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 односвязного линейного списка с использованием указателей»**

**Выполнил:** студент группы 381706-1

Силенко Дмитрий Игоревич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc536801962)

[2. Постановка задачи 4](#_Toc536801963)

[3. Руководство пользователя 5](#_Toc536801964)

[4. Руководство программиста 6](#_Toc536801965)

[4.1. Описание структуры программы 6](#_Toc536801966)

[4.2. Описание структур данных 6](#_Toc536801967)

[4.3. Описание алгоритмов 7](#_Toc536801968)

[5. Эксперименты 9](#_Toc536801969)

[6. Заключение 10](#_Toc536801970)

# Введение

**Основная цель данной работы** – разработка динамической структуры данных - линейного односвязного списка с использованием указателей.

Но для начала необходимо разобраться, что такое односвязный список на указателях и как он работает.

**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).

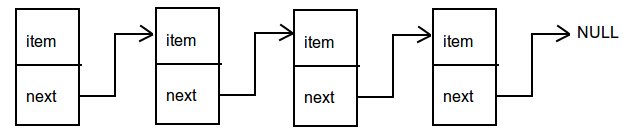


Рисунок 1 Структура односвязного списка

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.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Разработка и реализация вспомогательного класса элемента списка – TElem, а так же самого класса списка – TList.
2. Пример программы, демонстрирующая работу класса TList.
3. Написание набора автоматических тестов с использованием Google C++ Testing Framework и проверка работоспособности методов классов.

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.

# Руководство пользователя

При запуске программы пользователя просят ввести количество элементов, которые далее будут помещены в начало списка.



Рисунок 2 Ввод количества начальных элементов

После чего, пользователя снова просят ввести количество элементов, но на этот раз их поместят не в начало списка, а в его конец.

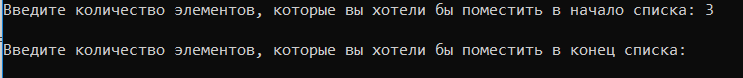


Рисунок 3 Вывод количества замыкающих элементов

Далее, программа выводит все элементы получившегося списка. Помимо этого, она так же выводит первый и последний элемент списка и список, в котором данные элементы уже отсутствуют.

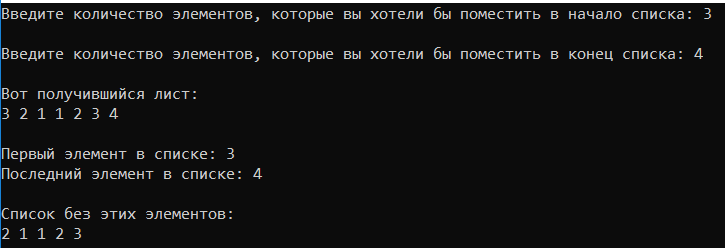


Рисунок 4 Вывод элементов списка, начального и конечного элемента, а также списка без них

После всего этого, пользователю предлагается ввести новый элемент в уже сформированный список. При этом у него спрашиваются позиция будущего элемента списка и его значение. И выводится окончательный список, включающий новый элемент.

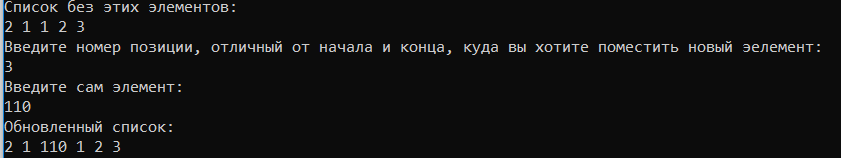


Рисунок 5 Ввод нового элемента

# Руководство программиста

## Описание структуры программы

Программа состоит из следующих модулей:

* Модуль ListLib. Статическая библиотека. Включает в себя заголовочный файл List.h, в котором описаны методы с реализаций шаблонного класса *TList.* А также файл Elem.h, в котором описаны методы с реализацией вспомогательного шаблонного класса *TElem*
* Модуль ListTest. Набор тестов для класса TList. Включает в себя файл *ListTest.cpp.* Разработаны они с помощью использования Google C++ Testing Framework.
* Модуль List. Пример использования списка. Включает в себя файл с реализацией *List\_main.cpp.*

## Описание структур данных

*Класс TElem:*

Поля:

T data; - значение элемента

TElem <T>\* next; - указатель на следующий элемент

Конструкторы и деструктор:

TElem(T \_data = 0, TElem <T>\* \_next = 0);

TElem(TElem<T> &el);

Virtual ~TElem();

Методы:

void SetData(T el); - положить значение элемента

void SetNext(TElem <T>\* \_next); - указать адрес следующего элемента

T GetData(); - получить значение следующего элемента

TElem\* GetNext(); - получить адрес следующего элемента

*Класс TList:*

Поля:

TElem<T>\* begin; - указатель на элемент в начале списка

int count; - количество элементов в списке

Конструкторы и деструктор:

TList();

TList(TList<T> &lst);

Virtual ~TList();

Методы:

int GetSize(); - получить количество элементов

void Put(int pos, T elem); - положить элемент на позицию n в списке

void PutBegin(T el); - положить в начало

void PutEnd(T el); - положить в конец

T Get(int pos); - получить элемент из списка

T GetBegin(); - получить первый элемент

T GetEnd(); - получить последний элемент

bool IsFull(); - проверка на полноту

bool IsEmpty(); - проверка на пустоту

void Print(); - вывод листа на экран

## 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 \*temp 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 элемента 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.

Если 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

Эксперименты проводились на ПК с следующими параметрами:

1. Операционная система: Windows 10 Домашняя
2. Процессор: Intel(R) Core™ i5-8250U CPU @ 1.60 GHz
3. Версия Visual Studio: 2017

|  |  |  |
| --- | --- | --- |
| 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 | 4 |
| 1 000 000 | 0 | 17 |

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

Заметим, что метод *PutBegin()* работает намного быстрее метода *PutEnd().* И это не трудно объяснить. При добавлении элемента в начало списка нет необходимости проходить по всем уже имеющимся элементам, а потому к памяти мы обращаемся всего один раз, отсюда и сложность O(1), не зависящая от количества элементов. Добавление элемента в конец списка заставляет нас пройтись по всем n элементам списка, откуда сложность уже O(n), что значительно дольше.

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

# Заключение

Эта лабораторная работа дала возможность более детально разобраться с устройством работы односвязного линейного списка на указателях. В ходе выполнения, был реализован шаблонный класс списка TList, описанный в специально разработанной библиотеке ListLib. Все это позволило нам проводить базовые операции со списками: добавление или изъятие элементов из него.

Помимо этого, для проверки работоспособности всех методов, были написаны автоматические тесты, реализованные с использованием Google C++ Testing Framework. Пример использования списка для пользователя также написан и успешно работает.

1. **Литература**
2. 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.
3. Википедия: свободная электронная энциклопедия: на русском языке: [https://ru.wikipedia.org/wiki/Список\_(информатика)]