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**

**«Структура хранения данных: Односвязный линейный список с использованием указателей»**

Научный руководитель

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

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Исполнитель

Студент группы 381706-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Паршина С.С.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Нижний Новгород

2019 г

**Содержание**

[Введение 3](#_Toc22869414)

[1 Постановка задачи 5](#_Toc22869415)

[2 Руководство пользователя 6](#_Toc22869416)

[3 Руководство программиста 8](#_Toc22869417)

[3.1 Описание структуры программы 8](#_Toc22869418)

[3.2 Класс TElem – класс элемента списка 8](#_Toc22869419)

[3.3 Класс TList – класс односвязного списка на указателях 9](#_Toc22869420)

[3.4 Класс TException – класс исключений 9](#_Toc22869421)

[4 Описание алгоритмов 9](#_Toc22869422)

[5 Анализ сложности 11](#_Toc22869423)

[Заключение 12](#_Toc22869424)

[Список используемой литературы 13](#_Toc22869425)

# Введение

Цель данной лабораторной работы – теоретическое и практическое изучения динамической структуры данных – линейного односвязного списка с использованием указателей.

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

Длина списка равна числу элементов, содержащихся в списке, список нулевой длины называется пустым списком. Списки представляют собой способ организации структуры данных, при которой элементы некоторого типа образуют цепочку. Для связывания элементов в списке используют систему указателей. В минимальном случае, любой элемент линейного списка имеет один указатель, который указывает на следующий элемент в списке или является пустым указателем, что интерпретируется как конец списка.

Структура, элементами которой служат записи с одним и тем же форматом, связанные друг с другом с помощью указателей, хранящихся в самих элементах, называют связанным списком. В связанном списке элементы линейно упорядочены, но порядок определяется не номерами, как в массиве, а указателями, входящими в состав элементов списка. Каждый список имеет особый элемент, называемый указателем начала списка (головой списка), который обычно по содержанию отличен от остальных элементов. В поле указателя последнего элемента списка находится специальный признак NULL, свидетельствующий о конце списка.

Линейные связные списки являются простейшими динамическими структурами данных. Из всего многообразия связанных списков можно выделить следующие основные:

* однонаправленные (односвязные) списки;
* двунаправленные (двусвязные) списки;
* циклические (кольцевые) списки.

Наиболее простой динамической структурой является однонаправленный список, элементами которого служат объекты структурного типа.

Однонаправленный (односвязный) список – это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка. В последнем элементе указатель на следующий элемент равен NULL.

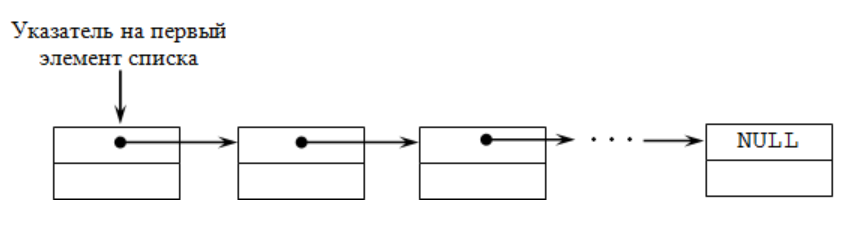


Рисунок 1. Структура односвязного списка на указателях

# Постановка задачи

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

Списки хранятся в динамической области памяти, используя указатели. Благодаря такой структуре возможно экономно расходовать память, при необходимости выделяя новую и не занимая лишние ее участки. Однако нельзя забывать о корректном удалении неиспользуемых указателей.

Для работы со списком будут реализованы операции:

* проверка списка на полноту;
* проверка списка на пустоту;
* добавление в начало нового звена списка;
* добавление нового звена в указанную позицию;
* добавление в конец нового звена списка;
* извлечение с удалением звена из начала списка;
* извлечение с удалением звена из указанной позиции;
* извлечение с удалением звена из конца списка.

Программное решение будет выглядеть следующим образом:

1. Класс односвязного линейного списка на указателях – TList.
2. Вспомогательный класс одного отдельного элемента списка – ­TElem.
3. Класс для обработки любых исключений, возникающих в ходе выполнения программы – TException.
4. Набор автоматических тестов с использованием Google C++ Testing Framework ListTest.
5. Программа, демонстрирующая работу класса TList на конкретных примерах, заданных вручную.

# Руководство пользователя

Действия пользователя начинается с запуска файла «main\_List.exe», полученного после компиляции файла «main\_List.cpp». Затем последовательно выполняются заполнение листа любыми целыми элементами с начала, с конца, а затем в необходимую позицию в листе по выбору. На рисунке 2 наглядно показан процесс добавления элементов. В качестве примера использования списка предлагается следующее. Выполняются различные операции добавления элементов в список. Пользователь заполняет их любыми целыми значениями. Процесс добавления элементов сопровождается наглядной демонстрацией того, как изменяется список.

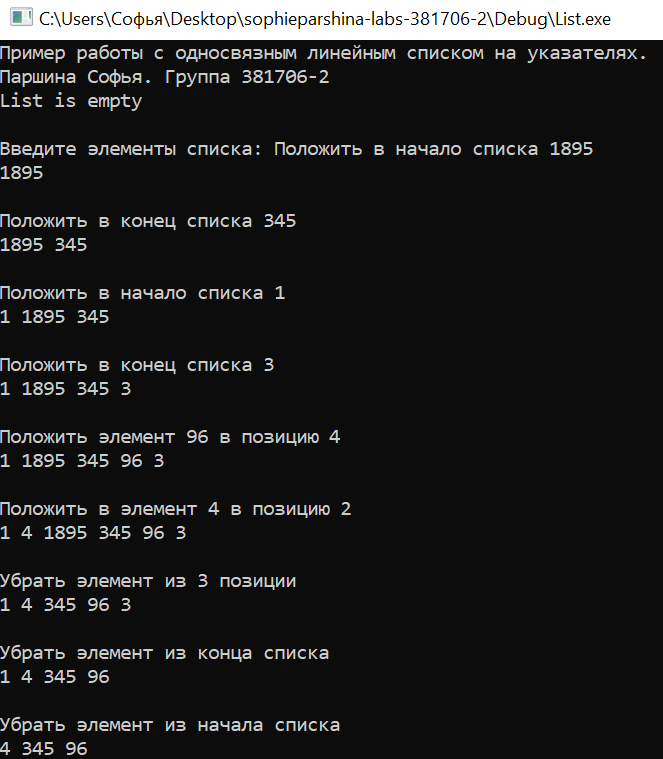


Рисунок 2. Добавление элементов в линейный список на различные позиции

Затем в несколько этапов осуществляется извлечение с удалением элементов из начала, конца и любой заданной позиции в списке по выбору (см. рис.3). На каждом этапе совершения любых действий со списком осуществляется его проверка на пустоту, полноту с подключением соответствующей библиотеки исключений и выводом сообщений об ошибке на экран. Последней проверяется работа конструктора копирования и отображаются соответствующие адреса старого и нового списков в памяти.

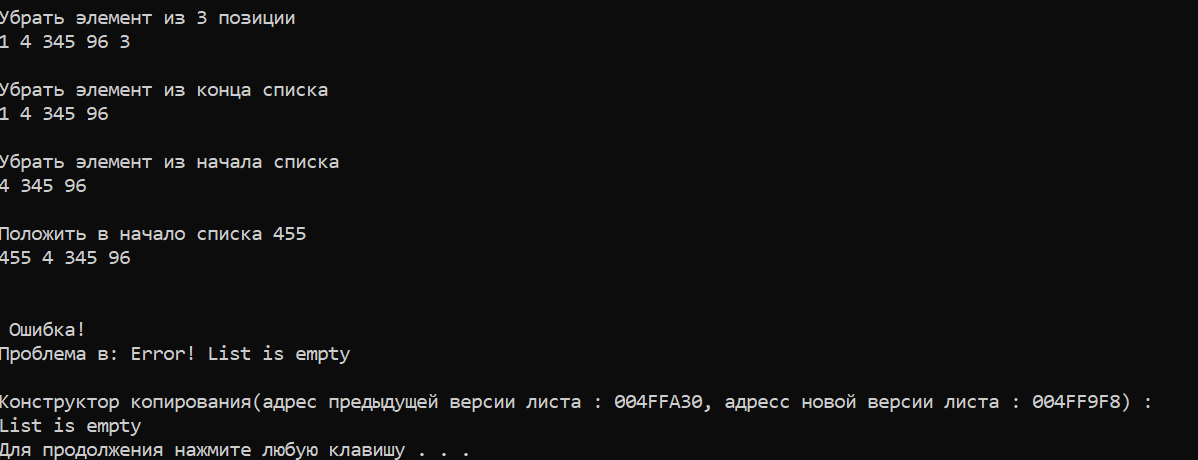


Рисунок 3. Извлечение элементов из списка и вывод сообщения об ошибке

# Руководство программиста

## 3.1 Описание структуры программы

Приведем описание программных модулей:

1. Модуль ListLib – статическая библиотека, состоящая из двух основных файлов: List.h, в котором описан интерфейс и реализация шаблонного класса TList и Elem.h, в котором представлен интерфейс и реализация вспомогательного шаблонного класса элемента списка TElem.
2. Модуль List. Содержит пример использования списка. Реализация представлена в файле main\_List.cpp.
3. Модуль ListTest. Содержит 19 тестов, описанных в файле ListTest.cpp и разработанных с помощью использования Google C++ Testing Framework.
4. Модуль ExceptionLib – библиотека, позволяющая создавать описания всех возможных исключений в программе.

## 3.2 Класс TElem – класс элемента списка

Рассмотрим класс *TElem* подробно.

template <class T> class TElem {…} – шаблонный класс элемента списка, используем его для хранения данных любого типа.

Элементы класса, объявленные со спецификатором protected, то есть доступ к этим элементам открыт классам, производным от данного:

TElem <T>\* next – указатель на следующий элемент односвязного списка.

T data – значение элемента односвязного списка.

Элементы класса, объявленные со спецификатором public:

TElem(T \_data = 0, TElem <T>\* \_next = 0) – конструктор класса с параметрами.

TElem(TElem<T> &element) – конструктор копирования.

virtual ~TElem() – деструктор.

T GetData() – получить значение элемента списка.

void SetData(T element) – назначить значение элемента списка.

TElem\* GetNext() – получить указатель на следующий элемент.

void SetNext(TElem <T>\* n) – назначить указатель на следующее звено списка.

## 3.3 Класс TList – класс односвязного списка на указателях

Рассмотрим элементы класса:

template <class T> class TList {…} – шаблонный класс списка, используем его для хранения данных любого типа.

Элементы класса, объявленные со спецификатором protected:

int count – количество элементов односвязного списка.

TElem <T>\* begin – указатель на начало списка.

Элементы класса, объявленные со спецификатором public:

TList() – конструктор класса по умолчанию.

TList(TList<T> &list) – конструктор копирования.

virtual ~TList() – деструктор.

void Put(int \_n, T element) – метод, реализующий размещение элемента в указаную позицию списка.

void PutBegin(T element) – метод, реализующий размещение элемента в начало списка.

void PutEnd(T element) – метод, реализующий размещение элемента в начало списка.

T Get(int \_n) – метод, реализующий забирание элемента по указанной позиции списка.

T GetBegin() – метод, реализующий забирание элемента из начала списка.

T GetEnd() – метод, реализующий забирание элемента из конца списка.

void Print() – печать элементов списка.

bool IsFull() – проверка списка на полноту.

bool IsEmpty() – проверка списка на пустоту.

## 3.4 Класс TException – класс исключений

Класс содержит поле со спецификатором private string msg – переменная, хранящая сообщение об ошибке в виде строки, а также два элемента со спецификатором public:

TException(std::string \_msg) : msg(\_msg) – конструктор с одним параметром.

void Print() – метод для печати ошибки.

# Описание алгоритмов

Рассмотрим некоторые алгоритмы, работа которых не очевидна на первый взгляд.

1 Добавление элемента списка в начало.

Чтобы добавить элемент в начало списка, необходимо создать указатель на объект класса TElem, выделить память под объект этого класса и с помощью конструктора с параметрами для TElem, передав в него значение, необходимое положить в список, и указатель на текущее начало, создаем очередной элемент списка. Указатель на начало списка переопределяем на только что добавленный элемент.

2 Добавление элемента списка в конец.

Для начала нужно проверить список на пустоту. Если в нем есть элементы, то создаем указатель \*bgn на объект класса TElem, в него записываем значение начала списка. В цикле ищем текущий последний элемент, путем изменения значения bgn*.* Когда 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 нет элементов, то указателю на начало списка присваиваем значение, указывающее на звено, созданное с помощью конструктора TElem.

3 Удаление элемента списка из начала.

Для начала нужно проверить список на пустоту. Если список пуст, то выдаем на экран сообщение об ошибке. Иначе создаем указатель \*bgn на объект класса TElem, которому присваиваем значение текущего начала списка. Создаем временную переменную bgn, в которую записываем значение, хранящееся в первом элементе списка. Начало списка устанавливаем на следующий за удаляемым элемент. Удаляем указатель \*bgn для того, чтобы очистить память, занимаемую бывшим первым элементом.

4 Удаление элемента 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 нуля, то создаем указатель \*a на объект класса TElem. Ищем в цикле предпоследнее звено списка. Создаем еще один указатель \*a1 на объект класса TElem. В него записываем указатель на последнее звено списка. Получаем данные из этого звена. Удаляем указатель \*a1 и тем самым освобождаем память, занимаемую бывшим последним элементом. Для \*a, устанавливаем в качестве следующего за ним 0, т.к. он теперь стал последним.

# Анализ сложности

Метод *PutBegin(),* работает значительно быстрее метода *PutEnd()*. Объясняется это тем, что при добавлении элемента в начало списка обращение к памяти происходит 1 раз, отсюда следует, что сложность этого метода равняется O(1). При добавлении элемента в конец списка, необходимо пройти и проверить все элементы списка. Из-за этого происходит n обращений к памяти, а значит сложность алгоритма равняется O(n), что и объясняет более долгую работу этого метода.

# Заключение

В ходе выполнения данной лабораторной работы были теоретически изучены структура и принципы построения односвязного линейного списка на указателях. Для его программной реализации была создана библиотека, представляющая шаблонный класс списка и дающая возможность выполнять проверку списка на пустоту и полноту, добавление, извлечение элементов списка в начало, конец или на указанную позицию.

Были созданы и успешно пройдены тесты, созданные с использованием Google C++ Testing Framework.

Программное решение было продемонстрировано с помощью простейшего набора операций над заданным списком. Описание примера работы со списком было представлено в разделе «Руководство пользователя».

# Список используемой литературы

1. А.О. Грудзинский А.О. Методы программирования. Курс на основеязыка Obiect Pascal / Грудзинский А.О., Мееров И.Б., Сысоев А.В. Учебное пособие. - 391 с. (Диплом Приволжского  регионального конкурса «Университетская книга – 2007»).
2. А.Н. Васильев Самоучитель С++ с примерами и задачами/Васильев А.Н. -СПб.: Наука и Техника, 2016. - 480с.
3. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015 – Режим доступа: <http://www.itmm.unn.ru/files/2018/11/1.5.-Struktury-hraneniya-s-ispolzovaniem-ukazatelej-spiski.pdf> – Загл. с экрана.
4. Википедия: свободная электронная энциклопедия: на русском языке. Линейный список – Режим доступа: <https://ru.wikipedia.org/wiki/Список_(информатика)> – Загл. с экрана.
5. ИНТУИТ Национальный Открытый Университет. Лекция 30: Динамические структуры данных: однонаправленные и двунаправленные списки – Режим доступа: <https://www.intuit.ru/studies/courses/648/504/lecture/11456> – Загл. с экрана.