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»**

**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**

Очень часто, при разработке приложений, оперирующих с большим количеством входных данных, возникает вопрос об их хранении во время выполнения программы. Приводить все из них не имеет смысла, остановлюсь лишь на массивах. Несомненно, данный тип решает вопрос хранения данных, однако, очевидно, что он не лишен недостатков. Главным из них, несомненно, является его фиксированный размер. Это свойство не поддается изменению даже у динамически созданных массивов, что довольно часто заставляет программистов, использующих исключительно их, выделять память "с запасом". Ну а во-первых, даже "запас" ограничен, и никто не может дать гарантии, что и его будет достаточно, а во-вторых, наоборот, "запаса" может хватить настолько, что немалая часть отведенной программе памяти будет занята понапрасну.

Данную проблему решает другой тип хранения данных - связанный список динамических переменных, или проще - динамический список. Компоненты добавляются и удаляются во время выполнения программы, и их количество зависит исключительно от размера доступной памяти. Однако за это преимущество приходится расплачиваться недостатком - если в случае с массивом, мы в любой момент получаем доступ к любому компоненту, то в случае со списком, в один момент времени нам доступны максимум 3 компонента (это зависит от способа представления списка в программе). В большинстве случаев, это очень даже приемлемая цена .

Подведем итоги:

Связный список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Достоинства:

* эффективное (за константное время) добавление и удаление элементов
* размер ограничен только объёмом памяти [компьютера](https://ru.wikipedia.org/wiki/%D0%AD%D0%92%D0%9C) и разрядностью указателей
* динамическое добавление и удаление элементов

Недостатки:

* сложность прямого доступа к элементу, а именно определения физического адреса по его индексу (порядковому номеру) в списке
* на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны)
* некоторые операции со списками медленнее, чем с массивами, так как к произвольному элементу списка можно обратиться, только пройдя все предшествующие ему элементы
* соседние элементы списка могут быть распределены в памяти не локально, что снизит эффективность кэширования данных в процессоре
* над связными списками, по сравнению с массивами, гораздо труднее (хоть и возможно) производить параллельные векторные операции, такие, как вычисление суммы: накладные расходы на перебор элементов снижают эффективность распараллеливания

https://upload.wikimedia.org/wikipedia/commons/thumb/3/37/Singly_linked_list.png/220px-Singly_linked_list.png

Рисунок 1. Разновидность связного списка — односвязный список, содержащий 3 элемента.

**Постановка задачи**

Основная задача данной работы – создание и реализация такой структуры данных, как список. Включая обязательную реализацию следующих операций:

Добавление узлов в начало и конец связного списка

Добавление узла в промежуточные узлы списка

Извлечение узла из промежуточного узла списка

Извлечение узлов из начала и конца списка

Проверка полноты связного списка

Проверка пустоты связного списка

**Руководство пользователя**

При запуске тестирования списка будет поэтапное заполнение списка, состоящего из 15 узлов. 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.

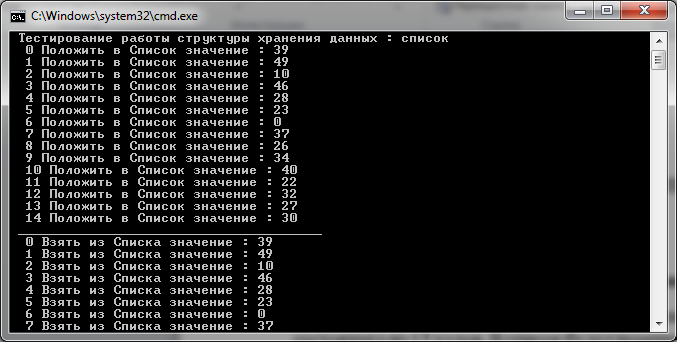


Рисунок 2. Тестирование заполнения списка

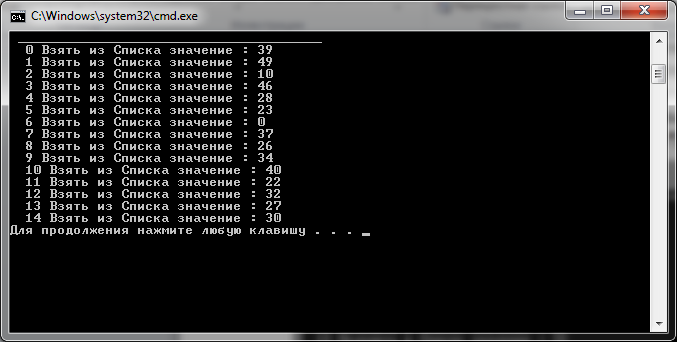


Рисунок 3. Тестирование освобождения списка

**Руководство программиста**

**Описание структур данных.**

**Класс TDatLink** – класс, описывающий и создающий элементы списка, то есть звенья.

Поля

ValType val – значение, вложенное в звено

TDatLink\* pNext – указатель на следующее звено

**Класс TDatList.**

Поля

TDatLink<ValType>\* pFirst – указатель на первое звено в списке

TDatLink<ValType>\* pLast – указатель на последнее звено в списке

int listCount – количество звеньев в списке

Методы

TDatList() – конструктор класса

TDatList(const TDatList& dl) – конструктор копирования класса

~TDatList() – деструктор класса

void DelLink(TDatLink<ValType>\* pLink) – метод удаления звена из списка

ValType& GetDatValue(int num) const – метод класса, выдающий значение из списка по нужному номеру

int IsEmpty() const – метод класса, проверяющий список на пустоту

int GetListLength() const – метод класса, выдающий количество звеньев в списке в настоящее время

void InsFirst(const ValType &Val) – метод класса, с помощью которого вставляют значение в начало списка

void InsLast(const ValType &Val) - метод класса, с помощью которого вставляют значение в конец списка

void InsTo(int num, const ValType& Val) – метод класса, который вставляет элемент в указанную позицию списка, определяется значением переменной num

void DelFirst() – метод класса, производящий удаление первого звена в списке

void Del(int num) – метод класса, производящий удаление запрашиваемого звена в списке, определяется значением num

void DelList() – метод класса, который при вызове производит удаление всего списка

**Описание алгоритмов.**

Добавление звена в начало списка.

При добавлении звена в список, в нем создается указатель, в который мы вкладываем данное значение и указатель на текущее первое звено, тем самым выделяя память под новое звено, используя конструктор класса TDatLink. После чего меняем значение указателя на первое звено на новое. В случае если у списка было 0 звеньев, значение нового указателя также присваиваем значению указателя последнего звена. Затем увеличиваем значение количества звеньев на 1.

**Описание структуры программы.**

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

DatListLib – модуль для хранения реализации класса список. В нем содержится заголовочный файл – TDatList.h

DatLsitTest – модуль, содержащий тесты данного класса для прохождения их с помощью использования Google C++ Testing Framework. В файле datlist\_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. В классе список были реализованы такие операции как: добавление узлов в начало и конец связного списка, добавление узла в промежуточные узлы списка, извлечение узла из промежуточного узла списка, извлечение узлов из начала и конца списка, проверка полноты связного списка, проверка пустоты связного списка.

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. <https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA> – Википедия. Связный список
3. <http://www.codenet.ru/progr/cpp/dlist.php> – Динамический список и его реализация