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

Остапович Денис Евгеньевич

\_\_\_\_\_\_\_\_\_\_\_ 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 10](#_Toc533088486)

[7. Литература 11](#_Toc533088487)

# Введение

На практике часто встречаются задачи, связанные с перечислением объектов. В некоторых случаях при решении таких задач важно сохранять информацию об уже сделанных шагах решения, чтобы их не повторять. Такую операцию удобно реализовывать с помощью списков.

**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.

**Цель лабораторной работы:** разработать структуру хранения данных список на указателях.

# Постановка задачи

Выполнение работы предполагает решение следующих задач:

1. Описать вспомогательный класс TElem.
2. Описать класс списка – TList.
3. Продемонстрировать работу класса TList.
4. Создать набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

Рассмотрим пример использования класса TList.

После запуска программы пользователю предлагается ввести количество элементов для заполнения списка с начала и с конца. После список заполняется элементами от 1 до введённого пользователем с начала и с конца. Далее на экран выводится получившийся список и изымается по одному элементу с начала и с конца списка, которые так же выводятся на экран. В конце выводится изменённый список. Программа завершается.

# Руководство программиста

## 4.1 Описание структуры программы

Программа состоит из модулей:

* Модуль Lists. Содержит пример использования списка. Реализация в файле *main.cpp.*
* Модуль ListsLib – статическая библиотека. Содержит файл list.h, в котором реализован шаблонный класс *TList*. А также содержит файл elem.h, в котором реализован шаблонный класс *TElem*.
* Модуль ListTest. Содержит 24 теста на Google C++ Testing Framework в файле ListTest.cpp

## 4.2 Описание структур данных

#### Класс TElem – класс узла списка

Класс TElem является шаблонным. В нём определены два поля, объявленные со спецификатором доступа **protected**:

* T data – переменная для хранения данных.ы
* TElem <T>\* next – указатель на следующий узел списка.

**Конструкторы и методы класса со спецификатором доступа public:**

* *TElem(T \_data = 0, TElem <T>\* n = 0)* – конструктор по умолчанию.
* *TElem(TElem<T> &A)* – конструктор копирования.
* *~TElem()* – деструктор.
* *TElem\* GetNext()* – получить указатель на следующий узел.
* *T GetData()* – получить данные из узла.
* *void SetData(T \_data)* – 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 нём два поля со спецификатором доступа **protected**:

* *TElem <T>\* begin* – указатель на начало списка
* *int count* – количество элементов в списке

**Конструкторы и методы класса, объявленные со спецификатором доступа public:**

* *TList()* – конструктор по умолчанию.
* *TList(TList<T> &L)* – конструктор копирования.
* *~TList()* – деструктор.
* *void Push(int \_n, T elem)* – положить элемент в списке на позицию *\_n*.
* *void Pop(int \_n)* – извлечь элемент в списке на позиции *\_n*.
* *void PushFront(T a)* – положить элемент в начало списка.
* *void PushBack(T a)* – положить элемент в конец списка.
* *T PopFront()* – взять элемент с удалением из начала.
* *T PopBack()* – взять элемент списка с удалением из конца.
* *bool IsEmpty()* – метод проверки списка на пустоту.
* *void Print()* – метод вывода списка в консоль.

## 4.3 Описание алгоритмов

**Добавление звена списка в начало.**

При добавлении звена в начало списка создаётся указатель на объект класса TElem. Затем выделяется 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, создаётся следующее звено списка. Указатель на начало списка переопределяем на добавленный элемент.

**Удаление звена списка из начала.**

Для удаления звена списка из начала выполняется проверка на пустоту списка. Если список пуст, то бросается исключение. Иначе создаётся указатель *\*temp* на объект класса TElem, которому присваивается значение текущего начала списка. Создаётся временная переменная *tmp,* в которую записывается значение, хранящееся в первом элементе списка. Начало списка устанавливается на следующий за удаляемым элемент. Удаляется указатель *\*temp* для того, чтобы очистить память, занимаемую бывшим первым элементом.

**Добавление звена списка в конец.**

При добавлении звена списка в конец проверяется, есть ли элементы в списке. Если есть, то создается указатель \*a на объект класса TElem, в него записывается значение начала списка. В цикле ведётся поиск последнего элемента*.* Как только конец списка будет найден, выделяется память под новое звено списка и с помощью конструктора по умолчанию TElem оно добавляется. Устанавливается для текущего последнего элемента указатель на следующий – только что созданный.

В том случае, если в списке не было элементов, то указателю на начало списка присваивается значение, указывающее на звено, созданное с помощью конструктора TElem.

**Удаление звена списка из конца.**

Для удаления звена списка из конца выполняется проверка на пустоту списка. Если список пуст, то бросается исключение. Иначе проверяется: в списке больше одного элемент или ровно один. Для этого рассматривается следующий за первым элемент. Если указатель на него равен нулю, то возвращаются только данные из первого элемента списка, начало списка обнуляется.

Ели элементов больше одного, то создается указатель *\*temp* на объект класса TElem. В цикле ведётся поиск предпоследнего звена списка. Создается еще один указатель *\*temp1* на объект класса TElem. В него записывается указатель на последнее звено списка. Извлекаются данные из этого звена. Удаляется указатель *\*temp1* и тем самым освобождается память, занимаемая бывшим последним элементом. Для *\*temp,* устанавливается 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 Pro
2. Процессор: Intel(R) Core™ i7-8550U CPU @ 1.80 GHz
3. Версия Visual Studio: 2010

|  |  |  |
| --- | --- | --- |
| Кол-во элементов в списке | Время работы метода *PutBegin()* в млс.  O(1) | Время работы метода *PutEnd()* в млс.  O(n) |
| 10 000 | 0 | 1 |
| 100 000 | 0 | 1 |
| 1 000 000 | 0 | 16 |

Как видно из таблицы сложность работы метода **GetBegin()** – O(1). А сложность работы метода **GetEnd()** – O(n)..

# Заключение

В ходе выполнения лабораторной был произведен анализ задачи - установлено понятие списка на указателях. Была разработана библиотека, реализующая шаблонный класс списка.

* Реализован вспомогательный класс элемента списка **TElem**.
* Реализован класс списка **TList**.
* Протестированы методы класса **TList**, а также обеспечена их работоспособность.

# Литература

* Книги

1. A.O. Грудзинский. Методы программирования, Издательство Нижегородского госуниверситета, 2006.
2. Bjarne Stroustrup The C++ programming language

* Ссылки в Internet

1. 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.
2. Википедия: [https://ru.wikipedia.org/wiki/Список\_(информатика)]