Министерство науки и высшего образования Российской Федерации
Федеральное государственное
бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Фундаментальные Науки»

КАФЕДРА ФН-12 «Математическое моделирование»

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ:

Обратная польская запись

Студент: _____ Мациевский И. М.

дата, подпись Ф.И.О.

Преподаватель: _____ Волкова Л. Л.

дата, подпись Ф.И.О.

Москва, 2023

Содержание

Введение

Цель лабораторной работы: разработать программу для вычисления значений функции с использованием обратной польской записи.

Для достижения поставленной цели требуется решить следующие **задачи**.

1. Описать две формы записи — инфиксную и постфиксную.
2. Разработать алгоритм проверки введенной инфиксной записи.
3. Реализовать алгоритм перевода из инфиксной записи в постфиксную форму записи.
4. Составить таблицу значений функции, при заданных границах и шаге аргумента.
5. Выполнить тестирование реализации разработанных алгоритмов.

1 Аналитическая часть

1.1 Инфиксная запись

Инфиксная запись — это традиционный и способ записи математических выражений, при котором операторы располагаются между операндами, как это привычно в математике. Например: $a + b$, где a и b — операнды, а $+$ является оператором сложения. Для явного указания порядка операций, в инфиксной записи используются скобки.

1.2 Постфиксная запись

Постфиксная запись — также известна как обратная польская запись, способ записи математических выражений, при котором операторы следуют после своих операндов. В постфиксной записи порядок операндов и операторов определяет последовательность выполнения операций. Ниже представлены примеры постфиксных выражений.

1. Инфиксная запись: $a + b$, постфиксная запись: $a\ b+$.
2. Инфиксная запись: $3 * (4 + 5)$, постфиксная запись: $3\ 4\ 5\ +*$.
3. Инфиксная запись: $8/(2 + 3)$, постфиксная запись: $8\ 2\ 3\ +/$.

Постфиксная запись имеет преимущество в том, что она не требует использования скобок для уточнения порядка операций, поскольку последовательность операторов определена и не зависит от приоритета операторов, вследствие чего она более удобно для обработки компьютером.

2 Конструкторская часть

2.1 Проверка валидности строки

Перед тем, как переводить строку из инфиксной записи в постфиксную, нужно проверить корректно ли составлена эта строка.

1. Совпадает ли количество открывающихся скобок с количеством закрывающихся.
2. Нет ли двух знаков операций подряд.
3. Количество операторов должно быть на единицу меньше количества операндов.

Все эти проверки можно сделать с помощью одного прохода по строке, сохраняя баланс скобок в переменную, делая проверку соседних элементов на то, не являются ли они оба операторами, подсчитывая количество операторов и операндов.

2.2 Перевод в ОПЗ

Для перевода выражения из инфиксной в постфиксную запись используется алгоритм, который просматривает каждый символ входной строки, двигаясь от начала до конца. При этом создается строка, где будут храниться результаты в постфиксной записи, а также реализуется стек для операторов. Когда встречается число или переменная, они добавляются в строку постфиксной записи, и курсор смещается. Если встречается открывающая скобка, она помещается в стек, и курсор также смещается. Если встречается оператор, его приоритет сравнивается с операторами в стеке. Если текущий оператор имеет приоритет выше или равный оператору на вершине стека, он помещается в стек. Если приоритет ниже, из стека извлекаются операторы и добавляются в постфиксное выражение, пока не выполняется условие приоритета. Если встречается закрывающая скобка, выполняется цикл извлечения операторов из стека и добавления их в постфиксное выражение до тех пор, пока не встретится открывающая скобка. По завершении обработки всех символов входной строки, извлекаются все оставшиеся операторы из стека и добавляются в конец постфиксного выражения.

2.3 Подсчет значения функции в ОПЗ

Для нахождения значения функции в ОПЗ нужно пройти по строке в ОПЗ по следующему алгоритму.

1. Если встретилось число, добавить его в стек.
2. Если встретился бинарный оператор, два последних значения берутся из стека в обратном порядке, над ними проводится арифмитическая операция, ее результат записывается в вершины стека.

3. Если встретился унарный оператора, в данном случае - унарный минус, берётся последнее значение из стека и вычитается из нуля, так как унарный минус является правосторонним оператором.

4. Если встретилась функция, она применяется к вершине стека, ее значение записывается в вершину стека.

5. Последнее значение, после отработки алгоритма, является решением выражения.

2.4 Реализованные функции

Ниже представлены функции и классы, реализованные в проекте.

1. $Stack$ — стек.

2. $isFunc$ — определение является ли элемент функцией от x.

3. $isOp$ — определение является ли элемент оператором.

4. $isNum$ — определение является ли элемент числом.

5. $valid_inf$ — проверка корректности введенной строки.

6. $priority$ — определение приоритета функции.

7. $infToPost$ — перевод из инфиксной записи в постфиксную.

8. $calculatePost$ — подсчет значения выражения в ОПЗ.

3 Технологическая часть

Для реализации выбран язык C++. На листинге 1 представлена реализация программы (Реализация 1)

Листинг 1 – Исходный код

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <cstdlib>
#include <limits>
#include <locale>
#include <stack>

using namespace std;

const double PI = 3.141592653589793;

template<typename T>
#define SIZE 10
class Stack
{
    T *arr;
    int top;
    int capacity;

public:
    Stack(int size = SIZE);
    ~Stack();

    void push(T);
    void pop();
    T peek();
    int size();
    bool isEmpty();
    bool isFull();
};

template<typename T>
Stack<T>::Stack(int size)
{
```

```
    arr = new T[size];
    capacity = size;
    top = -1;
}
template<typename T>
Stack<T>::~Stack() {
    delete[] arr;
}

template<typename T>
void Stack<T>::push(T x)
{
    if (isFull())
    {
        cout << "Ошибка. Стек переполнен";
        return;
    }
    arr[++top] = x;
}

template<typename T>
void Stack<T>::pop()
{
    if (isEmpty())
    {
        cout << "Ошибка. Стек пуст";
        exit(EXIT_FAILURE);
    }
    arr[top--];
}

template<typename T>
T Stack<T>::peek()
{
    if (!isEmpty()) {
        return arr[top];
    }
    else {
        exit(EXIT_FAILURE);
    }
}
```

```
template<typename T>
int Stack<T>::size() {
    return top + 1;
}

template<typename T>
bool Stack<T>::isEmpty() {
    if (top == -1) {
        return true;
    }
    return false;
}

template<typename T>
bool Stack<T>::isFull() {
    if (top == capacity - 1) {
        return true;
    }
    return false;
}


bool isFunc(string& token) {
    return (token == "sin" or token == "ctg");
}

bool isOp(string &c) {
    return (c == "+" or c == "-" or c == "*" or c == "/" or c == "^" or
    c == "(" or c == ")");
}

bool isNum(const string& s) {
    for (char c : s) {
        if (!isdigit(c) and c != '.' and c != '-') {
            return false;
        }
    }
    return true;
}
```

```
bool valid_inf(string &inf) {
    int brackets = 0;
    for (int i = 0; i < inf.size(); i++) {
        if (i == 0 and inf[i] == '-') {
            inf.insert(0, "0 ");
        }
        if (i > 1 and inf[i] == '-' and inf[i - 2] == '(') {
            inf.insert(i , "0 ");
        }
        if (i > 1 and (inf[i] == '-' or inf[i] == '+' or inf[i] == '*'
   or inf[i] == '/')
            and (inf[i - 2] == '-' or inf[i - 2] == '+' or inf[i - 2]
   == '*' or inf[i - 2] == '/')) {
            cout << "Ошибка в валидности операторов" << endl << endl;
            return 0;
        }
        if (i > 1 and inf[i] == '-' and (inf[i - 2] == '/' or inf[i -
   2] == '*')) {
            inf.insert(i + 4, ") ");
            inf.insert(i, "( 0 ");
        }
        if (inf[i] == '(') {
            brackets++;
        }
        else if(inf[i] == ')'){
            brackets--;
            if (brackets < 0) {
                cout << "Неправильно расставлены скобки" << endl << endl;
                return false;
            }
        }
    }
    if (brackets != 0) {
        cout << "Неправильно расставлены скобки" << endl << endl;
        return false;
    }
    int operators = 0, operands = 0;
    for (int i = 0; i < inf.size(); i++) {
        string check;
        while (i < inf.size() and inf[i] != ' ') {
```

```
            check += inf[i];
            i++;
        }
        if (isOp(check)) {
            operators++;
            continue;
        }
        else if (isFunc(check)) {
            operators++;
            continue;
        }
        else if (isNum(check)) {
            operands++;
            continue;
        }
        else if (check == "x") {
            operands++;
            continue;
        }
        else {
            cout << "Ошибка в валидности токенов" << endl << endl;
            return 0;
        }
    }
    if (operands - 1 != operators) {
        cout << "Ошибка в валидности операторов" << endl << endl;
        return 0;
    }
    return true;
}

int priority(char op) {
    if (op == 's' or op == 'c')
        return 4;
    else if (op == '^')
        return 3;
    else if (op == '*' or op == '/')
        return 2;
    else if (op == '+' or op == '-')
        return 1;
    else
```

```
        return -1;
}

string infToPost(const string& expression) {
    stack<char> operators;
    string postfix;

    for (char c : expression) {
        if (c == ' ' or c == 'i' or c == 'n' or c == 't'  or c == 'g')
   {
            continue;
        }
        if (c == '.') {
            postfix += c;
            continue;
        }
        if (c != 's' and c != 'c' and isalnum(c)) {
            postfix += c;
        }
        else if (c == '(') {
            operators.push(c);
        }
        else if (c == ')') {
            while (operators.size() > 0 and operators.top() != '(') {
                postfix += ' ';
                postfix += operators.top();
                if (operators.top() == 's') {
                    postfix += "in";
                }
                if (operators.top() == 'c') {
                    postfix += "tg";
                }
                operators.pop();
            }
            if (operators.size() > 0 and operators.top() == '(') {
                operators.pop();
            }
        }
        else {
            while (operators.size() > 0 and priority(c) <= priority(
   operators.top())) {
```

```
                postfix += ' ';
                postfix += operators.top();
                if (operators.top() == 's') {
                    postfix += "in";
                }
                if (operators.top() == 'c') {
                    postfix += "tg";
                }
                operators.pop();
            }
            if (c != 's' and c != 'c') {
                postfix += ' ';
            }
            operators.push(c);
        }
    }

    while (operators.size() > 0) {
        postfix += ' ';
        postfix += operators.top();
        operators.pop();
    }

    return postfix;
}

double calculatePost(const string& expression) {
    setlocale(0, "");
    stack<double> operands;

    for(int i = 0; i < expression.size(); i++){
        if (expression[i] == ' ') {
            continue;
        }
        if (expression[i] == '-' and isdigit(expression[i + 1])) {
            int num = expression[i + 1] - '0';
            operands.push(-num);
            i = i + 2;
            continue;
        }
        if (isdigit(expression[i])) {
```

```
            if (expression[i + 1] == '.') {
                string numm;
                while (expression[i] != ' ') {
                    numm += expression[i];
                    i++;
                }
                locale::global(std::locale("en_US.UTF-8"));
                double num = std::stof(numm);
                operands.push(num);
                continue;
            }
            operands.push(expression[i] - '0');
        }
        else if (expression[i] == '+' or expression[i] == '-' or
expression[i] == '*' or expression[i] == '/' or expression[i] == '^'
) {
            double operand2 = operands.top();
            operands.pop();
            double operand1 = operands.top();
            operands.pop();
            double result = 0.0;

            switch (expression[i]) {
            case '+':
                result = operand1 + operand2;
                break;
            case '-':
                result = operand1 - operand2;
                break;
            case '*':
                result = operand1 * operand2;
                break;
            case '/':
                if (operand2 == 0) {
                    return numeric_limits<float>::infinity();
                }
                result = operand1 / operand2;
                break;
            case '^':
                if (operand1 < 0 and (operand2 < 1 and operand2 > 0)) {
                    cout << "Корень из отрицательного числа" << endl;
```

```
                    return numeric_limits<float>::infinity();
                    break;
                }
                result = pow(operand1, operand2);
                break;
            }

            operands.push(result);
        }
        else if (expression[i] == 's') {
            double operand = operands.top();
            operands.pop();
            operands.push(sin(operand));
        }
        else if (expression[i] == 'c') {
            double operand = operands.top();
            operands.pop();
            if (PI / 2 == operand) {
                return numeric_limits<float>::infinity();
                break;
            }
            if (operand == 0) {
                return numeric_limits<float>::infinity();
                break;
            }
            operands.push(1.0 / tan(operand));
        }
    }
    return operands.top();
}

int main() {
    setlocale(0, "");
    string infix;
    getline(cin, infix);
    if (infix == "") {
        cout << "Пустая строка";
        return 0;
    }
    if (!valid_inf(infix)) {
        return 0;
```

```
 }

 string postfix = infToPost(infix);
 cout << "Постфиксная запись: " << postfix << endl << endl;

 int xmin, xmax, h;
 cout << "Введите xmin: ";
 cin >> xmin;
 cout << "Введите xmax: ";
 cin >> xmax;
 cout << "Введите h: ";
 cin >> h;
 cout << endl;

 for (int i = xmin; i <= xmax; i = i + h) {
     string dubler = postfix;
     for (int j = 0; j < postfix.size(); j++) {
         if (postfix[j] == 'x') {
             dubler.replace(j, 1, to_string(i));
         }
     }
     if (calculatePost(dubler) == numeric_limits<float>::infinity())
 {
         cout << "x = " << i << "   f(x) = " << "---" << endl;
     }
     else {
         cout << "x = " << i << "   f(x) = " << calculatePost(dubler
) << endl;
     }
     if (i + h > xmax and i < xmax) {
         for (int j = 0; j < postfix.size(); j++) {
             if (postfix[j] == 'x') {
                 dubler.replace(j, 1, to_string(xmax));
             }
         }
         if (calculatePost(dubler) == std::numeric_limits<float>::
infinity()) {
             cout << "x = " << xmax << "   f(x) = " << "---" << endl
;
         }
         else {
```

```
                cout << "x = " << xmax << "    f(x) = " << calculatePost
    (dubler) << endl;
            }
        }
     }

     return 0;
}
```

На рисунках 1—9 представлены **примеры работы** описаннных выше алгоритмов.

1. Входные файлы: корректная строка в инфиксной записи.
   Вывод программы — ее постфиксная запись и значения. Результат приведён на рис. 1.

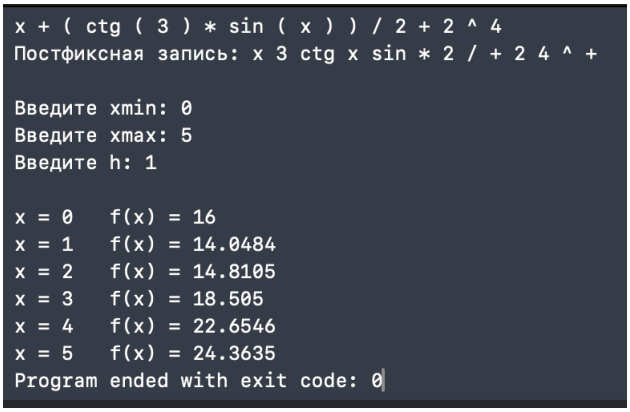

Рис. 1 – Пример работы 1

2. Входные файлы: корректная строка в инфиксной записи.
   Вывод программы — ее постфиксная запись и значения. Результат приведён на рис. 2.

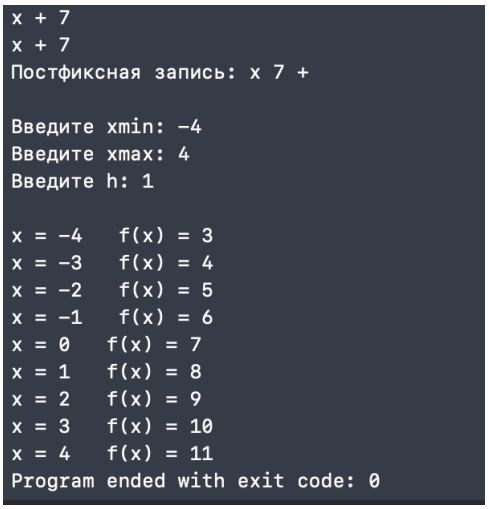

Рис. 2 – Пример работы 2

3. Входные файлы: корректная строка в инфиксной записи.
   Вывод программы — ее постфиксная запись и значения. Результат приведён на рис. 3.

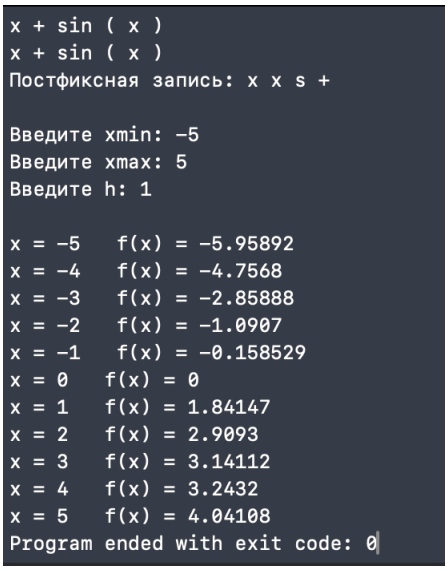

Рис. 3 – Пример работы 3

4. Входные файлы: корректная строка в инфиксной записи.
   Вывод программы — ее постфиксная запись и значения, причем в точке 0 котангенс не определен, поэтому и значение функции не определено в этой точке. Результат приведён на рис. 4.

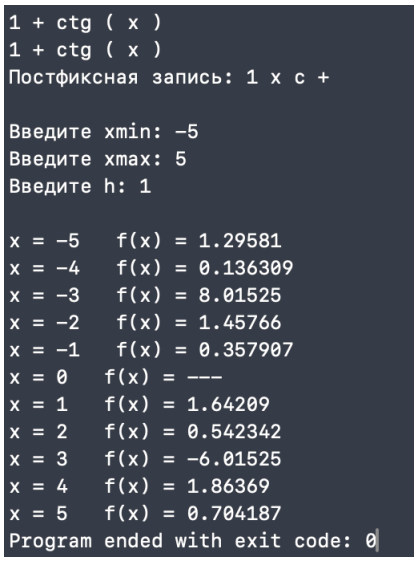

Рис. 4 – Пример работы 4

5. Входные файлы: некорректная строка в инфиксной записи, два знака подряд.
   Вывод программы — ошибка. Результат приведён на рис. 5.

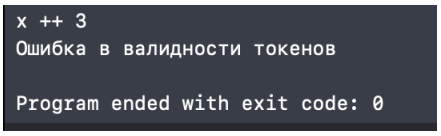

Рис. 5 – Пример работы 5

6. Входные файлы: некорректная с математической точки зрения строка, потому что котангенс 0 не определен.
   Вывод программы — все значения функции тоже не будут определены. Результат приведён на рис. 6.

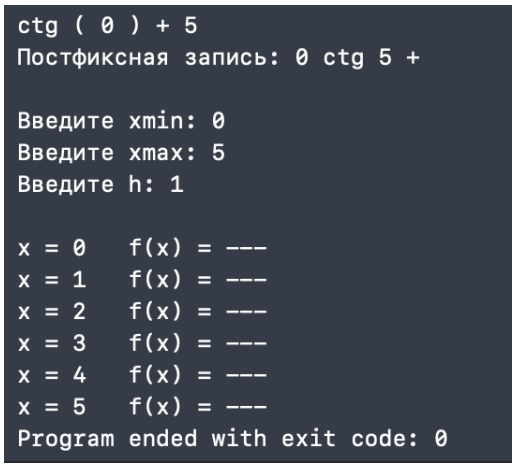

Рис. 6 – Пример работы 6

7. Входные файлы: строка, в которой неверно расставлены скобки.
   Вывод программы — ошибка. Результат приведён на рис. 7.

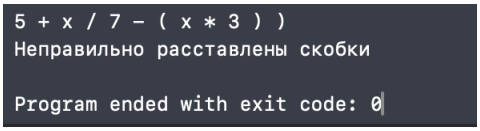

Рис. 7 – Пример работы 7

8. Входные файлы: строка, в которой квадратный корень берется из отрицательного числа.

   Вывод программы — ошибка. Результат приведён на рис. 8.

```
( - 3 ) ^ 0.5 + x - 1
Постфиксная запись: 0 3 - 0.5 ^ x + 1 -

Введите xmin: 0
Введите xmax: 5
Введите h: 1

Квадратный корень из отрицательного числа
x = 0   f(x) = ---
Квадратный корень из отрицательного числа
x = 1   f(x) = ---
Квадратный корень из отрицательного числа
x = 2   f(x) = ---
Квадратный корень из отрицательного числа
x = 3   f(x) = ---
Квадратный корень из отрицательного числа
x = 4   f(x) = ---
Квадратный корень из отрицательного числа
x = 5   f(x) = ---
Program ended with exit code: 0
```

Рис. 8 – Пример работы 8

9. Входные файлы: строка, в которой происходит деление на ноль.

   Вывод программы — ошибка. Результат приведён на рис. 9.

```
x ^ 4 + x ^ 2 + 1 / 0
Постфиксная запись: x 4 ^ x 2 ^ + 1 0 / +

Введите xmin: 0
Введите xmax: 7
Введите h: 1

x = 0   f(x) = ---
x = 1   f(x) = ---
x = 2   f(x) = ---
x = 3   f(x) = ---
x = 4   f(x) = ---
x = 5   f(x) = ---
x = 6   f(x) = ---
x = 7   f(x) = ---
Program ended with exit code: 0
```

Рис. 9 – Пример работы 9

Заключение

Цель достигнута: разработана программа для вычисления значений функции с использованием ОПЗ и стека. В результате выполнения лабораторной работы были выполнены все задачи.

1. Описаны инфиксная и префиксная формы записи.
2. Описан и реализован алгоритм перехода из инфиксной записи в префиксную запись.
3. Описан и реализован алгоритм поиска значения выражения в инфиксной записи.
4. Описан и реализован алгоритм проверки строки на корректность.

Список литературы

1. Фофанов О. Б., Алгоритмы и структуры данных: учебное пособие / Фофанов О. Б.; Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2014. – 126 с.