3

IT 2022: Object Oriented Programming

Constructors & Destructors

Sandali Goonatilleke

Department of Computer Engineering

Module Content

- Introduction to Object-Oriented Programming in C++
- Classes & Objects
- Constructors & Destructors
- Class Abstraction and Encapsulation
- Composition
- Inheritance
- Polymorphism

Overview

- Constructors
- Destructors
- *this* Pointer
- Array of Objects

A Problem in Brief

Developing a game

- Need to assign some default values to different variables such as initial location, health, acceleration whenever a new player registers
- Can have separate functions for each quantity and need to call a list of functions every time a new player registers
- This process is lengthy and complicated

Solution

What if we can assign the quantities along with the declaration of the new player automatically?

Ideal Solution: A **constructor** can help do this in a better and simpler way

Constructor

- The process of creating and deleting objects in C++ is a vital task
- Each time an instance of a class is created the constructor method is called
- It is a **special kind of member function** of a class
- Mainly used to initialize the objects of the class
- **Has the same name as the class**
- When an object is created, the constructor is automatically called

Constructor

- Does not have any return type at all, not even void
- Go into the **public section** of the class declaration
- It is named as "constructor" because it constructs the value of data member of a class
- Initial values can be passed as arguments to the constructor function when the object is declared

Constructor Vs Other Member Functions

- The constructor has the same name as the class name
- The constructor is called when an object of the class is created
- A constructor does not have a return type
- When a constructor is not specified, the compiler generates a default constructor which does nothing

Special Characteristics of Constructors

- They should be declared in the public section
- They do not have any return type, not even void
- They get automatically invoked when the objects are created
- They cannot be inherited though derived class can call the base class constructor
- Like other functions, they can have default arguments
- Constructors cannot be virtual

Syntax of the Constructor

```
class scaler {
  public:
    scaler() { //constructor
      // constructor body
    }
};
```

Constructors

- Calling the constructor can be done in two ways
  - By calling constructor explicitly
  - By calling constructor implicitly

Calling Explicitly

value valNew = value (10);

Calling Implicitly

value valNew(10);

Types of Constructors

C++ offers four types of constructors

- Default constructor
- Parameterized constructor
- Copy constructor
- Dynamic constructor

Types of Constructors

1. Default Constructor

- **Zero Argument Constructor**
- A constructor which does not receive any parameters
- Even if a constructor is not defined explicitly, the compiler will provide a default constructor implicitly

Types of Constructors

1. Default Constructor

```
class Employee {
  public:
    int age;

    //default constructor
    Employee() {
      //data member is defined with the help of default constructor
      age = 50;
    }
};

int main() {
  //object of class Employee declared
  Employee e1;
  //prints value assigned by default   constructor
  cout << e1.age;
  return 0;
}
```

Types of Constructors

2. Parameterized Constructor

- A constructor that can accept one or more arguments
- This helps programmers to assign varied initial values to an object at the time of creation

Types of Constructors

2. Parameterized Constructor

```
class Employee {
  public:
    int age;

    Employee(int x) { // parameterized constructor
      age = x; //age assigned to value passed as argument while object
    }
};

int main() {
  Employee c1(40); //object c1 declared with argument 40 which gets as
  Employee c2(30);
  Employee c3(50);
  cout << c1.age << "\n";
  cout << c2.age << "\n";
  cout << c3.age << "\n";
  return 0;
}
```

Types of Constructors

3. Copy Constructor

- A type of constructor which is used to create a copy of an already existing object of a class
- The compiler provides a default Copy Constructor to all the classes
- Whenever there is a need for an object with the same values for data members as of an already existing object, a copy constructor comes into picture
- Invoked when an existing object is passed as a parameter

Types of Constructors

3. Copy Constructor

```
int main() {
  Employee employee1(34000, 2); // Parameterized constructor
  Employee employee2 = employee1; // Copy constructor
  cout << "Employee1 using parameterized constructor : \n";
  employee1.display();
  cout << "Employee2 using copy constructor : \n";
  employee2.display();
  return 0;
}
```

Types of Constructors

4. Dynamic Constructor

- When allocation of memory is done dynamically using dynamic memory allocator new in a constructor, it is known as dynamic constructor
- Can dynamically initialize the objects

Destructor

- As the name implies, **destructors are used to destroy the objects**
- Destructor names are same as the class
- Can have only one destructor per class
- It has no return type, not even void
- They are preceded by a tilde (~)
- It is a good practice to declare the destructor after the end of using constructor

Declaration and Definition of the Destructor

```
~Cube()
{


}
```

Example: Implementation of Constructor and Destructor

```
class Z
{
public:
    // constructor
    Z()
    {
        cout<<"Constructor called"<<endl;
    }
```

Example: Implementation of Constructor and Destructor

```
    // destructor
    ~Z()
    {
        cout<<"Destructor called"<<endl;
    }
};
```

Exercise 1

- Compare and contrast **Constructor** and **Destructor**

Assignments of Objects

- You can assign one object to another of the same class.

```
value val1, val2;
val1 = val2;
```

this Pointer

- When defining member functions for a class you sometimes want to refer to the calling object
- The ***this*** pointer is a predefined pointer that points to the calling object
- ***this*** is not the name of the calling object but is the name of the pointer that points to the calling object
- The ***this*** pointer cannot have its value changed
  - It always points to the calling object.

An Array of Objects

- The Array of Objects stores ***objects***
- An array of a class type is also known as **an array of objects**

Syntax:

```
ClassName ObjectName[number of objects];
```

Thank You!