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 №7**

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

**Выполнил:** студент группы 381903-3

Зинкин Кирилл Сергеевич

**Проверил:**

ассистент каф. МОСТ ИИТММ Лебедев Илья Геннадьевич

Нижний Новгород

2020.

Содержание

[1. Введение 3](#_Toc533088478)

[2. Постановка задачи 4](#_Toc533088479)

[3. Руководство пользователя 5](#_Toc533088480)

[4. Руководство программиста 6](#_Toc533088481)

[4.1 Описание структуры программы 6](#_Toc533088482)

[4.2 Описание структур данных 6](#_Toc533088483)

[4.3 Описание алгоритмов 7](#_Toc533088484)

[5. Заключение 9](#_Toc533088486)

[6. Литература 10](#_Toc533088487)

# Введение

**Односвязный список** – это структура данных, представляющая собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза.

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).

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 – узел списка.
2. Класс списка – TList.
3. Класс для обработки исключений – Exception, которые могут возникнуть при выполнении различных операций.
4. Программа, демонстрирующая работу класса TList.
5. Набор автоматических тестов с использованием гугл тестов.

# Руководство пользователя

Рассмотрим пример использования классов TList.

При запуске программы с пользователя спрашивается количество элементов , которое следует положить в начало списка. Затем спрашивается количество элементов , которое необходимо положить в конец списка. Полученный список выводится на экран. Он будет выглядеть следующим образом: .

Затем выполняется изъятие элементов из конца и начала списка. Изъятые элементы выводятся на экран. В конце на консоль выводится список, полученный после изъятия из него элементов. На этом работа программы прекращается.

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из следующих модулей:

* Модуль List. Содержит пример использования списка. Реализация в файле *main\_List.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 24 теста, описанных в файле *ListTest.cpp* и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## 4.2 Описание структур данных

#### Класс TElem – класс узла списка

Класс TElem является шаблонным. Всего в классе два поля, объявленные со спецификатором доступа protected: T data – переменная под хранение данных в узле и 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 *TElem <T>\* begin* – указатель на начало списка и *int count* – количество элементов в списке, объявленное со спецификатором доступа protected.

**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.

## 4.3 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

В ходе выполнения лабораторной был произведен анализ задачи - установлено понятие списка на указателях. Была разработана библиотека, реализующая шаблонный класс списка. Она позволяет при работе со списком выполнять базовые операции извлечения/добавление элементов списка.

Проведены эксперименты и установлен наиболее оптимальный метод работы со списками.

Предоставлено описание примера работы со списком в разделе «Руководство пользователя».

Также разработаны и доведены до успешного выполнения тесты, проверяющие корректность методов классов TList и TElem.

# Литература

* Книги

A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.

Васильев А.Н. Самоучитель С++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.

* Ссылки в Internet

Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [http://www.itmm.unn.ru/files/2018/11/1.5.-Struktury-hraneniya-s-ispolzovaniem-ukazatelej-spiski.pdf], 2015.

Википедия: свободная электронная энциклопедия: на русском языке: [https://ru.wikipedia.org/wiki/Список\_(информатика)]