Objects and Classes in C Plus Plus

Module 3 - Constructors and Destructors in C Plus Plus

Constructors and Destructors in C++

Constructors and **Destructors** are special member functions in a class in C++ that are automatically called when objects are created and destroyed, respectively.

Constructors

A constructor is a special member function that initializes objects of a class. It has the same name as the class and no return type.

Key Features of Constructors:

- Automatically invoked when an object is created.
- Used to initialize data members of a class.
- Can be overloaded (multiple constructors with different parameters).
- Cannot have a return type, not even void.

Types of Constructors:

1. Default Constructor: No parameters; provides default initialization.
2. Parameterized Constructor: Accepts parameters to initialize specific values.
3. Copy Constructor: Initializes an object by copying another object of the same class.

Default Constructor in C++

A default constructor is a constructor that takes no arguments or provides default values for all parameters. It initializes objects with default values when no specific values are provided during object creation.

Solved Example on Default Constructor

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    int length, breadth;
```

```
public:
    // Default constructor
    Rectangle() {
        length = 0;
        breadth = 0;
        cout << "Default constructor is called!" << endl;
    }

    // Member function to display dimensions
    void showDimensions() {
        cout << "Length: " << length << ", Breadth: " << breadth << endl;
    }
};

int main() {
    Rectangle rect; // Default constructor is invoked

    rect.showDimensions();
    return 0;
}
```

OUTPUT:

```
Default constructor is called!
Length: 0, Breadth: 0
```

Parameterized Constructor in C++

A **parameterized constructor** in C++ is a constructor that takes arguments to initialize an object with specific values. It allows for the flexibility of initializing objects during their creation with different values, unlike the default constructor.

Solved Example on Parameterized Constructor

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    int length, breadth;

public:
    // Parameterized constructor
```

Introduction to 'C++' Language

```
    Rectangle(int l, int b) {
        length = l;
        breadth = b;
        cout << "Parameterized constructor is called!" << endl;
    }

    // Member function to display dimensions
    void showDimensions() {
        cout << "Length: " << length << ", Breadth: " << breadth << endl;
    }
};

int main() {
    Rectangle rect1(10, 5); // Parameterized constructor is invoked
    Rectangle rect2(15, 7); // Another object with different values

    rect1.showDimensions();
    rect2.showDimensions();
    return 0;
}
```

OUTPUT:

```
Parameterized constructor is called!
Length: 10, Breadth: 5
Parameterized constructor is called!
Length: 15, Breadth: 7
```

Copy Constructor in C++

A **copy constructor** is a special constructor in object-oriented programming (OOP) used to create a new object as a copy of an existing object. It is often used to initialize an object with the values of another object of the same class.

Solved Example on Copy Constructor

```
#include <iostream>
using namespace std;

class Student {
public:
    int age;
    string name;
```

```
    // Parameterized constructor
    Student(int a, string n) {
        age = a;
        name = n;
    }

    // Copy constructor
    Student(const Student &s) {
        age = s.age;
        name = s.name;
    }

    // Method to display the details of the student
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student student1(20, "John");
    Student student2 = student1;  // Copy constructor is called

    student1.display();  // Output: Name: John, Age: 20
    student2.display();  // Output: Name: John, Age: 20

    return 0;
}
```

OUTPUT:

```
Name: John, Age: 20
Name: John, Age: 20
```

Explanation of the code:

1. Student student1(20, "John");
This creates an object student1 of the Student class using the parameterized constructor, initializing the age to 20 and name to "John".

2. Student student2 = student1;
This line invokes the copy constructor, creating student2 as a copy of student1. The values of age and name from student1 are copied to student2.

3. student1.display();
Displays the details of student1:
Output: Name: John, Age: 20

4. student2.display();
Displays the details of student2:
Output: Name: John, Age: 20

Both objects, student1 and student2, have the same values for age and name after the copy constructor is called.

Solved Example on Constructors

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    int length, breadth;

public:
    // Default constructor
    Rectangle() : length(0), breadth(0) {}

    // Parameterized constructor
    Rectangle(int l, int b) : length(l), breadth(b) {}

    // Member function to display dimensions
    void showDimensions() {
        cout << "Length: " << length << ", Breadth: " << breadth << endl;
    }
};

int main() {
    Rectangle rect1; // Default constructor is called
    Rectangle rect2(10, 5); // Parameterized constructor is called

    cout << "Rectangle 1: ";
    rect1.showDimensions();

    cout << "Rectangle 2: ";
    rect2.showDimensions();
```

```
    return 0;
}
```

OUTPUT:

```
Rectangle 1: Length: 0, Breadth: 0
Rectangle 2: Length: 10, Breadth: 5
```

Constructor Overloading

Constructor overloading in C++ refers to the ability to define multiple constructors in a class with the same name, but with different parameter lists. This allows objects of a class to be initialized in various ways depending on the parameters passed at the time of creation.

Rules for Constructor Overloading:

- Constructors must have the same name as the class.
- They must differ in the number or type of their parameters.

Solved Example on Constructor Overloading

```
#include <iostream>
using namespace std;

class Student {
public:
    int age;
    string name;

    // Default constructor
    Student() {
        age = 0;
        name = "Unknown";
        cout << "Default Constructor Called!" << endl;
    }

    // Parameterized constructor
    Student(int a, string n) {
        age = a;
        name = n;
        cout << "Parameterized Constructor Called!" << endl;
    }

    // Copy constructor
```

```
    Student(const Student &s) {
        age = s.age;
        name = s.name;
        cout << "Copy Constructor Called!" << endl;
    }

    // Method to display the details of the student
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    Student student1;  // Default constructor is called
    Student student2(20, "John");  // Parameterized constructor is called
    Student student3 = student2;  // Copy constructor is called

    student1.display();  // Output: Name: Unknown, Age: 0
    student2.display();  // Output: Name: John, Age: 20
    student3.display();  // Output: Name: John, Age: 20

    return 0;
}
```

OUTPUT:

```
Default Constructor Called!
Parameterized Constructor Called!
Copy Constructor Called!
Name: Unknown, Age: 0
Name: John, Age: 20
Name: John, Age: 20
```

Explanation:

Default Constructor (Student()):

• When student1 is created, the default constructor is invoked, initializing age to 0 and name to "Unknown".

Parameterized Constructor (Student(int a, string n)):

• When student2 is created, the parameterized constructor is called, initializing age to 20 and name to "John".

Copy Constructor (Student(const Student &s)):

• When student3 is created as a copy of student2, the copy constructor is invoked, and the age and name are copied from student2 to student3.

Destructors

A **destructor** is a special member function that is automatically invoked when an object goes out of scope or is explicitly deleted. Its purpose is to clean up resources allocated by the object.

Key Features of Destructors:
1. Automatically invoked when an object is destroyed.
2. Has the same name as the class but is preceded by a ~ symbol.
3. Cannot take arguments or return values.
4. Used to release resources like memory, file handles, etc.

Solved Example on Destructors

```
#include <iostream>
using namespace std;

class Sample {
public:
    // Constructor
    Sample() {
        cout << "Constructor is called!" << endl;
    }

    // Destructor
    ~Sample() {
        cout << "Destructor is called!" << endl;
    }
};

int main() {
    cout << "Creating an object..." << endl;
    Sample obj; // Constructor is called here

    cout << "Exiting program..." << endl;
    return 0; // Destructor is called here
}
```

OUTPUT:

```
Creating an object...
```

Introduction to 'C++' Language

```
Constructor is called!
Exiting program...
Destructor is called!
```