Lesson 5: Structures And Classes

Objectives

After completing this lesson you will understand:

- About structures, initializing the structures and passing structures as arguments into functions.
- To define a class and know about the access specifier: public and private, defining a member function.
- To learn the difference between the structure and the class
- To study in detail about the constructors.

Structure Of The Lesson

5.1 Structures

Structures combine logically related data items into a single unit. The data enclosed within a structure are known as members and they can be of same type or different type. It is viewed as heterogeneous user-defined data type.

5.1.1 Defining Structure

A structure is a collection of variables of different data types grouped under a common name. The syntax of structure is as follows:

```
struct  tag name
{
data type1 member vari1
data type 2 member vari2
:
:
} ;
```

The keyword **struct** announces that it is a structure type definition. The tag name gives the name of the structure. The identifiers inside the braces are the member variables or of member names. The structure should end with a semi colon (;). Structure definition can be placed inside the main or before main function. After defining the structure definition, it can be used just like a predefined data type.

```
struct account
{
double balance;
double irate;
float term;
};
account acc1,acc2;
```

The structure variables can hold the values like any other variables. The structure value is a collection of smaller values called the member values. The smaller variables are called member variables.

The member variables can be accessed by using the structure variable followed by a dot(.), and then a member variable. Thus, the dot operator is used to specify a member variable of a structure variable.
Syntax: structure variable name. Member variable

In the above example the structure account has **three** member variables: balance, irate, term. acc1 and acc2 are structure variables of type account. Then the member variables of acc1 are:

acc1.balance
acc1.irate
acc1.term
Similarly acc2 has the member variables.
acc2.balance
acc2.irate
acc2.term

The first two variables are of type double and the third variable is of type float. The member variables can be assigned values using assignment statements. They can be used like any other variable.

```
e.g.:   acc1.balance = 1000.0;        //initialization using assignment
        acc1.irate = 2.0;
        acc1.term=2.0;
        acc1.balance = acc1.balance + interest;
        // arithmetic operation using structure variables.
```

The member variables can also be initialized during the declaration of the structure variable. To give a structure variable a value, it is followed by an equal sign and a list of member values enclosed within braces.

```
e.g.:   struct date
        {
        int month;
        int day;
        int year;
        };
```

Once the type date is defined, a structure variable birthday can be initialized as follows:

```
date birthday = { 12,3,1969};
```

During initialization, the order of the initializing values should match with the order of the member variables.

Then, birthday.month receives the value 12, birthday.day receives the value 3 and birthday.year is initialized with the value 1969. The number of member variables and the number of initializing values should be same.

Two or more structure types may use the same member names. The dot operator and the structure variable specify to which structure the member variable belongs.

eg:

```
struct fertilizerstock
{       double quantity;
        double nitrogencontent;
};
struct cropyield
{       int quantity;
        double size;  };
fertilizerstock  super;
cropyield apples,oranges;
```

fertilizerstock and **cropyield** are two structures types. Super and apples are variables of fertilizerstock and cropyield respectively. The quantity of super fertilizer is stored in **super.quantity** and the quantity of apples are stored in **apples.quantity.**

The structure value can be viewed as a collection of member values or a single variable. The structure value of one structure variable can be assigned to another structure variable using = sign. Thus a structure variable can be assigned into another structure variable of the same type by using an assignment operator.

If apples and oranges are two variables of type **cropyield**. Then

```
apples = oranges
```

is equivalent to

```
apples.quantity = oranges.quantity;
apples.size = oranges.size;
```

5.1.2 Initializing Structures

A structure can be initialized at the time of declaration. A structure variable can be given a value, by giving an equal sign and a list of member values enclosed in braces.

Eg:

```
struct date
{
        int month;
        int day;
        int year;
};
date duedate = { 12,31,1999};
```

The initializing values should be given in order with the member variables in the structure definition. If there are fewer initializing values than struct members, the provided values are initialized data members, in order. The remaining data members without initial values are initialized to a zero value of an appropriate type of the variable.

5.1.3 Structures As Function Arguments

Structures can be passed as arguments into functions. They can be passed as call by value or call by reference arguments.

Eg: A program to input your birth date and to display it.

```
#include<iostream.h>
struct date                 //structure definition
{
int day;
int month;
int year;
};
void getdate(date&);   //function prototypes
void putdate(date);
void main()
{
date bday;
getdate(bday);
putdate(bday);
return;
}
void getdate(date &bday)
{
cout<<"Day";
cin>> bday.day;
cout<<"Month";
cin>>bday.month;
```

```
cout<<"Year";
cin>>bday.year;
}
void putdate (date bday)
{
cout<<"My birth day is on";
cout<<bday.day<<" "<<bday.month<<" "<<bday.year;
}
```

output:

```
Day3
Month12
Year69
My birth day is on3 12 69
```

Program to input your birth date and to display it in another way.

```
#include<iostream.h>
#include<string.h>
struct date                    //structure definition
{
int day;
char month[15];
int year;
};

date getdate(int,char[],int);
void putdate(date);
void main()
{
date bday;
bday =getdate(15,"aug",1985);
putdate(bday);
}
date getdate(int x,char y[],int z)
{
date temp;
temp.day =x;
strcpy(temp.month,y);
temp.year=z;
return temp;
}
void putdate (date bday)
{
```

```
cout <<"my birth day is on";
cout<<bday.day<<" "<<bday.month<<" "<<bday.year;
}
```

output:

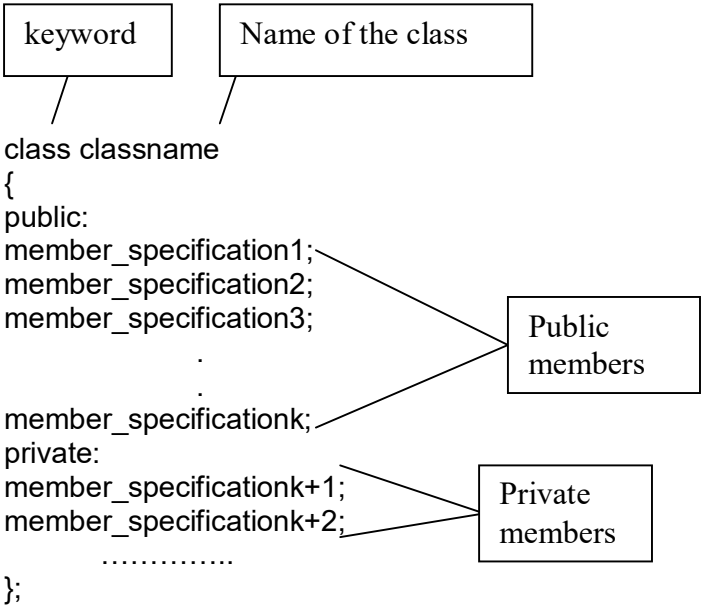
My birth day is on15 aug 1985

5.2 Classes

Classes are the basic language constructs of C++ for creating user defined data types. They are syntactically an extension of structures. Class follows the principle that the information about a module should be private to the module unless it is specifically declared public.

5.2.1 Defining A Class

A class is a data type whose variables are objects. Object is a variable that has data values as well as member functions. Classes are created using the keyword “class”. A class declaration defines a new data type that links the code and the data. This data type is used to declare the objects of that class. Hence class is a logical abstraction and an object is a physical abstraction. The syntax of the class is



The class definition contains the data members and can have prototypes of its member functions. The definition of the member function is given elsewhere, the class definition ends with a semicolon.

A sample class definition is given in the program below.

Program to demonstrate an example of classes

```
#include<iostream.h>
class dayofyear
{
public:
void output();//          <-------  member function prototype
int month;
int day;
};

int main()
{
dayofyear       today,birthday;
cout<<"Enter today's date\n"<<"Enter month as number:";
cin>>today.month;
cout<<"Enter day of the month:";
cin>>today.day;
cout<<"Enter your birthdate:\n"<<"Enter month as
number:";
cin>>birthday.month;
cout<<"Enter day of the month:";
cin>>birthday.day;
cout<<"Today's date is ";
today.output();
cout<<"Your birthday is ";
birthday.output();
if (today.month == birthday.month && today.day ==
birthday.day)
cout<<"Happy Birthday!\n";
else
cout<<"Good day!";
return 0;
}
void dayofyear::output()
{
        cout<<"month ="<<month<<",day = "<<day<<endl;
```

```
}
```

output:

```
Day3
Month12
Year69
my birth day is on3 12 69my birth day is on15 aug 1985Enter today's date
Enter month as number:12
Enter day of the month:3
Enter your birthdate:
Enter month as number:1
Enter day of the month:3
Today's date is month =12,day = 3
Your birthday is month =1,day = 3
Good day!
```

The type dayofyear defined is a class definition for objects whose values are dates. The member variables month and day stores the month and day of the year in integers. There is a member function called output, whose prototype is given in the definition. A class definition may contain the function prototype or the function definition. Two objects called today and birthday of data type day of year are declared. The member function output can be called with the object today or birthday as:

```
today.output();
birthday.output();
```

When a member function is defined the definition must include class name because there may be two or more classes that have member functions with the same class name. The member function is defined in the same way as any other function except the class name and the scope resolution operator are given in the function heading. The (.) dot operator and (::) scope resolution operators are used to tell which class the member belongs to. The scope resolution operator is used with class name, whereas the dot operator is used with the objects. The following function call will output the data values stored in the object today:

```
today.output();
```

The scope resolution operator specifies that the function output() belongs to the class dayofyear.

The class name that precedes the scope resolution operator is known as type qualifier because it specializes the function to one type of class. The member function is defined the same way as any other function except that the class name and scope resolution operator (::) are given in the function heading. The syntax of member functions is as follows.

Syntax:

```
Returntype   classname  :: function name(parameter list)
{
function body;
}
eg:
//uses iostream.h
void dayofyear::output()
{
        cout<<"month ="<<month<<",day = "<<endl;
}
```

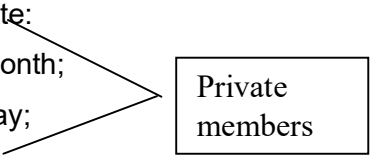
5.2.2 Public And Private Members

All the member variables and member functions, which are listed after keyword private in the class definition, are called private members of the class. When a variable is defined as private, it cannot be directly accessed in the program except within the definition of the member function. If it is accessed from the main program, an error message is given. When they are declared as public in the class definition, they can be accessed anywhere throughout the program. If private or public are not declared the compiler will take a default access specifier ie., private.

Program to demonstrate a simple example of a class

```
#include<iostream.h>
class dayofyear
{
public:
        void input();
        void output();//<------ member function prototype
        void set(int newmonth, int newday);
//Precondition:newmonth and newday take integer values
of //month and day of the date
//Postcondition:The data is reset according to the
arguments
int getmonth();
```

```
//Returns the month of its corresponding date object.
int getday();
//Returns the day of the month
private:
int month;
int day;
};
```




```
int main()
{
dayofyear       today,birthday;
cout<<"Enter today's date\n";
today.input();
cout<<"Todays date is :";
today.output();
birthday.set(3,21);
cout<<"Birthday:";
birthday.output();
if     (today.getmonth()     ==     birthday.getmonth()     &&
today.getday() == birthday.getday())
cout<<"Happy Birthday!\n";
else
cout<<"Good day!";
return 0;
}
//Memberfunction definitions
void dayofyear::input()
{
cout<<"Enter the month as a number:";
cin>>month;
cout<<"Enter the day of the month:";
cin>>day;
}
void dayofyear::output()
{
        cout<<"month ="<<month<<",day = "<<day<<endl;
}

void dayofyear::set(int newmonth, int newday)
{
month = newmonth;
day = newday;
}
```

```
int dayofyear::getmonth()
{
return month;
}

int dayofyear::getday()
{
return day;
}
```

output:

```
Enter today's date
Enter the month as a number:1
Enter the day of the month:2
Todays date is :month =1,day = 2
Birthday:month =3,day = 21
Good day!
```

To assign one object to another object we can use an assignment operator. The member variables of the source object, is copied into the target object.

```
birthday = today;
```

It works as follows:

```
birthday.month = today.month;
birthday.day = today.day;
```

5.2.4 Accessor Functions

Accessor functions are public functions of a class, which returns the private variables of the class. These functions provide some kind of access to the private variables and hence are called accessor functions.

e.g:

```
int dayofyear::getmonth()
{
return month;
}

int dayof year::getday()
{
return day;
}
```

As month and day cannot be directly accessed from the main program a function to return these values are written.
e.g: //This is illegal as month and day are private variables

```
if (today.month == birthday.month && today.day == birthday.day)
        cout<<"Happy Birthday!\n";
        else
        cout<<"Good day!";
        This can be modified to:
        //This is legal
if (today.getmonth() == birthday.getmonth() && today.getday() == birthday.getday())
        cout<<"Happy Birthday!\n";
        else
        cout<<"Good day!";
}
```

5.2.5 Difference Between Structures And Classes

Structures are normally used with all the member variables as public and with no member functions. A class contains member variables and member functions. The member variables and member functions can be declared as private or public. By default the members of structure are public and the members of a class are private. But at least one member function of the class should be defined as public function.

5.2.6 Properties Of A Class

- Classes have both member variables and member functions.
- A member of a class may be public or private.
- By default all the members of a class are labeled as private members.
- A private member of a class cannot be used elsewhere except within the definition of the member functions of the class. The name of the member functions for a class can be overloaded just like any other function.

- A class may use (any) another class type as its member variables.
- A function may have formal parameters of class type.
- A function may return an object of class type.

5.3 Constructors

Defining A Constructor : A constructor is a special type of member function defined in the class definition. It is used to initialize all or some of the member variables of the objects in the class.

A constructor is automatically called when an object of its associated class is declared.

A constructor must have the same name as the class name. A constructor definition cannot return any value. A constructor should be placed in the public section of the class definition.

The definition of a constructor can be given in the same way as a member function.

A constructor is declared and defined as follows:

```
// constructors defined outside the class
class integer
        {
        int m,n;
        public:
        integer();//default constructor declared
integer(int m,int n);//parameterized constructor ……..
        ……..
        };
integer::integer()        //constructor defined
        {
          m=0;n=0;
        }
integer::integer(int x, int y)
        {
         m = x;
          n = y;
        }
```

- The constructor can also be defined when it is declared in the class definition itself.

```
//class with constructor
class integer
        {
                int m,n;
        public:
        integer()
        {
         m=0;
          n=0;
        }
        integer(int x, int y)
        {
         m = x;
          n = y;
        }          ……..
        ……..
        };
```

5.3.1 Calling A Constructor

When a class contains a constructor like the one defined as above, the object created by the class is automatically initialized.

For example, the declaration, integer n1;//object n1 is created
not only creates the **object** int1 of type integer but also initializes its data members m and n to zero. There is no need to write any statement to invoke the constructor. The constructor is automatically called when the object is declared.

Constructor cannot be called like a member function.
eg: n1.integer(15,10);//illegal

When constructors are defined, an object cannot be declared with no arguments.The arguments must be added or a new constructor should be defined with no arguments.

5.3.2 Default Constructor

A constructor with no arguments is called default constructor. If no constructor is defined, the compiler will generate a default constructor.

If the user defines at least one constructor, then C++ will not generate any other constructor. If we do not want to initialize the default constructor with any values then an empty body can be defined.

```
integer()
{
}
```

Constructor can be called explicitly using an assignment operator.
e.g.: integer n1 = integer(10,5);//explicit call

Constructor can be called implicitly.
e.g.: integer int1(10,5);//implicit call

Constructor can be overloaded.In the above example, the constructor integer is overloaded by defining it once with no arguments and the other time by passing parameters. Thus it is overloaded.
e.g.:

```
class integer
{
int m,n;
public:
integer();//default constructor declared
integer(int m,int n);//parameterized constructor
        ……..
        ……..
};
```

Overloaded constructors

Write a program to add, sub, and divide rational numbers.

```
#include<iostream.h>
#include<stdlib.h>
class rno
{
int num,den;
public:
rno(int n, int d)
{
if (d == 0)
exit(1);
num=n;
den=d;
}
rno()
{
num=0 , den = 1;
}
```

```
rno(int n)
{
num = n;
den = 1;
}
void add(rno rn1,rno rn2)
{
num=rn2.den*rn1.num+rn1.den*rn2.num;
den=rn2.den*rn1.den;
}
void sub(rno rn1, rno rn2)
{
num=rn2.den*rn1.num-rn1.den*rn2.num;
den=rn2.den*rn1.den;
}
void mul(rno rn1,rno rn2)
{
num=rn2.num*rn1.num;
den=rn2.den*rn1.den;
}
void div(rno rn1, rno rn2)
{
num= rn1.num* rn2.den;
den=rn2.num*rn1.den;
}
void show()
{
cout<<num<<"/"<<den<<"\n";
}
};
void main()
{
rno r1(2,3);//constructor with 2 arguments is called
rno r2(3);//constructor with one argument is called
rno r3;//default constructor is invoked;
r3.add(r1,r2);
cout<<"Rational number 1:";
r1.show();
cout<<"Rational number 2:";
r2.show();
cout<<"Sum:";
r3.show();
cout<<"Subtraction:";
r3.sub(r1,r2);
r3.show();
```

```
cout<<"Multiplication:";
r3.mul(r1,r2);
r3.show();
cout<<"Division:";
r3.div(r1,r2);
r3.show();
}
```

output:

```
Rational number 1:2/3
Rational number 2:3/1
Sum:11/3
Subtraction:-7/3
Multiplication:6/3
Division:2/9
```

5.4 Destructors

A destructor is used to destroy the objects that have been created by the constructor. A destructor is a member function whose name is same as the class name but is preceded by a tilde. For example the destructor for the class **integer** can be defined as follows:

```
~integer(){ }
```

A destructor never takes any argument nor does it return any value. It will be invoked implicitly by the compiler upon exit from the program(or block or function as the case may be) to clean up the storage that is no longer accessible.

Note: Whenever memory is allocated dynamically using **new** operator in the constructors, delete operator should be used to free the memory. This is required because when the pointers go out of scope, a destructor is not called implicitly.

Program to demonstrate the destructors

```
#include<iostream.h>
int count=0;

class alpha
{
```

```
public:
alpha()//constructor
{
count++;
cout<<"\nnumber of object created"<<count;
}
~alpha()//destructor
{
cout<<"\nnumber of object destroyed"<<count;
count--;
}
};

int main()
{
        cout<<"\n\nenter main\n";
        alpha a1,a2,a3,a4;
        {
                cout<<"\n\nenter block1\n";
                alpha a5;
        }

        {
                cout<<"\n\nenter block2\n";
                alpha a6;
        }

        cout<<"\n\nre-enter main\n";

        return 0;
}
```

output:

```
enter main

number of object created1
number of object created2
number of object created3
number of object created4

enter block1

number of object created5
```

number of object destroyed5

enter block2

number of object created5
number of object destroyed5

re-enter main

number of object destroyed4
number of object destroyed3
number of object destroyed2
number of object destroyed1

5.5 Summary

- We have learnt how to define a structure and initialize the structure and to pass structure as parameter to a function.
- The definition of class, the public and private members of the class, how to assign an object to another object and accessor functions are covered.
- The properties of the classes and the difference between the structures and classes are learnt.
- The definition of constructor, the different types of constructors, and overloading of constructors are covered in detail.
- The destructors are also covered.

5.6 Technical Terms

Abstract data type: The data type in which the programmer do not have access to the details of how the values and operations are implemented.

Constructor: A special member function for automatically creating an instance of a class. This function has same name as the class.

Destructor: A function that is called to deallocate memory of objects of a class.

Data hiding: A property whereby the internal data structure of an object is hidden from rest of the program. The data can be accessed by the functions declared within the class.

Data member: A variable declared in the class.
Member function: A function declared within the class and not declared as friend. These functions can have access to data members and define operations that can be performed on the data.

Private member: A class member that is accessible only to the member and friend functions of the class.

Public member: A class member that is accessible to all the users of the class. The access is not restricted to member and friend functions. The public member of the base class can be easily inherited by the derived class.

5.7 Model Questions

1. What are the difference between classes and structures?
2. Explain the properties of classes.
3. What is a class? What is an object? Explain the definition of class, its access specifiers with an example?
4. What are accessor functions?
5. What are the different types of constructors? Explain with example.
6. What is overloading of constructor?
7. What is a default constructor?
8. What is a destructor?

5.8 References

Object-oriented programming with C++,

by E. Bala Gurusamy.

Problem solving with C++

by Walter Savitch

Mastering C++

by K.R.Venugopal, RajkumarBuyya, T.RaviShankar

AUTHOR:

M. NIRUPAMA BHAT, MCA., M.Phil.,
Lecturer,
Dept. Of Computer Science,
JKC College,
GUNTUR.