Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

Высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий математики механики

Выполнил:

студент ИИТММ гр. 381706-2

Антипин А.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

Список на указателях

Отчет по лабораторной работе

Нижний Новгород

2018 г.

**Содержание**

[Введение 3](#_Toc820706)

[Постановка целей и задач 4](#_Toc820707)

[Руководство пользователя 5](#_Toc820708)

[Руководство программиста 6](#_Toc820709)

[Описание структуры программы 6](#_Toc820710)

[Описание структур данных 8](#_Toc820711)

[Описание алгоритмов 9](#_Toc820712)

[Эксперименты 11](#_Toc820713)

[Время выполнения 11](#_Toc820714)

[Заключение 13](#_Toc820715)

[Литература 14](#_Toc820716)

# Введение

Стек и очередь являются очень удобными и незаменимыми структурами хранения данных в некоторых задачах, но они имеют один существенный недостаток – ограниченность выделяемой памяти под элементы этих структур. А что если заранее неизвестно сколько элементов будет входить в структуру? Тогда целесообразно создать новую структуру хранения данных типа «Список» или List на английском. Список – это структура хранения данных, в которую можно добавлять и из которой можно удалять элементы сколько угодно раз. Единственное ограничение – это объем физической памяти на компьютере.

Разберем устройство этой структуры данных. Список состоит из элементов списка, которые состоят из самого значения, которое надо хранить, и указателя на следующий элемент списка. По указателю на следующий элемент можно перейти на него и выполнить необходимые действия. Тем самым происходит работа со списком. Единственным недостатком такого способа работы является то, что постоянно приходится использовать циклы, что в случае большого списка замедляет время работы программы.

# Постановка целей и задач

Целью лабораторной работы является создание структуры хранения типа «Список» и методов работы с ним, таких как:

* Добавление элементов в начало, конец и произвольное место списка;
* Удаление элементов из начала и произвольного места списка;
* Получение доступа к элементу списка.

Для реализации алгоритмов будет использоваться 2 шаблонных класса:

* TDatLink;
* List.

Для проверки правильности работы этих классов будут написаны тесты с использованием фреймворка Google Test, а также тестовый образец программы, которая будет использует класс список.

# Руководство пользователя

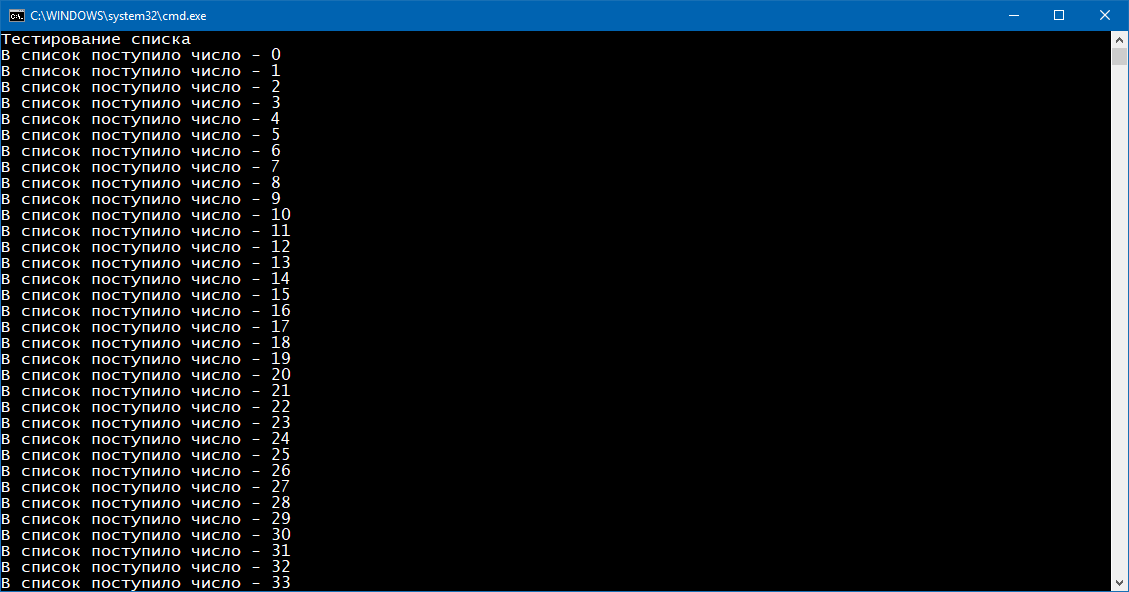
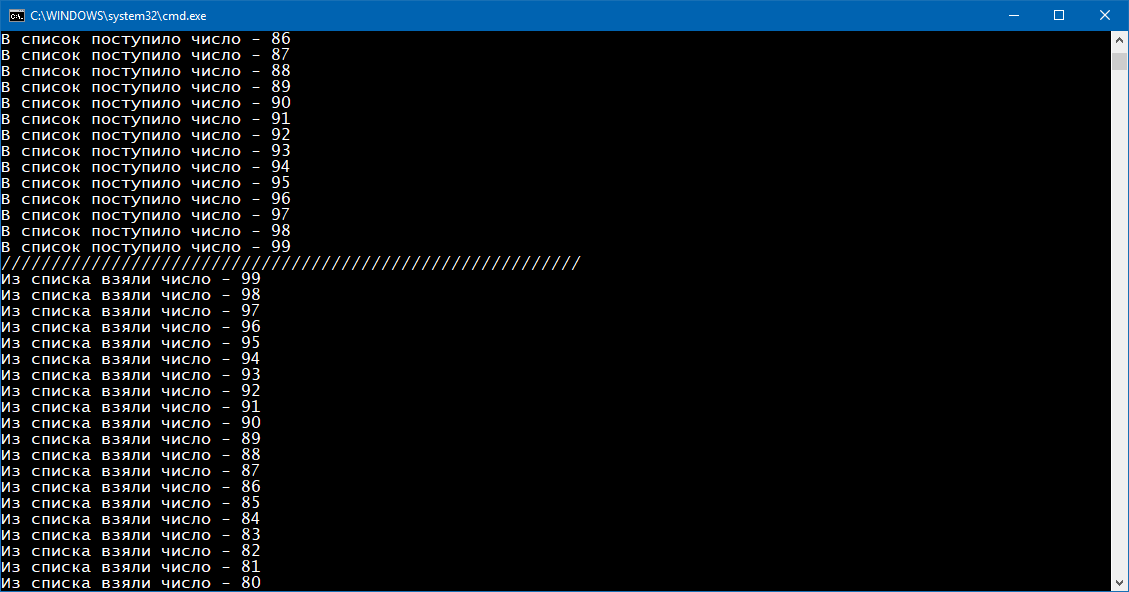
После запуска программы пользователя встречает консольное окно (рис. 1):

рис. 2 (возвращение чисел от 99 до 0)

рис. 1 (вывод программы тестирования списка для пользователя)

в которой сначала в список кладется числа от 0 до 99, а затем значения этих чисел снова выводятся на экран в обратном порядке без удаления (рис. 2).

# Руководство программиста

## Описание структуры программы

Список будет реализован с помощью набора элементов списка:

То есть для реализации алгоритмов будет использовано 2 класса:

* Класс «Звено списка» (TDatLink).
* Класс «Список» (List), который будет использовать класс TDatLink.

А также проект использующий фреймворк Google Test, для проверки правильности работы этих классов и тесовый проект, который будет показываться пользователю.

Древо классов

**Класс gtest**

**test\_main.cpp**

**list\_item\_test.cpp**

**list\_test.cpp**

**Проект для пользователя**

**main.cpp**

**List.h**

**Класс List**

**List\_Item.h**

**Класс TDatLink**

**Класс TDatLink:**

Класс звено списка задает работу с классом List посредством возможности создания элемента этого списка и работы с ним. Также можно выполнять операции, связанные с элементом списка, такие как получение следующего звена, задать указатель на следующий элемент др.

**Класс List:**

Класс List содержит реализацию работы со списками. В нем реализованы такие методы, как положить в список, удалить из списка, посмотреть значение и др.

**Класс gtest:**

Класс gtest реализует тестирование классов TDatLink и List, по средствам фреймворка Google Test. Тесты пишутся для каждого метода классов, каждого ветвления этих методов и для всех возможных исключений этих методов. В файле «list\_item\_test.cpp» реализованы тесты для класса TDatLink, а в файле «list\_test.cpp» соответственно для класса List.

**Проект List:**

В данном проекте реализован примет использования списка, в который поочередно кладутся и забираются элементы.

## Описание структур данных

**Класс TDatLink:**

template <class T> - шаблон класса T

T Mem; - элемент, который нужно сохранить;

TDatLink<T>\* m; - указатель на следующий элемент списка.

**Класс List:**

template< class T > - шаблон класса А

TDatLink<T>\* FirstItem; - указатель на первый элемент списка;

TDatLink<T>\* LastItem; - указатель на текущий последний элемент списка;

int ListLen; - текущее количество элементов в списке.

**Описание методов:**

|  |  |
| --- | --- |
| Метод: | Описание: |
| TDatLink<T>::TDatLink() | Конструктор по умолчанию для класса TDatLink. |
| TDatLink<T>::TDatLink(const T \_Mem, TDatLink<T>\* \_m) | Конструктор с параметром для класса TDatLink, который принимает элемент типа T и указатель на следующий элемент списка. |
| TDatLink<T>::TDatLink(const TDatLink<T>& A) | Конструктор копирования класса TDatLink. |
| void TDatLink<T>::SetMem(T \_Mem) | Метод, позволяющий изменить элемент, хранящийся в звене списка. |
| T TDatLink<T>::GetMem() | Метод, который возвращает элемент, хранящийся в звене списка. |
| void TDatLink<T>::Set\_m(TDatLink<T>\* \_m) | Метод, позволяющий изменить указатель на следующий элемент списка. |
| TDatLink<T>\* TDatLink<T>::GetNextLink() | Метод, который возвращает указатель на следующий элемент списка. |
| List<T>::List() | Конструктор по умолчанию для класса List. |
| List<T>::~List() | Деструктор для класса List. |
| TDatLink<T>\* List<T>::GetLink(T \_Mem, TDatLink<T>\* \_m) | Метод, позволяющий создать звено списка. |
| void List<T>::DelLink(TDatLink<T>\* Link) | Метод, позволяющий удалить звено списка. |
| T List<T>::GetValue(const int Pos) const | Получение значения элемента списка по его номеру. |
| bool List<T>::IsEmpty() const | Проверка списка на пустоту. |
| int List<T>::GetListLen() const | Возвращение текущей длины списка |
| void List<T>::InstFirst(T A) | Устанавливает элемент типа T в начало списка. |
| void List<T>::InstLast(T A) | Устанавливает элемент типа Т в конец списка. |
| void List<T>::InstCustom(const int Pos, T A) | Устанавливает элемент типа Т в произвольное место списка по номеру Pos, Pos не должен выходить за текущие границы списка. |
| void List<T>::DelFirst() | Удаление первого элемента списка. |
| void List<T>::DelCustom(const int Pos) | Удаление произвольного элемента в списке по его позиции в нем. |
| void List<T>::DelList() | Удаление всего списка целиком. |

## Описание алгоритмов

**Подробное описание некоторых методов**

Установка элемента в произвольное место списка:

* Проверка на выход индекса за текущие границы списка;
* Если индекс не равен текущим границам списка, т.е. 0 и последнему, то:
  + Создается указатель на первый элемент списка, указатель на новый элемент списка и звено списка с помещенным в него элемент типа Т (не указатель);
  + В цикле от 0 до необходимой позиции пропускаются элементы списка, а когда доходим до нужной позиции, то меняем указатели в соседних элементах;
  + Увеличение количества элементов в списке на 1;
* Если индекс равен 0, то вызывается метод вставки в начало списка;
* Если индекс равен текущей длине списка, то вызывается метод вставки в конец списка.

Удаление произвольного элемента:

* Если индекс удаляемого элемента выходит за границу текущей длины списка, то вызывается исключение;
* Если список пуст, то также вызывается исключение;
* Если индекс удаляемого элемента равен 0, то вызывается метод удаления начального элемента списка;
* Если индекс удаляемого элемента равен текущей длине списка, то:
  + Создается указатель на первый элемент списка и на пустой элемент списка;
  + В цикле по указателям доходим до предпоследнего элемента;
  + Присваиваем пустому указателю на элемент списка последний элемент списка и удаляем его;
  + Предпоследнему элементу списка ставим указатель на следующий элемент равный NULL и уменьшаем количество элементов на 1;
* Если индекс находится в диапазоне текущей длины списка, то делаем те же действия, что и в предыдущем пункте, только в пустой указатель записываем не последний элемент, а элемент под нужным индексом и предыдущему элементу присваиваем указатель на элемент больший текущего.

# Эксперименты

## Время выполнения

Рассмотрим время выполнения некоторых методов теоретической сложностью O(n) (время приведено в наносекундах): вставка в список в произвольную позицию (рис. 2) и удаление произвольного элемента списка (для максимально наглядного представления вставка и удаление будут производиться в конце списка)

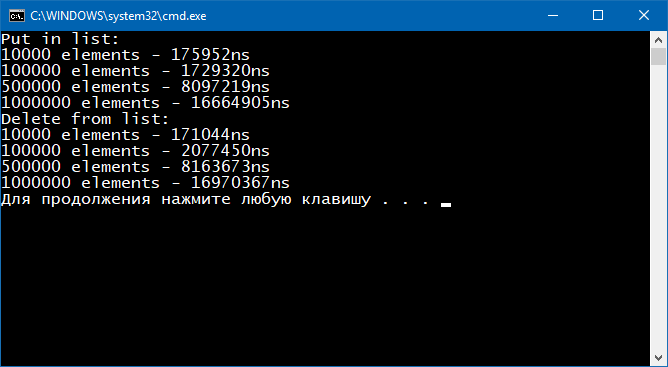
Составим диаграмму с временем работы:

рис. 2 (время работы вставки и удаления элемента из списка)

С учетом погрешности измерения, можно сказать, что теоретическая сложность алгоритмов совпадает с практическими измерениями.

Тесты проводились на системе:

Процессор Intel Core i5 7200U;

Оперативная память 12 GB.

# Заключение

В заключении можно сказать, что все поставленные цели и задачи были выполнены, а именно: созданы классы «TDatLink» и «List» с реализованными методами добавления, удаления и доступа к элементам списка, а также написаны к ним тесты, и они успешно пройдены.

# Литература

* Учебные материалы к учебному курсу «Методы программирования» - Гергель В.П.