DSA Lab 5 – Stacks

CSE102
March 7, 2025

Classes and Constructors in C++

Overview

In C++, a **class** is a blueprint for creating objects. It encapsulates data for the object (data members) and functions (member functions) that operate on that data. A class can have multiple types of constructors:

- **Default Constructor:** Initializes objects with default values.
- **Parameterized Constructor:** Allows objects to be initialized with user-provided values.
- **Copy Constructor:** Creates a new object as a copy of an existing object.

Example: A Simple `Point` Class

Below is an example that demonstrates:

- A default constructor.
- A parameterized constructor.
- A copy constructor.
- A destructor to manage any required cleanup.

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

class Point {
private:
    int x, y;
public:
    // Default constructor
    Point() {
        x = 0;
        y = 0;
        cout << "Default Constructor called" << endl;
    }

    // Parameterized constructor
    Point(int x, int y) {
        this->x = x;
        this->y = y;
        cout << "Parameterized Constructor called" << endl;
    }

    // Copy constructor
    Point(const Point &p) {
        x = p.x;
        y = p.y;
        cout << "Copy Constructor called" << endl;
    }

    // Member function to display the point coordinates
    void display() {
        cout << "Point(" << x << ", " << y << ")" << endl;
    }

    // Destructor
    ~Point() {
        cout << "Destructor called for Point(" << x << ", " << y << ")" <<
            endl;
    }
};

int main() {
    Point p1;           // Calls default constructor
    Point p2(3, 4);     // Calls parameterized constructor
    Point p3 = p2;      // Calls copy constructor

    cout << "Displaying points:" << endl;
    p1.display();
    p2.display();
    p3.display();

    return 0;
}
```

Listing 1: Point Class Example

Expected Output

```
Default Constructor called
Parameterized Constructor called
Copy Constructor called
Displaying points:
Point(0, 0)
Point(3, 4)
Point(3, 4)
```

```
Destructor called for Point(3, 4)
Destructor called for Point(3, 4)
Destructor called for Point(0, 0)
```

Note: The order of destructor calls may vary depending on the compiler.