МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**
(БГТУ им. В.Г. Шухова)

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

Лабораторная работа №9

по дисциплине: ООП
тема: «Использование стандартной библиотеки шаблонов STL»

Выполнил: ст. группы ПВ-223
Пахомов Владислав Андреевич

Проверили:
пр. Черников Сергей Викторович

Белгород 2024 г.

Лабораторная работа №9

«Использование стандартной библиотеки шаблонов STL»

Вариант 5

Цель работы: знакомство со стандартной библиотекой шаблонов в C++; получение навыков использования классов контейнеров, итераторов, алгоритмов.

Разработать программное обеспечения для решения соответствующего варианта. Разработать программное обеспечение для решения следующей задачи: преобразование математического выражения в обратную польскую запись, для хранения использовать stack, выполнить вычисления. Реализовать вычисление сложения, вычитания, и других арифметических операций над stack.

main.cpp

```
#include <iostream>
#include <string>
#include <set>
#include <map>
#include <stack>
#include <queue>
#include <cassert>
#include <cstdlib>
#include <algorithm>
#include <functional>
#include <math.h>
#include <sstream>
#include <windows.h>

class Token {
    public:
    std::string val;
    bool is_op;
    bool is_bin;
    int priority;
    std::function<Token(Token, Token)> bin_op = [](Token, Token) {throw std::runtime_error("Unsupported"); return
    ↪  Token("");};
    std::function<Token(Token)> un_op = [](Token) {throw std::runtime_error("Unsupported"); return Token("");};;

    Token(std::string val, std::function<Token(Token, Token)> bin_op, int priority) : val(val), bin_op(bin_op),
    ↪  priority(priority) {
        this->is_op = true;
        this->is_bin = true;
    };

    Token(std::string val, std::function<Token(Token)> un_op, int priority) : val(val), un_op(un_op), priority(priority)
    ↪  {
        this->is_op = true;
        this->is_bin = false;
    };

    Token(std::string val, int operation) {
        this->priority = operation;
        this->is_op = false;
        this->is_bin = false;
```

```
        this->val = val;
    }

    Token(std::string val) {
        this->priority = -1;
        this->is_op = false;
        this->is_bin = false;
        this->val = val;
    }

    Token() : val(""), is_op(false), is_bin(false), priority(0) {};

    bool isNumber() {
        return this->priority == -1;
    }

    bool isOpeningPar() {
        return this->priority == 0 && this->val == "(";
    }

    bool isClosingPar() {
        return this->priority == 0 && this->val == ")";
    }

    bool isOperation() {
        return this->is_op;
    }

    bool isBinOp() {
        return this->is_bin;
    }

    int getPriority() {
        return this->priority;
    }
};

std::vector<Token> tokens = {Token("-", [](Token a, Token b) {
    assert(a.isNumber());
    assert(b.isNumber());

    double aVal = std::stod(a.val);
    double bVal = std::stod(b.val);

    return Token(std::to_string(aVal - bVal));
}, 1), Token("+", [](Token a, Token b) {
    assert(a.isNumber());
    assert(b.isNumber());

    double aVal = std::stod(a.val);
    double bVal = std::stod(b.val);

    return Token(std::to_string(aVal + bVal));
}, 1), Token("*", [](Token a, Token b) {
```

```
    assert(a.isNumber());
    assert(b.isNumber());

    double aVal = std::stod(a.val);
    double bVal = std::stod(b.val);

    return Token(std::to_string(aVal * bVal));
}, 2), Token("/", [](Token a, Token b) {
    assert(a.isNumber());
    assert(b.isNumber());

    double aVal = std::stod(a.val);
    double bVal = std::stod(b.val);

    return Token(std::to_string(aVal / bVal));
}, 2),
Token("^", [](Token a, Token b) {
    assert(a.isNumber());
    assert(b.isNumber());

    double aVal = std::stod(a.val);
    double bVal = std::stod(b.val);

    return Token(std::to_string(std::pow(aVal, bVal)));
}, 3),
Token("sin", [](Token a) {
    assert(a.isNumber());

    double aVal = std::stod(a.val);

    return Token(std::to_string(std::sin(aVal)));
}, 4), Token("cos", [](Token a) {
    assert(a.isNumber());

    double aVal = std::stod(a.val);

    return Token(std::to_string(std::cos(aVal)));
}, 4), Token("~", [](Token a) {
    assert(a.isNumber());

    double aVal = std::stod(a.val);

    return Token(std::to_string(-aVal));
}, 5),
Token("(", 0),
Token(")", 0)};

std::queue<Token> infixToPostfix(std::queue<Token> input) {
    std::queue<Token> output;
    std::stack<Token> s;
    Token t;
    while (!input.empty()) {
                t = input.front();
        input.pop();
```

```
        if (t.isNumber()) {
            output.push(t);
                } else if (t.isOperation()) {
                        // Пока на вершине стека присутствует токен-операция op2
                        // и у op1 приоритет меньше либо равен приоритету op2, то:
                        while (!s.empty() && s.top().isOperation()
                                && t.getPriority() <= s.top().getPriority()
                        ) {
                                // переложить op2 из стека в выходную очередь
                                output.push(s.top());
                                s.pop();
                        }
                        // Положить op1 в стек
            s.push(t);
        } else if (t.isOpeningPar()) {
            s.push(t);
        } else if (t.isClosingPar()) {
            while (!s.empty() && !s.top().isOpeningPar()) {
                                assert (s.top().isOperation());
                                output.push(s.top());
                                s.pop();
            }
            if (s.empty()) {
                throw std::invalid_argument("Opening par is missing");
            } else {
                                s.pop();
                        }
        } else {
            std::string msg("Unknown character \'");
            msg += t.val + std::string("\'!");
            throw std::invalid_argument(msg);
                }
    }

    while (!s.empty()) {
        if (s.top().isOpeningPar()) {
            throw std::invalid_argument("Unclosed par");
        } else {
            assert(s.top().isOperation());
                        output.push(s.top());
                        s.pop();
                }
    }

        return output;
}

// Напечатать последовательность токенов
void printSequence(std::queue<Token> q) {
        while (!q.empty()) {
        std::cout << q.front().val << " ";
                q.pop();
        }
```

```
        std::cout << std::endl;
}

std::string getStringNumber(std::string expr, int& pos)
{
    int or_pos = pos;

    double v = .0;
    std::istringstream isexpr(expr);
    for (int i = 0; i < pos; i++)
        isexpr.ignore();

    isexpr >> v;

    pos = isexpr.tellg() - 1;
    return expr.substr(or_pos, isexpr.tellg() - or_pos);
}

std::queue<Token> stringToSequence(const std::string &s) {
    std::string tmp;
    std::queue<Token> res;
        for (int i = 0; i < s.size(); ++i) {
                if (isspace(s[i]))
            continue;
        else if (isdigit(s[i])) {
            if (!tmp.empty()) {
                auto pos = std::find_if(tokens.begin(), tokens.end(), [&tmp](Token v) {return v.val == tmp;});
                if (pos == tokens.end())
                    throw std::invalid_argument("Unknown operator");

                res.push(*pos);
            }

            tmp = getStringNumber(s, i);
            res.push(Token(tmp));
            tmp = "";
        } else if (s[i] == '(' || s[i] == ')') {
            tmp += s[i];

            auto pos = std::find_if(tokens.begin(), tokens.end(), [&tmp](Token v) {return v.val == tmp;});
            if (pos == tokens.end())
                throw std::invalid_argument("Unknown operator");

            res.push(*pos);

            tmp = "";
        } else {
            tmp += s[i];

            auto pos = std::find_if(tokens.begin(), tokens.end(), [&tmp](Token v) {return v.val == tmp;});
            if (pos == tokens.end())
                continue;

            res.push(*pos);
```

```
            tmp = "";
        }
        }

    auto pos = std::find_if(tokens.begin(), tokens.end(), [&tmp](Token v) {return v.val == tmp;});
    if (pos != tokens.end())
        res.push(*pos);

        return res;
}

void evalOpUsingStack(Token op, std::stack<Token> &s) {
    assert (op.isOperation());
    if (op.isBinOp()) {
        if (s.size() >= 2) {
                        Token b = s.top();
            s.pop();
                        Token a = s.top();
            s.pop();
                        s.push(op.bin_op(a, b));
                } else {
            throw std::invalid_argument("Syntax error");
        }
    } else if (!op.isBinOp() && !s.empty()) {
                Token a = s.top();
        s.pop();
                s.push(op.un_op(a));
        } else {
        throw std::invalid_argument("Syntax error");
        }
}

Token evaluate(std::queue<Token> expr) {
    std::stack<Token> s;
    Token t;
    while (!expr.empty()) {
                t = expr.front();
                assert (t.isNumber() || t.isOperation());
        expr.pop();
        if (t.isNumber()) {
            s.push(t);
        } else if (t.isOperation()) {
            evalOpUsingStack(t, s);
                }
    }

    if (s.size() == 1) {
                return s.top();
        } else {
        throw std::invalid_argument("Syntax error");
        }
}

int main() {
```

```
    SetConsoleOutputCP(CP_UTF8);

    std::string expr;
    std::cout << "Input expression: ";
    std::cout.flush();
    while (true) {
        char in = getchar();
        if (in == '\n') break;

        expr += in;
    }

    std::queue<Token> input = stringToSequence(expr);

    std::queue<Token> output = infixToPostfix(input);
    Token res = evaluate(output);
    std::cout << res.val;

        return 0;
}
```

Ссылка на репозиторий

Вывод: в ходе лабораторной работы ознакомились со стандартной библиотекой шаблонов в C++; получили навыки использования классов контейнеров, итераторов, алгоритмов.