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: линейный двусвязный список на массивах»**

**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](#_Toc536446482)

[2. Постановка задачи 4](#_Toc536446483)

[3. Руководство пользователя 5](#_Toc536446484)

[4. Руководство программиста 6](#_Toc536446485)

[4.1 Описание структуры программы 6](#_Toc536446486)

[4.2 Описание структур данных 6](#_Toc536446487)

[4.3 Описание алгоритмов 7](#_Toc536446488)

[5. Заключение 9](#_Toc536446489)

[6. Литература 10](#_Toc536446490)

# Введение

Лабораторная работа направлена на практическое освоение динамической структуры данных – линейного двусвязного списка на массивах.

Список (англ. list) – это абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Экземпляр списка является компьютерной реализацией математического понятия конечной последовательности. Экземпляры значений, находящихся в списке, называются элементами списка (англ. item, entry либо element); если значение встречается несколько раз, каждое вхождение считается отдельным элементом.

Для создания списка, который необходимо реализовать, используется три массива одинакового размера: содержательный массив и массивы индексов. Эти массивы связанны друг с другом по индексу. По фиксированному *i*, содержательный массив, в *i*-й ячейке, хранит значение элемента списка, а массивы индексов, в *i*-й ячейке, содержат индекс следующего/предыдущего элементов списка в содержательном массиве.

Если список заполнен не полностью, то для пустых ячеек в содержательном массиве, в соответствующих ячейках в массивах индексов лежит значение «-2». Для последнего элемента списка, в соответствующей ячейке массива индексов лежит значение -1.

Причем дополнительно будем хранить индексы первого и последнего элементов списка в содержательном массиве.

# Постановка задачи

В рамках лабораторной работы ставится задача эффективной реализации структуры данных – линейного двусвязного списка на массивах.

Как следует из названия лабораторной, хранить списки будем с использованием массивов.

Для работы со списком будут реализованы операции:

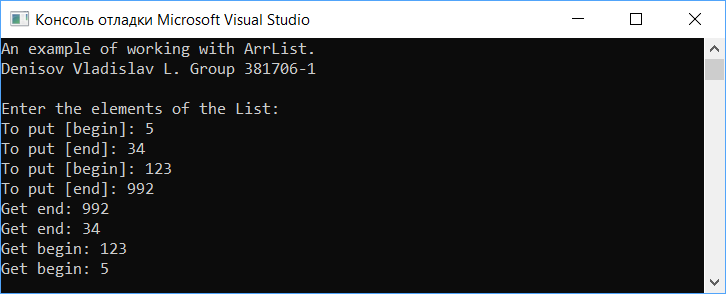
* добавления в начало нового элемента,
* добавления в конец нового элемента,
* извлечения звена из начала списка (с удалением),
* извлечения звена из конца списка (с удалением),
* проверка списка на полноту/пустоту.

Программное решение будет выглядеть следующим образом:

1. Класс списка – TArrList.
2. Класс для обработки исключений – TException, которые могут возникнуть при выполнении различных операций.
3. Программа, демонстрирующая работу класса TArrList.
4. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

В качестве примера использования списка предлагается следующее. Выполняются различные операции добавления элементов в список. Пользователь заполняет их любыми целыми значениями. Затем происходить изъятие элементов из конца и начала списка.

*Рисунок 1 Пример работы демонстрационной программы.*

Снова происходит попытка добавить элементы различными способами. Выполняется копирование списка и отображение соответствующих старому и новому спискам адресов в памяти. Пробуем удалить элементы из каждого экземпляра списка.

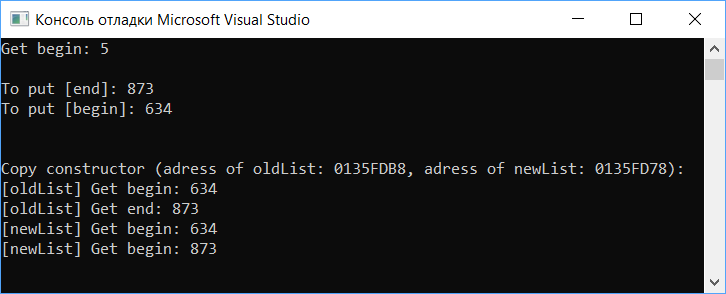


Рисунок 2 Пример работы демонстрационной программы

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль ArrList. Содержит пример использования списка. Реализация в файле *main\_ArrList.cpp.*
* Модуль ArrListLib – статическая библиотека. Содержит файл ArrList.h, в котором описан интерфейс и реализация шаблонного класса *TArrList*.
* Модуль ArrListTest. Содержит 15 тестов, описанных в файле *ArrListTest.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 TArrList – класс списка.

Рассмотрим класс *TArrList* подробно.

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

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

T\* mas – cодержательный массив.

int \*nextInd – массив индексов, указывающих на следующий элемент списка.

int \*prevInd – массив индексов, указывающих на предыдущий элемент списка.

int size – максимальный размер списка.

int count – текущее количество элементов в списке.

int start – индекс первого элемента списка.

int finish – индекс последнего элемента списка.

TQueue <int> freeElem – очередь свободных элементов в содержательном массиве.

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

TArrList(int \_size = 10) – конструктор с параметром.

TArrList(TArrList<T> &a) – конструктор копирования.

virtual ~TArrList() – деструктор.

void PushStart(T elem) – положить элемент в начало списка.

void PushFinish(T elem) – положить элемент в конец списка.

T PullStart() – взять элемент с удалением из начала.

T PullFinish() – взять элемент списка с удалением из конца.

bool IsEmpty() – метод проверки списка на пустоту.

## Описание алгоритмов

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

**Добавление элемента списка в начало.**

Первым делом выполняем проверку на переполнение списка. Если свободных ячеек нет, то бросаем соответствующее исключение. Если все в порядке, то в очереди свободных ячеек *freeElem*, берем индекс *ifree* первой свободной ячейки.

По полученному индексу в массив *mas* записываем значение, которое добавляем в список. Определяем, что следующим за добавленным элементом, будет тот, что сейчас находится в начале, т.е. имеет индекс *start*. Значит, *nextInd[ifree] = start*.

Если, перед добавлением, список не был пуст, то предыдущим для первого элемента списка делаем только что добавленный элемент, то есть *prevInd[start] = i.* Если же список был пуст, то *prevInd[start] = -1*, *finish = ifree*.

Затем индекс *start* переопределяем на только что добавленный элемент *start = ifree*. Увеличиваем количество элементов в списке *count++*.

Примерно аналогичные действия выполняются при добавлении нового элемента в конец списка.

**Удаление элемента списка из начала.**

Первым делом выполняем проверку на пустоту списка. Если список пуст, то бросаем исключение. Иначе выполняем следующие действия. Рассматривается два случае: если в списке один элемент, больше одного элемента.

В первом случае запоминаем индекс *tmp = start* в содержательном массиве, из которого происходит изъятие. В очередь свободных ячеек добавляем *start*. Т.к. при удалении одного оставшегося элемента список станет пустым, то *nextInd[start] = prevInd[start] = -2*, а *start = finish = -1*. Возвращаем значение *mas[tmp]*.

В случае, когда элементов больше одного запоминаем значение удаляемого элемента *elem = mas[start].* Ищем индекс элемента, который теперь станет первым во всем списке *newstart = nextInd[start]*. Добавляем в очередь свободных *start*. Для ячейки с индексом start указываем, что она отсутствует в текущем списке: *nextInd[start] = prevInd[start] = -2*. Если список не опустел, то *prevInd[newstart] = -1.* Переобозначаем новое начало списка, как *start = newstart*, уменьшаем число элементов *count--*, возвращаем нужный нам элемент *elem*.

Для удаления из конца выполняем аналогичные действия, однако удаление происходит для общего случая, независимо от текущего числа элементов в списке.

# Заключение

В результате лабораторной работы была разработана библиотека, реализующая шаблонный класс двусвязного списка с использованием массивов. Она позволяет создать объект класса списка и выполнить с ним такие операции, как:

* добавления в начало нового элемента списка,
* добавления в конец нового элемента списка,
* извлечения элемента из начала списка (с удалением),
* извлечения элемента из конца списка (с удалением),
* проверка списка на полноту/пустоту.

Были разработаны и доведены до успешного выполнения тесты, созданные 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 списком было представлено в разделе «Руководство пользователя».

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Список\_(информатика) (дата обращения: 21.12.2018)