МИНИСТЕРСТВО НАУКИ И ВЫСШЕГООБРАЗОВАНИЯ
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:

«Стек. Постфиксная форма»

Выполнила: студентка группы 3822Б1ФИ2
_____ / Резанцева А.А.
Подпись

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

Нижний Новгород
2023

Содержание

Введение

Целью данной лабораторной работы является реализовать стек, и для создания этой программы понадобится создать класс с шаблоном и различными функциями.

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (last in — first out, «последним пришёл — первым вышел»).

Область применения:

Программный вид стека используется для обхода структур данных, например, дерево или граф. При использовании рекурсивных функций также будет применяться стек, но аппаратный его вид. Кроме этих назначений, стек используется для организации стековой машины, реализующей вычисления в обратной инверсной записи.

Для отслеживания точек возврата из подпрограмм используется стек вызовов.

Применение стека упрощает и ускоряет работу программы, так как идет обращение к нескольким данным по одному адресу.

Обратная по́льская запись — форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Также именуется как обратная польская запись, обратная бесскобочная запись, постфиксная нотация, бесскобочная символика Лукасевича, польская инверсная запись, ПОЛИЗ.

Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) расположены перед знаком операции.

Постфиксная форма используется для оптимизации вычислений, так как помогает избежать проблем с приоритетом операторов и порядков операций, делая выражения более однозначными.

1 Постановка задачи.

Цель – реализовать шаблонный класс для представления стека TStack, и на его основе реализовать класс Postfix, содержащий функции для представления арифметического выражения с его постфиксной формой.

Задачи при реализации класса TStack:

1. Описать и реализовать конструктор, конструктор копирования, деструктор.
2. Описать и реализовать методы проверки заполнен ли стек и пуст ли стек.
3. Описать и реализовать методы добавления элемента на верхушку стека, удаление элемента с верхушки стека, получить элемент с верхушки стека.

Задачи при реализации класса Postfix:

1. Описать и реализовать конструктор и конструктор копирования.
2. Описать и реализовать метод парсинга арифметического выражения для поиска операторов, операндов и констант.
3. Реализовать алгоритм перевода арифметического выражения в его постфиксную форму.
4. Реализовать алгоритм вычисления значения арифметического выражения по его постфиксной форме.
5. Описать и реализовать разные способы ввода значений арифметического выражения.
6. Описать и реализовать метод получения постфиксной формы арифметического выражения.

Предоставить пример использования и обеспечить работоспособность тестов, покрывающих все методы классов TStack и Postfix.

2 Руководство пользователя.

2.1 Приложение для демонстрации работы стека.

1. Запустите приложение с названием sample_tstack.exe. В результате появится окно, показанное ниже (рис. 1).

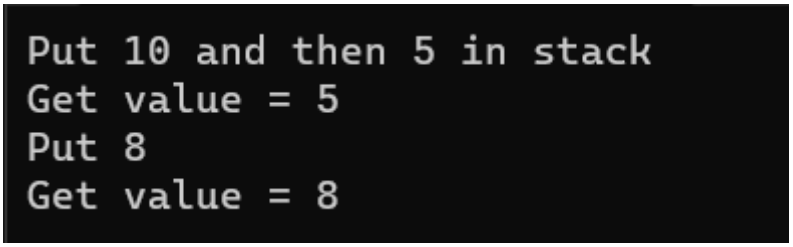

Рис. 1. Основное окно программы.

2. На первом шаге создается стек с элементами 10 и 5 (рис. 2).

Рис. 2. Создание стека с элементами 10 и 5.

3. На следующем шаге вызываем метод для получения значения элемента на верхушке стека и его удаления (метод Pop)(рис. 3).

Рис. 3. Получение значения элемента на верхушке стека.

4. Далее используем функцию Push для назначения нового элемента стека на его верхушку (рис. 4).

Рис. 4. Назначение элемента 8 на верхушку стека

5. На четвертом шаге проверяем метод Push(8), получением значения верхушки стека (рис. 5).

Рис. 5. Проверка метода Push(8) и Pop.

2.2 Приложение для демонстрации работы алгоритма перевода в постфиксную запись.

1. Запустите приложение с названием sample_talgorithm.exe. В результате появится окно, показанное ниже (рис. 6).

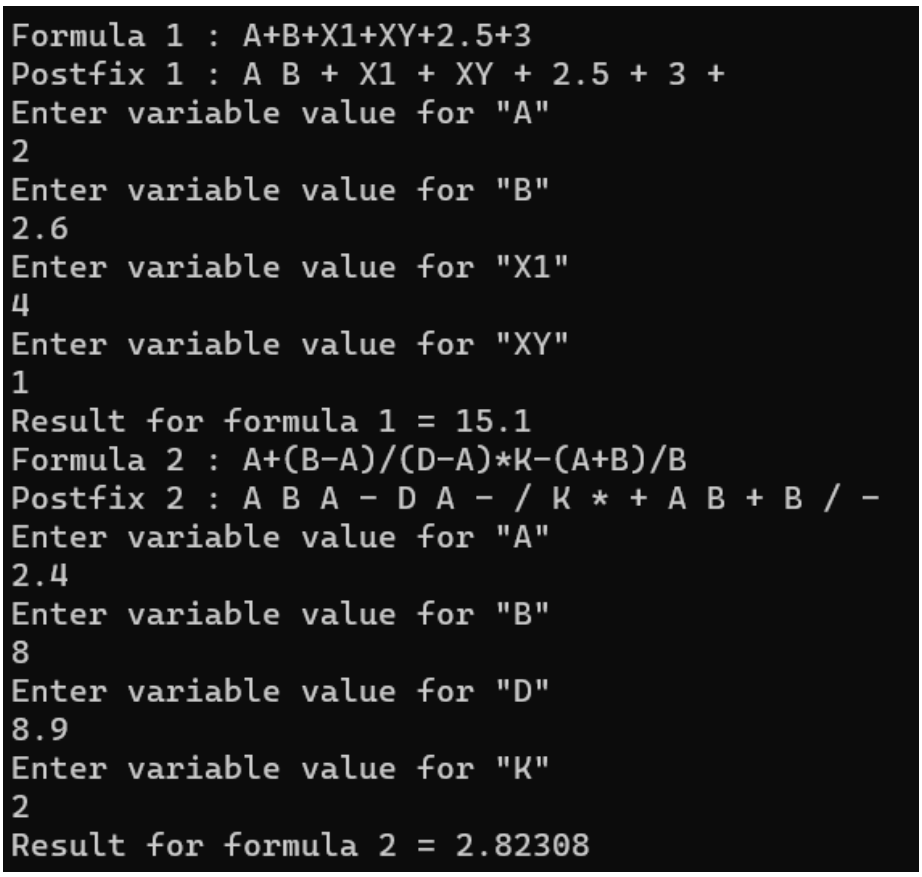

Рис. 6. Основное окно программы

2. На первом шаге строка Formula 1 преобразуется в постфиксную форму Postfix 1(рис. 7).

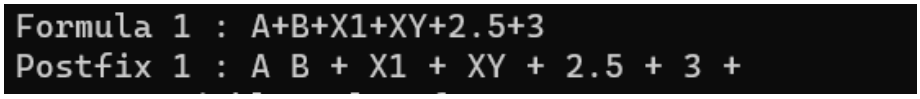

Рис. 7. Постфиксная запись Formula 1.

3. Далее пользователю предлагается ввести значения для переменных из Formula 1. Введены следующие значения: A=2.4, B=8, X1=4, XY=1 (рис. 8).

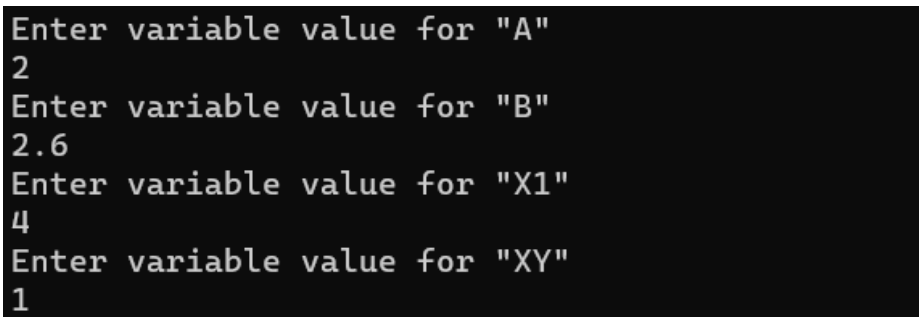

Рис. 8. Ввод значений переменных для Postfix 1.

4. Получаем результат вычисления строки в постфиксной форме (рис. 9).

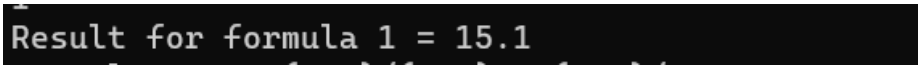

Рис. 9. Результат вычисления постфиксной формы Formula 1.

5. Далее преобразуется строка Formula 2 в постфиксную форму Postfix 2 (рис. 10).

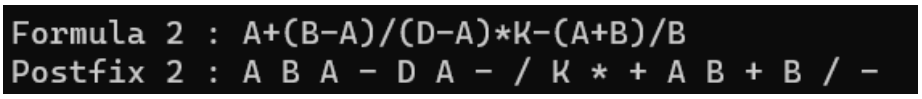

Рис. 10. Преобразование в постфиксную форму Formula 2.

6. Далее пользователю предлагается ввести значения для переменных из Formula 2. Введены следующие значения: A=2.4, B=8, D=8.9, K=2 (рис. 11).

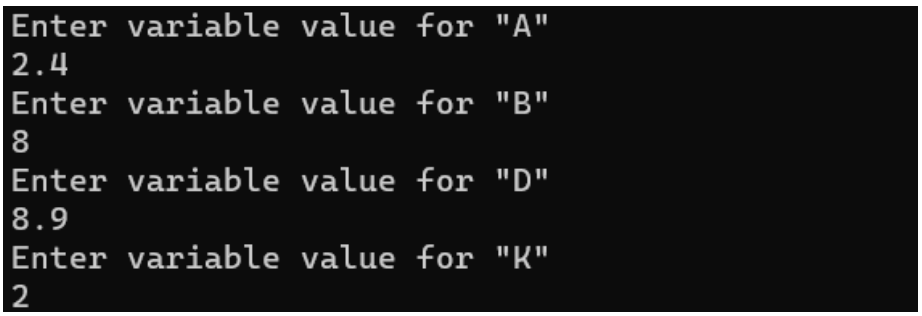

Рис. 11. Ввод значений переменных для Postfix 2.

7. Получаем результат вычисления строки Postfix 2 (рис. 12).

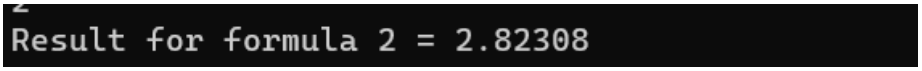

Рис. 12. Результат вычисления постфиксной формы Formula 2.

3 Руководство программиста.

3.1 Описание алгоритмов.

3.1.1 Стек.

Стек представляет собой абстрактную структуру данных, организованную по принципу "последний вошел - первый вышел" (Last In, First Out, LIFO). Это означает, что элементы добавляются и удаляются только с одного конца стека, который называется вершиной.

Операции, которые можно выполнять со стеком, включают добавление элемента (push) на вершину стека и удаление элемента (pop) с вершины стека. При этом доступ к остальным элементам стека осуществлять нельзя, кроме верхнего элемента.

Операция получения размера стека

5	12	3	2	

Длина стека равна 4.

Операция добавления в стек

Для этого используется флаг top, в данный момент top=3

5	12	3	2	

Добавим элемент 13

Результат:

5	12	3	2	13

top=4,

Операция удаления с вершины

Для этого также воспользуемся флагом top=4

5	12	3	2	13

Результат:

5	12	3	2	

Флаг top теперь стал равен 3.

Операция проверки на пустоту

5	12	3	2	13

Результат: false – стек не пуст

Результат: true – стек пуст

Операция проверки на полноту

5	12	3	2	13

Результат: true – стек полон

5	12	3	2	

Результат: false – стек не полон

Операция проверки последнего элемента

5	12	3	2	13

Результат: 13

3.1.2 Перевод в постфиксную запись.

1. Для операций вводится приоритет:

3: *, /

2: +, -

1: (

0: =

2. Для хранения данных используются два стека: первый для хранения операндов и результатов, второй для хранения операций.
3. Исходное выражение просматривается слева на право пока не дойдем до его конца.

3.1.Операнды по мере их появления помещаются в стек 1.

3.2.Символы операции (помещаются в стек 2 по следующему правилу:

3.2.1. При появлении) последовательно изымаются элементы (операции) из стека 2 и переносятся в стек 1 пока стек 2 не пуст или не обнаружена (.

3.2.2. Левая скобка кладется в стек 2.

3.2.3. Если текущая операция, полученная при обходе выражения имеет более низкий приоритет операции, чем на вершине стека 2, то все операции, приоритет которых больше или равен приоритету текущей операции перекладываются из стека 2 в стек 1, текущая операция переносится в стек 2.

4. По завершению выражения операции из стека 2 перекладываются в стек 1, пока он не пуст.

На выходе строка.

Пример.

A+(B-A)*D-F/(G+H)

Стек 1

																	/
																+	+
															H	H	H
													G	G	G	G	G
										F	F	F	F	F	F	F	F
									+	+	+	+	+	+	+	+	+
									*	*	*	*	*	*	*	*	*
								D	D	D	D	D	D	D	D	D	D
						-	-	-	-	-	-	-	-	-	-	-	-
						C	C	C	C	C	C	C	C	C	C	C	C
			B	B	B	B	B	B	B	B	B	B	B	B	B	B	B
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Стек 2

														+	+		

				-	-							(	(	(	(		
				(	(		*	*			/	/	/	/	/	/	
	+	+	+	+	+	+	+	+	-	-	-	-	-	-	-	-	

Результат: ABC-D*+FGH+/-

3.1.3 Алгоритм вычисления значения выражения, записанного в постфиксной форме.

Входные данные: строка, содержащая арифметическое выражение в постфиксной форме, множество пар.

Выходные данные: вещественное значение, равное результату вычисления выражения.

Алгоритм:

1. Пока не достигнут конец входной последовательности, читаем очередную лексему.
   1.1. Если прочитан операнд, то его значение помещается в стек.
   1.2. Если прочитана операция, то из стека изымаются значения двух последних операндов, выполняется операция над этими операндами, в обратном порядке поступления операндов в стек. Результат выполнения операции кладется обратно в стек.
2. Если достигнут конец входной последовательности, то в стеке хранится результат вычисления.

Пример.

Инфиксная запись: A+(B-A)*D-F/(G+H).

Постфиксная запись: ABC-D*+FGH+/-

Значения переменных:

A	0
B	1
C	2
D	-1
F	2
G	0,5
H	0,5

Стек:

									0,5				

		2		-1				0,5	0,5	1			
	1	1	-1	-1	1		2	2	2	2	2	2	
0	0	0	0	0	0	1	1	1	1	1	1	1	-1

Результат: -1

3.2 Описание программной реализации.

3.2.1 Описание класса TStack.

```
template <typename ValueType> class TStack {
private:
      int maxSize;
      int top;
      ValueType* elems;
public:
      TStack(int maxSize = 10);
      TStack(const  TStack<ValueType>& s, int size = 0);
      ~TStack();
      bool isEmpty(void) const;
      bool isFull (void) const;
      ValueType Top() const;
      void Push(const ValueType& e);
      ValueType Pop();
};
```

Назначение: представление стека.

Поля:

maxSize – количество доступной памяти, размер стека.

top – индекс верхнего элемента в стеке.

elems – память для представления стека.

Методы:

```
TStack(int maxSize = 10);
```

Назначение: конструктор по умолчанию и конструктор с параметрами.

Входные параметры: **maxSize** – количество выделяемой памяти.

```
TStack(const  TStack<ValueType>& s, int size = 0);
```

Назначение: конструктор копирования.

Входные параметры: **s** – экземпляр класса **TStack**, который нужно скопировать.

```
~TStack();
```

Назначение: освобождение выделенной памяти.

```
bool isEmpty(void) const;
```

Назначение: проверка на пустоту стека.

Выходные параметры: **true** – стек пуст, **false** в противном случае.

```
bool isFull (void) const;
```

Назначение: проверка на полноту стека.

Выходные параметры: **true** – стек стек заполнен, **false** в противном случае.

```
ValueType Top() const;
```

Назначение: метод, возвращающий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: элемент, который находится на вершине стека.

```
void Push(const ValueType& e);
```

Назначение: метод, помещающий элемент на вершинку стека.

Входные параметры: **e** – элемент, который требуется добавить на вершину стека.

Выходные параметры: отсутствуют.

```
ValueType Pop();
```

Назначение: метод, удаляющий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

3.2.2 Описание класса Postfix

```
class Postfix
{
    friend class PostfixTest;
private:
    std::vector<std::string> postfixArray;
    std::string s_postfix;
    std::map<std::string, double> operands;

    double calculateOperator(char operator_, double a, double b) const;
    int getCountNotSpecified();

    int GetPriority(const char op);
    bool isOperator(char c);
    bool isOperator(const std::string& str);
    std::pair<std::string, int> getOperand(const std::string& s, int pos);

public:
    Postfix();
    Postfix(const std::vector<std::string>& postf);
    std::string getStringView() const
    {
        return s_postfix;
    }
    void setValues();
    double getValue(const std::string& name);
```

```
    double calculate();
    void setValuesFromVector(const std::vector<double>& values);
    Postfix ConvertToPol(const std::string& s);

    friend std::ostream& operator<<(std::ostream& out, const Postfix& op);
};
```

Назначение: хранение выражения в постфиксной форме в виде вектора операндов и вычисления значения выражения.

Поля:

postfixArray – вектор строк, хранящий постфиксный вид строки.

s_postfix – хранит строковое представление постфиксное записи.

operands – мапа, хранящая имена операндов выражения и их значения.

Методы:

```
double calculateOperator(char operator_, double a, double b) const;
```

Назначение: метод распознавания и выполнения арифметических операций.

Входные параметры: **operator_** – символ оператора, **a, b** – операнды, над которыми выполнится операция.

Выходные параметры: результат выполнения соответствующей арифметической операции.

```
int getCountNotSpecified();
```

Назначение: метод подсчета операндов, значение которых не задано.

Выходные параметры: количество операндов.

```
int GetPriority(const char op);
```

Назначение: метод установки приоритета арифметических операций.

Входные параметры: **op** – символ оператора.

Выходные параметры: приоритет операции.

```
bool isOperator(char c);
```

Назначение: метод определения арифметической операции.

Входные параметры: **c** – символ выражения.

Выходные параметры: символ операции.

```
bool isOperator(const std::string& str);
```

Назначение: метод определения арифметической операции.

Входные параметры: **str** – строка выражения.

Выходные параметры: строка операции.

```
std::pair< std::string, int> getOperand(std::string s, int pos);
```

Назначение: метод определения имени операнда.

Входные параметры: **s** – строка с выражением, **pos** – позиция начала операнда.

Выходные параметры: имя операнда и его позиция.

```
Postfix();
```

Назначение: конструктор по умолчанию.

Выходные параметры: возвращает сконструированный объект класса **Postfix**, состоящий из нулевого вектора операндов и пустой строки.

```
Postfix(const std::vector<std::string>& postf);
```

Назначение: конструктор с параметром.

Входные параметры: **postf** – вектор строк, хранящий постфиксную запись.

Выходные параметры: возвращает сконструированный объект класса **Postfix**

```
std::string getStringView();
```

Назначение: метод получения постфиксной формы в виде строки.

Выходные параметры: строка, хранящая представление постфиксной записи.

```
void setValues();
```

Назначение: метод установки значения для одного операнда.

```
void getValue();
```

Назначение: метод получения значения для всех операндов от пользователя.

Выходные параметры: **value** – значение операнда.

```
double calculate();
```

Назначение: метод вычисления выражения в постфиксной форме.

Выходные параметры: элемент, находящийся на верхушке стека, который является результатом вычислений.

```
void setValuesFromVector(const std::vector<double>& values);
```

Назначение: метод установки значений операндов из вектора.

Входные параметры: **values** – вектор значений операндов.

```
Postfix ConvertToPol(const std::string& s);
```

Назначение: метод перевода в постфиксную форму.

Входные параметры: **`s`** – строка с выражением.

Выходные параметры: объект класса **`Postfix`**.

```
friend std::ostream& operator<<(std::ostream& out, const Postfix& op);
```

Назначение: оператор вывода объекта класса **`Postfix`**.

Входные параметры: **`out`** – поток вывода, **`op`** – ссылка на операнд, который выводим.

Выходные параметры: поток вывода.

Заключение

В рамках работы был разработан шаблонный класс для реализации стека, который поддерживает основные операции, такие как добавление, удаление и получения элемента с верхушки стека, методы проверки на полноту и пустоту стека. Также был разработан класс для реализации алгоритмов перевода выражения в постфиксную форму и вычисления её значения.

Литература

1. Лекция «Динамическая структура данных Стек» Сысоев А.В. [https://cloud.unn.ru/s/jXmxFzAQoTDGfNe]
2. Лекция «Разбор и вычисление арифметических выражений с помощью постфиксной формы» Сысоев А.В. https://cloud.unn.ru/s/4Pyf24EBmowGsQ2
3. Стуктура pair. [https://learn.microsoft.com/ru-ru/cpp/standard-library/pair-structure?view=msvc-170]
4. Класс vector. [https://learn.microsoft.com/ru-ru/cpp/standard-library/vector-class?view=msvc-170]
5. Класс map. [https://learn.microsoft.com/ru-ru/cpp/standard-library/map-class?view=msvc-170]

Приложения

Приложение А. Реализация класса TStack.

```
#ifndef TSTACK_H
#define TSTACK_H


template <typename ValueType> class TStack {
private:
  int maxSize;
  int top;
  ValueType* elems;
public:
  TStack(int maxSize = 10);
  TStack(const  TStack<ValueType>& s, int size = 0);
  ~TStack();
  bool isEmpty(void) const;
  bool isFull (void) const;
  ValueType Top() const;
  void Push(const ValueType& e);
  ValueType Pop();
};

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

template <typename ValueType> bool TStack<ValueType>::isEmpty(void) const
{
  return top == -1;
}

template <typename ValueType> bool TStack<ValueType>::isFull(void) const
{
  return top == maxSize - 1;
}

template<typename ValueType> TStack<ValueType>::TStack(int maxSize)
{
  if (maxSize <= 0)
    throw "Wrong size";
  top = -1;
  this->maxSize = maxSize;
  elems = new ValueType[maxSize];
}

template<typename          ValueType>          TStack<ValueType>::TStack(const
TStack<ValueType>& s, int size)
{
  if (size < 0) {
    throw "Error: Stack size less than 0";
  }
  maxSize = s.maxSize + size;
  top = s.top;
  elems = new ValueType[maxSize];
  for (int i = 0; i <= top; i++)
  {
    elems[i] = s.elems[i];
  }
}
```

```
template<typename ValueType> ValueType TStack<ValueType>::Top() const
{
  if (top == -1)
    throw "Stack is empty";
  return elems[top];
}
template<typename ValueType> ValueType TStack<ValueType>::Pop() {
  if (isEmpty())
    throw "Stack is empty";
  return elems[top--];
}
template<typename ValueType> void TStack<ValueType>::Push(const ValueType& e)
{
  if (isFull())
    throw "Stack is full";
  elems[++top] = e;
}

#endif
```

Приложение Б. Реализация класса Postfix.

```
#include "talgorithm.h"
#include "tstack.h"
#include <iostream>
#include <string>
#include <stdexcept>

int Postfix::GetPriority(const char op)
{
    switch (op)
    {
    case '(':
        return 1;
    case ')':
        return 1;
    case '+':
        return 2;
    case '-':
        return 2;
    case '*':
        return 3;
    case '/':
        return 3;
    default:
        throw "Error. Incorrect symbol.";
    }
}

bool Postfix::isOperator(char c) {
    return (c == '+'  c == '-'  c == '*'  c == '/'  c == '('  c == ')');
}

bool Postfix::isOperator(const std::string& str) {
    return (str == "+"  str == "-"  str == "*"  str == "/"  str == "("  str
== ")");
}

// cout for struct Postrix
std::ostream& operator<<(std::ostream& out, const Postfix& op)
{
    out << op.s_postfix;
    return out;
```

```
};

//for class Postfix

Postfix::Postfix()
{
    postfixArray = std::vector<std::string>();
    s_postfix = "";
    operands = std::map<std::string, double>();
}

//переделать, принимать строку и разбивать на лексемы(операнды)
Postfix::Postfix(const std::vector<std::string>& postf)
{
    postfixArray = postf;
    s_postfix = "";
    for (int i = 0; i < postf.size(); i++)
    {
        s_postfix += postf[i];
        s_postfix += " ";

        if     (!isOperator(postf[i])     &&     operands.find(postf[i])     ==
operands.end()) {
            double value = INFINITY;

            try {
                value = std::stof(postf[i]);
            }
            catch (std::invalid_argument const& ex)
            {
            };

            // put new unique operand to map
            operands[postf[i]] = value;
        }
    }
}

void Postfix::setValues()
{
    for (auto& operand : operands)
    {
        if (operand.second == INFINITY) {
            operand.second = getValue(operand.first);
        }
    }
};

double Postfix::getValue(const std::string& name)
{
    std::cout << "Enter variable value for \"" << name << "\"" << std::endl;
    double value = INFINITY;
    std::cin >> value;

    return value;
};

double Postfix::calculateOperator(char operator_, double a, double b) const
{
    switch (operator_)
    {
    case '+':
        return a + b;
```

```
    case '-':
        return a - b;
    case '*':
        return a * b;
    case '/':
        return a / b;
    }
}

double Postfix::calculate()
{
    TStack<double> S; //store value
    for (int i = 0; i < postfixArray.size(); i++)
    {
        if (!isOperator(postfixArray[i])) // if operand
        {
            S.Push(operands[postfixArray[i]]);
        }
        else // if operator
        {
            double b = S.Pop();
            double a = S.Pop();
            S.Push(calculateOperator(postfixArray[i][0], a, b));
        }
    }

    return S.Pop();
}

// Get string name of operand in source string s
std::pair<std::string, int> Postfix::getOperand(const std::string& s, int
pos)
{
    std::string operandName = "";

    while (pos < s.size() && !isOperator(s[pos]))
    {
        operandName += s[pos];
        pos++;
    }

    return {operandName, pos};
}
Postfix Postfix::ConvertToPol(const std::string& s)
{
    int open = 0;
    int close = 0;
    TStack<std::string> S1(s.size());  //store operand
    TStack<char> S2(s.size());  // store operations
    for (int i = 0; i < s.size(); i++)
    {
        if (!isOperator(s[i]))
        {
            std::pair<std::string, int> op = getOperand(s, i);
            S1.Push(op.first);
            i = op.second - 1;
        }
        else if(isOperator(s[i]))
        {
            if (S2.isEmpty() || s[i] == '(')
            {
                S2.Push(s[i]);
                continue;
```

```
        }
        if (s[i] == ')')
        {
            while (!S2.isEmpty())
            {
                char operation = S2.Pop();
                if (operation != '(')
                {
                    S1.Push(std::string(1, operation));
                }
                else
                {
                    break;
                }
            }
        }
        else
        {
            if (GetPriority(s[i]) > GetPriority(S2.Top()))
            {
                S2.Push(s[i]);
            }
            else {
                while (!S2.isEmpty())
                {
                    char operation = S2.Top();
                    if (GetPriority(operation) >= GetPriority(s[i]))
                    {
                        S1.Push(std::string(1, S2.Pop()));
                    }
                    else {
                        break;
                    }
                }
                S2.Push(s[i]);
            }
        }
    }
    else {
        throw "Uncorrect symbol";
    }

}
while (!S2.isEmpty())
{
    S1.Push(std::string(1, S2.Pop()));
}

// we got a postfix entry on stack 1, now convert it to a string

std::vector<std::string> result_inverse;
std::vector<std::string> result;

while (!S1.isEmpty())
{
    result_inverse.push_back(S1.Pop());
}
for (int i = result_inverse.size() - 1; i >= 0; i--)
{
    result.push_back(result_inverse[i]);
}

Postfix postForm(result);
```

```
    return postForm;
}

int Postfix::getCountNotSpecified() {
    int count = 0;
    for (auto& operand : operands)
    {
        if (operand.second == INFINITY) {
            count += 1;
        }
    }

    return count;
}

void Postfix::setValuesFromVector(const std::vector<double>& values) {
    int countOfNotSpecified = getCountNotSpecified();
    if (countOfNotSpecified != values.size())
    {
        std::cout << "Number of provided values not equal number of not
specified values\n";
        return;
    }

    int posValues = 0;
    for (auto& operand : operands)
    {
        if (operand.second == INFINITY) {
            operand.second = values[posValues++];
        }
    }
}
```

Приложение B. Sample_tstack.

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

int main()
{
    try {
        TStack<int> S;
        S.Push(10);
        S.Push(5);
        int b = S.Pop();
        std::cout << "Put 10 and then 5 in stack" << std::endl;
        std::cout << "Get value = " << b << std::endl;
        S.Push(8);
        b = S.Pop();
        std::cout << "Put 8" << std::endl;
        std::cout << "Get value = " << b << std::endl;
    }
    catch (std::invalid_argument const& ex)
    {
    };


    return 0;
}
```

Приложение Г. Sample_talgorithm.

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

int main()
{
  std::string s1 = "A+B+X1+XY+2.5+3";
  std::string s2 = "A+(B-A)/(D-A)*K-(A+B)/B";
  Postfix postfix1 = Polsk::ConvertToPol(s1);
  std::cout << "Formula 1 : " << s1 << std::endl;
  std::cout << "Postfix 1 : " << postfix1.getStringView()<<std::endl;


  postfix1.setValues();
  double result1 = postfix1.calculate();
  std::cout << "Result for formula 1 = " << result1 << std::endl;

  Postfix postfix2 = Polsk::ConvertToPol(s2);
  std::cout << "Formula 2 : " << s2 << std::endl;
  std::cout << "Postfix 2 : "<< postfix2.getStringView() << std::endl;;

  postfix2.setValues();
  double result2 = postfix2.calculate();
  std::cout << "Result for formula 2 = " << result2 << std::endl;

  return 0;

}
```