Student: Alex Johnson
ID: AJ2023001

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

Solution:

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

class Calculator {
private:
    double result;

public:
    Calculator() : result(0) {}

    void add(double num) {
        result += num;
    }

    void subtract(double num) {
        result -= num;
    }

    void multiply(double num) {
        result *= num;
    }

    void divide(double num) {
        if (num != 0) {
            result /= num;
        } else {
            cout << "Error: Division by zero" << endl;
        }
    }

    double getResult() const {
        return result;
    }

    void clear() {
        result = 0;
    }
};
```

```
int main() {
    Calculator calc;

    cout << "Calculator Test:" << endl;
    cout << "Initial value: " << calc.getResult() << endl;

    calc.add(10);
    cout << "After adding 10: " << calc.getResult() << endl;

    calc.subtract(5);
    cout << "After subtracting 5: " << calc.getResult() << endl;

    calc.multiply(2);
    cout << "After multiplying by 2: " << calc.getResult() << endl;

    calc.divide(5);
    cout << "After dividing by 5: " << calc.getResult() << endl;

    calc.divide(0);
    cout << "After attempting to divide by 0: " << calc.getResult() << endl;

    calc.clear();
    cout << "After clearing: " << calc.getResult() << endl;

    return 0;
}
```

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

Solution:

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

class Circle {
private:
    double radius;
    const double PI = 3.14159;

public:
    Circle(double r) : radius(r) {}

    double calculateArea() {
        return PI * radius * radius;
    }

    double calculateCircumference() {
```

```
        return 2 * PI * radius;
    }

    double getRadius() const {
        return radius;
    }

    void setRadius(double r) {
        radius = r;
    }
};

int main() {
    Circle circle1(5);
    Circle circle2(7.5);

    cout << "Circle Test:" << endl;
    cout << "Circle 1 (radius " << circle1.getRadius() << "):" << endl;
    cout << "  Area: " << circle1.calculateArea() << endl;
    cout << "  Circumference: " << circle1.calculateCircumference() << endl;

    cout << "Circle 2 (radius " << circle2.getRadius() << "):" << endl;
    cout << "  Area: " << circle2.calculateArea() << endl;
    cout << "  Circumference: " << circle2.calculateCircumference() << endl;

    circle1.setRadius(10);
    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>
using namespace std;

class Stack {
private:
    int data[100];
    int top;

public:
```

```
    Stack() : top(-1) {}

    void push(int value) {
        if (top < 99) {
            data[++top] = value;
        } else {
            cout << "Stack overflow" << endl;
        }
    }

    int pop() {
        if (top >= 0) {
            return data[top--];
        } else {
            cout << "Stack underflow" << endl;
            return -1;
        }
    }

    bool isEmpty() const {
        return top == -1;
    }

    int peek() const {
        if (top >= 0) {
            return data[top];
        } else {
            cout << "Stack is empty" << endl;
            return -1;
        }
    }
};

int main() {
    Stack stack;

    cout << "Stack Test:" << endl;
    cout << "Is empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

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

    cout << "Top element: " << stack.peek() << endl;
    cout << "Is empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;
```

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

    cout << "Is empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;
    cout << "Trying to pop from an empty stack: " << stack.pop() << endl;
    cout << "Trying to peek at an empty stack: " << stack.peek() << 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;

int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target) {
            return mid;
        }

        if (arr[mid] < target) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}

int main() {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 45, 56, 72, 91};
    int size = sizeof(arr) / sizeof(arr[0]);
```

```
    cout << "Binary Search Test:" << endl;
    cout << "Array: ";
    for (int i = 0; i < size; ++i) {
        cout << arr[i] << " ";
    }
    cout << endl;

    int target1 = 23;
    int result1 = binarySearch(arr, size, target1);
    cout << "Searching for " << target1 << ": ";
    if (result1 != -1) {
        cout << "Found at index " << result1 << endl;
    } else {
        cout << "Not found" << endl;
    }

    int target2 = 42;
    int result2 = binarySearch(arr, size, target2);
    cout << "Searching for " << target2 << ": ";
    if (result2 != -1) {
        cout << "Found at index " << result2 << endl;
    } else {
        cout << "Not found" << endl;
    }

    return 0;
}
```