1. Basic Concepts of Polymorphism

1.1 What is Polymorphism?

Polymorphism means "many forms" - the ability of objects to take on multiple forms or behave differently based on their type.

Real-World Analogy: Think of a smartphone's "share" button:

- Share a photo → Opens image sharing options
- Share a document → Opens document sharing options
- Share a location → Opens map sharing options
- Same button, different behavior based on what you're sharing

Types of Polymorphism in C++:

1. **Compile-Time Polymorphism** (Static Binding)
   - Function Overloading
   - Operator Overloading
2. **Runtime Polymorphism** (Dynamic Binding)
   - Virtual Functions
   - Abstract Classes

Benefits of Polymorphism:

- **Flexibility**: Write code that works with different types
- **Extensibility**: Add new types without changing existing code
- **Maintainability**: Reduce code duplication
- **Abstraction**: Hide implementation details

1.2 Compile-Time vs Runtime Polymorphism

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

// COMPILE-TIME POLYMORPHISM: Function Overloading
class Calculator {
```

```
public:
    // Same function name, different parameters
    int add(int a, int b) {
        return a + b;
    }

    double add(double a, double b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
};

// RUNTIME POLYMORPHISM: Virtual Functions
class Shape {
public:
    virtual void draw() {
        cout << "Drawing a shape" << endl;
    }

    virtual ~Shape() {}
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing a circle" << endl;
    }
};

class Rectangle : public Shape {
public:
    void draw() override {
        cout << "Drawing a rectangle" << endl;
    }
};

int main() {
```

```
    cout << "=== Compile-Time Polymorphism ===" << endl;
    Calculator calc;
    cout << "add(5, 3) = " << calc.add(5, 3) << endl;
    cout << "add(5.5, 3.2) = " << calc.add(5.5, 3.2) << endl;
    cout << "add(1, 2, 3) = " << calc.add(1, 2, 3) << endl;

    cout << "\n=== Runtime Polymorphism ===" << endl;
    Shape* shape1 = new Circle();
    Shape* shape2 = new Rectangle();

    shape1->draw();  // Calls Circle::draw()
    shape2->draw();  // Calls Rectangle::draw()

    delete shape1;
    delete shape2;

    return 0;
}
```

Output:

```
=== Compile-Time Polymorphism ===
add(5, 3) = 8
add(5.5, 3.2) = 8.7
add(1, 2, 3) = 6

=== Runtime Polymorphism ===
Drawing a circle
Drawing a rectangle
```

Revision #1
Created 2025-11-25 03:29:53 UTC by DS
Updated 2025-11-25 03:30:48 UTC by DS