Student: Samantha Lee
ID: SL2023042

Question 1: Implement a simple Calculator class with add, subtract, multiply, and divide operations.
Solution:

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

// Simple calculator class with basic operations
class Calculator {
private:
    double result; // Stores the current result

public:
    // Constructor initializes result to 0
    Calculator() {
        result = 0;
    }

    // Method to add a number to the result
    void add(double num) {
        result = result + num;
    }

    // Method to subtract a number from the result
    void subtract(double num) {
        result = result - num;
    }

    // Method to multiply the result by a number
    void multiply(double num) {
        result = result * num;
    }

    // Method to divide the result by a number
    void divide(double num) {
        // Check for division by zero
        if (num != 0) {
            result = result / num;
        } else {
            cout << "Error: Cannot divide by zero" << endl;
        }
    }

    // Method to get the current result
```

```
    double getResult() const {
        return result;
    }

    // Method to reset the result to 0
    void clear() {
        result = 0;
    }
};

int main() {
    // Create a new calculator object
    Calculator calc;

    // Display the initial result (should be 0)
    cout << "Calculator Demonstration:" << endl;
    cout << "Initial result: " << calc.getResult() << endl;

    // Add 15 to the result
    calc.add(15);
    cout << "After adding 15: " << calc.getResult() << endl;

    // Subtract 7 from the result
    calc.subtract(7);
    cout << "After subtracting 7: " << calc.getResult() << endl;

    // Multiply the result by 3
    calc.multiply(3);
    cout << "After multiplying by 3: " << calc.getResult() << endl;

    // Divide the result by 4
    calc.divide(4);
    cout << "After dividing by 4: " << calc.getResult() << endl;

    // Attempt to divide by zero
    cout << "Attempting to divide by zero: ";
    calc.divide(0);
    cout << "Result remains: " << calc.getResult() << endl;

    // Clear the calculator
    calc.clear();
    cout << "After clearing: " << calc.getResult() << endl;

    return 0;
}
```

Question 2: Implement a Circle class with methods to calculate area and circumference.
Answer:

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

// Circle class to calculate area and circumference
class Circle {
private:
    double radius; // Private member to store the radius
    const double PI = 3.14159; // Constant value for PI

public:
    // Constructor to initialize the Circle object with a radius
    Circle(double rad) {
        radius = rad;
    }

    // Member function to calculate the area of the circle
    double calculateArea() {
        return PI * radius * radius; // Formula to calculate the area
    }

    // Member function to calculate the circumference of the circle
    double calculateCircumference() {
        return 2 * PI * radius; // Formula to calculate the circumference
    }

    // Getter method for radius
    double getRadius() const {
        return radius;
    }

    // Setter method for radius
    void setRadius(double r) {
        radius = r;
    }
};

int main() {
    // Create two circle objects with different radii
    Circle circle1(5.0);
    Circle circle2(7.5);

    cout << "Circle Demonstration:" << endl;

    // Demonstrate circle1
    cout << "Circle 1 (Radius = " << circle1.getRadius() << "):" << endl;
```

```
    cout << "  Area: " << circle1.calculateArea() << endl;
    cout << "  Circumference: " << circle1.calculateCircumference() << endl;

    // Demonstrate circle2
    cout << "Circle 2 (Radius = " << circle2.getRadius() << "):" << endl;
    cout << "  Area: " << circle2.calculateArea() << endl;
    cout << "  Circumference: " << circle2.calculateCircumference() << endl;

    // Change radius of circle1 and show the new calculations
    circle1.setRadius(10.0);
    cout << "Circle 1 after changing radius to " << circle1.getRadius() << ":" << endl;
    cout << "  Area: " << circle1.calculateArea() << endl;
    cout << "  Circumference: " << circle1.calculateCircumference() << endl;

    return 0;
}
```

Question 3: Implement a Stack data structure with push, pop, isEmpty, and peek operations.
Solution:

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

// Generic stack implementation using template
template <typename T>
class Stack {
private:
    T elements[100];  // Array to store elements
    int topIndex;     // Index of the top element

public:
    // Constructor initializes the stack
    Stack() {
        topIndex = -1;
    }

    // Method to push an element onto the stack
    void push(const T& value) {
        if (topIndex < 99) {
            elements[++topIndex] = value;
        } else {
            throw runtime_error("Stack overflow");
        }
    }

    // Method to pop an element from the stack
```

```
    T pop() {
        if (topIndex >= 0) {
            return elements[topIndex--];
        } else {
            throw runtime_error("Stack underflow");
        }
    }

    // Method to check if the stack is empty
    bool isEmpty() const {
        return topIndex == -1;
    }

    // Method to peek at the top element without removing it
    T peek() const {
        if (topIndex >= 0) {
            return elements[topIndex];
        } else {
            throw runtime_error("Stack is empty");
        }
    }
};

int main() {
    cout << "Stack Demonstration:" << endl;

    try {
        // Create a stack of integers
        Stack<int> stack;

        // Check if the stack is initially empty
        cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

        // Push elements onto the stack
        cout << "Pushing elements: 10, 20, 30, 40, 50" << endl;
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(40);
        stack.push(50);

        // Check if the stack is empty now
        cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

        // Peek at the top element
        cout << "Top element: " << stack.peek() << endl;

        // Pop elements from the stack
```

```
        cout << "Popping elements: ";
        while (!stack.isEmpty()) {
            cout << stack.pop() << " ";
        }
        cout << endl;

        // Check if the stack is empty after popping all elements
        cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

        // Try to peek at an empty stack
        cout << "Trying to peek at an empty stack: ";
        cout << stack.peek() << endl;
    } catch (const runtime_error& e) {
        cout << "Exception caught: " << e.what() << endl;
    }

    return 0;
}
```

Question 4: Implement a binary search function that searches for a target value in a sorted array.

Solution:

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

// Binary search function implementation
int binarySearch(int arr[], int low, int high, int target) {
    if (high >= low) {
        int mid = low + (high - low) / 2;

        // If found at mid, return it
        if (arr[mid] == target) {
            return mid;
        }

        // If element is smaller than mid, search in the left subarray
        if (arr[mid] > target) {
            return binarySearch(arr, low, mid - 1, target);
        }

        // Else search in the right subarray
        return binarySearch(arr, mid + 1, high, target);
    }

    // Element not present
```

```
    return -1;
}

int main() {
    cout << "Binary Search Demonstration:" << endl;

    // Create a sorted array
    int arr[] = {3, 7, 11, 15, 18, 21, 29, 36, 42, 55, 67, 78, 90};
    int size = sizeof(arr) / sizeof(arr[0]);

    // Print the array
    cout << "Sorted array: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    // Perform a successful search
    int target1 = 29;
    int result1 = binarySearch(arr, 0, size - 1, target1);
    if (result1 != -1) {
        cout << target1 << " found at index " << result1 << endl;
    } else {
        cout << target1 << " not found in the array" << endl;
    }

    // Perform an unsuccessful search
    int target2 = 35;
    int result2 = binarySearch(arr, 0, size - 1, target2);
    if (result2 != -1) {
        cout << target2 << " found at index " << result2 << endl;
    } else {
        cout << target2 << " not found in the array" << endl;
    }

    return 0;
}
```