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»**

**Выполнил:** студент группы 381706-1

Лембриков Степан Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

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. Эксперименты 9](#_Toc533088485)

[6. Заключение 9](#_Toc533088486)

[7. Литература 10](#_Toc533088487)

# Введение

Односвязный список - это совокупность нескольких объектов, каждый из которых представляет собой элемент списка, состоящий из двух частей. Первая часть элемента - значение, которое он хранит, вторая - информация о следующем элементе списка.

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).

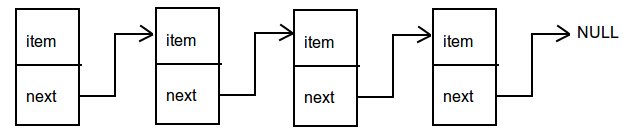


Рисунок 1 Структура односвязного списка.

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.

Лабораторная работа направлена на практическое освоение динамической структуры данных – линейного односвязного списка с использованием указателей.

# Постановка задачи

* Разработка и реализация класса список – TList.
* Создание класса MyException для обработки исключений, которые могут возникнуть при выполнении различных операций.
* Программа, демонстрирующая работу класса TList.
* Написание Google тестов, позволяющих проверить работу программы в автоматическом режиме.

# Руководство пользователя

Рассмотрим пример использования классов TList.

При запуске программы пользователю нужно будет ввести количество элементов , которое следует положить в начало списка. Потом нужно будет ввести количество элементов , которое надо положить в конец списка. Полученный список будет выведен экран. Вид списка после вывода: .

После этого элементы будут изъяты из конца и начала списка. Элементы при этом будут выводиться на экран. В конце будет выведен новый список, без изъятых элементов.

После этого программа завершит свою работу.

# Руководство программиста

## 4.1 Описание структуры программы

**Проект “List” включает в себя:**

1. *main\_List.cpp* (в нём находится main)

**Проект “ListLib” включает в себя:**

1. List.h (описание и реализация класса “ TList ”)
2. List.cpp
3. Elem.h (реализация класса TElem)

**Проект “** **QueueTest” включает в себя:**

1. ListTest.cpp (тесты, разработанные с помощью Google C++ Testing Framework)

**Проект “** ExceptionLib **” включает в себя:**

1. библиотека, содержащая класс исключений.

# 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 списка, 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.

# Эксперименты

Комплектующие компьютера:

Процессор Intel Core i-3 8100 3,6 GHz;

Оперативная память: 16 ГБ;

Система: windows 10, 64-х разрядная.

|  |  |  |
| --- | --- | --- |
| Кол-во элементов в списке | Время работы метода *PutBegin()* в млс.  O(1) | Время работы метода *PutEnd()* в млс.  O(n) |
| 10 000 | 0 | 0 |
| 100 000 | 0 | 2 |
| 1 000 000 | 0 | 12 |

Таблица 1. Время работы методов добавления элементов в начало и в конец списка

Как видно из таблицы метод *PutBegin(),* работает значительно быстрее метода *PutEnd()*. Это происходит за счет того, что при добавлении элемента в начало списка, к памяти мы обращаемся всего 1 раз, за O(1), а для добавления элемента в конец списка, мы должны пройти по всем n элементам списка, тем самым мы n раз обратимся к памяти, и сложность алгоритма равна O(n), что существенно замедляет работу.

Делаем вывод, при работе со списками в приоритете стоит использовать метод *PutBegin().*

# Заключение

В результате проведённой работы я понял, что такое очередь, разработал библиотеку ListLib и реализовал класс стека TList.

Также я создал программу – реализацию, демонстрирующую работу основных методов класса TList.

Проведены эксперименты и установлен наиболее оптимальный метод работы со списками.

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

Разработал тесты, разработанные для данного программного проекта с использованием Google C++ Testing Framework, и добился их выполнения.

# Литература

* Книги

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

* Ссылки в Internet

1. Гергель В.П. Методические материалы по курсу «Методы программирования 2»: [<http://www.itmm.unn.ru/files/2018/10/Primer-1.3.-Struktury-hraneniya-spiska.pdf>], 2015.
2. METANIT.COM: список: [<https://metanit.com/sharp/algoritm/2.3.php>]
3. Википедия: свободная электронная энциклопедия: на русском языке: [https://ru.wikipedia.org/wiki/Список]