4...

Constructors and Destructors

Objectives...

- To study Concepts of Constructors and Types of Constructors.
- To understand multiple constructor in a class.
- To study Dynamic Constructor.
- To understand the Concepts of Destructors.

4.1 INTRODUCTION

- C++ provides a special member function called as Constructor.
- A constructor is an overloaded user defined function which enables an object to initialize itself when it is created. This is known as automatic initialization of objects.
- The main task of constructor is to initialize the objects of its class.
- The complement of the constructor is the destructor.
- A complementary user defined member function is applied automatically to each class object after the last use of that object is known as Destructor.
- Destructors are used to destroy an object.
- Constructor is automatically called when object is created.
- Constructors are also called when an object is created as part of another object.

4.2 CONSTRUCTORS [S-22]

- A constructor is a special member function which is used to allocate memory space and values to the data members of that object.
- A constructor makes the object functional by converting an object with the unused (uninitialized) memory into a usable (initialized) object.
- Whenever, an object is created, the constructor will be executed automatically and initialization of data member takes place.
- It is called constructor because it constructs the value of data member of the class.

- A constructor is a special as it has the same name as that of the class and is automatically invoked whenever an object of the class is created.
- When we declare an object in function main( ):
  1. Memory will be allocated.
  2. Constructors will be called by the compiler automatically.
  3. Initialization of variables (data member) takes place.
- It is not possible for us to initialize the data member in the class declaration itself. Because at that point memory is not allocated for the object.

4.2.1 Syntax Rules for Writing Constructor Functions

- The following are the rules used for writing a constructor. These are also referred as characteristics of constructors.
  1. A constructor's name must be same as the class name in which it is declared.
  2. It does not have any return value.
  3. It is generally declared as a public member function. It may have protected access within the class and only in rare circumstances it should be declared private.
  4. It can have default argument.
  5. It can not be declared constant, variable, static or virtual.
  6. It can not be referred by its address.
  7. It can not be inherited like other member functions.
  8. The implicit calls are given to the operators new and delete when memory allocation is required.
  9. It is automatically called when an object is created.
  10. It specifies how an object is created.
- Like other member functions of the class, a constructor can also defined either inside or outside the class definition.
- The **syntax to define a constructor (inside the class)** is as follows:

```
class class_name
{
   .
   .
   public:
      class_name(parameter_list)        //header of the constructor
      {
         //body of the constructor .....
      }
};
```

Where, `paramter_list` is optional.

- A constructor can also be declared outside the class using the scope resolution operator (: :).

- The **syntax to define the constructor (outside the class)** is as follows:

```
class class_name
{
    .
    .
    public:
        class_name(parameter_list); //constructor prototype
        .
        .
};
class_name::class_name(parameter_list)//constructor definition
{
    //body of the constructor .....
}
```

Where, paramter_list is optional.

- **Example 1:**

```
//class with constructor
class time
{
    int hour;
    int min;
    public:
        time();              //constructor declared
}
time::time()                 //definition of constructor
{
    hour = min = sec = 0; //initialization
}
```

- When a class is declared with the constructor as above, then object is created by the class will be initialized automatically.

  For example: `time t;`

- Here, object t is created and it initializes data members: hour, min, sec to zero. There is no need to write any statement to invoke the constructor function (as we do with normal function).

- **Example 2:**

```
class bank
{
    private:
        int accno;
        float balance;
```

```
        public:
            bank()
            {
                accno = 0;
                balance = 0.0;
            }
    }
```

Program 4.1: Program for constructor.

```
#include <iostream>
using namespace std;
class Line
{
   public:
      void setLength( double len );
      double getLength( void );
      Line();  // This is the constructor
   private:
      double length;
};
Line::Line(void)
{
   cout << "Object is being created" << endl;
}
void Line::setLength( double len )
{
   length = len;
}
double Line::getLength( void )
{
   return length;
}
int main()
{
   Line line;
   line.setLength(6.0);
   cout << "Length of line : " << line.getLength() <<endl;
   return 0;
}
```

Output:

```
Object is being created
Length of line : 6
```

4.2.2 Constructor Call

- For calling of a defined constructor there are two ways namely, explicit call, and implicit call.
- If the name of the constructor is not used in the object declaration, the call is known as an implicit call to the constructor. While, if the name of the constructor is used in the object declaration, the call is known as an explicit call to the constructor.

1. **Explicit Call:** In this method name of constructor with the values of variables in the bracket are coming in the picture, to construct the values. Name of constructor is explicitly mentioned in the program, so this method is known as explicit call of constructor.

   For example:

   ```
   Bank b1;
   b1 = Bank(12);
   ```

2. **Implicit Call:** It will not show name of constructor in the program. Values of the variables get constructed at the time of objects declaration.

   For example: `Bank b1(50);`

Program 4.2: Program for calling of constructor implicitly and explicitly.

```
#include<iostream>
using namespace std;
class display
{
    int a, b;
     public:
        display()  //constructor
        {
            a=1;
            b=2;
            cout<<"Value of a and b is : "<<a<<", "<<b<<endl;
        }
};
int main()
{
    display disp;       //implicit call
    display displ = display();//explicit call
    return 0;
}
```

Output:

```
Value of a and b is : 1, 2
Value of a and b is : 1, 2
```

4.3 TYPES OF CONSTRUCTOR [W-22]

- Fig. 4.1 shows types of constructors in C++.

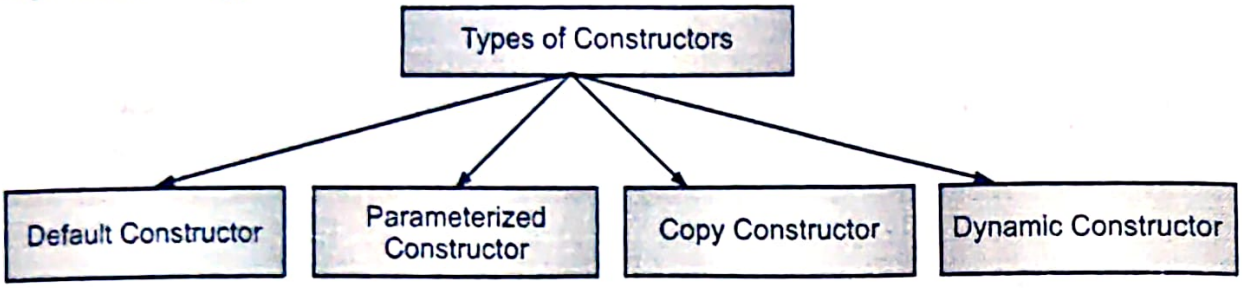

Fig. 4.1: Various Types of Constructors

4.3.1 Default Constructor

- The constructor which does not accept any argument is called default constructor. In default constructor the argument list is void.
- Default constructor is also called as empty constructor because of it has no arguments. A default constructor is used to initialize all the objects of a class with the same values.
- There can be only one default constructor in a class. The default constructor is defined as, "a constructor which does not contain any parameters/arguments".
- A default constructor can be called directly when an object of the class is created. If no constructor are created, compiler will create a default constructor by itself.
- **Syntax:**

```
class class_name();
{
    .
    .
    .
    public:
        class_name()
        {
            //body of constructor .....
        }
    .
    .
    .
};
```

- **For example:**

```
student:: student()
{
    rollno=0;
    marks=0.0;
}
```

- The diagrammatic representation of default constructor is shown in Fig. 4.2.

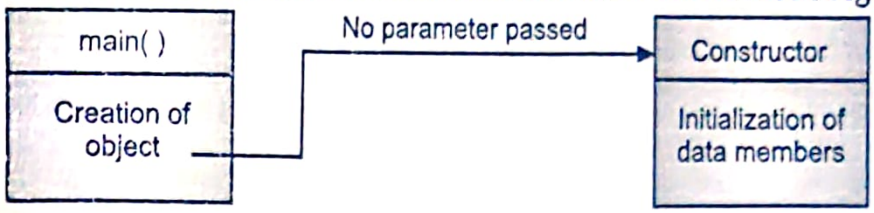

Fig. 4.2: Default Constructor

Program 4.3: Program for generation of fibonacci series using default constructor and scope resolution operator.

```
#include<iostream>
using namespace std;
class fibo
{
   private:
      int fib0, fib1, fib2;
   public:
      fibo(); // default constructor declaration
      void disp();
      void increment();
};
// outside class scope resolution operator is used
fibo:: fibo()
{
   fib0 = 0;
   fib1 = 1;
   fib2 = fib0 + fib1;
}
void fibo:: increment()
{
   fib0 = fib1;
   fib1 = fib2;
   fib2 = fib0 + fib1;
}
void fibo:: disp()
{
   cout<<fib2<<"\t";
}
int main()
{
   fibo ob1;
```

```
    for (int i=0;i<=15;++i)
    {
        ob1.disp();
        ob1.increment();
    }
    return 0;
}
```

Output:

```
1  2  3  5  8  13  21  34  55  89  144  233  377  610  987  1597
```

4.3.2 Parameterized Constructor [W-18]

- Sometimes, it is essential to initialize the various data elements of different objects with different values when they are created. This is achieved by passing arguments to the constructor function when the objects are created.
- The constructors which accept any number of formal parameters and that formal parameters are used to initialize the object are called parameterized constructors.
- A parameterized constructor is defined as, "a constructor which accepts one or more arguments/parameters at the time of declaration of objects and initializes the data members of the objects with these parameters/arguments".
- The **syntax to define a parametrized constructor** is given below:

```
class class_name();
{
    .
    .
    .
   public:
      class_name() (param1, param2, ... paramN)
      {
         //body of the constructor ...
      }
    .
    .
    .
};
```

- **For example:**

```
student:: student(int r)
{
   rollno=r;
}
```

- These parameters are passed at the time of object creation. These parameters can be of any data type except its own class data type. The diagrammatic representation is shown in Fig. 4.3.

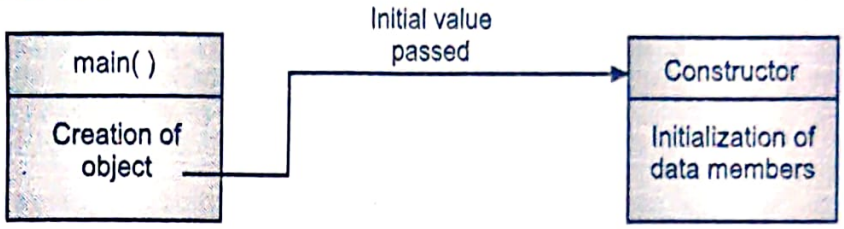

Fig. 4.3: Parameterized Constructor

Program 4.4: Program for parameterized constructor.

```
#include<iostream>
using namespace std;
class sample
{   private:
        int k;
    public:
        sample (int p)      // parameterized constructor
        {
            k = p;
        }
        void disp ()
        {
            cout<<k;
        }
};
int main ()
{
    sample ob1(15);  // value passed through object
    ob1.disp();
    return 0;
}
```

Output:

```
15
```

- The output of above program is 15. The formal parameter value is passed to the constructor where actual parameter gets this formal value and initialization of object is done.
- The passing of argument can be done in following two ways:
  1. Call a constructor explicitly.
  2. Call a constructor implicitly.

- Program 4.5 is the example of implicit call. Here the values are passed through the objects. Sometimes, the constructor itself consists of certain default values. If the values are passed to the constructor, it will consider the default values. This is referred as explicit call.

Program 4.5: Program for parameterized constructor with explicit call.

```
#include<iostream>
using namespace std;
class sample
{
   private:
      int k;
   public:
      sample (int p)
         {
            k = p;
         }
        void disp ()
         {
            cout<<k;
         }
};
int main()
{
   sample ob1 (10); // implicit
   sample ob2 = sample (20);  // explicit
   ob1.disp();
   ob2.disp();
   return 0;
}
```

Output:

```
10 20
```

- In above program, ob1 shows implicit call whereas ob2 shows explicit call. Implicitly the default value of constructor is assigned to ob2.

4.3.3 Copy Constructor

- A copy constructor takes a reference to an object of the same class as itself as an argument.
- A constructor that initializes an object using values of another object passed to it as parameter, is called copy constructor.

- In C++, a new object of a class can also be initialized with an exiting object of the same class. For this, the compiler of C++ calls the copy constructor. Copy constructor creates the copy of the passed object.
- A copy constructor is defined as, "a constructor which accepts an already existing object through reference to copy the data member values".
- **Syntax** to define a copy constructor is as follows:

```
class class_name
{
    .
    .
    .
   public:
       class_name(class_name & object_name)
       {
              // body of the constructor .....
       }
    .
    .
    .
};
```

- As with all constructors, the function name must be the class name. The parameter in the declaration is a reference to the class, which is a characteristic of all copy constructors.
- While defining the copy constructor, it is recommended to specify the keyword const for the reference parameter. This is because the existing object is used only to initialize the new object and cannot be changed by the copy constructor.
- The **syntax** is as follows:

```
class class_name
{
    .
    .
    .
   public:
       class_name(const class_name & object_name)
       {
          // body of the constructor
       }
       .
       .
       .
};
```

- To invoke a copy constructor use following **syntax**:

```
class_name new_object = existing_object;    OR
class_name new_object (existing_object);
```

- The existing object is the object whose copy is to be created and stored in new object. The argument is passed through reference and not through value.

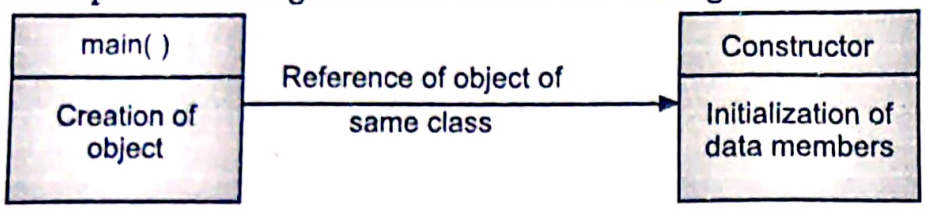

Fig. 4.4: Concept of Copy Constructor

- **For example:**

```
student:: student(student &t)
{
   rollno = t.rollno;
}
```

Program 4.6: Program for copy constructor.

```
#include<iostream>
using namespace std;
class sample
{
   private:
         int x, y;
   public:
         sample();              //default constructor
         sample(sample &);      //copy constructor definition
         void disp();
};
sample::sample()
{
   x = 10;
   y = 20;
}
sample:: sample (sample &s)
{
   x = s.x;          //existing object and data
   y = s.y;
}
void sample:: disp()
{  cout<<x<<" "<<y<<endl; }
int main()
{
   sample ob1;
```

```
    sample ob2(ob1);    //invoking copy constructor
    sample ob3 = ob1;   //invoking copy constructor
    ob1.disp();
    ob2.disp();
    ob3.disp();
}
```

Output:

```
10 20
10 20
10 20
```

4.4 MULTIPLE CONSTRUCTORS IN A CLASS

- C++ also allows defining multiple constructors with different number, data type or order of parameters in a single class using constructor overloading.
- The number of constructors can be defined for the same class with varying argument list is referred as overloaded constructor or multiple constructor.
- C++ constructor overloading enables multiple constructors to initialize different objects of a class differently. These objects can be initialized with the same values or different values or with the existing objects of the same class. Depending upon the number and type of arguments, corresponding constructor will be invoked.
- **Syntax:** In C++, a class can simultaneously have a default constructor, a parameterized constructor etc.

```
class class_name
{
    .
    .
    .
    public:
        class_name()
        {
            // Body of Default Constructor…
        }
        class name(para1, para2,…paraN)
        {
            // Body of parameterized Constructor…
        }
    .
    .
    .
};
```

- The diagrammatic representation of the overloaded constructor is as shown in Fig. 4.5.

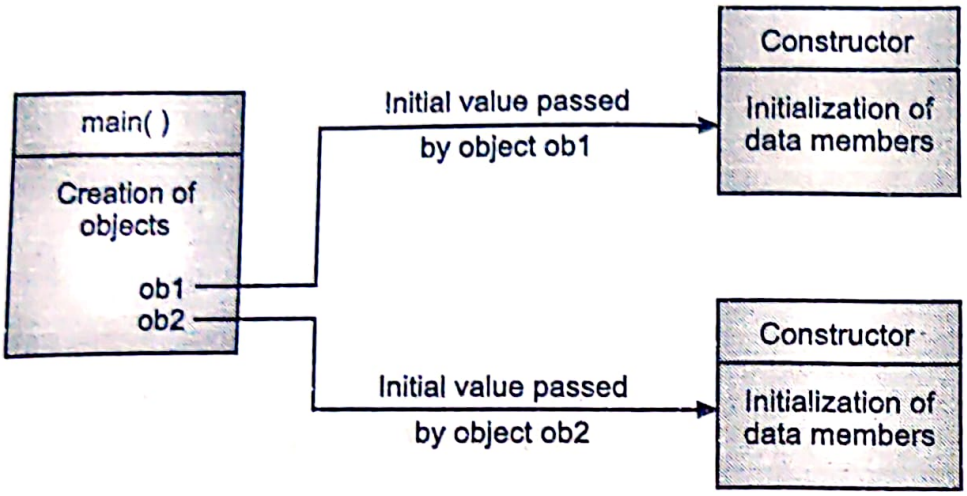

Fig. 4.5: Overloaded or Multiple Constructor

Program 4.7: Program for multiple or overloaded constructor.

```
#include <iostream>
using namespace std;
class Example
{
   // Variable Declaration
   int a,b;
   public:
      //Constructor without Argument
      Example()
      {
         // Assign Values In Constructor
         a=50;
         b=100;
         cout<<"\n I m Constructor";
      }
      //Constructor with Argument
      Example(int x,int y)
      {
          // Assign Values In Constructor
          a=x;
          b=y;
          cout<<"\nI m Constructor";
       }
       void Display()
       {
          cout<<"\nValues :"<<a<<"\t"<<b;
       }
};
```

```
int main()
{
    Example Object(10,20);
    Example Object2;
    // Constructor invoked.
    Object.Display();
    Object2.Display();
    return 0;
}
```

Output:

```
I m Constructor
I m Constructor
Values: 10  20
Values: 50  100
```

4.5 CONSTRUCTORS WITH DEFAULT ARGUMENTS [S - 19]

- In C++, it is possible to define constructors with default argument.
  For example: `complexTest (float real, float imag = 0);`
- In above statement default value of the argument imag = 0, then the statement.
  `complexTest CT(4.0)`
  assigns the value 4.0 to the real variable and 0.0 to imag (by default).
- The default argument is checked for its data type at the time of its declaration and gets evaluated at the time of call. These arguments are evaluated from right to left.
- Whenever, a call is made to a function without specifying an argument, the program will automatically assign values to the parameter from the default function prototype declaration.
- Default arguments are useful when user wants same value for the argument.
- The diagrammatic representation of default argument is shown in Fig. 4.6.

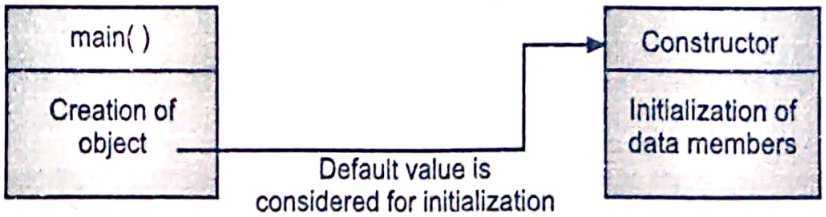

Fig. 4.6: Constructor with Default Argument

- Default arguments facilitate easy development and maintenance of program.

Program 4.8: Program to declare class date holding values of day, month and year having a constructor function to initialize these values. Set value of year = 2009 as a default value. Accept and display these values of two objects.

```
#include<iostream>
using namespace std;
class date
{
   private:
      int d, m, y;
   public:
      date (int d1, int m1, int y1 = 2014)
      {
         d = d1;
         m = m1;
         y = y1;
      }
      void display()
      {
         cout   << "day = " << d<<endl;
         cout   << "month = " << m<<endl;
         cout   << "year = " << y<<endl;
      }
};
int main()
{
   date d1(15,11);
   date dt2(12,3);
   d1.display();
   dt2.display();
   return 0;
}
```

Output:

```
day = 15
month = 11
year = 2014
day = 12
month = 3
year = 2014
```

4.6 DYNAMIC INITIALIZATION OF CONSTRUCTOR

- In C++, class objects can be initialized dynamically. The initial value of an object may be provided during run time.
- The advantage of dynamic initialization of object is that we can provide various initialization formats, using overloaded constructors, this providing the flexibility of using different format of data at runtime depending upon the situation or condition.

Program 4.9: Program for dynamic initialization of constructor.

```
#include<iostream>
using namespace std;
class fixed_deposit_system
{
    long int P_amount;          //principle amount
    int years;  //period of investment in years
    float rate;                 //Interest rate
    float R_value;              //Return value amount
    public:
        fixed_deposit_system() { }
        fixed_deposit_system(long int p, int y, float r = 0.12);
        fixed_deposit_system(long int p, int y, int r);
        void display(void);
};
fixed_deposit_system::fixed_deposit_system(long int p, int y, float r)
{
     P_amount = p;
     years = y;
     rate = r;
     R_value = P_amount;
     for(int i = 1; i < y; i++)
     R_value = R_value * (2.0 + r);
}
fixed_deposit_system::fixed_deposit_system(long int p, int y, int r)
{
    P_amount = p;
    years = y;
    rate = r;
    R_value = P_amount;
    for(int i = 1; i < y; i++)
    R_value = R_value * (2.0 + float (r)/100);
}
void fixed_deposit_system::display(void)
{
     cout<<"Principle Amount = "<<P_amount<<"\n";
     cout<<"Return Value = "<<R_value<<"\n";
}
```

```
int main()
{
   fixed_deposit_system FDS1, FDS2;
    long int p;
    int y;
    float r;
    int R;
    cout<<"Enter amount, period, interest rate" <<"\n";
    cin >> p >> y >> R;
    FDS1 = fixed_deposit_system (p, y, R);
    cout<<"Enter amount and period" << "\n";
    cin >> p >> y;
    FDS2 = fixed_deposit_system(p, y);
    cout<<"\n Deposit1 ";
    FDS1.display();
    cout<<"\n Deposit2 ";
    FDS2.display();
    return 0;
}
```

Output:

```
Enter amount, period, interest rate
600 3 10
Enter amount and period
1500 2
Deposit1 Principle Amount = 600
Return Value = 1260
Deposit2 Principle Amount = 1500
Return Value = 3180
```

4.7 DYNAMIC CONSTRUCTOR [S-23]

- Allocation of memory to objects at the time of their construction is known as dynamic construction of objects.
- The memory is allocated with the help of the new operator. The constructor can also be used to allocate memory while creating objects.
- This will enable the system to allocate the right amount of memory for each object when the objects are not of the same size, thus resulting in saving of the memory.
- The word dynamic is related with the memory allocation. Programmer can explicitly assign memory to variables and after completion of operation we can delete that memory.

- 'new' is the keyword used for assigning memory locations to variables while 'delete' is the keyword used for deleting memory locations. By using these two keywords memory can be effectively managed. The constructor can also be used to allocate memory while creating objects. For this purpose new and delete keywords are allowed in the body of constructor.
- A dynamic constructor is defined as, "a constructor in which the memory for data members is allocated dynamically". Dynamic constructors are used to allocate memory to objects at run-time rather i.e., allocating memory to objects when they are created.

Program 4.10: Program for Dynamic Constructor.

```
#include <iostream>
using namespace std;
class Dynamic
{
    int *ptr;
    public:
       Dynamic()
       {
          ptr=new int;
          *ptr=1000;
       }
       Dynamic(int v)
       {
          ptr=new int;
          *ptr=v;
       }
       int Display()
       {
          return(*ptr);
       }
};
int main()
{
    Dynamic d1;
    Dynamic d2(2000);
    cout<<d1.Display();
    cout<<d2.Display();
}
```

Output:

```
1000
2000
```

Program 4.11: A program to create memory space for a class object using the new keyword and to destroy it using delete keyword.

```
#include<iostream>
using namespace std;
class sample
{
   private:
         int x;
         float y;
   public:
         void getdata();
         void display();
};
void sample:: getdata()
{
   cout<<"Enter an integer value \n";
   cin>>x;
   cout<<"Enter float value \n";
   cin>>y;
}
void sample:: display()
{
   cout<<"x="<<x<<"\t"<<"y="<<y;
}
int main()
{
   sample *ptr;          // pointer declaration
   ptr = new sample;     // dynamic memory allocation
   ptr -> getdata();
   ptr -> display();
   delete ptr;           // releasing allocated memory
}
```

Output

```
Enter an integer value
7
Enter float value
7.7
x = 7  y = 7.7
```

4.8 DESTRUCTOR [S-23; W-18, 22]

- Destructor is a special member function which is called automatically whenever a class object is destroyed.
- The primary usage of the destructor function is to release space on the heap. Whenever, a particular object goes out of the scope of its existence, destructors will be called automatically and it takes out the allocated memory.
- Destructors are invoked by the compiler implicitly when the termination of program takes place. But it is always better to invoke the destructor explicitly by writing it into the program code.
- The diagrammatic representation of the destructor is shown in Fig. 4.7.

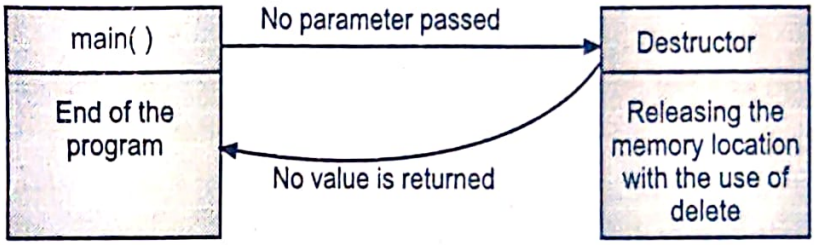

Fig. 4.7: Destructor

- A destructor is also special as it has same name as that of the class of which it is a member but with a ~ (tilde) sign prefixed to its name and which is used to destroy the objects that have been created by a constructor. It gets invoked when an object's scope is over.
- **Syntax of Destructor:**

```
class class_name
{
    .
    .
    .
    public:
        class_name(); { }
        ~class_name()    //header of the destructor
        {
            //body of the destructor.....
        }
        .
        .
        .
};
```

- **For example,**

```
class sample
{
    private:
        //data variables
        //member functions
```

```
    public:
        sample()      //constructor
        ~sample() //destructor
        //member functions
};
```

- A destructor releases the resources and memory at run-time to clean up the unused storage area.

Syntax Rules for Writing Destructor Functions:

1. A destructor name must be same as the class name in which it is declared, prefixed or preceded by a symbol tilde (~).
2. It does not have any return value.
3. It is declared as a public member function.
4. It takes no argument and therefore cannot be overloaded.
5. It cannot be declared static, constant, variable or volatile i.e. virtual.
6. It is automatically called when the object is destroyed.
7. It specifies how an object is deleted.

Program 4.12: Program for destructor.

```
#include<iostream>
using namespace std;
class sample
{
    private:
        int k, m;
    public:
        sample()
        {
            k = 10;
            m = 20;
            cout<<k<<"\t"<<m<<endl;
        }
        ~ sample() {    }   // destructor
};
void main()
{
    sample ob1, ob2, ob3;
}
```

Output:

```
10      20
10      20
10      20
```

- The objects deleted are:
  ob3
  ob2
  ob1
- If more than one object is defined, constructors are invoked from **right to left** i.e., from first object to last. But destructors are invoked in reverse order like a stack. So first created object will get deleted last.

Program 4.13: Program to declare class having data members as hrs. mins. and secs. Write constructor the values and destructor to destroy values. Accept and display data for two objects.

```
#include<iostream>
using namespace std;
class time
{
    int hr;
    int min;
    int sec;
    public:
        time (int h, int m, int s)
        {
            hr = h;
            min = m;
            sec = s;
        }
        void display()
        {
            cout<<"Time = "<<endl;
            cout<<hr<<": "<<min<<":"<<sec;
        }
        ~time ()
        {
            cout<<"Destructor invoked";
        }
    };
    int main()
        {
            time t1(3, 30, 50);
            time t2(4, 15, 35);
            t1.display();
            t2.display();
            return 0;
        }
```

Output:

```
Time =
        3: 30: 50
Time =
        4: 15: 35
Destructor invoked
Destructor invoked
```

- In above program t2 object will get destroyed first and then t1 object will destroyed because destructors are called in opposite direction of constructors.

Difference between Constructor and Destructor:

Table 4.1: Difference between Constructor and Destructor

Terms	Constructor	Destructor
Purpose	It allocates the memory to an object.	It de-allocates the memory of an object.

Declaration Syntax	`class_name (arguments if any)`
	`{`
	`//Body of Constructor...`
	`};`
		`~class_name ()`
		`{`
		`//Body of Destructor...`
		`};`

Arguments	Constructor accepts arguments.	Destructor does not accept any argument.
Calling	Constructor is called automatically, when a new object of a class is created.	Destructors are invoked/called by the compiler implicitly when the termination of program takes place.
Name	Constructor has the same name as class name.	Destructor also has the same name as class name but with a tilde (~) prefixed to its name.
Use	Constructor is used for initializing the values to the data members of the class.	Destructor is used when the object is destroyed or goes out of scope.
In numbers	There can be multiple constructors in the class.	There is always a single destructor in the class.
Overloading	Constructors can be overloaded.	Destructor cannot be overloaded.
Return type	Constructors never have a return type even void.	A destructor neither accepts any parameter nor has a return type (not even void).
Types	In a C++ program we can use different types of constructors like default constructor, parameterized constructor, copy constructor etc.	In C++ program a single destructor is used to destroy all the objects of a class created either using default, parameterized or copy constructor.

contd. ...

Example:

```
class Student
{
private:
int marks;
char grade;
public:
Student(int m, char g)
{
marks= m;
grade= g;
}
void show()
{
cout<<"Marks ="<<marks<<endl;
cout<<"Grade = "<<grade<<endl;
}
};
```

```
class test
{
private:
int a;
public:
test()
{
cout<<"object is created
:"<<endl<<endl;
}
~test()
{
cout<<"Object is
destroyed"<<endl<<endl;
}
};
```

Summary

- Constructor is a special member function which is having same name as class name which is used to automatically initialize data members to some values when an object of the class is created.
- Constructor is a special member function of the class. A constructor's task is to allocate the memory and initialize the objects of its class. It is special because its name is same as the class name.
- The constructor is invoked whenever; an object of its associated class is created. It is called constructor because it constructs the values of data members of the class.
- A constructor which does not contain any parameters is called a default constructor or also called as no parameter constructor or empty constructor. It is used to initialize the data members to some default values.
- A constructor which accepts one or more parameters is called a parameterized constructor.
- A constructor which accepts an already existing object through reference to copy the data member values is called a copy constructor. As the name implies a copy constructor is used to create a copy of an already existing object.
- Constructor overloading is as similar as function overloading where we can have multiple functions with same name but its arguments and data type should be different.
- Constructor overloading is a concept which allows class to have more than one constructors in a class i.e., in a single class we can have default constructor, parameterized constructor, and copy constructor as well.

- A constructor that accepts default arguments is called as default argument constructor.
- A constructor that allocates memory to the objects at the time of their construction using the new operator is known as dynamic constructor.
- A destructor is a special member function of a class that is executed whenever an object of its class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will have exact same name as the class prefixed with a tilde (~) sign and it can neither return a value nor can it take any parameters.
- Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Check Your Understanding

1. Constructors are used to ..... the object.
   (a) increment (b) initialize
   (c) destroy (d) both (a) & (b)
2. Destructors are used to ..... the object.
   (a) increment (b) initialize
   (c) destroy (d) modify
3. The name of the construction is similar to the name of the .....
   (a) object (b) class
   (c) data (d) none of these
4. Name of constructor is same as the .....
   (a) class (b) object
   (c) member function (d) both (a) & (b)
5. The memory is allocated to the data members of class, when ..... of the class is declared.
   (a) constructor (b) destructor
   (c) both (a) & (b) (d) object
6. Constructors, like other member functions, can be declared anywhere in the class.
   (a) True (b) False
7. Constructors do not return any value.
   (a) True (b) False
8. A constructor that accepts no parameters is known as the default constructor.
   (a) True (b) False
9. A class should have at least one constructor.
   (a) True (b) False
10. Destructor takes arguments.
    (a) True (b) False

Answers

1. (b)	2. (c)	3. (b)	4. (a)	5. (d)	6. (a)	7. (a)	8. (a)	9. (b)	10. (b)

Practice Questions

Q.I Answer the Following Questions in short:

1. What are constructor and destructor?
2. How many times will the constructor of class student are invoked for the following statement. student S, * P;
3. State the difference between constructor and destructor.
4. What is destructor? How many destructors can be defined in a single class?
5. Give any four characteristics of constructor.
6. What are the ways of constructor calling in main program? Give their syntax.
7. What is mean by copy constructors?

Q.II Answer the following Questions:

1. Write short note on: Default constructor.
2. Explain multiple constructor with example.
3. What is static data member? Where to declare them?
4. What is meant by default argument? Illustrate concept of constructor with default argument with suitable example.
5. Define parameterized constructor with its syntax and example.
6. Declare a class simple interest having data members as principle amount rate of interest, number of years. The constructor will have default value of rate of interest as 11.5%. Accept this data for two objects. Calculate and display simple interest for each object.
7. Write a program to declare class time having data members as hrs, min. sec. Write a constructor to accept data and use display function for two objects.
8. What are the rules for writing destruction function and when they are invoked?
9. Write a program to find sum of nos. between 1 to n using constructor where value of n will be passed to the constructor.
10. Write a program in C++ which prints the factorial of a given number using a constructor and destructor member function.
11. Write a program in C++ which prints the factorial of a given number using a copy constructor and a destructor member function.
12. Write a program in C++ that determines whether a given number is prime or not and print that number using default constructor and destructor member function.
13. State any four differences between constructor and destructor.

Q.III Define the term:

1. Copy constructor
2. Constructor
3. Parameterized constructor
4. Dynamic constructor
5. Overloaded constructor.

Previous Exam Questions

October 2018

1. Define destructor. [2 M]

Ans. Refer to Section 4.8.

2. Explain parameterized constructor with a suitable example. [4 M]

Ans. Refer to Section 4.3.

April 2019

1. Write a program for constructor with default arguments. [4 M]

Ans. Refer to Section 4.5.