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
{
 private:
      int rollno;
     float marks;
public:
     student( )                    //Constructor
     { rollno=0; marks=0.0; }
};
```

CONSTRUCTOR CALLING

A constructor can be called in two ways: implicitly and explicitly.

```
class student
 {
  private:
        int rollno;
       float marks;
 public:
       student( )                    //Constructor
       { rollno=0; marks=0.0; }
};
void main()
{
student s1; //implicit call
}
```

IMPLICIT CALL OF CONSTRUCTOR

```
class student
 {
  private:
        int rollno;
       float marks;
 public:
       student( )                  //Constructor
       { rollno=0; marks=0.0; }
};
void main()
{
student s1 = student();  //explicit call
}
```

EXPLICIT CALL OF CONSTRUCTOR

FEATURES OF CONSTRUCTORS

- ❑ Constructor has the same name as class.
- ❑ Constructor does not have any return type not even void.
- ❑ Constructor can be declared only in the public section of the class.
- ❑ Constructor is called automatically when the object of the class is created.

TYPES OF CONSTRUCTOR

1. **Default Constructor**: A constructor that accepts no parameter is called the Default Constructor.

2. **Parameterized Constructor**: A constructor that accepts parameters for its invocation is known as parameterized Constructor, also called as Regular Constructor.

3. **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.

DEFAULT CONSTRUCTOR

- ❑Default constructor takes no argument.
- ❑It is used to initialize an object of a class with legal initial values.

Example :

```
 class student
 {
  private:
        int rollno;
       float marks;
 public:
       student( )                 // Default Constructor
       { rollno=0; marks=0.0; }
       void display();
       void enter();
};
void main()
{
student s1; // default constructor called
}
```

PARAMETRISED CONSTRUCTOR

- Parametrized constructor accepts arguments.
- It initializes the data members of the object with the arguments passed to it.
- When the object s1 is created, default constructor is called and when the objects s2 is created, parametrized constructor is called and this is implicit call of parametrized constructor.
- When the object s3 is created, again the parametrized constructor is called but this explicit call of parametrized constructor.

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
       student( )
    // Default Constructor
       { rollno=0; marks=0.0; }
       student(int roll, char nam[20],
float mar)
       // Parametrized Constructor
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;
       }
```

```
       void display();
       void enter();
};

void main()
{
student s1;
student s2(1, "RAMESH",
350);
student s3=student(2,
"KAJAL", 400);
}
```

COPY CONSTRUCTOR

- ❑ Copy constructor accepts an object as an argument.
- ❑ It creates a new object and initializes its data members with data members of the object passed to it as an argument.

Syntax :
```
class_name( class_name & object_name)
          {



          }
```

Note: It is necessary to pass an object by reference in copy constructor because if do not pass the object by reference then a copy constructor will be invoked to make the copy of the object.
In this process copy constructor will be invoked again and it will be a recursive process ie this process will repeat again and again endlessly resulting in running short of memory. Ultimately our system will hang.

COPY CONSTRUCTOR EXAMPLE

Example :

Statement1: default constructor is called.

Statement2: parametrized constructor is called.

Statement 3: copy constructor is called.

Statement4: Parametrized constructor is called.

statement5: Copy constructor is called.

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
    student( )
       // Default Constructor
       { rollno=0; marks=0.0; }
    student(int roll, char nam[20],
              float mar)
     // Parametrized Constructor
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;  }
```

```
student( student & s)   // Copy
Constructor
       { rollno= s.rollno;
       strcpy(name, s.name);
       marks =s.marks;
       }
}
void main()
{
student s1; //statement1
student s2( 1, “RAJESH”, 450); //
statemet2
student s3(s2);    //statement3
student s4(2, “MOHIT”,320);
//statement4
student s5=s4;      //statement5
}
```

COPY CONSTRUCTOR INVOCATION

Case 1: 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. Example:

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
     student( )   // Default
       { rollno=0; marks=0.0; }

   student(int roll, char nam[20],
       float mar)  //Parametrized
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;       }
```

```
student( student & s)   // Copy
Constructor
       { rollno= s.rollno;
       strcpy(name, s.name);
       marks =s.marks;
       }
}
void job( student stu)
{     }

void main( )
{
student s6(6, “KAPIL”, 245);
job(s6);
}
```

Explanation:

As we are passing the object s6 by value so when the function call job is invoked a copy of stu is created and values of object s6 are copied to object stu.

COPY CONSTRUCTOR INVOCATION Contd…..

Case 2: When a function returns an object : When an object is returned by a function the copy constructor is invoked.

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
    student( )   // Default
       { rollno=0; marks=0.0; }
   student(int roll, char nam[20],
       float mar)   // Parametrised
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;
       }
```

```
student( student & s)   // Copy
Constructor
       { rollno= s.rollno;
       strcpy(name, s.name);
       marks =s.marks;
       }
}
student  job( )
{   student  t1(12, "AJAY", 443);
    return t1; }

void main( )
{
student s6(6, "KAPIL", 245);
student s7=job();
}
```

Explanation:

- ❑ When the function job is invoked it creates a temporary object t1 and then returns it.
- ❑ At that time the copy constructor is invoked to create a copy of the object t1 returned by job() and its value is assigned to s7.
- ❑ The copy constructor creates a temporary object to hold the return value of a function returning an object.

COPY CONSTRUCTOR INVOCATION Contd.....

Case 3: Copy constructor is invoked when the value of one object is copied to another object at the time of creating an object. In all the other cases copy constructor is not invoked and value of one object is copied to another object simply data member by data member. Example :

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
       student( )       // Default Constructor
       { rollno=0; marks=0.0; }

       student(int roll, char nam[20], float
mar)   // Parametrized Constructor
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;
       }
```

```
student( student & s)   // Copy
Constructor
       { rollno= s.rollno;
       strcpy(name, s.name);
       marks =s.marks;
       }   }
void main()
{
student s1; //statement1
student s2( 1, “RAJESH”, 450); //
statemet2
student s3(s2);    //statement3
student s4(2, “MOHIT”,320);
//statement4
Student s5=s4;      //statement5
s1=s5; } // statement6
```

EXPLANATION:

Statement1: default constructor is called
Statement2: parametrized constructor is called
Statement 3: copy constructor is called
statement4: Parametrized constructor is called
statement5: Copy constructor is called
statement6: Copy constructor is not called

CONSTRUCTOR OVERLOADING

C++ allows more than one constructors to be defined in a single class.
Defining more than one constructors in a single class provided they all differ in either type or number of argument is called constructor overloading.

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
     student( )      // Default Constructor
       { rollno=0; marks=0.0; }
     student(int roll, char nam[20],
float mar) // Parametrized Constructor
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;
       }
```

```
student(int roll, char nam[20])   //
Parametrized Constructor
       { rollno=roll;
       strcpy(name, nam);
       marks =100;
       }

student( student & s)   // Copy
Constructor
       { rollno= s.rollno;
       strcpy(name, s.name);
       marks =s.marks;
       }
}
```

- ❑ In the class student, there are 4 constructors:
  1 default constructor,
  2 parametrized constructors
  and 1 copy constructor.
- ❑ Defining more than one constructor in a single class is called constructor overloading.
- ❑ We can define as many parametrized constructors as we want in a single class provided they all differ in either number or type of arguments.

DESTRUCTOR

- ❑ A destructor is a member function whose name is the same as the class name and is preceded by tilde(“~”).
- ❑ It is automatically by the compiler when an object goes out of scope.
- ❑ Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed.
- ❑ **Note**: In this program first the object s1 is created. At that time constructor is called. When the program goes over the object s1 will also go out of scope. At that time destructor is called.

Example :

```
 class student
 {  private:
        int rollno;
       float marks;
 public:
       student( )                 //Constructor
       { rollno=0; marks=0.0; }
       ~student()         //Destructor
       {   }
};
void main()
{ student s1;}
```

FEATURES OF DESTRUCTOR

- It has the same name as that of the class and preceded by tilde sign(~).
- It is automatically by the compiler when an object goes out of scope.
- Destructors are usually used to deallocate memory and do other cleanup for a class object and its class members when the object goes out of scope.
- Destructor is always declared in the public section of the class.
- We can have only one destructor in a class.

POINTS TO REMEMBER

- Constructors and destructors do not have return type, not even void nor can they return values.

- 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 constructor of the base class is called first and then the constructor of the child class/es.

- The destructor for a child class object is called before destructors for base class/es are called.

- If we do not declare our own constructor and destructor in the class, then they are automatically defined by the compiler.

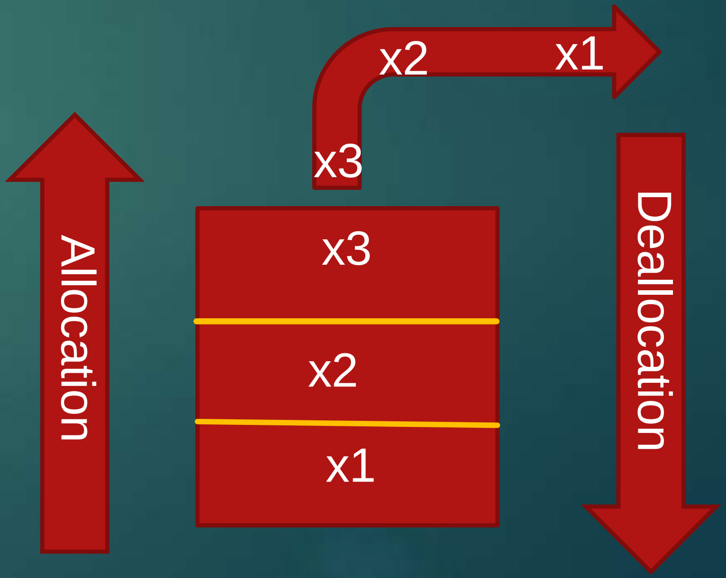
POINTS TO REMEMBER Contd……

- I f we declare our own constructor of any one type then it is necessary to define constructors of other types also otherwise we will not be able to create an object whose constructor definition is not given in the class.

- C++ compiler uses the concept of stack in memory allocation and deallocation. Thus compiler creates the memory in sequential order and delete the memory in reverse order.

Consider the class student declared in the previous slides. If the main program is as given below then allocation and deallocation of memory to objects will take place as given in the figure::

```
void main()
{
student x1, x2, x3;
}
```



EXAMPLE EXPLAINED

```
class student
 {
  private:
        int rollno;
       char name[20];
       float marks;
 public:
       student( )         // Default Constructor
       { rollno=0; marks=0.0;
       cout<<"Default constructor called\n"; }

       student(int roll, char nam[20], float mar)
// Parametrized Constructor
       { rollno=roll;
       strcpy(name, nam);
       marks =mar;
        cout<<"Parametrized constructor
called\n";
       }
```

```
student( student & s)   // Copy
Constructor
       { rollno= s.rollno;
       strcpy(name, s.name);
       marks =s.marks;
       cout<<"copy constructor
called\n";
       }
~student()    //Destructor
       { cout<<"Destructor
called\n";   }
}
void main()
{
student s1;
student s2( 1, "RAJESH", 450);
student s3(s2);
student s4(2, "MOHIT",320);
student s5=s4;     }
```

OUTPUT

```
Default constructor called
Parametrized constructor
called
Copy constructor called
Parametrized constructor
called
Copy constructor called
Destructor called
Destructor called
Destructor called
Destructor called
Destructor called
```