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**

**«Односвязный линейный список с использованием указателей»**

**Выполнил:** студент группы 381706-1

Суслов Егор Игоревич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Лебедев И.Г

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533083472)

[2. Постановка задачи 4](#_Toc533083473)

[3. Руководство пользователя 5](#_Toc533083474)

[4. Руководство программиста 6](#_Toc533083475)

[4.1. Описание структуры программы 6](#_Toc533083476)

[4.2. Описание структур данных 6](#_Toc533083477)

[4.3. Описание алгоритмов 7](#_Toc533083478)

[5. Заключение 8](#_Toc533083479)

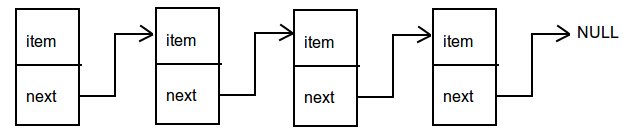
[6. Литература 9](#_Toc533083480)

# Введение

**Линейный однонаправленный список** — это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на [NULL](https://ru.wikipedia.org/wiki/NULL_(%D0%A1%D0%B8)). Элемент, на который нет указателя, является первым (головным) элементом списка. Здесь ссылка в каждом узле указывает на следующий узел в списке. В односвязном списке можно передвигаться только в сторону конца списка. Узнать адрес предыдущего элемента, опираясь на содержимое текущего узла, невозможно.

В [информатике](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) **линейный список** обычно определяется как [абстрактный тип данных](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) (АТД), формализующий понятие упорядоченной [коллекции данных](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D1%8F_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). На практике линейные списки обычно реализуются при помощи [массивов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и связных списков. Иногда термин «список» неформально используется также как синоним понятия «связный список».

Начало списка называют головным элементом, а звенья списка - узлами. Каждый узел односвязного списка помимо лежащего в нем значения, содержит поле указателя на следующий узел. Поле указателя последнего узла содержит нулевое значение (указывает на NULL).



Структура односвязного списка.

В односвязном списке вставка и удаление узлов производится либо в начало, либо в конец списка.

Структура списка ограничивает доступ к его узлам по индексу. Список нельзя индексировать, как массив. Чтобы попасть на некоторый узел односвязного списка, необходимо последовательно пройти весь путь от головного элемента до нужного узла.

# Постановка задачи

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

Для работы со списком необходимо реализовать операции:

* добавления в начало узла списка,
* добавления в конец узла списка,
* добавление и извлечение узла в промежуточные узлы списка,
* извлечения с удалением узла из начала списка,
* извлечения с удалением узла из конца списка,
* проверка списка на полноту/пустоту.

Программное решение будет выглядеть следующим образом:

1. Вспомогательный класс TElem – единичная составляющая списка.
2. Класс списка – TList.
3. Класс для обработки исключений – MyException.
4. Программа, демонстрирующая работу класса TList. (получается из файла кода «List\_main.cpp».
5. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

При запуске консольного приложения «List\_main.cpp» пользователю выводятся на консоль сначала список заполненный с одной стороны 5 элементами добавлением в начало и аналогично 5 элементами добавлением в конец. Итого список состоит из 10 элементов. Затем дважды выводятся на консоль и соответственно из списка первый и последний элементы. Затем выводится список уже без этих элементов (т.е. из 6 элементов).

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль List. Содержит пример использования списка. Реализация в файле «List\_main*.*cpp*».*
* Модуль ListLib – статическая библиотека. Содержит файл «List.h», в котором описан интерфейс и реализация шаблонного класса TList. А также содержит файл «Elem.h», в котором представлен интерфейс и реализация вспомогательного шаблонного класса TElem.
* Модуль ListTest. Содержит 26 тестов, описанных в файле «ListTest.cpp» и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## Описание структур данных

#### Класс TElem – класс единичного элемента списка

Класс TElem является шаблонным. Всего в классе два поля, объявленные со спецификатором доступа protected:

* T data – переменная под хранение данных в узле
* TElem <T>\* next – указатель на следующий узел списка.

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

* TElem(T \_data = 0, TElem <T>\* \_next = 0) – конструктор по умолчанию.
* TElem(TElem<T> &L) – конструктор копирования.
* virtual ~TElem() – деструктор.
* TElem\* GetNext() – получить указатель на следующий узел.
* T GetData() – получить значение из узла.
* void Set(T e) – установить значение в звено списка.
* void SetNext(TElem <T>\* n) – установить указатель на следующее звено списка.

#### Класс TList – класс списка.

Класс TList списка является шаблонным. В классе всего два поля , объявленные со спецификатором доступа protected:

* TElem <T>\* begin – указатель на начало списка
* int count – количество элементов в списке

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

* TList() – конструктор по умолчанию.
* TList(TList<T> &L) – конструктор копирования.
* virtual ~TList() – деструктор.
* void Put(int \_n, T elem) – положить элемент в списке на позицию \_n.
* void Get(int \_n) – извлечь элемент в списке на позиции \_n.
* void PutBegin(T a) – положить элемент в начало списка.
* void PutEnd(T a) – положить элемент в конец списка.
* T GetBegin() – взять элемент с удалением из начала.
* T GetEnd() – взять элемент списка с удалением из конца.
* bool IsEmpty() – метод проверки списка на пустоту.
* void Print() – метод вывода списка на консоль.

## Описание алгоритмов

**Добавление звена списка в начало.**

При добавлении звена в начало списка мы создаем указатель на объект класса TElem. Затем выделяем память под объект этого класса и с помощью конструктора с параметрами для TElem, передав туда значение, которое необходимо положить в начало списока, и указатель на текущее начало, создаем очередное звено списка. Указатель на начало списка переопределяем на только что добавленный элемент.

**Удаление звена списка из начала.**

Для удаления звена списка из начала выполняем проверку на пустоту списка. Если список пуст, то бросаем исключение. Иначе создаем указатель \*temp на объект класса TElem, которому присваиваем значение текущего начала списка. Создаем временную переменную tmp, в которую записываем значение, хранящееся в первом элементе списка. Начало списка устанавливаем на следующий за удаляемым элемент. Удаляем указатель \*temp для того, чтобы очистить память, занимаемую бывшим первым элементом.

**Добавление звена списка в конец.**

При добавлении звена списка в конец проверяем, есть ли элементы в списке. Если есть, то создаем указатель \*a на объект класса TElem, в него записываем значение начала списка. В цикле ищем текущий последний элемент. Как только конец списка будет найден, выделяем память под новое звено списка и с помощью конструктора по умолчанию TElem создаем его. Устанавливаем для текущего последнего элемента указатель на следующий – только что созданный.

В том случае, если в списке не было элементов, то указателю на начало списка присваиваем значение, указывающее на звено, созданное с помощью конструктора TElem.

**Удаление звена списка из конца.**

Для удаления звена списка из конца выполняем проверку на пустоту списка. Если список пуст, то бросаем исключение. Иначе необходимо проверить: в списке больше одного элемент или ровно один. Для этого смотрим на следующий за первым элемент. Если указатель на него равен нулю, то мы возвращаем только данные из первого элемента списка, начало списка обнуляем.

Ели элементов больше одного, то создаем указатель \*temp на объект класса TElem. Ищем в цикле предпоследнее звено списка. Создаем еще один указатель \*temp1 на объект класса TElem. В него записываем указатель на последнее звено списка. Получаем данные из этого звена. Удаляем указатель \*temp1 и тем самым освобождаем память, занимаемую бывшим последним элементом. Для \*temp, устанавливаем в качестве следующего за ним 0, т.к. он теперь стал последним.

# Заключение

Все поставленные задачи были выполнены это подтверждается Google C++ Testing Framework (все 26 тестов на оба класса TElem и TList успешно выполняются) и успешным выполнение файла кода «List\_main.cpp».

# Литература

* Книги:

1. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.
2. Топп У., Форд У. Структуры данных в С++. - М. Бином, 1999.
3. Мейн М., Савитч У. Структуры данных и другие объекты в С++. - М.: Издательский дом “Вильямс”, 2003.
4. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.

* Ссылки в Internet:

1. Учебно-методическое пособие из электронной библиотеки ННГУ: «ЛАБОРАТОРНЫЙ ПРАКТИКУМ» : [http://www.unn.ru/books/met\_files/Pract\_ADS.pdf].