# CONSTRUCTORS AND DESTRUCTORS

By Avani M. Sakhapara

Asst Professor

IT Dept, KJSCE

# What is a Constructor?

- It is a member function which initializes a class.
- A constructor has:

    (i) the same name as the class itself

    (ii) no return type

# Constructor

- A constructor is called automatically whenever a new instance of a class is created.
- If you do not specify a constructor, the compiler generates a default constructor for you (expects no parameters and has an empty body).
- Default Constructor Example:

```
rectangle()
{

}
```

# Characteristics of Constructor

- They should be declared in the public section
- They are automatically invoked and have no return types, neither can they return values
- They can not be inherited
- They can not be virtual nor can we refer to their addresses & they can have default arguments

# Parameterized Constructors

- The constructors that can take arguments

```
 rectangle(float w, float h)
 {
 height = h;
 width = w;
 xpos = 0;
 ypos = 0;
}
```

- The parameters of a constructor can be any type except the class name to which it belongs

# Parameterized Constructor Example

```
class rectangle {
 private:
   float height;
   float width;
   int xpos;
   int ypos;

 public:
// constructor
 rectangle(float w, float h)
    {
 height = h;
 width = w;
 xpos = 0;
 ypos = 0;
}
// draw member function
 void draw()
{
//logic for drawing
rectangle
}

};

void main()
{
//calling constructor
rectangle r1(3.0, 2.0);
r1.draw();
}
```

# Constructors with Default Arguments

```
rectangle(float w, float h = 100)
{
height = h;
 width = w;
 xpos = 0;
 ypos = 0;
}
```

```
int main()
{
rectangle(200);
return 0;
}
```

height=100

width=200

xpos=0

ypos=0

# Copy Constructors

- The constructor that can accept a reference to its own class as a parameter which is known as copy constructor
- We can not pass by value to a copy constructor

# Copy Constructor Example

```
//copy constructor
rectangle(rectangle &t)
{
height = t.height;
 width = t.width;
 xpos = t.xpos;
 ypos = t.ypos;
}
//another constructor
rectangle(float w, float h = 100)
{
 height = h;
 width = w;
 xpos = 0;
 ypos = 0;
}
```

```
int main()
{
rectangle r1(200,300);
rectangle r2(r1);
return 0;
}
```

# Constructor Overloading

- You can have more than one constructor in a class, as long as each has a different list of arguments.

# Constructor Overloading Example

```
class rectangle {
 private:
   float height;
   float width;
   int xpos;
   int ypos;
 public:
// constructor
rectangle()
{
 height = 10;
 width = 10;
 xpos = 0;
 ypos = 0;
}
 // another constructor
 rectangle(float w, float h)
 {
 height = h;
 width = w;
 xpos = 0;
 ypos = 0;
}
```

```
// draw member function
void draw()
{
//logic for drawing
rectangle
}

};


void main()
{
 rectangle r1(3.0, 2.0);
 rectangle r2;

 r1.draw();
 r2.draw();
}
```

# DESTRUCTOR

- A destructor is a member function having sane name as that of its class preceded by ~(tilde) sign and which is used to destroy the objects that have been created by a constructor.
- It is implicitly invoked when an object's scope is over.
- Example: ~rectangle()

  {

  }

# Characteristics of Destructor

- It’s primary purpose is to destroy memory used by an object
- A default destructor is called when not defined
- If we are using dynamic objects, we have to do our own memory cleanup (garbage collection!)
- It never takes an argument, nor it returns any value
- Whenever new is used to allocate memory in the constructors, we should use delete to free it
- Destructors are called in the reverse order of Constructors