## Постановка задачі

Розробити об’єктно-орієнтовану бібліотеку для роботи зі стеком та обчислення виразів у постфіксному вигляді.

## Загальні вимоги

1. **Методи ініціалізації**:
   - Конструктор за замовчуванням.
   - Конструктор з параметрами.
   - Конструктор копіювання.
2. **Перевантаження операторів**:
   - Індексація (`[]`).
   - Присвоювання (`=`).
   - “Розумний доступ” (`->`).
3. **Методи візуалізації, збереження та відновлення**:
   - Візуалізація об’єктів.
   - Збереження на диск.
   - Відновлення з диска.
4. **Діалогове керування**:
   - Меню для створення, введення/виведення, візуалізації та виконання операцій над об’єктами.
5. **Перевантаження потокового введення/виведення**:
   - Введення з файлу.
   - Виведення в файл.
6. **Створення та використання файла бібліотеки**:
   - Створення файлу бібліотеки (*.lib або *.obj).
7. **Повторне використання класів**:
   - Наслідування без перекомпіляції.
8. **Застосування структури даних для розв’язання типової задачі**:
   - Обчислення виразів у постфіксному вигляді.

## Реалізація

### Клас `Stack`

```
#include <iostream>
#include <vector>
#include <fstream>

template <typename T>
class Stack {
private:
    std::vector<T> elements;

public:
    // Конструктор за замовчуванням
    Stack() {}

    // Конструктор з параметрами
    Stack(const std::vector<T>& elems) : elements(elems) {}

    // Конструктор копіювання
    Stack(const Stack& other) : elements(other.elements) {}

    // Перевантаження оператора присвоювання
    Stack& operator=(const Stack& other) {
```

```
        if (this != &other) {
            elements = other.elements;
        }
        return *this;
    }

    // Додавання елемента до стеку
    void push(const T& elem) {
        elements.push_back(elem);
    }

    // Видалення елемента зі стеку
    T pop() {
        if (elements.empty()) {
            throw std::out_of_range("Stack is empty");
        }
        T elem = elements.back();
        elements.pop_back();
        return elem;
    }

    // Перевантаження оператора індексації
    T& operator[](size_t index) {
        return elements[index];
    }

    // Візуалізація стеку
    void visualize() const {
        for (const auto& elem : elements) {
            std::cout << elem << " ";
        }
        std::cout << std::endl;
    }

    // Збереження стеку на диск
    void save(const std::string& filename) const {
        std::ofstream file(filename, std::ios::binary);
        size_t size = elements.size();
        file.write(reinterpret_cast<const char*>(&size), sizeof(size));
        file.write(reinterpret_cast<const char*>(elements.data()), size * sizeof(T));
    }

    // Відновлення стеку з диска
    void load(const std::string& filename) {
        std::ifstream file(filename, std::ios::binary);
        size_t size;
        file.read(reinterpret_cast<char*>(&size), sizeof(size));
        elements.resize(size);
        file.read(reinterpret_cast<char*>(elements.data()), size * sizeof(T));
    }
};
```

### Клас `PostfixCalculator`

```
#include <iostream>
#include <sstream>
#include <string>
#include "Stack.h"

class PostfixCalculator {
private:
    Stack<int> stack;

public:
    int evaluate(const std::string& expression) {
        std::istringstream iss(expression);
        std::string token;
        while (iss >> token) {
            if (isdigit(token[0])) {
                stack.push(std::stoi(token));
            } else {
                int b = stack.pop();
                int a = stack.pop();
```

```
                if (token == "+") stack.push(a + b);
                else if (token == "-") stack.push(a - b);
                else if (token == "*") stack.push(a * b);
                else if (token == "/") stack.push(a / b);
            }
        }
        return stack.pop();
    }
};
```

## Демонстраційна програма

```
#include <iostream>
#include "PostfixCalculator.h"

int main() {
    PostfixCalculator calculator;
    std::string expression = "23 10 + *";
    int result = calculator.evaluate(expression);
    std::cout << "Result: " << result << std::endl;
    return 0;
}
```

## Звіт

1. **Вступ**: Опис мети та завдань лабораторної роботи.
2. **Опис реалізації**: Детальний опис класів та методів.
3. **Тести**: Опис тестів та результати їх виконання.
4. **Висновки**: Аналіз отриманих результатів та можливі покращення.