Chapter Table of Contents

Chapter 2.2

Classes and Objects

Aim

To provide the students with a basic knowledge of object oriented programming and their implementation using objects, classes and methods

Instructional Objectives

After completing this chapter, you should be able to:

- Explain Classes and objects
- Elaborate access of member functions outside the class
- Compare classes and structures
- Explain constructor and its types and illustrate the role of destructor
- Discuss the concept of constructor overloading and copy constructor

Learning Outcomes

At the end of this chapter, you are expected to:

- Elaborate on Access specifiers in C++
- Describe the mechanism of accessing data members and member functions
- Demonstrate array as class member data and array of objects
- Write programs to demonstrate classes and structures
- Compare constructor and destructor with a programming example

2.2.1 Introduction

The central feature of C++ is the addition of object oriented programming to C language, often called user-defined types and classes. The class specifies the form of an object, combines data representation and uses manipulating the data into a neat package.

The members within a class are defined by the data and its functions, which are unified as a self-contained unit called an object. A class can be regarded as a comprehensive idea similar to that of structures; this class describes the data properties alone.

2.2.2 Classes and Objects

In C++, the data and functions are bundled together as a self-contained unit called an object. A class is an extended concept similar to that of structures; this class describes the data properties alone. In C++, the class describes both the properties (data) and behaviour (functions) of objects. Classes are not objects, but they are used to instantiate objects.

(i) Defining the Class

Class is an abstract data type. Class definition doesn't occupy any memory. Once defined it lives till the program terminates. In object oriented programming we can place data and functions into a single entity. This is shown in *Figure 2.2.1.*

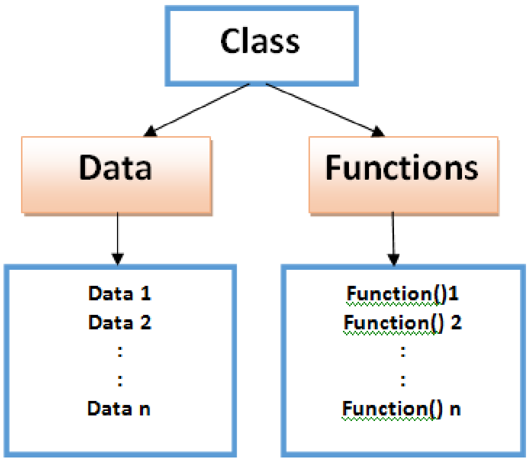


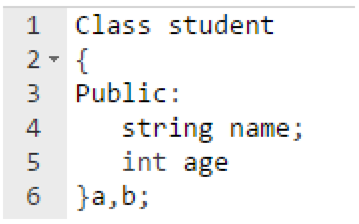
Figure 2.2.1: Classes Contain Data and Functions

Syntax:

```
Class ClassName
{
  Access specifier:
      Data members;
  Member functions () {}
};
```

Class definition should start with a keyword class followed by the class name. Class name can be any valid identifier. The body of the class should be delimited by braces and terminated by a semicolon.

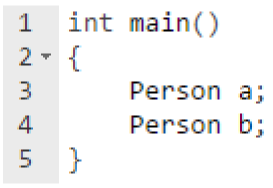
Example:



```
Class student
{
Public:
   string name;
   int age
}a,b;
```

In the above program, class definition starts with the keyword "class", followed by a class name "person". The class body is enclosed in curly braced and terminated with a semicolon. The class body contains data members and functions. In this example the data members are name and age specified with is data type. Access to these data members is bounded by access specifier. In this example the access specifier is public. Object declarations can be provided at the end of the class separated with commas. In this example a and b are the 2 objects of class person.

In this case the data type is the class name and variable is the object.



```
int main()
{
    Person a;
    Person b;
}
```

Both objects a and b will have their own copies of data members name and age.

(ii) Data Members and Member Functions

Data members are the variables declared inside the class defined with its appropriate data type. **Example:** int age.

A member function of a class is a function that has its definition or its prototype within the class definition. Example: void getdata()

Data members and methods or member functions must be declared with in the class definition.

For example,

```
Class P
{
   Int x; // x is a data member of a class p
   Int y; // y is a data member of a class p
   Int x; // Error re-declaration of x
}
```

Members can only be added in class definition.

In the below ***example,*** A and B are member function of class p. They determine the behaviour of the objects of class p.

For example,

```
Class p
{
   Int x;
   Int y;
   Void A(int, int);
   Void B();
}
```

(iii) Access Specifiers - Private, Public, Protected

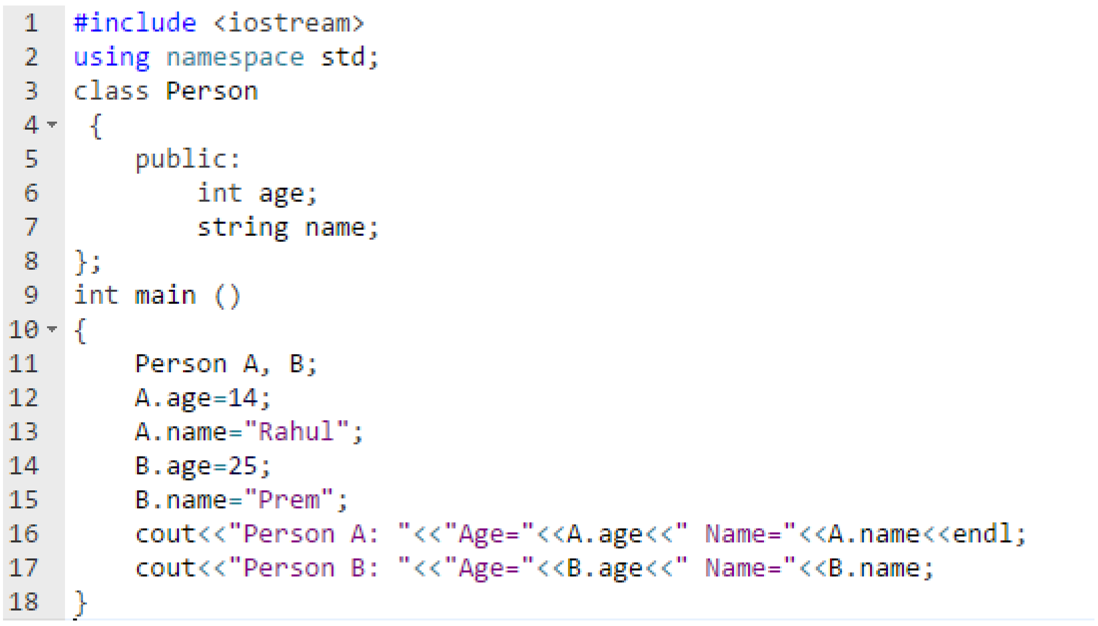
We can control access to class members and member functions by using access specifiers. Access specifiers are used to protect data from misuse. Access specifiers are of 3 types: public, private and protected.

Public class members and member functions can be used from outside of a class by any function or other classes. We can access public members and functions directly by using dot operator (.) or (arrow operator-> with pointers).

If, it is defined-private or protected, then we cannot access the data variables directly. Then we will have to create special public member functions to access, use or initialize the private and protected data members.

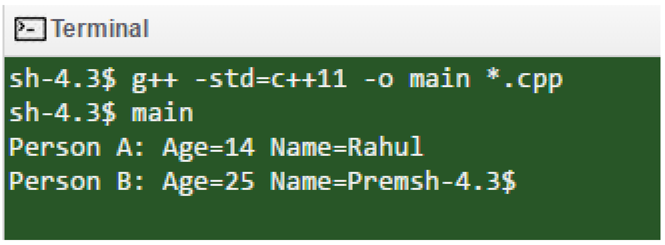
Accessing Public Data Members

Public data members can be accessed using the dot (.) operator and the object of that class. The following ***example*** shows you how to initialize and use the public data members.



```cpp
#include <iostream>
using namespace std;
class Person
 {
    public:
        int age;
        string name;
};
int main ()
{
    Person A, B;
    A.age=14;
    A.name="Rahul";
    B.age=25;
    B.name="Prem";
    cout<<"Person A: "<<"Age="<<A.age<<" Name="<<A.name<<endl;
    cout<<"Person B: "<<"Age="<<B.age<<" Name="<<B.name;
}
```

Output:

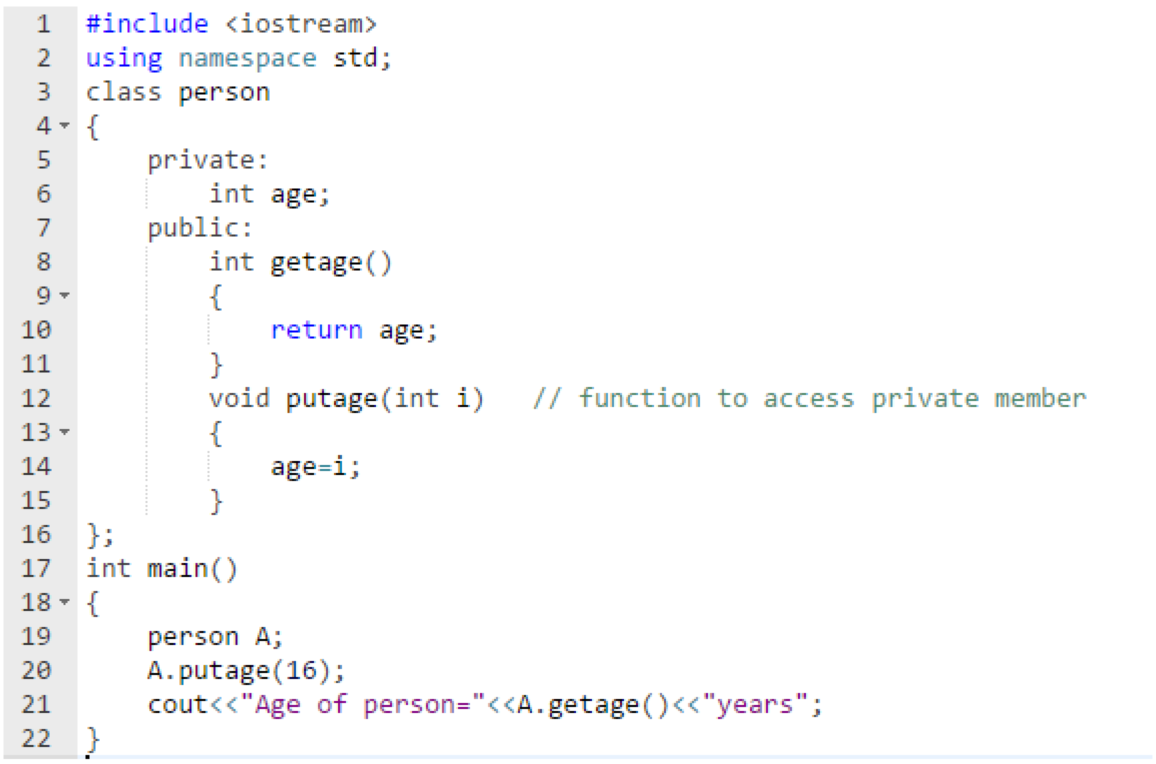


```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Person A: Age=14 Name=Rahul
Person B: Age=25 Name=Premsh-4.3$
```

Accessing Private Data Members

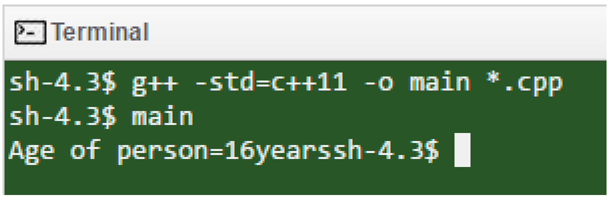
Public data members can be accessed using a dot operator and object name. But it is not same with the private members. Hence to access any private data members we have to define member functions. These member functions must be defined under public section of the class.

These functions are called as Getter and Setter functions. Getter method will return the value of any private data member whereas setter function sets the value for any private data member.



```cpp
#include <iostream>
using namespace std;
class person
{
    private:
        int age;
    public:
        int getage()
        {
            return age;
        }
        void putage(int i)   // function to access private member
        {
            age=i;
        }
};
int main()
{
    person A;
    A.putage(16);
    cout<<"Age of person="<<A.getage()<<"years";
}
```

Output:



Terminal

```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Age of person=16yearssh-4.3$
```

In the above program age is the private data member. So as to access this data member we have created 2 member functions in public section. getage() method will return the value of “age” and putage() method will set the value of “age” data member.

Accessing Protected Data members

Protected data members are accessed directly using dot (.) operator, for non-subclass we will have to follow the steps same as to access private data member.

Self-assessment Questions

1) An object is an __________ of a class.

2) We can have a class with all private members

   a) True b) False

3) A local class is defined inside a

   a) Class b) Function

   c) Program d) File

2.2.3 Objects as Function Arguments

Just like variables, either by value or by reference, member functions or non-member functions is passed as objects. A copy of function is created inside the actual objects when it is passed by value. The actual object is not reflected if the changes made to the copy are inside the function. Whereas, the object is passed into the function in pass by reference only as a reference to that object thus changing the reflected function of the actual object.

To access data members inside a function of the same class, an object of a class to a member function of the same class is passed using the dot operator and object name. However, the calling object in data members is accessed in the function without an object name or dot operator.

To understand how objects are passed and accessed within a member function, consider this example.

Example: A program to demonstrate passing objects by value to a member function of the same class

```
#include <iostream>
using namespace std;
class rational
{
    private:
    int num;
    int dnum;
    public:
    rational():num(1),dnum(1)
    {}
    void get ()
    {
         cout<<"enter numerator";
         cin>>num;
         cout<<"enter denomenator";
         cin>>dnum;
    }
    void print ()
    {
         cout<<num<<"/"<<dnum<<endl;
    }
    void multi(rational r1,rational r2)
    {
         num=r1.num*r2.num;
         dnum=r1.dnum*r2.dnum;
    }
};
void main ()
{
    rational r1,r2,r3;
    r1.get();
    r2.get();
    r3.multi(r1,r2);

    r3.print();

}
```

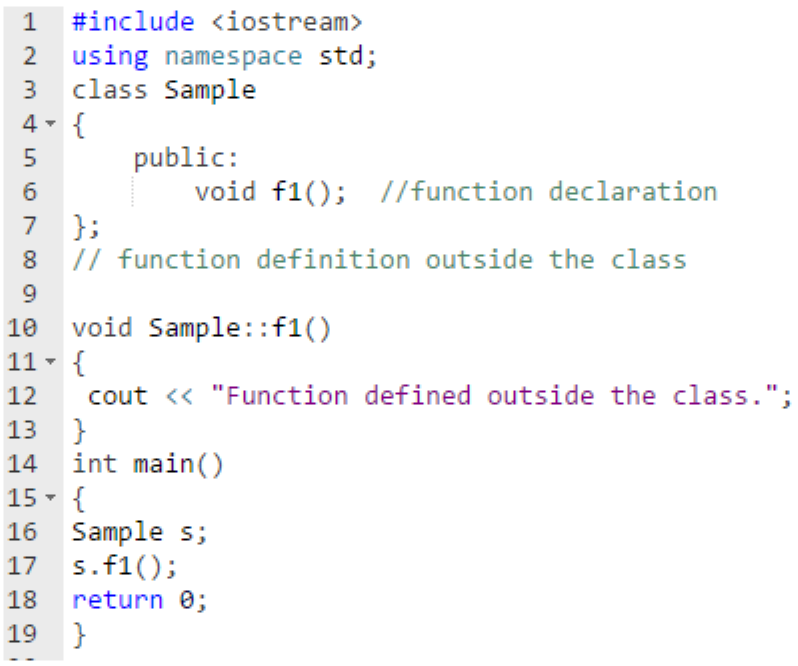
(i) Member Function Defined Outside the Class

The scope resolution operator (::) can be used to access data members outside the class passed to a member function of an object in a class.

Scope resolution operator

Scope resolution operator (::) is used to define a function outside a class.

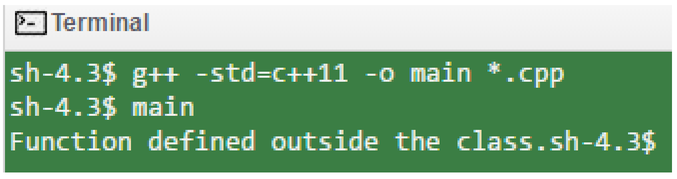
For example,



```cpp
#include <iostream>
using namespace std;
class Sample
{
    public:
        void f1();  //function declaration
};
// function definition outside the class

void Sample::f1()
{
 cout << "Function defined outside the class.";
}
int main()
{
Sample s;
s.f1();
return 0;
}
```

Output:



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Function defined outside the class.sh-4.3$
```

Self-assessment Questions

4) Member functions can be accessed with __________ and ________.

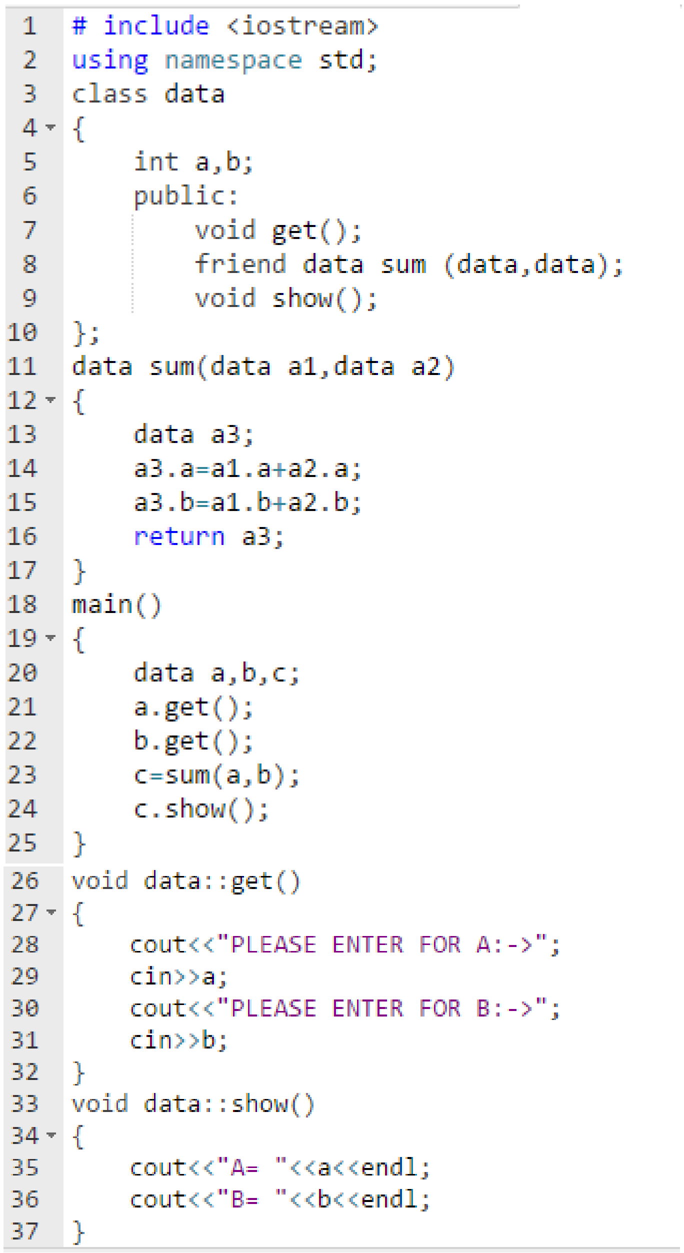
5) Members can be accessed outside the function using the
   - a) Dot operator
   - b) Scope resolution operator
   - c) Unary operator
   - d) Binary operator

6) When an object is passed by value, a copy of the ______is created inside the function.

2.2.4 Returning Objects from the Functions

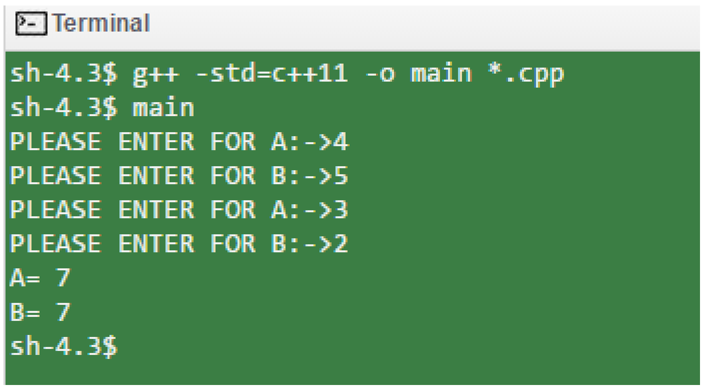
The language does not provide a direct support although it is more efficient to return a class object by pointer than to value it.

For example,



```cpp
# include <iostream>
using namespace std;
class data
{
    int a,b;
    public:
        void get();
        friend data sum (data,data);
        void show();
};
data sum(data a1,data a2)
{
    data a3;
    a3.a=a1.a+a2.a;
    a3.b=a1.b+a2.b;
    return a3;
}
main()
{
    data a,b,c;
    a.get();
    b.get();
    c=sum(a,b);
    c.show();
}
void data::get()
{
    cout<<"PLEASE ENTER FOR A:->";
    cin>>a;
    cout<<"PLEASE ENTER FOR B:->";
    cin>>b;
}
void data::show()
{
    cout<<"A= "<<a<<endl;
    cout<<"B= "<<b<<endl;
}
```

Output:



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
PLEASE ENTER FOR A:->4
PLEASE ENTER FOR B:->5
PLEASE ENTER FOR A:->3
PLEASE ENTER FOR B:->2
A= 7
B= 7
sh-4.3$
```

Self-assessment Questions

7) It is more efficient to return a class object by reference or pointer rather than a _______.

8) Returning objects can be done using __________ statement.

a) Return   b) Cout

c) Cin   d) Dot Operator

2.2.5 Difference between Classes and Structures

Classes and structures are syntactically similar. In C++, the role of the structure was expanded, making it an alternative way to specify a class. In C, the structures include data members; in C++ they can have function members as well. This makes structures in C++ and classes to be virtually same. The only difference between a C++ structure and a class is that, by default all the struct members are public while by default class members are private. The below table provides the differences between structures and classes.

Table 2.1.1: Differences between Structures and Classes

Classes	Structures
Class is a reference type and its object is created on the heap memory.	Structure is a value type and that is why its object is created on the stack memory.
Class can inherit another class.	Structure does not support the inheritance.
Class can have the all types of constructor and destructor.	Structure can only have the parameterized constructor. It means a structure cannot have the non-parameterized constructor, default constructor and destructor also.
The member variable of class can be initialized directly.	The member variable of structure cannot be initialized directly.
Class object cannot be created without using the new keyword, it means we have to use it. `   Demo obj=new Demo();`	Structure object can be created without using the new keyword.(optional) `   Demo obj;`

Here's the format we've been using for classes:
`class foo`
`{`
`   private:`
`   int data1;`
`   public:`
`   void func();`
`};`
	Here's the format we've been using for structures:
	
	`struct foo`
	`{`
	`   void func();`
	`   private:`
	`   int data1;`
	`};`

Self-assessment Questions

9) Class can inherit another class.

   a) True b) False

10) Structure can only have the _____________constructor.

   a) Parameterized b) Non-parameterized

   c) Default d) Dynamic

11) Structure object can also be created without using the __________ keyword.

12) Class object cannot be created without using the __________keyword.

   a) New b) Struct

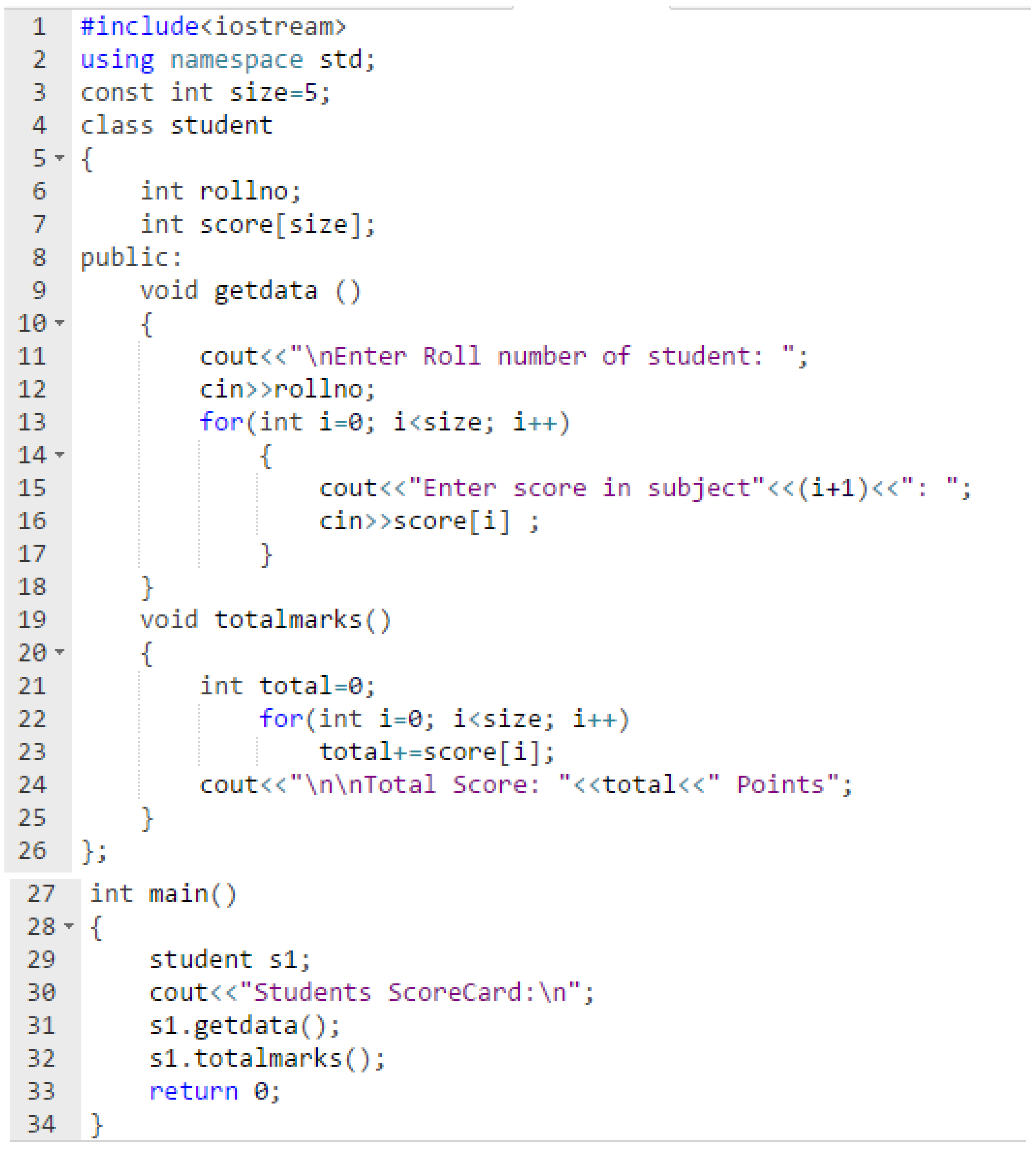
   c) Delete d) Class

2.2.6 Array as Class Member Data

Arrays can also be declared as the members of a class. They can be declared as private, public or protected members of the class. The following program prepares a student's score card using arrays.

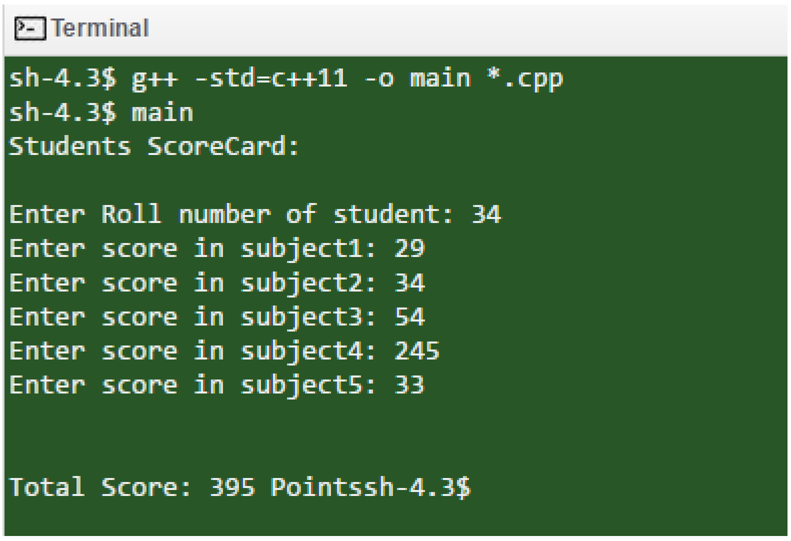
To understand the concept of arrays as members of a class, consider this example.

For example,



```
#include<iostream>
using namespace std;
const int size=5;
class student
{
    int rollno;
    int score[size];
public:
    void getdata ()
    {
        cout<<"\nEnter Roll number of student: ";
        cin>>rollno;
        for(int i=0; i<size; i++)
            {
                cout<<"Enter score in subject"<<(i+1)<<": ";
                cin>>score[i] ;
            }
    }
    void totalmarks()
    {
        int total=0;
            for(int i=0; i<size; i++)
                total+=score[i];
        cout<<"\n\nTotal Score: "<<total<<" Points";
    }
};
int main()
{
    student s1;
    cout<<"Students ScoreCard:\n";
    s1.getdata();
    s1.totalmarks();
    return 0;
}
```

The output of the program is



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Students ScoreCard:

Enter Roll number of student: 34
Enter score in subject1: 29
Enter score in subject2: 34
Enter score in subject3: 54
Enter score in subject4: 245
Enter score in subject5: 33


Total Score: 395 Pointssh-4.3$
```

In the above program, we store students score in an array and add them to get the total score. Hence, an array named "score" is created under private section. This array will store students score in respective 5 subjects. The member function getdata() is used to take the score in 5 subjects and store them in array.

Similar to other data members of a class, the memory space for an array is allocated when an object of the class is declared. In addition, different objects of the class have their own copy of the array. Note that the elements of the array occupy contiguous memory locations along with other data members of the object.

In our example when an object s1 of the class student is declared, the memory space is allocated for both roll no and score array.

Self-assessment Questions

13) Arrays can be declared as the members of a________.

a) Class b) Object

c) Function d) Structure

14) The arrays can be declared as________, ________ or _______ members of the class.

15) The memory space for an array is allocated when an object of the class is ________.

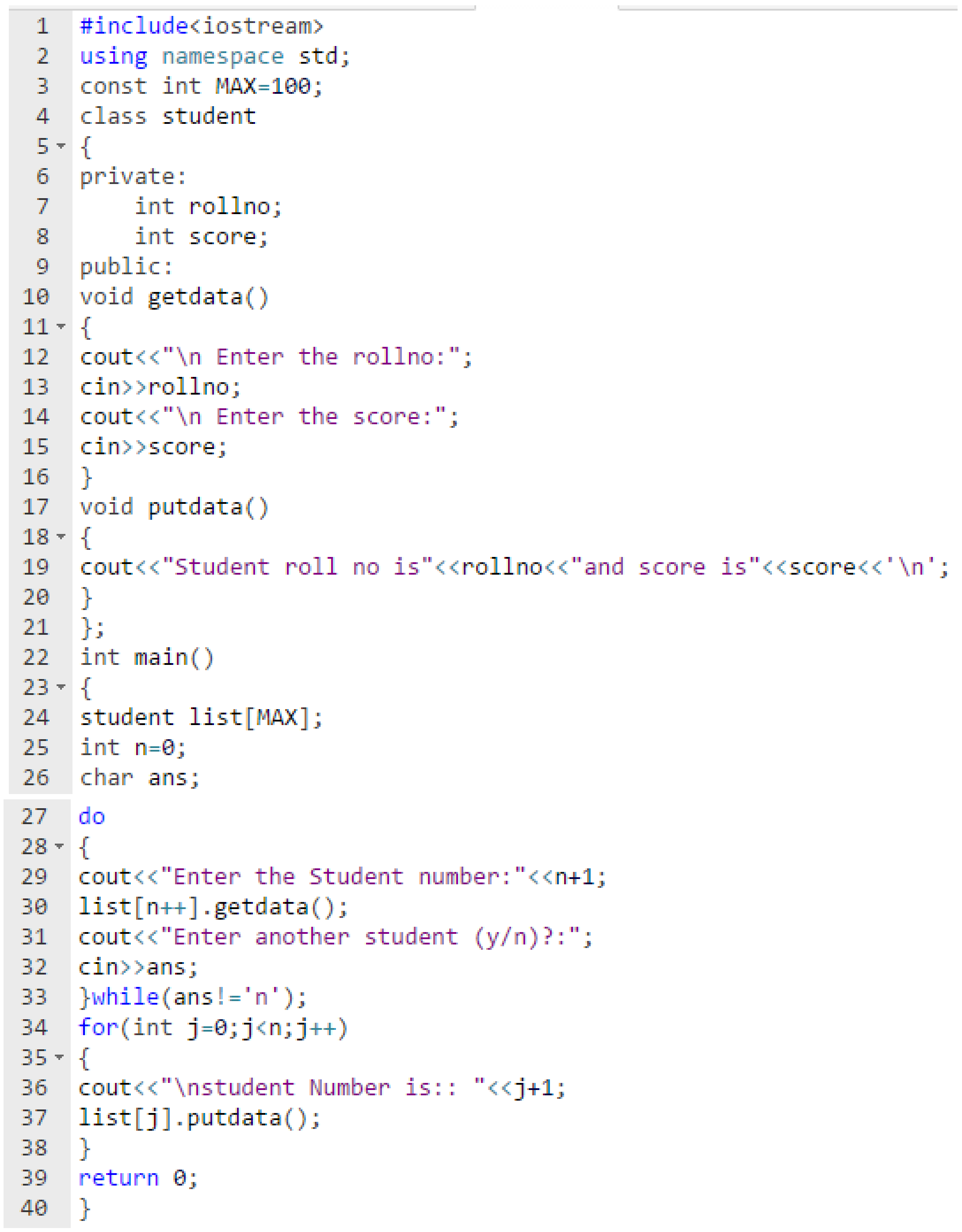
a) Declared b) Initialized

c) Defined d) Created

2.2.7 Array of Objects

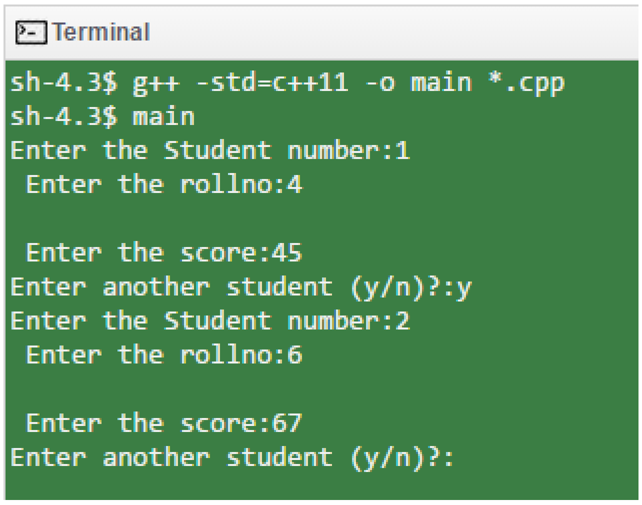
Like array of other user-defined data types, an array of type class can be also created. The array of type class contains the objects of the class as its elements. Thus, an array of type "class" is also known as an array of objects. Consider the following program to demonstrate the use of array of objects.

For example,



```
#include<iostream>
using namespace std;
const int MAX=100;
class student
{
private:
    int rollno;
    int score;
public:
void getdata()
{
cout<<"\n Enter the rollno:";
cin>>rollno;
cout<<"\n Enter the score:";
cin>>score;
}
void putdata()
{
cout<<"Student roll no is"<<rollno<<"and score is"<<score<<'\n';
}
};
int main()
{
student list[MAX];
int n=0;
char ans;
do
{
cout<<"Enter the Student number:"<<n+1;
list[n++].getdata();
cout<<"Enter another student (y/n)?:";
cin>>ans;
}while(ans!='n');
for(int j=0;j<n;j++)
{
cout<<"\nstudent Number is:: "<<j+1;
list[j].putdata();
}
return 0;
}
```

Result:



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Enter the Student number:1
 Enter the rollno:4

 Enter the score:45
Enter another student (y/n)?:y
Enter the Student number:2
 Enter the rollno:6

 Enter the score:67
Enter another student (y/n)?:
```

In the above example we store the data of n students. Hence an array of object "list" is created using the user defined data type "student". The student class stores roll no and marks of student. The function "getdata()" is used to get the input that is stored in this array of objects and putdata() is used to display the information of student.

Self-assessment Questions

16) Arrays of variables of type _________is known as "**Array of objects.**

a) Class	b) int
c) Structure	d) Object

17) The "identifier" used to refer the array of objects is a____________ data type.

a) User defined	b) Predefined
c) Structure	d) Class

18) The "identifier" used to refer the array of objects is a user defined data type.

a) True	b) False

2.2.8 Constructors and Destructors

Classes in C++ may have many data members. So giving them values is an important task. We should have a solution to initialize an object and also to destroy them. Hence, constructors and destructors are used.

Constructor is a member function in a class having same name as its class name and is used to initialize the objects of that class type with valid initial value. Constructor is automatically called when object is created.

A destructor is a member function in a class having same name as that of its class preceded by ~(tilde) sign and which is used to destroy the objects and release memory that have been created by a constructor. It gets invoked when an object's scope is over.

Constructors and destructors are declared within a class declaration (as like any other member function). We may declare some default arguments when we make a constructor. There are some rules specific to constructors and destructors:

- Constructors and destructors do have a return type (not even void).
- Constructors are executed when objects are created.
- Pointers and references cannot be used on constructors and destructors (It is not possible to get there address)
- Constructors can have default values and can be overloaded.
- Constructors and destructors cannot be declared static, const or volatile.
- Constructors cannot be declared with the keyword virtual.
- Destructors can be virtual but not constructors.
- Destructors neither have default values nor can be overloaded.

```
Class A
{
  int x;
  Public:
  A(); // Constructor
};
```

While defining a constructor you must remember that the name of constructor will be same as the name of the class and constructors never have return type.

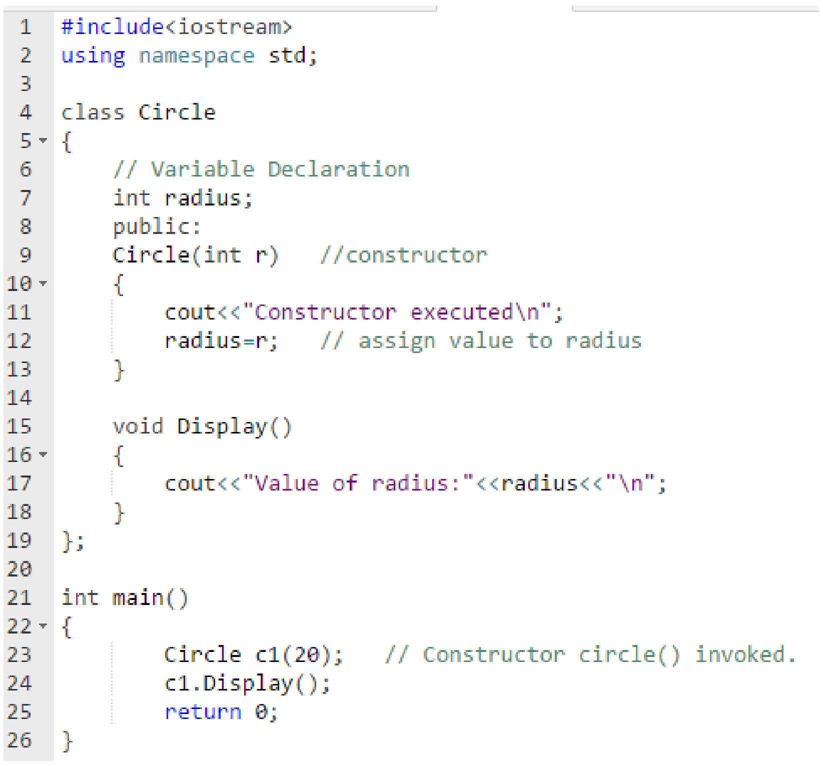
Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution: operator.

```
Class A
{
   Int I;
   Public:
   A();  // constructor declared
};
A::A()   //constructor definition
{
     I=1;
}
```

(i) Constructor with Argument

We can pass one or more parameters to the constructors similar to how parameters are passed to functions. The constructors with one or more parameters or arguments are called parameterized constructor. Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

Consider the following program:



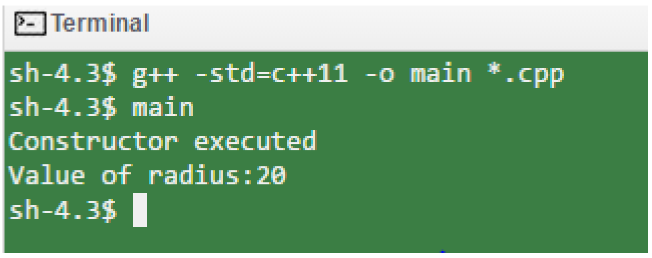
```
#include<iostream>
using namespace std;

class Circle
{
    // Variable Declaration
    int radius;
    public:
    Circle(int r)   //constructor
    {
        cout<<"Constructor executed\n";
        radius=r;   // assign value to radius
    }

    void Display()
    {
        cout<<"Value of radius:"<<radius<<"\n";
    }
};

int main()
{
        Circle c1(20);   // Constructor circle() invoked.
        c1.Display();
        return 0;
}
```

Output:



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Constructor executed
Value of radius:20
sh-4.3$
```

In the above ***example,*** we have created a class named "Circle" having only one data member radius. There is one parameterized constructor accepting one integer parameter. In the main function we have initialized one object named c1 of class Circle with user defined values using parameterized constructor. Thus, once the objects are created constructor will be called to initialize the data member. We can have any number of parameters in a constructor separated with commas.

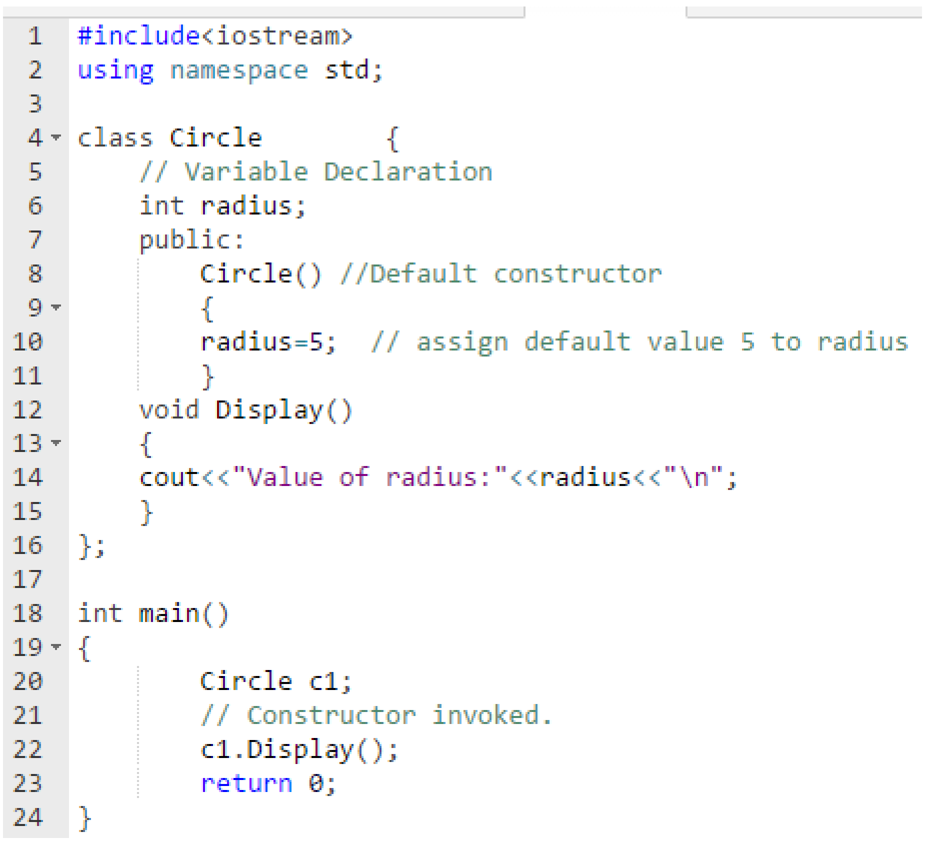
(ii) Constructor without Arguments

Sometimes constructors may not accept any parameters or arguments. The constructor which doesn't take any argument is called as a default constructor. It has no parameter.

Syntax:

```
class_name ()
{ Constructor Definition }
```

Consider the following program:

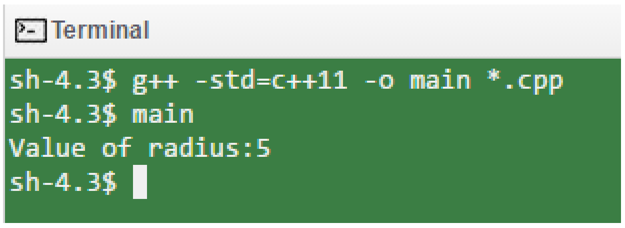


```
#include<iostream>
using namespace std;

class Circle        {
    // Variable Declaration
    int radius;
    public:
        Circle() //Default constructor
        {
        radius=5;  // assign default value 5 to radius
        }
    void Display()
    {
    cout<<"Value of radius:"<<radius<<"\n";
    }
};

int main()
{
        Circle c1;
        // Constructor invoked.
        c1.Display();
        return 0;
}
```

Output:



Terminal

```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Value of radius:5
sh-4.3$
```

In above program a default constructor is defined which is not accepting any parameters. This default constructor initializes the object with any default value. ***For example,*** radius=5. As soon as the object c1 is created the constructor is called which initializes its data members to value 5 for radius.

Advantage of a default constructor is that, even if user does not supply any parameter, the default constructor will always assign default value to the data member.

```
.class Circle
{
   int radius ;
};
```

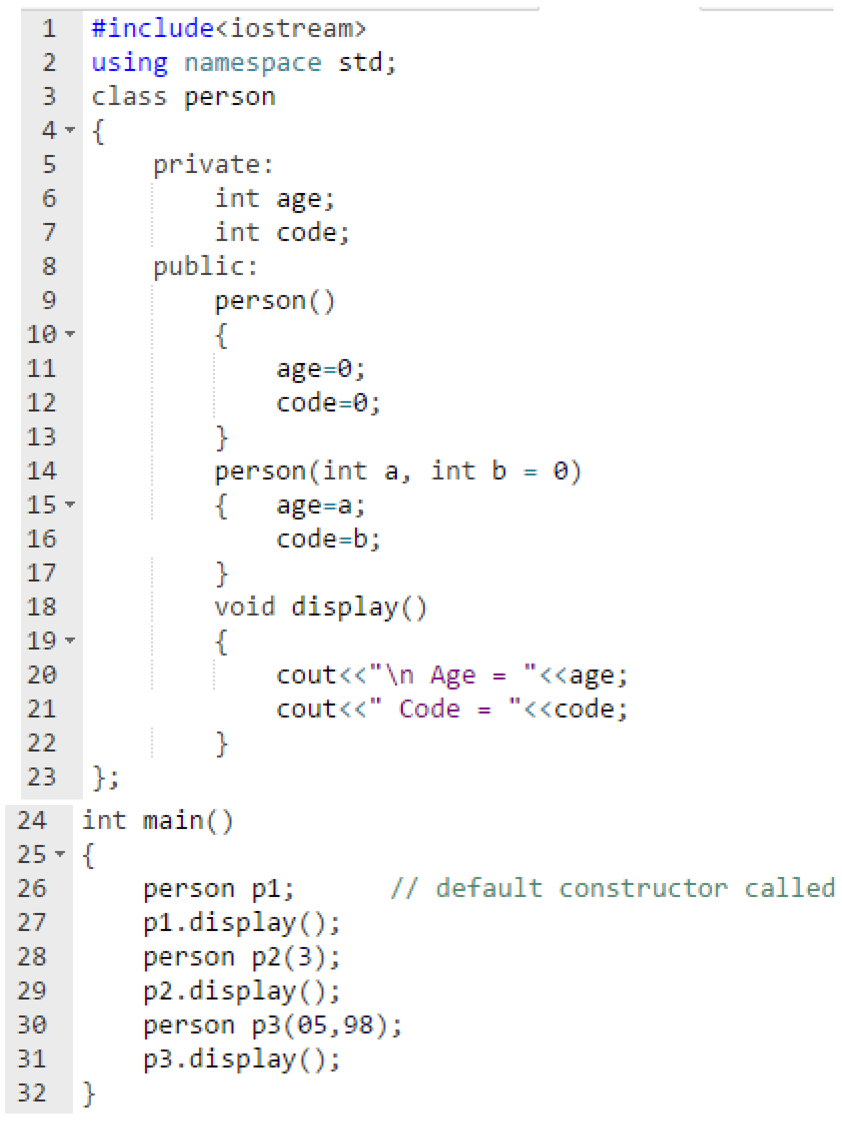
```
int main()
{
   Circle  c;
   cout<< c.radius ;
}
```

Output: 0

In this case, default constructor provided by the compiler will be called which will initialize the object data members to default value, which will be 0 in this case.

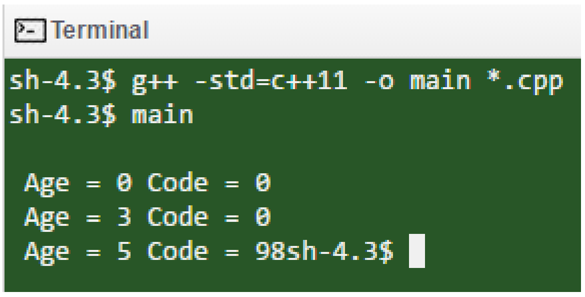
(iii) Constructor with Default Arguments

Similar to functions, it is also possible to declare constructors with default arguments. You can give one or more arguments to the constructor. While creating objects we need to pass those many arguments to the constructor. Consider the following ***example***



```
#include<iostream>
using namespace std;
class person
{
    private:
        int age;
        int code;
    public:
        person()
        {
            age=0;
            code=0;
        }
        person(int a, int b = 0)
        {   age=a;
            code=b;
        }
        void display()
        {
            cout<<"\n Age = "<<age;
            cout<<" Code = "<<code;
        }
};
int main()
{
    person p1;      // default constructor called
    p1.display();
    person p2(3);
    p2.display();
    person p3(05,98);
    p3.display();
}
```

Output:



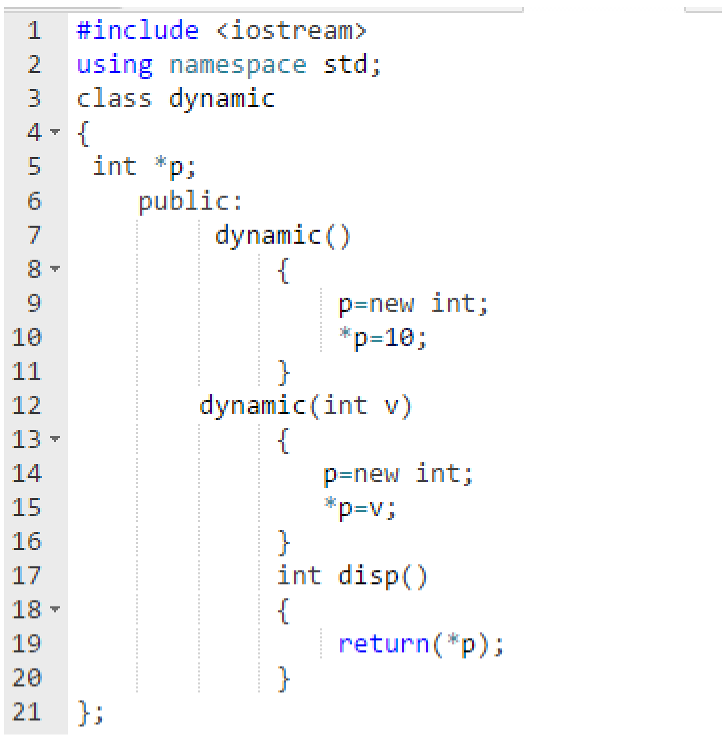
```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main

 Age = 0 Code = 0
 Age = 3 Code = 0
 Age = 5 Code = 98sh-4.3$
```

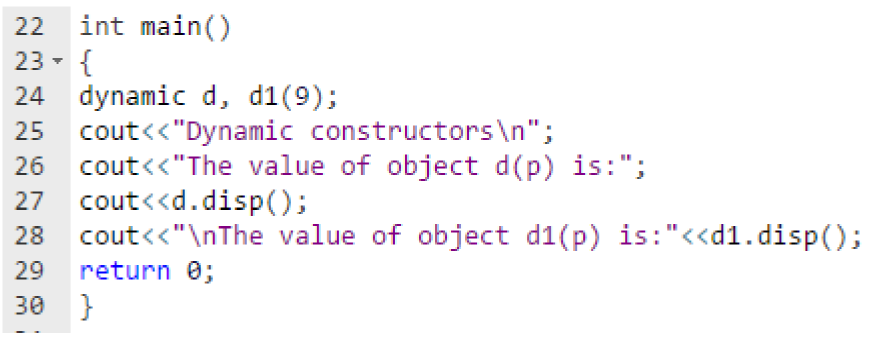
(iv) Dynamic Constructor

Dynamic constructors are the constructors which allocates memory to the objects at the runtime. In other words Dynamic constructors allocate memory for data members dynamically. Memory is dynamically allocated using new operator. It enables the program to allocate the right amount of memory to data members of the object during execution. The memory allocated to the data members are released when the object is no longer required and when the object goes out of scope. Thus we can dynamically initialize objects.

For example,

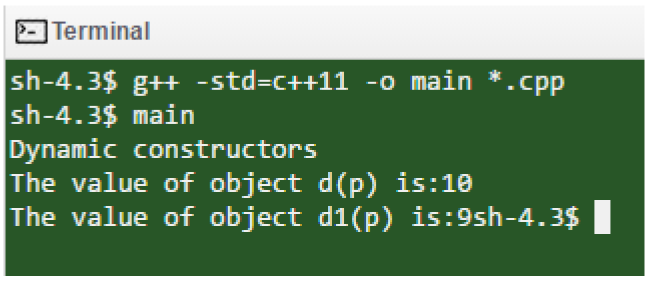


```
#include <iostream>
using namespace std;
class dynamic
{
 int *p;
     public:
          dynamic()
              {
                  p=new int;
                  *p=10;
              }
         dynamic(int v)
              {
                 p=new int;
                 *p=v;
              }
              int disp()
              {
                  return(*p);
              }
};
```



```
int main()
{
dynamic d, d1(9);
cout<<"Dynamic constructors\n";
cout<<"The value of object d(p) is:";
cout<<d.disp();
cout<<"\nThe value of object d1(p) is:"<<d1.disp();
return 0;
}
```

Output:



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Dynamic constructors
The value of object d(p) is:10
The value of object d1(p) is:9sh-4.3$
```

(v) Constructor Overloading

Similar to functions, constructors can also be overloaded. When a class has multiple constructors, they are called overloaded constructors. It is used to increase the flexibility of a class by having more number of constructors for a single class. More than one way of initializing objects can be done using overloaded constructors. Some important features of overloaded constructors are as follows:

- The name of all the constructors has the same name of the class.
- Overloaded constructors differ in their signature with respect to the number and sequence of argument passed.

For example,

```
#include <iostream.h>
class Overclass
{
   public:
   int x;
   int y;
   Overclass() { x = y = 0; }
   Overclass(int a) { x = y = a; }
   Overclass(int a, int b) { x = a; y = b; }
};
int main()
```

```
{
   Overclass A;
   Overclass A1(4);
   Overclass A2(8, 12);
   cout << "Overclass A's x,y value:: " <<
   A.x << " , "<< A.y << "\n";
   cout << "Overclass A1's x,y value:: "<<
   A1.x << " ,"<< A1.y << "\n";
   cout << "Overclass A2's x,y value:; "<<
   A2.x << " , "<< A2.y << "\n";
   return 0;
}
```

Result:

```
Overclass A's x,y value:: 0 , 0

Overclass A1's x,y value:: 4 ,4

Overclass A2's x,y value:; 8 , 12
```

In the above example the constructor "Overclass" is overloaded thrice with different initialized values.

(vi) Copy Constructor

Copy constructor takes an object of the class as an argument and copies data values of members of one object into values of members of another object. Since it takes only one argument, it is also known as a one argument constructor. The primary use of a copy constructor is to create a new object from an existing one by initialization. For this, the copy constructor takes a reference to an object of the same class as an argument.

For example,

```
#include<iostream.h>
Class Numbers
{
   Private:
   Int x;
   Public:
   Numbers(Numbers & N)  // copy constructor
   {
      X=N.x;
   }
   Numbers(int i)
   {
      X=1;
   }
   Void show_data()
   {
      Cout<<"\n x="<<x;
   }
};
main()
{
   Numbers N1(20);  // parameterised constructor is called
   Numbers N2(N1);  // copy constructor is called
   N2.show_data();
   Numbers N3=N1;   //    copy constructor is called
   N3.show_data();
}
```

Output:

```
X=20

X=20
```

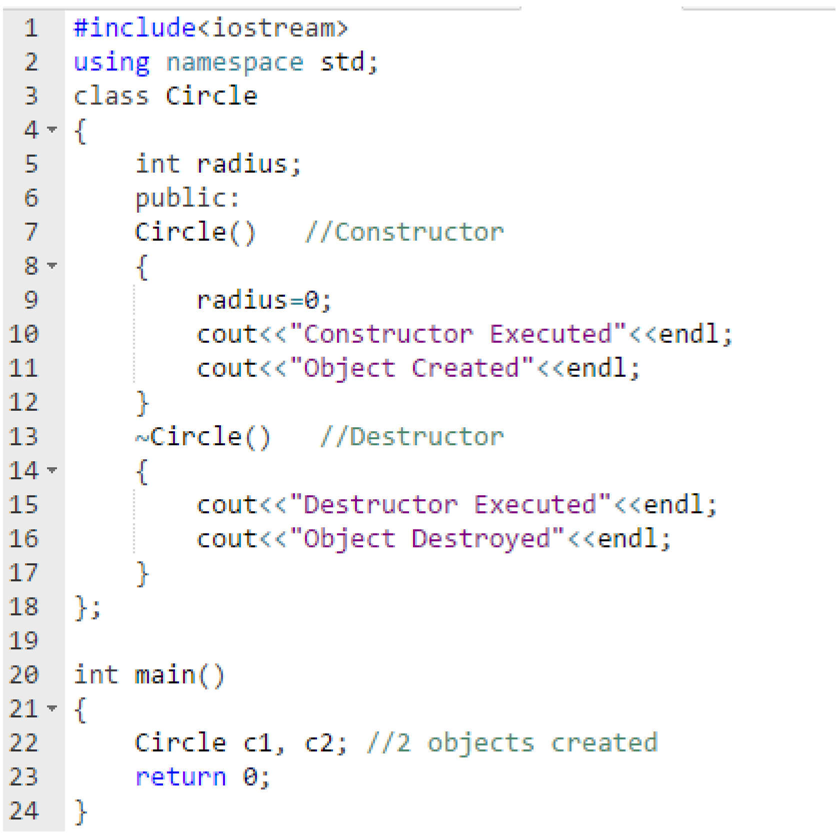
(vii) Destructors

A Destructor is also a member function having same name as that of class but preceded by a tilde sign and it helps to destroy objects. Unlike constructor, a destructor is called when a program has finished using an instance of an object. A destructor helps in releasing the resources hold by objects. Like the default constructor, the compilers always create a default destructor if you don't create one. Like the default constructor, a destructor also has the same name as its object.

Features

- Destructors are special member functions of the class required to free the memory of the object whenever it goes out of scope.
- Destructors are parameter less functions.
- Name of the Destructor should be exactly same as that of name of the class. But proceeded by '~' (tilde).
- Destructors do not have any return type. Not even void.
- The Destructor of class is automatically called when object goes out of scope.

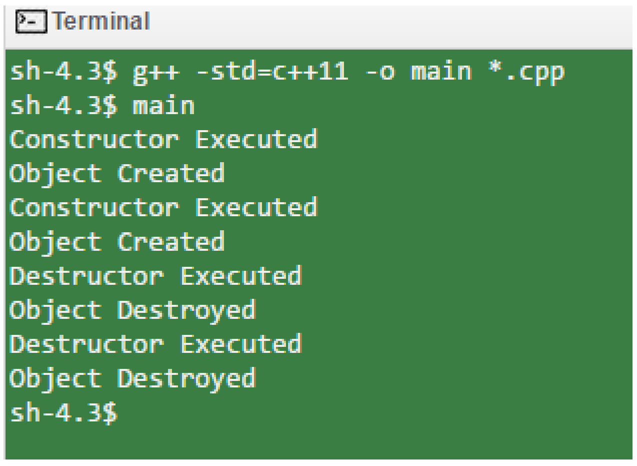
For example,



```cpp
#include<iostream>
using namespace std;
class Circle
{
    int radius;
    public:
    Circle()   //Constructor
    {
        radius=0;
        cout<<"Constructor Executed"<<endl;
        cout<<"Object Created"<<endl;
    }
    ~Circle()   //Destructor
    {
        cout<<"Destructor Executed"<<endl;
        cout<<"Object Destroyed"<<endl;
    }
};

int main()
{
    Circle c1, c2; //2 objects created
    return 0;
}
```

Output:



```
sh-4.3$ g++ -std=c++11 -o main *.cpp
sh-4.3$ main
Constructor Executed
Object Created
Constructor Executed
Object Created
Destructor Executed
Object Destroyed
Destructor Executed
Object Destroyed
sh-4.3$
```

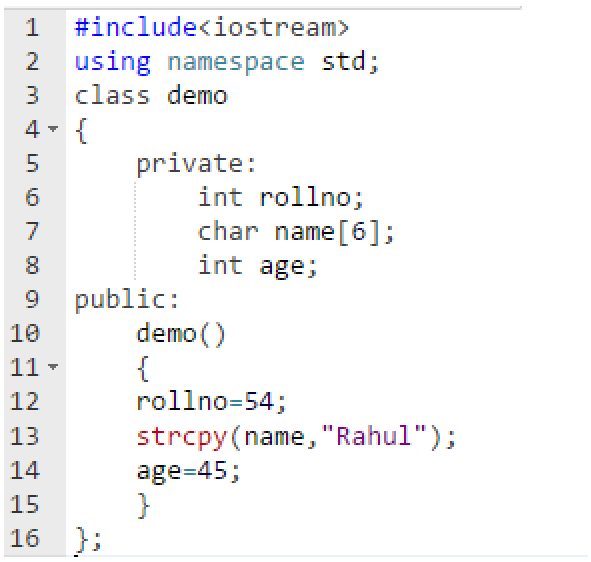
C++ Destructors also have same name as that of constructor but it is just identified by tilde (~) symbol before constructor name. In the above program 2 objects are created using constructor and same objects are destroyed using a destroyed.

(viii) Manipulating Private Data Members

In C++, when we declare a class with public and private members; by default, we cannot access the private data members from outside the class. One way is to declare a function as friend function of that class.

But there is an alternative method using which we may access and manipulate the private data members from outside the class without the use of a friend.

In the below example it is demonstrated how to access and manipulate private data members.



```
#include<iostream>
using namespace std;
class demo
{
    private:
        int rollno;
        char name[6];
        int age;
public:
    demo()
    {
    rollno=54;
    strcpy(name,"Rahul");
    age=45;
    }
};
```

In the above program class contains 3 data members, namely “age”, “rollno” and “name”. It also has a constructor, which initialises the value of the data members.

Now we may declare a demo type object named obj1:-

Demo obj1;

When any class type object is declared the data members occupy the space starting from the initial address of the object and they do so in the order in which they are declared within a class. So this object obj1 will be created and a memory space will be allocated to it in the RAM. And all of its data members will be present in that memory space of obj1, starting from &obj1. With this we now know that the integer will be allocated the starting address of object itself, followed by the string and float. Now in order to access these members, store the address of object in a void type pointer, because it has to be type-casted later.

void *prt=&obj1;

Now we can access the first data member by typecasting it into int* but in order to access all the members in a simpler way well create a structure with same type of data members as in obj1 and too in the same order.

```
struct Data
{
   int number;
   char name[6];
   float pi;
};
```

Then we will create a pointer to the structure of type Data and store in it the address of obj1

```
   Data *ptr = (Data*)ptr;
```

Now we have a pointer which points to the private data members of the object obj1. To access them we can use this pointer with an indirection operator (->).

```
   cout<<ptr->number<<endl;
   cout<<ptr->name<<endl;
   cout<<ptr->pi<<endl;
```

This code will display the private data members of obj1 from the function, which contains the declarations and assignments of pointers. J

We can manipulate the private data members by manipulating the same code as shown below.

For example,

```
   ptr->number=222 ;
   strcpy(ptr->name, Raj);
```

This code will not work for the static members present in class because they are not present within the memory space of the object but at another location in RAM, which starts at 0x00AA. U may use this address to access the first static data member and operate on this address to access the other static data members.

Did you know?

The copy constructor can access the private members of its argument.

Self-assessment Questions

19) Constructors and destructors are declared within a ______declaration

    a) Class   b) Object

    c) Function   d) Structure

20) Constructors and destructors cannot have a ____________.

21) Dynamic constructor enables the program to allocate the right amount of memory to data members of the object during execution.

    a) True   b) False

Summary

- A class is a specification or blueprint for a number of objects.
- Objects consist of both data and functions that operate on that data.
- In a class the data members and member functions—can be private or public.
- Private data members can be accessed only by member functions of that class.
- Public data members can be accessed by any function in the program..
- A member function is a function that is a member of a class. Member functions have access to an object's private data, while non-member functions do not.
- A constructor is a member function, with the same name as its class, which is executed every time an object of the class is created.
- A constructor has no return type but can take arguments
- It is often used to give initial values to object data members.
- Constructors can be overloaded, so an object can be initialized in different ways.
- A destructor is a member function with the same name as its class but is preceded by a tilde (~).
- A destructor is called when an object is destroyed. A destructor takes no arguments and has no return value.
- When objects are created there is a separate copy of the data members for each object that is created from a class, but there is only one copy of a class's member functions.
- You can restrict data item to a single instance for all objects of a class by making it static.

Terminal Questions

1. Explain classes and objects

2. Compare and contrast the access specifier available in C++.

3. Differentiate between classes and structures

4. Explain the constructors and the destructors

5. Discuss the concept of

   a) Copy constructor   b) Constructor overloading

Answer Keys

Self-assessment Questions	
Question No.	Answer
1	Instance
2	TRUE
3	b
4	Object name and dot operator
5	b
6	Actual object
7	Value
8	a
9	a
10	a
11	New
12	a
13	Class
14	Public, private and protected
15	a
16	a
17	a
18	TRUE
19	a
20	Return type
21	TRUE

Activities

Activity Type: Online **Duration:** 45 Minutes

Description:

1. Define a class to represent a bank account. Include the following members

   a) Data members - Name of the depositor

   - Account number

   - Type of Account

   - Balance amount in the account

   b) Member functions

   - To assign initial values

   - To deposit an account

   - To withdraw an amount after checking the balance

   - To display name and balance

   Write a main program to test the program.

2. Design Constructor for the classes designed above

Bibliography

e-References

- tutorialspoint.com, (2016). *C++ Classes and Objects.* Retrieved 7 April, 2016 from http://www.tutorialspoint.com/cplusplus/cpp_classes_objects.htm
- geeksforgeeks.org, (2016). *Structure vs class in C++.*Retrieved 7 April, 2016 from http://www.geeksforgeeks.org/g-fact-76/
- tutorialspoint.com, (2016). *C++ Class Constructor and Destructor.* Retrieved 7 April, 2016 from http://www.tutorialspoint.com/cplusplus/cpp_constructor_destructor.htm

External Resources

- Balaguruswamy, E. (2008). *Object Oriented Programming with C++.* Tata McGraw Hill Publications.
- Lippman. (2006). *C++ Primer* (3rd ed.). Pearson Education.
- Robert, L. (2006). *Object Oriented Programming in C++* (3rd ed.). Galgotia Publications References.
- Schildt, H., & Kanetkar, Y. (2010). *C++ completer* (1st ed.). Tata McGraw Hill.
- Strousstrup. (2005). *The C++ Programming Language* (3rd ed.). Pearson Publications.

Video Links

Topic	Link
Classes and objects	https://www.youtube.com/watch?v=kj5fV4Ibb2w
Member function defined outside the class	https://www.youtube.com/watch?v=59fy7la7yEI
Classes and structures	https://www.youtube.com/watch?v=wigjR4LkFPA

Notes: