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**

**«Мультистек»**

**Выполнил:** студент группы 381706-1

Корнев Никита Алексеевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc1061278)

[2. Постановка задачи 4](#_Toc1061279)

[3. Руководство пользователя 5](#_Toc1061280)

[4. Руководство программиста 6](#_Toc1061281)

[4.1. Описание структуры программы 6](#_Toc1061282)

[4.2. Описание структур данных 7](#_Toc1061283)

[4.3 Описание алгоритмов 9](#_Toc1061284)

[5. Заключение 11](#_Toc1061285)

[6. Литература 12](#_Toc1061286)

# Введение

Структура данных (англ. data structure) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных данных в вычислительной технике. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс. Одной из важных структур является список.

Стек (от англ. stack — стопка) — структура данных, представляющая из себя упорядоченный набор элементов, в которой добавление новых элементов и удаление существующих производится с одного конца, называемого вершиной стека. Притом первым из стека удаляется элемент, который был помещен туда последним, то есть в стеке реализуется стратегия «последним вошел — первым вышел» (last-in, first-out — LIFO). Примером стека в реальной жизни может являться стопка тарелок : когда мы хотим вытащить тарелку, мы должны снять все тарелки выше.

В данной лабораторной работе нам предстоит реализовать мальтистек и стек, для работы с ним.

# Постановка задачи

1. Написать класс, реализующий стек.
2. Написать класс на основе стека, реализующий мультистек.
3. Написать тесты на основе Google Tests для проверки работы класса.

# Руководство пользователя

Чтобы использовать данный класс в своем проекте, необходимо подключить библиотеку «MultiStack.h». Библиотека позволяет:

* Создавать объекты типа *мультистек*:

MultiStack ms1; **конструктор по умолчанию**

MultiStack ms(10, 10); **конструктор с параметром**

MultiStack ms3 (ms2); **конструктор копирования**

* Добавлять элементы в *мультистек*:

ms1.Set(10, 10);

ms1.Get(5);

* Проверять на *пустоту*:

A = ms1.IsEmpty();

* Проверятьна *полноту*:

A = ms1.IsFull();

# Руководство программиста

## Описание структуры программы

Проект состоит из следующих модулей:

1. MultiStack

Модуль cодержит файл код «main.cpp», в котором продемонстрирован пример использования структуры.

1. MultiStack Lib

Модуль содержит файл заголовок «MultiStack.h», описывающий структуру «мультистек», файл заголовок «NewStack.h, описывающий структуру элементов мультистека, а также файл кода «main.cpp». Так как класс шаблонный, main содержит лишь подключение.

1. MultiStack Test

Модуль содержит файлы кода «test\_main.cpp», «test\_multistack.cpp». В последнем реализованы тесты для проверки корректности работы методов данного класса.

## Описание структур данных

#### *Структура NewStack*

Поля со спецификатором доступа «public»:

#### TNewStack(int \_size, T\* \_mas); конуструктор по-умолчанию

#### TNewStack(TNewStack<T>& A); конструктор копирования

#### int GetFreeMem(); проверка на наличие свободной памяти

#### T Get(); метод, возращающий значение последнего элемента стека

#### void Push(T \_A); метод, позволяющий положить значение в стек

#### int GetSize(); метод, возвращающий размер стека

#### int GetTop(); метод, возращающий значение верхушки

#### void SetMas(int \_size, T\* \_mas); метод, позволяющий положить в стек массив

#### *Структура MultiStack*

Поля со спецификатором доступа «protected»:

* **int size;** размер
* **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 GetFreeMem();** количество свободных элементов в мультистеке.
* **void Repack(int \_n);** перепаковка стека, с увеличением свободной памяти в n-м стеке.

Поля со спецификатором доступа «public»:

* **TMStack(int \_n, int \_size);** конструктор с параметром
* **TMStack(TMStack &A);** констурктор копирования
* **int GetSize();** метод возвращающий размер мультистека
* **void Set(int \_n, T \_elem);** метод, позволяющий положить элемент в мультистек
* **T Get(int \_n);** метод, позволяющий взять элемент из мультистека
* **bool isFull(int \_n);** проверка на полноту
* **bool isEmpty(int \_n);** проверка на пустоту

## Описание алгоритмов

#### Алгоритм перепаковки

Допустим, нам необходимо поместить элемент A в n-ый стек мультистека. Проверяем n-ый стек на наличие свободного места. Но все место в стеке n уже заполнено, зато в других стеках есть незанятые ячейки. Возникает необходимость перепаковки мультистека, чтобы освободить место в n-ом стеке. Рассмотрим код алгоритма.

template <class T>

void TMStack<T>::Repack(int \_n)

{

int FM = GetFreeMem(); // находим кол-во свободных ячеек во всем мультистеке

int add\_ev = FM / n; // находим кол-во свободных ячеек, которые можно добавить в каждый стек

int add\_full = FM % n;

int\* new\_size = new int[n];

T\*\* new\_start = new T\*[n];

T\*\* old\_start = new T\*[n];

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

new\_size[i] = add\_ev + stackmas[i]->GetTop(); // Увеличиваем старые размеры стеков

new\_size[\_n] += add\_full; // Если кол-во свободных ячеек во всем мультистеке не кратно кол-ву стеков n, то оставшиеся свободные ячейки добавляем в \_n-й стек:

new\_start[0] = old\_start[0] = mas; // Определяем новое начало

for (int i = 1; i < n; i++) // Определяем новое начало для каждого стека с учетом их размера

{

new\_start[i] = new\_start[i - 1] + new\_size[i - 1];

old\_start[i] = old\_start[i - 1] + stackmas[i - 1]->GetSize();

}

for (int i = 0; i < n; i++) // Если новый индекс начала i-го стека меньше, чем индекс старого начала, то копируем элементы по порядку, в котором они хранились раньше

{

if (new\_start[i] <= old\_start[i])

for (int j = 0; j < stackmas[i]->GetTop(); j++)

new\_start[i][j] = old\_start[i][j];

else // В противном случае идем по новым позициям стеков до тех пор, пока не выполнится условие. Затем копируем элементы, в котором они хранились раньше, но в обратном порядке

{

int s = i + 1;

for (s; s < n; s++)

if (new\_start[s] <= old\_start[s])

break;

for (int j = s - 1; j >= i; j--)

for (int r = stackmas[j]->GetTop() - 1; r >= 0; r--)

new\_start[j][r] = old\_start[j][r];

i = s - 1;

}

}

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

stackmas[i]->SetMas(new\_size[i], new\_start[i]);

delete[] new\_size;

delete[] new\_start;

delete[] old\_start;

}

# Заключение

В данном лабораторной работе мне удалось:

* Успешно реализовать класс для стека
* Успешно реализовать класс для мультистека
* Продемонстрировать пример использования данного класса
* Написать тесты на основе Google Tests для проверки корректной работы данного класса

# Литература

Интернет-источники:

1. <https://neerc.ifmo.ru/wiki/index.php?title=%D0%A1%D1%82%D0%B5%D0%BA>
2. <https://ru.wikipedia.org/wiki/Стек>
3. <https://habr.com/ru/post/341586/>

Книги:

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
2. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.