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

Остапович Денис Евгеньевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc2286718)

[2. Постановка задачи 3](#_Toc2286719)

[3. Руководство пользователя 3](#_Toc2286720)

[4. Руководство программиста 3](#_Toc2286721)

[4.1 Описание структуры программы 3](#_Toc2286723)

[4.2 Описание структур данных 3](#_Toc2286724)

[4.3 Описание алгоритмов 3](#_Toc2286725)

[5. Заключение 3](#_Toc2286726)

[6. Литература 3](#_Toc2286727)

# Введение

**Стек** — [абстрактный тип данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), представляющий собой [список элементов](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), организованных по принципу FILO – первым вошел, последним вышел. Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Из стека нельзя извлечь произвольный элемент. Есть только вершина стека, и мы можем или положить туда элемент, или извлечь его.

Мультистек - [абстрактный тип данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), представляющий собой набор из стеков. Стеки хранятся в памяти последовательно.

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

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Описание вспомогательного класса стека – TNewStack.
2. Описание класса мультистека – TMultiStack.
3. Разработка программы, демонстрирующей работу классa TMultiStack.
4. Реализация набора автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

После запуска программы автоматически создаётся и заполняется мультистек. После этого результат выводится на экран. После этого производится поэлементный вывод мултистека, производится перепаковка и её результат выводится на экран. Программа завершается.

# Руководство программиста

# Описание структуры программы

Программа состоит из следующих модулей:

* Модуль MultiStack. Содержит пример использования мультистека. Реализация в файле *main.cpp.*
* Модуль MultiStackLib – статическая библиотека. Содержит файл multistack*.h*, в котором описан интерфейс и реализация шаблонного класса *TMultiStack* (2 конструктора и 8 методов), и *newstack.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.

# 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 Pop()* – 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 Print()* – вывод стека в 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:

* *TMultiStack* (int \_n = 1, int \_size = 10) - конструктор по умолчанию.
* *TMultiStack* (*TMultiStack* &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 Print() – вывод мультистека в 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.

# Заключение

В ходе выполнения лабораторной работа 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. Разработаны тесты для данного программного проекта с использованием Google C++ Testing Framework.

# Литература

* Книги

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

1. Bjarne Stroustrup The C++ programming language

* 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