Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе №3

**Вычисление арифметических выражений (стеки)**

**Выполнила:**

студентка гр. 381903-3 Семибабнова А.В.

**Проверил:**

ассистент каф. МОСТ, ИИТММ Лебедев И.Г

Нижний Новгород

2021

Содержание

[Введение 3](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962758)

[Постановка задачи 4](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962759)

[Руководство программиста 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962761)

[Описание структур данных 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962762)

[Описание алгоритмов 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962763)

[Описание структуры программы 6](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962764)

[Заключение 7](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962765)

[Литература 8](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962766)

[Приложения 9](file:///C:\Users\Dragonsnom\Desktop\github\1-laba\mp2-lab1-set\report.doc#_Toc270962767)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Стек. С этой целью в лабораторной работе изучаются различные варианты структуры хранения стеков и разрабатываются методы и программы решения ряда задач с использованием стеков.

# Постановка задачи

В рамках лабораторной работы ставится задача реализации программ, обеспечивающих поддержку стеков, и разработки программных средств, производящих обработку арифметических выражений, включая проверку правильности записи выражения, перевод в постфиксную форму и вычисление результата.

# 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

//Доп задания

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

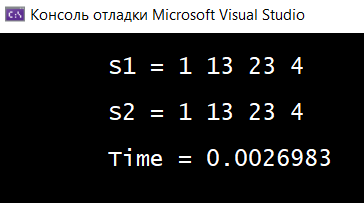


Рис.1. Консоль отладки

# 

# Заключение

В ходе выполнения данной лабораторной работы были реализован класс “Stack”. Были реализованы разные операции над стеками (удаление элемента, переполнения и тд).

# Литература

1. 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.
2. Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
3. Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
4. Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
5. Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
6. Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
7. Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
8. Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.

# Приложения

Stack.h

#ifndef STACK\_H

#define STACK\_H

#include <iostream>

#include <fstream>

using namespace std;

template <typename T>

class TStack

{

private:

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

//Доп задания

inline int min\_elem(); //Поиск минимального элемента

inline int max\_elem(); //Поиск максимального элемента

friend ostream& operator<<(ostream& out, const TStack& st)

{

for (int ix = st.num - 1; ix >= 0; ix--)

cout << st.stackPtr[ix] << endl;

return out;

}

};

#endif

Stack.cpp

#include "Stack.h"

// конструктор Стека

template <typename T>

TStack<T>::TStack(int maxSize) : size(maxSize) // инициализация константы

{

if (maxSize < 0)

{

throw logic\_error("ERROR");

}

stackPtr = new T[size]; // выделить память под стек

num = 0; // инициализируем текущий элемент нулем;

}

template <typename T> // контроль пустоты

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> // контроль переполнения

inline int TStack<T>::IsFull() const

{

for (int i = 0; i < size; i++)

{

if (stackPtr[i] != 0)

{

continue;

}

else

{

throw logic\_error("ERROR");

}

}

}

// конструктор копирования

template <typename T>

TStack<T>::TStack(const TStack<T>& otherStack) : size(otherStack.getStackSize()) // инициализация константы

{

stackPtr = new T[size]; // выделить память под новый стек

num = otherStack.getNum();

for (int ix = 0; ix < num; ix++)

{

stackPtr[ix] = otherStack.getPtr()[ix];

}

}

//деструктор

template <typename T>

TStack<T>::~TStack()

{

if (this->stackPtr != NULL)

{

delete[] stackPtr; // удаляем стек

}

num = 0;

}

// функция добавления элемента в стек

template <typename T>

inline void TStack<T>::put(const T& value)

{

if (num > size - 1 || num < 0)

{

throw logic\_error("ERROR");

}

stackPtr[num++] = value; // помещаем элемент в стек

}

// функция удаления элемента из стека

template <typename T>

inline T TStack<T>::deleteElem()

{

if (num < NULL)

{

throw logic\_error("ERROR");

}

stackPtr[--num]; // удаляем элемент из стека

return stackPtr[num];

}

// функция возвращает n-й элемент от вершины стека

template <class T>

inline const T& TStack<T>::Peek(int Elem) const

{

if (Elem > num)

{

throw logic\_error("ERROR");

}

return stackPtr[num - Elem]; // вернуть n-й элемент стека

}

// вернуть размер стека

template <typename T>

inline int TStack<T>::getStackSize() const

{

return size;

}

// вернуть указатель на стек (для конструктора копирования)

template <typename T>

inline T\* TStack<T>::getPtr() const

{

return stackPtr;

}

// вернуть размер стека

template <typename T>

inline int TStack<T>::getNum() const

{

return num;

}

//Поиск максимального элемента

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;

}

//Поиск минимального элемента

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;

}