Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

Высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

Мультистек

Отчет по лабораторной работе

Выполнил:

студент ИИТММ гр. 381706-2

Антипин А.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc819699)

[Постановка целей и задач 4](#_Toc819700)

[Руководство пользователя 5](#_Toc819701)

[Руководство программиста 6](#_Toc819702)

[Описание структуры программы 6](#_Toc819703)

[Описание структур данных 7](#_Toc819704)

[Описание алгоритмов 8](#_Toc819705)

[Эксперименты 10](#_Toc819706)

[Время выполнения 10](#_Toc819707)

[Заключение 11](#_Toc819708)

[Литература 12](#_Toc819709)

# Введение

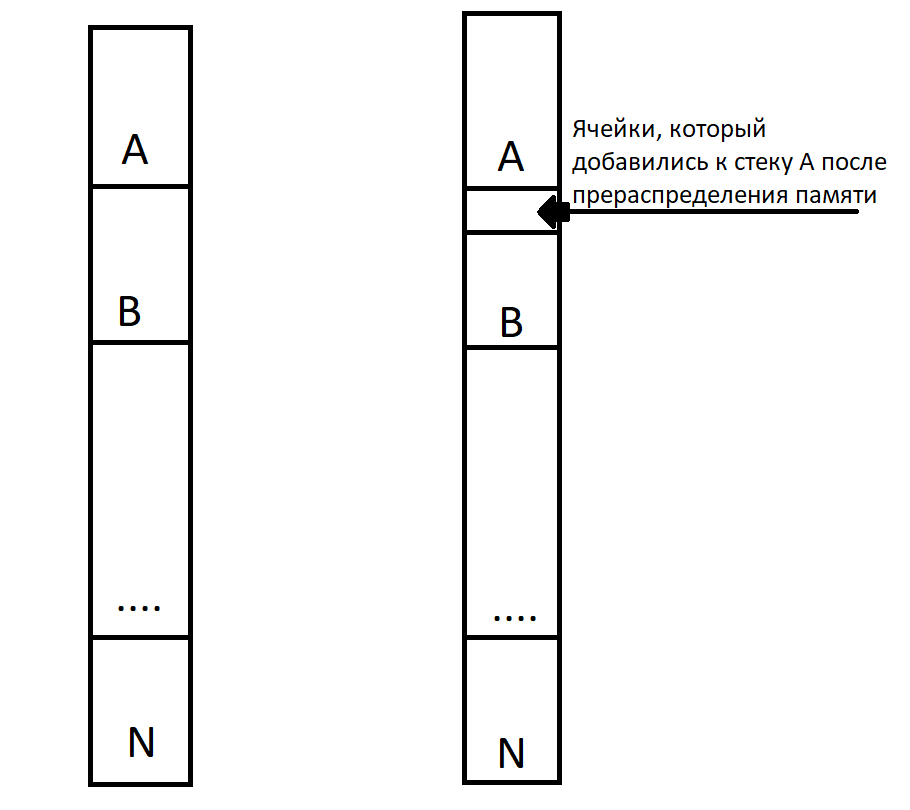
В одной из предыдущих работ мы разобрали, что такое стек, как он работает, его основные функции и применение на практике. Но в некоторых задачах требуется применение системы из нескольких стеков, хранящих схожие типы данных. В таких задачах может использоваться мультистек. Отличительной особенностью мультистека является то, что это несколько стеков, для которых выделена одна общая память, и если в одном из стеков заканчивается память, а в других еще есть место, то память перераспределяется, предоставляя место для хранения элемента в нужном стеке (рис. 1).

рис. 1 (принцип работы мультистека)

# Постановка целей и задач

Основной целью лабораторной работы является создание структуры данных типа мультистек и реализация таких методов, как:

* Добавление элемента в нужный стек;
* Возврат элемента из нужного стека;
* Метод перераспределения памяти.

Для реализации алгоритмов будет использоваться 2 шаблонных класса:

* Multistack;
* MStack, который наследуется от класса Stack с добавлением методов для управления памятью.

Для проверки правильности работы этих классов будут написаны тесты с использованием фреймворка Google Test, а также тестовый образец программы, которая использует данный класс.

# Руководство пользователя

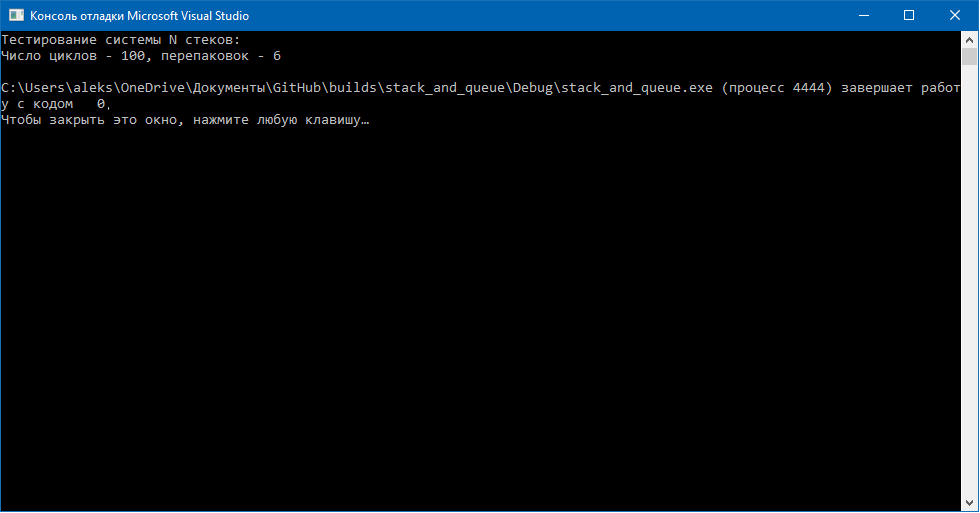
После запуска программы пользователя встречает консольное окно (рис. 2):

рис. 2 (вывод работы мультистека для пользователя)

в котором представлено количество циклов записи в мультистек общего размера 100 с количеством стеков равным четырем, ста элементов, причем в первый стек записывается 40 элементов, во второй 30, третий – 20, четвертый - 10.

# Руководство программиста

## Описание структуры программы

Мультистек будет реализован с помощью набора указателей на стеки и общей памяти, выделенной сразу на все стеки:

Таким образом для реализации алгоритмов будет использовано 2 класса:

* Класс «MStack», который наследуется от класса Stack с добавлением некоторых методов.
* Класс «Multistack», который будет использовать класс MStack.

А также проект использующий фреймворк Google Test, для проверки правильности работы этих классов и тесовый проект, который будет показываться пользователю.

Древо классов

**Класс gtest**

**test\_main.cpp**

**Multistack\_test.cpp**

**Проект для пользователя**

**main.cpp**

**Multistack.h**

**Класс Multistack**

**MStack.h**

**Класс MStack**

**Класс MStack:**

Класс, который наследуется от класса Stack с добавлением методов изменения указателя на начало стека и метода изменения длины стека.

**Класс Multistack:**

Класс Multistack содержит реализацию структуры данных мультистек и методов, таких как: положить в стек, забрать из него и перераспределить память.

**Класс gtest:**

Класс gtest реализует тестирование класса Multistack, по средствам фреймворка Google Test. Тесты пишутся для каждого метода классов, каждого ветвления этих методов и для всех возможных исключений этих методов.

**Проект Multistack:**

В данном проекте реализован примет использования мультистека, который в конечном итоге и будет доступен пользователю.

## Описание структур данных

**Класс Multistack:**

template <class T> - шаблон класса T

int Counts - количество стеков;

int L - длина общего стека;

int \*Len; - массив длина каждого стека;

T\*\* Index – массив указателей на начало каждого стека;

T\* GeneralStack – память под общий стек;

Stack<T>\*\* stacks – массив указателей на стеки;

int ResizeCount – количество перераспределений памяти;

Exceptions\_from\_stack\_queue\_multystack exception – переменная для вызова исключений.

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| MStack<T>::MStack() | Конструктор по умолчанию для класса MStack. |
| MStack<T>::MStack(T\* \_Mem, const int \_Len) | Конструктор с параметрами для класса MStack, первым параметром которого является указатель на элемент стека, а второй – длина стека. |
| MStack<T>::~MStack() | Деструктор для наследника стека. |
| void MStack<T>::SetMem(T\* \_Mem, int \_Len) | Метод, который изменяет параметры из предыдущего метода. |
| void MStack<T>::SetLen(const int N) | Метод, который изменяет длину стека. |
| T\* MStack<T>::GetMem() | Метод, который возвращает указатель на первый элемент стека. |
| T MStack<T>::GetValue(const int N) | Метод, который может вернуть любой элемент из стека, игнорирую принцип доступа к элементам стека. |
| void MStack<T>::SetIndex(const int \_i) | Метод, который позволяет изменить индекс в текущем стеке. |
| Multistack<T>::Multistack(int \_Counts, int \_L) | Конструктор с параметрами для класса Multistack. |
| Multistack<T>::Multistack(const Multistack& A) | Конструктор копирования класса Multistack. |
| Multistack<T>::~Multistack() | Деструктор Multistack. |
| int Multistack<T>::CalcFree() | Функция, которая позволяет сосчитать количество оставшийся памяти во всех стеках. |
| bool Multistack<T>::Resize(const int \_i) | Метод, который перераспределяет память между стеками и возвращает 1 в случае успеха, 0 в случае, если памяти нет. |
| void Multistack<T>::Put(const int i, const T& A) | Метод, который позволяет положить элемент в определенный стек. |
| T Multistack<T>::Get(const int i) | Метод, который возвращает элемент из нужного стека. |
| bool Multistack<T>::IsFull(const int i) | Метод, который позволяет узнать не заполнен ли текущий стек. |
| bool Multistack<T>::IsEmpty(const int i) | Метод, который проверяет пустой ли текущий стек или нет. |

## Описание алгоритмов

**Подробное описание некоторых методов**

Конструктор с параметрами для класса Multistack:

* Метод принимает целочисленные значения количества стеков и общей длины выделяемой памяти;
* Если один из параметров отрицательный или количество стеков больше общего количества элементов стеках, то вызываются исключения;
* Параметры, переданные конструктору, присваиваются соответствующим полям класса мультистек, выделяется память под элементы стеков;
* Выделяется память под массив длин стеков, которые вычисляются по принципу: «каждому поровну», а первому стеку присваиваются остатки памяти.
* Выделяется память под указатели на начало каждого стека и тут же им присваиваются значения;
* Выделяется память под массив указателей на стеки, которым потом присваиваются нужные указатели на первый элемент и длина.

Перераспределение памяти между стеками:

* Метод принимает целочисленный параметр – номер стека в котором не хватает памяти;
* Создаются переменные для циклов и сообщения прошло ли перераспределение памяти успешно, считается количество свободной памяти;
* Если есть свободная память, то указатели на начало стеков смещаются с учетом занятой памяти и остатка незанятой памяти, разделенной на количество стеков, нужному стеку присваивается и остаток от деления;
* Далее смещается память, которая уже была занята в стеках, причем с конца, чтобы не произошло «утечки».
* Стекам присваивается новое значение длины.
* Возвращается успех или не успех перераспределения.

# Эксперименты

## Время выполнения

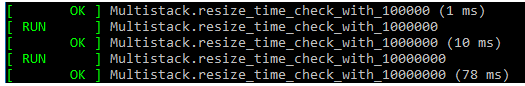
Рассмотрим время выполнения метода перераспределения O(n + n\*m) (где n – количество стеков, m – текущее количество элементов стеке) памяти (рис. 3):

рис. 3 (время выполнения перераспределения памяти)

Составим диаграмму с временем работы:

Тесты проводились на системе:

Процессор Intel Core i5 7200U;

Оперативная память 12 GB.

# Заключение

В заключении можно сказать, что цели и задачи, поставленные перед выполнением данной работы, были достигнуты, а именно: созданы классы «MStack» и «Multistack» с реализованными методами добавления и возврата из стека элементов и перераспределения свободной памяти, а также написаны к ним тесты, и они успешно пройдены.

# Литература

* Учебные материалы к учебному курсу «Методы программирования» - Гергель В.П.