; 29
    cout<<"M3= ";M3.showvalue();cout<<endl; 30
    return 0;                              31
}                                           32

```

### Explanation

Line 16

**void mul(multiply m1, multiply m2);**

In this line, the function **mul** is declared inside the definition of the class **multiply**. Also the argument list contains two arguments **m1** and **m2**, which are objects of the class **multiply**.

Line 25

**M1.getvalue(10);**

In this line, a function call is made to the **getline( )** function and value 10 is passed as an argument. Here, value 10 is assigned to the variable **a**, which is the copy of object **M1**.

Line 26

**M2.getvalue(20);**

In this line, a function call is made to the **getline( )** function and value 20 is passed as an argument. Here, value 20 is assigned to variable **a**, which is the copy of object **M2**.

Line 27

**M3.mul(M1, M2);**

Here, a function call is made to function **mul( )**, and two objects **M1** and **M2** of class **multiply** are passed as arguments. The copies of these objects are assigned to **m1** and **m2** respectively, in line 18, respectively. After this assignment, value 10 stored in variable **a** of object **M1** and value 20 stored in variable **a** of object **M2** are multiplied and the resultant value 200 is assigned to object **M3**.

The output of the program is as follows:

M1= 10

M2= 20

M3= 200

## ASSIGNING AN OBJECT

You can also assign an object to another object but both the objects should be of the same type. This means both the objects should belong to the same class. The syntax for assigning an object to another object is given next.

```
obj1=obj2;
```

In the above syntax, **obj1** and **obj2** belong to the same class. The data of the object **obj2** is copied to the data of the object **obj1**.

The following example illustrates the use of assignment of an object.

### Example 6

Write a program to assign the data of an object to another object of a class.

The following program will assign the data of an object to another object and display the resultant values on the screen.

```
//Write a program to assign the data of an object to another object      1
#include<iostream>                                                         2
using namespace std;                                                     3
class assign                                                               4
{                                                                           5
    float x;                                                               6
    public:                                                                7
        void input(float a)                                              8
        {                                                                 9
            x=a;                                                         10
        }                                                                11
        float show( )                                                    12
        {                                                                 13
            return x;                                                    14
        }                                                                15
};                                                                           16
int main( )                                                                17
{                                                                           18
    assign a1,a2;                                                         19
    a1.input(10.23);                                                      20
    a2=a1;                                                                21
    cout<<"The value assigned to object a2 is: "<<a2.show( )<<endl;    22
    return 0;                                                            23
}                                                                           24
```

### Explanation

Line 19

**assign a1,a2;**

In this line, variables **a1** and **a2** are declared as objects of the class **assign**.

Line 20

**a1.input(10.23);**

Here, a call is made to the **input()** function and a float value 10.23 is passed as an argument. This value is assigned to the variable **a** in line 8. The function call is made with the object **a1**. So, the value 10.23 is assigned to the copy of the data member **x** of object **a1**.

Line 21

**a2=a1;**

In this line, the value 10.23 stored in the copy of the data member **x** of object **a1** is assigned to the copy of the data member **x** of object **a2**.

The output of the program is as follows:

The value assigned to object a2 is: 10.23

---

## POINTERS TO OBJECTS

Like the pointer to a variable, you can also create pointers to objects. The starting address of the object is assigned to the pointer. After that, you can access the members of a class with the help of the arrow operator (**->**) rather than the dot operator (**.**). The syntax is as follows:

```
pointer_var =&obj_name;
```

In the above syntax, the **&** operator returns the starting address of the object specified by the **obj\_name**. Next, the resultant address is assigned to the pointer variable specified by the **pointer\_var**.

For example:

```
class demo
{
    int i;
    public:
        void input(int a)
        {
            i=a;
        }
        int show( )
        {
            return i;
        }
};

int main( )
{
    demo d1, *p;
    p=&d1;
    p->input(10);
}
```

```
        p->show( );
        -----;
        -----;
    }
```

In the above example, variable **d1** is declared as an object of the class **demo**. A pointer variable **p** is declared as a pointer to the object. The **&** operator returns the starting address of object **d1**, which is assigned to **p**. Now, the member functions **input( )** and **show( )** of class **demo** can be accessed with the help of the pointer to the object **p**.

You can also assign an array of objects to the pointer to an object variable. The syntax for assigning an array of objects to the pointer to an object is as follows:

```
pointer_var = obj_name[number of objects];
```

In the above syntax, the address of the first object of the **array of objects** is assigned to the pointer variable. If the pointer variable is incremented, it will point to the next object in the array.

For example:

```
class demo
{
    int i;
    public:
        void input(int a)
        {
            i=a;
        }
        int show( )
        {
            return i;
        }
};
int main( )
{
    demo d1[5], *p;
    int j;
    p=d1;
    for(j=0; j<5; j++)
    {
        p->input(10);
        p++;
    }
    for(j=0; j<5; j++)
        p->show( );
    -----;
```

```

    -----;
}

```

In the above example, an array of objects **d1** of class **demo** is declared and the size of the array is 5. A pointer variable **p** is declared as a pointer to an object. In the statement **p=d1;**, by default, the address of the first object of the array **d1** is assigned to the pointer to an object **p**. When **p** is incremented in statement **p++;**, it will point to the next object of the array.

If the data members of a class are **public**, you can assign the address of the copy of data member of an object to a pointer variable. The syntax is as follows:

```
pointer_var = &obj_name.data_member;
```

In the above syntax, the **&** operator returns the address of the copy of the data member **data\_member** of the object **obj\_name**. This address is then assigned to the pointer variable specified by the **pointer\_var**.

For example:

```

class demo
{
    public:
        int i;
        void input(int a)
        {
            i=a;
        }
};
int main( )
{
    demo d1;
    int *p;
    d1.input(10);
    p=&d1.i;
    cout<<*p;
    -----;
    -----;
}

```

In the above example, the data member **i** of class **demo** is declared as **public** and **d1** is declared as an object of type **demo**. In the statement **p=&d1.i;**, the address of the copy of data member **i** of object **d1** is assigned to the pointer variable **p**. Now, you can directly access the value 10, which is stored in the variable **i**, by using the **\*p** statement.

## STATIC DATA MEMBERS

Until now, you have learned that each object of a class has its own copy of data members. To make a data member common to all the objects, you need to declare the data member as

**static** by using the **static** keyword in the declaration statement. The characteristic of a static data member is that only one copy is created for the entire class and all the objects share the same copy. All the **static** data members are initialized to zero before the creation of the first object. The syntax for declaring a **static** data member is as follows:

```
class class_name
{
    static data_type var_name;//declaration
    -----;
    -----;
};
data_type class_name :: var_name;//definition
```

In the above syntax, the declaration part contains a keyword **static**, which directs the compiler that only one copy of the data member, specified by **var\_name**, is created for the entire class. When a **static** variable is declared inside the definition of a class, no memory space is allocated to it and it cannot be used for the entire class. So, to make it global and to provide enough memory space to it, it must be defined outside the definition of a class with the help of the scope resolution operator (**::**). The **static** data member is defined outside the class in the same way as a function is declared outside the definition of a class.

You can also assign an initial value to a **static** data member in the definition. The syntax is as follows:

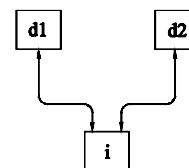
```
data_type class_name :: var_name=value;
```

In this syntax, an initial value specified by **value** is assigned to the variable specified by **var\_name**.

For example:

```
class demo
{
    static int i;//declaration
    -----;
    -----;
}d1, d2;
int demo :: i;//definition
```

In the above example, **d1** and **d2** are the objects of the class **demo** and the data member **i** is declared as a **static** data member. So both the objects **d1** and **d2** will share the same copy of data member **i**, as shown in Figure 8-2.



**Figure 8-2** Representation of a static data member

The following example illustrates the use of the **static** data members.

### Example 7

Write a program to illustrate the working of a **static** data member.

The following program will assign the values to a static data member, pass it as an argument to the member functions, and display the resultant values on the screen.

```
//Write a program to illustrate the working of the static data members      1
#include<iostream>                                                           2
using namespace std;                                                         3
class static_demo                                                            4
{                                                                             5
    static int a;//declaration                                              6
public:                                                                       7
    void input(int x)                                                        8
    {                                                                         9
        a=x;                                                                10
    }                                                                        11
    void show()                                                              12
    {                                                                         13
        cout<<"The value stored in a is: "<<a;                             14
    }                                                                        15
};                                                                            16
int static_demo::a;//definition                                             17
int main( )                                                                  18
{                                                                             19
    static_demo d1, d2, d3;                                                 20
    d1.input(10);                                                            21
    d1.show( );                                                              22
    d2.show( );                                                              23
    d3.show( );                                                              24
    d2.input(20);                                                            25
    d1.show( );                                                              26
    d2.show( );                                                              27
    d3.show( );                                                              28
    return 0;                                                                29
}                                                                             30
```

### Explanation

Line 6

**static int a;**

Here, the integer type variable **a** is declared as a **static** data member of the class **static\_demo**. It means that only one copy of the variable **a** exists for the entire class and all the objects will share the same copy.



Line 17

**int demo::a;**

Here, variable **a** is defined outside the class **static\_demo**. This definition makes the variable **a** global and also initializes it with the value 0.

Line 21

**d1.input(10);**

Here, a function call is made to the member function **input( )** and value 10 is passed as an argument. This value is then assigned to the variable **a** in the body of the function. Now, the variable **a** contains value 10.

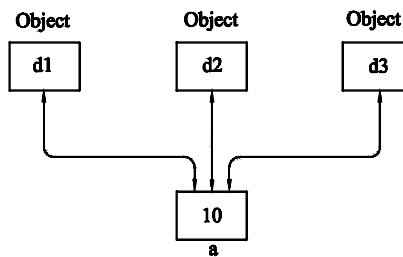
Line 22 to Line 24

**d1.show();**

**d2.show();**

**d3.show();**

In these lines, a call is made to the member function **show( )** with three different objects **d1**, **d2**, and **d3**. This function will display the same value 10 for all the three objects because only one copy of the variable **a** exists and it is shared by all the three objects, as shown in Figure 8-3.



*Figure 8-3 Representation of variable sharing*

Line 25

**d2.input(20);**

In this line, the member function **input( )** is called with the object **d2** and the value 20 is passed as an argument. This value is then assigned to the variable **a** in the body of the function. Now, the previous value 10 in the variable **a** is overwritten with the new value 20. Now, the variable **a** contains the value 20.

The output of the program is as follows:

The value stored in a is: 10

The value stored in a is: 10

The value stored in a is: 10

The value stored in a is: 20

The value stored in a is: 20

The value stored in a is: 20

## STATIC MEMBER FUNCTIONS

You can also declare the member functions of a class as **static** member functions. A member function is declared **static** in the same way as a data member is declared **static** by using the **static** keyword. The syntax is as follows:

```
static return_type fun_name (arguments list);
```

When a function is declared as a **static** member function in a class, it can only access the **static** data members and **static** member functions of that particular class. You can make a call to a **static** member function of a class by using a class name and a scope resolution operator (::), as follows:

```
class_name :: fun_name( );
```

For example:

```
class demo
{
    int a;
    static int b;
    public:
        void input( );
        static void show( );
};
-----;
int main( )
{
    demo d1, d2;
    d1.input( );
    demo :: show( );
    -----;
    -----;
}
```

In the above example, the member function **show( )** is declared as **static**. So, it can only access the **static** data member **b**. In the statement **demo :: show( )**, a call is made to the **static** member function by using the class name **demo** and the scope resolution operator (::).

The following example illustrates the use of the **static** member functions.

### Example 8

Write a program to illustrate the working of a **static** member function.

The following program will assign the values to a static data member, pass it as an argument to a static member function, and display the resultant values on the screen.

```
//Write a program to illustrate the use of static member functions      1
#include<iostream>                                                         2
using namespace std;                                                     3
class static_demo                                                         4
{                                                                           5
    static int a;                                                         6
    public:                                                               7
        void input(int x)                                                8
        {                                                                9
            a=x;                                                         10
        }                                                                11
        static void show( )                                             12
        {                                                                13
            cout<<"Value stored in a is: "<<a<<endl;                  14
        }                                                                15
};                                                                         16
int static_demo :: a;                                                    17
int main( )                                                              18
{                                                                           19
    static_demo d1, d2;                                                 20
    d1.input(10);                                                        21
    d2.input(20);                                                        22
    d1.display( );                                                       23
    static_demo :: show();                                               24
    return 0;                                                            25
}                                                                           26
```

### Explanation

Lines 12 to 15

**static void show( )**

```
{
    cout<<"Value stored in a is: "<<a<<endl;
}
```

In these lines, the **static** member function **show( )** is defined. This function can only access the static data member **a** of the class **static\_demo**.

Line 24

**static\_demo :: show();**

Here, a call is made to the **static** member function **show( )**.

The output of the program is as follows:

Value stored in a is: 20

## LOCAL CLASSES

You can also define a class within the body of a function or inside a block of code. These classes are known as local classes. In a local class, you cannot define a **static** data member or a **static** member function. The enclosing function or the block of code in which a local class is defined cannot access the **private** data member of that class. The syntax for the local class is as follows:

```

return_type
{
    -----;
    class class_name
    {
        data members;
        access_specifier:
            member functions;
    };
    -----;
    class_name obj1;//object creation
    -----;
}

```

The following example illustrates the use of local classes.

### Example 9

Write a program to illustrate the working of a local class.

The following program will multiply two numbers. These numbers are passed as arguments to a member function in a local class. The program will display the resultant value on the screen.

```

//Write a program to illustrate the use of local classes                                1
#include<iostream>                                                                    2
using namespace std;                                                                  3
void mul();//function declaration                                                       4
int main()                                                                            5
{                                                                                      6
    mul( );                                                                           7
    return 0;                                                                         8
}                                                                                      9
void mul( )//function definition                                                       10
{                                                                                      11
    class multiply//definition of local class                                         12
    {                                                                                  13
        int a, b, c;                                                                  14
        public:                                                                        15
            void input(int x, int y)                                                  16

```

```

        {
            a=x;
            b=y;
        }
        int show()
        {
            c=a*b;
            return c;
        }
    };
    multiply m1;
    m1.input(10,20);
    cout<<"After multiplication, c contains: "<<m1.show()<<endl;
}

```

The output of the program is as follows:  
After multiplication, c contains: 200

### The this pointer

When a function call is made to the member function of a class with an object, an implicit argument is passed automatically by the compiler. This argument is a pointer, which points to the invoking object (the object on which the function is called). The pointer is known as the **this** pointer.

For example, a call is made to the member function **demo( )** with an object **obj1** in the following way:

```
obj1.demo( );
```

Here, the **this** pointer will contain the address of object **obj1**.

In a member function, you can directly access the private data member of a class without using any object. You can also do so by using the **this** pointer.

For example:

```

class demo
{
    int a;
    -----;
    public:
        void show( )
        {
            this->a;//accessing variable a
        }
        -----;
};

```

In this example, the **private** data member **a** will be directly accessed by using the **this** pointer.

The following example illustrates the use of the pointer **this**.

### Example 10

Write a program to multiply two numbers using the pointer **this**.

The following program will multiply two numbers, assign the resultant value to another variable, and display it on the screen.

```
//Write a program to multiply two numbers
#include<iostream>
using namespace std;
class multiply
{
    int a, b, c;
    public:
        void input(int x, int y)
        {
            this->a=x;
            this->b=y;
        }
        int show( )
        {
            this->c=this->a*this->b;
            return this->c;
        }
};
int main( )
{
    multiply m1, m2;
    m1.input(10,20);
    cout<<"After multiplication c contains: "<<m1.show( )<<endl;
    m2.input(20,30);
    cout<<"After multiplication c contains: "<<m2.show( )<<endl;
    return 0;
}
```

### Explanation

Line 22

**m1.input(10,20);**

In this line, the member function **input(10,20)** is invoked with the object **m1**. In the function definition (Lines 8 to 12), the **this** pointer points to the object **m1**.

Line 23

```
cout<<"After multiplication c contains: "<<m1.show()<<endl;
```

In this line, the member function **show()** is invoked with the object **m1**. In the function definition (Lines 13 to 17), the **this** pointer points to the object **m1**.

Line 24

```
m2.input(20,30);
```

In this line, the member function **input(10,20)** is invoked with the object **m2**. In the function definition (Lines 8 to 12), the **this** pointer points to the object **m2**.

Line 25

```
cout<<"After multiplication c contains: "<<m1.show()<<endl;
```

In this line, the member function **show()** is invoked with the object **m2**. In the function definition (Lines 13 to 17), the **this** pointer points to the object **m2**.

The output of the program is as follows:

After multiplication c contains: 200

After multiplication c contains: 600

---

## INLINE FUNCTIONS

The **inline** functions are the small functions that may have one or two statements in the function body. The main characteristic of the **inline** functions is that when they are invoked, they expand in a line. This means in the inline functions, the function call is replaced with the corresponding function code by the compiler. You can create an **inline** function by using the **inline** keyword in the function definition. The syntax for **inline** functions is as follows:

```
inline return_type fun_name
{
    function code;
}
```

For example:

```
inline int mul(int a, int b)
{
    return (a*b);
}
int main( )
{
    cout<<mul(10,20);
    return 0;
}
```

In the above example, when **mul(int a, int b)** is invoked at some point in the main program, the entire code of function definition is replaced with the function call.

The **inline** functions are mainly used to increase the speed of execution. The **inline** functions have an advantage when the function body contains one or two statements. As the number of statements increase, the benefits of the **inline** functions get reduced.

### Self-Evaluation Test

Answer the following questions and then compare them to the answers given at the end of this chapter:

1. A class is a collection of \_\_\_\_\_ and \_\_\_\_\_.
2. The \_\_\_\_\_ member of a class cannot be accessed from outside the class.
3. An \_\_\_\_\_ is an instance of a class.
4. A \_\_\_\_\_ function is not a member of a class.
5. The entire class has only one copy of a \_\_\_\_\_ data member.

### Review Questions

Answer the following questions:

1. Define a class.
2. Define an object and also its creation.
3. Explain a **friend** function.
4. How does a **static** data member differ from a simple data member of a class?
5. Explain **inline** function.

### Exercise

#### Exercise 1

Write a program to define a class **Area** that uses the member functions to input the dimensions (length and breadth) of a rectangle, calculates the area, and also displays the result.

#### Answers to Self-Evaluation Test

1. data members and member functions, 2. **private**, 3. object, 4. **friend**, 5. **static**