Novelty Journals

ISSN 2394-7314

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

Implementation of Constructor and Destructor Using C++

Zobair Ullah

Sam Higginbottom Institute of Agriculture, Technology & Sciences, Allahabad, India

Abstract: **The paper is intended to introduce the concept of constructor, destructor and related terms/concepts with suitable example. The paper briefly defines and describes the necessary terms and sufficient condition to perform constructor and destructor operation in C++.**

Keywords: **OOP, class, object, inheritance, function overloading, friend function, private, public, protected, scope resolution operator, constructor, default constructor, parameterized constructor, copy constructor, constructor overloading, dynamic constructor, constructor operator overloading, destructor.**

1. INTRODUCTION

Constructors and destructors are special class methods. Constructors and destructors are special member functions of classes that are used to construct and destroy class objects. A class constructor is a special member function of a class that is executed whenever we create new objects of that class. On the other hand, an object destructor is called whenever an object goes out of scope or when the "delete" operator is called on a "pointer" to the object. The purpose of a constructor is to initialize data members and sometimes to obtain resources such as memory or a mutex or lock on a shared resource such as a hardware device. The purpose of a destructor is to clean up. It is used to free memory and to release locks or mutexes on system resources. Constructors can be very useful for setting initial values for certain member variables whereas destructors can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc. Now, let us discuss some basic/fundamental terminologies needed to define, describe and explain the concept of constructor and destructor.

2. DEFINITIONS

OOP (Object Oriented Programming):

OOP can be defined as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand.

OR

OOP is an approach for writing software in which data and behaviour are packaged together i.e encapsulated together as classes whose instances are objects.

OR

OOP is a method of implementation in which programs are organized as cooperative collections of objects. Each of which represents an instance of same class and whose class are all members of a hierarchy of classes united through the property called inheritance.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

Three important concepts of OOP are: classes, objects, and inheritance:

- Classes allow the mechanism of data abstraction for creating new data types.
- Objects are the fundamental building blocks of OOP. Each object is an instance of same class. The objects with the same data structure (attributes) and behaviour (operations) are grouped into a class.
- Inheritance allows building of new classes from the existing classes.

Now, let us try to define and understand the concept of class, objects and inheritance separately.

Class:

A class refers to a group of similar objects. A class is a way/approach/technique to bind the data describing an entity and its associated function together.

OR

Classes are user defined data types and behave like the built – is type of a programming language.

OR

Classes can be defined as a list of abstract attributes such as size, weight, cost and functions to operate on these attributes. The attributes are sometimes called data members because they hold information. The functions that operate on these data are sometimes called methods or member function.

A class is defined as

```
class<class name>              //class declaration statement
{
private:      //hidden data members / methods
protected:        //Unimportant implementation details are given here
public:         //Exposed important details here
};
```

In object oriented programming, data and its associated function are enclosed in a single entity called class. Object oriented programming hides the implementation details .Whenever there is any change in the definition of type, users interface remains unaffected generally. The above techniques can be easily understood by the following example.

For example,

```
class employee            //class declaration statement
{
private:              //visibility mode or access control specifier
int emp-code;         //Implementation details remains hidden
char *name;          // Data declaration statement
float sal;
public:             //visibility mode or access control specifier
void getdata( );       //User interface and public member function
void display( );
};
```

3. OBJECT

Object refers to a combination or collection of data and code designed to emulate a physical or abstract entity. Each object has its own identity and is distinguishable from other objects.

OR

An object can be defined as a bundle of variables and related methods.

OR

An object refers to a distinct instance of a given class that is structurally identical to all other instances of that class.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

OR

An object refers to a partitioned area of computer memory that stores data and set of operations that can access that data. Objects are the basic run time entities in an object oriented system. They may represent a person, a place, a bank account, a table of data or any item that the program has to handle. They may also represent user defined data such as vectors, time and lists.

- Objects have physical characteristics (state) and behaviour.

For example, a class car has the following characteristics and behaviour:

Characteristics:- 4 wheels, number of gears

Behaviour:- Braking , accelerating

- The physical characteristics of an object are shown/indicated through data items whereas functionality (i.e behaviour) is implemented through functions which are called methods.

- Objects are variables of the type class. Once a class has been defined, we can create any number of objects belonging to that class. Each object is associated which the data of type class with which type are created.

4. INHERITANCE

Inheritance ------ refers to the process of creating new classes from existing classes.

New classes inherit some of the properties and behavior of the existing classes.

- Inheritance is a technique of code reuse.

- Inheritance also provides possibility to extend existing classes by creating derived classes.

Function overloading ------- refers to several functions with the same name but different signature. In C++, two or more functions can be given the same name provided each has a unique signature (in either the number or data type of then arguments).

5. FRIEND FUNCTIONS

Friend class ----- refers to a class whose members have access to the private or protected members of another class.

- A friend function requires objects to be passed by reference or value.

- A friend function accesses the private data variables of another class.

- Friend functions play a very important role in operator overloading by providing the flexibility, which is derived by the member function of a class. It allows overloading of stream operators (<<or>>) for stream computation on user defined data types.

- The friend function requires all formal arguments to be specified explicitly.

- Friend functions can neither be used with a unary or binary operator.

6. TYPES OF VISIBILITY MODES OR ACCESS CONTROL SPECIFIERS USED IN C++

Private:

- The private members can be accessed only from within the class.

- If no label (private or public) is specified then by default, members are private.

- Private members of a class implement the concept of data hiding. Private members of a class are hidden from the outside world.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

Public:

- The public members of a class can be accessed from within and outside the class.
- The public members can be directly accessed by any function whether member function of the class or non member function of the class.
- The public visibility mode identifies class members both (data and functions) that constitute the public interface of the class.

Protected:

- The protected members can be accessed only by the member functions of the class.
- Protected members are the members that can be used only by member functions and friends of the class in which it is declared.

Scope resolution operator (::)

- Permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.
- Scope resolution operator (::) directs the compiler to access a global variable, instead of one defined as a local variable.
- If the scope resolution operator (::) is placed in front of the variable name then the global variable is referenced.
- When no resolution operator is placed then the local variable is referenced.
- Scope resolution operator (::) helps to identify and specify the context to which and identifier refers.
- The scope resolution operator (::) is the highest precedence operator in the C++ language.

The following program illustrates the use of scope resolution operator (::).

```
#include<iostream.h>
#include<conio.h>
int a=10;          //Global variable declaration statement
void main()
{
int a=20;           //Local variable declaration statement
cout<<"\n"<<"Local a="<<a;
cout<<"\n"<<"Global a="<<::a;      // accessing a global variable
getch();
}
```

Output:

Local a=20

Global a=10

- Scope resolution operator (::) comes in two forms unary and binary.
- The scope resolution operator (::) in C++ is used to define the already declared member functions (in the header file with the .h extension) of a particular class.
- To distinguish between the normal functions and the member functions of the class, one needs to use the scope resolution operator (::) in between the class name and the member function name i.e. class-name :: function-name(); e.g. apple ::getdata(); where apple is the class-name and getdata() is a member function of the class apple.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

- If the resolution operator is placed between the class name and the data member belonging to the class then the data name belonging to the particular class is referenced.

- We can also use the class scope operator to qualify class names or class member names. If a class member name is hidden, we can use it by qualifying it with its class name and the class scope operator.

The following program illustrates the use of scope resolution operator (::).

```
#include<iostream.h>
#include<conio.h>
class apple
{
public:
static int a;
};
int apple :: a=10;     //defines static data member
void main()
{
int apple=0;           //hides class type apple
cout<<"\n"<<apple::a<<endl;      //use static member of class apple
getch();
}
```

Output: 10

In this case, the declaration of the variable apple hides the class type apple, but we can still use the static class member 'a' by qualifying it with the class type "apple" and the scope resolution operaor.

- The other use of the resolution operator is to resolve the scope of a variable when the same identifier is used to represent the following: 1.global variable, 2. A local variable and 3. Member of one or more classes.

7. CONSTRUCTOR

Constructor ------ refers to a member function with the same name as its class name.

OR

Constructor refers to a special member function of a class whose main operation is to allocate the required memory and initialize the objects of its class.

Types of constructor

Default constructor ------ refers to a constructor that accepts no parameters. If no constructor is defined then the compiler supplies a default constructor.

For example,

```
class apple                 //class declaration statement
{
private:              //visibility mode or access control specifier
int a;                 //private data member
public:                     //visibility mode or access control specifier
apple();            //default constructor declaration statement
void display();       //public member function
};
apple:: apple()              //default constructor outside the class
{
a=10;
}
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

Parameterized constructor ----- refers to a constructor that receives arguments/parameters, is called parameterized constructor.

For example,

```
class apple                 //class declaration statement
{
private:              //visibility mode or access control specifier
int a;                 //private data member
public:                    //visibility mode or access control specifier
apple(int r);           //parametric constructor declaration statement
void display();       //public member function
};
 apple::apple (int r)         //parametric constructor outside the class
{
a=r;
}
```

Copy constructor ------- refers to a constructor that initializes an object using values of another object passed to it as parameter. It creates the copy of the object passed.

For example,

```
class apple                 //class declaration statement
{
private:              //visibility mode or access control specifier
int a;                 //private data member
public:                    //visibility mode or access control specifier
apple(apple  &t);           //copy constructor declaration statement
void display();       //public member function
};
apple :: apple(apple  &t)      //copy constructor outside the class
{
a=t.a;
}
```

Constructor overloading ------- refers to overloaded constructors that have same name (name of the class) but different numbers of argument passed.

- Depending upon the number and type of argument passed, specific constructor is called.
- Arguments to the constructor are passed while creating object.

For example,

```
class apple                 //class declaration statement
{
private:          //visibility mode or access control specifier
int a;                 //private data member
float b;
double c;
public:            //visibility mode or access control specifier
apple (int p)              //parametric constructor declaration statement
{
a=p;                    //accessing and initializing private data member in public
}
apple (float q)       //parametric constructor declaration statement
{
b=q;
}
apple (double r)         //parametric constructor declaration statement
```

```
{
c=r;
}
```

Dynamic constructor:

Dynamic initialization through constructors:

- Dynamic constructor is used to allocate the memory to the objects at the run time. Memory is allocated at runtime with the help of 'new' operator.
- By using dynamic constructors, we can dynamically initialize the objects.
- Objects data members can be dynamically initialized during runtime, even after their creation.
- The advantage of this feature is that, it supports different initialization formats using overloaded constructors.
- It provides flexibility of using different forms of data at runtime depending upon the users used.

Some special characteristics of the constructor function:

- Constructor should be declared in the public section.
- Constructors are invoked automatically when the objects are created or class is instantiated.
- Constructors do not have return type, not even void and therefore, cannot return value.
- Constructor is generally used to initialize object member parameters and allocate the necessary resources (memory) to the object members.
- The constructor of a class is the first member function to be executed automatically when an object of the class is created.
- The constructor is executed every time an object of that class is defined.
- Constructors cannot be virtual.
- Constructors cannot be declared with the keyword virtual.
- Constructors are called automatically by the compiler when defining class objects.
- Constructors cannot be called explicitly as if they were regular member function.
- We use a constructor, where we dynamically allocate some memory.
- Constructors can be very useful for setting initial values for certain member variables.

8. SYNTAX TO DECLARE A CONSTRUCTOR USING OUTSIDE THE CLASS DEFINITION METHOD

```
class <class name>              //class declaration statement
{
private:              //visibility mode or access control specifier
Data type variable- name ;          //private data and member function
Function declaration;
public:                         //visibility mode or access control specifier
<class name> ( );                   //Constructor  declaration statement
Data type function-name();
};
class-name :: class-name()       //constructor declaration outside the class
{
Statement(s);
}
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
Data type class-name :: function-name()
{
Statement(s);
}
void main()                    //Declaration of main function
{
<class name> ob1;              // object declaration statement
<class name>( );               // calling constructor
ob1.function-name ();          // calling function using object
getch();
}
```

Default Constructor:

1. Program to display "apple" using constructor and outside the class definition method.

```
#include<iostream.h>
#include<conio.h>
class apple             //class declaration statement
{
public:          //visibility mode or access control specifier
apple();              //default constructor declaration statement
};
apple :: apple()       //accessing class/public member function outside the class
{
cout<<"\n"<<"Apple";
}
void main()         //declaration of main function
{
apple ob1;        //object declaration statement
getch();          //freeze the screen until any key is pressed
}
```

Output:

Apple

2. Program to display a number using default constructor and outside the class definition method:

```
#include<iostream.h>
#include<conio.h>
class apple                //class declaration statement
{
private:             //visibility mode or access control specifier
int a;               //private data member
public:                 //visibility mode or access control specifier
apple();           //default constructor declaration statement
void display();      //public member function
};
apple :: apple()     //accessing class/public member function outside the class
{
a=10;              //accessing and initializing private data member outside the class
}
void apple :: display()   //accessing public member function outside the class
{
cout<<"\n"<<"a="<<a;
}
void main()         //declaration of main function
{
apple ob1;        //object declaration statement
getch();          //freeze the screen until any key is pressed
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
}
```

Output:

a=10

3. Program to display numbers using default constructor outside the class:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a;                 //private data member
float b;
public:                     //visibility mode or access control specifier
apple();       //default constructor declaration statement
void display();
};
apple:: apple():a(10),b(10.5)
{
}
void apple::display()
{
cout<<"\n"<<"a"<<a;
cout<<"\n"<<"b"<<b;
}
void main()           //declaration of main function
{
apple ob1;               //object declaration statement
ob1.display();        //calling member function using object
getch();
}
```

Output:

a=10
b=10.5

Parametric constructor:

4. Program to display numbers using parametric constructor outside the class:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a;                 //private data member
float b;
public:                     //visibility mode or access control specifier
apple(int p, float q);          //parametric constructor declaration statement
void display();
};
apple:: apple(int p, float q):a(p),b(q)      //initializing parametric constructor outside
{
}
void apple::display()
{
cout<<"\n"<<"a"<<a;
cout<<"\n"<<"b"<<b;
}
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
void main()          //declaration of main function
{
apple ob1(10,10.5);              //implicit object declaration statement
ob1.display();          //calling member function using object
getch();
}
```

Output:

a=10

b=10.5

5. Program to display a number using parameterized constructor:

```
#include<iostream.h>
#include<conio.h>
class apple                  //class declaration statement
{
private:              //visibility mode or access control specifier
int a;
public:
apple(int p);                // Declaration of parametric constructor
void display()
{
cout<<"\n"<< "Number :"<< a;
}
};
apple   ::  apple (int p)          //parametric constructor  declaration outside the class
{
a=p;
}
void main()
{
apple ob1(10);          //Constructor called implicitly
ob1.display();
apple ob2= apple (20);          //Constructor called explicitly
ob2.display();              // calling function using object
getch();
}
```

Output:

Number: 10

Number: 20

6. Program to display names using parameterized constructor:

```
#include<iostream.h>
#include<conio.h>
class apple
{
private:              //visibility mode
char *str1;
public:
apple (char *str2);                  //Parameterized Constructor declared
void display()
{
cout<<"\n"<< "Name :"<< str1;
}
};
```

```
apple :: apple (char *str2)
{
str1=str2;
}
void main()
{
cout<<"\n"<< "----------Name1----------";
apple ob1("John");            //Constructor called implicitly
ob1.display();
cout<<"\n"<< "----------Name2----------";
apple ob2=P("Joseph");        //Constructor called explicitly
ob2.display();
getch();
}
```

Output:

----------Name1----------

Name: John

----------Name2----------

Name: Joseph

C***onstructor overloading:***

7. Program to overload constructors outside the class:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a,b,c,d;              //private data member
public:
apple ();             //default constructor declaration statement
apple (int p);
apple (int q, int r);      //declaration of parametric constructor
void display1();
void display2() ;
void display3();
};
 apple :: apple()      //accessing class/public member function outside the class
{
a=10;                 //accessing and initializing private data member outside the class
}
apple :: apple(int p)
{
b=p;
}
apple :: apple(int q, int r)
{
c=q;
d=r;
}
void apple :: display1()    //accessing public member function outside the class
{
cout<<"\n"<<"a="<<a;
}
void apple :: display2()    //accessing public member function outside the class
```

```
{
cout<<"\n"<<"b="<<b;
}
void apple :: display3()    //accessing public member function outside the class
{
cout<<"\n"<<"c="<<c;
cout<<"\n"<<"d="<<d;
}
void main()            //declaration of main function
{
apple ob1;             //object declaration statement
ob1.display1();
apple ob2(20);
ob2.display2();
apple ob3(100,200);
ob3.display3();
getch();            //freeze the screen until any key is pressed
}
```

Output:

a=10

b=20

c=100

d=200

Constructor operator overloading:

8. Program to illustrate operator overloading using constructors outside the class.

```
#include<iostream.h>                  //function prototype
#include<conio.h>
class apple                    //class declaration statement
{
private:                //visibility mode or access control specifier
int a;
public:
apple();               //default constructor
void display();
void operator++();          //increment operator function
};
apple::apple()        //accessing class/public member function outside the class
{
a=0;
}
void apple::display()
{
cout<<"\n"<<"a="<<a;
}
void apple::operator++()
{
a=a+1;
}
void main()
{
apple ob1;             //object declaration statement
ob1.display();
++ob1;
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
ob1.display();
getch();
}
```

Output:

a=0

a=1

9. Program to illustrate operator overloading using parametric constructors outside the class:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a;
public:
apple(int b);              //parametric constructor
void display();
void operator--();
};
apple::apple(int b)              //parametric constructor outside the class
{
a=b;
}
void apple::display()
{
cout<<"\n"<<"a="<<a;
}
void apple::operator--()
{
a=a-1;
}
void main()
{
apple ob1(11);       //object declaration statement
ob1.display();
--ob1;
ob1.display();
getch();
}
a=11
a=10
```

Copy constructor:

10. Program to display numbers using a parametric constructor and a copy constructor using outside the class definition method:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a;              //private data member
public:
apple(int p);              //parametric constructor declaration statement
apple( apple &q);              //declaration of copy constructor
void display();
```

```
};
apple :: apple (int p)              //parametric constructor declaration outside the class
{
a=p;
}
apple ::apple (apple  &q)        //copy constructor declaration statement
{
a=q.a;
}
void apple:: display()                    //public member function
{
cout<<"\n"<<"a="<<a;
}
void main()              //declaration of main function
{
apple ob1(10);                  //object declaration statement
ob1.display();
apple ob2(ob1);
ob2.display();
getch();
}
```

Output:

a=10

a=10

9. SYNTAX TO DECLARE A CONSTRUCTOR USING INSIDE THE CLASS DEFINITION METHOD

```
class <class name>              //class declaration statement
{
private:              //visibility mode or access control specifier
Data type variable- name ;          //private data and member function
Function declaration;
public:                        //visibility mode or access control specifier
<class name> ( )             //Constructor  declaration inside the class statement
{
Statement(s);
}
};
void main()                    //Declaration of main function
{
<class name> ob1;                 // object declaration statement
<class name>( );                       // calling constructor
ob1.function-name ();               // calling function using object
getch();
}
```

The following programs demonstrate/ illustrate the working of a constructor.

Default constructor:

1. Program to display "Apple" (any text) using a default constructor and inside the class definition method:

```
#include<iostream.h>
#include<conio.h>
class apple       //class declaration statement
{
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
public:              //visibility mode or access control specifier
apple()        //default constructor statement
{
cout<<"\n"<<"Apple";
}
};
void main()          //declaration of main function
{
apple ob1;         //object declaration statement
getch();             //freeze the screen until any key is pressed
}
```

Output:

Apple

2. Program to display a number using default constructor and inside the class definition method:

```
#include<iostream.h>        //function prototype
#include<conio.h>
class apple                 //class declaration statement
{
private:          //visibility mode or access control specifier
int a;                 //private data member
public:          //visibility mode or access control specifier
apple()            //default constructor declaration statement
{
a=10;                    //accessing and initializing private data member in public
}
void display()                  //public member function
{
cout<<"\n"<<"a="<<a;
}
};
void main()             //declaration of main function
{
apple ob1;         //object declaration statement
ob1.display();       //calling member function of the class
getch();                 //freeze the screen until any key is pressed
}
```

Output:

a=10

3. Program to display a number using default constructor:

```
#include<iostream.h>
#include<conio.h>
class apple                 //class declaration statement
{
private:              //visibility mode or access control specifier
int a;
public:
apple()                   //Constructor  declaration inside the class statement
{
cout<<"\n"<<"Enter a number:";
cin>>a;
cout<<"\n"<< "Number :"<<a;
}
};
```

```
void main()                    //Declaration of main function
{
apple();              // calling constructor
getch();
}
```

Output :

Enter a number : 5

Number :5

OR

```
#include<iostream.h>
#include<conio.h>
class apple                 //class declaration statement
{
private:                //visibility mode or access control specifier
int a;
public:
apple()                          //default constructor declaration statement
{
cout<<"\n"<<"Enter a number:";
cin>>a;
}
void display()              //declaration of  member function of the class
{
cout<<"\n"<< "Number :"<< a;
}
};
apple ob1;                  //global declaration of object
void main()
{                    //The compiler automatically initializes constructor as it is created
ob1.display();
getch();
}
```

Output:

Enter a number: 5

Number: 5

4. Program to find product of two numbers using default constructor:

```
#include<iostream.h>
#include<conio.h>
class apple                 //class declaration statement
{
private:              //visibility mode or access control specifier
int a,b,pro;
void get()
{
cout<<"\n"<<"Enter two numbers:";
cin>>a>>b;
}
public:
apple ()                         // default constructor declaration statement
{
get();                  //accessing private members inside the class
pro=a*b;
```

```
}
void display()
{
cout<<"\n"<< "Product :"<< pro;
}
};
apple ob1;            //global declaration of object
void main()
{                     //The compiler automatically initializes constructor as it is created
ob1.display();
getch();
}
```

Output:

Enter two numbers: 5

6

Product: 30

5. Program to display numbers using default constructor and inside the class:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a;                //private data member
float b;
public:                       //visibility mode or access control specifier
apple():a(10),b(10.5)         //default constructor initialization statement
{
}
void display()        //public member function
{
cout<<"\n"<<"a"<<a;
cout<<"\n"<<"b"<<b;
}
};
void main()           //declaration of main function
{
apple ob1;                    //object declaration statement
ob1.display();        //calling member function using object
getch();
}
```

Output:

a=10

b=10.5

Parametric constructor:

Parameterized constructor -------- refers to the constructor that can take arguments.

Syntax of parameterized constructor:

```
class <classname>
{
private:
Data declaration ;
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
Function declaration;
public:
<classname> (data type variable1, data type variable2,-------);  //Parameterized
{                                                                  Constructor
Statement(s);
}
};
<classname>::<classname> (data type variable1, data type variable2,-------)
{
Statement(s);
}
void main()
{
<classname> ob1(value1,value2,---------);       // Constructor called implicitly
<classname> ob2=<classname> (value1,value2,-----);    //Constructor called explicitly
getch();
}
```

6. Program to display a number using a parametric constructor inside the class:

```
#include<iostream.h>        //function prototype
#include<conio.h>
class apple                 //class declaration statement
{
private:          //visibility mode or access control specifier
int a;                 //private data member
public:
apple(int p)      //declaration of parametric function
{
a=p;                  //member initialization in constructor
}
void display()                  //public member function
{
cout<<"\n"<<"a="<<a;
}
};
void main()
{
apple ob1(10);          //object declaration statement
ob1.display();
getch();
}
```

Output:

a=10

Constructor overloading:

7. Program to overload constructors using a parameter having different data types:

```
#include<iostream.h>        //function prototype
#include<conio.h>
class apple                 //class declaration statement
{
private:          //visibility mode or access control specifier
int a;                 //private data member
float b;
double c;
public:           //visibility mode or access control specifier
apple (int p)             //parametric constructor declaration statement
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
{
a=p;                    //accessing and initializing private data member in public
}
apple (float q)       //parametric constructor declaration statement
{
b=q;
}
apple (double r)        //parametric constructor declaration statement
{
c=r;                  //member initialization in constructor
}
void display1()                 //public member function
{
cout<<"\n"<<"a="<<a;
}
void display2()                 //public member function
{
cout<<"\n"<<"b="<<b;
}
void display3()                 //public member function
{
cout<<"\n"<<"c="<<c;
}
};
void main()             //declaration of main function
{
apple ob1(10);          //object declaration statement
apple ob2=3.5f;
apple ob3(55.6755f);
ob1.display1();       //calling member function of the class
ob2.display2();
ob3.display3();
getch();                  //freeze the screen until any key is pressed
}
```

Output:

a=10

b=3.5

c=1.943138e-77

8. Program to illustrate constructor overloading using different number of parameters and different data types:

```
#include<iostream.h>       //function prototype
#include<conio.h>
class apple                 //class declaration statement
{
private:              //visibility mode or access control specifier
int a,b,c,d;
float p,q,r;
public:
apple()             //default constructor declaration statement
{
a=10;                   //accessing and initializing private data member in public
}
apple (int m)       //parametric constructor declaration statement
{
b=m;
```

```
}
apple(int x, int y)          //parametric constructor declaration statement
{
c=x;
d=y;
}
apple(float g)               //parametric constructor declaration statement
{
p=g;
}
apple(float j, float k)           //parametric constructor declaration statement
{
q=j;
r=k;
}
void display1()                    //public member function
{
cout<<"\n"<<"a="<<a;
}
void display2()                    //public member function
{
cout<<"\n"<<"b="<<b;
}
void display3()                    //public member function
{
cout<<"\n"<<"c="<<c;
cout<<"\n"<<"d="<<d;
}
void display4()                    //public member function
{
cout<<"\n"<<"p="<<p;
}
void display5()                    //public member function
{
cout<<"\n"<<"q="<<q;
cout<<"\n"<<"r="<<r;
}
};
void main()                //declaration of main function
{
apple ob1;
ob1.display1();
apple ob2(20);             //object declaration statement
ob2.display2();
apple ob3(100,200);        //object declaration statement
ob3.display3();
apple ob4(10.20f);              //object declaration statement
ob4.display4();
apple ob5(10.55,20.55f);        //object declaration statement
ob5.display();
getch();
}
```

Output:

a=10

b=20

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

c=100

d=200

p=10.2

q=10.55

r=20.549999

9. Program to illustrate operator overloading using constructor inside the class:

```
#include<iostream.h>                    //function prototype
#include<conio.h>
class apple                    //class declaration statement
{
private:              //visibility mode or access control specifier
int a;
public:
apple()         //default constructor statement
{
a=0;
}
void display()
{
cout<<"\n"<<"a="<<a;
}
void operator++()
{
a=a+1;
}
};
void main()
{
apple ob1;                    //object declaration statement
ob1.display();
++ob1;
ob1.display();
getch();
}
```

Output:

a=0

a=1

10. Program to illustrate the implementation of operator overloading using parametric constructor and inside the class method:

```
#include<iostream.h>                  //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:                //visibility mode or access control specifier
int a;
public:
apple(int b)              //parametric constructor declaration statement
{
a=b;
}
void display()
```

```
{
cout<<"\n"<<"a="<<a;
}
void operator--()
{
a=a-1;
}
};
void main()
{
clrscr();
apple ob1(11);            //object declaration statement
ob1.display();
--ob1;
ob1.display();
getch();
}
```

Output:

a=11

a=10

Copy constructor:

11. Program to display numbers using a parametric constructor and a copy constructor:

```
#include<iostream.h>        //function prototype
#include<conio.h>
class apple                  //class declaration statement
{
private:              //visibility mode or access control specifier
int a;
public:
apple (int p)        //parametric constructor declaration statement
{
a=p;
}
apple (apple  &q)        //copy constructor declaration statement
{
a=q.a;               //member initialization in copy constructor
}
void display()                  //public member function
{
cout<<"\n"<<"a="<<a;
}
};
void main()               //declaration of main function
{
apple ob1(20);
apple ob2(ob1);           //object declaration statement
ob1.display();
ob2.display();                  //calling member function of the class
getch();
}
```

Output:

a=20

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

a=20

12. Program to display numbers using a default constructor and a copy constructor:

```
#include<iostream.h>          //function prototype
#include<conio.h>
class apple                   //class declaration statement
{
private:              //visibility mode or access control specifier
int a;               //private data member
public:
apple ()              //default constructor declaration statement
{
a=10;
}
apple (apple  &q)        //copy constructor declaration statement
{
a=q.a;
}
void display()                  //public member function
{
cout<<"\n"<<"a="<<a;
}
};
void main()               //declaration of main function
{
apple ob1;                  //object declaration statement
apple ob2(ob1);                  //copying object
ob1.display();
ob2.display();                  //calling member function of the class
getch();
}
```

Output:

a=10

a=10

13. Program to display numbers using different constructors:

```
#include<iostream.h>
#include<conio.h>
class apple                  //class declaration statement
{
private:
int a,b;
public:
apple ()         // Default Constructor1
{
a=10;                //member initialization in default constructor
b=20;
}
apple (int p, int q)       //Parameterized Constructor2
{
a=p;                          //member initialization in parameterized constructor
b=q;
}
apple (apple &x)          //Copy Constructor3
{
```

```
a=x.a;
b=x.b;                  //member initialization in copy constructor
}
void display()
{
cout<<"\n"<< "Number1 :"<< a;
cout<<"\n"<< "Number2 :"<< b;
}
};
void main()
{
cout<<"\n"<< "-----------Result1--------------";
apple ob1;
ob1.display();
cout<<"\n"<< "-----------Result2--------------";
apple ob2(50,100);
ob2.display();
cout<<"\n"<< "-----------Result 3--------------";
apple ob3(ob2);                 //copies the values of ob2 into ob3
ob3.display();
getch();
}
```

Output:

```
-----------Result1--------------

Number1 : 10

Number2 : 20

-----------Result2--------------

Number1 : 50

Number2 : 100

-----------Result3--------------

Number1 : 50

Number2 : 100
```

Constructor and inheritance:

14. Program to display numbers using default constructor and inheritance:

```
#include<iostream.h>
#include<conio.h>
class apple                     //base class declaration statement
{
protected:              //visibility mode
int a;
public:
apple ()        // Default Constructor declaration statement
{
a=10;                   //member initialization in constructor
}
void display1()                 //public member function
{
cout<<"\n"<<"a="<<a;
}
```

```
};
class apple1: public apple          //the base class apple is publicly inherited by the
{                                     derived class apple1
protected:
int b;
public:
apple1(): apple()
{
b=20;
}
void display2()                     //public member function
{
cout<<"\n"<<"b="<<b;
}
};
void main()               //declaration of main function
{
apple1 ob1;                    //object declaration statement
ob1.display1();
ob1.display2();                    //calling member function of the class
getch();
}
```

Output:

a=10

b=20

Here, Class apple1: public apple derives a new class apple1 known as derived class from the base class apple. The base class apple is publicly inherited by the derived class apple1.

15. Program that illustrates how parametric constructors are invoked in derived class:

```
#include<iostream.h>              //function prototype
#include<conio.h>
class apple1                 //class declaration statement
{
private:              //visibility mode or access control specifier
int x;
public:
apple1(int i)             //parametric constructor apple1 initialized
{
x=i;
}
void display1()
{
cout<<"\n"<<"x = "<<x;
}
};
class apple2               //class declaration statement
{
private:
float y;
public:
apple2(float j)        //parametric constructor appl2 initialized
{
y=j;
}
void display2()
```

```
{
cout<<"\n"<<"y="<<y;
}
};
class apple3 : public apple2, public apple1
{
private:
int n,m;
public:
apple3(int a, float b, int c, int d):apple1(a),apple2(b)
{
m=c;                    //parametric constructor apple3 initialized
n=d;
}
void display3()
{
cout<<"\n"<<"m="<<m;
cout<<"\n"<<"n="<<n;
}
};
void main()
{
apple3 ob1(10, 7.75, 50, 100);          //object declaration statement
cout<<"\n";
ob1.display1();
ob1.display2();
ob1.display3();
getch();
}
```

Output:

x=10

y=7.75

m=50

n=100

Dynamic constructor:

16. Program to display numbers using dynamic constructor and inside the class method:

```
#include<iostream.h>
#include<conio.h>
class apple                     //class declaration statement
{
private:          //visibility mode or access control specifier
int *ptr;
public:
apple()               //default constructor
{
ptr=new int;
*ptr=100;
}
apple(int b)              //parametric constructor
{
ptr=new int;
*ptr=b;
}
```

```
void display()
{
cout<<"\n"<<*ptr;
}
};
void main()
{
apple ob1;            //object declaration statement
apple ob2(10);
ob1.display();
ob2.display();
getch();
}
```

Output:

100

10

17. Program to concatenate two strings using dynamic constructor and inside the class method:

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class apple                    //class declaration statement
{
private:                 //visibility mode or access control specifier
char *name;
int a;
public:                //visibility mode or access control specifier
apple()                  //default constructor
{
a=0;
name= new char[a+1];
}
apple(char *s)
{
a=strlen(s);
name=new char[a+1];
strcpy(name, s);
}
void display()
{
cout<<"\n"<<name;
}
Void join(apple &p, apple &q)
{
a=p.a+q.a;
delete name;
name=new char[a+1];
strcpy(name, p.name);
strcat(name, q.name);
}
};
Void main()
{
apple ob1("c++"),ob2("program"),ob3;        //object declaration statement
ob3.join(ob1,ob2);
ob1.display();
```

```
ob2.display();
ob3.display();
getch();
}
```

Output:

C++

Program

C++ program

Friend class:

18. Program to find area of a square using friend class and constructor:

```
#include<iostream.h>
#include<conio.h>
class square;
class rectangle          // class declaration statement
{
int b,h;
public:
void area()
{
cout<<"\n"<<"Area="<<b*h;
}
void convert(square a);
};
class square                    //class declaration statement
{
friend class rectangle;              //friend class declaration
private:
int side;
public:
square (int a):side(a)          //initialising parametric function
{
}
};
void rectangle::convert(square a)
{
b=a.side;
h=a.side;
}
void main()
{
rectangle ob1;                  //object declaration statement
square sqr(4);
ob1.convert(sqr);
ob1.area();
getch();
}
```

Output:

Area=16

9. DESTRUCTOR

Destructor ------ refers to a member function having the character ~(tilde) followed by a function name, which is same as the class-name (i.e ~class-name()) and is invoked automatically when class object goes out of scope (i.e the object no longer needed).

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

Syntax to declare a constructor/destructor inside the class definition method:

```
class <class name>              //class declaration statement
{
private:              //visibility mode or access control specifier
Data type variable- name ;          //private data and member function
Function declaration;
public:                        //visibility mode or access control specifier
<class name> ( )              //Constructor  declaration inside the class statement
{
Statement(s);
}
~<class name> ( )              //Destructor declaration inside the class statement
{
Statement(s);
}
};
void main()                    //Declaration of main function
{
<class name> ob1;                 // object declaration statement
<class name>( );                       // calling constructor
ob1.function-name ();              // calling function using object
getch();
}
```

Syntax to declare a constructor/destructor outside the class definition method:

```
class <class name>              //class declaration statement
{
private:              //visibility mode or access control specifier
Data type variable- name ;          //private data and member function
Function declaration;
public:                        //visibility mode or access control specifier
<class name> ( );                  //Constructor  declaration statement
Data type function-name();
~<class name> ( );                    //Destructor declaration statement
};
class-name :: class-name()       //Constructor declaration outside the class
{
Statement(s);
}
class-name :: ~class-name()       //Destructor declaration outside the class
{
Statement(s);
}
Data type class-name :: function-name()
{
Statement(s);
}
void main()                    //Declaration of main function
{
<class name> ob1;                 // object declaration statement
<class name>( );                       // calling constructor
ob1.function-name ();              // calling function using object
getch();
}
```

Some special characteristics of the destructor function:

- A destructor function will have exact same name as the class prefixed with a tilde (~) sign.

- The destructors are called when a class object goes out or scope.
- 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.
- The destructor destroys the object when it is no longer required by releasing all the resources allocated to it. This operation is also called clean up.
- Destructor is generally used to reclaim all the resources allocated to the object.
- A class can have at the most one destructor.
- Destructor can neither return a value nor can it take any parameters.
- The destructor does not take any arguments. This is because there is only one way to destroy an object.
- There can be only one destructor in each class.
- Destructor cannot be overloaded because destructors cannot have arguments.
- Destructors can be virtual.
- Destructor can be very useful for releasing resources before coming out of the program like closing file, releasing memories etc.
- Destructors are usually/commonly used to "clean up" when an object is no longer necessary.

1. The following programs illustrate the implementation of a destructor using inside the class method:

```
#include<iostream.h>
#include<conio.h>
class apple                    //class declaration statement
{
private:
int a;
public:
apple ()         // Default Constructor1
{
cout<<"\n"<<"object created";
}
apple (int b)       //Parameterized Constructor2
{
a=b;                           //member initialization in parameterized constructor
}
~apple()           //destructor declaration statement
{
cout<<"\n"<<"object destroyed";
}
void display()
{
cout<<"\n"<<"a="<<a;
}
};
void main()              //declaration of main function
{
apple ob1;                  //object declaration statement
apple ob2(10);                  //accessing parametric function
ob2.display();                  //calling member function of the class
getch();
}
```

Output:

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

object created

a=10

object destroyed

object created

a=10

2. The following programs illustrate the implementation of a destructor using outside the class method:

```
#include<iostream.h>
#include<conio.h>
class apple
{
private:          //visibility mode
int a;              //private data member
public:
apple();
apple(int b);     //parametric constructor
~apple();             //destructor declaration statement
void display();
};
apple::apple ()         // Default Constructor1
{
cout<<"\n"<<"object created";
}
apple ::apple (int b)       //Parameterized Constructor2
{
a=b;                          //member initialization in parameterized constructor
}
apple::~apple()           //destructor declaration statement
{
cout<<"\n"<<"object destroyed";
}
void apple:: display()
{
cout<<"\n"<<"a="<<a;
}
void main()             //declaration of main function
{
apple ob1;                  //object declaration statement
apple ob2(10);                  //accessing parametric function
ob2.display();                  //calling member function of the class
getch();
}
```

Output:

object created

a=10

object destroyed

object destroyed

object created

a=10

3. Program to find area of a rectangle using dynamic constructor and destructor:

```
#include<iostream.h>
```

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

```
#include<conio.h>
class apple
{
private:          //visibility mode
int *b,*h;              //private data member
public:
apple(int, int);
~apple();
void area()
{
cout<<"\n"<<"length="<<*b;
cout<<"\n"<<"breadth="<<*h;
cout<<"\n"<<"Area of a rectangle="<< (*b)*(*h);
}
};
apple::apple(int x, int y)          //parametric function outside the class
{
b= new int;         //dynamically allocate some memory
h= new int;
*b=x;
*h=y;
}
apple::~apple()          //declaration of destructor outside the class
{
delete b;              // deletes the allocated memory
delete h;
cout<<"\n"<<"object destroyed";
}
void main()
{
apple  ob1(20,20);          //object declaration statement
ob1.area();
getch();
}
```

Output:

Length= 20

Breadth=20

Area of a rectangle=400

Object destroyed

Length= 20

Breadth=20

Area of a rectangle=400

Some common features of constructor and destructor:

- Constructors and destructors are declared within a class declaration.
- A constructor or a destructor can be defined inline or external to the class declaration.
- Constructors and destructors cannot have return type (not even void).
- Pointers and references cannot be used on constructors and destructors (it is not possible to get there address).
- Constructors and destructors cannot be declared static, constant or volatile.
- In the destructor we delete the memory, we dynamically allocated in the constructor.

International Journal of Novel Research in Computer Science and Software Engineering
Vol. 3, Issue 2, pp: (1-33), Month: May - August 2016, Available at: www.noveltyjournals.com

Differences between constructors and destructors:

- Arguments cannot be passed to destructors.
- Only one destructor can be declared for a given class.
- Destructors cannot be overloaded.
- Destructors can be virtual.

10. CONCLUSION

The paper is intended to define, describe, understand and discuss the concept of constructor and destructor using C++, giving suitable examples. Emphasis has been given to implement constructor and destructor using different methods available in C++. The paper also discusses the key features and characteristics of constructor and destructor.

ACKNOWLEDGEMENT

I express my deep gratitude to my students and people around me who most often raise some computer programming related issues/problems that really inspired and motivated me to undertake this research work and make the things simple and precise. In the end, I would like to thank the great almighty who has given wisdom, strength and knowledge to visualise and explore things from grass root level and put on papers for the benefit of mankind.

REFERENCES

[1] Computer Science a structured approach using C++ by Behrouz A. Forouzan and Richard F.Gilberg.

[2] Henricson, Mats; Nyquist, Erik(1997), Industrial strength C++. Practice Hall ISBN 0-13-120965-5.

[3] Scott Meyers: Effective C++, Addison – Wesley, ISBN -0-321-33487-6

[4] Erickson, Jon (2008). Hacking the art of exploitation No starch press. ISBN 1-59327-144-1.

[5] C++ standard, ISO/IEC 14882:1998,12.1.5

[6] C++ standard, ISO/IEC 14882:2003,12.1.5

[7] ISO/IEC (2003) ISO/IEC 2003(E). Programming languages C++.

[8] cplusplus.com tutorial lesson 5.2

[9] cplusplus.com tutorial lesson 2.5

[10] Default constructors – cppreference.com

[11] Constructors and destructors from PHP online documentation.