Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по ЛАБОРАТОРНОЙ РАБОТЕ №6**

**«Односвязный список с использованием массивов»**

**Выполнила:** студентка группы 381903-3 Семибабнова А.В.

**Проверил:**

ассистент каф. МОСТ ИИТММ Лебедев Илья Геннадьевич

Нижний Новгород

2021

Содержание

[1. Введение 3](#_Toc533097873)

[2. Постановка задачи 4](#_Toc533097874)

[3. Руководство пользователя 5](#_Toc533097875)

[4. Руководство программиста 6](#_Toc533097876)

[4.1 Описание структур данных 6](#_Toc533097877)

[4.2 Описание структур данных 6](#_Toc533097878)

[4.3 Описание алгоритмов 7](#_Toc533097879)

[5. Заключение 8](#_Toc533097880)

[6. Литература 9](#_Toc533097881)

# Введение

**Односвязный список на массивах** – это структура данных, представляющая собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза.

Для создания такого списка используется два массива одинакового размера: содержательный массив и массив индексов. Эти массивы связанны друг с другом по индексу. По фиксированному *i*, содержательный массив, в *i*-й ячейке, хранит значение элемента списка, а массив индексов, в *i*-й ячейке, содержит индекс следующего элемента списка в содержательном массиве.

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

# Постановка задачи

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

Для работы со списком необходимо реализовать операции:

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

Программное решение будет выглядеть следующим образом:

1. Класс списка – TArrList.
2. Класс для обработки исключений – Exception, 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

Рассмотрим пример использования класса TArrList.

При запуске программы с пользователя спрашивается размер списка . Затем в начало списка кладется Затем спрашивается количество элементов , которое необходимо положить в конец списка. Полученный список выводится на экран. Он будет выглядеть следующим образом: .

Затем выполняется изъятие элементов из конца и начала списка. Изъятые элементы выводятся на экран. В конце на консоль выводится список, полученный после изъятия из него элементов. На этом работа программы прекращается.

# Руководство программиста

## Описание структур данных

Программа состоит из следующих модулей:

* Модуль ArrList. Содержит пример использования списка на массивах. 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 23 теста, 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 TArrList – класс списка на массивах.

Класс TArrList является шаблонным. В классе всего 8 полей, объявленных со спецификатором доступа private:

* *T\* mas* – указатель на область памяти для хранения элементов списка.
* *int \*nextInd* - указатель на область памяти для хранения индексов, указывающих на следующий элемент списка.
* *int \*predInd* - указатель на область памяти для хранения 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 end* - индекс последнего элемента списка.
* *TQueue <int> freeElem* - Очередь свободный ячеек в массиве mas.

**Конструкторы и методы класса, объявленные со спецификатором public:**

* *TArrList(int \_size = 10)* - конструктор по умолчанию.
* *TArrList(TArrList<T> &A)* - конструктор копирования.
* *~ TArrList()* – деструктор.
* *void PutStart(T elem)* – метод, позволяющий добавить элемент в начало списка.
* *void PutEnd(T elem)* – метод, позволяющий добавить элемент в конец списка.
* *void Put(int n, T elem)* - добавить промежуточный элемент на позицию n в списке
* *T Get(int n)* - извлечь из списка элемент на позиции n.
* *T GetStart()* – метод, позволяющий получить с удалением элемент из начало списка.
* *T GetEnd()* – метод, позволяющий получить с удалением элемент из конца списка.
* *bool IsFull()* - проверка списка на полноту.
* *bool IsEmpty()* – проверка списка на пустоту.
* *void Print()* – вывод элементов списка на консоль.

## Описание алгоритмов

**Добавление звена списка в начало и в конец.**

Сначала проверяем не заполнен ли список. Если он заполнен, то бросаем исключение. Если нет, то в очереди свободных позиций *freeElem*, берем первую свободную ячейку *i*. По полученному индексу в массив *mas* записываем значение, которое хотим положить в список. Определяем, что следующим для этого элемента, будет элемент с текущим индексом start, то есть *nextInd[i] = start*. Если, перед добавлением, список не был пуст, то предыдущим для первого элемента списка делаем только что добавленный элемент, то есть: *predInd[start] = i.* Если же список был пуст, то определяем, что добавленный элемент является и последним элементом в списке. Затем индекс *start* переопределяем на только что добавленный элемент: *start = i*. Увеличиваем количество элементов в списке *count++*.

Для добавления элемента в конец списка, рассуждения аналогичны, с поправкой на то, что добавляем в конец списка.

**Удаление звена списка из начала и из конца.**

Сначала проверяем не пуст ли список. Если он пуст, то бросаем исключение. Если нет, то в переменную *elem* записываем элемент в начале списка *mas[start].* Затем переопределяем индекс первого элемента списка, он будет равен индексу элемента следующим за первым: *newstart = nextInd[start].* Затем в очередь свободных позиций *freeElem*, добавляем освободившуюся после изъятия первого элемента свободную ячейку *start*. Идентифицируем пустыми соответствующие ячейки массивов *nextInd* и *predInd*: *nextInd[start] = predInd[start]* = -2. Если после извлечения первого элемента, список не пуст, то предыдущего элемента для нового первого элемента не существует: *predInd[newstart] = -1*.Затем индекс *start* переопределяем на *newstart*: *start = newstart*. Уменьшаем количество элементов в списке *count--*.

Для извлечения элемента из конца списка, рассуждения аналогичны.

# Заключение

В ходе выполнения лабораторной был произведен анализ задачи - установлено понятие списка на массивах. Была разработана библиотека, реализующая шаблонный класс списка. Она позволяет при работе со списком выполнять базовые операции извлечения/добавление элементов списка.

# Литература

* Книги

1. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.
2. Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.

* Ссылки в Internet

1. Википедия: свободная электронная энциклопедия: на русском языке: [https://ru.wikipedia.org/wiki/Список\_(информатика)]
2. Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [ http://www.itmm.unn.ru/files/2018/11/Primer-1.7.-Razrabotka-obshhego-predstavleniya-linejnogo-spiska.pdf], 2015.