

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(а): студент(ка) группы
3822Б1ФИ1

_____ / Лысов И.М./
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 стека.....	5
2.3 Результат работы арифметических выражений	6
3 Руководство программиста	8
3.1 Описание алгоритмов	8
3.1.1 Стек.....	8
3.1.2 Арифметическое выражение	9
3.2 Описание программной реализации	11
3.2.1 Описание класса TStack.....	11
Описание класса MathArithmetics.....	13
Заключение	16
Литература	17
Приложения	18
Приложение А. Реализация класса TStack	18
Приложение Б. Реализация класса MathArithmetics	19

Введение

Стек — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO. Error! Reference source not found.

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.

Обратная польская 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.

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 классы TStack и MathArithmetics для преобразования инфиксного арифметического выражения в постфиксную форму.

Задачи:

1. Исследовать необходимую литературу для реализации задачи.
2. Реализовать класс TStack.
3. Реализовать класс MathArithmetics
4. Провести тестирование разработанных классов для проверки их корректной работы.
5. Сделать выводы о проделанной работе.

2 Руководство пользователя

2.1 Приложение для демонстрации стека

1. Запустите приложение с названием sample_stack.exe. В результате появится окно, показанное ниже, где вам нужно будет ввести размерность стека (рис. 1).



Рис. 1. Ввод максимального значения стека

2. Далее вам нужно будет выбрать одно из следующих действий: (рис. 2)
 - 1) Поместить элемент на вершину стека;
 - 2) Вывести элемент на вершине стека;
 - 3) Удалить верхний элемент из стека
 - 4) Выход

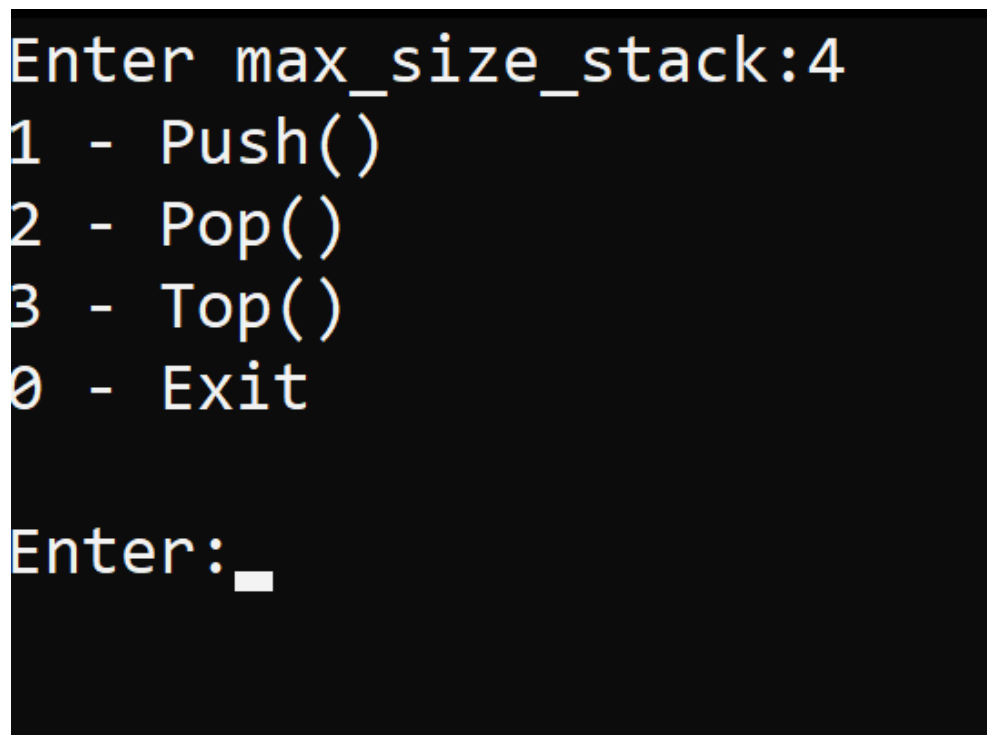


Рис. 2. Основное окно программы

3. После применения данных действий результатом программы будет следующее окно (рис. 3)

```
Enter max_size_stack:4
1 - Push()
2 - Pop()
3 - Top()
0 - Exit

Enter:1
Enter element please:3
1 - Push()
2 - Pop()
3 - Top()
0 - Exit

Enter:2
Pop:3
1 - Push()
2 - Pop()
3 - Top()
0 - Exit
```

Рис. 3. Результат работы приложения

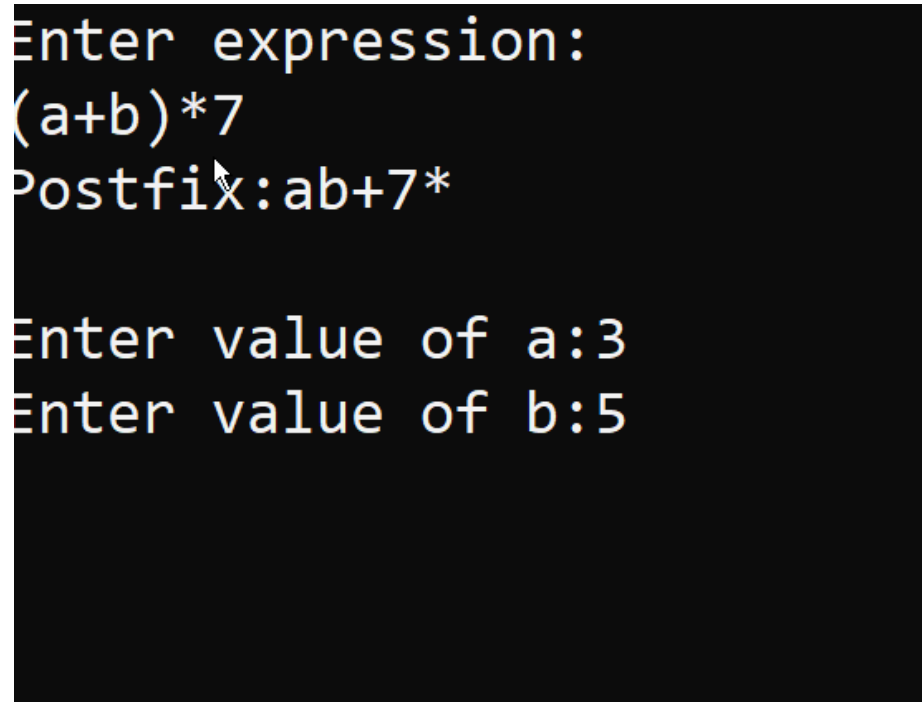
2.3 Результат работы арифметических выражений

1. После запуска программы она запрашивает арифметическое выражение (рис. 4)

```
Enter expression:
```

Рис. 4. Начальное окно работы приложения

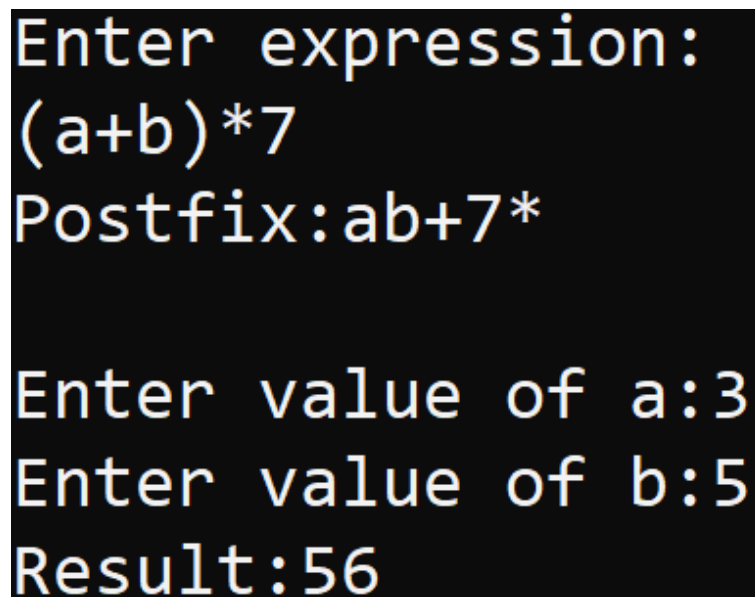
2. После ввода выражения программа просит ввести значения переменных и выводит постфиксную форму (рис. 5).



```
Enter expression:  
(a+b)*7  
Postfix:ab+7*  
  
Enter value of a:3  
Enter value of b:5
```

Рис. 5. Ввод выражения

3. После нажатия клавиши Enter программа выведет результат вычислений. (рис. 6)



```
Enter expression:  
(a+b)*7  
Postfix:ab+7*  
  
Enter value of a:3  
Enter value of b:5  
Result:56
```

Рис. 6. 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», то есть мы можем взаимодействовать (добавлять, удалять) только с вершиной стека. Операции, которые можно выполнять со стеком, включают добавление элемента на вершину стека и удаление элемента с вершины стека.

- **Операция добавления в стек**

Добавление элемента включает увеличение значения флага `top`, который указывает на вершину стека, и затем помещение нового элемента на позицию, указанную этим обновленным значением `top`. Если структура хранения еще не заполнена, элемент может быть добавлен на позицию `top+1`.

Пример:

До добавления элемента в стек:

1	2	3		
---	---	---	--	--

После добавления элемента в стек:

1	2	3	4	
---	---	---	---	--

- **Операция удаления с вершины**

Удаление элемента включает уменьшение значения флага `top`, который указывает на вершину стека, и затем удаление элемента на вершине стека. Если структура хранения еще не пуста, элемент на верхней позиции удаляется.

Пример:

До добавления элемента в стек:

1	2	3		
---	---	---	--	--

После добавления элемента в стек:

1	2			
---	---	--	--	--

- **Операция взятия элемента с вершины**

Взятие элемента с вершины стека осуществляется с помощью флага `top`, который указывает на вершину стека. Если структура хранения еще не пуста, элемент на верхней позиции может быть взят.

Пример:

До добавления элемента в стек:

1	2	3		
---	---	---	--	--

Результат выполнения операции: 3.

- **Операция проверки на полноту (пустоту)**

Операция проверки на полноту (пустоту) проверяет, полон (пуст) ли стек в зависимости от значения флага.

Пример:

полный стек:

1	2	3	4	5
---	---	---	---	---

пустой стек:

--	--	--	--	--

3.1.2 Арифметическое выражение

Арифметическое выражение – выражение, которое составлено из операндов, соединенных арифметическими операциями (+, -, *, /). Программа основывается на использовании стека. Алгоритм ожидает на входе строку, представляющую математическое выражение, а также хэш-таблицу, где элементы соответствуют операндам в данном выражении.

Данный класс поддерживает следующие операции:

- **Получение постфиксной записи.**

Входные данные: строка, содержащая арифметическое выражение в инфиксной форме.

Выходные данные: строка, содержащая арифметическое выражение в постфиксной форме.

Пользователь вводит выражение в инфиксной форме, после чего алгоритм убирает лишние пробелы, проверяет на корректность, разделяет строку на лексемы и константы.

Алгоритм:

1. Создаем пустой стек операторов.
2. Создаем пустой массив для хранения постфиксной записи.
3. Проходим по каждому символу в инфиксной записи слева направо:
 - Если символ является операндом, добавляем его в массив постфиксной записи.
 - Если символ является открывающей скобкой, помещаем его в стек операторов.
 - - Если символ является закрывающей скобкой, извлекаем операторы из стека и добавляем их в массив постфиксной записи до тех пор, пока не встретится открывающая скобка. Удаляем открывающую скобку из стека.

- - Если символ является оператором, извлекаем операторы из стека и добавляем их в массив постфиксной записи до тех пор, пока не будет найден оператор с меньшим или равным приоритетом. Затем помещаем текущий оператор в стек.

4. Извлекаем оставшиеся операторы из стека и добавляем их в массив постфиксной записи.

5. В результате получим выражение в постфиксной форме

Пример:

Инфиксное выражение: $(a+b*c)*(c/d-e)$

Постфиксная запись: $abc*+cd/e-*$

- Стек постфиксной формы:

												*
												-
										e	e	e
									/	/	/	/
								d	d	d	d	d
						c	c	c	c	c	c	c
				+	+	+	+	+	+	+	+	+
				*	*	*	*	*	*	*	*	*
			c	c	c	c	c	c	c	c	c	c
	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

- Стек операторов:

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

- **Вычисление значения выражения**

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 входного арифметического выражения:
 - Если текущий символ - операнд, поместить его в стек.
 - Если текущий символ - оператор, извлечь два последних операнда из стека, выполнить операцию над этими операндами и поместить результат обратно в стек.
- 2) Когда достигнут конец входной последовательности, результат вычисления будет находиться в стеке.

Пример:

- 1) Инфиксная форма: $(a+b*c)*(c/d-e)$
- 2) Постфиксная форма: $abc*+cd/e-*$

3) Переменная	Значение
a	1
b	2
c	3
d	4
e	5

Стек вычисления:

		3			4	5		
	2	2	6	3	3	0,75	-4,25	
1	1	1	1	7	7	7	7	-29,75

3.2 Описание программной реализации

3.2.1 Описание класса TStack

```
template <typename T>
class TStack
{
private:
    T* elems;
    int maxSize;
    int top;

    void Realloc(int step = _step);
};
```

```

public:
    TStack(int Size = _maxSize);
    TStack(const TStack<T>& stack);
    virtual ~TStack();

    T Pop();
    T Top() const;
    void Push(const T& element);
    bool IsEmpty(void) const;
    bool IsFull(void) const;
};

```

Поля:

maxSize – количество доступной памяти, размер стека.

top – индекс верхнего элемента в стеке.

elems – память для представления стека.

Методы:

- `TStack(int Size = _maxSize);`

Назначение: конструктор по умолчанию и конструктор с параметрами.

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

- `TStack(const TStack<T>& stack);`

Назначение: конструктор копирования.

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

- `virtual ~TStack();`

Назначение: освобождение выделенной памяти.

- `void Realloc(int step = _step);`

Назначение: перераспределение памяти.

Входные параметры: **step** – шаг выделяемой памяти.

- `T Top() const;`

Назначение: метод, возвращающий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: элемент, который находится на вершине стека.

- `T Pop();`

Назначение: метод, удаляющий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

- `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.

- `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.

Описание класса MathArithmetics

```
class MathArithmetics
{
private:
    string infix;
    vector<string> postfix;
    vector<string> lexems;
    static map<string, int> priority;
    map<string, double> operands;

    void Parse();
    bool IsConst(const string& st) const;
    bool IsOperator(char c) const;
    bool IsParenthesis(char c) const;
    bool IsDigitOrLetter(char c) const;
    double Calculate(const map<string, double>& values);
    void RemoveSpaces(string& str) const;
    bool isCorrectInfixExpression();
public:
    MathArithmetics(const string& _infix);
    string ToPostfix();
    string GetInfix()
    {
        return infix;
    }
    void SetValues();
    void SetValues(const vector<double>& values);
    double Calculate();
};
```

Поля:

Infix– выражение в инфиксной записи.

Postfix– выражение в постфиксной записи.

Lexems– набор лексем инфиксной записи

Priority– приоритет арифметических операндов

Operands– операнды и их значения

Методы:

- `TAithmeticExpression (const string& _infix);`

Назначение: конструктор с параметрами.

Входные параметры: **Infix**– инфиксное выражение.

Выходные параметры: отсутствуют.

- `string GetInfix();`

Назначение: получение инфиксной формы.

Входные параметры отсутствуют.

Выходные параметры: инфиксная форма записи.

- `string ToPostfix();`

Назначение: получение постфиксной формы.

Входные параметры отсутствуют.

Выходные параметры: постфиксная форма записи.

- `void SetValues() ;`

Назначение: ввод с клавиатуры значений операндов

Входные параметры отсутствуют.

Выходные параметры: **values**- значения операндов.

- `void SetValues(const vector<double>& values) ;`

Назначение: установка значений операндов из вектора.

Входные параметры: вектор значений.

Выходные параметры: **values**- значения операндов.

- `double Calculate() ;`

Назначение: метод вычисления выражения в постфиксной форме.

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

- `double Calculate (const map<string, double>& values) ;`

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.

- `void Parse () ;`

Назначение: подготовка к конвертированию инфиксной формы в постфиксную.

Разделение инфиксной формы на лексемы.

Входные параметры отсутствуют.

Выходные параметры: отсутствуют.

- `bool IsConst(const char c) const;`

Назначение: проверяет строку константа ли это.

Входные параметры: **c** – оператор или операнд.

Выходные параметры: 1, если строка – это константа, 0 иначе.

- `bool IsOperator(const char c) const;`

Назначение: проверяет строку арифметический оператор ли это.

Входные параметры: **c** – оператор или операнд.

Выходные параметры: 1, если строка – это арифметический оператор, 0 иначе.

- **bool IsParenthesis(char c) const;**

Назначение: проверяет строку скобка ли это.

Входные параметры: **c** – открывающаяся или закрывающаяся скобка.

Выходные параметры: 1, если строка – это скобка, 0 иначе.

- **bool IsDigitOrLetter(char c) const;**

Назначение: проверяет строку буква или цифра ли это.

Входные параметры: **c** – буква или цифра.

Выходные параметры: 1, если строка – это буква или цифра, 0 иначе.

- **void RemoveSpaces(string& str) const;**

Назначение: удаляет пробелы из строки.

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

Выходные параметры: строка без пробелов.

- **bool isCorrectInfixExpression();**

Назначение: проверяет инфикс на равенство открывающихся и закрывающихся скобок.

Входные параметры отсутствуют.

Выходные параметры: 1, количество открывающихся и закрывающихся скобок равно, 0 иначе.

Заключение

В результате этой лабораторной работы был разработан шаблонный класс TStack, который поддерживает операции добавления элемента, удаления элемента с вершины стека, просмотра элемента на вершине стека без его удаления и другие. На основе этого класса был разработан класс для работы с польской формой записи арифметического выражения, который также поддерживает различные операции.

В целом, проведенный анализ показал, что использование постфиксной формы может быть очень полезным в решении определенных задач.

Литература

- [1] Стек [<https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA>]
- [2] Лекция «Постфиксная форма» Сысоев А.В[<https://cloud.unn.ru/s/6g44ey6HFB4ncDy>].

Приложения

Приложение А. Реализация класса TStack

```
template <typename T>
TStack<T>::TStack(int Size) //конструктор класса
{
    if (Size <= 0)
        throw "The maxSize must be greater than 0.";
    maxSize = Size;
    top = -1;
    elems = new T[maxSize];
}

template <typename T>
TStack<T>::TStack(const TStack<T>& stack) //конструктор копирования класса
{
    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()
{
    if (elems != NULL)
    {
        delete[] elems;
        top = -1;
        maxSize = 0;
    }
}

template <class T> //функция меняющая размер стека
void TStack<T>::Realloc(int step)
{
    if (step <= 0) { throw "The step should be greater than 0."; }
    T* tmp = new T[step + maxSize];
    for (int i = 0; i < maxSize; i++)
    {
        tmp[i] = elems[i];
    }
    delete[] elems;
    elems = tmp;
    maxSize = maxSize + step;
}

template <typename T> //проверка на заполненность
bool TStack<T>::IsFull(void) const
{
    return (top == maxSize - 1);
}

template <typename T> //проверка на пустоту
bool TStack<T>::IsEmpty(void) const
{
    return (top == -1); // по дефолту top = -1
}

template <typename T>
T TStack<T>::Top() const
{
    if (IsEmpty())
        throw "Stack does not contain any data";
}
```

```

        return elems[top];
    }

    template <typename T>
    void TStack<T>::Push(const T& element)
    {
        if (IsFull())
            Realloc(_step);
        elems[++top] = element;
    }

    template <typename T>
    T TStack<T>::Pop()
    {
        if (IsEmpty())
            throw "Stack does not contain any data";
        return elems[top--];
    }

```

Приложение Б. Реализация класса MathArithmetics

```

MathArithmetics::MathArithmetics(const string& _infix) //конструктор класса
{
    if (_infix.empty())
    {
        throw("Expression contains no data");
    }
    infix = _infix;
    RemoveSpaces(infix);
    if (!isCorrectInfixExpression())
        throw("Wrong number of brackets");
}

map<string, int> MathArithmetics::priority = {
    {"*", 3},
    {"/", 3},
    {"+", 2},
    {"-", 2},
    {"(", 1},
    {")", 1}
};

void MathArithmetics::RemoveSpaces(string& str) const
{
    str.erase(remove(str.begin(), str.end(), ' '), str.end());
}

bool MathArithmetics::IsConst(const string& s) const
{
    for (char symbol : s)
    {
        if (!isdigit(symbol) && symbol != '.')
            return false;
    }
    return true;
}

bool MathArithmetics::IsOperator(char symbol) const
{
    return (symbol == '+' || symbol == '-' || symbol == '*' || symbol == '/');
}

bool MathArithmetics::IsParenthesis(char symbol) const
{
    return (symbol == '(' || symbol == ')');
}

bool MathArithmetics::IsDigitOrLetter(char symbol) const
{

```

```

        return (isdigit(symbol) || symbol == '.' || isalpha(symbol));
    }

void MathArithmetics::SetValues()
{
    double value;
    for (auto& op : operands)
    {
        if (!IsConst(op.first))
        {
            cout << "Enter value of " << op.first << ":";
            cin >> value;
            operands[op.first] = value;
        }
    }
}

void MathArithmetics::SetValues(const vector<double>& values)
{
    int i = 0;
    for (auto& op : operands)
    {
        if (!IsConst(op.first))
        {
            operands[op.first] = values[i++];
        }
    }
}

void MathArithmetics::Parse()
{
    string currentElement;
    for (char symbol : infix) {
        if (IsOperator(symbol) || IsParenthesis(symbol) || symbol == ' ')
        {
            if (!currentElement.empty())
            {
                lexems.push_back(currentElement);
                currentElement = "";
            }
            lexems.push_back(string(1, symbol));
        }
        else if (IsDigitOrLetter(symbol))
        {
            currentElement += symbol;
        }
    }
    if (!currentElement.empty())
    {
        lexems.push_back(currentElement);
    }
}

string MathArithmetics::ToPostfix()
{
    Parse();
    TStack<string> stack;
    string postfixExpression;
    for (string item : lexems)
    {
        if (item == "(")
            stack.Push(item);
        else if (item == ")")
        {
            while (stack.Top() != "(")
            {
                postfixExpression += stack.Top();
                postfix.push_back(stack.Top());
                stack.Pop();
            }
        }
    }
}

```

```

        stack.Pop();
    }
    else if (IsOperator(item[0]))
    {
        while (!stack.IsEmpty() && priority[item] <= priority[stack.Top()])
        {
            postfixExpression += stack.Top();
            postfix.push_back(stack.Top());
            stack.Pop();
        }
        stack.Push(item);
    }
    else
    {
        double value = IsConst(item) ? stod(item) : 0.0;
        operands.insert({ item, value });
        postfix.push_back(item);
        postfixExpression += item;
    }
}
while (!stack.IsEmpty())
{
    postfixExpression += stack.Top();
    postfix.push_back(stack.Top());
    stack.Pop();
}
return postfixExpression;
}

double MathArithmetics::Calculate(const map<string, double>& values)
{
    for (auto& val : values)
    {
        try
        {
            operands.at(val.first) = val.second;
        }
        catch (out_of_range& e) {}
    }
    TStack<double> stack;
    double leftOperand, rightOperand;
    for (string lexem : postfix)
    {
        if (lexem == "+")
        {
            rightOperand = stack.Top();
            stack.Pop();
            leftOperand = stack.Top();
            stack.Pop();
            stack.Push(leftOperand + rightOperand);
        }
        else if (lexem == "-")
        {
            rightOperand = stack.Top();
            stack.Pop();
            leftOperand = stack.Top();
            stack.Pop();
            stack.Push(leftOperand - rightOperand);
        }
        else if (lexem == "*")
        {
            rightOperand = stack.Top();
            stack.Pop();
            leftOperand = stack.Top();
            stack.Pop();
            stack.Push(leftOperand * rightOperand);
        }
        else if (lexem == "/")
        {
            rightOperand = stack.Top();

```

```

        stack.Pop();
        leftOperand = stack.Top();
        stack.Pop();
        if (rightOperand == 0)
            throw "You cant divide by zero";
        stack.Push(leftOperand / rightOperand);
    }
    else
    {
        stack.Push(operands[lexem]);
    }
}
return stack.Top();
}

double MathArithmetics::Calculate()
{
    return Calculate(operands);
}

bool MathArithmetics::isCorrectInfixExpression()
{
    int count = 0;
    for (char symbol : infix)
    {
        if (symbol == '(')
            count++;
        else if (symbol == ')')
            count--;
        if (count < 0)
            return false;
    }
    return (count == 0);
}

```