МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

Высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Структура хранения данных: мультистек»**

**Выполнил:** студент группы 381706-1

Максимова Ирина Игоревна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc536381293)

[2. Постановка задачи 4](#_Toc536381294)

[3. Руководство пользователя 5](#_Toc536381295)

[4. Руководство программиста 6](#_Toc536381296)

[4.1. Описание структуры программы 6](#_Toc536381297)

[4.2. Описание структур данных 6](#_Toc536381298)

[4.3. Описание алгоритмов 7](#_Toc536381299)

[5. Эксперименты 9](#_Toc536381300)

[6. Заключение 10](#_Toc536381301)

[7. Литература 11](#_Toc536381302)

# Введение

**Мультистек** — структура данных, представляющая собой упорядоченный набор N стеков, фиксированного размера. Каждый отдельный стек организован по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Стеки хранятся в памяти друг за другом единым блоком, размер которого m:

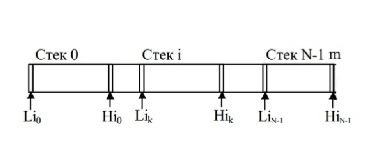


Рисунок 1. Структура памяти для хранения мультистека

*Где*  – индекс начала j-го стека, – индекс последнего элемент j-го стека.

Свойства такой структуры памяти:

1. – условие неподвижности первого стека;
2. – условие пустоты;
3. – условие неперекрытия;
4. – условие переполнения;

**Цель данной лабораторной работы** – разработка структуры хранения мультистека на массиве.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Разработка и реализация вспомогательного класса стека – TNewStack.
2. Разработка и реализация класса мультистека – TMultiStack.
3. Создание класса для обработки исключений – MyException, которые могут возникнуть при выполнении различных операций.
4. Разработка программы, демонстрирующей работу классa TMultiStack.
5. Реализация набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

Рассмотрим пример использования класса TMultiStack.

При запуске программы создается мультистек под хранение целых чисел. Размером мультистека - 12 элементов, количество стеков в мультистеке - 3. Затем мультистек заполняется числами от 1 до 12: числа от 1 до 4 помещаются в первый стек, 5 – 8 во второй стек, 9 – 12 – в третий. На каждой итерации на экран выводится сообщение о довалении числа в соответствующий стек. После того, как мультистек заполнен он выводится на консоль.

На следующем шаге программы из 1-го и 2-го стека в мультистеке извлекается по одному элементу. Полученный мультистек выводится на консоль.

Затем осуществляется попытка перепаковки мультистека путем добавления элемента «10» в 3-й стек. Полученный в результате перепаковки мультистек выводится на консоль. На этом работа программы прекращается.

# Руководство программиста

# *Описание структуры программы*

Программа состоит из следующих модулей:

* Модуль MultiStack. Содержит пример использования мультистека. Реализация в файле *main\_MultiStack.cpp.*
* Модуль MultiStackLib – статическая библиотека. Содержит файл *MultiStack.h*, в котором описан интерфейс и реализация шаблонного класса *TMultiStack* (2 конструктора и 8 методов), и *TNewStack.h*, в котором описан интерфейс и реализация шаблонного класса *TNewStack* (2 конструктора и 7 методов)
* Модуль MultiStackTest. Содержит 27 тестов, описанных в файле *MultiStackTest.cpp* и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, содержащая класс исключений.

## ***Описание структур данных***

#### **Класс TNewStack**

Класс *TNewStack* является шаблонным классом. Он наследуется от класса *TStack* со спецификатором public. Внутри класса определены 2 конструктора и 7 методов со спецификатором доступа public:

* *TNewStack(int \_size, T\* \_mas)* - конструктор по умолчанию.
* *TNewStack(TNewStack<T>& A)* – конструктор копирования.
* int GetFreeMem() – получить количество свободных ячеек в стеке.
* *T Get()* – забрать элемент из стека.
* *void Push(T \_A)* – положить элемент в стек.
* *int GetSize()* – получить размер стека.
* *int GetTop()* - получить позицию вершины стека.
* *void SetMas(int \_size, T\* \_mas)* – преобразование массива *mas* размером *size* в стек.
* *void PrintNewStack()* – вывод стека на консоль.

***Класс TMultiStack***

Класс *TMultiStack* является шаблонным классом. Внутри класса определены 4 поля и 2 метода со спецификатором доступа protected:

* int size – размер мультистека.
* T\* mas – указатель на область память под хранение мультистека.
* int n – количество стеков в мультистеке.
* TNewStack<T>\*\* stackMas – массив указателей на начало каждого стека в мультистеке.
* int CountFree() – количество свободных элементов в мультистеке.
* void Repack(int \_n) – перепаковка стека, с увеличением свободной памяти в n-м стеке.

В публичной зоне описаны 2 конструктора 6 методов:

* TMStack(int \_n = 1, int \_size = 10) - конструктор по умолчанию.
* TMStack(TMStack &A) - конструктор копирования.
* int GetSize() – возвращает размер мультистека.
* void Set(int \_n, T \_elem); - положить в n-й стек элемент elem.
* T Get(int \_n) – взять элемент из n-го стека.
* bool IsFull(int \_n) – проверка на полноту n-го стека.
* bool IsEmpty(int \_n) – проверка на пустоту n-го стека.
* void PrintMStack() – вывод мультистека на консоль.

## ***Описание алгоритмов***

**Перепаковка мультистека.**

Потребность в перепаковке мультистека возникает, когда нужно добавить элемент в n-й стек, а он уже переполнен, в то время как в других стеках еще осталась хотя бы одна свободные ячейка.

Опишем алгоритм перепаковки мультистека относительно k-го стека:

* + - 1. Количество свободных ячеек, которые можно добавить в каждый стек add\_ev находим путем деления количества свободных ячеек во всем мультистеке free на количество стеков n:

|  |  |
| --- | --- |
|  | (1) |

* + - 1. Увеличиваем старые размеры size[i] стеков на add\_ev, получаем новые размеры стеков new\_size[i]:

|  |  |
| --- | --- |
|  | (2) |

* + - 1. Если количество свободных ячеек во всем мультистеке free не кратно количеству стеков n, то оставшиеся свободные ячейки добавляем в k-й стек:

|  |  |
| --- | --- |
|  | (3) |

* + - 1. Определяем новый индекс начала каждого стека new\_ind[i], с учетом их размера new\_size[i]:

|  |  |
| --- | --- |
|  | (4) |
|  | (5) |

* + - 1. Перемещение элементов стеков на новые позиции:
  1. Если индекс нового начала i-го стека new\_ind[i] не больше, чем индекс старого начала i-го стека ind[i], то копируем элементы по порядку, в котором они хранятся в старом стеке.
  2. Иначе идем по новым позициям стеков до тех пор, пока не выполняется 5.1. Затем копируем элементы, в котором они хранятся в старом стеке, НО в обратном порядке. Иначе произойдет затирание памяти.

# Эксперименты

В данном разделе изучается перемещение стеков впарво и влево во время работы алгоритма перепаковки мультистека.

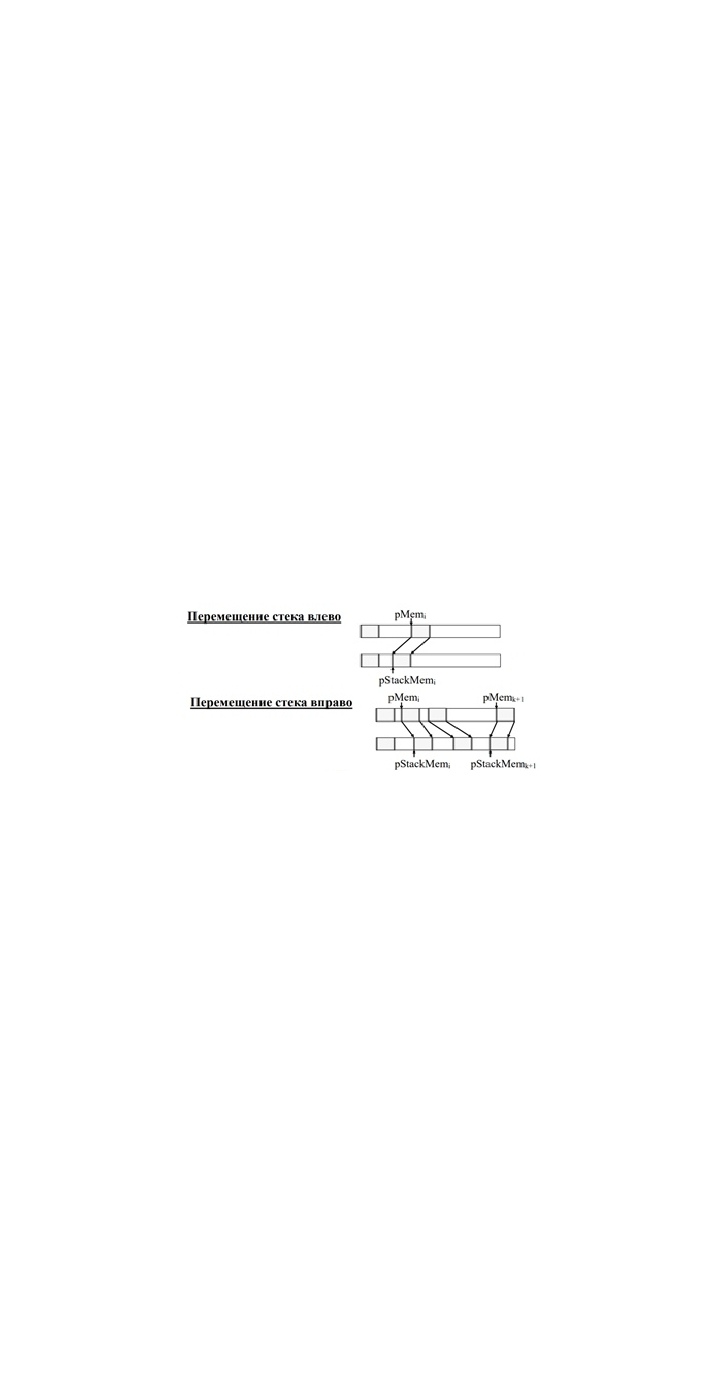


Рисунок . Перемещение стеков в мультистеке

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10 Домашняя
2. Процессор: Intel(R) Core™ i5-7200U CPU @ 2.70 GHz
3. Версия Visual Studio: 2017

Ниже, в таблице, приведены результаты экспериментов. Был создан мультистек из 10 тыс. стеков, в каждом стеке по 5 элементов. Индекс i указывает на номер стека, вплоть до которого все стеки заполнялись полностью, а оставшиеся стеки заполнялись наполовину (в этом случае, при попытке добавить элемент в один из полных стеков, будет вызван метод перепаковки и все стеки сместятся вправо), или наоборот – до i-го полностью, остальные наполовину (в этом случае, при попытке добавить элемент в один из полных стеков, будет вызван метод перепаковки и все стеки сместятся влево).

|  |  |  |  |
| --- | --- | --- | --- |
| Индекс | 1000 | 6000 | 9000 |
| Время перемещения стеков влево в млс | 1.36 | 1.39 | 1.44 |
| Время перемещения стеков вправо в млс | 1.34 | 1.42 | 1.45 |

Таблица 1. Смещение стеков вправо и влево.

Как видно из таблицы время перемещение стека вправо и влево одинаково по времени выполнения.

# Заключение

В ходе выполнения лабораторной был произведен анализ задачи - установлено понятие мультистека. Была разработана библиотека MultiStackLib, содержащая шаблонный класс мультистека TMultiStack и вспомогательный класс TNewStack. В библиотеке реализованы методы работы с мультистеком описанные в разделе «Структуры данных».

Программная реализация стека на массиве была продемонстрирована на примере, описывающем основные методы класса TMultiStack.

Разработаны и доведены до успешного выполнения тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework.

# Литература

* Книги

A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.

Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.

* Ссылки в Internet

Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [http://www.itmm.unn.ru/files/2018/11/Primer-1.5.-Struktury-hraneniya-neskolkih-stekov-v-obshhej-pamyati.pdf], 2015.

Википедия: свободная электронная энциклопедия: на русском языке: https://ru.wikipedia.org/wiki/Стек