

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА

на тему:

**«Постфиксная форма записи арифметических
выражений»**

Выполнил: студент группы
3822Б1ФИ2
_____/Рысев М. Д./
Подпись

Проверил: к.т.н, доцент каф. ВВиСП
_____/Кустикова В.Д./
Подпись

Нижний Новгород
2023

Содержание

Введение.....	3
1 Постановка задачи.....	4
2 Руководство пользователя.....	5
2.1 Приложение для демонстрации работы стека.....	5
2.2 Приложение для демонстрации работы перевода арифметического выражения в постфиксную запись.....	5
3 Руководство программиста.....	7
3.1 Описание алгоритмов.....	7
3.1.1 Стек.....	7
3.1.2 Арифметическое выражение.....	8
3.2 Описание программной реализации.....	10
3.2.1 Описание класса TStack.....	10
3.2.2 Описание класса ArithmeticExpression.....	11
Заключение.....	14
Литература.....	15
Приложения.....	16
Приложение А. Реализация класса TStack.....	16
Приложение Б. Реализация класса Expression.....	17

Введение

Данная работа направлена на получение навыка преобразования инфиксной формы записи арифметического выражения в постфиксную. Постфиксная форма хороша тем, что по ней гораздо легче запрограммировать вычисление конечного результата выражения. Для реализации перехода из инфиксной формы в постфиксную в данной работе будет использоваться стек.

1 Постановка задачи

Цель:

Для начала реализовать шаблонный класс TStack. Затем и использованием стека реализовать класс перевода арифметического выражения в постфиксную форму.

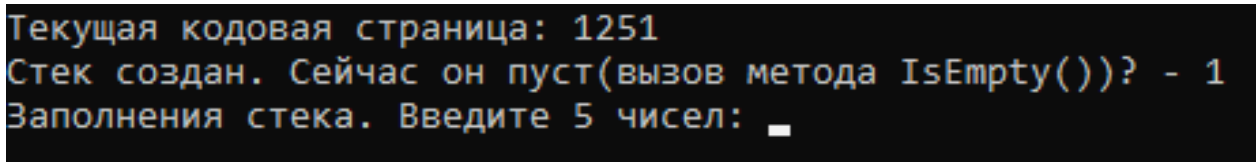
Задачи:

1. Реализация стека.
2. Изучение правил преобразования инфиксного выражения в постфиксное.
3. Написание программы, способной преобразовывать инфиксную форму в постфиксную и вычислять конечный результат.

2 Руководство пользователя

2.1 Приложение для демонстрации работы стека

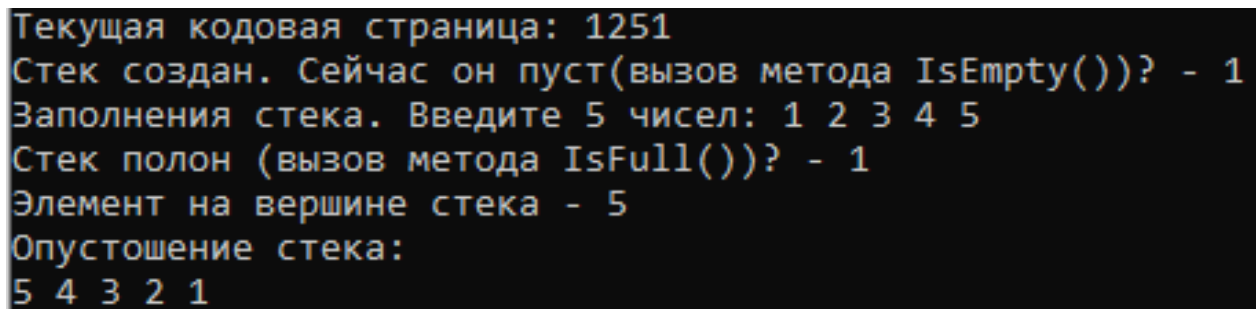
1. Запустите приложение с названием `sample_stack.exe`. В появившемся окне программа объявит о создании стека с вместимостью пять элементов. Также будет предложено ввести пять элементов. (рис. 1).



```
Текущая кодовая страница: 1251
Стек создан. Сейчас он пуст(вызов метода IsEmpty())? - 1
Заполнения стека. Введите 5 чисел: _
```

Рис. 1. Основное окно программы.

2. После ввода будет выведены результаты работы программы.(рис. 2).

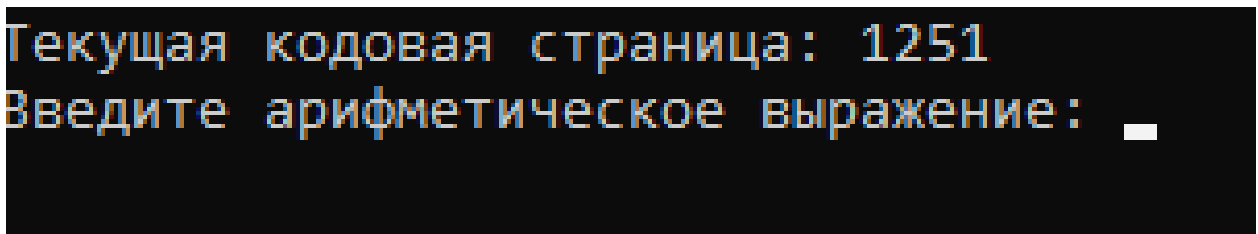


```
Текущая кодовая страница: 1251
Стек создан. Сейчас он пуст(вызов метода IsEmpty())? - 1
Заполнения стека. Введите 5 чисел: 1 2 3 4 5
Стек полон (вызов метода IsFull())? - 1
Элемент на вершине стека - 5
Опустошение стека:
5 4 3 2 1
```

Рис. 2. Результат работы программы.

2.2 Приложение для демонстрации работы перевода арифметического выражения в постфиксную запись

1. Запустите приложение с названием `sample_agexr.exe`. В результате появится окно, в котором будет предложено ввести арифметическое выражение (рис. 3).



```
Текущая кодовая страница: 1251
Введите арифметическое выражение: _
```

Рис. 3. Основное окно программы.

2. После ввода арифметического выражения будет выведена его постфиксная форма и будет предложено ввести значения переменных (если таковые имеются) для вычисления результата (рис. 4).

```
Текущая кодовая страница: 1251
Введите арифметическое выражение: a+2.5-(b+d)*5
a+2.5-(b+d)*5  --->  a2.5+bd+5*-
a = 2.5
b = 5
d = 2
вычисление выражения: -30
```

Рис. 4. Результат работы программы.

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Стек

Стек – это структура хранения, основанная на принципе «Last in, first out». Операции, доступные для стека: добавление элемента в вершину стека, взятие элемента с вершины стека, проверить элемент на вершине стека, проверка на полноту, проверка на пустоту.

Добавление элемента на вершину стека

Если стек не полон, то кладём новый элемент на его вершину.

Пример:

Операция добавления элемента со значением 3 в вершину стека:

	3
2	2
1	1

Взятие элемента с вершины стека.

Если стек не пуст, то берём элемент с вершины, при этом удаляя этот элемент из стека.

Пример:

Операция взятия элемента со значением 3 с вершины:

3	
2	2
1	1

$A = 3$ (взятый элемент можно положить в какую-нибудь переменную для последующего использования).

Просмотр элемента на вершине стека.

Просмотр значения верхнего элемента без его удаления из стека.

Пример:

Просмотр значения элемента с вершины стека:

3
2
1

Элемент на вершине равен 3.

Операция проверки на полноту.

Если стек полон вернёт 1, иначе - 0.

Пример:

3
2
1

Вернёт 1.

Операция проверки на пустоту.

Если стек пуст вернёт 1, иначе - 0.

Пример:

Вернёт 1.

3.1.2 Арифметическое выражение

Программа получает на вход некоторое арифметическое выражение. Затем она преобразовывает входное выражения в постфиксную форму и подсчитывает результат выражения.

Получение инфиксной записи.

На этом этапе нужно просто передать программе арифметическое выражение. Например, $A+B-C$.

Получение постфиксной записи.

Программа разбивает строку на лексемы, каждая из которых является либо операндом, либо оператором.

Алгоритм:

1. Создаем пустой стек операторов.
2. Создаем результирующую строку.
3. Проходим по каждому символу в инфиксной записи слева на право:

Если символ является операндом, добавляем его в результирующую строку.

Если символ является открывающей скобкой, помещаем его в стек операторов.

Если символ является закрывающей скобкой, извлекаем операторы из стека и добавляем их в результирующую строку до тех пор, пока не встретится открывающая скобка. Удаляем открывающую скобку из стека.

Если символ является оператором, то из вершины стека извлекаем операторы в результирующую строку до тех пор, пока не будет встречен оператор с меньшим приоритетом.

4. Извлекаем оставшиеся операторы из стека и добавляем их в результирующую строку.

Пример:

Выражение: $A + B * C$

Результирующая строка S (пока пуста).

1. $S = A$ 2. $S = A$ 3. $S = AB$ 4. $S = AB$ 5. $S = ABC$ 6. $S = ABC*+$

Stack: (пуст) Stack: + Stack: + Stack: +* Stack: +* Stack: (пуст)

Результат: $ABC*+$

Вычисление результата.

1. Проходим по каждому символу в постфиксной записи:

Если символ является операндом, помещаем его в стек операндов.

Если символ является оператором, извлекаем два операнда из стека, применяем оператор к этим операндам и помещаем результат обратно в стек.

2. После завершения прохода по всем символам, результат вычисления будет находиться на вершине стека операндов.

Полученное значение на вершине стека будет являться результатом вычисления постфиксной записи.

Пример:

Выражение: $A + B * C$

Постфиксная запись: $ABC*+$

Значение операндов: $A = 1, B = 2, C = 3$

1. Stack: 1 2. Stack: 1,2 3. Stack: 1,2,3 4. Stack 1,(2*3) 5. Stack: $1+(2*3) = 7$.

3.2 Описание программной реализации

3.2.1 Описание класса TStack

```
template <typedef T>
class TStack {
private:
    int MaxSize;
    int top;
    T* elems;
public:
    TStack(int size = 10);
    TStack(const TStack<T>& stack);
    ~TStack();
    T Top();
    bool IsEmpty() const;
    bool IsFull() const;
    void Push(const T& elem);
    T Pop() const;
};
```

Назначение: представление стека .

Поля:

MaxSize – максимальный размер стека.

elems – память для представления элементов стека.

top – индекс вершины стека (-1, если стек пустой).

Методы:

TStack(int size = 10);

Назначение: конструктор по умолчанию/конструктор с параметрами.

Входные параметры:

size – максимальный размер стека.

Выходные параметры: отсутствуют.

TStack(const TStack<T>& stack);

Назначение: конструктор копирования.

Входные параметры:

stack – копируемы стек.

Выходные параметры: отсутствуют.

~TStack();

Назначение: деструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

T Top() const;

Назначение: копирование элемента на вершине стека.

Входные параметры отсутствуют.

Выходные параметры: элемент с вершины стека.

bool IsEmpty() const;

Назначение: проверка на пустоту.

Входные параметры: отсутствуют.

Выходные параметры: 1, если стек пуст, 0 иначе.

bool IsFull() const;

Назначение: проверка на полноту.

Входные параметры: отсутствуют.

Выходные параметры: 1, если стек полон, 0 иначе.

void Push(const T& elem);

Назначение: добавление элемента в стек.

Входные параметры:

elem – добавляемый элемент.

Выходные параметры отсутствуют.

T Pop();

Назначение: удаление элемента из вершины стека.

Входные параметры отсутствуют.

Выходные параметры: элемент с вершины стека.

3.2.2 Описание класса **ArithmeticExpression**

```
using namespace std;
```

```
class ArithmeticExpression {
```

```
private:
```

```
    string arexp;
```

```
    vector<string> mas;
```

```
    vector<string> post_form;
```

```
    map<string, double> values;
```

```
    bool IsOperation(const string& symbol) const noexcept;
```

```
    bool IsOperation(const char& symbol) const noexcept;
```

```
    bool IsInvalidSign(const char& symbol) const noexcept;
```

```
    void GetTokens();
```

```
    void GetValues();
```

```
    bool IsParam(const string& tok) const noexcept;
```

```
    bool IsConst(const string& tok) const noexcept;
```

```
public:
```

```
    ArithmeticExpression();
```

```
    ArithmeticExpression(const string& str);
```

```
    void ToPostfixForm();
```

```
    float Calculate();
```

```
    string InfixForm() const;
```

```
    string PostfixForm() const;
```

```
};
```

Назначение: работа с инфиксной формой записи арифметических выражений

Поля:

arexp – инфиксная форма арифметического выражения.

mas – вектор лексем.

post_form – вектор лексем, записанных в постфиксной форме.

values – словарь со значениями переменных.

Методы:

bool IsOperation(const string& symbol) const noexcept;

Назначение: определение является ли лексема оператором.

Входные параметры:

symbol – лексема типа данных string.

Выходные параметры: 1, если оператор, 0 иначе.

bool IsOperation(const char& symbol) const noexcept;

Назначение: определение является ли лексема оператором.

Входные параметры:

symbol – лексема типа данных char.

Выходные параметры: 1, если оператор, 0 иначе.

bool IsValidSign(const char& symbol) const noexcept;

Назначение: проверка валидности знака.

Входные параметры:

symbol – символ для проверки.

Выходные параметры: 1, если валидный, 0 иначе.

void GetTokens () ;

Назначение: разбиение выражения на лексемы.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

void GetValues () ;

Назначение: получение значений переменных

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

bool IsParam(const string& tok) const noexcept;

Назначение: проверка является ли лексема переменной

Входные параметры:

tok – лексема для проверки.

Выходные параметры: 1, если переменная, 0 иначе

bool IsConst(const string& tok) const noexcept;

Назначение: проверка является ли лексема константой.

Входные параметры:

tok – лексема для проверки.

Выходные параметры: 1, если константа, 0 иначе.

ArithmeticExpression () ;

Назначение: конструктор.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

ArithmeticExpression(const string& str);

Назначение: конструктор с параметром.

Входные параметры:

str – выражение для преобразования.

Выходные параметры: отсутствуют.

void ToPostfixForm();

Назначение: конвертация в постфиксную форму.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

float Calculate();

Назначение: вычисление результата.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

string InfixForm() const;

Назначение: вывод оригинальной формы выражения.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

string PostfixForm() const;

Назначение: возвращение инфиксной формы.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

Заключение

В ходе выполненной работы были изучены принципы работы со стеками. Реализован шаблонный класс стека.

Изучен алгоритм преобразования инфиксной формы выражения в постфиксную и реализован класс, хранящий в себе арифметическое выражение, преобразовывающий его в инфиксную форму и вычисляющий его результат.

Литература

1. Польская запись [https://ru.wikipedia.org/wiki/Польская_запись].

Приложения

Приложение А. Реализация класса TStack

```
template <typename T>
class TStack {
private:
    int MaxSize;
    int top;
    T* elems;
public:
    TStack(int size = 10);
    TStack(const TStack<T>& stack);
    ~TStack();
    bool IsEmpty() const;
    bool IsFull() const;
    T Pop();
    void Push(const T& elem);
    T Top() const;
};

template <typename T>
TStack<T>::TStack(int size) {
    if (size < 0) throw "out_of_range";
    MaxSize = size;
    top = -1;
    elems = new T[MaxSize];
}

template <typename T>
TStack<T>::TStack(const TStack<T>& stack) {
    MaxSize = stack.MaxSize;
    top = stack.top;
    elems = new T[MaxSize];
    for (int i = 0; i <= top; i++) elems[i] = stack.elems[i];
}

template <typename T>
TStack<T>::~~TStack() {
    delete[] elems;
}

template <typename T>
bool TStack<T>::IsEmpty() const {
    return (top == -1);
}

template <typename T>
bool TStack<T>::IsFull() const {
    return (top == MaxSize - 1);
}

template <typename T>
T TStack<T>::Pop() {
    if (IsEmpty()) throw "stack_is_empty";
    return elems[top--];
}

template <typename T>
void TStack<T>::Push(const T& elem) {
    if (IsFull()) throw "stack_is_full";
    elems[++top] = elem;
}

template <typename T>
T TStack<T>::Top() const{
    if (IsEmpty()) throw "stack_is_empty";
    return elems[top];
}
```


Приложение Б. Реализация класса Expression

```
map <string, int> pr = {
    {"*", 3},
    {"/", 3},
    {"+", 2},
    {"-", 2},
    {"(", 1},
};

ArithmeticExpression::ArithmeticExpression() {
    cin >> arexp;
    arexp += '\0';
    GetTokens();
    ToPostfixForm();
}

ArithmeticExpression::ArithmeticExpression(const string& str) {
    arexp = str;
    arexp += '\0';
    GetTokens();
    ToPostfixForm();
}

bool ArithmeticExpression::IsOperation(const string& symbol) const noexcept {
    switch (symbol[0])
    {
        case '+': return true;
        case '-': return true;
        case '/': return true;
        case '*': return true;
        case '(': return true;
        case ')': return true;
        default:
            return false;
    }
}

bool ArithmeticExpression::IsOperation(const char& symbol) const noexcept {
    switch (symbol)
    {
        case '+': return true;
        case '-': return true;
        case '/': return true;
        case '*': return true;
        case '(': return true;
        case ')': return true;
        default:
            return false;
    }
}

bool ArithmeticExpression::IsValidSign(const char& symbol) const noexcept {
    return !IsOperation(symbol) && !isdigit(symbol) && !isalpha(symbol) &&
    symbol != '.';
}

void ArithmeticExpression::GetTokens() {
    int i = 0;
    while (arexp[i] != '\0') {
        string tok = "";
        char symb = arexp[i];
        while (!IsOperation(symb) && symb != '\0' && symb != ' ') {
            tok += symb;
            i += 1;
            symb = arexp[i];
        }
        if (tok != "") mas.push_back(tok);
        if (IsOperation(symb)) {
```

```

        string c(1, symb);
        mas.push_back(c);
    }
    i += 1;
}

void ArithmeticExpression::GetValues() {
    for (const string& i : mas) {
        if (!IsOperation(i) && !IsConst(i) && !values.count(i)) {
            double val;
            cout << i << " = ";
            cin >> val;
            values[i] = val;
        }
        else if (IsConst(i)) values[i] = stod(i);
    }
}

bool ArithmeticExpression::IsParam(const string& tok) const noexcept{
    if (isdigit(tok[0])) return false;
    int i = 0;
    while (tok[i] != '\\0'){
        if (IsValidSign(tok[i]) || IsOperation(tok[i])) return false;
        i += 1;
    }
    return true;
}

bool ArithmeticExpression::IsConst(const string& tok) const noexcept {
    int i = 0;
    while (tok[i] != '\\0') {
        if (IsValidSign(tok[i]) || isalpha(tok[i]) ||
            IsOperation(tok[i])) return false;
        i += 1;
    }
    return true;
}

void ArithmeticExpression::ToPostfixForm() {
    TStack<string> oper(mas.size());
    for (const string& i : mas) {
        if (IsConst(i) || IsParam(i)) post_form.push_back(i);
        else if (IsOperation(i)) {
            if (i == "(") oper.Push(i);
            else if (i == ")") {
                while (oper.Top() != "(")
                    post_form.push_back(oper.Pop());
                if (oper.IsEmpty()) throw "not found left bracket";
                else oper.Pop();
            }
            else if (oper.IsEmpty() || pr[oper.Top()] < pr[i])
                oper.Push(i);
            else {
                while (!oper.IsEmpty() && pr[oper.Top()] >= pr[i])
                    post_form.push_back(oper.Pop());
                oper.Push(i);
            }
        }
        else throw ("invalid name %s", i);
    }
    while (!oper.IsEmpty()) {
        if (oper.Top() == "(") throw "right bracket not found";
        post_form.push_back(oper.Pop());
    }
}

float ArithmeticExpression::Calculate() {

```

```

    GetValues();
    TStack<float> st(post_form.size());
    st.Push(0);
    for (const string& i : post_form) {
        if (!IsOperation(i)) st.Push(values[i]);
        else {
            switch (i[0]) {
                case '+': {
                    float b = st.Pop(), a = st.Pop();
                    st.Push(a + b);
                    break;
                }
                case '*': {
                    float b = st.Pop(), a = st.Pop();
                    st.Push(a * b);
                    break;
                }
                case '-': {
                    float b = st.Pop(), a = st.Pop();
                    st.Push(a - b);
                    break;
                }
                case '/': {
                    float b = st.Pop(), a = st.Pop();
                    st.Push(a / b);
                    if (b == 0) throw "Division by zero";
                    break;
                }
            }
        }
    }
    return st.Pop();
}

string ArithmeticExpression::InfixForm() const{
    string str = "";
    for (string i : mas) {
        str += i;
    }
    return str;
}

string ArithmeticExpression::PostfixForm() const{
    string str = "";
    for (string i : post_form) {
        str += i;
    }
    return str;
}

```