Classes and Objects

8

CHAPTER

CHAPTER OUTLINE

CHAPTER OUTLINE

8.1 INTRODUCTION

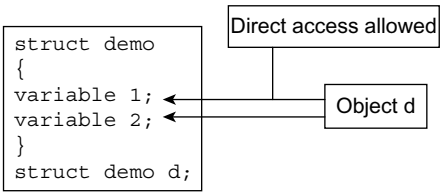

Fig. 8.1 Structures in C

In C, structures are used to define custom or user-defined data type. Structure is a combination of same or different data types. Only variables are declared inside a structure. The initialization of member variables inside the structure is not permitted. Objects can directly access the data members of the structures.

Functions are not permitted as members in structure. Outside functions are also able to access the data members of the structure through the object. Thus, there is no security to data members declared inside the structure in C as shown in Figure 8.1.

In C++, structure also combines function with data members. C++ introduces new keyword `class`. A `class` in C++ is similar to structure. Using `class` or structure, a programmer can merge one or more dissimilar data types and a new custom data type can be created. A `class` is nothing but grouping of variables of different data types with functions. Each variable of a `class` is called as member variable. Functions are called as member functions or methods.

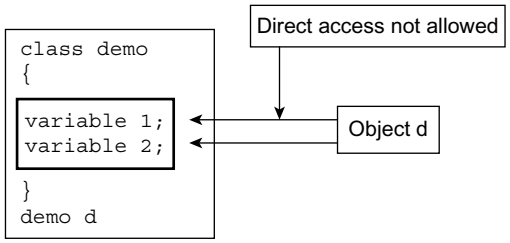

Fig. 8.2 `class` in C++

In C++, it is possible to access data members directly by objects, which is not possible in C. It is always the programmer's choice to allow or disallow the direct access of data members. The mechanism of restricting access of data outside the `class` is called as data hiding or encapsulation. In such a case, only the member functions can access the data. If `class` is used in place of `struct`, it restricts the access of data members as shown in Figure 8.2.

8.2 STRUCTURE IN C

In C, it is possible to combine dissimilar data types using structure but it has the following limitations:

a) Functions are not allowed as members of structure.
b) Direct access to data members is possible. So, by default all the structure members are public. Hence, security to data or data hiding is not provided.
c) The `struct` data type is not treated as built-in type; that is the use of `struct` keyword is necessary to declare objects.
d) The data members cannot be initialized inside the structure.

The syntax of structure declaration is as follows:

```
syntax:
struct <struct name>
{
   Variable1;
   Variable2;
};
```

```
Example:
sruct item
{
   int codeno;
   float prize;
   int qty;
};
```

In the above example, item is a structure name. The data members are codeno, prize, and qty. Thus, a custom data type is created by combination of one or more data members.

The object of structure `item` can be declared as follows:

```
struct item a,*b
```

The object declaration is same as the declaration of variables of built-in data types. The object a and the pointer *b are variables of type item and each of them has three data members. The a and *b can access the data members of `struct` item. Use of keyword `struct` is necessary.

Access to structure members: The data members of a structure are accessed by using object name and operators such as dot (.) or arrow (->). The dot (.) or arrow (->) operators are known as referencing operators. The dot operator is used when simple object is declared and `arrow` operator is used when object is pointer to structure. The access of members can be accomplished as given in the following syntax:

[Object name][Operator][Member variable name]

When an object is a simple variable, access of members is done as follows:

```
Object name dot (.) member variable name
   a.codeno
   a.price
   a.qty
```

When an object is a pointer to structure then members are accessed as follows:

```
a->codeno
a->price
a->qty
```

Note: A structure declaration must have a name or identifier followed by keyword `struct`, else the compiler will display an error. This is because it will not recognize the type of object declared. Hence, there is no possibility of accessing the structure member variables.

The following program illustrates the above discussion.

8.1 Write a program to access data members of a structure.

```
#include<stdio.h>
#include<conio.h>
struct item // struct declaration
{
  int codeno; // codeno=200 not possible as per 8.2 section
  float prize;
  int qty;
};
void main()
{
  struct item a,*b; // object declaration
  clrscr();
  a.codeno=123; // direct access & initialization of member variables
  a.prize=150.75;
  a.qty= 150;
  printf ("\n With simple variable");
  printf ("\n Codeno : %d ",a.codeno);
  printf ("\n Prize : %d",a.prize);
  printf ("\n Qty : %d",a.qty);
  b->codeno=124; // direct access & initialization of member vari-
                    ables
  b->prize=200.75;
  b->qty= 75;
  printf ("\n\n With pointer variable");
  printf ("\n Codeno : %d ",b->codeno);
  printf ("\n Prize : %d",b->prize);
  printf ("\n Qty : %d",b->qty);
}
```

OUTPUT

With simple variable
Codeno : 123
Prize : 150.75
Qty : 150

With pointer to structure
Codeno : 124
Prize : 200.75
Qty : 75

Explanation: The above program is compiled with C compiler. The following discussion is according to C compiler. In the above program, the structure item is declared with three member variables. The initialization of member variables inside the `struct` is not permitted. The declaration of member variables is enclosed within the curly braces. The `struct` declaration is terminated with a semicolon.

In function `main()`, the objects a and b are declared. Consider the following statement.

```
struct item a,*b; // object declaration
```

The `struct` must be preceded by structure name. The member variables can be accessed and initialization is done directly by object. The dot and arrow operators are used to access the member variables.

8.3 STRUCTURE IN C++

No doubt, C++ has made various improvements in structure. To know the improvements made in C++, the last program is compiled and executed with C++ compiler. The explanation followed by this program discusses the various improvements.

8.2 Write a program to declare `struct`. Initialize and display contents of data members.

```
#include<iostream.h>
#include<conio.h>
struct item // struct declaration
{
  int codeno; // codeno=200 not possible
  float prize;
  int qty;
};
int main()
{
  item a,*b; // object declaration
  clrscr();
  a.codeno=123; //direct access & initialization of member variables
  a.prize=150.75;
  a.qty= 150;
  cout<<"\n With simple variable";
  cout<<"\n Codeno : "<<a.codeno;
  cout<<"\n Prize : "<<a.prize;
  cout<<"\n Qty : "<<a.qty;
  b->codeno=124; // direct access & initialization of member variables
  b->prize=200.75;
```

```
    b->qty= 75;
    cout<<"\n\n With pointer to structure";
    cout<<"\n Codeno : "<<b->codeno;
    cout<<"\n Prize : "<<b->prize;
    cout<<"\n Qty : "<<b->qty;
    return 0;
}
```

Explanation: The above program is the same as previous one. The output is also same, hence not shown. Consider the following statement.

```
item a,*b; // object declaration in c++
```

While declaring an object, the keyword `struct` is omitted, which is compulsory in C. The structure item is a user-defined data type. C++ itself behaves as a structure data type which is built-in type and allows variable declaration.

C++ introduces new keyword `class`, which is similar to structure. The other improvements are discussed with the use of `class` in the following section.

8.4 CLASSES IN C++

A `class` is used to pack data members and member function together. The `class` has a mechanism to prevent direct access to its members, which is the central idea of object-oriented programming. The whole declaration of `class` is given in Table 8.1. The `class` declaration is also known as formation of new abstract data type. The abstract data type can be used as basic data type such as `int`, `float`, `etc.`

Table 8.1 Syntax and an Example of `class`

Syntax of `class` Declaration	**Example of `class`**

`class` <name of `class`>
{
`private:`
`declaration of variables;`
`prototype declaration of function;`
`public:`
`declaration of variables;`
`prototype declaration of function;`
};
	`class` item // `class` declaration
	{
	`private:`
	`int codeno;`
	`float prize;`
	`int qty;`
	`void values();`
	`public:`
	`void show();`
	};

The `class` is a keyword. The `class` declaration is same as `struct` declaration. The declaration of a `class` is enclosed with curly braces and terminated with a semicolon. The data members and member functions can be declared in two sections, that is `private` and `public`. The `private` and `public` keywords are terminated with a colon (**:**). The object cannot directly access the data members and member functions declared in private section, but it can access the data member and member functions declared in public section. The private members of a `class` can only be accessed by a public member function of the same `class`. Different sections of `class` are illustrated with examples in the following sections.

It is also possible to access private member variables directly like public member variables provided that the `class` should have at least one public member variable. Both the private and public member variables are stored in consecutive memory locations in the memory. A pointer

to member variable provides address of member variable. By applying increment (++) and decrement (--) operations on pointer, we can access all private and public member variables of the `class`. The object of a `class` contains address of the first member variable. It can be also used to access the private or public data.

8.5 DECLARING OBJECTS

A `class` declaration only builds the structure of an object. The member variables and functions are combined in the `class`. The declaration of objects is same as declaration of variables of basic data types. Defining objects of `class` data type is known as `class` ***instantiation***. Only when objects are created, memory is allocated to them.

Consider the following examples.

```
a) int x,y,z; // Declaration of integer variables
b) char a,b,c; // Declaration of character variables
c) item a,b, *c; // Declaration of object or class type variables
```

In the example `(a)`, three variables `x`, `y`, and `z` of `int` types are declared. In the example `(b)`, three variables `a`, `b`, and `c` of `char` type are declared. In the same fashion the third example declares the three objects `a`, `b`, and `c` of `class item`. The object `*c` is pointer to `class` item.

An object is an abstract unit with the following properties:

a) It is individual.
b) It points to a thing, either physical or logical that is identifiable by the user.
c) It holds data as well as operation method that handle data.
d) Its scope is limited to the block in which it is defined.

Access to `class` members: The object can access the public data member and member functions of a `class` by using dot (.) and arrow (->) operators. The syntax is as follows:

`[Object name] [Operator] [Member name]`

To access data members of `class` the statement would be as follows:

```
a.show();
```

where `a` is an object and `show()` is a member function. The `dot` operator is used because `a` is a simple object.

In statement

```
c->show();
```

`*c` is pointer to `class` item; therefore, the arrow operator is used to access the member.

Consider the given example.

```
class item // class declaration
{
        int codeno;
        float prize;
        int qty;
};
```

We replaced the `struct` keyword with `class`. If programs 8.1 and 8.2 are executed with `class`, they will not work. For example,

```
void main()
{
   item a,*b; // object declaration
   clrscr();
   a.codeno=123; // Direct access is not allowed
   a.prize=150.75;
   a.qty= 150;
}
```

The above program will generate error messages such as "'item::codeno' is not accessible". This is because the object cannot directly access the member variables of a `class`, which is possible with structure. Hence, we can say that the difference between `class` and `struct` is that the member variables of `struct` can be accessed directly by the object, whereas the member variables of `class` cannot be accessed directly by the object.

8.6 THE `public` KEYWORD

In Section 8.3, we noticed that the object directly accesses the member variables of structure, whereas the same is not possible with `class` members. The keyword `public` can be used to allow objects to access the member variables of a `class` directly like structure. The `public` keyword is written inside the `class`. It is terminated with a colon (**:**). The member variables and functions declared followed by the keyword `public` can be accessed directly by the object. The declaration can be done as follows:

```
class item // class declaration
{
   public: // public section begins
   int codeno;
   float prize;
   int qty;
};
```

Consider the following program that illustrates the use of `public` keyword with `class`.

8.3 Write a program to declare all members of a `class` as public. Access the elements using object.

```
#include<iostream.h>
#include<conio.h>
class item
{
  public: // public section begins
  int codeno;
  float prize;
  int qty;
}; // end of class
```

```
int main()
{
  clrscr();
  item one; // object declaration
  one.codeno=123; // member initialization
  one.prize=123.45;
  one.qty=150;
  cout<<"\n Codeno = "<<one.codeno;
  cout<<"\n Prize = "<<one.prize;
  cout<<"\n Quantity = "<<one.qty;
  return 0;
  }
```

OUTPUT

Codeno = 123
Prize =123.449997
Quantity =150

Explanation: In the above program, the members of `class` item are declared followed by keyword `public`. The object `one` of `class` item accesses the member variables directly. The member variables are initialized and values are displayed on the screen.

8.7 THE `private` KEYWORD

The `private` keyword is used to prevent direct access of member variables or function by the object. The `class` by default produces this effect.

The structure variables by default are `public`. To prevent member variables and functions of `struct` from direct access, the `private` keyword is used. The `syntax` of private keyword is same as `public`. The `private` keyword is terminated with a colon. Consider the following example.

```
struct item
{
   private: // private section begins
   int codeno;
   float prize;
   int qty;
}; // end of class
int main()
{
   clrscr();
   item one; // object declaration
   one.codeno=123; // member initialization
   one.price=123.45;
   one.qty=150;
}
```

As soon as the above program is compiled, the compiler will display the following error message:

```
'item::codeno' is not accessible
'item::prize' is not accessible
'item::qty' is not accessible
'item::codeno' is not accessible
'item::prize' is not accessible
```

From the above discussion, we noticed that by default (without applying `public` or `private` keyword) the `class` members are private (not accessible) whereas the `struct` members are public (accessible).

The private members are not accessible by the object directly. Then the question is how will they be accessed? To access the private members of a `class`, member functions of the same `class` are used. The member function must be declared in the `class` in public section. A program on accessing private members is given in the forthcoming sections. Through the public member function, an object can access the private members.

8.8 THE `protected` KEYWORD

The access mechanism of `protected` keyword is same as `private` keyword. We cannot access protected section members from outside the `class` by any object.

The following example clears the above concept:

8.4 Write a program using `class` to declare member variable and function under protected section and make an attempt to access them using object.


```
#include<iostream.h>
#include<conio.h>
class A
{
  protected:
  int num;
  void display()
  {
  cout<<num;
  }
};
int main()
{
  A a;
  a.num=100;
  a.display();
  return 0;
}
```

Note: The above program gives an error **`A::num`** and **`A::display()`** are not accessible.

The protected keyword is frequently used in inheritance of classes. Its detailed description is given Chapter 11.

8.9 ACCESS SPECIFIERS AND THEIR SCOPE

As described in Table 8.2, the `class` object can access a public member of the `class` directly without the use of member function. The private and protected mechanisms do not allow an object to access data directly. The object can access private or protected members only through public member functions of the same `class`.

Table 8.2 Access Limits of `class` Members

Access Specifiers	Access Permission	
	`class` **Members**	`class` **Object**
Public	Allowed	Allowed
Private	Allowed	Disallowed
Protected	Allowed	Disallowed

The following program explains the working of the above keywords:

8.5 Write a program using `class` to declare member variable and functions private, public and protected section and make an attempt to access them using object.

```
#include<iostream.h>
#include<conio.h>
class sample
{
  private:
  int num;
  void show1()
  {
  cout<<"Inside the private section";
  cout<<"\nEnter a number:";
  cin>>num;
  cout<<"Number is:"<<num;
  }
  public:
  int num1;
  void show()
  {
  show1();
  cout<<"\n\nInside the public section";
  cout<<"\nEnter a number:";
  cin>>num1;
  cout<<"Number is:"<<num1;
  show2();
  }
```

```
  protected:
  int num2;
  void show2()
{
  cout<<"\n\nInside the protected section";
  cout<<"\nEnter a number:";
  cin>>num2;
  cout<<"Number is:"<<num2;
  }
};
int main()
{
  sample s;
  s.show();
return 0;
}
```

OUTPUT

```
Inside the private section
Enter a number:4
Number is:4
Inside the public section
Enter a number:5
Number is:5
Inside the protected section
Enter a number:6
Number is:6
```

8.10 DEFINING MEMBER FUNCTIONS

The member function must be declared inside the `class`. They can be defined as `(a) private` or `public` section and (b) `inside` or `outside` the `class`. The member functions defined inside the `class` are treated as inline function. If the member function is small then it should be defined inside the `class`. Otherwise, it should be defined outside the `class`.

If function is defined outside the `class`, its prototype declaration must be done inside the `class`. While defining the function, scope access operator and `class` name should precede the function name. The following programs illustrate all about member function and how to access the private member of the `class`.

8.10.1 Member Function Inside the `class`

Member function inside the `class` can be declared in public or private section. The following program illustrates the use of a member function inside the `class` in public section.

8.6 Write a program to access private members of a `class` using member function.

```
#include<iostream.h>
#include<conio.h>
class item
{
  private: // private section starts
  int codeno;
  float price;
  int qty;
  public: // public section starts
  void show() // member function
  {
  codeno=125; // access to private members
  price=195;
  qty=200;
  cout<<"\n Codeno =”<<codeno;
  cout<<"\n Price =”<<price;
  cout<<"\n Quantity=”<<qty;
  }
};
int main()
{
  clrscr();
  item one; // object declaration
  one.show(); // call to member function
  return 0;
}
```

OUTPUT

Codeno =125
Price =195
Quantity=200

Explanation: In the above program, the member function `show()` is defined inside the `class` in public section. In function `main()`, object one is declared. We know that an object has a permission to access the public members of the `class`. The object `one` invokes the public member function `show()`. The public member function can access the private members of the same `class`. The function `show()` initializes the private member variables and displays the contents on the console. For the sake of understanding only one function is defined.

In the above program the member function is defined inside the `class` in public section. Now the following program explains how to define private member function inside the `class`.

8.10.2 Private Member Function

In the last section, we learned how to access private data of a `class` using public member function. It is also possible to declare a function in private section like data variables. To execute private member function, it must be invoked by public member function of the same `class`. A member function of a `class` can invoke any other member function of its own `class`. This method of invoking function is known as nesting of member function. When one member function invokes other member function, the frequent method of calling function is not used. The member function can be invoked by its name terminated with a semicolon only like normal function. The following program illustrates this point.

8.7 Write a program to declare private member function and access it using public member function.

```
#include<iostream.h>
#include<conio.h>
struct item
{
  private: // private section starts
  int codeno;
  float price;
  int qty;
  void values() // private member function
  {
  codeno=125;
  price=195;
  qty=200;
  }
  public: // public section starts
  void show() // public member function
  {
  values(); // call to private member functions
  cout<<"\n Codeno ="<<codeno;
  cout<<"\n Price ="<<price;
  cout<<"\n Quantity="<<qty;
  }
};
int main()
{
  clrscr();
  item one; // object declaration

  // one.values(); // not accessible

  one.show(); // call to public member function
```

```
  return 0;
}
```

OUTPUT

```
Codeno =125
Price =195
Quantity=200
```

Explanation: In the above program, the private section of a `class` item contains one-member function `values()`. The function `show()` is defined in public section. In function `main()`, `one` is an object of `class item`. The object `one` cannot access the private member function. In order to execute the private member function, the private function must be invoked using public member function. In this example, the public member function `show()` invokes the private member function `values()`. In the invocation of function `values()`, object name and operator are not used.

8.10.3 Member Function Outside the `class`

In the previous examples, we observed that the member functions are defined inside the `class`. The function prototype is also not declared. The functions defined inside the `class` are considered as inline functions. If a function is small, it should be defined inside the `class` and if large it must be defined outside the `class`. To define a function outside the `class`, following care must be taken.

a) The prototype of function must be declared inside the `class`.
b) The function name must be preceded by `class` name and its `return` type separated by scope access operator.

The following example illustrates the function defined outside the `class`.

8.8 Write a program to define member function of `class` outside the `class`.

```
#include<iostream.h>
#include<conio.h>
class item
{
  private: // private section starts
  int codeno; // member data variables
  float price;
  int qty;
  public: // public section starts
  void show (void); // prototype declaration
}; // end of class
void item:: show() // definition outside the class
{
  codeno=101;
  price=2342;
  qty=122;
```

```
    cout<<"\n Codeno ="<<codeno;
    cout<<"\n Price ="<<price;
    cout<<"\n Quantity="<<qty;
}
int main()
{
    clrscr();
    item one; // object declaration
    one.show(); // call to public member function
    return 0;
}
```

OUTPUT

Codeno =101
Price =2342
Quantity=122

Explanation: In the above program, the prototype of function `show()` is declared inside the `class` terminated by `class` definition. The body of function `show()` is defined inside the `class`. The `class` name that it belongs to and its return type precede the function name. The function declarator of function `show()` is as follows:

```
void item:: show()
```

where `void` is return type; that is function is not returning a value. The item is a `class` name. Scope access operator separates the `class` name and function name, followed by the body of function that is defined.

8.11 CHARACTERISTICS OF MEMBER FUNCTIONS

(1) The difference between member and normal function is that the formal function can be invoked freely, whereas the latter function only by using an object of the same `class`.
(2) The same function can be used in any number of classes. This is possible because the scope of the function is limited to their classes and cannot overlap one another.
(3) The private data or private function can be accessed by public member function. Other functions have no access permission.
(4) The member function can invoke one another without using any object or dot operator.

8.12 OUTSIDE MEMBER FUNCTION AS INLINE

In Chapter 7, we learned how inline mechanism is useful for small function. It is good practice to declare function prototype inside the `class` and definition outside the `class`. The inline mechanism reduces overhead relating to access the member function. It provides better efficiency and allows quick execution of functions. An inline member function is similar to macros. Call to inline function in the program, puts the function code in the caller program. This is known as inline expansion. Inline functions are also called as open subroutines because

their code is replaced at the place of function call in the caller function. The normal functions are known as closed subroutines because when such functions are called, the control passes to the function.

By default, all member functions defined inside the `class` are inline function. The member function defined outside the `class` can be made inline by prefixing the keyword `inline` to function declarator as shown in Figure 8.3.

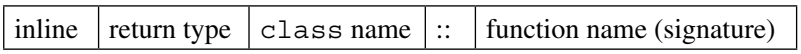

Fig. 8.3 Inline function outside the `class`

The `inline` is a keyword and acts as function qualifier. The `return` type is functions return type; that is the function returns values of this type. The `class` name is the name of `class` that the function belongs to. Scope access operator separates `class` name and function name. The signature means argument list passed function. The following program illustrates inline function outside the `class`.

8.9 Write a program to make an outside function as inline.

```
#include<iostream.h>
#include<conio.h>
class item
{
  private: // private section starts
  int codeno; // member data variables
  float price;
  int qty;
  public: // public section starts
  void show (void); // prototype declaration
}; // end of class
inline void item:: show() // outside inline function
{
  codeno=213;
  price=2022;
  qty=150;
  cout<<"\n Codeno ="<<codeno;
  cout<<"\n Price ="<<price;
  cout<<"\n Quantity="<<qty;
}
int main()
{
  clrscr();
  item one; // object declaration
```

```
    one.show(); // call to public member function (inline)
    return 0;
}
```

OUTPUT

Codeno =213
Price =2022
Quantity=150

Explanation: The above program is same as last one. The only difference is that the function `show()` is defined as inline outside the `class`. The function declarator is as follows `inline void item::show()`.

8.13 RULES FOR INLINE FUNCTIONS

(1) Inline function should be used rarely. It should be applied only at appropriate circumstances.
(2) Inline function can be used when the member function contains few statements. For example,

```
inline int item :: square (int x)
{
    return (x*x);
}
```

(3) If function takes more time to execute, then it must be declared as inline. The following inline function cannot be expanded as inline.

```
inline void item:: show()
{
    cout<<"\n Codeno ="<<codeno;
    cout<<"\n Price ="<<price;
    cout<<"\n Quantity="<<qty;
}
```

The member function that performs input and output operation requires more times. Inline functions have one drawback, the entire code of the function is placed at the point of call in caller function and it must be noted at compile time. Therefore, inline functions cannot be placed in standard library or run time library.

8.14 DATA HIDING OR ENCAPSULATION

Data hiding is also known as encapsulation. It is a procedure of forming objects. An encapsulated object is often called as an abstract data type. We need to encapsulate data, because programmer often makes various mistakes and the data get changed accidentally. Thus, to protect

data, we need to construct a secure and impassable wall to protect the data. Data hiding is nothing but making data variable of the `class` or `struct` `private`. Thus, private data cannot be accessed directly by the object. The objects using public member functions of the same `class` can access the private data of the `class`. The keywords `private` and `protected` are used for hiding the data. Table 8.2 shows the brief description of access specifiers. The following program explains data hiding:

8.10 Write a program to calculate simple interest. Hide the data elements of the `class` using `private` keyword.


```
#include<iostream.h>
#include<conio.h>
class interest
{
  private :
  float p_amount; // principle amount
  float rate; // rate of interest
  float period; // numbers of years
  float interest;
  float t_amount; // total amount

  public :
  void in()
  {
  cout<<" Principle Amount : "; cin>>p_amount;
  cout<<" Rate of Interest : "; cin>>rate;
  cout<<" Number of years : "; cin>>period;
  interest=(p_amount*period*rate)/100;
  t_amount=interest+p_amount;
  }
  void show()
  {
  cout<<"\n Principle Amount : "<<p_amount;
  cout<<"\n Rate of Interest : "<<rate;
  cout<<"\n Number of years : "<<period;
  cout<<"\n Interest : "<<interest;
  cout<<"\n Total Amount : "<<t_amount;
  }
};
int main()
{
  clrscr();
  interest r;
```

```
    r.in();
    r.show();
    return 0;
}
```

OUTPUT

```
Principle Amount : 5000
Rate of Interest : 2
Number of years : 3

Principle Amount : 5000
Rate of Interest : 2
Number of years : 3
Interest : 300
Total Amount : 5300
```

Explanation: In the above program, the `class interest` is declared with the data members `p_amount`, `float rate`, `period`, `4`, and `t_amount` of `float` type. These entire data elements are declared in private section; hence, it is hidden (encapsulated) and cannot be directly accessed by the object. Here, member functions `in()` is used to read data through the keyboard. The function `show()` is used to display the contents of the variables.

8.11 Write a program to declare `class` with private, public and private sections. Declare object and access data elements of these different sections.

```
#include<iostream.h>
#include<conio.h>
class access
{
  private :
  int p;
  void getp()
  {
  cout<<" In pget() enter value of p :";
  cin>>p;
  }
  public:
  int h;
  void geth()
  {
  cout<<" In geth() : "<<endl;
  getp();
  getm();
```

```
    cout<<" p = "<<p <<" h = "<<h <<" m = "<<m;
    }
    protected :
    int m;
    void getm()
    {
    cout<<" In mget() enter value of m : ";
    cin>>m;
    }
};
int main()
{
    clrscr();
    access a; // object declaration
    // a.p=2; // access to private member is not possible.
    // a.pget() // -----------" -----------------
    // a.m=5; // access to protectd member is not possible
    // a.mget(); // ----------------" ---------------------
    a.h=4; // direct access to public member is possible
    a.geth();
    return 0;
}
```

OUTPUT

In `geth()`**:**
In `pget()` **enter value of p:7**
In `mget()` **enter value of m: 4**
p = 7 h = 4 m = 4

Explanation: In the above program, the `class access` is declared with `private`, `protected`, and `public` sections. Each section holds one integer variable and one member function. The object cannot directly access the data variable and member function of the private and protected sections. The object can access only the `public` section of the `class` and through `public` section it can access the `private` or `protected` section. Here, the object `an` accesses the `public` variable `h` and initializes it with 4 whereas the `private` and `protected` variables are accessed using member functions. The function `getp()` and `getm()` are invoked by the `public` member function `geth()`. The `geth()` also displays the contents of the variables on the screen.

8.15 CLASSES, OBJECTS, AND MEMORY

Objects are the identifiers declared for `class` data type. Object is a composition of one more variables declared inside the `class`. Each object has its own copy of public and private data members.

An object can access to its own copy of data members and have no access to data members of other objects.

Only declaration of a `class` does not allocate memory to the `class` data members. When an object is declared, memory is reserved for only data member and not for member functions.

Consider the following program.

8.12 Write a program to declare objects and display their contents.

```
#include<iostream.h>
#include<constream.h>
class month
{
  public:
  char *name;
  int days;
}; // end of class
int main()
{
  clrscr();
  month M1,M3; // object declaration
  M1.name="January";
  M1.days=31;
  M3.name="March";
  M3.days=31;
  cout<<"\n Object M1 ";
  cout<<"\n Month name : "<<M1.name <<" Address :"<<(unsigned)&M1.
  name;
  cout<<"\n Days :" <<M1.days <<"\t\t Address : "<<(unsigned)&M1.
  days;
  cout<<"\n\n Object M3 ";
  cout<<"\n Month name : "<<M3.name <<"\t Address : "<<(unsigned)&M3.
  name;
  cout<<"\n Days :" <<M3.days <<"\t\t Address : "<<(unsigned)&M3.
  days;
  return 0;
}
```

OUTPUT

Object M1
Month name : January Address :65522
Days : 31 Address : 65524
Object M3
Month name : March Address : 65518
Days : 31 Address : 65520

Explanation: M1 and M3 are objects of `class` month. Separate memory is allocated to each object. The contents and address of the member variables are displayed in the output. Figure 8.4 shows it more visibly.

From the last program it is clear that memory is allocated for data members. What about functions? Member functions are created and memory is allocated to them only once when a `class` is declared. All objects of a `class` access the same memory location where member functions are stored. Hence, separate copies of member functions are not present in every object like member variables, which is illustrated in the following program and Figure 8.5.

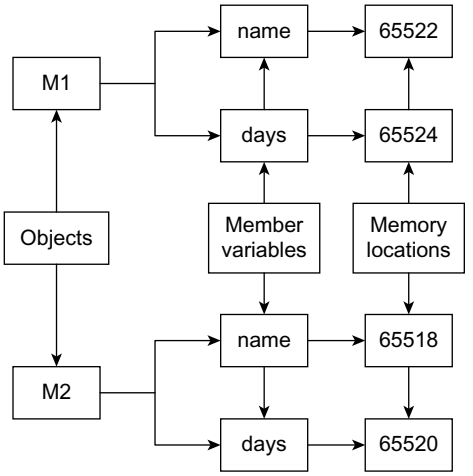

Fig. 8.4 Memory occupied by objects

8.13 Write a program to display the size of the objects.

```
#include<iostream.h>
#include<constream.h>
class data
{
  long i; // By default private
  float f;
  char c;
};
int main()
{
  clrscr();
  data d1,d2;
  cout<<endl<<" Size of object d1 = "<<sizeof(d1);
  cout<<endl<<" Size of object d2 = "<<sizeof(d2);
  cout<<endl<<" Size of class ="<<sizeof(data);
  return 0;
}
```

OUTPUT

Size of object d1 = 9
Size of object d2 = 9
Size of class = 9

Explanation: In the above program, the `class data` has three member variables of `long`, `float`, and `char` type. The `d1` and `d2` are objects of the `class data`. The `sizeof` operator displays the size of objects. The size of any object is equal to the sum of sizes of all the data members of the `class`. In the `class` data, the data type `long` occupies 4 bytes, `float` occupies 4 bytes, and `char` occupies 1 byte. Their sum is 9, which equals the size of an individual object.

The member functions are not considered in the size of the object. All the objects of a `class` use the same member functions. Only one copy of member function is created and stored in the memory whereas each object has its own set of data members.

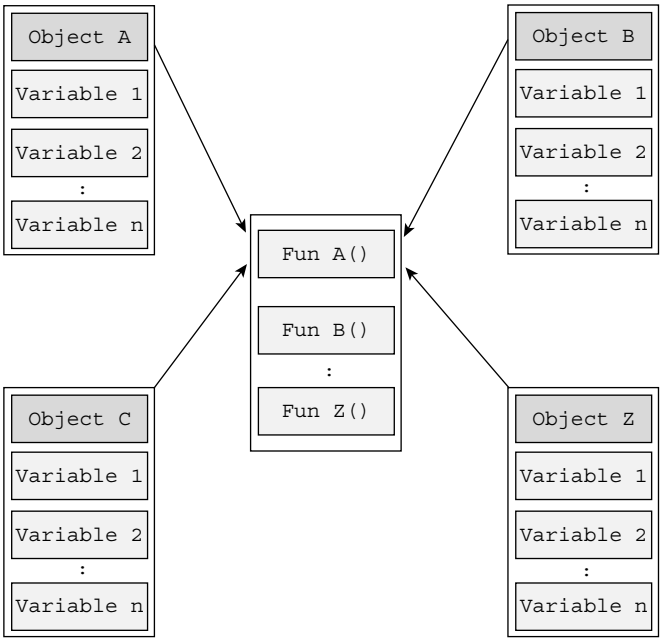

Fig. 8.5 Data members and member functions in memory

8.16 `static` MEMBER VARIABLES

We noticed that each object has a separate set of data member variables in memory. The member functions are created only once and all objects share the functions. No separate copy of function of each object is created in the memory like data member variables.

It is possible to create common member variables like function using the `static` keyword. Once a data member variable is declared as `static`, only one copy of that member is created for the whole `class`. The `static` is a keyword that is used to preserve the value of a variable. When a variable is declared as `static` it is initialized to zero. A `static` function or data element is only recognized inside the scope of the present compile.

In earlier version of Turbo C++, it was not necessary to explicitly define `static` data members. It was linkers' responsibility to find undefined `static` data. The linker would implicitly define the `static` data and allocate the required memory without showing error message. In new versions of Turbo C++, it is necessary to explicitly define `static` members.

```
Syntax:
static <variable definition> ;
static <function definition>;
```

If a local variable is declared with `static` keyword, it preserves the last value of the variable. A `static` data item is helpful when all the objects of the `class` should share a common data. The `static` data variable is accessible within the `class`, but its value remains in the memory throughout the whole program (Figure 8.6).

```
Examples:
static int c;
static void display() {}
a) int sumnum :: c=0;
```

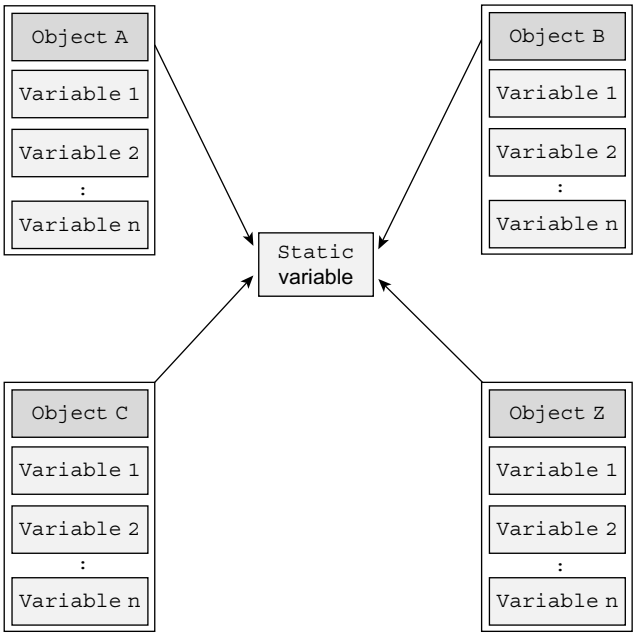

Fig. 8.6 `static` members in memory

The `class` and scope of the `static` member variable is defined outside the `class` declaration as per the statement `(a)`. The reasons are as follows:

(1) The `static` data members are associated with the `class` and not with any object.
(2) The `static` data members are stored individually rather than an element of an object.
(3) The `static` data member variable must be initialized otherwise the linker will generate an error.

(4) The memory for `static` data is allocated only once.
(5) Only one copy of `static` member variable is created for the whole `class` for any number of objects. All the objects have common `static` data member.

8.14 Write a program to declare `static` data member. Display the value of `static` data member

```
#include<iostream.h>
#include<constream.h>
class number
{
  static int c;

  public:
  void count()
  {
  ++c;
  cout<<"\n c="<<c;
  }
};
int number :: c=0; // initialization of static member variable
int main()
{
  number a,b,c;
  clrscr();
  a.count();
  b.count();
  c.count();
  return 0;
}
```

OUTPUT

```
c=1
c=2
c=3
```

Explanation: In the above program, the `class number` has one `static` data variable `c`. The `count()` is a member functions' increment value of `static` member variable c by one when called. The statement `int number::c=0` initializes the `static` member with 0. It is also possible to initialize the `static` data members with other values. In the function `main()` `a`, `b`, and `c` are three objects of `class number`. Each object calls the function `count()`. At each call to the function `count()`, the variable c gets incremented and the `cout` statement displays the value of the variable c. The objects `a`, `b`, and `c` share the same copy of `static` data member `c`.

8.15 Write a program to show the difference between `static` and non-`static` member variables.

```
#include<iostream.h>
#include<conio.h>
class number
{
   static int c; // static variable
   int k; // non-static variable
   public:
   void zero()
   {
   k=0;
   }
   void count()
   {
   ++c;
   ++k;
   cout<<"\n Value of c = "<<c <<" Address of c = "<<(unsigned)&c;
   cout<<"\n Value of k = "<<k <<" Address of k = "<<(unsigned)&k;
   }
};
int number :: c=0; // initialization of static member variable
int main()
{
   number A,B,C;
   clrscr();
   A.zero();
   B.zero();
   C.zero();
   A.count();
   B.count();
   C.count();
   return 0;
}
```

OUTPUT

Value of c = 1 Address of c = 11138
Value of k = 1 Address of k = 85524
Value of c = 2 Address of c = 11138
Value of k = 1 Address of k = 65522
Value of c = 3 Address of c = 11138
Value of k = 1 Address of k = 65520

Explanation: This program compares between `static` and non-`static` member variables. The `class number` has two member variables `c` and `k`. The variable `c` is declared as `static` and `k` as normal variable. The function `zero()` is used to initialize the variable `k` with zero. The `static` member variable `c` is initialized with zero as follows:

```
int number::c=0; // initialization of static member variable
```

The function `count()` is used to increment values of c and k. In function `main()`, A, B, and C are objects of `class` number. The function `zero()` is invoked three times by object A, B, and C. Each object has its own copy of variable `k`, and, hence, each object invokes the function `zero()` to initialize its copy of `k`. The `static` member variable c is common among the object A, B, and C. Hence, it is initialized only once. Figure 8.7 shows the object and member variables in memory.

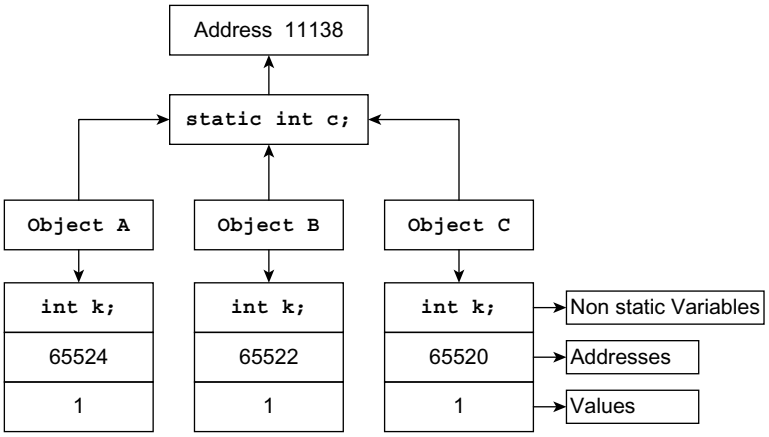

Fig. 8.7 `static` and non-`static` members

8.16 Write a program to enter a number. Count the total number of digits from 0 to 9 occurring from 1 to entered number.

```
#include<iostream.h>
#include<conio.h>
class digit
{
  static int num[10];
  public :
  void check(int n);
  void show();
  void input();
  void ini();
};
int digit :: num[]={0,0,0,0,0,0,0,0,0,0};
void digit :: show()
```

```
{
  for (int j=0;j<10;j++)
  {
  if (num[j]==0) continue;
  cout<<"\n Number "<<j <<" occurs " <<num[j] <<" times";
  }
}
void digit :: ini()
{
  for (int k=0;k<10;k++)
  num[k]=0;
}
void digit :: input()
{
  int x,y;
  cout<<endl<<"\n Enter a Number : ";
  cin>>y;
  check(y);
}
void digit :: check (int u)
{
  int m;
  while (u!=0)
  {
  m=u%10;
  num[m]++;
  u=u/10;
  }
}
int main()
{
  clrscr();
  digit d;
  // d.ini();
  d.input();
  d.show();
  return 0;
}
```

OUTPUT

Enter Numbers Number : 22151
Number 1 occurs 2 times
Number 2 occurs 2 times
Number 5 occurs 1 times

Explanation: In the above program, the `class digit` is declared with one `static array` member `num[10]` and four member functions `check()`, `show()`, `input()`, and `ini()`. The function `input()` reads an integer through the keyboard. The entered number is passed to function `check()`. The function `check()` is invoked by function `input()`. The function `check()` separates individual digits of the entered number using repetitive modular division and division operation. The separated digits are counted and the count value is stored in the array `num[10]` according to the element number. The function `show()` displays the contents of array `num[]`. The function `ini()` is declared and when called initializes all array elements with zero. In case the array is not declared as `static`, this function is useful. Here, in this program the array is `static`, hence we initialized it with the statement `int digit::num[]={0,0,0,0,0,0,0,0,0,0}`. If this statement is removed, we need to call the function `ini()`.

8.17 `static` MEMBER FUNCTIONS

Like member variables, function can also be declared as `static`. When a function is defined as `static`, it can access only `static` member variables and functions of the same `class`. The not-`static` members are not available to these functions. The `static` member function declared in public section can be invoked using its `class` name without using its objects. The `static` keyword makes the function free from the individual object of the `class` and its scope is global in the `class` without creating any side effect for other part of the program. The programmer must follow the following points while declaring `static` function:

1) Just one copy of a `static` member is created in the memory for entire `class`. All objects of the `class` share the same copy of `static` member.
2) `static` member function can access only `static` data members or functions.
3) `static` member function can be invoked using `class` name.
4) It is also possible to invoke `static` member functions using objects.
5) When one of the objects changes the value of data member variables, the effect is visible to all the object of the `class`.

8.17 Write a program to declare `static` member functions and call them from the `main()` function.

```
#include<iostream.h>
#include<conio.h>
class bita
{
   private :
   static int c;
   public :
   static void count() { c++; }
   static void display()
   {
   cout<<"\nValue of c : "<<c;
   }
};
```

```
int bita ::c=0;
int main()
{
   clrscr();
   bita:: display();
   bita::count();
   bita::count();
   bita::display();
   return 0;
}
```

OUTPUT

Value of c : 0
Value of c : 2

Explanation: In the above program, the member variables c and functions of class bita are static. The function count() when called increases the value of static variable c. The function display() prints the current value of the variable c. The static function can be called using class name and scope access operator from the following statements:

```
bita::count(); // invokes count() function
bita::display(); // invokes display() function
```

8.17.1 static Private Member Function

A static member function can also be declared in private section. The private static function must be invoked using static public function. The following program illustrates the point.

8.18 Write a program to define private static member function and invoke it.

```
#include<iostream.h>
#include<conio.h>
class bita
{
   private :
   static int c;
   static void count() { c++; }
   public:
   static void display()
   {
   count(); // Call to private static member function
   cout<<"\nValue of c : "<<c;
   }
};
int bita ::c=0;
int main()
```

```
{
   clrscr();
   bita:: display();
   bita::display();
   return 0;
}
```

OUTPUT

Value of c : 1
Value of c : 2

Explanation: In the above program, `count()` is a private `static` member function. The public `static` function `display()` invokes the private `static` function `count()`. The function `display()` also displays the value of `static` variable `c`.

8.17.2 `static` Public Member Variable

The `static` public member variable can also be initialized in function `main()` like other variables. The `static` member variable using `class` name and scope access operator can be accessed. The scope access operator is also used when variables of same name are declared in global and local scope, which is illustrated in the following program:

8.19 Write a program to declare `static` public member variable, global and local variable with the same name. Initialize and display their contents.

```
#include<iostream.h>
#include<constream.h>
int c=11; // global variable
class bita
{
   public:
   static int c;
};
int bita ::c=22; // class member variable
int main()
{
   clrscr();
   int c=33; // local variable
   cout<<"\nClass member c = "<<bita::c;
   cout<<"\nGlobal variable c = "<<::c;
   cout<<"\nLocal variable c = "<<c;
   return 0;
}
```

OUTPUT

```
Class member c = 22
Global variable c = 11
Local variable c = 33
```

Explanation: In the above program, the variable c is declared and initialized in three different scopes such as global, local, and inside the class. The variable c declared inside is static variable and initialized to 22. The global variable c is initialized to 11 and local variable c is initialized to 33.

static member variable: The value of static variable is displayed using variable name preceded by class name and scope access operator as per the statement cout<<"\nClass member c = "<<bita::c;.

Global variable: The global variable can be access using variable name preceded by scope access operator from the statement cout<<"\nGlobal variable c = "<<::c;.

Local variable: The local variable can be access only by putting its name as per the statement cout<<"\nLocal variable c = "<<c;.

8.18 static OBJECT

In C, it is common to declare variable static and it gets initialized to zero. The object is a composition of one or more member variables. There is a mechanism called constructor to initialize member variables of the object to desired values. The constructors are explained in the next chapter. The static keyword can be used to initialize all class data member variables to zero. Declaring object itself as static can do this. Thus all its associated members get initialized to zero. The following program illustrates the working of static object.

8.20 Write a program to declare static object. Display its contents.

```
#include<iostream.h>
#include<constream.h>
class bita
{
  private:
  int c;
  int k;
  public :
  void plus()
  {
  c+=2;
  k+=2;
  }
  void show()
  {
  cout<<" c= "<<c<<"\n";
```

```
   cout<<" k= "<<k;
   }
};
int main()
{
   clrscr();
   static bita A;
   A.plus();
   A.show();
   return 0;
}
```

OUTPUT

c= 2
k= 2

Explanation: The `class bita` has two member variables c and k and two member functions `plus()` and `show()`. In function `main()`, the object A is declared. It is also declared as `static`. The data members of object A are initialized to zero. The function `plus()` is invoked, which adds 2 to the values of `c` and `k`. The function displays values of c and k. Declare object `static` does not mean that entire `class` is `static` including member function. The declaration of `static` object removes garbage of its data members and initializes them to zero.

8.19 ARRAY OF OBJECTS

Arrays are collection of similar data types. Arrays can be of any data type including user-defined data type created using `struct`, `class`, and `typedef` declarations. We can also create an array of objects. The array elements are stored in continuous memory locations as shown in Figure 8.8. Consider the following example.

```
class player
{
   private:
   char name [20];
   int age;
   public:
   void input (void);
   void display (void);
};
```

The `player` is a user-defined data type and can be used to declare an array of object of type `player`. Each object of an array has its own set of data variables.

```
player cricket[5];
player football [5];
player hockey [5];
```

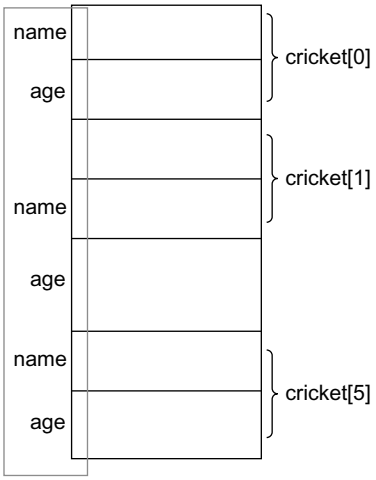

Fig. 8.8 Arrays of objects

As shown above, arrays of object of `type player` are created. The array `cricket[5]` contains `name` and `age` information for five objects. The next two declarations can maintain the same information for other player in arrays `hockey[5]` and `football[5]`. These arrays can be initialized or accessed like an ordinary array. The following program describes the working of array of objects.

8.21 Write a program to declare the array of objects. Initialize and display the contents of arrays.

```
#include<iostream.h>
#include<constream.h>
class player
{
  private:
  char name [20];
  int age;
  public:
  void input (void);
  void display (void);
};
void player :: input()
{
  cout<<"\n Enter Palyer name : ";
  cin>>name;
  cout<<" Age : ";
  cin>>age;
}
void player :: display()
```

```
{
   cout<<"\n Player name : "<<name;
   cout<<"\n Age : "<<age;
}
int main()
{
   clrscr();
   player cricket[3]; // array of objects
   cout<<"\n Enter Name and age of 3 players ";
   for (int i=0;i<3;i++)
   cricket[i].input();
   for (i=0;i<3;i++)
   cricket[i].display();
   return 0;
}
```

OUTPUT

```
Enter Name and age of 3 players
Enter Palyer name : Sachin
Age : 29
Enter Palyer name : Rahul
Age : 28
Enter Palyer name : Saurav
Age : 30
Player name : Sachin
Age : 29
Player name : Rahul
Age : 28
Player name : Saurav
Age : 30
```

Explanation: In the above program, the member function `input()` reads information of players. The `display()` function displays information on the screen. In function `main()` the statement `player cricket[3];` creates an array `cricket[3]` of three objects of type `player`. The `for` loops are used to invoke member function `input()` and `display()` using array of objects.

8.20 OBJECTS AS FUNCTION ARGUMENTS

Similar to variables, object can be passed to functions. The following are the three methods to pass argument to a function:

a) Pass-by-value – A copy of object (actual object) is sent to function and assigned to the object of callee function (formal object). Both actual and formal copies of objects are

stored at different memory locations. Hence, changes made in formal object are not reflected to actual object.

b) Pass-by-reference – Address of object is implicitly sent to function.

c) Pass-by-address – Address of the object is explicitly sent to function.

In pass-by-reference and pass-by-address methods, an address of actual object is passed to the function. The formal argument is reference/pointer to the actual object. Hence, changes made in the object are reflected to actual object. These two methods are useful because an address is passed to the function and duplicating of object is prevented.

The following examples illustrate both the methods of passing objects to the function as an argument.

8.22 Write a program to pass objects to the function by pass-by-value method.


```
#include<iostream.h>
#include<conio.h>
class life
{
  int mfgyr;
  int expyr;
  int yr;
  public :
  void getyrs()
  {
  cout<<"\nManufacture Year : ";
  cin>>>mfgyr;
  cout<<"\n Expiry Year : ";
  cin>>>expyr;
  }
  void period ( life);
};
void life :: period (life y1)
{
  yr=y1.expyr-y1.mfgyr;
  cout<<"Life of the product : " <<yr <<" Years";
}
int main()
{
  clrscr();
  life a1;
  a1.getyrs();
  a1.period(a1);
  return 0;
}
```

OUTPUT

Manufacture Year : 1999
Expiry Year : 2002
Life of the product : 3 Years

Explanation: In the above program, the `class life` is declared with three member integer variables. The function `getrys()` reads the integers through the keyboard. The function `period()` calculates the difference between the two integers entered. In the function `main()`, `a1` is an object to the `class life`. The object `a1` calls the function `getyrs()`. Immediately after this, the same object (a1) is passed to the function `period()`. The function `period()` calculates the difference between two integers (dates) using the two data members of the same `class`. Thus, an object can be passed to the function. To pass an object by reference, the prototype of function `period()` should be as follows:

```
void period ( life &);
```

8.23 Write a program to pass objects to the function pass-by-address method.

```
#include<iostream.h>
#include<conio.h>
class life
{
  int mfgyr;
  int expyr;
  int yr;
  public :

  void getyrs()
  {
  cout<<"\nManufacture Year : ";
  cin>>>mfgyr;
  cout<<"\n Expiary Year : ";
  cin>>>expyr;
  }
  void period ( life*);
};
void life :: period (life *y1)
{
  yr=y1->expyr-y1->mfgyr;
  cout<<"Life of the product : " <<yr;
}
int main()
{
  clrscr();
  life a1;
```

```
   a1.getyrs();
   a1.period(&a1);
   return 0;
}
```

OUTPUT

Manufacture Year : 1999
Expiry Year : 2002
Life of the product : 3 Years

Explanation: The above program is same as previous one. In this program, the object `a1` is passed by address. Consider the following statements.

```
a) void period ( life*);
b) a1.period (&a1);
a) Yr=y1->expyr-y1->mfgyr;
```

The statement `(a)` is the prototype of the function `period()`. In this statement, the deference operator `(*)` indicates that the function will accept address of the actual argument. The statement `(b)` is used to pass the address of the argument to the function `period()`. The statement `(c)` is used to access the member variables of the `class`. When an object is a pointer to the `class` members, then its elements are accessed by using arrow `(->)` operator. In this case, use of a `dot operator (.)` is invalid.

8.21 `friend` FUNCTIONS

The central idea of encapsulation and data hiding concept is that any non-member function has no access permission to the private data of the `class`. The private members of the `class` are accessed only from member functions of that `class`.

C++ allows a mechanism, in which a non-member function has access permission to the private members of the `class`. This can be done by declaring a non-member function friend to the `class` whose private data is to be accessed. The `friend` is a keyword. Consider the following example.

```
class ac
{ private:
    char name [15];
    int acno;
    float bal;
    public:
    void read();
    friend void show-
bal();
};
```

The keyword `friend` must precede the function declaration, whereas function declarator must not. The function can be defined at anyplace in the program like normal function. The function can be declared as friend function in one or more classes. The keyword `friend` or `scope access operator` must not precede the definition of the `friend` function. The declaration of friend function is done inside the `class` in private or public part and a function can be declared as friend function in any number of classes. These functions use objects as arguments. Thus the following statement is wrong.

```
a) friend void :: showbal (ac a) // Wrong function defini-
   tion
   {
       statement1;
       statement2;
   }
```

The above declaration of function is wrong because the function declarator precedes the keyword friend.

The friend functions have the following properties:

a) There is no scope restriction for the friend function; hence, they can be called directly without using objects.
b) Unlike member functions of `class`, the friend cannot access the member directly. On the other hand it uses `object` and `dot` operator to access the private and public member variables of the `class`.
c) By default, friendship is not shared (mutual). For example, if `class` X is declared as friend of Y, this does not meant that Y has privileges to access private members of `class` X.
d) Use of friend functions is rare, since it violates the rule of encapsulation and data hiding.
e) The function can be declared in public or private sections without changing its meaning.

8.24 Write a program to access private data using non-member function. Use friend function.

```
#include<iostream.h>
#include<conio.h>
class ac
{
  private:
  char name[15];
  int acno;
  float bal;
  public:
  void read()
  {
  cout<<"\nName :" ;
```

```
    cin>>>name;
    cout<<"\nA/c No. :";
    cin>>>acno;
    cout<<"\n Balance :";
    cin>>>bal;
    }
    friend void showbal(ac ); // friend function declaration
};
void showbal (ac a)
{
    cout<<"\n Balance of A/c no. " <<a.acno <<" is Rs." <<a.bal;
}
int main()
{
    ac k;
    k.read();
    showbal(k); // call to friend function
    return 0;
}
```

OUTPUT

```
Name :Manoj
A/c No. :474
Balance :40000
Balance of A/c no. 474 is Rs.40000
```

Explanation: In the above program, `class ac` is declared. It has three member variables and one member function. Also, inside the `class ac`, `showbal()` is a function, which is declared as `friend` of the `class ac`. Once the outside function is declared as `friend` to any `class`, it gets an authority to access the private data of that `class`. The function `read()` reads the data through the keyboard such as name, account number, and balance. The `friend` function `showbal()` display the `balance` and `acno`.

8.25 Write a program to declare `friend` function in two classes. Calculate the sum of integers of both the classes using friend `sum()` function.

```
#include<iostream.h>
#include<conio.h>
class first;
class second
{
    int s;
    public :
```

```
   void getvalue()
   {
   cout<<"\nEnter a number : ";
   cin>>>s;
   }
   friend void sum (second, first);
};
class first
{
   int f;
   public :
   void getvalue()
   {
   cout<<"\nEnter a number : " ;
   cin>>>f;
   }
   friend void sum (second , first);
};
void sum (second d, first t)
{
   cout<<"\n Sum of two numbers : " <<t.f + d.s;
}
int main()
{
   clrscr();
   first a;
   second b;
   a.getvalue();
   b.getvalue();
   sum(b,a);
}
```

OUTPUT

Enter a number: 7
Enter a number: 8
Sum of two numbers: 15

Explanation: In the above program, two classes `first` and `second` are declared with one integer and one member function in each. The member function `getvalue()` of both classes reads integers through the keyboard. In both the classes the function `sum()` is declared as `friend`. Hence, this function has an access to the members of both the classes. Using `sum()`, function addition of integers is calculated and displayed.

8.26 Write a program to exchange values between two classes. Use `friend` functions.

```
#include<iostream.h>
#include<conio.h>
class second;
class first
{
  int j;
  public:
  void input()
  {
  cout<<"Enter value of j : ";
  cin>>>j;
  }
  void show (void)
  {
  cout<<"\n Value of J = ";
  cout<<j <<"\n";
  }
  friend void change (first &, second &);
};
class second
{
  int k;
  public :
  void input()
  {
  cout<<"\nEnter value of k : ";
  cin>>>k ;
  }
  void show (void)
  {
  cout<<" Value of K = ";
  cout<<k ;
  }
  friend void change (first & , second &);
};
void change ( first &x, second &y)
{
  int tmp=x.j;
  x.j=y.k;
  y.k=tmp;
}
```

```
main()
{
   clrscr();
   first c1;
   second c2;
   c1.input();
   c2.input();
   change (c1,c2);
   cout<<"\nAfter change values are" <<"\n";
   c1.show();
   c2.show();
   return 0;
}
```

OUTPUT

Enter value of j : 4
Enter value of k : 8
After change values are
Value of J = 8
Value of K = 4

Explanation: In the above program, two classes `first` and `second` are defined. Each `class` contains one integer variable and two member functions. The function `input()` is used to read an integer through the keyboard. The function `show()` is used to display the integer on the screen. The function `change()` is declared as friend function for both the classes. Passing values by reference of member variables of both the classes, values are exchanged.

8.27 Write a program to declare three classes. Declare integer array as data member in each `class`. Perform addition of two data member array into array of third `class`. Use friend function.

```
#include<iostream.h>
#include<conio.h>
class B;
class C;
class A
{
   int a[5];
   public :
   void input();
   friend C sum (A,B,C);
};
void A :: input()
{
```

```
   int k;
   cout<<"\n Enter five integers : ";
   for (k=0;k<5;k++)
   cin>>a[k];
}
class B
{
   int b[5];
   public :
   void input();
   friend C sum (A,B,C);
};
void B:: input()
{
   int k;
   cout<<"\n Enter five integers : ";
   for (k=0;k<5;k++)
   cin>>b[k];
}
class C
{
   int c[5];
   public :
   void show();
   friend C sum (A,B,C);
};
void C :: show()
{
   cout<<"\n\t Addition : ";
   for (int k=0;k<5;k++)
   cout<<" " <<c[k];
}
C sum (A a1 ,B b1, C c1)
{
   for (int k=0;k<5;k++)
   c1.c[k]=a1.a[k]+b1.b[k];
   return c1;
}
int main()
{
   clrscr();
   A a;
   B b;
   C c;
```

```
    a.input();
    b.input();
    c=sum(a,b,c);
    c.show();
}
```

OUTPUT

```
Enter five integers : 5 4 8 7 5
Enter five integers : 2 4 1 2 3
Addition : 7 8 9 9 8
```

Explanation: In the above program, three classes `A`, `B`, and `C` are declared. Each `class` contains single integer arrays as data member `a[5]`, `b[5]`, and `c[5]`, respectively. The `class` `A` and `B` contains member function `input()` to read integers. The function `sum()` is declared as `friend` in all the three classes. This function performs addition of arrays of `class` `A` and `B` and stores results in the array of `class` `C`. The result obtained is returned in `main()` where the return value is assigned to object `c`. In `main()` `a`, `b`, and `c` are objects of classes `A`, `B`, and `C`, respectively. The member function `show()` of `class` `C` displays the contents of object `c`.

8.21.1 `friend` Classes

It is possible to declare one or more functions as friend functions or an entire `class` can also be declared as friend `class`. When all the functions need to access another `class` in such a situation we can declare an entire `class` as friend `class`. The friend is not transferable or inheritable from one `class` to another. Declaring `class` `A` to be a friend of `class` `B` does not meant `class` `B` a friend of `class` `A`; that is, friendship is not exchangeable. The friend classes are applicable when we want to make available private data of a `class` to another `class`.

8.28 Write a program to declare friend classes and access the private data.

```
#include<iostream.h>
#include<constream.h>
class B;
class A
{
  private :
  int a;
  public :
  void aset() {a=30;}
  void show (B);
};
class B
{
  private :
```

```
  int b;
  public :
  void bset() { b=40 ;};
  friend void A :: show (B bb);
};
void A :: show (B b)
{
  cout<<"\n a = "<<a;
  cout<<"\n b = "<<b.b;
}
int main()
{
  clrscr();
  A a1;
  a1.aset();
  B b1;
  b1.bset();
  a1.show(b1);
}
```

OUTPUT

a = 30
b = 40

Explanation: In the above program, two `classes` `A` and `B` are declared. The `class` `A` is friend of `class` `B`. The member function of `class` `A` can access the data of `class` `B`. Thus, the `show()` function displays the values of data members of both the classes.

8.29 Write a program to demonstrate friend classes.

```
#include<iostream.h>
#include<conio.h>
class CPP;
class C
{
  private :
  int j;
  public :
  void set() { j=22; }
  friend CPP;
};
class CPP
```

```
{
  public :
  void joy(C a)
  {
  cout<<endl<<" j = "<<a.j;
  }
  void joya(C o)
  {
  cout<<endl<<" j = "<<o.j;
  }
};
int main()
{
  clrscr();
  C x;
  CPP y;
  x.set();
  y.joy(x);
  y.joya(x);
  return 0;
}
```

OUTPUT

j = 22
j = 22

Explanation: In the above program, `class C` and `class CPP` are declared as `friend`. The `class` CPP is declared as friend `class` of `class` C. Member function of `class CPP` can access the private data variables of the `class C`.

8.22 THE `const` MEMBER FUNCTIONS

The member functions of a `class` can also be declared as constant using `const` keyword. The constant functions cannot modify any data in the `class`. The `const` keyword is suffixed to the function prototype as well as in function definition. If these functions attempt to change the data, compiler will generate an error message.

8.30 Write a program to declare `const` member function and attempt any operation within it.

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

class A
{
```

```
   int c;
   public :
   void add (int a,int b) const
   {
   // c=a+b; // invalid statement
   a+b;
   cout<<"a+b = "<<_AX ;
   }
};
int main()
{
   clrscr();
   A a;
   a .add(5,7);
   return 0;
}
```

OUTPUT

a+b = 12

Explanation: In the above program, the `class A` is declared with one member variable (`c`) and one constant member function `add()`. The `add()` function is invoked with two integers. The constant member function cannot perform any operation. Hence, the expression `c=a+b` will generate an error. The expression `a+b` is valid and cannot alter any value. The result obtained from the equation `a+b` is displayed using CPU register.

8.23 THE VOLATILE MEMBER FUNCTION

In C++, one can declare a member function with volatile specifies. This step leads to call safely the volatile object. Calling volatile member function with volatile object is safe. This concept is supported with the following programming example.

8.31 Write a program to call a volatile member function from a volatile object.

```
#include<iostream.h>
#include<conio.h>
class A
{
   private:
   int x;
   public:
   void f() volatile // The volatile member function
   {
   int x=10;
   cout<<"Value of x:"<<++x;
   }
```

```
};
int main()
{
  clrscr();
  volatile A c; // The c is a volatile object
  c.f(); // Call a volatile member function safely
  return 0;
}
```

OUTPUT

Value of x:11

Explanation: The `volatile` member function `f()` is called from a volatile object c and the value of x is initialized to 10. Its value is increased by one and displayed on the screen.

8.24 RECURSIVE MEMBER FUNCTION

Like C, C++ also supports recursive feature; that is, a function is called repetitively by itself. The recursion can be used directly or indirectly. The direct recursion function calls to itself until the condition is true. In indirect recursion, a function calls to another function and then the called function calls to the calling function. Here, the recursion with member function is illustrated in the following program:

8.32 Write a program to calculate triangular number by creating a member function. Call it recursively.


```
#include<iostream.h>
#include<conio.h>
class num
{
  public :
  tri_num(int m)
  { int f=0;
  if (m==0)
  return(f);
  else
  f=f+m+tri_num(m-1);
  return (f);
  }
};
int main()
{
  clrscr();
  num a;
  int x;
```

```
   cout<<"\n Enter a number : ";
   cin>>x;
   cout<<"Triangular number of "<<x<<" is : "<<a.tri_num(x);
   return 0;
}
```

OUTPUT

Enter a number : 5
Triangular number of 5 is : 15

Explanation: In the above program, `class num` is declared with one member function `tri_num()`. This function is used to calculate the triangular number of the entered number. The triangular number is nothing but the sum from 1 to the number entered. In function `main()`, a number is read through the keyboard and it is passed to function `tri_num()`, which is invoked by the object `a` of `class num`. The `tri_num()` is invoked and `tri_num()` invokes itself repetitively till the value of `m` becomes 0. The variable `f` holds the cumulative total of successive numbers and `return()` statement returns value of `f` in function `main()`, where it displays triangular number on the screen.

8.25 LOCAL CLASSES

When classes are declared inside the function then such classes are called as local classes. The local classes have access permission to global variables as well as `static` variables. The global variables need to be accessed using scope access operator when the `class` itself contains member variable with same name as global variable. The local classes should not have `static` data member and `static` member functions. If at all they are declared, the compiler provides an error message. The following programs illustrate the local classes.

8.33 Write a program to define classes inside and outside `main()` function and access the elements.

```
#include<iostream.h>
#include<conio.h>
class A
{
   private :
   int a;
   public :
   void get()
   {
   cout<<"\n Enter value for a : ";
   cin>>>a;
   }
   void show()
   { cout<<endl<<" a = " <<a; }
};
```

```
main()
{
  clrscr();
  class B
  {
  int b;
  public :
  void get()
  {
  cout<<"\n Enter value for b : ";
  cin>>>b;
  }
  void show()
  {
  cout<<" b = " <<b;
  }
  };
  A j;
  B k;
  j.get();
  k.get();
  j.show();
  k.show();
  return 0;
}
```

OUTPUT

```
Enter value for a : 8
Enter value for b : 9
a = 8 b = 9
```

Explanation: In the above program, `class A` is declared before `main()` function as usual. The `class B` is declared inside the `main()` function. Both the functions have two member functions `get()` and `show()`. The `get()` function reads integers through the keyboard. The `show()` functions display the values of data members on the screen.

8.34 Write a program to declare global variables, read and display data using member functions.

```
#include<iostream.h>
#include<conio.h>
int j, k, l, m; // global variable
class A
{
  private :
```

```
   int a;
   int j;
   public :
   void get()
   {
   cout<<"\n Enter value for a,j,j and k : ";
   cin>>>a >>j>>::j >>k;
   }
   void show()
   {
   cout<<endl<<" a = " <<a <<" j = "<<j <<" ::j = "<<::j <<" k = "<<k ;
   }
};
int main()
{
   clrscr();
   class B
   {
   int b;
   int l;
   public :
   void get()
   {
   cout<<"\n Enter value for b,l,l and m : ";
   cin>>>b >>l>>::l>>m;
   }
   void show()
   {
   cout<<"\n b = " <<b <<" l = "<<l <<" ::l = "<<::l <<" m = "<<m;
   }
   };
   A x;
   B y;
   x.get();
   y.get();
   x.show();
   y.show();
   return 0;
}
```

OUTPUT

Enter value for a,j,j and k : 1 2 3 4
Enter value for b,l,l and m : 5 6 4 3
a = 1 j = 2 ::j = 3 k = 4
b = 5 l = 6 ::l = 4 m = 3

Explanation: The above program is so far same as previous one. In addition, in this program global variables `j`, `k`, `l`, and `m` are declared. The member functions `get()` and `show()` read and display values of member variables as well as global variables. Here, both the classes contain a single data member variable with the same name as global variables. Thus, to access the global variable where necessary, scope access operator is used. The output of the program is as shown in the above program.

8.26 `empty`, `static`, and `const` CLASSES

The classes without any data members or member functions are called as empty classes. These types of classes are not frequently used. The empty classes are useful in exception handling. For exception handling, refer the chapter 19 `exception handling`. The syntax of empty `class` is as follows:

EMPTY CLASSES
`class` nodata { }; `class` vacant { };

We can also precede the declaration of classes by the keywords `static`, `const`, `volatile`, etc. But there is no any effect in the `class` operations. Such declaration can be done as follows:

CLASSES AND OTHER KEYWORDS
`static class` boys { }; `const class` data { }; volatile `class` area { };

8.27 MEMBER FUNCTION AND NON-MEMBER FUNCTION

So far we used non-member function `main()` for declaring objects and calling member functions. Apart from `main()` function other non-member functions can also be used. The member function can also invoke a non-member function and vice versa. When a member function calls a non-member function, it is necessary to put its prototype inside the calling function or at the beginning of the program. It is a better practice to put prototype at the beginning of the program that is visual to the entire program. It is also possible to put definition of the non-member function before `class` declaration. This method allows member function to invoke outside non-member function without need of prototype declaration. But this approach creates problem when an outside non-member function attempts to invoke member function.

We know that member functions can be called using object of that `class`. If a non-member function is defined before `class` declaration, it is not possible to create an object in that function. Hence, the best choice is to put prototype of the non-member function at the beginning of the program that makes easy for both non-member function and member function to call each other. The following program explains practically whatever we learned about member function and non-member function in this section.

8.35 Write a program to call a member function using non-member function.

```
#include<iostream.h>
#include<conio.h>
void moon(void); // Function prototype declaration
class mem
{
  public :
  void earth() { cout<<"On earth"; }
};
int main()
{
  clrscr();
  mem k;
  moon();
  return 0;
}
void moon()
{
  mem j;
  j.earth();
  cout<<endl<<"On moon ";
}
```

OUTPUT

```
On earth
On moon
```

Explanation: In the above program, `moon()` is a non-member function and its prototype is declared at the beginning of the program. The function `main()` calls function `moon()`. In function `moon()`, object `j` of type `class mem` is declared and a member function `earth()` is invoked. Thus, non-member function calls the member function.

8.28 THE `main()` FUNCTION AS A MEMBER FUNCTION

We know that the function `main()` is the starting execution point of every C/C++ program. The `main()` can be used as a member function of the `class`. But the execution of program will not start from this member function. The compiler treats member function `main()` and the user-defined `main()` differently. No ambiguity is observed while calling function. The following program narrates this concept.

8.36 Write a program to make `main()` as a member function.

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

```
class A
{
  public:
  void main()
  {
  cout<<endl<<"In member function main()";
  }
};
int main()
{
  clrscr();
  A *a;
  a->main();
  return 0;
}
```

OUTPUT

In member function `main()`

Explanation: In the above program, `class A` is declared and has one member function `main()`. In the non-member function `main()`, the object `a` invokes the member function `main()` and a message is displayed as shown in the output.

8.29 OVERLOADING MEMBER FUNCTIONS

Member functions are also overloaded in the same fashion as other ordinary functions. We learned that overloading is nothing but a function is defined with multiple definitions with same function name in the same scope. The following program explains the overloaded member function.

8.37 Write a program to overload member function of a `class`.

```
#include<iostream.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>
class absv
{
  public :
  int num (int);
  double num (double);
};
int absv:: num (int x)
{
  int ans;
  ans=abs(x);
```

```
   return (ans);
}
double absv :: num (double d)
{
   double ans;
   ans=fabs(d);
   return (ans);
}
int main()
{
   clrscr();
   absv n;
   cout<<"\n Absolute value of -25 is "<<n.num(-25);
   cout<<"\n Absolute value of -25.1474 is "<<n.num(-25.1474);
   return 0;
}
```

OUTPUT

```
Absolute value of -25 is 25
Absolute value of -25.1474 is 25.1474
```

Explanation: In the above program, the `class absv` has a member function `num()`. The `num()` function is overloaded for integer and `double`. In function `main()` the object `n` invokes the member function `num()` with one value. The compiler invokes the overloaded function according to the value. The function returns the absolute value of the number.

8.30 OVERLOADING `main()` FUNCTIONS

In the last two subtitles, we learnt how to make `main()` as member function and how to overload member function. Like other member function, `main()` can be overloaded in the `class` body as a member function. The following program explains this concept:

8.38 Write a program to declare `main()` as a member function and overload it.

```
#include<conio.h>
#include<iostream.h>
class A
{
   public:
   void main(int i)
   {
   cout<<endl<<"In main (int) :"<<i;
   }
   void main (double f)
   {
```

```
   cout<<"\nIn main(double) :"<<f;
   }
   void main (char *s)
   {
   cout<<endl<<"In main (char ) : "<<s;
   }
};
int main()
{
   clrscr();
   A *a;
   a->main(5);
   a->main(5.2);
   a->main("C++");
   return 0;
}
```

OUTPUT

In `main` **(**`int`**) :5**
In `main` **(**`double`**) :5.2**
In `main` **(**`char` **) : C++**

Explanation: This program is same as the previous one. Here, the `main()` function is used as a member function and it is overloaded for integer, `float`, and character.

It is not possible to overload the non-member `main()` function, which is the source of the C/C++ program and hence the following program will not be executed and displays the error message "`Cannot overload 'main'`".

```
#include<iostream.h>
void main()
{ }
main (float x, int y)
{
   cout<<x<<y;
   return 0;
}
```

☞ **TIP**

The `main()` is the only function that cannot be overloaded.

8.31 THE `main()`, MEMBER FUNCTION, AND INDIRECT RECURSION

When a function calls itself, then this process is known as recursion or direct recursion. When two functions call each other repetitively, such type of recursion is known as indirect recursion.

Consider the following program and explanation to understand the indirect recursion using OOP. The program without using OOP is also described for programmers who are learning C++.

8.39 Write a program to call function `main()` using indirect recursion.

```
//Indirect Recursion Using OOP//
#include<iostream.h>
#include<conio.h>

int main (int);
class rec
{
   int j;
   public:
   int f;
   rec (int k, int i)
   {
   clrscr();
   cout<<"[ ";
   f=i;
   j=k;
   }
   ~rec()
   { cout<<"\b\b] Factorial of number : "<<f ; }
   void pass()
   {
   cout<<main (j--)<<" * ";
   }
};
rec a(5,1);
main (int x)
{
   if (x==0)
   return 0;
   a.pass();
   a.f=a.f*x;
   return x;
}
```

OUTPUT

[0 * 1 * 2 * 3 * 4 * 5] Factorial of number : 120

Explanation: In the above program, `class rec` is declared with `constructor, destructor, member function pass()`, and `two integer variables`. The integer variable `j is private` and `f` is `public`. The function `main()` is defined with one integer argument. Usually,

`main()` with arguments is used for the applications on the dos prompt. In this program, `main()` is called recursively by member function `pass()`. When a function call is made, the value of member data variable is decreased first and then passed. Thus, the `main()` passes value to itself. In function `pass()`, `main()` function is invoked and its return value is displayed. The public data member is directly used and by applying multiplication operation, factorial of a number is calculated.

Generally, objects are declared inside the function `main()`. But in this program function `main()` is used in recursion. Hence, if we put the object declaration statement inside the `main()`, in every call of `main()` object is created and the program will not run properly. To avoid this, the object is declared before `main()`.

Constructor is used to initialize data members as well as to clear the screen. Destructor is used to display the factorial value of the number. All the statements that we frequently put in `main()` are written outside of `main()`.

Before the `class` declaration, prototype of `main()` is given, because the member functions do not know about; and the prototype declaration provides information about `main()` to member function.

TIP

For C programmers the previous program in C style is explained as follows.

8.40 Write a program to call function `main()` using in-direct recursion in C style.

```
//Indirect Recursion in C style//
#include<iostream.h>
#include<conio.h>
#include<process.h>
int m=5;
int f=1;
int j;
main (int x)
{
  void pass (void);
  if (x==0)
  { clrscr();
  cout<<endl<<"Factorial of number ="<<f;
  return 0;
  }
  f=f*x;
  pass();
  return x;
}
void pass() { main (m--); }
```

OUTPUT

Factorial of number =120

Explanation: The logic of the program is same as the previous one. The user-defined function `pass()` has only one job to invoke function `main()`. The `if` conditions inside `main()` check the value of variable x. If value of `x` is zero, then `if` block is executed that displays factorial of the number and terminates the programs.

8.32 BIT FIELDS AND CLASSES

Bit field provides exact amount of bits required for storage of values. If a variable value is 1 or 0 we need a single bit to store it. In the same way, if the variable is expressed between 0 and 3, then the two bits are sufficient for storing these values. Similarly if a variable assumes values between 0 and 7, then three bits will be enough to hold the variable and so on. The number of bits required for a variable is specified by non-negative integer followed by a colon.

To hold the information, we use the variables. The variables occupy minimum one byte for `char` and two bytes for `integer`. Instead of using complete integer if bits are used, memory space can be saved. For example, to know the information about the vehicles, following information has to be stored in the memory:

(1) PETROL VEHICLE
(2) DIESEL VEHICLE
(3) TWO_WHEELER VEHICLE
(4) FOUR_WHEELER VEHICLE
(5) OLD MODEL
(6) NEW MODEL

In order to store the status of the above information, we may need two bits for the type of fuel as to whether the vehicle is of petrol or diesel type, three bits for its type as to whether the vehicle is two- or four-wheeler, and similarly, three bits for the model of the vehicle. Total bits required for storing the information would be 8 bits, that is one byte. It means that the total information can be packed into a single byte. Eventually bit fields are used for conserving the memory. The amount of memory saved by using bit fields will be substantial which is proved from the above example.

However, there are restrictions on bit fields when arrays are used. Arrays of bit fields are not permitted. Also the pointer cannot be used for addressing the bit field directly, although the use of the member access operator (->) is acceptable. The unnamed bit fields could be used for padding as well as for alignment purposes.

(1) Bits fields should have integral type. A pointer and array type is now allowed.
(2) Address of bit fields cannot be obtained using & operator.

The `class` for the above problem would be as follows:

```
class vehicle
{
   unsigned type: 3;
   unsigned fuel: 2;
   unsigned model: 3;
};
```

The colon (**:**) in the above declaration tells to the compiler that bit fields are used in the `class`

and the number after it indicates how many bits are required to allot for the field. A simple program is illustrated as follows:

8.41 Write a program to use bit fields with classes and display the contents of the bit fields.

```
#include<conio.h>
#include<iostream.h>
#define PETROL 1
#define DISEL 2
#define TWO_WH 3
#define FOUR_WH 4
#define OLD 5
#define NEW 6
class vehicle
{
  private :
  unsigned type : 3;
  unsigned fuel : 2;
  unsigned model :3;
  public:
  vehicle()
  {
  type=FOUR_WH;
  fuel=PETROL;
  model=NEW;
  }
  void show()
  {
  if (model==NEW)
  cout<<"\n New Model ";
  else
  cout<<"\n Old Model ";
  cout<<"\n Type of Vehicle : "<< type;
  cout<<"\n Fuel : "<<fuel;
  }
};
int main()
{
  clrscr();
  vehicle v;
  cout<<" Size of Object : "<<sizeof(v)<<endl;

  v.show();
```

```
    return 0;
}
```

OUTPUT

Size of Object : 1
New Model
Type of Vehicle : 4
Fuel : 1

Explanation: In the above program, using `#define` macros are declared. The information about the vehicle is indicated with integers from `1` to `6`. The `class vehicle` is declared with `bit fields`. The number of `bits` required for each member is initialized. As per the program, `type of vehicle` requires `3 bits`, `fuel` requires `2 bits`, and `model` requires `3 bits`. An object `v` is declared. The constructor initializes bits fields with data. The output of the program displays integer value stored in the bit fields, which can be verified with macro definitions initialized at the beginning of the program.

8.33 NESTED `class`

When a `class` is defined in another `class`, it is known as nesting of classes. In nested `class` the scope of inner `class` is restricted by outer `class`. Simple programming example using public specifier is demonstrated for understanding.

8.42 Write a program to display some message using nested `class`.

```
#include<iostream.h>
#include<conio.h>
class one
{
  public:
  class two
  {
  public:
  void display()
  {
  cout<< "Wonderful language C++\n";
  }
  };
};
int main()
{
  clrscr();
  one::two x;
  x.display();
```

```
    return 0;
}
```

OUTPUT

Wonderful language C++

Explanation: In the above example, one `class` is nested in another; that is `class` two is nested in `class` one. By using scope resolution operator, the inner `class` member function is accessed and "`Wonderful language C++`" is displayed in the above program.

8.34 MORE PROGRAMS

8.43 Write a program to accept string and display the string. Use null character for determining the end of string.

```
#include<iostream.h>
#include<conio.h>
class text
{
  char str[50];
  public :
  void get()
  {
  cout<<"Enter text : ";
  cin.getline(str,50);
  }
  show (int x)
  {
  if (str[x]=='\0')
  return 0;
  else
  {
  cout<<str[x];
  return 1;
  }
  }
};
int main()
{
  clrscr();
  int k=-1;
  text s;
  s.get();
```

```
    cout<<"Entered text : ";
    while (s.show (++k));
    return 0;
}
```

OUTPUT

Enter text : Object Oriented Programming
Entered text : Object Oriented Programming

Explanation: In the above program, the `class text` is defined with one data member of character type. The `get()` function is used to read text through the keyboard. The `show()` function is defined with integer argument. The `show()` function displays the string character by character. The integer variable `x` shows the element number in the character array `str[50]`. The `if` statement checks whether the current character is null or other. If the character is null then it returns `0`, otherwise it displays the character and returns `1`. In function `main()`, `s` is an object of `class text`. The object `s` calls `gets()` and reads text.

The `show()` function is called within the bracket of `while` loop. The integer variable `k` is incremented before passing it to the function `show()`. In the statement, `++k` increment first takes place and then incremented value is sent. The array element counting starts from zero. In order to display the string from first character, the initial value of `k` should be `0`. Thus, initializing `k` to `−1` and applying increment operator as prefix can do this.

We know that the `while` loop is executed till the given test condition evaluates to 1. Thus, till the function `show()` returns 1 the loop is executed and when it returns 0 the loop is terminated. The function `show()` returns `0` only when it reaches the end of text. Meanwhile, the entire text is displayed on the screen.

8.44 Write a program to enter two strings and concatenate them. Display the resulting string.


```
#include<iostream.h>
#include<string.h>
#include<conio.h>
class text
{
    char str1[15];
    char str2[15];
    char str3[30];
    public :

    void get()
    {
    cout<<"\n Enter First String : ";
    cin.getline(str1,15);
    cout<<"\n Enter Second String : ";
```

```
   cin.getline (str2,15);
   }

   len()
   {
   return (strlen(str1));}

   void show()
   {
   cout<<"\nFirst String : "<<str1;
   cout<<"\nSecond Strng : "<<str2;
   cout<<"\nThird String : "<<str3;
   }
   combine (int x,int y)
   {
   str3[x]=str1[x];
   if (x>=y)
   str3[x]=str2[x-y];
   if (str2[x-y]=='\0')
   return 0;
   else
   return 1;
   }
};
int main()
{
   clrscr();
   int k=-1,y;
   text s;
   s.get();
   y=s.len();
   while (s.combine (++k,y));
   s.show();
return 0;
}
```

OUTPUT

Enter First String : CPLUS
Enter Second String : PLUS
First String : CPLUS
Second String : PLUS
Third String : CPLUSPLUS

Explanation: In the above program, the `class text` has three character array data members, namely `str1[15]`, `str2[15]`, and `str3[30]`, and has also four member functions, namely `get()`, `len()`, `show()`, and `combine()`. The `get()` function is used to read strings through the keyboard; the `len()` function returns the length of the first string; the `show()` function displays the strings; and the `combine()` function combines the first and second strings and assigns to the third string. The integer variable `k` is initialized to `-1`. The integer variable `y` contains the length of the first string. The function `combine()` is called within the bracket of `while` loop and two integers variable `k` and `y` are passed.

In `combine()` function, `x` indicates character element position in character arrays `str3[]` and `str1[]`. The statement `str3[x]=str1[x]` assigns characters of `str1[]` array-to-array `str3[]`. This assignment continuous till `x >= y` (y is length of the first string). When `x` is greater than `y`, the statement `str3[x]=str2[x-y]` is executed; that is, assignment of second string is now carried out.

The `x-y` displays the element number of second string. The second `if` statement in the `combine()` function checks whether the null character has met or not. If yes it returns `0`, otherwise `1`. The values `0` or `1` returned by `combine()` are collected by the `while` loop. The `while` loop is executed till it gets `1` from function `combine()`. When the `combine()` function returns `0`, the `while` loop is terminated. The `show()` function after `while` loop displays all the strings.

TIP

The reader may be in confusion as to why so much effort is required to write simple programs in C++. We can easily solve these problems with very short codes in C. The objective is to master how to solve these problems using OO concepts and to make the reader think more on objects. In the previous examples, the reader might have noticed that even for a small task, it might be calculation or comparison, and for every task, we defined member functions. The `class` should provide every operation in the form of method or member function needed by objects and the object should not be dependent on `main()` or other non-member function for any requirement. This is the pure object-oriented programming.

8.45 Write a program to declare data member of a `class` as public. Initialize and display them without using function.

```
#include<iostream.h>
#include<conio.h>
class boy
{ public:
   int weight;
   float height;
};
int main()
{
   clrscr();
   boy raj, sonu;
```

```
   raj.weight=35;
   raj.height=4.5;

   sonu.weight=30;
   sonu.height=4.1;
   cout<<"\n Weight of Raj = "<<raj.weight;
   cout<<" Height of Raj = "<<raj.height;
   cout<<"\n Weight of Sonu = "<<sonu.weight;
   cout<<" Height of Sonu = "<<sonu.height;
   return 0;
}
```

OUTPUT

Weight of Raj = 35 Height of Raj = 4.5
Weight of Sonu = 30 Height of Sonu = 4.1

Explanation: In the above program, the `class boy` contains two public members, namely `weight` and `height`. The members are public and can be accessed directly without using member function. In function `raj` and `sonu` are two objects of the `class` boy. The data elements of weight and height of both objects are initialized and displayed.

8.46 Write a program to declare constant member function arguments.

```
#include<iostream.h>
#include<conio.h>
class data
{
  private :
  int d;
  public :
  void set() { d=10; }
  void show() { cout<<endl<<"d="<<d; }
  void sub (data const &a, data const &b)
  { d=a.d-b.d; }
};
int main()
{
  clrscr();
  data a,b,c;
  a.set();
  b.set();
  c.sub(a,b);
  c.show();
```

```
  return 0;
}
```

OUTPUT

D=0

Explanation: In the above program, the arguments of function `sub()` are declared as constant. Hence, an attempt to modify these arguments will generate an error. Thus, by declaring the member variable arguments `const`, we can prevent them from modification.

8.47 Write a program to show the difference between private and public data members of a `class`.

```
#include<iostream.h>
#include<conio.h>
class player
{
  private :
  char name[20];
  int age;
  public :
  float height;
  float weight;
};
int main()
{
  clrscr();
  class player a;
  // a.name ="Sanjay"; // not accessible
  a.height=5.5;
  a.weight=38;
  cout<<"Height : " <<a.height;
  cout<<"\nWeight : " <<a.weight;
  return 0;
}
```

OUTPUT

Height : 5.5
Weight : 38

Explanation: In the above program, a `class player` is defined with four member variables `char name[20]`, `int age`, `float height`, and `float weight`. The first two members are `private` and last two members are `public`. In the function `main()`, `a` is an object

of type `class player`. The public member variables of `class player` are `height` and `weight` initialized with `5.5` and `38`, respectively, through object `a`. It is not possible to access the `private` member of the `class` directly. Hence, the variables `name` and `age` cannot be accessed. Any attempt to access them through the object will display an error message "'player::name' is not accessible" or " 'player::age' is not accessible".

8.48 Write a program to initialize private and public member variables of the `class`. Display the contents of member variables.

```
#include<iostream.h>
#include<conio.h>
class num
{
   int x;
   float y;

   public :
   char z;
   void readdata(int j, float k)
   {
   x=j;
   y=k;
   }

   void display()
   {
   cout<<"x="<<x <<" y="<<y;
   }
};

int main()
{
   clrscr();
   class num j;
   j.z='C';
   j.readdata(10,10.5);
   j.display();
   cout<<" z= "<<j.z;
   return 0;
}
```

OUTPUT

x=10 y=10.5 z= C

Explanation: In the above example, the `class num` contains `private` as well as `public` member variables. The two private member variables x and y are assigned values using member

functions. The public member variable `z` is assigned directly in function `main()`. The variable **j** is an object, `readdata()` is a member function. The variables `j` and `k` are parameters to be passed. Consider the following statements.

```
a) j.y=10.5 // invalid statement
b) j.z='C' // valid statement
```

The statement `(a)` is invalid because member variable `y` is private whereas statement `(b)` is a valid because member variable `z` is public and accessible by the object of the same `class`.

8.34.1 Member Function Inside the `class`

8.49 Write a program to declare `class` with member variables and functions. Read and display the data using the member functions.

```
#include<iostream.h>
#include<conio.h>
class player
{
  private :
  char name[20];
  int age;
  float height;
  float weight;
  public:
  void setdata()
  {
  cout<<"Enter Name Age Height Weight \n";
  cin>>> name >>age >> height >> weight;
  }
  void show()
  {
  cout<<"\nName :" <<name;
  cout<<"\nAge :" <<age;
  cout<<"\nHeight :" <<height;
  cout<<"\nWeight :" <<weight;
  }
};
int main()
{
  clrscr();
  class player a;
  a.setdata();
  a.show();
```

```
    return 0;
}
```

OUTPUT

```
Enter Name Age Height Weight
Sanjay 24 5.5 54
Name :Sanjay
Age :24
Height :5.5
Weight :54
```

Explanation: In the above program, the `class player` contains four `private` member variables and two `public` member functions `setdata()` and `show()`.The definition of both the functions is inside the `class`. In the previous example, we noticed that the `object` of any `class` could not access the private member variables of the `class`. To access the private member variables of the `class`, member function of that `class` are used. In this program, `setdata()` and `show()` are the member functions of `class player`. The `setdata()` function reads data through the keyboard and `show()` functions display the data on the screen. These member functions cannot be called directly as ordinary functions. Consider the statement `a.setdata()`. Here, `a` is an `object` of `class player` followed `dot(.) operator` and `function name`. An attempt to call these member functions directly without using object of that `class` will generate an error in the program such as "`Function ‘ ‘ should ‘ ‘ have a prototype`".

In this program, function definition of the member function is inside the `class`. It is also possible to put prototype of the function inside the `class` and definition outside the `class`. While defining the function, it is necessary to specify `class name` followed by `scope access operator`. This declaration tells the compiler the relation between the `class` and the member function. For understanding this concept, the following program is illustrated as follows.

8.34.2 Member Function Outside the `class`

8.50 Write a program to declare `class` with member variables and functions. Read and display the data using the member functions. Declare function definition outside the `class`.

```
#include<iostream.h>
#include<conio.h>
class player
{
   private :
   char name[20];
   int age;
   float height;
   float weight;
   public:
```

```
   void setdata();
   void show();
};
void player :: setdata()
{
   cout<<"Enter Name Age Height Weight \n";
   cin>>> name >>age >> height >> weight;
}
void player :: show()
{
   cout<<"\nName :" <<name;
   cout<<"\nAge :" <<age;
   cout<<"\nHeight :" <<height;
   cout<<"\nWeight :" <<weight;
}
int main()
{
   clrscr();
   class player a;
   a.setdata();
   a.show();
   return 0;
}
```

OUTPUT

Enter Name Age Height Weight
Ajay 24 5.2 45
Name :Ajay
Age :24
Height :5.2
Weight :45

Explanation: The above program is same as previous one. The only difference is that the function definition of the member functions is outside the `class`. The prototypes of these functions are inside the `class`, which are enough for these functions to join membership with `class player`.

Consider the following statements.

```
void player:: setdata()
Void player:: show()
```

Both these statements tell the compiler that the functions `setdata()` and `show()` are member functions of `class player`.

So far we declared the entire member functions as `public`. It is also possible to declare member functions as `private`. A private function can be called only from public member functions. Even an

object of same `class` cannot access the `private` function. An example is illustrated below based on this idea.

8.34.3 Private Member Functions

8.51 Write a program to declare private member function and call this member function using another public member function.

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class num
{
  private :
  int x;
  int sqr(int);
  public:
  input()
  {
  int n;
  cout<<"Enter a Non-zero Number :";
  cin>>>n;
  return sqr(n);
  }
};
num :: sqr (int k)
{
  return k*k;
}
int main()
{
  clrscr();
  int sq;
  class num j;
  sq=j.input();
  cout<<"\nSquare of "<<sqrt(sq) <<" is " <<sq;
  // j.sqr(5); not accessible
  return 0;
}
```

OUTPUT

Enter a Non-zero Number : 25
Square of 25 is 625

Explanation: In the above program, the function `sqr()` is declared as `private` and the function `input()` as `public`. The `input()` function reads an integer through the keyboard and calls the `private` function `sqr()`. The function `sqr()` calculates the square of the number and returns result to the `input()` function. The `input()` function again returns this value to the variable `sq` declared in `main()`. The last `cout` statement displays the entered number and its square. The statement given in remark is invalid statement, because an object cannot access private member. The `input() public` function acts as an inter-mediator in between `private` function and an `object` of the `class`. Without `public` functions, it is not possible for the object to access `private` functions.

8.52 Write a program to enter hours. Convert it into seconds and minutes. Display results.

```
#include<iostream.h>
#include<conio.h>
class hour
{
   int hours;
   int minutes;
   int seconds;
   public :
   void input(void);
   void show(void);
   void convert();
};
void hour :: input()
{
   cout<<"\n Enter Hour: ";
   cin>>>hours;
}
void hour :: show()
{
   cout<<" Hour = " <<hours <<" Minute = "<<minutes <<" Second = " <<sec-
   onds;
}
void hour :: convert()
{ minutes=hours*60;
   seconds=minutes*60;}
main()
{ clrscr();
   hour X,Z;
   cout<<"\n Object X";
   X.input();
   X.convert();
   cout<<"\nX : ";
```

```
   X.show();
   return 0;
}
```

OUTPUT

```
Object X
Enter Hour : 4
X : Hour = 4 Minute = 240 Second = 14400
```

Explanation: In the above program, the `class hour` is declared with three integer members and three member functions. The `input()` function reads number of hours through the keyboard. The function `convert()` calculates minutes and seconds and assigns results to minutes and seconds variables. The function `show()` displays the contents of all the member variables. In the function `main()`, `X` is an object of `class hour`. The object X calls all the member functions one by one. The values of member variables after execution are displayed at the output.

8.53 Write a program to count the number of vowels present in the entered string.

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
struct vowels
{
   private :
   char str [20];
   public :
   void input()
   {
   cout<<"Enter text in small case : ";
   gets(str);
   }
   length()
   {
   int l=strlen(str);
   return l;
   }
   vowel (int x)
   {
   if (str[x]=='a' || str[x]=='e' || str[x]=='i' || str[x]=='o' ||
   str[x]=='u')
   {
   cout<<" "<<str[x];
   return 1;
```

```
   }
   else
   return 0;
   }
};
main()
{
   clrscr();
   int i,l,c=0;
   vowels b;
   b.input();
   l=b.length();
   for (i=0;i<l;i++)
   {
   c=c+b.vowel(i);
}
cout<<"\n"<<c<<" Vowels are present in the string";
return 0;
}
```

OUTPUT

```
Enter text in small case : Programming
o a i
3 Vowels are present in the string
```

Explanation: In the above program, instead of `class` keyword `struct` keyword is used. The `struct vowels` has only one member variable, which is a character array `str`. The function `input()` reads text through the keyboard and assigns it to the member variable `str[20]`. The function `length()` determines the length of the string. The function `vowel()` when called receives an integer from calling function. The `if` statement checks whether the (**x**) the character of the string is vowel or consonant. If it is one of the vowels, the function returns `1` otherwise `0`.

In `main()`, `b` is an object of `struct vowels`. Using `for` loop, the object `b` calls the function `vowel()` repetitively. The return value (`0` or `1`) is added to a variable `c` during repetitive calls. The value of `c` finally displays the number of vowels present in the string.

8.54 Write a program to enter text. Find a given character in the string and replace it with another given character.

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
#include<string.h>
class fandr
```

```
{
  char str [40];
  char f;
  char r;
  public :
  void accept()
  {
  cout<<" Enter text : ";
  cin.getline(str,40);
  cout<<" Find what (char) : ";
  f=getche();
  cout<<"\n Replace with (char) : ";
  r=getche();
  }
  void display (int d)
  {
  cout.put(str[d]);
  }
  len()
  { int l=strlen(str);
  return(l);
  }
  void find ( int i)
  {
  if (str[i]==f)
  replace(i);
  }
  void replace ( int k)
  { str[k]=r; }
};
main()
{
  clrscr();
  int i,l;
  fandr b;
  b.accept();
  l=b.len();
  cout<<"\n Replaced text : ";
  for (i=0;i<l;i++)
  {
  b.find(i);
  b.display(i);
```

```
    }
return 0;
}
```

OUTPUT

Enter text : Progra__er
Find what (char) : _
Replace with (char) : m
Replaced text : Programmer

Explanation: In the above program, the `class fandr` have three character data type member variables. The function `accept()` reads the text through the keyboard. Here, the function `cin.getline(str, 40)` is used to connect with the keyboard. The `first argument` is the `name of array` and `second` is `size of array`. Followed by this, the character to be traced and replaced is entered in variable `f` and `r`. The function `len()` is obviously used for calculating length of the string. The function `find()` receives an integer from calling function. The `if` statement checks whether the specified character by the integer is equal to a given character or not. If the character is found, the `replace()` function is called which replaces the traced character with value of `'r'` variable. The `for` loop in `main()` function repetitively calls the functions `find()` and `display()`. Remember, the function `replace()` is called by `find()` function whenever the `if` condition is true. The `cout.put()` is used to display the character on the screen. Thus, `find` and `replace` operations execute.

8.55 Write a program to find largest out of ten numbers.

```
#include<iostream.h>
#include<conio.h>
#include<process.h>
class num {
int number[10];
public :
input (int i)
{
  cout<<" Enter Number ("<<i+1<<") :" ;
  cin>>>number[i];
  return(number[i]);
}
void large ( int s, int m)
{
  if (number[s]==m)
  {
  cout<<"\n Largest number : "<<number[s];
  exit(1);
  }
```

```
}
};
main()
{
clrscr();
int i,sum=0,k;
num b;
for (i=0;i<10;i++)
sum=sum+ b.input(i);
for (k=sum;k>=0;k--)
{
  for (i=0;i<10;i++)
  b.large(i,k);
}
return 0;
}
```

OUTPUT

```
Enter Number (1) :125
Enter Number (2) :654
Enter Number (3) :246
Enter Number (4) :945
Enter Number (5) :258
Enter Number (6) :159
Enter Number (7) :845
Enter Number (8) :940
Enter Number (9) :944
Enter Number (10) :485
Largest number : 945
```

Explanation: In the above program, the `class num` has one `integer array` as its member variable with element size 10 (`number[10]`). The `input()` function reads an integer through the keyboard and returns it to `main()` function. The first `for` loop repetitively calls function `input()`. The return value of `input()` function is added to variable `sum`. The `second` and `third for` loops together are used to find the largest number in the array. The `second` and `third for` loop variables are used as arguments for the function `large()`. As per iteration, the successive element numbers and decremented value of sum assigned to variable `k`. The values of variables k and loop variable i are sent to function `large()`. In each `third for` loop iteration, the `second for` loop executes once and value of sum is decremented. The `if` condition in `large()` function checks both the arguments. If the arguments are the same, the largest number is displayed, and the `exit(1)` function terminates the program, otherwise the program execution continues.

8.56 Write a program to count the numbers between 1 and 100, which are not divisible by 2,3, and 5.

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
class div
{
  static int num;
  public :
  void check(int n)
  {
  if (n%2!=0 && n%3!=0 && n%5!=0)
  {
  cout<<n <<"\t";
  num++;
  }
  }
  void show()
{ cout<<endl <<"\n Total numbers : "<<num; }
  void loop();
};
void div :: loop()
{
  int x;
  cout<<endl<<"\n Numbers from 1 to 100 not divisible by 2,3 & 5\n\n";
  for ( x=0 ;x<=100;x++)
  check(x);
}
int div :: num=0;
int main()
{
  clrscr();
  int x;
  div d;
  d.loop();
  d.show();
  return 0;
}
```

OUTPUT

Numbers from 1 to 100 not divisible by 2,3 & 5
1 7 11 13 17 19 23 29 31 37

41 43 47 49 53 59 61 67 71 73
77 79 83 89 91 97
Total numbers : 26

Explanation: In the above program, the `class div` is declared with one `static` data member variable, that is `num`. The `class div` also contains member functions `check()`, `show()`, and `loop()`. The function `check()` checks the received value with `if` statement whether it is divisible by `2`, `3`, and `5` or not. If it is not divisible, the number is displayed and the value of `static` data member `num` is incremented. The `loop()` function contains `for` loop and calls the function `check()` with one argument. The prototype of function is given inside the `class` and definition outside the function, because the member function with any loop defined inside the `class` cannot be expanded inline and will result in a warning message. The more applicable practice is to write the definition of this kind of function outside the `class`. The function `show()` displays the total number of numbers that satisfies the condition. The object `d` declared in function `main()` invokes the member functions `loop()` and `show()`.

8.57 Write a program to generate the nth Fibonacci number by using recursion.

Fibonacci series:1 1 2 3 5 8 13

```
#include<iostream.h>
#include<conio.h>
class Fibonacci
{
  public:
  fib(int x)
  {
     int f;
  if(x==0)
  return 0;
  if(x==1)
  return 1;
  else
  f=fib(x-1)+fib(x-2);
  return f;
  }
};
int main()
{
  clrscr();
  Fibonacci a;
  int y;
  cout<<"\nEnter the number:";
```

```
    cin>>y;
    cout<<"\nSum of Fibonacci numbers upto "<<y<<" is "<<a.fib(y);
    getch();
    return 0;
}
Output:
Enter the number:7
Sum of Fibonacci numbers upto 7 is 13
```

SUMMARY

(1) A `class` in C++ is similar to structure in C. Using `class` or structure, a programmer can merge one or more dissimilar data types and a new custom data type can be created.

(2) In C++, classes and structures contain member variables and member functions in their declarations with `private` and `public` access blocks that restrict the unauthorized use. The defined classes and structures further can be used as custom data type in the program to declare objects.

(3) The `private` and `public` are new keywords in C++. The `private` keyword is used to protect specified data and functions from illegal use whereas the `public` keyword allows access permission.

(4) The member function can be defined as `private` or `public` inside the `class` or outside the `class`.

(5) To access `private` data members of a `class`, member functions are used.

(6) The difference between member function and normal function is that the normal function can be invoked freely, whereas the member function can be invoked only using the object of the same `class`.

(7) The `static` is a keyword used to preserve the value of a variable. When a variable is declared as `static`, it is initialized to zero. A `static` function or data element is only recognized inside the scope of the present compile.

(8) When a function is defined as `static`, it can access only `static` member variables and functions of the same `class`. The `static` member functions are called using its `class` name without using its objects.

(9) The member functions of a `class` can also be declared as constant using `const` keyword. The constant functions cannot modify any data in the `class`.

(10) An object of a `class` can be passed to the function as arguments like variables of other data type. When an object is passed-by-value, then this method is called as `pass-by-value`, whereas when reference of an object is passed to the function then this method is called as `pass-by-reference`.

(11) When classes are declared inside a function, then such classes are called as local classes. The local classes have access permission to global variables as well as `static` variables. The local classes should not have `static` data member and functions.

EXERCISES

(A) Answer the following questions

(1) Explain `class` and `struct` with their differences.
(2) Which operators are used to access members?
(3) Explain the uses of `private` and `public` keywords. How are they different from each other?
(4) Explain the features of a member function.
(5) What are `static` member variables and functions?
(6) How are `static` variables initialized? Explain with a statement.
(7) What are `friend` functions and `friend` classes?
(8) How are `static` functions and friend functions invoked?
(9) What do you mean by `constant` function?
(10) What are local classes?
(11) What is recursion?
(12) What are bit fields?
(13) List the keywords terminated with a colon with their use.
(14) Can member functions be private?
(15) What is the concept of data hiding? What are its advantages in applications?
(16) Is it possible to access private data members without using member function? If yes, explain the procedure with an example.
(17) What are `static` objects?
(18) What is the difference between object and variable?

(B) Answer the following by selecting the appropriate option

(1) The members of a `class` are by default
(a) private
(b) public
(c) protected
(d) none of the above

(2) `class` data members can be accessed by
(a) public member functions
(b) private member function
(c) both (a) and (b)
(d) none of the above

(3) The objects of `class` can directly access
(a) public members
(b) private members
(c) protected members
(d) all of the above

(4) The private data members can be accessed through functions declared under
(a) private section
(b) public section
(c) protected section
(d) all of the above

(5) The protected data members can be accessed by private and protected member functions using
(a) non-member functions
(b) public member functions
(c) directly by object
(d) scope resolution operator

(6) For data hiding, data are generally declared in
(a) private section
(b) public section
(c) both (a) and (b)
(d) none of the above

(7) `class` declaration provides
(a) data hiding
(b) abstraction
(c) encapsulation
(d) all of the above

(8) Member functions can be defined
(a) only inside the `class`
(b) only outside the `class`

(c) inside as well as outside `class`
(d) none of the above

(9) A `class` contains
(a) only data
(b) only functions
(c) data and functions
(d) neither data nor functions

(10) When a `class` is defined, memory allocation is done
(a) at the same time
(b) when objects are declared
(c) when functions are called
(d) when values are passed

(11) Private data are accessible to
(a) objects directly
(b) objects through public member functions
(c) main function
(d) none of the above

(12) Public data members and functions are accessible by
(a) objects directly
(b) private members
(c) external function
(d) both (a) and (c)

(13) Inline function is not possible when
(a) a function contains a loop
(b) goto statement exists
(c) both (a) and (b)
(d) none of the above

(14) Functions defined outside the `class` can be accessed using
(a) scope resolution operator
(b) logical operator
(c) reference
(d) arithmetic operator

(15) Inline function
(a) saves memory
(b) takes extra time
(c) executes slowly
(d) wastes memory

(16) An inline function
(a) duplicates code
(b) speeds up the execution
(c) both (a) and (b)
(d) neither (a) nor (b)

(17) When inline function is called
(a) control is passed to function
(b) function is executed directly
(c) function code is replaced at that point
(d) none of the above

(18) The public member function can access
(a) private data
(b) public data
(c) protected data
(d) all of the above

(19) Public data can be accessed by
(a) private member function
(b) public member function
(c) protected member function
(d) all of the above

(20) By default, all member functions defined inside the `class` are treated as
(a) inline function
(b) external functions
(c) main function
(d) none of the above

(21) An encapsulated object is often called as
(a) abstract data type
(b) abstract `class`
(c) both (a) and (b)
(d) neither (a) nor (b)

(22) When one member function is called inside another member function, it is called
(a) cascading functions
(b) referencing functions
(c) nesting member function
(d) none of the above

(23) `static` data member is stored in memory as
(a) only one copy
(b) number of copies
(c) two copies
(d) no copy

(24) When a variable is declared as `static`, it is
(a) initialized to one
(b) initialized to zero
(c) not initialized
(d) none of the above

(25) A `static` data member can be recognized
(a) inside scope of `class`
(b) outside the `class`

(c) in `main()` function
(d) in non-member function

(26) When a variable is declared as `static`, objects share
(a) common data
(b) different data
(c) no data
(d) none of the above

(27) A `static` variable is accessible only within
(a) `class`
(b) `main()` function
(c) non-member function
(d) none of the above

(28) The memory for `static` data member is allocated
(a) only once
(b) twice
(c) thrice
(d) number of times

(29) If count is a `static` data member and item is a `class`, then
`int item::count;`
what is the value of `count` initially?
(a) zero
(b) one
(c) no value
(d) cannot be determined.

(30) Initialization of `static` data member must be done
(a) inside the `class`
(b) right after `class` definition
(c) in `main()` function
(d) in member function

(31) Which `static` data member can be accessed by objects of the same `class`?
(a) private
(b) protected
(c) public
(d) none of the above

(32) A `static` member function can access
(a) only `static` member variable
(b) non-`static` member variable
(c) static as well as non-`static` variables
(d) only non-`static` member variable

(33) A `static` member function can be called by
(a) objects only
(b) `class` name and scope resolution operator
(c) object as well as `class` name and scope resolution operator
(d) none of the above

(34) A private `static` member function can be accessed by
(a) private `static` member function
(b) public `static` member function
(c) objects of `class`
(d) both (a) and (b)

(35) A non-member function that can access the private data of `class` is known as
(a) friend function
(b) `static` function
(c) member function
(d) library function

(36) The size of object is equal to
(a) total size of data variables
(b) total size of member functions
(c) both (a) and (b)
(d) none of the above

(37) The `class` without any data members or member functions is called as
(a) empty `class`
(b) non-empty `class`
(c) both a and b
(d) none of the above

(C) Attempt the following programs

(1) Write a program to declare a `class` with three integer public data variables. Initialize and display them.

(2) Write a program to declare private data member variables and public member function. Read and display the values of data member variables.

(3) Write a program to declare private data member and a function. Also declare a public member function. Read and display the data using private function.

(4) Write a program to declare three classes `S1`, `S2`, and `S2`. The classes have a private data member variable of character data type. Read strings for the classes `S1` and `S2`. Concatenate the strings read and assign it to the data member variable of `class S3`.

(5) Write a program to enter positive and negative numbers. Enter at least 10 numbers. Count the positive and negative numbers using classes and objects.

(6) Write a program to declare `class` with private member variables. Declare member function as `static`. Read and display the values of member variables.

(7) Write a program to declare a `class temp_ture` as given below. Declare an array of five objects. Read and display the data using arrays.

```
class temp_ture
{
    private:
    char date[12],
    ct1[15], ct2[15],
    ct3[15], ct4[15];
    int temp [4]
}
```

(8) Write a program to declare a `class` with two integers. Read values using a member function. Pass the object to another member function. Display the difference between them.

(9) Write a program to define three classes. Define a `friend` function. Read and display the data for three classes using common functions. Use `friend` functions.

(10) Write a program to define a local `class`. Also define global variables with the same name as that of member variables of `class`. Read and display the data for global and member variables.

(11) Write a program to generate Fibonacci series using recursion with member function.

(12) Write a program to overload a member function and display the `string`, `int`, and `float` using overloading function.

(13) Write a program to calculate sum of digits of an entered number using indirect recursion. Recursion should be between `main()` and member function.

(14) Write a program to display a string in reverse using recursion.

(15) Write a program to add numbers from 1 to 10.

(16) Write a program to find area of a right angle triangle.

(17) Write a program to compute area and perimeter of a square.

(18) Write a program to compute average marks obtained by a student in five subjects.

(19) Write a program to display even numbers from 1 to 10.

(20) Write a program to display numbers divisible by 5 between 1 and 100.

(21) Write a program to display numbers divisible by 11 and 3 between 1 and 100.

(22) Write a program to find area of a cube.

(23) Write a program to find negative and positive numbers of an array.

(D) Find the bugs in the following programs

(1)

```
class data
{
   private ;
};
```

(2)

```
class data
{
   private:
   int x=20;
}
```

(3)

```
#include<iostream.h>
#include<conio.h>
class data
{ int x; };
void main()
{ data A;
A.x=30; }
```

(4)

```
#include<iostream.h>
#include<conio.h>
class data
{ int x;
   public:
   void show(void);
};
void show() { cout<<"\n
In show()"; }
void main()
{ data A;
A.show(); }
```

(5)

```
class
{
   public:
   void print()
   { cout<<a; }
};
void main()
{
}
```