МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Национальный исследовательский**

**Нижегородский государственный университет им. Н.И. Лобачевского»**

**(ННГУ)**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**Отчет по учебной практике**

**«Структура данных – односвязный линейный список с использованием указателей»**

**Выполнил:** студент группы 381706-1

Голованова Елена Александровна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Оглавление

[1. Введение 3](#_Toc1351811)

[2. Постановка задачи 4](#_Toc1351812)

[3. Руководство пользователя 5](#_Toc1351813)

[4. Руководство программиста 6](#_Toc1351814)

[a. Описание структуры программы 6](#_Toc1351815)

[b. Описание структур данных 6](#_Toc1351816)

[c. Описание алгоритмов 7](#_Toc1351817)

[5. Заключение 9](#_Toc1351818)

[6. Список литературы 10](#_Toc1351819)

# Введение

Одной из главных проблем программировании является нехватка памяти. Очень много программ требуется достаточно большого объема вместимости, и поэтому появляются многие разные структуры хранения данных.

**Структура данных** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *data structure*) — программная единица, позволяющая хранить и обрабатывать множество однотипных и/или логически связанных [данных](https://ru.wikipedia.org/wiki/%D0%94%D0%B0%D0%BD%D0%BD%D1%8B%D0%B5_(%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)) в [вычислительной технике](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80).

В этой лабораторной работе мы рассмотрим такую структуру, как односвязный линейный список с использованием указателей.

**Связный список** — базовая динамическая структура данных в информатике, состоящая из [узлов](https://ru.wikipedia.org/wiki/%D0%A3%D0%B7%D0%B5%D0%BB_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), каждый из которых содержит как собственно [данные](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B5_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)), так и одну или две [ссылки](https://ru.wikipedia.org/wiki/%D0%A1%D1%81%D1%8B%D0%BB%D0%BA%D0%B0_(%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)) («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед [массивом](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)) 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.



*Рис. 1 - Организация 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.

Цель данной лабораторной работы – реализовать такую структуру данных как стек.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Реализация класса TList, при помощи вспомогательного класса TElem.
2. Выполнение определенных функций, операторов и конструкторов в ранее упомянутых классах.
3. Создать класс TException для избежания ошибок.
4. Выполнение примера использования программы.
5. Реализация простых тестов на базе Google Test.

# Руководство пользователя

Работа пользователя происходит так:

1. В начале программы у пользователя спрашивают количество элементов, которое кладут в начало списка.
2. Далее пользователь должен ввести количество элементов, которые будут лежать в конце списка.
3. После этого на экране выводится полученный список.
4. Следующим действием будет изъятия элементов с конца и начала списка, после чего они появятся на экране.
5. Последним пунктом программы является вывод списка без изъятых элементов.

# 4. Руководство программиста

## Описание структуры программы

Для реализации лабораторной работы создается несколько модулей:

* Модуль List

Этот модуль содержит реализацию примера использования программы.

* Модуль ListLib

Этот модуль содержит реализацию списка. 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

Этот модуль содержит тесты, проверяющие работу каждого методов класса TList. Они разработаны с помощью использования Google C++ Testing Framework.

* Модуль Exception

Этот модуль содержит класс исключений.

## Описание структур данных

Класс TElem – класс узлов списка. Он является шаблонным.

1. В зоне protected определены два поля:

* T dannyye - переменная под хранение данных;
* TElem<T>\* next - указатель на следующий узел.

2. В зоне public:

* TElem<T>(T \_dannyye = 0, TElem<T>\* n = 0) - конструктор ;
* Elem<T>(TElem<T>& A) - конструктор копирования;
* T GetDannyye() - получение значения из узла;
* TElem<T>\* GetNext() - получение указателя на следующий узел;
* void SetDannyye(T \_dannyye) - установление данных;
* void SetNext(TElem<T>\* n) - установления указателя на следующее звено списка.

Класс TList – класс шаблонный.

1. В зоне protected определены два поля:

* TElem<T>\* begin - указатель на элемент в начале списка;
* int num - количество элементов в списке.

2. В зоне public:

* TList() - конструктор по умолчанию;
* TList(TList<T>& F) - конструктор копирования;
* virtual ~TList() – деструктор;
* int GetSize() - получение количества элементов в листе;
* void Put(int \_n, T elem) - установление элемента на позицию n в списке;
* T Get(int \_n) - получение элемента из списка;
* void PutBegin(T A) – установление элемента в начало;
* void PutEnd(T A) – установление элемента в конец;
* T GetBegin() - получение элемента в начале списка;
* T GetEnd() - получение элемента в конце списка;
* bool IsFull() - проверка на полноту;
* bool IsEmpty() - проверка на пустоту;
* void Print() - печать листа.

## Описание алгоритмов

* Добавление звена списка в начало:

Для этого мы добавляем в начало списка указатель на объект класса TElem. После для объекта этого класса выделяем памяти, и создаем очередное звено списка при помощи конструктора с параметрами для TElem, передав туда значение, которое необходимо положить в начало списка, и указатель на текущее начало. Далее мы обновляем указатель на начало списка на только что добавленный элемент.

* Удаление звена списка из начала:

Для выполнения этой функции в начале мы выполняем проверку на пустоту списка во избежания ошибок.(Для таких ситуаций у нас создан класс исключений) Иначе создаем указатель \*temp на объект класса TElem, которому присваиваем значение текущего начала списка. Создаем временную переменную tmp*,* в которой будет лежать значение, хранящееся в первом элементе списка. Потом устанавливаем начало списка на следующий за удаляемым элемент. Далее мы удаляем указатель \*temp для очищения памяти.

* Добавление звена списка в конец:

В начале мы делаем с помощью исключения проверку, пуст ли список. Если в нем есть элементы, то создаем указатель \*a на объект класса TElem, в него записываем значение начала списка. После этого ищем в цикле текущий последний элемент*.* Сразу же как конец списка будет найден, мы должны выделить под новое звено списка память и с помощью конструктора по умолчанию TElem создаем его. Далее устанавливаем для текущего последнего элемента указатель на следующий – только что созданный. Если же у нас список был не пустой, то указателю на начало списка присваиваем значение, указывающее на звено, созданное с помощью конструктора TElem.

* Удаление звена списка из конца:

Чтобы удалить звено списка из конца мы делаем с помощью исключения проверку, пуст ли список. Если список пуст, то бросаем исключение. В другом случае нужна проверка: сколько у нас элементов в списке (1 или >1). Для этой проверкимы смотрим на второй по счету элемент. Если указатель на него равен нулю, то мы возвращаем только данные из первого элемента списка, начало списка обнуляем.

В том случае если элементов больше одного, то создаем указатель \*temp на объект класса TElem. Далее происходит поиск в цикле предпоследнего звена списка. Создаем еще один указатель *\**temp1на 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.

# 5. Заключение

В результате выполнения данной лабораторной работы была разработана структура хранения данных – односвязные списки с использованием указателей. Они были реализованы при помощи нескольких классов (их реализация была описана выше).

Получены новые знания и навыки.

Были реализованы тесты для проверки работоспособности вышеперечисленных классов на базе GoogleTest.

# 6. Список литературы

1. Википедия. Статья «Структура данных»

[https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0\_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85]

1. Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.
2. Гергель В.П. Методические материалы по курсу «Методы программирования 2», Нижний Новгород, 2015.
3. Википедия. Статья «Односвязные списки»

[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]