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 ЛАБОРАТОРНОЙ РАБОТЕ**

**«Структура хранения данных: Список на массиве»**

**Выполнил:** студент группы 381706-2

Мышкин Андрей Александрович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

ассистент кафедры МОСТ ИИТММ,

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

**Содержание**

1. Введение………………………………………………………………………...3

2. Постановка задачи……………………………………………………...………5

3. Руководство пользователя…………………………………………...………...6

4. Руководство программиста…………………………………………...………..7

4.1. Описание структур данных…..……………………………………….7

4.2. Описание алгоритмов………………………………………………….8

4.3. Описание структуры программы….……………………………..…...8

6. Заключение……………………………………………………………………..9

7. Литература……………….…………………………………………...……….10

**Введение**

Списком называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения. Список, отражающий отношения соседства между элементами, называется линейным.

С реализациями линейных списков в императивных языках программирования могут выполняться следующие операции:

* получение доступа к некоторому элементу списка для проверки и/или изменения содержимого его полей;
* вставка нового элемента сразу перед или после произвольного элемента;
* удаление произвольного элемента;
* объединение в одном списке двух (или более) линейных списков;
* разбиение линейного списка на два (или более) списка;
* создание копии линейного списка;
* определение количества элементов в списке;
* сортировка элементов списка;
* поиск элементов с заданным значением.

В одной программе крайне редко возникает необходимость использовать все девять типов операций. При этом достаточно трудно создать единую реализацию линейных списков, при которой эффективно выполнялись бы все эти операции. Поэтому линейные списки могут быть реализованы по-разному в зависимости от класса операций, которые наиболее часто должны с ними выполняться в данной программе, или наиболее критичных к времени выполнения.

Основные способы хранения линейных списков в памяти компьютера можно разделить на способы последовательного и связанного хранения. При последовательном хранении элементы списка располагаются в памяти в последовательных ячейках, при этом один элемент следует сразу же за другим. Связанное хранение представляет собой более гибкую схему, при которой каждый элемент списка содержит связь со следующим элементом, а их взаимное расположение в памяти может быть произвольным. Каждый способ имеет свои преимущества и недостатки. При выборе способа хранения в конкретной программе следует учитывать, какие операции и с какой интенсивностью будут выполняться над линейными списками, стоимость их выполнения и объем необходимой памяти для хранения списка.

Последовательное хранение линейных списков рассмотрено во множестве источников. Также широко рассмотрено связанное хранение линейных списков, но в подавляющем большинстве случаев такая реализация основана на ссылочной реализации. В то же время существует изящная, но малоизвестная широкому кругу молодых программистов реализация связного хранения линейных списков на базе массивов.

**Постановка задачи**

Основная задача данной работы – создание и реализация структуры данных такой, как список на массиве. В классе списка на массиве должны быть реализованы и описаны все методы, используемые в нем.

**Руководство пользователя**

При запуске тестирования списка будет поэтапное заполнение списка, размером в 20 значений. В список будут помещены случайно сгенерированные числа от 0 до 50. Для упрощения тестирования проходит с целочисленными значениями в небольшом интервале. После того как список будет полон, начнется извлечение всех элементов из данного списка. Извлеченные значения также будут поэтапно выписываться на консоль для большего понимания работы такой структуры.

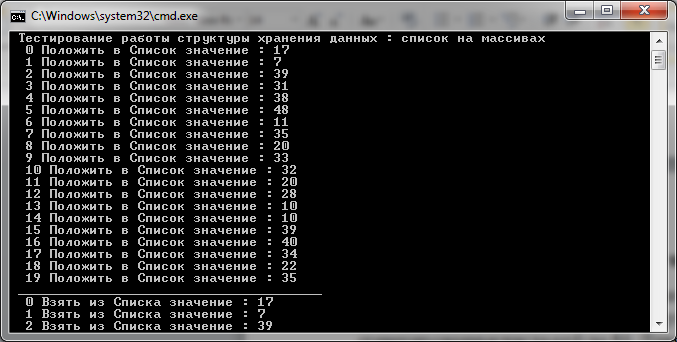


Рисунок 1. Тестирование заполнения списка

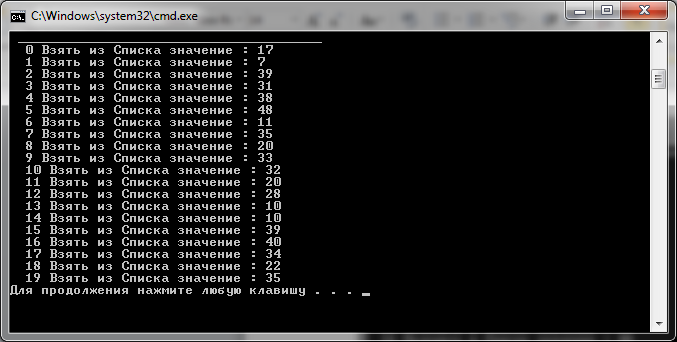


Рисунок 2. Тестирование освобождения списка

**Руководство программиста**

**Описание структур данных.**

**Класс TArrList**

Поля

ValType \* mem – выделенная память под элементы списка

int\* indexNext – массив индексов следующих элементов списка

int size – максимально возможный размер списка

int firstInd – индекс первого элемента в списке

int firstFreeInd – индекс первого пустого элемента в массиве

int count – число элементов в списке в данный момент

Методы

int NextFreeInd() – метод класса, который находит свободное место в массиве

TArrList(int n = 10) – конструктор класса с параметром

TArrList(TArrList& al) – конструктор копирования класса

~TArrList() – деструктор класса

void InsFirst(ValType val) – метод класса, отвечающий за постановку данного элемента первым элементов в списке

void InsLast(ValType val) – метод класса, отвечающий за постановку элемента на посседнее место в списке

ValType GetFirst() – метод класса, при вызове которого выдается значение первого элемента в списке

ValType GetDatValue(int num) – метод класса, при вызове которого выдается значение элемента из списка по указанному номеру. Номер определяется переменной num

ValType GetLast() – метод класса, при вызове которого выдается последнее значение элемента в списке

void DelFirst() – метод класса, который производит удаление элемента из начала списка

void DelLast() – метод класса, производящий удаление элемента из конца списка

void Del(int num) – метод класса, производящий удаление произвольного элемента в списке по указанному в вызываемом методе номеру. Номер определяется значение переменной num

bool IsFull() – метод класса, проверяющий список на его полноту

bool IsEmpty() – метод класса, проверяющий список на его пустоту

int GetListLength() – метод класса, который при вызове возращает максильмально возможный размер списка

**Описание алгоритмов.**

Добавление элемента в начало списка.

Если список уже полон, то выходим с исключением. Если список ещё не полностью заполнен, на месте первого пустого элемента массива выделяем память под новое значение. В массив следующих индексов на месте пустого элемента присваиваем индекс начала списка, далее меняем значение индекса начала списка на новый индекс, тем самым смещая весь список. Затем находим новый пустой элемент в массиве и присваиваем его индекс к индексу первого пустого элемента в массиве. В конце увеличиваем настоящий размер списка на 1.

**Описание структуры программы.**

Arrlist – модуль для тестирования работы структуры данных типа список на массиве. Главным, файлом которого является main.cpp

Arrlistlib - модуль для хранения реализации класса список на массиве. В нем содержится заголовочный файл – TArrList.h

Arrlisttest - модуль, содержащий тесты данного класса для прохождения их с помощью использования Google C++ Testing Framework. В файле arrlist\_tests.cpp содержатся написанные тесты, а в test\_main.cpp код, запускающий тестирование

**Заключение**

В ходе проведение данной лабораторной работы была создана и протестирована такая структура хранения данных как список на массиве. В классе список на массиве были реализованы все необходимые методы для комфортной работы с данной структурой. Усвоены все тонкости, с которыми пришлось столкнуться при выполнении данной лабораторной работы.

Также были освоены инструменты разработки программного обеспечения, как система контроля версий Git и фрэймворк для разработки автоматических тестов Google Test.

**Литература**

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015
2. Реализация связных списков на базе массивов - <https://rsdn.org/article/alg/list.xml>