Федеральное агентство по образованию Российской Федерации
Государственное образовательное учреждение
высшего профессионального образования
Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Вычисление арифметических выражений (стеки)

Выполнил:
студент института ИТММ гр. 381908-4
Галиндо Хавьер Э.

Проверил:
ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Нижний Новгород
2020 г.

Содержание

Введение

Актуальностью данной работы является то, что, массив или структура списка вызовов функций и параметров, используемых в современном компьютерном программировании и архитектуре ЦП – это «стек» - «последним пришёл — первым вышел».

Данный эффект можно описать примером из жизни – когда книги лежат стопкой и добавить или убрать книгу или элементы можно только из вершины стопки.

Сам процесс добавления данных в стек называется “Push”, в то время как получение данных из стека называется «выталкиванием».

Это происходит в верхней части стека. Указатель стека указывает размер стека, изменяясь по мере того, как элементы помещаются или выталкиваются в стек.

Когда функция вызывается, адрес следующей инструкции помещается в стек.

Когда функция завершается, адрес извлекается из стека, и выполнение продолжается с этого адреса.

Целью и практической значимостью данной работы является - реализация программ, обеспечивающих поддержку стеков, и разработка программного обеспечения, обрабатывающего арифметические выражения.

.

Постановка задачи

Задача работы является — реализация программ, обеспечивающих поддержку стеков, и разработка программного обеспечения, обрабатывающего арифметические выражения.

Первоначальная формулировка будет проверять написание выражения, состоит из проверки правильности размещения скобок, преобразование в постфиксную форму выполняется только для правильных выражений, а расчет выполняется для правильных выражений, содержащих только числовые операнды и допустимые знаки операции.

Руководство пользователя

Пользователю нужно запустить файл Main.exe.

Откроется консольное приложение для стэка.

Выведится 2 стэка и операции с ними.

Руководство программиста

Описание структуры программы

Реализует операции:

- **T* stackPtr;** указатель на стек
- **const int size;** максимальное количество элементов в стеке
- **int num;** номер текущего элемента стека

public:

- **TStack(int = 25);** по умолчанию размер стека равен 25 элементам
- **TStack(const TStack<T>&);** конструктор копирования
- **~TStack();** деструктор
- **inline int IsEmpty(void) const;** контроль пустоты
- **inline int IsFull(void) const;** контроль переполнения
- **inline void put(const T&);** поместить элемент в вершину стека
- **inline T deleteElem();** удалить элемент из вершины стека и вернуть его
- **inline const T& Peek(int) const;** n-й элемент от вершины стека
- **inline int getStackSize() const;** получить размер стека
- **inline T* getPtr() const;** получить указатель на стек
- **inline int getNum() const;** получить номер текущего элемента в стеке
- **inline int min_elem();** Поиск минимального элемента
- **inline int max_elem();** Поиск максимального элемента

Описание алгоритмов

Для работы со стеком предлагается реализовать следующие операции:

- **Метод Put:** добавить элемент;

При добавлении элемента в стек необходимо переместить указатель вершины стека, записать элемент в соответствующую позицию динамического массива и увеличить количество элементов.

- **Метод Get:** удалить элемент;

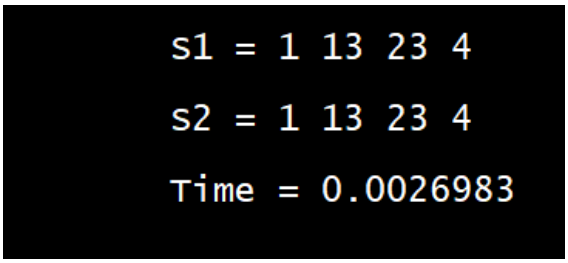
При удалении элемента из стека необходимо возвратить значение из динамического массива по индексу вершины стека, переместить указатель вершины стека и уменьшить количество элементов.

- **Метод IsEmpty:** проверить стек на пустоту; Стек пуст, если в нем нет ни одного элемента, т.е. когда количество элементов равно нулю.

- **Метод IsFull:** проверить стек на полноту. Стек полон при исчерпании всей отведенной под хранение элементов памяти, т.е. когда значение DataCount совпадает со значением size.

Эксперименты

Конструктор копирования



Заключение

Подытожим, что целью данной работы являлось – реализация программ, обеспечивающих поддержку стеков, и разработка программного обеспечения, обрабатывающего арифметические выражения.

Как видно из структуры данной работы результатом стала – реализация класса и его операций помогает нам достичь желаемых результатов при создании этого программного обеспечения, программа была протестирована с помощью фреймворка, предоставленного Google Tests.

А также реализована практическая значимость данной работы – применить приобретенные навыки непосредственно на практике.

Литература

1. Круз Р.Л. Структуры данных и проектирование программ. – М.: Бином. Лаборатория знаний, 2014.
2. Топп У., Форд У. Структуры данных в C++.- М.: Бином, 1999
3. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ.- М.: МЦМО, 1999.
4. Ахо Альфред В., Хопкрофт Джон, Ульман Джеффри Д. Структуры данных и алгоритмы- М.: Издательский дом Вильямс, 2000.
5. Гергель В.П. и др. Методы программирования. Учебное пособие. Н.Новгород: ННГУ, 2016.
6. Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
7. Страуструп Б. Язык программирования C++. – М.: Бином, 2001

Приложения

Stack.h

```
#ifndef STACK_H
#define STACK_H

#include <iostream>
#include <fstream>

using namespace std;

template <typename T>
class TStack
{
private:
	T* stackPtr;
	const int size;
	int num;
public:
	TStack(int = 25);
	TStack(const TStack<T>&);
	~TStack();


	inline int IsEmpty(void) const;
	inline int IsFull(void) const;
	inline void put(const T&);
	inline T deleteElem();
	inline const T& Peek(int) const;
	inline int getStackSize() const;
	inline T* getPtr() const;
	inline int getNum() const;


	inline int min_elem();
	inline int max_elem();


	friend ostream& operator<<(ostream& out, const TStack& st)
	{
		for (int ix = st.num - 1; ix >= 0; ix--)
			cout << st.stackPtr[ix] << endl;
		return out;
	}
};


template <typename T>
TStack<T>::TStack(int maxSize) : size(maxSize)
{
	if (maxSize < 0)
	{
		throw  logic_error("ERROR");
	}
	stackPtr = new T[size];
	num = 0;
}
```

```
template <typename T>
inline int TStack<T>::IsEmpty() const
{
	return stackPtr == NULL;

	for (int i = 0; i < size; i++)
	{
		if (stackPtr[i] == 0)
		{
			continue;
		}
		else
		{
			throw  logic_error("ERROR");
		}

	}
}


template <typename T>
inline int TStack<T>::IsFull() const
{

	for (int i = 0; i < size; i++)
	{
		if (stackPtr[i] != 0)
		{
			continue;
		}
		else
		{
			throw  logic_error("ERROR");
		}
	}
}



template <typename T>
TStack<T>::TStack(const TStack<T>& otherStack) : size(otherStack.getStackSize())
{
	stackPtr = new T[size];
	num = otherStack.getNum();

	for (int ix = 0; ix < num; ix++)
	{
		stackPtr[ix] = otherStack.getPtr()[ix];
	}
}

template <typename T>
TStack<T>::~TStack()
{
	if (this->stackPtr != NULL)
	{
		delete[] stackPtr;
```

```
	}
	num = 0;
}


template <typename T>
inline void TStack<T>::put(const T& value)
{
	if (num > size - 1 || num < 0)
	{
		throw  logic_error("ERROR");
	}

	stackPtr[num++] = value;
}


template <typename T>
inline T TStack<T>::deleteElem()
{
	if (num < NULL)
	{
		throw  logic_error("ERROR");
	}

	stackPtr[--num];

	return stackPtr[num];
}


template <class T>
inline const T& TStack<T>::Peek(int Elem) const
{
	if (Elem > num)
	{
		throw  logic_error("ERROR");
	}
	return stackPtr[num - Elem];
}


template <typename T>
inline int TStack<T>::getStackSize() const
{
	return size;
}


template <typename T>
inline T* TStack<T>::getPtr() const
{
	return stackPtr;
}


template <typename T>
inline int TStack<T>::getNum() const
```

```
{
	return num;
}


template<typename T>
inline int TStack<T>::max_elem()
{
	int res = stackPtr[0];
	for (int i = 1; i < size; i++)
	{
		if (stackPtr[i] > res)
		{
			res = stackPtr[i];
		}
	}
	return res;
}

template<typename T>
inline int TStack<T>::min_elem()
{
	int res = stackPtr[0];
	for (int i = 1; i < size; i++)
	{
		if (stackPtr[i] < res)
		{
			res = stackPtr[i];
		}
	}
	return res;
}

     #endif
```