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**

**«Структур хранения нескольких стеков в**

**общей памяти: мультистек»**

**Выполнил:** студент группы 381706-1

Нечаева Екатерина Владимировна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1 Введение 3](#_Toc954265)

[2 Постановка задачи 4](#_Toc954266)

[3 Руководство пользователя 5](#_Toc954267)

[4 Руководство программиста 6](#_Toc954268)

[4.1 Описание структуры программы 6](#_Toc954269)

[4.2 Описание структур данных 6](#_Toc954270)

[4.3 Описание алгоритмов 7](#_Toc954271)

[5 Заключение 9](#_Toc954272)

[6 Литература 10](#_Toc954273)

# Введение

**Мультистек** — структура хранения данных, представляющая собой упорядоченный набор любого количества стеков, фиксированного размера. Будем предполагать, что все стеки используются с одинаковой интенсивностью - память распределяется всем стекам поровну.

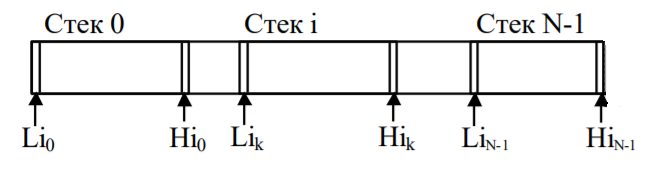


Рисунок 1. Структура хранения стеков в памяти.

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

Свойства такой структуры памяти:

* Li0 = 0 – условие неподвижности первого стека;
* Hij = Lij – 1 – условие пустоты;
* Hij < Lij – условие неперекрытия;
* Hij = Lij – 1 – условие переполнения;

# Постановка задачи

Для работы с мультистеком будут реализованы следующие операции:

* Подсчет количества свободных элементов,
* Перепаковка заполненного стека с данным номером,
* Добавление и извлечение элемента в выбранный стек,
* Проверка на пустоту/полноту выбранного стека.

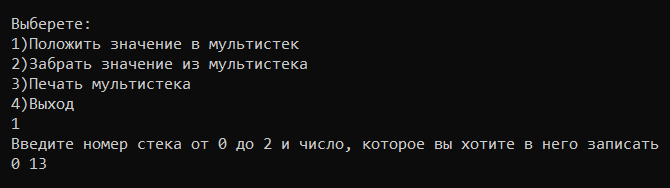
Для этого нужно реализовать три класса:

* TNewStack, который отвечает за инициализацию стеков,
* TMultiStack, класс для реализации мультистека,
* TException, класс исключений.

# Руководство пользователя

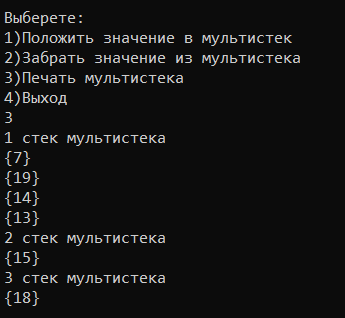
Программа начинает свою работу с вывода на консоль меню, в котором пользователю предлагается выбрать одну из операций над мультистеком.

Для начала надо записать что-то в мультистек, для этого выбрав 1 пункт, пользователю предлагается выбрать место, т.е номер стека, куда положить значение.



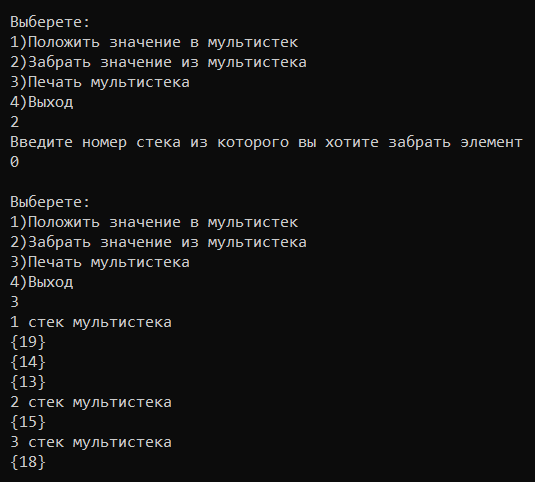
*Рисунок 2. Пример ввода значения в мультистек.*

После занесения данных в мультистек, пользователь может вывести его на экран.



*Рисунок 3. Печать мультистека на консоль.*

Также, выбрав 2 пункт меню, можно забрать из выбранного стека элемент и вывести новый мультистек.

**

*Рисунок 4. Пример извлечения элемента из мультистека.*

# Руководство программиста

## **Описание структуры программы**

Программа состоит из следующих модулей:

* Модуль libmultistack – статическая библиотека. Содержит заголовочный файл multistack.h, в котором реализованы шаблонные классы TMStack и TNewStack*.*
* Модуль multistack. Содержит пример использования мультистека (mainmultistack.cpp*).*
* Модуль multistackTest. Содержит тесты, описанные в файле multistackTest.cpp.
* Модуль libthrows – библиотека для исключений. Содержит заголовочный файл throws.h, в котором он и реализован.

## **Описание структур данных**

#### **Класс TNewStack**

Класс TNewStack является шаблонным классом, который наследуется со спецификатором доступа public от реализованного ранее TStack<T>. Поэтому полей у данного класса нет.

Содержит методы:

* TNewStack(int \_size = 0, T\* \_mas = 0); //конструктор инициализатор
* TNewStack(TNewStack<T> &A); //конструктор копирования
* int Free(); //количество свободных ячеек в стеке
* void SetMas(int n, T\* m); // преобразование массива m размером n в стек
* int GetSize(); //получить размер стека
* int GetTop(); //позиция вершины стека

**Класс TMultiStack**

*Поля класса:*

* int size; //размер мультистека
* T\* Mas; //указатель на область памяти под хранение мультистека
* int n; //количество стеков в мультистеке
* TNewStack<T>\*\* h; //массив указателей на начало каждого стека в мультистеке

*Методы: первые два со спецификатором protected, остальные – public*

* int GetFreeMem(); //общее количество свободных ячеек в каждом стеке
* void Repack(int \_n); //перепаковка стека с номером \_n
* TMStack(int \_size, int \_n); //конструктор инициализатор с количеством стеков \_n и его размером \_size
* TMStack(TMStack<T> &A); //конструктор копирования
* void Put(int \_n, const T &A); //положить в стек номер \_n значение A
* T Get(int \_n); //взять из стека с номером \_n элемент
* bool IsFull(int \_n); //проверка на полноту
* bool IsEmpty(int \_n); //проверка на пустоту
* int GetSizeSt(int \_n); //получить размер стека с номером \_n
* void PrintTM(); //печать мультистека
* int Count(); //количество стеков

## **Описание алгоритмов**

**Создание мультистека.**

Для начала проверяем входные данные, если они не удовлетворяют условиям неотрицательности, то бросается исключение. Иначе соответствующим полям присваиваются входные значения.

Следующим этапом создается массив размеров стеков *p* и каждому элементу p[i] присваивается значение – размер каждого стека. Делается это следующим образом: первому стеку считается по формуле *(size / n) + (size%n),* т.е находится целое число от деления размера мультистека *size* на количество стеков *n* и прибавляется остаток от деления*.* Остальным же присваивается целая часть от этого деления*.*

Теперь нужно указать адрес на начало каждого стека, за это отвечает переменная *h,* которая хранит массив указателей. Для этого первому элементу h[0] присваивается указатель на общую память (*Mas*), а каждому следующему будет присваиваться этот указатель, сдвинутый на суммарный размер предыдущего стека.

**Конструктор копирования.**

В качестве параметра он принимает мультистек, назовем его *А*.

После чего, данные мультистека *А* копируются в новый, а именно: размер, количество стеков и указатель, в котором хранится мультистек, приравниваются *size = A.size; n = A.n, Mas=A.Mas.*

Следующим этапом создается указатель *p*, который отвечает за массив размеров стеков, размером *n* и каждому его элементу ставится в соответствии размер каждого стека мультистека *А* с помощью функции *GetSize().*

После чего *h* - массиву указателей на начало каждого стека в мультистеке нужно присвоить начало каждого стека *А.* Делается это следующим образом: первому присваиваем указатель на начало общей памяти, а остальным с помощью метода *SetMas()* класса TNewStack, который в качестве параметров принимает размер каждого стека, в нашем случае *p[i]*, и адрес начала каждого стека, который считается с помощью сдвига указателя общей памяти на суммарный размер предыдущего стека.

**Перепаковка мультистека.**

Выполняется при попытке вставки нового значения в стек s, у которого отсутствует свободная память.

Чтобы воспользоваться данным методом, нужно проверить, есть ли свободные ячейки памяти в мультистеке. Если таковых нет, бросается исключение. Иначе создаются две целочисленные переменные (*add\_every* и *add\_full*), первая отвечает за целое деление свободных ячеек на количество стеков, вторая переменная – остаток от этого деления. Также создается массив для новых размеров стеков (*newSize*) и два массива указателей (*old\_start* и *new\_start*), один указывает на старое начало каждого стека, второй указывает на новое начало измененных стеков. После чего, заполняем данные массивы:

В цикле заполняем *newSize[i] = add\_every + h[i]->TNewStack<T>::GetTop() + 1* (т.е увеличиваем старые размеры стеков от вершины на *add\_every*) Также к стеку, который был заполнен, прибавляется остаток от деления ( *add\_full*).

Теперь нужно заполнить массив указателей на старое и новое начало. Для этого первому элементу массива *old\_start* присваивается указатель на общую память, тем самым копируя все элементы старого мультистека. А каждый следующий элемент получается присваиванием предыдущего указателя, сдвинутого на старый размер предыдущего стека с помощью функции *GetSize()*. Аналогично поступает с массивом new\_start: первому элементу присваивается указатель на общую память, а каждому следующему присваивается предыдущий указатель, сдвинутый на новый размер предыдущего стека с помощью массива *newSize.*

Теперь нужно переместить элементы на новые позиции в массиве *new\_start.* Для этого надо ввести два условия:

1. Если новое начало стека меньше или равно старому началу, то мы просто копируем элементы из *old\_start* по порядку.
2. Иначе переходим к следующему стеку и так до тех пор, пока 1 условие не будет выполняться. Тогда найденным стекам присваиваются элементы старого начиная с конца (вершины стека) до первого элемента. Т.е, так мы копируем элементы в обратном порядке, чтобы не произошло затирание памяти.

После всех манипуляций мы получили новые данные о стеках и нужно эти данные присвоить нашему мультистеку.

# Заключение

В ходе выполнения данной лабораторной была реализована структура хранения данных в виде мультистека. Были реализованы методы, позволяющие производить операции над стеками, а именно: добавление и извлечение элемента в выбранный стек, перепаковка заполненного стека.

Были разработаны тесты, проверяющие корректную работу методов данных классов, с помощью Google C++ Testing Framework.

# Литература

Гергель В.П. Методические материалы по курсу «Методы программирования 2» [Электронный ресурс]

<http://www.itmm.unn.ru/files/2018/11/Primer-1.5.-Struktury-hraneniya-neskolkih-stekov-v-obshhej-pamyati.pdf>

Структура данных стек [Электронный ресурс]

<https://studfiles.net/preview/1417256/page:50/>