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

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc536473711)

[2. Постановка задачи 4](#_Toc536473712)

[3. Руководство пользователя 5](#_Toc536473713)

[4. Руководство программиста 7](#_Toc536473714)

[4.1 Описание структуры программы 7](#_Toc536473715)

[4.2 Описание структур данных 7](#_Toc536473716)

[4.3 Описание алгоритмов 8](#_Toc536473717)

[5. Оценка производительности 10](#_Toc536473718)

[6. Заключение 11](#_Toc536473719)

[7. Литература 12](#_Toc536473720)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных Мультистек на общей памяти.

Мультистек – структура данных, представляющая собой упорядоченный набор N стеков, фиксированного размера. Каждый отдельный стек организован по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Стеки хранятся в памяти друг за другом единым блоком, имеющим определенный размер.

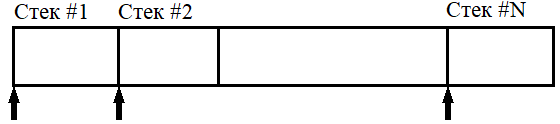


Рисунок 1 Хранение мультистека в памяти (стрелками указаны начала каждого из стеков).

# Постановка задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – мультистек на общей памяти.

Хранить данную структуру в этой лабораторной работе будем с помощью массива. Такая организация мультистека удобна, если элемент информации занимает в памяти фиксированное количество длины, например, какую-то одну условную единицу. При этом отпадает необходимость хранения в элементе стека явного указателя на следующий элемент стека, что экономит память.

Для работы с мультистеком будут реализованы операции:

* добавления элемента в мультистек,
* извлечения элемента из мультистека (с удалением),
* проверка мультистека на полноту/пустоту.

Программное решение будет выглядеть следующим образом:

1. Класс мультистека – TMStack.
2. Вспомогательные классы TStack и TNewStack.
3. Класс для обработки исключений – TException, которые могут возникнуть при выполнении различных операций.
4. Программа, демонстрирующая работу класса TMStack.
5. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

В качестве примера использования стека предлагается следующее. Выполняется создание мультистека с размером выделенной памяти под хранение 12 элементов целого типа, стеков в мультистеке – 3 штуки. Мультистек заполняется элементами.

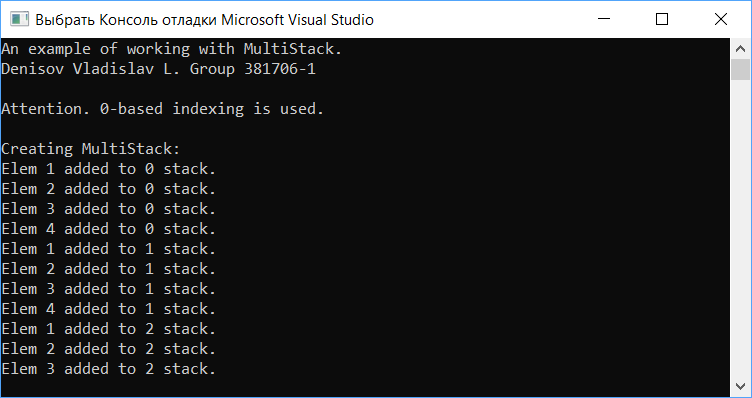


Рисунок 2 Пример работы демонстрационной программы.

И выполняется печать мультистека на консоль.

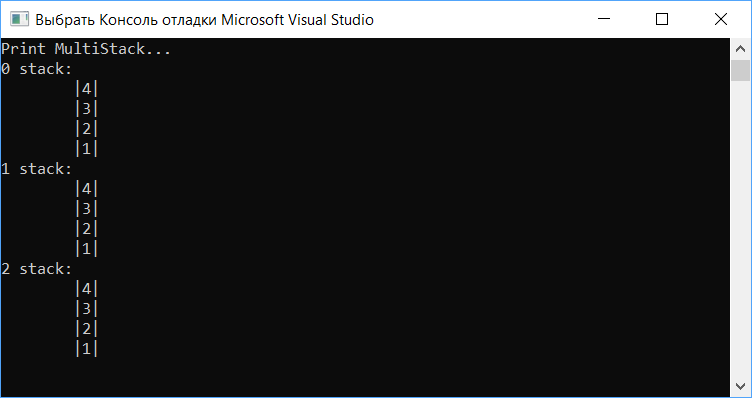


Рисунок 3 Пример работы демонстрационной программы.

Затем производится изъятие элементов из нулевого и первого стеков. Оставшиеся в мультистеке элементы выводятся на консоль.

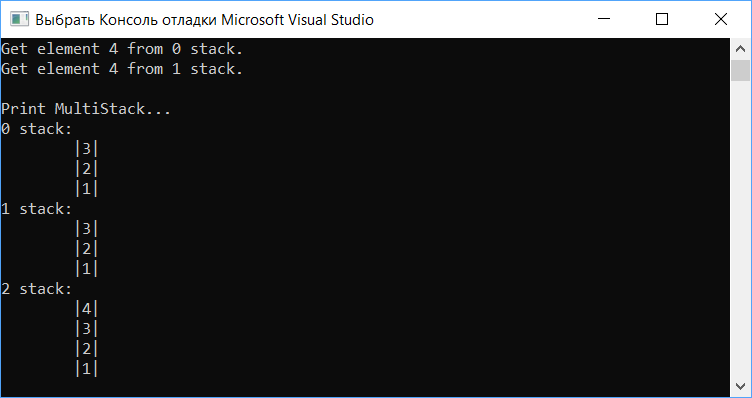


Рисунок 4 Пример работы демонстрационной программы.

Выполняется попытка добавить новый элемент в переполненный третий стек. Для этого требуется перепаковка, которая вызывается автоматически.

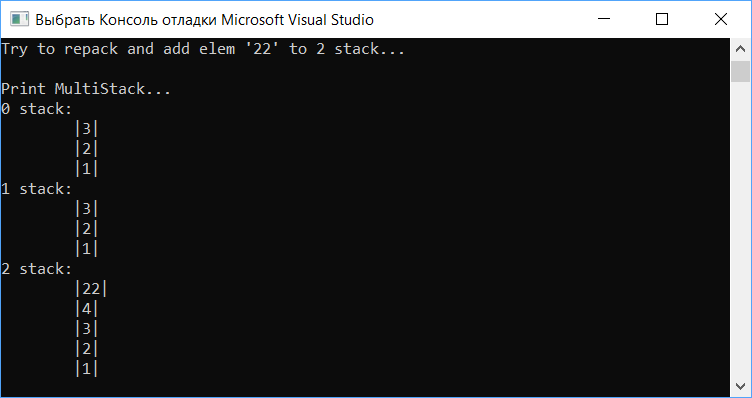


Рисунок 5 Пример работы демонстрационной программы.

Наконец, демонстрируется работа конструктора копирования. Проверяется то, что под новый мультистек выделена новая память, путем изъятия элемента из одного и того же стека.

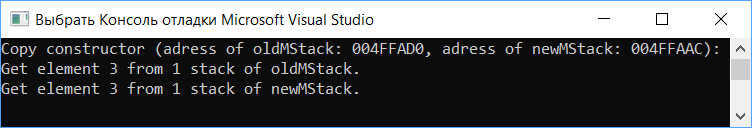


Рисунок 6 Пример работы демонстрационной программы.

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль MultiStack. Содержит пример использования стека. Реализация в файле *main\_MultiStack.cpp.*
* Модуль MultiStackLib – статическая библиотека. Содержит файл MultiStack.h, в котором описан интерфейс и реализация шаблонного класса *TMStack*, а также вспомогательный класс *TNewStack*.
* Модуль MultiStackTest. Содержит 24 теста, описанных в файле *MultiStackTest.cpp* и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## Описание структур данных

#### Класс TException – класс исключений.

Класс содержит 1 **private** поле *std::string msg* – переменная, хранящая сообщение об ошибке в виде строки.

И содержит 2 **public** элемента:

*TException(std::string \_str) : msg(\_msg)* – конструктор с одним параметром.

*void Print()* – метод отображения ошибки на консоль.

#### Класс TNewStack – вспомогательный класс, стек в составе мультистека.

Рассмотрим класс *TNewStack* подробно.

template <class T> class TNewStack :public TStack<T> {…} – класс вспомогательного стека является шаблонным, что позволяет использовать его для хранения данных любого типа. Является наследником со спецификатором открытого доступа к полям и метода класса стек.

**Элементы класса, объявленные со спецификатором public:**

TNewStack(int \_size = 0, T\* \_mas = 0) – конструктор с параметрами.

TNewStack(TNewStack <T> &NS) – конструктор копирования.

int CountFree() – метод, возвращающий число свободных элементов в стеке.

int GetSize() – метод, возвращающий максимальный размер стека.

int GetTop() – метод, возвращающий текущее число элементов в стеке.

void SetMas(int \_size, T\* \_mas) – метод, позволяющий связать стек в составе мультистека с общей памятью.

#### Класс TMStack – класс мультистека.

Рассмотрим класс TMStack подробно.

template <class T> class TMStack {…} – класс мультистека является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

int size – общий размер памяти, выделяемый под мультистек.

T\* mas – указатель на область в памяти, выделенную под хранение мультистека.

int n – число стеков в составе мультистека.

TNewStack<T>\*\* ns – массив указателей на начало каждого стека в мультистеке.

int GetFreeMem() – метод, возвращающий количество свободных ячеек в мультистеке.

void Repack(int k) – метод перепаковки памяти.

**Элементы класса, объявленные со спецификатором public:**

TMStack(int \_size, int \_n) – конструктор с параметрами.

TMStack(TMStack<T> &MS) – конструктор копирования.

void Put(int \_n, T A) – метод добавления элемента в мультистек.

T Get(int \_n) – метод изъятия элемента из мультистека (с удалением).

bool IsFull(int \_n) – метод проверки мультистека на полноту.

bool IsEmpty(int \_n) – метод проверки мультистека на пустоту.

void Print() – метод печати мультистека на консоль.

## Описание алгоритмов

Рассмотрим работу алгоритма перепаковки, работа которого не очевидна на первый взгляд.

Необходимость перепаковки мультистека возникает, когда требуется добавить новый элемент в какой-либо стек, в котором закончилось свободное место, но в любых других стеках свободные ячейки еще есть.

Алгоритм перепаковки мультистека относительно k-го стека:

* + - 1. Вычислим число ячеек *addEvery,* которые будут добавлены в каждый стек.

, где *freeMem* – число свободных ячеек в мультистеке, n – число стеков в составе мультистека.

* + - 1. В случае, если поровну распределить свободные ячейки между стеками не удается, добавляем весь остаток к переполненному стеку .
      2. Получаем текущее число элементов в каждом стеке, увеличиваем его на *addEvery*, сохраняем новые размеры стеков в массив *new\_size*:

.  
В k-ый стек добавляем остаток:

* + - 1. Сохраняем индекс старого начала для каждого из стеков в массив *oldStart,* для удобства работы. Определяем индекс нового начала каждого стека с помощью массива указателей *newStart*, учитывая новый размер каждого из них, записанный в массив *newSize*:
      2. Выполняем сдвиг элементов мультистека на новые позиции с учетом условий:
  1. Если индекс нового начала i-го стека newStart[i] не больше, чем индекс старого начала i-го стека oldStart[i], то копируем элементы по порядку, в котором они хранятся в старом стеке.
  2. Иначе пропускаем стеки до тех пор, пока не выполняется пункт 5.1. Как только это произошло, копируем пропущенные элементы в обратном порядке. Иначе можем потерять данные. Затем продолжаем сдвиг элементов, начиная с той точки, в которой был выполнен пункт 5.1 и произведено “обратное копирование”.

# Оценка производительности

Рассмотрим выполнение операции перепаковки для мультистека большого размера. Мультистек будет содержать 1 миллион элементов, 10 тысяч стеков, т.е в каждом стеке изначально максимум 100 элементов.

Заполним стеки в составе мультистека полностью сначала до индекса k (затем наоборот после индекса k), остальные на 70%. Попробуем добавить элемент в переполненный стек и посмотрим на время, затраченное на операцию (заметим, что добавлять элемент будем так, чтобы пришлось сдвинуть наибольшее число элементов).

|  |  |  |  |
| --- | --- | --- | --- |
| Индекс **k** | 1000 | 5000 | 9000 |
| Сдвиг стеков вправо (заполнение стеков полностью до k) | 56ms | 64ms | 70ms |
| Сдвиг стеков влево (заполнение стеков полностью после k) | 72ms | 64ms | 55ms |

Таблица 1 Сравнение времени затраченного на перепаковку.

Из полученных данных можем сделать вывод, что время сдвига влево и вправо равно. Рост времени, зависящий от числа элементов в мультистеке, происходит достаточно медленно.

Результаты тестов, представленных выше, были получены на компьютере со следующим характеристиками:

Операционная система – Microsoft Windows 10 Home Single Language,

Тип ЦП – DualCore Intel Core i5-7200U, 3100 MHz,

Системная память – 6012 МБ (DDR4 SDRAM).

# Заключение

В результате лабораторной работы была разработана библиотека, реализующая шаблонный класс мультистека. Она позволяет создать объект класса мультистека и выполнить с ним простейшие операции, задача реализации которых была поставлена в начале данной лабораторной работы.

Были разработаны и доведены до успешного выполнения тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework.

Программное решение было продемонстрировано с помощью простейшего набора операций над стеком. Описание примера работы со стеком было представлено в разделе «Руководство пользователя».

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Стек (дата обращения: 11.12.2018)
2. Гергель В.П. Методические материалы по курсу «Методы программирования 2» [Электронный ресурс] //URL <http://www.itmm.unn.ru/files/2018/11/Primer-1.5.-Struktury-hraneniya-neskolkih-stekov-v-obshhej-pamyati.pdf> (дата обращения: 28.12.2018)