**CLASSES AND OBJECTS**

**4.1 CLASS**

A class is a way to bind the data and its associated functions together it’s similer syntactically to a structure. Generally, a class specification has two parts:

1. Class declaration
2. Class function definitions

The ***class declaration*** describes the type and scope of its members. The ***class function definitions*** describe how the class functions are implemented.

The general form of a **class** **declaration** is:

 **class** *class\_name*

 {

 **private:**

*variable declarations;*

 *function declarations;*

 **public:**

 *variable declarations;*

 *function declarations ;*

 };

 However, the public members (both functions and data) can be accessed from outside the class. This is illustrated in fig. 3.1.

 Private area

### Data

Functions

 Public area

### Data

Functions

####  No entry to

 private area

 x

Enter allowed

to public area

Fig. 4.1 Data hiding in classes

**4.1.1 A Simple Class example1:**

The program contains a class and two objects of that class. Although it’s simple, the program demonstrates the syntax and general features of classes in C++. Here’s the listing for the SMALLOBJ program:

**Class Example 1: // smallobj**

#include <iostream.h>

**class smallobj //define a class**

**{ private:**

**int somedata; //class data**

**public:**

**void setdata(int d) //member function to set data**

**{ somedata = d; }**

**void showdata() //member function to display data**

**{ cout << “Data is “ << somedata << endl; }**

};

int main()

**{**

 smallobj s1, s2; //define two objects of class smallobj

s1.setdata (1066); //call member function to set data

s2.setdata (1776);

s1.showdata (); //call member function to display data

s2.showdata ();

return 0;

**}**

1. The class **smallobj** defined in this program contains one data item and two member functions.
2. The first member function sets the data item to a value, and the second displays the value.
3. Each of the two objects is given a value, and each displays its value.

Here’s the output of the program:

Data is 1066 ← object s1 displayed this

Data is 1776 ← object s2 displayed this

**4.1.2 Private and Public**

 The body of the class contains two keywords: private and public. What is their purpose?. The primary mechanism for hiding data is to put it in a class and make it private. Private data or functions can only be accessed from within the class The use of the keyword **private** is optional. By default, the members of a class are **private**. Public data or functions, on the other hand, are accessible from outside the class. This is shown in Figure 4.2.

****

**Fig(4.2) a two objects of class smallobj**

**4.1.3 Class Data**

The smallobj class contains one data item: somedata, which is of type int. The data items within a class are called data members. The data member **somedata** follows the keyword private, so it can be accessed from within the class, but not from outside.

**4.1.4 Member Functions**

 Member functions are functions that are included within a class. There are two member functions in **smallobj: setdata()** and **showdata().**Because setdata() and showdata() follow the keyword public, they can be accessed from outside the class. Figure 3 shows the syntax of a class definition.

****

**Fig (4.3) Syntax of a class definition.**

**4.1.5 Defining Objects (Creating Objects):**

The first statement in main() smallobj s1, s2; defines two objects, s1 and s2, of class **smallobj.** The definition of the class **smallobj** does not create any objects. Defining an object is similar to defining a variable of any data type: Space is set aside for it in memory. Defining objects in this way means creating them.

**4.1.6 Accessing Class members ( Calling members functions)**

 The private data of a class can be accessed only through the member functions of that class. The **main( )** cannot contain statements that access to **somedata** directly . The following is the format for calling a member function:

The next two statements in main() call the member function setdata():

**s1.setdata(1066);**

**s2.setdata(1776);**

 cobject-name **.** function-name (actual-arguments);

 The first call to setdata() s1.setdata(1066); executes the setdata() member function of the s1 object. This function sets the variable somedata in object s1 to the value 1066. The second call s2.setdata(1776); causes the variable somedata in s2 to be set to 1776. Now we have two objects whose somedata variables have different values, as shown in Figure 4.

****

**FIGURE 4.4 Two objects of class smallobj.**

Similarly, the following two calls to the showdata() function will cause the two objects to display their values:

**s1.showdata();**

**s 2.showdata();**

**4.1.7 Messages**

Some object-oriented languages refer to calls to member functions as messages. Thus the call s1.showdata(); can be thought of as sending a message to s1 telling it to show its data. Talking about messages emphasizes that objects are discrete entities and that we communicate with them by calling their member functions**.**

**4.1.8 Defining Member Functions:-**

Member functions can be defined in two places:

 **.** Outside the class definition

 **.** Insidethe class definition

**4.1.8.1 Outside The Class Definition:-**

They should have a function header and a function body. The general form of a member function definition is:

 *return-type* **class-name : :** **function-name** (argument declaration)

 {

######  Function body

 }

The *class-name*:: tells the compiler that the function *function-name* belongs to the class *class-name.* That is, the scope of the function is restricted to the *class-name* specified in the header line. The symbol :: is called the *scope resolution* operator.

 For instance, consider the member functions **getdata ( )** and **putdata ( )** member functions. They may be coded as follows:

 **Class item**

 **{**

 **int number;**

 **float cost;**

 **public:**

 **void getdata (int a, float b ); // declaration**

 **void putdata (void); // declaration**

 **};**

 void **item** :: **getdata** (int a , float b )

 {

 number = a;

 cost = b;

 }

 void **item** :: **putdata**(void)

 {

 cout << “number : “ << number << ”\n”;

 cout << ”cost : “ << cost << ”\n”;

 }

Since these functions do not return any value, their *return-type* is **void**.

* Member function can access the private data of the class. A non-

 member function cannot do so.

* A member function can call another member function directly, without using the dot operator.

**4.1.8.2 Inside the class Definition:-**

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class. For example, we could define the **item** class as follows:

 Class item

 {

 int number;

 float cost;

 public:

 ***void getdata (int a, float b ); // definition***

 ***{***

 ***number = a;***

 ***cost = b;***

 ***}***

 ***void putdata (void) // definition***

 ***{***

 ***cout << number << “\n”;***

 ***cout << cost << ”\n”;***

 ***}***

 };

When a function is defined inside a class, it is treated as an **inline** function.

**A C++ program with class:-**

/////////////////////////[Class implementation ]///////////////////////////////

 # include <iostream.h>

 class item // class declaration

 { private:

 int number; // private by default

 float cost ; // private by default

 public:

 void getdata (int a , float b ) ; // prototype declaration

 void putdata (void ) // function defined here

 {

 cout <<”number :’’ << number << “\n” ;

 cout <<”cost :” << cost << “\n” ;

 }

 };

//…………………..Member Function Definition ………………………

void item :: getdata ( int a , float b ) // use membership label

{

 number = a ; // private variables

 cost = b; // directly used

}

// ……………………….Main Program …………………………….

main( )

{

 item x; // create object x

 cout << “\n object x “ << “ \n” ;

 x. getdata ( 100,299.95 ); // call member function

 x.putdata ( ); // call member function

 item y; // create object y

 cout << “\no bject y “ << “\n” ;

 y.getdata (200, 175.50);

 y. putdata ( );

}

Here is the output of above program:

 **object x**

 **number : 100**

 **cost : 299.950012**

 **object y**

 **number : 200**

 **cost :175.5**

ex2:W.P to creat class to compute value of Y from equation Y=$\frac{X^{2}}{2X}$ if X>0

#include <iostream.h>

class equation

{

float X,Y,Z;

public:

void getdata (float a, float b);

void compute\_disp();

};

void equation :: getdata(float a ,float b)

{ X=a;

 Z=b;}

void equation ::compute\_disp( )

{

 if (X>0)

{ Y=(X\*X -Z)/2\*X; cout <<”\n Y= “<<Y;}

else cout<< “\n value of X less than zero”;

 }

 void main()

 {

 equation XE; float a,b;

 cin>> a>>b;

 XE.getdata (a,b);

 XE.compute\_disp();

 }

EX3: :W.P to creat class to compute area of rectangle ?

#include <iostream.h>

class rectangle

{

float length,width ,Ar;

public:

void readdata (float a, float b);

void AREA(void);

void display(void);

};

void rectangle :: readdata(float a, float b )

{ length=a; width=b;}

void rectangle ::AREA( )

{ Ar= length \* width;}

 void rectangle :: display( )

{ cout <<"\n area of rectangle = "<< Ar; }

 void main()

 {

 rectangle R; float L,W;

cout <<"enter length and width of rectangle :"<<endl;

cin>> L>>W;

 R.readdata (L,W);

 R.AREA ();

 R.display ();

 }

***Widget Parts as Objects***

// widget part as an object

#include <iostream.h>

class part //define class

{

private:

int modelnumber; //ID number of widget

int partnumber; //ID number of widget part

float cost; //cost of part

public:

void setpart(int mn, int pn, float c) //set data

{

modelnumber = mn;

partnumber = pn;

cost = c;

}

void showpart() //display data

{

cout << “Model “ << modelnumber;

cout << “, part “ << partnumber;

cout << “, costs $” << cost << endl;

}

};

main()

{ part part1; //define object of class part

part1.setpart(6244, 373, 217.55); //call member function

part1.showpart(); //call member function

}

**4.2 Nesting of member functions:-**

A member function can be called by using its name inside another member function of the same class. This is known as *nesting of member functions.*

//////////////////////////////[Nesting of Member Functions ]/////////////////////////////

 # include <iostream.h>

 class set

 {

 int m , n;

 public:

 void input (void);

 void display (void);

 int largest (void );

 };

int set :: largest (void )

{

 if ( m >=n )

 return (m);

 else

 return (n);

}

void set :: input (void )

{

 cout << “ input values of m and n “ << “\n” ;

 cin >> m >> n ;

}

 void set :: display (void )

{

 cout <<”largest value = “

 << largest ( ) << “\n”; // calling member function

}

 main ( )

{

set A ;

A. input ( );

A. display ( );

} ////////////////////////////////< end program >///////////////////////////////

The output of program would be:

 Input values of m and n

 30 17

 largest value = 30

EX:W.P to creat class to find summation of any two integer numbers?

#include <iostream.h>

class SUM1

{

int X,Z;

**public:**

void readdata (float a, float b)

{ X=a; Z=b; }

int sum(void)

{ return (X+Z) ; }

void putdata ()

{ cout <<" \n summation of "<<X <<"+"<<Z <<" = "<<sum()<<endl; }

};

 void main()

 {

 SUM1 S; int a,b;

 cout<<" \n enter two integer number :"<<endl;

 cin>> a>>b;

 S.readdata (a,b);

 S.putdata ();

S.readdata (30,15);

S.putdata();

}

**4.3 Private member functions:-**

Although it is normal practice to place all the data item in a private section and all the functions in public, some situations may require certain functions to be hidden (like private data) from the outside calls. Tasks such as deleting an account in a customer file, or providing increment to an employee are events of serious consequences and therefore the functions handling such tasks should have restricted access. We can place these functions in the private section.

 A private member function can only be called by another function that is a member of its class. Example.

 class sample

 {

 int m ;

 **void read (void);** //private member function

 public:

 void update (void);

 void write (void);

 };

If **s1** is an object of **sample,** then

 **s1.read( );**  // won’t work ; objects cannot access private

 // members

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

 void sample : : update ( void )

 {

 **read ( )**; // simple call; no object used

 }

**4.4 Arrays Within a Class: -**

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

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

 class array

 {

 **int a[size]**; // ‘a’ is int type array

 public:

 void setval(void);

 void display(void);

 };

Example:

#include <iostream.h>

#include <conio.h>

const m=5;

class ITEMS

{

 int itemcode [m] ;

 float itemprice [m] ;

 int count ;

 public:

 void CNT(void) { count=0 ;} // initializes count to 0

 void getitem(void) ;

 void displaysum(void) ;

 void remove(void) ;

 void displayitems(void) ;

};

// ================Member functions ============

void ITEMS :: getitem(void) //assign values to members

{

 cout <<"Enter item code :" ;

 cin >> itemcode [count] ;

 cout << "Enter item cost :" ;

 cin >> itemprice [count] ;

 count ++;

 }

void ITEMS :: displaysum(void) // display total value

{ float sum = 0;

 for(int i= 0 ; i< count ; i++)

 sum = sum + itemprice[i] ;

 cout << "\n Total value :"<< sum << "\n" ;

 }

void ITEMS :: remove(void) // Delete a specified item

{

 int a;

 cout << "Enter item code :" ;

 cin >> a;

 for(int i=0 ; i < count ; i++)

 if (itemcode [i]==a)

 { itemcode[i] =0;itemprice [i]=0;}

 }

void ITEMS :: displayitems(void) // displaying items

{

 cout << "\n code price\n" ;

 for (int i=0 ; i< count ; i++)

 {

 cout <<"\n"<< itemcode[i];

 cout <<" "<< itemprice[i];

 }

 cout <<"\n" ;

 }

// =====================Main program ===================

main( )

{

 ITEMS order;

 order. CNT();

 int x;

 do //do . . . while loop

 {

 cout <<"\n You can do this following ;"

 << "\n Enter appropriate number \n";

 cout << "\n1 : Add an item ";

 cout << "\n2 : Display total value ";

 cout << "\n3 : Deleting an item ";

 cout << "\n4 : Display all items ";

 cout << "\n5 : Quit ";

 cout <<"\n\nwhat is your option?";

 cin >> x ;

 switch (x)

 {

 case 1 : order. getitem( ) ; break ;

 case 2 : order. displaysum( ) ; break ;

 case 3 : order. remove( ) ; break ;

 case 4 : order. displayitems( ) ; break ;

 case 5 : break ;

 default : cout << "Error in input ; try again\n" ;

 }

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

getch();

 }

**4.5 Arrays Of Objects:-**

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

class employee

{

 char name[30];

 float age;

 public:

 void getdata(void);

 void putdata(void);

};

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

**employee manager[3];** // array of manager

**employee foreman[15];** // array of foreman

**employee worker[75];** // array of worker

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

 We can use the usual array-accessing methods to access individual elements and then the dot member operator to access the member functions. For example, the statement

 **Manager [I ]. Putdata ();**

Will display the data of the ith element of the array **manager.** That is, this statement

/////////////////////////////////Arrays Of Objects //////////////////////////////

#include <iostream.h>

class employee

{

 char name[30]; //string as class member

 float age;

 public:

 void getdata(void);

 void putdata(void);

};

void employee :: getdata(void)

{

 cout << “Enter name: “;

 cin >> name;

 cout << “Enter age: “;

 cin >> age;

}

void employee :: putdata(void)

{

 cout << “Name: “ << name << “\n”;

 cout << “Age: “ << age << “\n”;

}

const int size =3;

main()

{ employee manager[size]; //Array of managers

 for (int i = 0; i < size; i++)

 {

 cout << “\nDetails of manager” << i+1 << “\n”;

 manager[i] . getdata();

 }

 cout << “\n”;

 for (int t = 0; t < size; t++)

 {

 cout << “\nManager” << t+1 << “\n”;

 manager[t] . putdata();

 }

}

3.14 **Objects As Function Arguments:-**

 Like any other data type, an object may be used as function argument. This can be done in two ways:

* A copy of the entire object is passed to the function.
* Only the address of the object is transferred to the function.

The first method is called *pass-by-value*. Since a copy of the object is passed to the function, any changes made to the object inside the function do not affect the object used to call the function. The second method is call *pass-by-reference*. When an address of the object is passed, the called function works directly on the actual object used in the call.

 /////////////////////////////[Objects As Arguments ]////////////////////////

 # include <iostream.h>

class time

{ int hours, minutes ;

 public:

 void gettime (int h , int m )

 { hours = h; minutes = m; }

 void puttime (void)

 {

 cout << hours <<” hours and “;

 cout << minutes <<”minutes “ << ”\n”;

 }

 void sum(time, time); //objects are arguments

 };

 void time :: sum(time t1, time t2) // t1, t2 are objects

 {

 minutes = t1 . minutes+ t2 . minutes;

 hours = minutes/60;

 minutes = minutes %60;

 hours = hours + t1 . hours + t2 . hours;

 }

main()

{ time T1 , T2 , T3;

 T1. gettime (2,45); //get T1

 T2. gettime (3,30); //get T2

 T3 . sum(T1,T2); //T3=T1+T2

 cout << “T1= “; T1.puttime ( ); //display T1

 cout << “T2= “; T2.puttime ( ); //display T2

 cout << “T3= “; T3.puttime ( ); //display T3

}

The output of program would be:

 T1 = 2 hours and 45 minutes

 T2 = 3 hours and 30 minutes

 T3 = 6 hours and 15 minutes