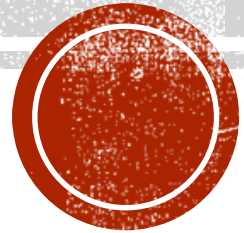
# OBJECT ORIENTED PROGRAMMING...

Working with Classes, Constructors and Destructors




# CONSTRUCTORS/DESTRUCTORS


# ACTIVITY

Write a simple C++ class to represent a rectangle, including methods to calculate its perimeter and area

In Object-Oriented Programming (OOP), initializing variables when creating objects is generally a good practice.

Proper initialization ensures that the object is in a valid, predictable state from the moment it is instantiated.


# CONSTRUCTORS

A **constructor** is a **special function** in a class that is **automatically called** when an object of the class is **created**.

# CONSTRUCTORS

- Special class member functions **automatically called** when an object of a class is **created/instantiated**.
- Purpose: initialise the newly created object.
- A constructor's name: Same as the name of the class.
  - defined inside or outside the class definition
- A constructor CANNOT return a value and has NO return type (NOT even void).
- To create a constructor, use the same name as the class, followed by parentheses ():
- There are 3 types of constructors:
  - Default constructors
  - Parametrized constructors
  - Copy constructors

# DEFAULT CONSTRUCTOR: EXAMPLE

The constructor which does NOT take any argument (has no parameters).


# ACTIVITY

Write a default constructor to initialize the variables to Zero.

# DEFAULT CONSTRUCTOR: EXAMPLE

The constructor which does NOT take any argument (has no parameters).

***Example:***

```
Rectangle()
{
        width=0;
        length=0;
}
```

# DEFAULT CONSTRUCTOR

- A constructor that can be invoked **without any arguments** e.g. a constructor with an **empty parameter list**, or a constructor with **default values** for all of its arguments.

**ACTIVITY:**

Is it possible to define more than one default constructor for a class? [YES/NO]

Justify…

Even if any constructor is NOT defined explicitly, the compiler will automatically provide a default constructor implicitly that leaves all data members un-initialized.

# PARAMETERIZED CONSTRUCTOR: EXAMPLE

Pass arguments to constructors. The initial values are passed as arguments to the constructor function.

The arguments passed to a constructor are typically used to initialise the object's data members

Used to initialize the various data elements of different objects with different values when they are created.

Example:

```
Rectangle(int w, int l)
{
        width=w;
        length=l;
}
```

# INVOKE CONSTRUCTORS

- A constructor CANNOT be called directly like a normal member function.

Example:

```
int main()
{
Rectangle rect1;
        rect1.Rectangle();
}
```

# PASSING ARGUMENTS TO CONSTRUCTOR

- Constructor is not invoked like a normal function.
- Arguments can ***only*** be passed to it when an object is declared.
- The *comma-separated list* of arguments must appear in parentheses after the object name.

Example:

```
int main()
{
        Rectangle rect1(1,1);
}
```

# CONSTRUCTORS

Whenever one or more non-default constructors (with parameters ) are defined for a class …

A default constructor(without parameters) should be explicitly defined as the compiler will not provide a default constructor.

# CONSTRUCTORS VS. MEMBER FUNCTIONS

## Differences

- **Constructor** name must be same as class name but **Functions** cannot have same name as class name.
- **Constructor** does not have a return type but **Functions** must have a return type.
- **Constructors** are invoked automatically at the time of object creation and cannot call explicitly using class objects. **Functions** are called explicitly using class objects.

## Similarities

- **Constructors** and **Member Functions** can be defined with different parameters.
- Parameters can be passed into **Constructors** and **Member Functions** as arguments.