UNIT 4 CONSTRUCTORS AND DESTRUCTORS

Structure **Page Nos.**

4.0 INTRODUCTION

In the previous unit, we have discussed concept of Objects and Class and their use in object oriented programming. In this unit we shall discuss about constructors and destructors and their use in memory management for C++ programming.

C++ allows different categories of data types, that is, built-in data types (e.g., int, float, etc.) and user defined data types (e.g., class). We can initialize and destroy the variable of user defined data type (class), by using constructor and destructor in object oriented programming.

Constructor is used to create the object in object oriented programming language while destructor is used to destroy the object. The constructor is a function whose name is same as the object with no return type. The name of the destructor is the name of the class preceded by tilde **(~).** First we will discuss briefly about constructor and destructor and then move on to the types of constructor and memory management.

Whenever an object is created, the special member function, that is, constructor will be executed automatically. Constructors are used to allocate the memory for the newly created object and they can be overloaded so that different form of initialization can be accommodated. If a class has constructor, each object of that class will be initialized. It is called constructor because it constructs the value of data members of the class.

Let us take an example to illustrate the syntax and declaration of constructor and destructor, inside a class in object oriented programming.

Example: A constructor is declared and defined as follows:

```
// class with constructor

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

        integer :: integer (void)           // constructor defined
        {
                m = 0, n = 0;
        }
```

In the above example class integer contains a constructor, and the object created by the class is initialized automatically.

4.1 OBJECTIVES

After studying this unit, you should be able to:

- explain the basic concepts of constructor and destructor;
- describe purpose of Constructor and Destructor;
- explain types of constructor and destructor;
- use Automatic Initialization, and
- memory management by using constructor and destructor in programming.

4.2 CONSTRUCTOR AND DESTRUCTOR

C++ is an object oriented programming (OOP) language which provides a special member function called constructor for initializing an object when it is created. This is known as automatic initialization of objects. It also provides another member function called destructor which is used to destroy the objects when it is no longer required. Constructor has the same name as that of the class's name and its main job is to initialize the value of the class. Constructors and destructors have no return type, not even void.

Declaration of destructor function is similar to that of constructor function. The destructor's name should be exactly the same as the name of constructor; however it should be preceded by a tilde (~). For example if the class name is student then the prototype of the destructor would be ~ student ()

To illustrate the use of constructor, let us take the following C++ program:

C++ Program to illustrate constructor:

```
# include<iostream.h>
# include<conio.h>
class student
{
public:
student()                    // user defined constructor
{
cout<< "object is initialized"<<end1;
}
};
void main ()
{
student x,y,z;
getch();
}
```

Output:

Object is initialized

Here in the above program, one default constructor student has been created which is similar to the class name student. When objects of the student class are created, then default constructor is automatically executed, and three times it displays the output "Object is initialized". Since here, three objects, x, y and z have been defined for every execution student constructor.

4.2.1 Characteristics of Constructors

A constructor for a class is needed so that the compiler automatically initializes an object as soon as it is created. A class constructor if defined is called whenever a program creates an object of that class. The constructor functions have some special characteristics which are as follows:

- They should be declared in the public section.
- They are invoked directly when an object is created.
- They don't have return type, not even void and hence can't return any values.
- They can't be inherited; through a derived class, can call the base class constructor.
- Like other C++ functions, they can have default arguments.
- Constructors can't be virtual.
- Constructor can be inside the class definition or outside the class definition.
- Constructor can't be friend function.
- They can't be used in union.
- They make implicit calls to the operators new and delete when memory allocation is required.

When a constructor is declared for a class, initialization of the class objects becomes necessary after declaring the constructor

Limitations of Constructor:

C++ constructors have the following limitations:

- **No return type**
  A constructor cannot return a result, which means that we cannot signal an error, during the object initialization. The only way of doing it is to throw an exception from a constructor.
- **Naming**
  A constructor should have the same name as the class, which means we cannot have two constructors that both take a single argument.
- **Compile time bound**
  At the time when we create an object, we must specify the name of a concrete class which is known at compile time. There is no way of dynamic binding constructors at run time.
- **There is no virtual constructor**
  We cannot declare a virtual constructor. We should specify the exact type of the object at compile time, so that the compiler can allocate memory for that specific type. If we are constructing derived object, the compiler calls the base class constructor first and the derived class hasn't been initialized yet. This is the reason why we cannot call virtual methods from the constructor.

4.2.2 Declaration of Constructors

A member function with the same name as its class is called constructor and it is used to initialize the objects of that class type with an initial value. Objects generally need to initialize variables or assign dynamic memory during their process of creation to become operative and to avoid returning unexpected values during their execution. For example, to avoid unexpected results in the example given below we have initialized the value of rollno as 0 and marks as 0.0.

In order to avoid that, a class can include a special function called constructor, which is automatically called whenever a new object of this class is created. This constructor function must have the same name as the class, and cannot have any return type; not even void.

A constructor is declared and defined as follows:

```
// class with constructor

class student
{
        private:
                int rollno;
                float marks;
        public:
              .
              .
              student ()
             {
        // constructor student declared having same name as
                        // that of its class
                        rollno = 0;
        // constructor value initialized
                        marks = 0.0;
                }
                .
                        .
}
```

In the above example class student has the constructor function defined with the same name and its values are initialized after creating it. When a class contains a constructor like one defined above, it is granted and understood that an object created by the class will be initialized automatically.

The output of the above C++ program is: rollno and marks of the student class.

4.2.3 Application of Constructors

A constructor is a special method that is created when the object is created or defined. This particular method holds the same name as that of the object and it initializes the instance of the object whenever that object is created. The constructor also usually holds the initializations of the different declared member variables of its object. Unlike some of the other methods, the constructor does not return a value, not even void.

When you create an object, if you do not declare a constructor, the compiler would create one for your program; this is useful because it lets all other objects and functions of the program know that this object exists. This compiler created constructor is called the default constructor. If you want to declare your own constructor, simply add a method with the same name as the object in the public section of the object. When you declare an instance of an object, whether you use that object or not, a constructor for the object is created and signals itself.

Generally, a constructor should be defined under the public section of a class, so that its objects can be created in any function.

A class can have multiple constructors for various situations. Constructors overloading are used to increase the flexibility of a class by having more number of constructor for a single class. To illustrate this let us take the following C++ program

```
# 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;
  }
```

Here in the above example, the constructor "Overclass" is overloaded thrice with different initialized values.

4.2.4 Constructor with Arguments

Parameters and arguments are sometimes loosely used interchangeably; in particular, "argument" is sometimes used in place of "parameter". Nevertheless, there is a

difference. Properly, parameters appear in procedure definitions; arguments appear in procedure calls.

A parameter is an intrinsic property of the procedure, included in its definition. For example, in many languages, a minimal procedure to add two supplied integers together and calculate the sum total would need two parameters, one for each expected integer. In general, a procedure may be defined with any number of parameters or no parameters at all. If a procedure has parameters, the part of its definition that specifies the parameters is called its parameter list.

By contrast, the arguments are the values actually supplied to the procedure when it is called. Unlike the parameters, which form an unchanging part of the procedure's definition, the arguments can, and often do, vary from call to call. Each time a procedure is called, the part of the procedure call that specifies the arguments is called the argument list. Many programmers use parameter and argument interchangeably, depending on context to distinguish the meaning.

To better understand the difference, let us consider the following function:

```
int sum(int addend1, int addend2)
{
   return addend1 + addend2;
}
```

The function sum has two parameters, named addend1 and addend2. It adds the values passed into the parameters, and returns the result

The code which calls the *sum* function might look like this:

```
int sumvalue;
int value1 = 40;
int value2 = 2;
sumvalue = sum(value1, value2);
```

The variables value1 and value2 are initialized with values. Value1 and value2 are both arguments to the sum function in this context.

At runtime, the values assigned to these variables are passed to the function sum as arguments. In the sum function, the parameters addend1 and addend2 are evaluated, yielding the arguments 40 and 2, respectively.

The values of the arguments are added, and the result is returned to the caller, where it is assigned to the variable sumvalue.

☞ Check Your Progress 1

Multiple choice questions:

a) Default constructor takes _____

1) one parameter
2) two parameters
3) no parameters
4) character type parameter

b) A constructor is called when ever _____

1) An object is declared
2) An object is used
3) A class is declared
4) A class is used

c) The difference between constructor and destructor are_____

1) Constructor can take arguments but destructor didn't
2) Constructor can be overloaded but destructor didn't
3) Both a & b
4) None of these

d) A class having no name_____
1) is not allowed
2) can't have a constructor
3) can't have a destructor
4) can't be passed as an argument

Short answer type questions:

1: What do you mean by constructor in C++ programming?

……

……

……

2: Explain the application of constructors and destructors in memory management with reference to C++ programming.

……

……

……

3: What is the difference between parameter and argument with respect to constructor and destructor in C++ programming?

……

……

……

4: In what ways a constructor is different from an automatic initialization?

……

……

……

4.3 TYPES OF CONSTRUCTOR

In object oriented programming (OOP's), a constructor in a class is used to initialize the value of the object. It prepares the new object for use by initializing its legal value. We will discuss main types of constructors, their features and applications in the following section:

4.3.1 Default Constructor

A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values.

If no user-defined constructor exists for a class A and one is needed, the compiler implicitly declares a default parameter less constructor A::A(). This constructor is an inline public member of its class. The compiler will implicitly define A::A() when the compiler uses this constructor to create an object of type A. The constructor will have no constructor initializer and a null body. For instance, consider the following example

```
class A {
                int i;
                public:
                void getval(void);
                void prnval(void);
                // member function definitions
        }

        A 0b 1;
// uses default constructor for creating   0b1. Since user can use it,
// that means, this implicitly defined default constructor is public
// member of the class
        0b1. Getval();
        0b1. Getval();
```

Having a default constructor simply means that an application can declare instances of the class. The compiler first implicitly defines the implicitly declared constructors of the base classes and non-static data members of a class A before defining the implicitly declared constructor of A. No default constructor is created for a class that has any constant or reference type members.
A constructor of a class A is trivial if all the following statements are true:

- It is implicitly defined
- A has no virtual functions and no virtual base classes
- All the direct base classes of A have trivial constructors
- The classes of all the non-static data members of A have trivial constructors
- The default constructor provided by the compiler does not do anything specific. It simply allocates memory to data members of the object.

4.3.2 Parameterized Constructor

We can write a constructor in C++ which can accept parameters for its invocation. Such constructor that can take the arguments are called parameterized constructor.

In other words; "Declaring a constructor with arguments hides the default constructor". This means that we can always specify the arguments whenever we

declare an instance of the class. To illustrate parameterized constructor well, let us write the syntax and take one example:

Syntax:

```
class <cname>
{
   //data
   public: cname(arguments); // parameterized constructor
         .....
         .....
};
```

Examples:

```
class student
{
     char name[20];
     int rno;
     public: student(char,int);       //parameterized constructor
};
student :: student(char n,int r)
{
    name=n;
    rno=r;
}
```

In the above example parameterized constructor has been shown by comment line. When the constructor is parameterized, we must provide appropriate arguments for the constructor.

4.3.3 Copy Constructor

A copy constructor is used to declare and initialize an object from another object. For example, the statement

```
integer 12(11);
```

would define the object 12 and at the same time initialize it to the value of 11. Another form of this statement is

```
integer 12 = 11;
```

Thus the process of initializing through a copy constructor is known as copy initialization. A copy constructor is always used when the compiler has to create a temporary object of a class object. The copy constructors are used in the following situations

- The initialization of an object by another object of the same class.
- Return of objects as a function value.
- Stating the object as by value parameters of a function.

The syntax of copy constructor is:

```
class_name :: class_name(class_name &ptr)
```

Another Example of copy constructor is:

X :: X(X &ptr)

Here, X is user defined class name and ptr is pintor to a class object X.

- Normally, the copy constructor takes an object of their own class as arguments and produces such an object.

- The copy constructors usually do not return a function value. Constructors cannot return any function values.

Program to illustrate the use of Copy Constructor:

```
#include<iostream>
#include<conio.h>
using namespace std;
class student
{

     int roll;
     char name[30];
   public:
       student()                          // default constructor
         {
               roll =10;
               strcpy(name,"x");
         }
       student( student &O)               // copy constrcutor
         {
               roll =O.roll;
               strcpy(name, O.name);
         }

         void input_data()
          {
                    cout<<"\n Enter roll no :"; cin>>roll;
                    cout<<"\n Enter name :"; cin>>name;
           }
         void show_data()
          {
             cout<<"\n Roll no :"<<roll;
             cout<<"\n Name    :"<<name;
          }
};
int main()
 {
        student s;                          // default construtor
        s.show_data();
        student A(s);                       // copy constructor
        A.show_data();
        getch();
        return 0;
        }
```

Here in the above C++ program we have taken a user defined class student and used copy constructor to initialize another object student rollno and name from the student constructor.

4.3.4 Constructor Overloading

When more than one constructor function is defined in a class, then it is called constructor overloading or use of multiple constructor in a class. It is used to increase the flexibility of a class by having more number of constructors for a single class. Overloading constructors in C++ programming gives us more than one way to initialize objects in a class.

To illustrate the constructor overloading, let us take following examples:

```
#include<iostream>
#include<conio.h>
using namespace std;
class student
{
int roll;
     char name[30];
   public:
      student(int x, char y[])                  // parameterized constructor
        {
              roll =x;
              strcpy(name,y);
        }
       student()                                 // normal constructor
        {
              roll =100;
              strcpy(name,"y");
        }

        void input_data()
         {
           cout<<"\n Enter roll no :"; cin>>roll;
           cout<<"\n Enter name :"; cin>>name;
         }
        void show_data()
         {
            cout<<"\n Roll no :"<<roll;
            cout<<"\n Name    :"<<name;
         }
};

int main()
 {
        student s(10,"z");
        s.show_data();
        getch();
        return 0;
        }
```

The above C++ program is used to create a constructor student inside the class student. Here in this program, the student constructor has been defined in many ways and the output of the program is student name and his roll no.

```
// overloading class constructors

#include <iostream>
using namespace std;
class CRectangle
{
   int width, height;
  public:
   CRectangle ();
   CRectangle (int,int);
   int area (void)
            {
                     return (width*height);
            }
};

CRectangle::CRectangle ()
            {
              width = 5;
              height = 5;
            }

CRectangle::CRectangle (int a, int b)
            {
              width = a;
              height = b;
            }

int main ()
{
  CRectangle rect (3,4);
  CRectangle rectb;
  cout << "rect area: " << rect.area() << endl;
  cout << "rectb area: " << rectb.area() << endl;
return 0;
}
```

Output:

rect area: 12
rectb area: 25

In this case, rectb was declared without any arguments, so it has been initialized with the constructor that has no parameters, which initializes both width and height with a value of 5.

4.3.5 Constructing two Dimensional Constructor

A typical two-dimensional array is like a time-table. To locate a piece of information, you determine the required row and column and then read the location where they meet.

In the same way, a two dimensional array is a grid containing rows and columns in which each element is uniquely specified by means of its row and column coordinates. Two-dimensional character arrays hold an array of strings wherein a row represents a string and a column represents a single character in each of the strings.

Declaration of Two Dimensional Arrays:

The general form of declaration of a two-dimensional character array is:

```
char arrayname[x][y];
```

where 'x' is the number of rows and 'y' is the number of columns.

The following guidelines need to be followed while declaring two-dimensional character arrays.

- The number of rows should be equal to the number of strings in the array.
- Column specification should be greater than or equal to the length of the longest string in the array plus one.

For example, if there are seven strings in an array and the length of the longest string is nine, the array can be declared in the following manner:

char cWeekdays[7][10];

Initializing of Two-dimensional Arrays:

The rules for initializing two-dimensional arrays are the same as for one-dimensional arrays.

For example, to declare and initialize an array that would hold the days of the week, the array definition would be as follows:

```
char cWeekdays[7][10] = {"Sunday","Monday","Tuesday","Wednesday",
"Thursday","Friday","Saturday"};
```

Explanation of the code:

In the above example, cWeekdays[0] will refer to the string "Sunday" and cWeekdays[4][1] would refer to the character 'h' of the string "Thursday".

We have discussed the constructor and their types in the above paragraphs and we have come to the end of constructor with the discussion of two dimensional array in object oriented programming.

We will discuss destructor and memory management in coming pages of this unit. The function of destructor is opposite to that of constructor and is used to free the memory which was allocated for the creation of constructor. Let us now discuss the destructor in detail in the next paragraph.

☞ Check Your Progress 2

Multiple choice questions:

1: A constructor is called whenever

a) An object is declared
b) An object is used
c) A class is created
d) A class is used

2: A destructor takes

a) One argument
b) Two argument
c) Three argument
d) Zero argument

3: The difference between constructor and destructor is

a) Constructor can take argument but destructor can not
b) Constructor can be overloaded but destructor can not
c) Both a and b
d) None of the above

Short Answer type questions:

1: What do you mean by default constructor? Explain by taking example.

……
……
……

2: What do you mean by copy constructor? Explain it with a C ++ Program.

……
……
……

3: Define parameterized constructor by taking a C++ program.

……
……
……

4: What do you mean by constructor overloading in C ++. Explain by taking one example.

……
……
……

4.4 DESTRUCTORS

Destructors are functions that are complimentary to constructors. A destructor, as the name implies, is used to destroy the objects that have been created by using constructor. They de-initialize objects when they are destroyed. A destructor is invoked when an object of the class goes out of scope, or when the memory occupied by it is de allocated using the delete operator. A destructor is a function that has the same name as that of the class but is prefixed with a ~ (tilde).

For example the destructor for the class students will bear the name ~student(). Destructor takes no arguments or specifies a return value, or explicitly returns a value not even void. It is called automatically by the compiler when an object is destroyed. A destructor cleans up the memory that is no longer required or accessible. To understand the syntax of destructor let us take following example.

4.5 DECLARATION OF DESTRUCTOR

Let us take the following C++ program to illustrate the declaration of destructor

Program: 1

```
#include<iostream.h>
#include<conio.h>
class student                          // student is a class
{
public :
~student()                             // destructor declaration
{
cout <<"Thanx for using this program" <<endl;
}
};
main()
{
clrscr();
student s1; // s1 is an object
getch();
}
```

Here in the above program ~student () function has been used for destroying the use of memory which was allocated to constructor class by the operating system.

Program: 2

```
#include<iostream>
#include<conio.h>
using namespace std;
class xyz
  {
     public:
          xyz()
            {
               cout<<"\n Constructor ";
            }
          ~xyz()              // use of destructor with  -- tilde
            {
               cout<<"\n Destructor ";
            }
  };
int main()
  {
     {
       xyz B;
      }
     getch();
     return 0;
  }
```

Here in the above program ~xyz() function has been used for destroying the use of memory which was allocated during the initialization of the constructor to constructor class.

Destructors are less complicated than constructors. You don't call them explicitly

(they are called automatically for you), and there's only one destructor for each object. The name of the destructor is the name of the class, preceded by a tilde (~).

Only the function having access to a constructor and destructor of a class can define objects of this class types otherwise compiler reports error at the time of compilation of the program. Generally, a destructor should be defined under the public section of the class, so that its objects can be destroyed in any function.

Further, destructors are usually used to de-allocate memory and do other cleanup for a class object and its class members when the object is destroyed. A destructor is called for a class object when that object passes out of scope or is explicitly deleted. A destructor is a member function with the same name as its class prefixed by a ~ (tilde). For example:

```
class X
 {
  public:
  // Constructor for class X
  X();
  // Destructor for class X
  ~X();
};
```

A destructor takes no arguments and has no return type. Its address cannot be taken. Destructors cannot be declared const, volatile, const volatile or static. A destructor can be declared virtual or pure virtual.

4.6 APPLICATION OF DESTRUCTOR

During construction of an object by the constructor, a resource may be allocated for use. For example, a constructor may have opened a file and a memory area may be allocated to it. Similarly, a constructor may have allocated memory to some other objects. These allocated resources must be deallocated before the object is destroyed. In object oriented programming such as C++, this work is done by using destructor which performs all clean-up tasks and is equally useful as constructor is in managing the memory and resources of the computer system.

The main application and characteristics of destructors are given as follows:

1. Destructor functions are invoked automatically when the objects are destroyed.
2. We can have only one destructor for a class, i.e. destructors can't be overloaded.
3. If a class have destructor, each object of that class will be de initialized before the object goes out of the scope
4. Destructor function, also obey the usual access rules of as other member function do.
5. No argument can be provided to a destructor, neither does it return any value.
6. Destructor can't be inherited.
7. A destructor may not be static.
8. It is not possible to take the address of destructor.
9. Member function may be called from within a destructor.
10. An object of a class with a destructor can't be a member of a union.
11. We may make an object const if it does not change any of its data value.

Limitations of destructors:

C++ destructors have mainly two disadvantages:

1) They are case sensitive.
2) They are no good for big programs having thousand lines of code.

4.7 PRIVATE CONSTRUCTOR AND DESTRUCOTR

In object oriented programming such as C++, private constructor is used to create object of class only one time. It means that your class does not have more than one instance of your class. Private constructor is same as any other constructor but it is declared as private. Due to this, you can only access object of that class and not in other class. Most of the people use private constructor which do all the work desired in the program by constructor.

Let us take an example to understand the use of private constructor:

```
class student()
        {
        private:
```

```
Student ()  {}
        public:
        student  & instances()
                {
                Static student * aGlobalInsts = new student ();
                Return * student;
                }
        }
```

In the above example, private constructor student has been defined in the private section of the program and can be accessed within the private section only. The output of the above C++ program is details of student.

In object oriented programming such as C++, destructor are declared in private section to prevent the object of one class from automatic deleting by the destructor of any other class. If a class is in singleton then it is possible to declare constructor and destructor as private. However, in standard programming of C++ destructors are declared as public and not private.

4.8 PROGRAMS ON CONSTRUCTOR & DESTRUCTOR

A program to print student details using constructor and destructor:

```
#include<iostream.h>
  #include<conio.h>
  class stu
  {
            private: char name[20],add[20];
                     int roll,zip;
                public: stu ( );//Constructor
                     ~stu( );//Destructor
                     void read( );
                     void disp( );
  };
            stu :: stu( )
  {
            cout<<"This is Student Details"<<endl;
  }
  void stu :: read( )
  {
  cout<<"Enter the student Name";
  cin>>name;
  cout<<"Enter the student roll no ";
  cin>>roll;
  cout<<"Enter the student address";
  cin>>add;
  cout<<"Enter the Zipcode";
  cin>>zip;
  }
  void stu :: disp( )
  {
```

```
        cout<<"Student Name :"<<name<<endl;
                cout<<"Roll no   is      :"<<roll<<endl;
                cout<<"Address is       :"<<add<<endl;
                cout<<"Zipcode is       :"<<zip;
        }
        stu : : ~stu( )
        {
                cout<<"Student Detail is Closed";
        }

        void main( )
        {
        stu s;
        clrscr( );
        s.read ( );
        s.disp ( );
        getch( );
        }
```

Output:

Enter the student Name
Sushil
Enter the student roll no
01
Enter the student address
Delhi
Enter the Zipcode
110092
Student Name : Sushil
Roll no is : 01
Address is : Delhi
Zipcode is :110092

A program to calculate factorial of a given number using copy constructor

```
#include<iostream.h>
#include<conio.h>
class copy
{
        int var,fact;
        public:

         copy(int temp)
         {
          var = temp;
         }

         double calculate()
         {
                fact=1;
                for(int i=1;i<=var;i++)
                {
                fact = fact * i;
```

```
                    }
                    return fact;
                }
};
void main()
{
   clrscr();
   int n;
   cout<<"\n\tEnter the Number : ";
   cin>>n;
   copy obj(n);
   copy cpy=obj;
   cout<<"\n\t"<<n<<" Factorial is:"<<obj.calculate();
   cout<<"\n\t"<<n<<" Factorial is:"<<cpy.calculate();
   getch();
}
```

Output:

Enter the Number: 5
Factorial is: 120
Factorial is: 120

C++ Program for Add two time variables using constructor and destructor

```
#include<iostream.h>
#include<conio.h>
class Time
{
int minutes,hours,a;
static int i;
public:
Time(int a)
{
this->a=hours=a;
this->a+=5;
minutes=i++;
cout<<"\nObj address : "<<this;
cout<<"\nAddress of i : "<<&i;
cout<<"\na= "<<this->a<<"\t\t"<<a;
getch();
}
~Time()
{
cout<<endl<<"\t\t"<<hours<<" : "<<minutes;
getch();
}
};
int Time ::i;
void main()
{
clrscr();

Time t3(10),t2(1);
}
```

A C++ program by taking a "patient" class and by using constructor and destructor

```
#include <iostream>
#include <string>

using namespace std;

class Patient {
public:
   Patient(string name, int heartRate = 0, double moneyOwed = 0.0);

   Patient()
   {
      //default constructor for string, _name, implicitly called
      _heartRate = 0; //Bad for business
      _moneyOwed = 0.0;
   }

   int getHeartRate() {return _heartRate;}
   void setHeartRate(int heartRate) {_heartRate = heartRate;}
   double getMoneyOwed() {return _moneyOwed;}
   void setMoneyOwed(double moneyOwed) {_moneyOwed = moneyOwed;}
   string getName() {return _name;}
   void setName(string name) {_name = name;}
   ~Patient() {} // This could be defined outside
                 // the class definition instead

private:
   int _heartRate;
   double _moneyOwed;
   string _name;
};

// This could be defined within the class definition instead.
// Always initialize member classes within the member initialization list.
Patient::Patient(string name, int heartRate, double moneyOwed) : _name(name)
{
   _heartRate = heartRate;
   _moneyOwed = moneyOwed;
}

int main()
{

   Patient jk("John",70,0.0);

   //Before visit to doctor
   cout << "Patient " << jk.getName() << " owes " << jk.getMoneyOwed() << endl;
   cout << "His heart rate is " << jk.getHeartRate() << endl;

   //After visit to doctor
```

```
    jk.setMoneyOwed(1000.0);
    jk.setHeartRate(140);
    cout << "Patient " << jk.getName() << " owes " << jk.getMoneyOwed()
<< endl;
    cout << "His heart rate is " << jk.getHeartRate() << endl;

    return 0;
}
```

Finally, let us sum up the advantages and disadvantages of constructor and destructors:

- Constructors and destructors do not have return types nor can they return values.
- References and pointers cannot be used on constructors and destructors because their addresses cannot be taken.
- Constructors cannot be declared with the keyword virtual.
- Constructors and destructors cannot be declared static, const, or volatile.
- Unions cannot contain class objects that have constructors or destructors.

4.9 MEMORY MANAGEMENT

Memory management is a large subject area and will be discussed length other courses of BCA. However, we will have a brief discussion on memory management in this unit of constructor and destructor for the learners.

C++ offers a wide range of choices for the management of memory. There are various techniques to manage the memory in object oriented programming such as C++. Some of the techniques for memory management are given as follows:

Manual Memory Management: Up until the mid 1990s, the majority of programming languages used in industry supported manual memory management. Today, C++ is the main manually memory management languages. Manual memory management refers to the usage of manual instructions by the programmer to identify and de-allocate unused objects, or garbage.

All programming languages use manual techniques to determine when to *allocate* a new object from the free store. C++ language uses the new operator to determine when an object ought to be created and memory is allocated to it. There are two types of memory management operators in C++. These are:

- **new**
- **delete**

These two memory management operators are used for allocating and freeing memory blocks in efficient and convenient way.

The fundamental issue is the determination of when an object is no longer needed (i.e. is garbage), and arranging for its underlying storage to be returned to the free store so that it may be re-used to satisfy future memory requests.

Constructors and destructor are used in C++ programming language to initialize and destroy the memory blocks, when it is no longer required by the program. In manual memory allocation, this is also specified manually by the programmer; at the time of writing the codes by programmer.

Manual Memory Management and Correctness

- Manual memory management is known to enable several major classes of bugs into a program, when used incorrectly.
- When an unused object is never released back to the free store, this is known as a memory leak in C++ programming. In some cases, memory leaks may be tolerable, such as a program which "leaks" a bounded amount of memory over its lifetime, or a short-running program which relies on an operating system to de-allocate its resources when it terminates.
- Programmers are expected to invoke dispose() manually as appropriate; to prevent "leaking" of scarce graphics resources. However, in many cases memory leaks occur in long-running programs, and in such cases an unbounded amount of memory is leaked. Whenever this occurs, the size of the available free store continues to decrease over time; when it finally exhaust then the program crashes/terminates abnormally.
- When an object is deleted more than once, or when the programmer attempts to release a pointer to an object not allocated from the free store, catastrophic failure of the dynamic memory management system can result and cause bugs in the program.

☞ Check Your Progress 3

Multiple choice questions:

1. What is the only function all C++ programs must contain?
   A. start()
   B. system()
   C. main()
   D. program()

2. What punctuation is used to signal the beginning and end of code blocks?
   A. { }
   B. -> and <-
   C. BEGIN and END
   D. (and)

3. What punctuation ends most lines of C++ code?
   A. . (dot)
   B. ; (semi-colon)
   C. : (colon)
   D. ' (single quote)

Short Answer type questions

1: What do you mean by destructor? Explain by taking example.

……

……

……

2: What do you mean by private constructor and destructor?

……

……

……

3: Explain the role of destructor in C++ in memory management.

……

……

……

4: Write a program in C++ to demonstrate the use of destructor.

……

……

……

4.10 SUMMARY

A class constructor is a class method having the same name as the class name. The constructor does not have a return type. A constructor may take zero or more parameters. If we don't apply a constructor, the compiler provides a no-argument default constructor. It is important to understand that if we write our own constructor, the compiler does not provide the default constructor. A class may define one or more constructor. It is up to us to decide which constructor to call during object creation by passing an appropriate parameter list to the constructor. We may set the default value for the constructor parameter.

A class destructor is a class method having the same name as the class name and is prefixed with tilde (~) sign. The destructor does not return anything and does not accept any argument. A class definition may contain one and only one destructor. The runtime calls the destructor, if available, during the object creation. A destructor is typically used for freeing the resources allocated in the class constructor.

4.11 ANSWERS TO CHECK YOUR PROGRESS

Multiple choice questions

a) - 3, b)-1 c)-1 d)-3

Answers to short type questions

1: A member function with the same name as its class is called constructor and it is used to initialize the objects of that class type with an initial value. Objects generally need to initialize variables or assign dynamic memory during their process of creation to become operative and to avoid returning unexpected values during their execution.

2: There are various techniques to manage the memory in C++ programming. These are:

1) By using constructor and destructor
2) By using New and Delete operator
3) By using manual garbage collection feature of C++ programming
4) By using the function dispose() in C++ programming

3: Parameters and arguments are sometimes loosely used interchangeably; in particular, "argument" is sometimes used in place of "parameter". Nevertheless, there is a difference. Properly, parameters appear in procedure definitions; arguments appear in procedure calls.

4: Constructor needs to initialize before it is used and it is not automatically incremented or decremented during the execution of program, while automatic initialization takes place automatically, without any interaction of the user.

Check Your Progress 2

Multiple choice questions

1) a 2) d 3) c

Answers to short type answer questions

1: A default constructor is a constructor that either has no parameters, or if it has parameters, all the parameters have default values. Following program depicts the use of default constructor in C++ programming:

```
class A {       int i;
                        public:
                          void getval(void);
                          void prnval(void);
                           .
                           .                    // member function definitions
                }

        A 0b 1;         // uses default constructor for creating 0b1. Since user
can use it,
                        // that means, this implicitly defined default constructor
is public
                        // member of the class
        0b1. Getval();
        0b1. Getval();
```

2: A copy constructor is used to declare and initialize an object from another object. For example, the statement

```
integer 12(11);
```

would define the object 12 and at the same time initialize it to the value of 11. Another form of this statement is

```
integer 12 = 11;
```

3: We can write a constructor in C++ which can accept parameters for its invocation. Such constructor that can take the arguments are called parameterized constructor.

4: When more than one constructor function is defined in a class, then it is called constructor overloading or use of multiple constructor in a class. It is used to increase the flexibility of a class by having more number of constructors for a single class. Overloading constructors in C++ programming gives us more than one way to initialize objects in a class.

Check Your Progress 3

Multiple choice questions

1) c 2) a 3) b

Answers to short type answer questions

1: A destructor, as the name implies, is used to destroy the objects that have been created by using constructor. They de-initialize objects when they are destroyed. A destructor is invoked when an object of the class goes out of scope, or when the memory occupied by it is de allocated using the delete operator. A destructor is a function that has the same name as that of the class but is prefixed with a ~ (tilde). For example the syntax of class student could be written as follows

```
~student()
```

2: In object oriented programming such as C++ private constructor is used to create object of class only one time. It means that your class does not have more than one instance of your class. Private constructor is same as any other constructor but it is declared as private. Due to this you can only access object of that class in only that class and not in other class.

3: The role of destructors in C++ programming is given as follows:

1. Destructor functions are invoked automatically when the objects are destroyed.
2. We can have only one destructor for a class, i.e. destructors can't be overloaded.
3. If a class have destructor, each object of that class will be de initialized before the object goes out of the scope
4. Destructor function, also obey the usual access rules of as other member function do.
5. No argument can be provided to a destructor, neither does it return any value.
6. Destructor can't be inherited.
7. A destructor may not be static.
8. It is not possible to take the address of destructor.
9. Member function may be called from within a destructor.

4: A C++ program by using destructor is given as follows

```
#include<iostream.h>
#include<conio.h>
class student                    // student is a class
{
public :
~student()                       // destructor declaration
{
cout <<"Thanx for using this program" <<endl;
}
};
main()
{
clrscr();
student s1; // s1 is an object
getch();
}
```

4.12 FURTHER READINGS AND REFERENCES

1) E Balagurusamy, *Object Oriented Programming with C++*, Tata McGraw-Hill Publishing Company Ltd, 2004, New Delhi - 110008

2) Er V. K. Jain, *Object Oriented Programming with C++*, Cyber Tech Publication, Daryaganj N Delhi-110002

3) Robert Lafore, *Object Oriented Programming in C++*, Galgotia Publications Pvt. Ltd. Daryaganj N Delhi-11002

4) Rajesh K Shukla, *Object Oriented Programming in C++*, Wiley India Publishing Pvt. Ltd. Daryaganj, June 2008, New delhi-110002

5) Bjarne AT&T Labs Murray Hill, New Jersey Stroustrup, *Basics of C++ Programming*, Special Edition, Publisher: Addison-Wesley Professional.

Reference Websites:

(1) www.sciencedirect.com
(2) www.ieee.org
(3) www.webpedia.com
(4) www.microsoft.com
(5) www.freetechbooks.com
(6) www.computer basics.com
(7) www.youtube.com