# Computer Programing (CP)

Lecture # 9

# Topic(s)

- Constructor
- Destructor
- Copy Constructor

# Constructor

- A **constructor** is a kind of member function that initializes an instance of its class.
- A **constructor** has the same name as the class and no return value.
- A constructor is automatically called when an object is created.

# Constructor

- If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).

- A **constructor** can have any number of parameters and a class may have any number of overloaded **constructors**.

# Types of Constructor

**Constructors can be divided into 5 types:**

- Default Constructor.
- Parametrized Constructor.
- Copy Constructor.
- Static Constructor (Advanced Concept).
- Private Constructor (Advanced Concept).

# Default Constructor

- A constructor with no parameters is a default constructor.

- If you do not provide a constructor, the compiler provides a default constructor.
  - All numeric fields in the class to zero.
  - All string and object fields to null.

- You can also define a default constructor explicitly.

- If you define a constructor for a class, C++ will not create a default constructor.

# Default Constructor:

```
class Calculator{
        private:
                float op1,op2;
        public:
                Calculator()
                {}
```

```
class Calculator{
        private:
                float op1,op2;
        public:
                Calculator()
                {
                        op1 = 0.0;
                        op2 = 0.0;
                }
```

Default Constructor

# Parametrized Constructor:

- It is possible to pass arguments to constructors.
- Typically, these arguments help initialize an object when it is created.
- To create a parameterized constructor, simply add parameters to it the way you would to any other function.
- When you define the constructor's body, use the parameters to initialize the object
- *Note: Must add a default constructor before parameterized constructor*

# Parametrized Constructor:

```
class Calculator{
        private:
                float op1,op2;
        public:
//              Calculator() // default constructor
//              {}
                Calculator() // Another default constructor
                {
                        op1 = 0.0;
                        op2 = 0.0;
                }
                Calculator(int p_Op1,int p_Op2)   // Parameterized Constructor
                {
                        op1 = p_Op1;
                        op2 = p_Op2;
                }
```

# Overloaded Constructor:

- Constructor can be overloaded in a similar way as function overloading.
- Overloaded constructors have the same name (name of the class) but different number of arguments.
- Depending upon the *number and type of arguments* passed, specific constructor is called.
- Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

# Overloaded Constructor:

```
class Calculator{
        private:
                float op1,op2;
        public:

//              Calculator() // default constructor
//              {}

                Calculator() // Another default constructor
                {
                        op1 = 0.0;
                        op2 = 0.0;
                }

                Calculator(int p_Op1)   // Parameterized Constructor
                {
                        op1 = p_Op1;
                        op2 = 0.0;
                }

                Calculator(int p_Op1,int p_Op2)  // overloaded Constructor
                {
                        op1 = p_Op1;
                        op2 = p_Op2;
                }
```

# Constructor can also Defined as:

- Constructor using Initializer List

```
class Calculator{
        private:
                float op1,op2;
        public:

//              Calculator() // default constructor
//              {}

                Calculator():op1(0.0), op2(0.0)   // Another default constructor
                {}

                Calculator(int p_Op1):op1(p_Op1), op2(0.0)    // Parameterized Constructor
                {}

                Calculator(int p_Op1,int p_Op2) :op1(p_Op1), op2(p_Op2) // overloaded Constructor
                {}
```

# Constructor using Set Methods:

```
class Calculator{
        private:
                float op1,op2;
        public:

//              Calculator() // default constructor
//              {}

                Calculator():op1(0.0), op2(0.0)   // Another default constructor
                {}

                Calculator(int p_Op1):op1(p_Op1)    // Parameterized Constructor
                {
                        setOp2(0.0);
                }

                Calculator(int p_Op1,int p_Op2)   // overloaded Constructor
                {
                        setOp1(p_Op1);
                        setOp2(p_Op2);
                }
```

# Destructor:

- Destructor" functions are the inverse of constructor functions.

- Destructor is a member function which destructs or deletes an object

- A **destructor** is a special member function of a class that is executed whenever an object of it's class goes out of scope.

or

- whenever the delete expression is applied to a pointer to the object of that class.

# Destructor:

- Destructors have same name as the class preceded by a tilde (~).
- Destructors don’t take any argument and don’t return anything.
- If we do not write our own destructor in class, compiler creates a default destructor for us.
- No overloaded Destructor in a Class.

# Destructor:

- A destructor function is called automatically when the object goes out of scope:

  (1) the function ends
  (2) the program ends
  (3) a block containing local variables ends
  (4) a delete operator is called

# Destructor:

- The default destructor works fine unless we have dynamically allocated memory or pointer in class. When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed. This must be done to avoid memory leak.

- *Note: This will be observed in Polymorphism implementation*

# Destructor:

```
class Calculator{
    private:
        float op1,op2;
    public:

//------------------------------------------------ Construcrtors ---------------------------------

//      Calculator() // default constructor
//      {}

        Calculator():op1(0.0), op2(0.0)   // Another default constructor
        {}

        Calculator(int p_Op1):op1(p_Op1), op2(0.0)    // Parameterized Constructor
        {}

        Calculator(int p_Op1,int p_Op2) :op1(p_Op1), op2(p_Op2) // overloaded Constructor
        {}

//------------------------------------------------ Destructor ---------------------------------

        ~Calculator() // Destructor
        {
            cout<<endl<<"Destructor Runs ..... ";
        }
```

# Copy Constructor:

- A copy constructor is a member function which initializes an object using another object of the same class.

- A copy constructor has the following general function prototype

***ClassName (const ClassName &old_obj);***

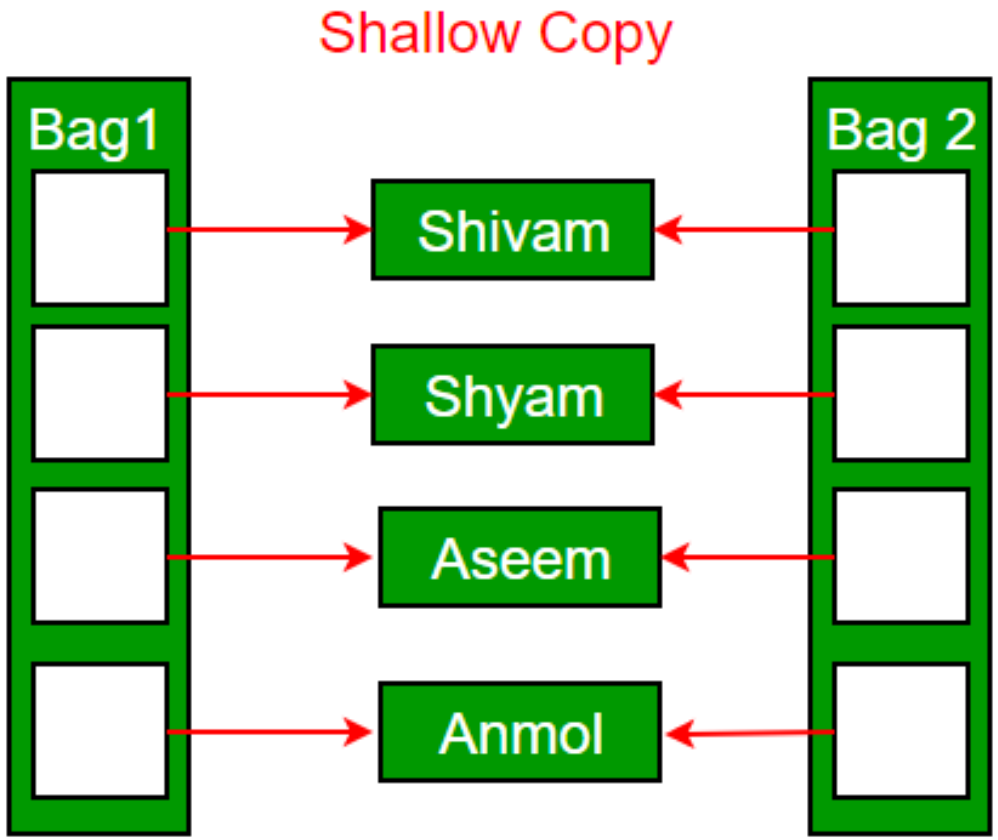
# Types of Copy Constructor:

- Copy constructor are of 2 type(s)

1. Default Copy Constructor (Shallow Copying)
2. User Defined Copy Constructor (Deep Copying)

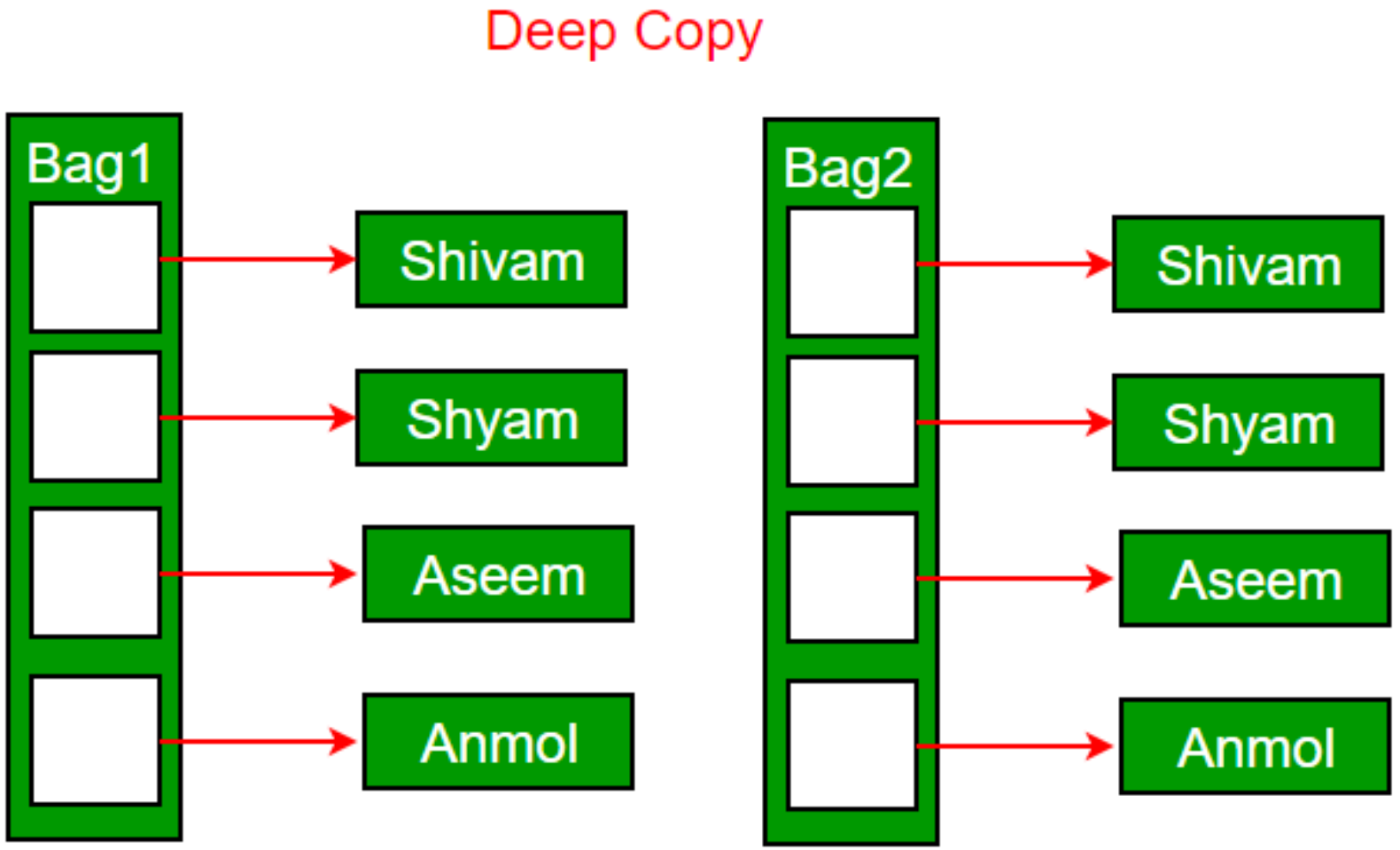
# Default Copy  Constructor (Shallow Copying)

- If we don’t define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member wise copy between objects.



# User Defined Copy Constructor (Deep Copying)

- In user defined copy constructor, we make sure that pointers (or references) of copied object point to new memory locations.



# Analyze (?)

Which of the following two statements call Deep copying and which one calls Shallow Copying?

```
MyClass t1, t2;
MyClass t3 = t1;  // ----> (1)
t2 = t1;          // -----> (2)
```

# Answer

- Copy constructor is called when a new object is created from an existing object, as a copy of the existing object.

- Assignment operator is called when an already initialized object is assigned a new value from another existing object.

- In the above example (1) calls copy constrictor and (2) assignment operator.