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:
«Стек»

Выполнил(а): студент(ка) группы 3822Б1ФИ2
_____ / Чижов М.А./
Подпись

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

Нижний Новгород
2023

Содержание

Введение

Стек — это одна из основных структур данных в программировании. Он представляет собой упорядоченную коллекцию элементов, где добавление новых элементов и удаление существующих осуществляется в определенном порядке — «последним пришел, первым ушел». Такой принцип работы стека называется принципом LIFO (Last In, First Out) или «последним пришел, первым вышел».

Особенностью стека является то, что доступ к элементам коллекции осуществляется только через вершину стека. Это значит, что при добавлении нового элемента он становится на вершину стека, а при удалении — удаляется элемент, находящийся на вершине. Такая организация стека позволяет реализовывать различные алгоритмы работающие с ограниченным набором данных, где необходимы операции добавления и удаления элементов в определенном порядке.

Применение стека в программировании широко распространено и находит свое применение в различных задачах. Одним из основных применений стека является реализация алгоритмов и структур данных, таких как обходы графов, рекурсивные функции, обратная польская запись и многое другое.

Стек также используется для сохранения состояния программы во время выполнения. Например, когда функция вызывается, текущее состояние программы (адрес возврата, значения локальных переменных и т.д.) сохраняется в стеке. При завершении функции состояние восстанавливается из стека, чтобы выполнение программы продолжилось с того места, где оно было прервано.

Кроме того, стек может использоваться для обработки и хранения временных данных в программе. Например, стек может использоваться для реализации механизма отката действий в текстовом редакторе, где каждое действие добавляется в стек и может быть отменено путем извлечения последнего элемента из стека.

Также стек может быть полезен при работе с рекурсией. Каждый раз, когда функция вызывает саму себя, новые переменные и значения сохраняются в стеке, позволяя программе вести учет всех рекурсивных вызовов и правильно возвращаться обратно при завершении рекурсии.

Использование стека позволяет программистам решать различные задачи эффективно и удобно. Однако при неправильном использовании стека может возникнуть переполнение или недостаток памяти, поэтому важно использовать его правильно и осуществлять контроль за его состоянием.

1 Постановка задачи

Цель – реализовать структуру данных стек.

Задачи:

1. Реализовать класс для работы со стеком.
2. Написать следующие операции для работы со стеком: изъятие с вершины, вставка на вершину, проверка последнего элемента, проверка на пустоту, проверка на полноту.
3. Добавить вспомогательные операции получения размера и элемента по индексу.
4. Написать следующие алгоритмы для работы со стеками: перевод в постфиксную форму, вычисление выражения, записанного в постфиксной форме, перемещение элементов из одного стека в другой.

2 Руководство пользователя

2.1 Приложение для демонстрации работы стека

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

Рис. 1. Основное окно программы

2. Введите арифметическое выражение. В результате выведется выражение в постфиксной форме (рис. 2).

Рис. 2. Вывод постфиксной формы

3. Введите значения переменных (рис. 3).

Рис. 3. Ввод значений переменных

4. После ввода значений появится результат арифметического выражения (рис. 4).

Рис. 4. Вывод результата

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Стек

Структура данных TStack представляет собой шаблонный класс. Для создания экземпляра класса TStack необходимо указать его размер. По умолчанию размер устанавливается как 10. Структура данных TStack также поддерживает ряд операций, включая добавление на вершину, изъятие с вершины, проверка последнего элемента, проверка на пустоту и полноту. Также присутствуют операции проверки элемента по индексу и размера стека.

Операция добавления в стек

Функция добавляет элемент на вершину стека. Для этого используется флаг top, который показывает, элемент с каким индексом считается вершиной стека. Если стек пуст top=-1. В примере top=2.

5	6	1		

Добавим элемент 7

Результат:

top=3, элемент в стеке

5	6	1	7	

Операция изъятия с вершины

Функция удаляет элемент из стека. Для этого также воспользуемся флагом top=4. Если стек пуст, то будет выбрасываться исключение.

5	6	1	7	2

Результат:

Флаг top теперь стал равен 3.

5	6	1	7	2

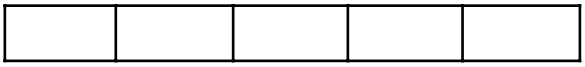
Операция проверки на пустоту

Функция проверяет есть ли в стеке элементы. Если в стеке нет элементов, то функция вернет true, в противном случае false.

5	6	1	7	

Результат:

false – стек не пуст

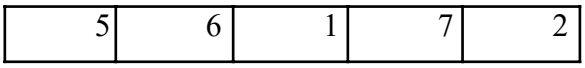


Результат:

true – стек пуст

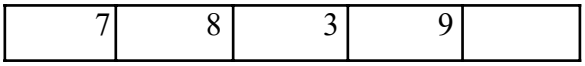
Операция проверки на полноту

Функция проверяет , достиг ли стек своей максимальной вместимости элементов. Она возвращает true, если количество элементов в стеке соответствует его максимальному размеру, тем самым указывая на то, что добавление дополнительных элементов невозможно. В противном случае вернет false.



Результат:

true – стек полон

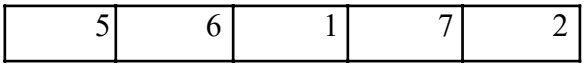


Результат:

false – стек не полон

Операция проверки последнего элемента

Функция позволяет получить значение последнего добавленного элемента, не удаляя его из стека.



Результат: 2

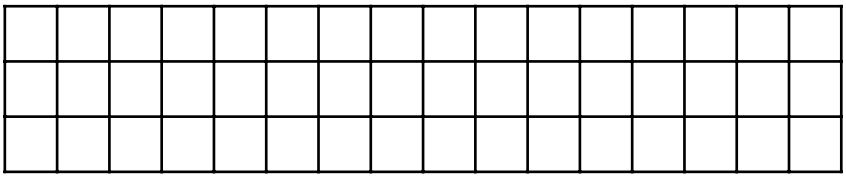
3.1.2 Арифметические выражения

Операция перевода в постфиксную форму

Функция перевода в обратную польскую запись в стеке выполняет преобразование инфиксного выражения в соответствующее постфиксное выражение.

A+(B-C)*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
A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A

Стек 2.

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

Операция вычисления по постфиксной форме

Эта функция проходит по каждому токену в постфиксном выражении. Если токен является операндом, он помещается в стек. Если токен является оператором, из стека извлекаются два операнда, над которыми выполняется соответствующая операция. Результат операции помещается обратно в стек. После обработки всех токенов в постфиксном выражении, в стеке остается только один элемент - результат вычисления выражения, который извлекается из стека и возвращается в качестве результата функции.

Вычислим значение выражения, представленного выше.

A=0, B=1, C=2, D=-1, F=2, G=0.5, H=0.5

												+					
			-		*						0.5	0.5		/			
		2	2	-1	-1		+			0.5	0.5	0.5	1	1		-	

	1	1	1	-1	-1	1	1		2	2	2	2	2	2	2	2	
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	-1

Результат: -1

3.2 Описание программной реализации

3.2.1 Описание класса TStack

```
template <class T>
class TStack
{
private:
	int maxSize;
	int top;
	T* elems;
public:
	TStack(int maxSize = 10);
	TStack(const TStack<T>& s);
	~TStack();
	TStack<T>& operator=(const TStack<T>& s);
	bool IsEmpty(void) const;
	bool IsFull(void) const;
	void ResizeStack();
	void ReverseStack();
	T Top() const;
	void Push(const T& elem);
	void Pop();

	friend ostream& operator <<(ostream& out, const TStack& s)
	{
		for (int i = 0; i <= s.top; i++)
			out << s.elems[i] << ' ';
		return out;
	};
};
```

Назначение: представление вектора.

Поля:

elems– указатель на массив типа **T**.

maxSize – размер вектора.

start_ind – индекс верхнего элемента в стеке.

Методы:

TStack(int maxSize)

Назначение: Конструктор класса, инициализирующий стек заданного размера.

Входные параметры:

maxSize - максимальный размер стека, заданный при инициализации.

Выходные параметры: отсутствуют.

TStack(const TStack& s)

Назначение: Конструктор копирования, создающий копию стека.

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

Выходные параметры: отсутствуют.

~TStack();

Назначение: освобождает выделенную память для стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

bool IsEmpty() const;

Назначение: Метод, проверяющий, пуст ли стек.

Входные параметры: отсутствуют.

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

bool IsFull() const;

Назначение: Метод, проверяющий, заполнен ли стек.

Входные параметры: отсутствуют.

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

void ResizeStack();

Назначение: Метод, увеличивающий размер стека при его переполнении.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

const TStack<T>& operator=(const TStack<T>& s);

Назначение: присваивание одного стека другому.

Входные параметры: s - объект класса TStack, который нужно присвоить.

Выходные параметры: ссылка на текущий объект класса **TStack**.

void ReverseStack();

Назначение: изменение порядка элементов в стеке на обратный.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

T Top() const;

Назначение: Метод, возвращающий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: верхний элемент стека.

```
void Push(const T& elem);
```

Назначение: Метод, помещающий элемент на вершину стека.

Входные параметры: elem - элемент, который требуется добавить на вершину стека.

Выходные параметры: отсутствуют.

```
void Pop();
```

Назначение: Метод, удаляющий верхний элемент стека.

Входные параметры: отсутствуют.

Выходные параметры: отсутствуют.

```
friend ostream& operator <<(ostream& out, const TStack& s);
```

Назначение: оператор вывода для класса **TStack**.

Входные параметры:

ostr – ссылка на объект типа **ostream**, который представляет выходной поток.

s – ссылка на объект типа **TStack**, который будет выводиться.

Выходные параметры: ссылка на объект типа **ostream**.

3.2.2 Описание класса ArithmeticExpression

```
class ArithmeticExpression
{
public:
    static TStack<string> Postfix_Form(const string& s);
      static double Calculate(TStack<string>& st, const map<string, double>&
values);
    static int Is_Symbol(string sm);
    static int Get_Priority(const string symbol);
       static void Add_to_Stack1(TStack<string>& st1, TStack<string>& st2,
string s);
    static bool Is_Number(const string& str);
    static bool isValidExpression(const string& expression);
    static bool isOperand(char c);
    static string FilteredExpression(const string& s);
              static   map<string,   double>   GetVariables(TStack<string>&
postfixExpression);
};
```

Методы:

```
TStack<string> Postfix_Form(const string& s);
```

Назначение: преобразование инфиксного выражения в постфиксное.

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

Выходные параметры: стек, представляющий постфиксное выражение.

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

Назначение: вычисление значения постфиксного выражения.

Входные параметры:

st – постфиксное выражение.

values – словарь переменных и их значений.

Выходные параметры: значение выражения.

```
int Is_Symbol(string sm);
```

Назначение: определяет, является ли переданный символ оператором.

Входные параметры: **sm** - символ для проверки.

Выходные параметры: 1 – символ является оператором, 2 – скобка, иначе 0.

```
int Get_Priority(const string symbol);
```

Назначение: получение приоритета оператора.

Входные параметры: **symbol** - оператор.

Выходные параметры: целое число – приоритет оператора.

```
void Add_to_Stack1(TStack<string>& st1, TStack<string>& st2, string s);
```

Назначение: добавление элемента в стек **st1** и удаление элемента из стека **st2**

Входные параметры: стек **st1**, стек **st2**, **s** – элемент для добавления в стек **st1**

Выходные параметры: отсутствуют.

```
bool Is_Number(const string& str);
```

Назначение: проверка, является ли переданная строка числом.

Входные параметры: строка **str** для проверки.

Выходные параметры: 0 – не является числом, 1 – число.

```
bool isValidExpression(const string& expression);
```

Назначение: проверка валидности математического выражения.

Входные параметры: **expression** – математическое выражение для проверки.

Выходные параметры: 1 – выражение введено верно, 0 – в противном случае.

```
bool isOperand(char c);
```

Назначение: проверка, является ли переданный символ операндом.

Входные параметры: символ **c**.

Выходные параметры: 1 – операнд, 0 – в противном случае.

```
string FilteredExpression(const string& s);
```

Назначение: удаление пробелов из переданной строки.

Входные параметры: строка **s** – выражение для фильтрации.

Выходные параметры: отфильтрованная строка.

```
map<string, double> GetVariables(TStack<string>& postfixExpression);
```

Назначение: получение значения переменных из постфиксного выражения.

Входные параметры: стек **postfixExpression** - постфиксное выражение.

Выходные параметры: словарь с уникальными переменными и их значениями.

Заключение

В ходе выполнения работы была представлена реализация стека для обработки арифметических выражений. Разработанный стек обладает основными методами, необходимыми для работы с данными структуры: Push для добавления элемента, Pop для удаления элемента, Top для доступа к вершине стека, а также методы проверки на пустоту и заполненность стека.

Кроме того, была предусмотрена возможность изменения размера стека в случае его заполненности, что обеспечивает гибкость и эффективное использование структуры данных в различных сценариях.

Реализация стека позволяет эффективно работать с арифметическими выражениями, что особенно актуально в контексте перевода таких выражений в постфиксную форму и последующего вычисления. Методы работы со стеком предоставляют удобный и надежный инструмент для обработки данных и манипуляций с ними.

Таким образом, разработанный стек представляет собой важный компонент для решения задачи перевода арифметических выражений в постфиксную форму и может быть успешно интегрирован в различные программные системы, где требуется работа с арифметическими выражениями.

Литература

1. Что такое стек и как его использовать в программировании [https://podarkiyarki.ru/chto-takoe-stek-v-programmirovanii-i-kak-ego-ispolzovat]

Приложения

Приложение А. Реализация класса TStack

```
template <class T>
TStack<T>::TStack<T>(int maxSize)
{
	if (maxSize <= 0)
	{
		string msg = "Error";
		throw msg;
	}
	this->maxSize = maxSize;
	top = -1;
	elems = new T[maxSize];
}

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

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

template <class T>
const TStack<T>& TStack<T>::operator=(const TStack<T>& s)
{
	if (this == &s) {
		return *this;
	}

	delete[] elems;
	maxSize = s.maxSize;
	top = s.top;
	elems = new T[maxSize];
	for (int i = 0; i <= top; i++) {
		elems[i] = s.elems[i];
	}

	return *this;
}

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

template <class T>
bool TStack<T>::IsFull(void) const
{
	if (maxSize - 1 == top)
		return true;
```

```
	return false;
}

template <class T>
void TStack<T>::ResizeStack()
{
	int newMaxSize = maxSize * 5;
	T* newElems = new T[newMaxSize];

	for (int i = 0; i <= top; i++)
		newElems[i] = elems[i];

	delete[] elems;
	elems = newElems;
	maxSize = newMaxSize;
}

template <typename T>
void TStack<T>::ReverseStack()
{
	TStack<T> tempStack;
	while (!IsEmpty())
	{
		T element = Top();
		Pop();
		tempStack.Push(element);
	}

	*this = tempStack;
}

template <class T>
T TStack<T>::Top() const
{
	if (top == -1)
	{
		string msg = "Error: stack is empty";
		throw msg;
	}
	return elems[top];
}

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

template <class T>
void TStack<T>::Pop()
{
	if (IsEmpty())
	{
		string msg = "Error: stack is empty";
		throw msg;
	}
	top -= 1;
}
```

Приложение Б. Реализация класса ArithmeticExpression

```
map<string, double> ArithmeticSymbol::symbolDict = {
	{"*", 3},
	{"/", 3},
	{"+", 2},
	{"-", 2},
};

int ArithmeticExpression::Is_Symbol(string sm)
{
	if (ArithmeticSymbol::symbolDict.find(sm) != 
ArithmeticSymbol::symbolDict.end())
		return 1;
	else if (sm == "(" || sm == ")")
		return 2;
	return 0;
}

int ArithmeticExpression::Get_Priority(const string symbol)
{
	int priority = ArithmeticSymbol::symbolDict[symbol];
	if (symbol == "(" || symbol == ")")
		priority = 1;
	return priority;
}

void ArithmeticExpression::Add_to_Stack1(TStack<string>& st1, TStack<string>& 
st2, string s)
{
	st1.Push(s);
	st2.Pop();
}

bool ArithmeticExpression::Is_Number(const string& str)
{
	for (int i = 0; i< str.length(); i++)
	{
		char c = str[i];
		if (!isdigit(c)) {
			return false;		}
	}
	return true;
}

bool ArithmeticExpression::isOperand(char c)
{
	return ((c >= '0' && c <= '9') || (c >= 'a' && c <= 'z') || (c >= 'A' && 
c <= 'Z'));
}


string ArithmeticExpression::FilteredExpression(const string& s)
{
	string filteredExpression = "";
	int l = s.length();
```

```
	for (int i = 0; i<l; i++)
	{
		char c = s[i];
		if (c != ' ')
		{
			filteredExpression += c;
		}
	}
	return filteredExpression;
}


bool ArithmeticExpression::isValidExpression(const string& expression)
{
	int k1 = 0, k2 = 0;
	int l = expression.length();
	for (int i = 0; i < l; i++)
	{
		char c = expression[i];
		string s(1, c);
		if (i == 0)
			if (Is_Symbol(s)==1)
				return false;
		if (s == "(" || s == ")")
		{
			if (s == "(")
				k1++;
			else
			{
				k2++;
				char c1 = expression[i + 1];
				string s1(1, c);
				if (isOperand(c1) || c1=='(')
					return false;
			}
		}
		else if ((Is_Symbol(s) != 0) || isOperand(c))
		{
			char c1 = expression[i + 1];
			string s1(1, c1);
			if (Is_Symbol(s) == 1)
			{
				if (Is_Symbol(s1) == 1)
					return false;
				if (i == l - 1)
					return false;
			}
			if ((isOperand(c)) && (s1 == "("))
				return false;
			continue;
		}
		else
			return false;
	}
	if (k1 != k2)
		return false;
	return true;
}

map<string,	double>	ArithmeticExpression::GetVariables(TStack<string>&
postfixExpression) {
	map<string, double> uniqueVariables;
	for (int i = 0; i < postfixExpression.Length(); i++) {
```

```
            string token = postfixExpression.GetElement(i);
            if ((Is_Symbol(token) != 0) || Is_Number(token))
                  continue;
            else
            {
                  if (uniqueVariables.find(token) == uniqueVariables.end()) {
                        double value;
                        cout << "Enter the value for variable " << token << ":
";
                        cin >> value;
                        uniqueVariables[token] = value;
                  }
            }
      }
      return uniqueVariables;
}

TStack<string> ArithmeticExpression::Postfix_Form(const string& s)
{
      string str = FilteredExpression(s);
      if (!isValidExpression(str))
      {
            string msg = "Input error";
            throw msg;
      }
      TStack<string> st1(5);
      TStack<string> st2(5);
      string numStr;
      string varStr;
      for (int i = 0; i < str.length(); i++)
      {
            char s1 = str[i];
            string s(1,s1);
            if (isdigit(s1))
                  if (!varStr.empty())
                        varStr += s;
                  else
                        numStr += s;
            else
            {
                  if (!numStr.empty())
                  {
                        st1.Push(numStr);
                        numStr.clear();
                  }
                  if (!varStr.empty())
                  {
                        st1.Push(varStr);
                        varStr.clear();
                  }
                  if (Is_Symbol(s) != 0)
                  {
                        if (!st2.IsEmpty())
                        {
                              if (s == ")")
                              {
                                    string sm = st2.Top();
                                    while (sm != "(")
                                    {
                                          Add_to_Stack1(st1, st2, sm);
                                          sm = st2.Top();
                                    }
                                    st2.Pop();
```

```
                        }
                        else if (s == "(")
                              st2.Push(s);
                        else
                        {
                              string priveous = st2.Top();
                              int pr1 = Get_Priority(priveous);

                              int pr2 = Get_Priority(s);
                              if (pr2 != 1)
                              {
                                    while (pr1 >= pr2)
                                    {
                                          string sm1 = st2.Top();
                                          Add_to_Stack1(st1, st2, sm1);
                                          if (st2.IsEmpty())
                                                break;
                                          else
                                          {
                                                priveous = st2.Top();
                                                pr1                    =
Get_Priority(priveous);
                                          }
                                    }
                              }
                              st2.Push(s);
                        }
                  }
                  else
                        st2.Push(s);
            }
            else
                  varStr += s;
      }
  }
  if (!numStr.empty())
  {
        st1.Push(numStr);
        numStr.clear();
  }
  if (!varStr.empty())
  {
        st1.Push(varStr);
        varStr.clear();
  }
  while (!st2.IsEmpty())
  {
        string a = st2.Top();
        Add_to_Stack1(st1, st2, a);
  }
  return st1;
}

double ArithmeticExpression::Calculate(TStack<string>& st, const map<string,
double>& values)
{
  TStack<double> stack(20);
  double rightOp, leftOp, resOp;
  for (int i = 0; i < st.Length(); i++)
  {
        string c = st.GetElement(i);
        if (Is_Symbol(c) == 1)
        {
```

```
                if (c == "+")
                {
                        rightOp = stack.Top();
                        stack.Pop();
                        leftOp = stack.Top();
                        stack.Pop();
                        resOp = leftOp + rightOp;
                        stack.Push(resOp);
                }
                if (c == "-")
                {
                        rightOp = stack.Top();
                        stack.Pop();
                        leftOp = stack.Top();
                        stack.Pop();
                        resOp = leftOp - rightOp;
                        stack.Push(resOp);
                }
                if (c == "*")
                {
                        rightOp = stack.Top();
                        stack.Pop();
                        leftOp = stack.Top();
                        stack.Pop();
                        resOp = leftOp * rightOp;
                        stack.Push(resOp);
                }
                if (c == "/")
                {
                        rightOp = stack.Top();
                        stack.Pop();
                        leftOp = stack.Top();
                        stack.Pop();
                        resOp = leftOp / rightOp;
                        stack.Push(resOp);
                }
        }
        else
        {
                if (Is_Number(c))
                        stack.Push(stod(c));
                else
                        stack.Push(values.at(c));
        }

    }
    double r = stack.Top();
    stack.Pop();
    return r;
}
```