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](#_Toc532933808)

[Постановка целей и задач 4](#_Toc532933809)

[Руководство пользователя 5](#_Toc532933810)

[Руководство программиста 6](#_Toc532933811)

[Описание структуры программы 6](#_Toc532933812)

[Описание структур данных 7](#_Toc532933813)

[Описание алгоритмов 7](#_Toc532933814)

[Эксперименты 10](#_Toc532933815)

[Результаты работы программы 10](#_Toc532933816)

[Время выполнения 11](#_Toc532933817)

[Заключение 13](#_Toc532933818)

[Литература 14](#_Toc532933819)

[Приложение 15](#_Toc532933820)

# Введение

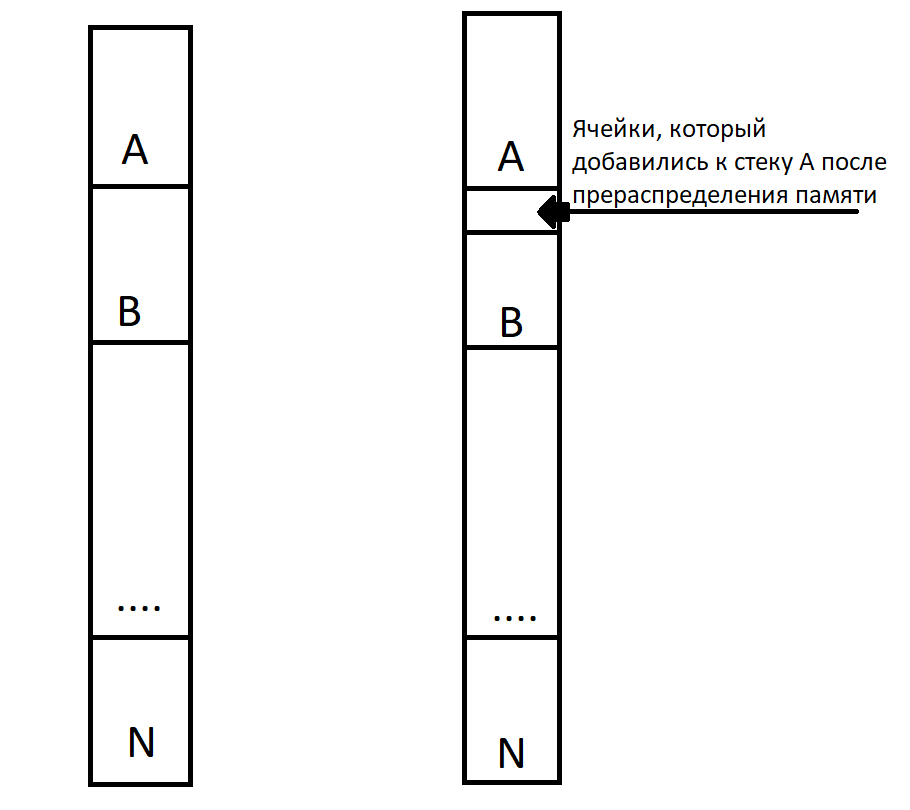
В одной из предыдущих работ мы разобрали, что такое стек, как он работает, его основные функции и применение на практике. Но в некоторых задачах требуется применение системы из нескольких стеков, хранящих схожие типы данных. В таких задачах может использоваться мультистек. Отличительной особенностью 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;
* Stack, который уже был реализован.

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

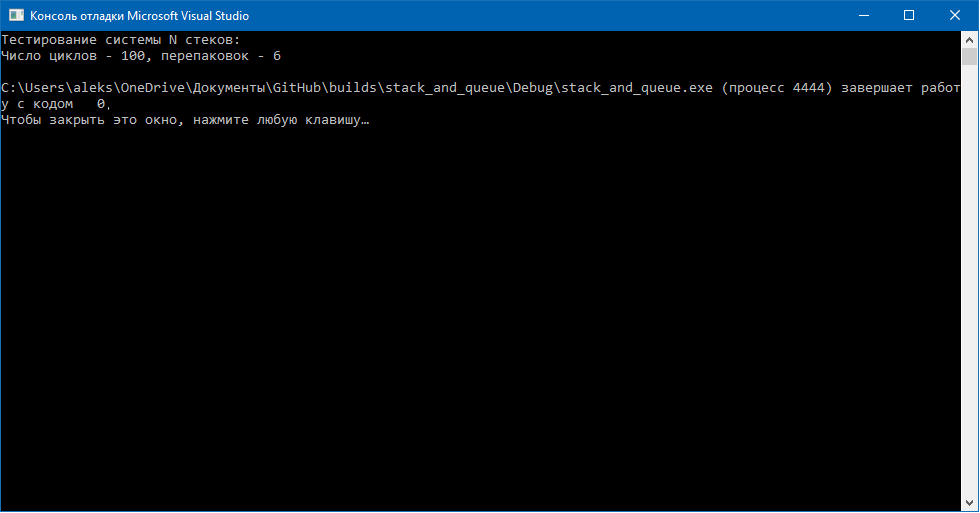
После запуска программы пользователя встречает консольное окно (рис. 2):

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 «Stack».
* Класс «Multistack», который будет использовать класс Stack.

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**

**Stack.h**

**Класс Stack**

**Класс Stack:**

Класс стека, который был написан в одной из предыдущих работ.

**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: |
| 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**

Оператор присваивания вектора:

* Принимает константную ссылку на объект этого же класса, т.е. vector
* Если не идет попытка присвоить vector самому себе (if (m != A.m)), то поля вектора А присваиваются полям текущего вектора.
* Если в поле m присутствовали элементы, то массив очищается, и выделяется память нужного размера. После в цикле поэлементно присваиваются элементы вектора.
* В конце возвращается указатель на текущий объект.

Умножение матриц:

* Принимает ссылку на объект этого же класса, т.е. matrix.
* Создает матрицу размером l (результирующая матрица).
* В цикле от 0 до l (прохождение по строкам) идет цикл (столбцы матрицы) с i – го элемента (если отталкиваться от обычной матрицы, без представления «вектора векторов») до l, в котором еще 1 цикл от i до g + 1, в котором происходит перемножение элементов матрицы и их запись в результирующую матрицу.
* После всех циклов происходит возврат результирующей матрицы.

# Эксперименты

## Время выполнения

Рассмотрим время выполнения некоторых методов, различающихся свой теоретической сложностью (время приведено в наносекундах):

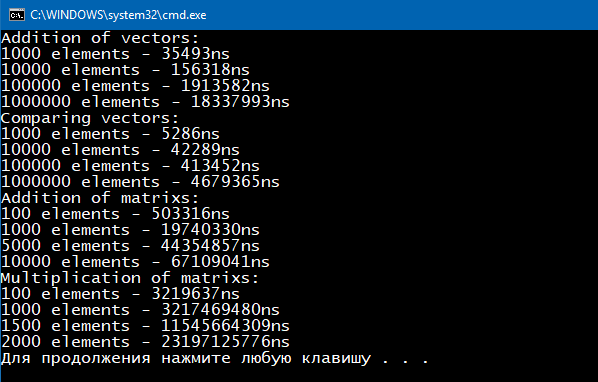
Сложение векторов – теоретическая сложность O(n), сравнение векторов – теоретическая сложность O(n), сложение матриц – теоретическая сложность O(n), умножение матриц – сложность O(n3). На практике получаются такие числа (рис. 2):

рис. 2 (время работы некоторых методов при разном количестве элементов)

Составим диаграмму с временем работы:

С учетом погрешности измерения, можно сказать, что теоретическая сложность алгоритмов совпадает с практическими измерениями.

Тесты проводились на системе:

Процессор Intel Core i5 7200U;

Оперативная память 12 GB.

# Заключение

В заключении можно сказать, что все поставленные цели и задачи были выполнены, а именно: созданы классы «vector» и «matrix» с реализованными методами сложения, вычитания и умножения матриц, а также написаны к ним тесты, и они успешно пройдены.

# Литература

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