

**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**  
**«Система поддержки нескольких стеков»**

**Выполнил:** студент группы 381706-1  
Митягина Дарья Сергеевна

\_\_\_\_\_ Подпись

**Научный руководитель:**  
ассистент каф. МОСТ ИИТММ  
\_\_\_\_\_ Лебедев И.Г

Нижний Новгород  
2018.

## Оглавление

1. Введение .....	2
2. Постановка задачи .....	3
3. Руководство пользователя .....	4
4. Руководство программиста.....	5
4.1. Описание структуры программы. ....	5
4.2. Описание структур данных. ....	5
4.3. Описание алгоритмов.....	6
5. Заключение.....	8
6. Список литературы.....	9

## 1. Введение

В данной лабораторной работе будет реализована система хранения нескольких стеков в общей памяти. *Мультистек* — структура данных, представляющая собой упорядоченный набор  $N$  стеков, фиксированного размера.

Разберемся в способах распределения памяти, ведь именно это и является основной работы.

Распределение памяти до начала процесса вычислений называется статическим. Распределение памяти в ходе выполнения программы называется динамическим распределением памяти.

Процедура динамического перераспределения памяти путем переписи части хранимых значений в другую область памяти называется перепакровкой памяти или просто перепакровкой.

*Пример.*

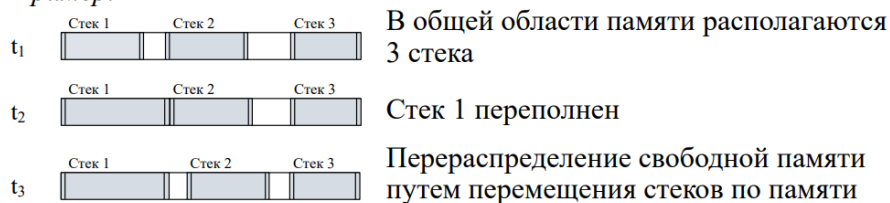


Рис. 1 - пример

## 2. Постановка задачи

В данной работе стояла задача разработать систему поддержки нескольких стеков.

Для этого необходимо было разобраться в следующем:

- Структура памяти и ее свойства
- Начальное распределение памяти
- Оценка имеющейся свободной памяти
- Перераспределение памяти

Кроме того, нужно было

- Протестировать класс `TMStack` с помощью `GT`.
- Реализовать класс `TException` для обработки исключений.

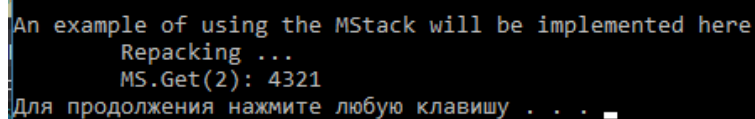
### 3. Руководство пользователя

При запуске программы пользователю представляется простой пример использования списков.

Создается объект класса TMSStack, мультистек. Затем в него кладутся значения и на экран выводятся элементы одного из стеков.

На определенном этапе стек, в который пытаемся добавить элемент, переполняется, тогда выполняется т.н. перепакровка.

Выглядит это следующим образом.



```
An example of using the MStack will be implemented here
Repacking ...
MS.Get(2): 4321
Для продолжения нажмите любую клавишу . . .
```

Рис. 2 – пример использования

## 4. Руководство программиста

### 4.1. Описание структуры программы.

Программа состоит из модулей:

1. stackL- содержит файл Stack.h, в котором реализован класс с.
2. MStackLib – содержит файл MStack.h, в котором реализованы классы TMSStack и TNewStack.
3. MStack – содержит в себе файл реализации примера использования класса TMSStack.
4. mStackTest – содержит в себе файл mStackTest.cpp, в котором находится набор тестов, для проверки работоспособности класса TMSStack.
5. Exception – содержит класс исключений.

### 4.2. Описание структур данных.

#### Класс TNewStack

Этот класс является шаблонным, наследуется от класса TStack.

Рассмотрим public-часть:

1. TNewStack (int \_size = 0, T\* \_mas = 0) - конструктор
2. TNewStack(TNewStack <T> &NS) - конструктор копирования
3. int GetFreeMemory() - получение свободной памяти (кол-во свободных позиций)
4. int GetSize() - получение размера
5. int GetTop() - получение первого элемента
6. void SetMas(int \_size, T\* \_mas) - задать массив размера \_size, содержащий элементы \_mas.
7. void Put(T \_A) - положить элемент \_A
8. T Get() - получить значение элемента
9. ~ TNewStack() – деструктор.

#### Класс TMSStack

Рассмотрим protected-часть:

1. int size - размер
2. T\* mas - массив элементов

3. `int n` - кол-во стеков
4. `TNewStack<T>** newS` - массив указателей на начало каждого стека в мультистеке
5. `int GetFreeMemory()` – возвращает число свободных элементов
6. `void Repack(int k)` – перепакровка (память k-ого стека увеличивается)

Рассмотрим public-часть:

1. `TMStack(int _n, int _size)` - конструктор
2. `TMStack(TMStack<T> &A)` - конструктор копирования
3. `int GetSize()` - возвращает размер
4. `T Get(int _n)` - возвращает значение из n-ого стека
5. `void Set(int _n, T p)` - положить значение в \_n-ый стек
6. `bool IsFull(int _n)` - проверка стека на полноту \_n-ого стека
7. `bool IsEmpty(int _n)` - проверка стека на пустоту \_n-ого стека
8. `~TMStack()` - деструктор

#### 4.3. Описание алгоритмов

В данной части не будут рассматриваться тривиальные методы, внимание уделим лишь некоторым.

##### *1. Перепакровка k-ого стека*

Запоминаем старые, новые индексы начала, а также новый размер каждого из стеков.

Распределяем свободную память поровну между стеками, кроме k-ого, ему выделяется память, размер которой определяется выражением :  $\text{sizeNewOne}[k] += \text{FreeForNow} \% n$ , где `FreeForNow` – число свободных позиций.

Индекс начального элемента каждого стека рассчитывается следующим образом: индекс начала предыдущего стека + его новый размер. Не изменяется лишь начало первого стека.

Затем выполняем проверку на выполнение одного из следующих условий:

а) Если индекс нового начала i-го стека `startNewOne[i]` не больше, чем индекс старого начала i-го стека `startOldOne [i]`, то копируем элементы по порядку, в котором они хранятся в старом стеке.

б) Иначе идем по новым позициям стеков до тех пор, пока не выполняется пункт а). Затем копируем элементы, в котором они хранятся в старом стеке, но в обратном порядке.

## *2. Конструктор инициализации*

Размер и количество стеков определяются пришедшими значениями.

Создается массив (smth), элементы которого, кроме первого, равны целой части от  $\text{size} / n$ ; первый же элемент больше на остаток от деления на  $n$ .

Затем мы задаем значения стеков мультистека через конструктор-инициализатор.

Для первого элемента: `newS[0] = new TNewStack<T>(smth[0], &mas[0]);`

Для всех остальных: `int temp = smth[0] + (i - 1)*smth[i];`

`newS[i] = new TNewStack<T>(smth[i], &mas[temp]);`



## 5. Заключение

В процессе работы над данной лабораторной работой мне удалось разработать систему поддержки нескольких стеков.

Кроме того, было достигнуто более глубокое понимание принципов работы с данной структурой, принципов тестирования программы с помощью GT.

## 6. Список литературы

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015. 2. Статья, посвященная теме списков [<https://prog-cpp.ru/data-ols/>]
2. Лекция, посвященная теме «Методы и алгоритмы перепакетки памяти» [<http://kit.znu.edu.ua/iLec/2sem/swpci/SOURCE/L8.pdf>]