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 мультистека»**

**Выполнил:** студент группы 381706-1

Силенко Дмитрий Игоревич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533165735)

[2. Постановка задачи 4](#_Toc533165736)

[3. Руководство пользователя 5](#_Toc533165737)

[4. Руководство программиста 6](#_Toc533165738)

[4.1. Описание структуры программы 6](#_Toc533165739)

[4.2. Описание структур данных 6](#_Toc533165740)

[4.3. Описание алгоритмов 6](#_Toc533165741)

[5. Заключение 8](#_Toc533165742)

# Введение

**Основная цель данной работы** – разработка динамической структуры данных – мультистека на массиве.

Но для начала необходимо разобраться, что такое мультистек и как он работает.

**Мультистек** – это структура данных, состоящая из упорядоченного набора из N последовательно хранящихся стеков фиксированного размера. Каждый стек реализован по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). По сути это означает, что мы будем иметь доступ только к последнему добавленному элементу.

Стеки хранятся в памяти единым блоком друг за другом. Размер такого блока m.

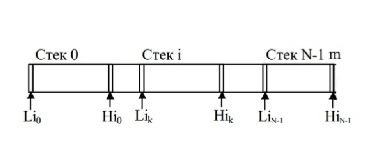


Рисунок 1 Представление мультистека в памяти компьютера

Здесь, – 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: ;

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Разработка и реализация вспомогательного класса стека TNewStack, а так же класса мультистека TMStack
2. Пример программы, демонстрирующая работу класса TMStack.
3. Написание набора автоматических тестов с использованием Google C++ Testing Framework и проверка работоспособности методов классов.

# Руководство пользователя

При запуске программы создается мультистек, состоящий из 5 стеков и общей длинной 10. После чего он постепенно заполняется значениями от 1 до общей длины мультистека. После чего полностью выводится на экран.

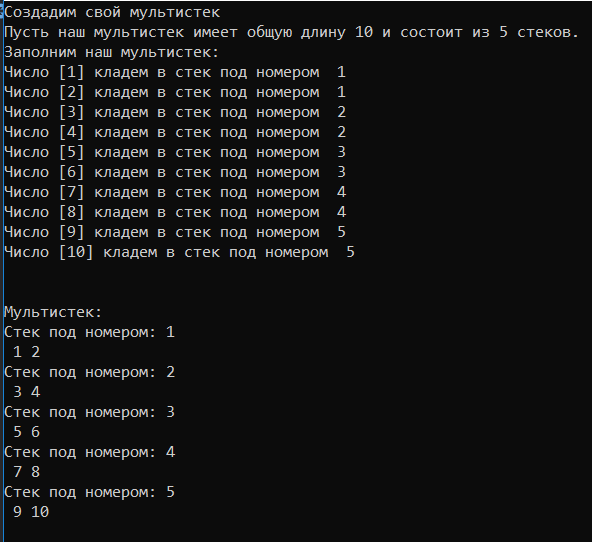


Рисунок 2 Создание мультистека и вывод его на экран

Далее, пользователю предлагается ввести номер стека, из которого будет изъят один элемент. Этот изъятый элемент выводится на экран, вместе с обновленным мультистеком уже без этого элемента.

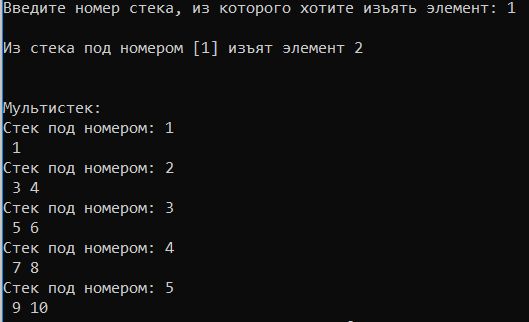


Рисунок 3 Изъятие элемента из введенного стека

Затем, пользователь может ввести новый элемент в мультистек. Для этого его просят ввести номер стека, куда бы он хотел поместить свой элемент и его значение. Если пользователь выберет стек, который уже был заполнен, то память в мультистеке перепакуется.

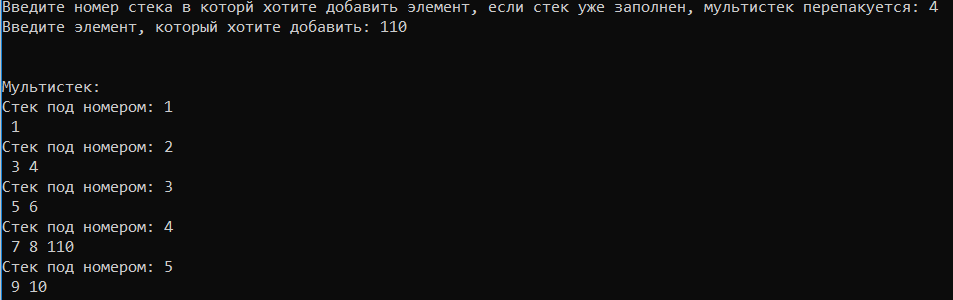


Рисунок 4 Ввод нового элемента в выбранный пользователем стек

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль MultiStackLib. Статическая библиотека. Включает в себя заголовочный файл NewStack.h, в котором описаны методы с реализаций вспомогательного шаблонного класса *TNewStack.* А так же файл MultiStack.h, в котором описаны методы с реализацией шаблонного класса *TMStack*.
* Модуль MultiStackTest. Набор тестов для класса TMStack. Включает в себя файл MultiStack*Test.cpp.* Разработаны они с помощью использования Google C++ Testing Framework.
* Модуль MultiStack. Пример использования списка. Включает в себя файл с реализацией MultiStack*\_main.cpp.*

## Описание структур данных

*Класс TNewStack наследуется от класса TStack:*

Конструкторы:

TNewStack(int \_length, T\* \_elem);

TNewStack(TNewStack<T>& newst);

Методы:

int GetFreeCell(); - получить количество свободных ячеек в стеке

T Get(); - изъять элемент из стека

int GetLength(); - получить длину стека

int GetTop(); - получить позицию вершины стека

void Push(T \_elem); - положить элемент в стек

void SetMas(int \_length, T\* \_elem); - преобразовать массив в стек

void PrintNewStack(); - вывод стека на экран

*Класс TMStack наследуется от класса TNewStack:*

Поля:

int length; - длина мультистека

T\* elem; - массив элементов мультистека

int countst; - количество стеков в мультистеке

TNewStack<T>\*\* stackMas - массив указателей на начало каждого стека в мультистеке

Защищенные методы:

int CountFree(); - количество свободных элементов в мультистеке

void Repack(int \_st); - перепаковка стека с увеличением свободной памяти в стеке st

Конструкторы:

TMStack(int \_countst = 1, int \_length = 10);

TMStack(TMStack &mst);

Методы:

int GetLength(); - получить длину мультистека

T Get(int \_st); - получить элемент из стека st

void Set(int \_st, T \_elem); - положить в стек st элемент elem

bool IsFull(int \_st); - проверка на полноту стека st

bool IsEmpty(int \_st) – проверить на пустоту стек st

void PrintMStack(); - вывод мультистека на экран

## Описание алгоритмов

**Перепаковка мультистека.**

К перепаковке необходимо прибегать, когда необходимо положить новый элемент в стек, который уже полон, в то время как в остальных есть хотя бы одна свободная 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.

# Заключение

Эта лабораторная работа дала возможность более детально разобраться с устройством работы мультистека на массиве. В ходе выполнения, был реализован вспомогательный шаблонный класс стека TNewStack, а также основной шаблонный класс мультистека TMStack, описанны они в специально разработанной библиотеке MultiStackLib.

Помимо этого, для проверки работоспособности всех методов, были написаны автоматические тесты, реализованные с использованием Google C++ Testing Framework. Пример использования списка для пользователя также написан и успешно работает.

1. **Литература**
2. 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.
3. Википедия: свободная электронная энциклопедия: на русском языке: https://ru.wikipedia.org/wiki/Стек