is illegal. However, the function **read()** can be called by the function **update()** to update the value of **m**.

```
void sample :: update(void)
{
    read();     // simple call; no object used
}
```

5.9 Arrays within a Class

The arrays can be used as member variables in a class. The following class definition is valid.

```
const int size=10;      // provides value for array size

class array
{
    int a[size];        // 'a' is int type array
  public:
    void setval(void);
    void display(void);
};
```

The array variable **a[ ]** declared as a private member of the class **array** can be used in the member functions, like any other array variable. We can perform any operations on it. For instance, in the above class definition, the member function **setval()** sets the values of elements of the array **a[ ]**, and **display()** function displays the values. Similarly, we may use other member functions to perform any other operations on the array values.

Let us consider a shopping list of items for which we place an order with a dealer every month. The list includes details such as the code number and price of each item. We would like to perform operations such as adding an item to the list, deleting an item from the list and printing the total value of the order. Program 5.3 shows how these operations are implemented using a class with arrays as data members.

PROCESSING SHOPPING LIST

```
#include <iostream>

using namespace std;

const m=50;

class ITEMS
```

(Contd)

```
{
        int itemCode[m];
        float itemPrice[m];
        int count;
  public:
        void CNT(void){count = 0;}          // initializes count to 0
        void getitem(void);
        void displaySum(void);
        void remove(void);
        void displayItems(void);
};
//=============================================================
void ITEMS :: getitem(void)                 // assign values to data
                                            // members of item
{
        cout << "Enter item code :";
        cin >> itemCode[count];

        cout << "Enter item cost :";
        cin >> itemPrice[count];
        count++;
}
void ITEMS :: displaySum(void)              // display total value of
{                                           // all items
        float sum = 0;
for(int i=0; i<count; i++)
                sum = sum + itemPrice[i];

        cout << "\nTotal value :" << sum << "\n";
}
void ITEMS :: remove(void)
{                                           // delete a specified item
        int a;
        cout << "Enter item code :";
        cin >> a;

        for(int i=0; i<count; i++)
                if(itemCode[i] == a)
                        itemPrice[i] = 0;
}

void ITEMS :: displayItems(void)
{                                           // displaying items
```

(Contd)

```
        cout << "\nCode  Price\n";

        for(int i=0; i<count; i++)
        {
                cout <<"\n" << itemCode[i];
                cout <<"    " << itemPrice[i];
        }
        cout << "\n";
}
//=============================================================

int main()
{
        ITEMS order;
        order.CNT();
        int x;
        do              // do....while loop
        {
                cout << "\nYou can do the following;"
                     << "Enter appropriate number \n";
                cout << "\n1 : Add an item ";
                cout << "\n2 : Display total value";
                cout << "\n3 : Delete an item";
                cout << "\n4 : Display all items";
                cout << "\n5 : Quit";
                cout << "\n\nWhat is your option?";

                cin >> x;

                switch(x)
                {
                        case 1 : order.getitem(); break;
                        case 2 : order.displaySum(); break;
                        case 3 : order.remove(); break;
                        case 4 : order.displayItems(); break;
                        case 5 : break;
                        default : cout << "Error in input; try again\n";


        } while(x != 5);                // do ..while ends

        return 0;
}
```

PROGRAM 5.3

The output of Program 5.3 would be:

```
You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?1
Enter item code :111
Enter item cost :100

You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?1
Enter item code :222
Enter item cost :200

You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?1
Enter item code :333
Enter item cost :300

You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?2
Total value :600
```

(Contd)

```
You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?3
Enter item code :222

You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?4
Code      Price
111             100
222             0
333             300

You can do the following; Enter appropriate number
1 : Add an item
2 : Display total value
3 : Delete an item
4 : Display all items
5 : Quit

What is your option?5
```

note

The program uses two arrays, namely **itemCode**[ ] to hold the code number of items and **itemPrice**[ ] to hold the prices. A third data member **count** is used to keep a record of items in the list. The program uses a total of four functions to implement the operations to be performed on the list. The statement

```
const int m = 50;
```

defines the size of the array members.

The first function **CNT**( ) simply sets the variable **count** to zero. The second function **getitem**() gets the item code and the item price interactively and assigns them to the array members **itemCode[count]** and **itemPrice[count]**. Note that inside this function **count**

is incremented after the assignment operation is over. The function **displaySum** ( ) first evaluates the total value of the order and then prints the value. The fourth function **removeItem** ( ) deletes a given item from the list. It uses the item code to locate it in the list and sets the price to zero indicating that the item is not 'active' in the list. Lastly, the function **displayItems** ( ) displays all the items in the list.

The program implements all the tasks using a menu-based user interface.

5.10 Memory Allocation for Objects

We have stated that the memory space for objects is allocated when they are declared and not when the class is specified. This statement is only partly true. Actually, the member functions are created and placed in the memory space only once when they are defined as a part of a class specification. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. Only space for member variables is allocated separately for each object. Separate memory locations for the objects are essential, because the member variables will hold different data values for different objects. This is shown in Fig. 5.3.

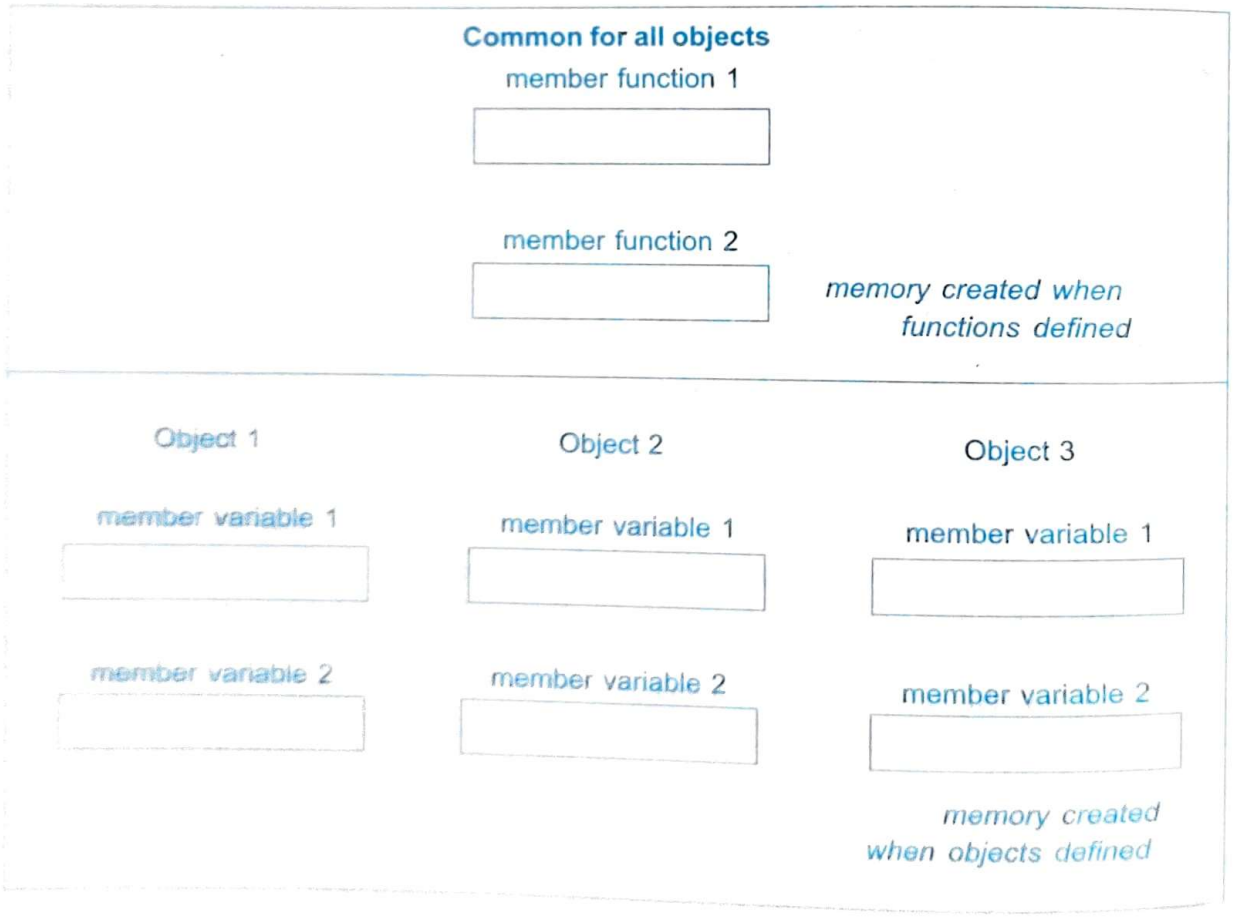

Fig. 5.3 ⇔ Object of memory

5.11 Static Data Members

A data member of a class can be qualified as static. The properties of a **static** member variable are similar to that of a C static variable. A static member variable has certain special characteristics. These are :

- It is initialized to zero when the first object of its class is created. No other initialization is permitted.
- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is visible only within the class, but its lifetime 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 occurrences of all the objects. Program 5.4 illustrates the use of a static data member.

STATIC CLASS MEMBER

```
#include <iostream>

using namespace std;

class item
{
        static int count;
        int number;
  public:
        void getdata(int a)
        {
                number = a;
                count ++;
        }
        void getcount(void)
        {
                cout << "count: ";
                cout << count  << "\n";
        }
};

int item :: count;

int main()
{
```

(Contd)

```
    item a, b, c;          // count is initialized to zero
    a.getcount();          // display count
    b.getcount();
    c.getcount();

    a.getdata(100);        // getting data into object a
    b.getdata(200);        // getting data into object b
    c.getdata(300);        // getting data into object c

    cout << "After reading data" << "\n";

    a.getcount();          // display count
    b.getcount();
    c.getcount();
    return 0;
}
```

PROGRAM 5.4

The output of the Program 5.4 would be:

```
count: 0
count: 0
count: 0
After reading data
count: 3
count: 3
count: 3
```

note

Notice the following statement in the program:

```
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. Since they are associated with the class itself rather than with any class object, they are also known as *class variables*.

The **static** variable **count** is initialized to zero when the objects are 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 only one copy of count shared by all the three objects, all the three output statements cause the value 3 to be displayed. Figure 5.4 shows how a static variable is used by the objects.

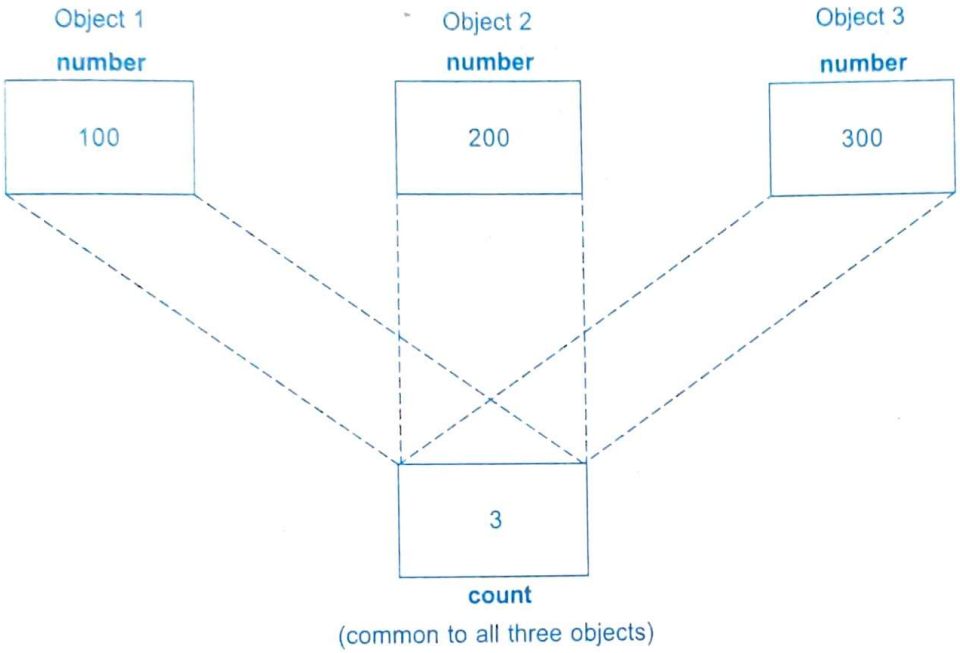

Fig. 5.4 ⇔ *Sharing of a static data member*

Static variables are like non-inline member functions as they are declared in a class declaration and defined in the source file. While defining a static variable, some initial value can also be assigned to the variable. For instance, the following definition gives count the initial value 10.

```
int item :: count = 10;
```

5.12 Static Member Functions

Like **static** member variable, we can also have **static** member functions. A member function that is declared **static** has the following properties:

- A **static** function can have access to only other static members (functions or variables) declared in the same class.
- A **static** member function can be called using the class name (instead of its objects) as follows:

```
class-name :: function-name;
```

Program 5.5 illustrates the implementation of these characteristics. The **static** function **showcount()** displays the number of objects created till that moment. A count of number of objects created is maintained by the **static** variable count.

The function **showcode()** displays the code number of each object.

STATIC MEMBER FUNCTION

```
#include <iostream>

using namespace std;

class test
{
        int code;
        static int count;              // static member variable
public:
        void setcode(void)
        {
        code = ++count;
        }
        void showcode(void)
        {
        cout << "object number: " << code << "\n";
        }
        static void showcount(void)    // static member function
        {
        cout << "count: " << count << "\n";
        }
};

int test :: count;
int main()
{
        test t1, t2;

        t1.setcode();
        t2.setcode();

        test :: showcount();    // accessing static function

        test t3;
        t3.setcode();

        test :: showcount();

        t1.showcode();
        t2.showcode();
        t3.showcode();

        return 0;
}
```

PROGRAM 5.5

Output of Program 5.5:

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

note

Note that the statement

```
code = ++count;
```

is executed whenever **setcode( )** function is invoked and the current value of **count** is assigned to **code**. Since each object has its own copy of **code**, the value contained in **code** represents a unique number of its object.

Remember, the following function definition will not work:

```
static void showcount()
{
     cout << code;      // code is not static
}
```

5.13 Arrays of Objects

We know that an array can be of any data type including **struct**. Similarly, we can also have arrays of variables that are of the type **class**. Such variables are called *arrays of objects*. Consider the following class definition:

```
class employee
{
          char name[30];
          float age;
     public:
          void getdata(void);
          void putdata(void);
};
```

The identifier **employee** is a user-defined data type and can be used to create objects that relate to different categories of the employees. Example:

```
employee manager[3];          // array of manager
employee foreman[15];         // array of foreman
employee worker[75];          // array of worker
```

The array **manager** contains three objects(managers), namely, **manager[0]**, **manager[1]** and **manager[2]**, of type **employee** class. Similarly, the **foreman** array contains 15 objects (foremen) and the **worker** array contains 75 objects(workers).

Since an array of objects behaves like any other array, we can use the usual array-accessing methods to access individual elements, and then the dot member operator to access the member functions. For example, the statement

```
manager[i].putdata();
```

will display the data of the ith element of the array **manager**. That is, this statement requests the object **manager[i]** to invoke the member function **putdata()**.

An array of objects is stored inside the memory in the same way as a multi-dimensional array. The array manager is represented in Fig. 5.5. Note that only the space for data items of the objects is created. Member functions are stored separately and will be used by all the objects.

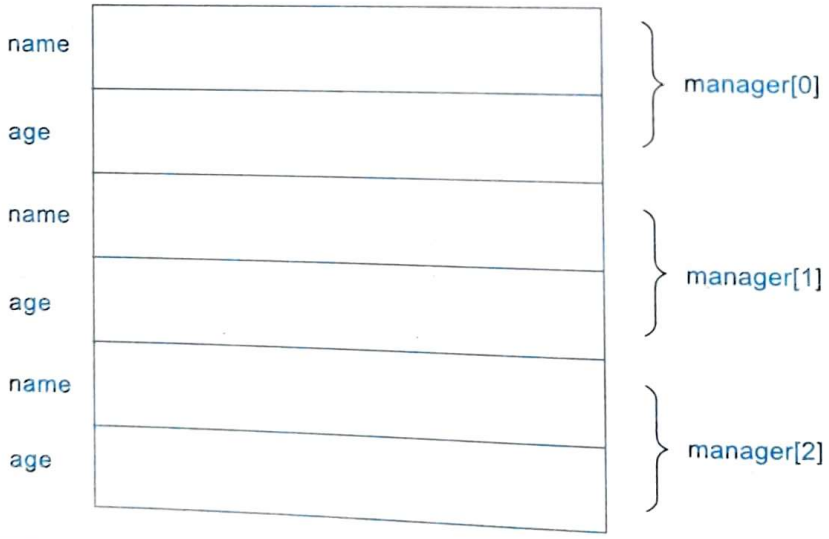

Fig. 5.5 ⇔ *Storage of data items of an object array*

Program 5.6 illustrates the use of object arrays.

ARRAYS OF OBJECTS

```
#include <iostream>
using namespace std;
class employee
```

(Contd)