CONSTRUCTORS AND DESTRUCTORS

CONSTRUCTORS :

A member function with the same as its class is called Constructor and it is used to initialize the object of that class with a legal initial value.

Example :

```
class Student
{
  int rollno;
float marks;
public:
student( )              //Constructor
{
rollno=0;
marks=0.0;
}
                  //other public members
};
```

TYPES OF CONSRUCTORS:

1. Default Constructor:

A constructor that accepts no parameter is called the Default Constructor. If you don't declare a constructor or a destructor, the compiler makes one for you. The default constructor and destructor take no arguments and do nothing.

2. Parameterized Constructors:

A constructor that accepts parameters for its invocation is known as parameterized Constructors , also called as Regular Constructors.

DESTRUCTORS:

- A destructor is also a member function whose name is the same as the class name but is preceded by tilde(“~”).It is automatically by the compiler when an object is destroyed. Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed.
- A destructor is called for a class object when that object passes out of scope or is explicitly deleted.

Example :

```
class TEST
{   int Regno,Max,Min,Score;
Public:
 TEST( )                              // Default Constructor
 {                          }
 TEST (int Pregno,int Pscore)         //  Parameterized Constructor
{
Regno = Pregno ;Max=100;Max=100;Min=40;Score=Pscore;
}
~ TEST ( )                            // Destructor
{ Cout<<”TEST Over”<<endl;}
};
```

The following points apply to constructors and destructors:

- Constructors and destructors do not have return type, not even void nor can they return values.
- References and pointers cannot be used on constructors and destructors because their addresses cannot be taken.

- Constructors cannot be declared with the keyword virtual.
- Constructors and destructors cannot be declared static, const, or volatile.
- Unions cannot contain class objects that have constructors or destructors.
- The compiler automatically calls constructors when defining class objects and calls destructors when class objects go out of scope.
- Derived classes do not inherit constructors or destructors from their base classes, but they do call the constructor and destructor of base classes.
- The default destructor calls the destructors of the base class and members of the derived class.
- The destructors of base classes and members are called in the reverse order of the completion of their constructor:
- The destructor for a class object is called before destructors for members and bases are called.

Copy Constructor

- **A copy constructor is a special constructor in the C++ programming language used to create a new object as a copy of an existing object**.
- A copy constructor is a constructor of the form **classname(classname &).**The compiler will use the copy constructors whenever you initialize an instance using values of another instance of the same type.
- Copying of objects is achieved by the use of a copy constructor and a assignment operator.

Example :

```
class Sample{    int i, j;}
public:
Sample(int a, int b)        // constructor
{ i=a;j=b;}
Sample (Sample & s)      //copy constructor
{ j=s.j  ; i=s.j;
    Cout  <<"\n Copy constructor working \n";
}
void print (void)
{cout <<i<< j<< "\n";}
:
};
```

Note : ***The argument to a copy constructor is passed by reference, the reason being that when an argument is passed by value, a copy of it is constructed. But the copy constructor is creating a copy of the object for itself, thus ,it calls itself. Again the called copy constructor requires another copy so again it is called.in fact it calls itself again and again until the compiler runs out of the memory .so, in the copy constructor, the argument must be passed by reference.***

The following cases may result in a call to a copy constructor:

- **When an object is passed by value to a function**:
  The pass by value method requires a copy of the passed argument to be created for the function to operate upon .Thus to create the copy of the passed object, copy constructor is invoked
  If a function with the following prototype :
  void cpyfunc(Sample); // Sample is a class
  then for the following function call
  cpyfunc(obj1); // obj1 is an object of Sample type
  the copy constructor would be invoked to create a copy of the obj1 object for use by cpyfunc().

- **When a function returns an object** :
  When an object is returned by a function the copy constructor is invoked

  Sample cpyfunc(); // Sample is a class and it is return type of cpyfunc()

  If func cpyfunc() is called by the following statement

  obj2 = cpyfunc();

  Then the copy constructor would be invoked to create a copy of the value returned by cpyfunc() and its value would be assigned to obj2. The copy constructor creates a temporary object to hold the return value of a function returning an object.

1 & 2 Marks Solved Problems :

Q1 :- Answer the questions after going through the following class.

```
class Exam
{char Subject[20] ;
        int Marks ;
public :
        Exam()                                  // Function 1
        {strcpy(Subject, "Computer" ) ; Marks = 0 ;}
        Exam(char P[ ])                         // Function 2
        {strcpy(Subject, P) ;
        Marks=0  ;
        }
       Exam(int M)                              // Function 3
      {strcpy(Subject, "Computer") ; Marks = M ;}
       Exam(char P[ ], int M)                   // Function 4
      {strcpy(Subject, P) ; Marks = M ;}
      };
```

a) Which feature of the Object Oriented Programming is demonstrated using Function 1, Function2, Function 3 and Function 4 in the above class Exam?

Ans:- Function Overloading (Constructor overloading)

b) Write statements in C++ that would execute Function 3 and Function 4 of class Exam.

Ans:- Exam a(10); and Exam b("Comp", 10);

Q2 Consider the following declaration :

```
class welcome
{public:
        welcome (int x, char ch);       // constructor with parameter
        welcome();                      // constructor without parameter
        void  compute();
private:
        int x;    char ch;
};
```

which of the following are valid statements

```
welcome obj (33, 'a9');
welcome obj1(50, '9');
welcome obj3();
obj1= welcome (45, 'T');
obj3= welcome;
```

Ans. Valid and invalid statements are

welcome obj (33, 'a9');	**valid**
welcome obj1(50, '9');	**valid**
welcome obj3();	**invalid**
obj1= welcome (45, 'T');	**valid**
obj3= welcome;	**invalid**

2 Marks Practice Problems

Q1 What do you understand by constructor and destructor functions used in classes ? How are these functions different from other member functions ? 2

Q2 What do you understand by default constructor and copy constructor functions used in classes ? How are these functions different from normal constructors ? **2**

Q3 Given the following C++ code, answer the questions (i) & (ii). 2

```
class TestMeOut
{
public :
~TestMeOut() // Function 1
{ cout << "Leaving the examination hall " << endl; }
TestMeOut() // Function 2
{ cout << "Appearing for examination " << endl; }
void MyWork() // Function 3
{ cout << "Attempting Questions " << endl; }
};
```

(*i*) In Object Oriented Programming, what is Function 1 referred as and when does it get invoked / called ?

(*ii*) In Object Oriented Programming, what is Function 2 referred as and when does it get invoked / called ?