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

Денисов Владислав Львович

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Научный руководитель:**

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

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc533151918)

[2. Постановка задачи 4](#_Toc533151919)

[3. Руководство пользователя 5](#_Toc533151920)

[4. Руководство программиста 7](#_Toc533151921)

[4.1 Описание структуры программы 7](#_Toc533151922)

[4.2 Описание структур данных 7](#_Toc533151923)

[4.3 Описание алгоритмов 8](#_Toc533151924)

[4.4 Сравнение производительности. 9](#_Toc533151925)

[5. Заключение 11](#_Toc533151926)

[6. Литература 12](#_Toc533151927)

# Введение

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

Список (англ. list) – это абстрактный тип данных, представляющий собой упорядоченный набор значений, в котором некоторое значение может встречаться более одного раза. Экземпляр списка является компьютерной реализацией математического понятия конечной последовательности. Экземпляры значений, находящихся в списке, называются элементами списка (англ. item, entry либо element); если значение встречается несколько раз, каждое вхождение считается отдельным элементом.



Рисунок 1 Структура односвязного списка из трех элементов.

# Постановка задачи

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

Как следует из названия лабораторной, хранить списки будем в динамической области памяти, используя указатели. Благодаря этому мы сможем экономнее расходовать память, при необходимости выделяя новую и не занимая лишнего. Однако стоит помнить, что необходимо следить за корректным удалением неиспользуемых укащателей.

Для работы со списком будут реализованы операции:

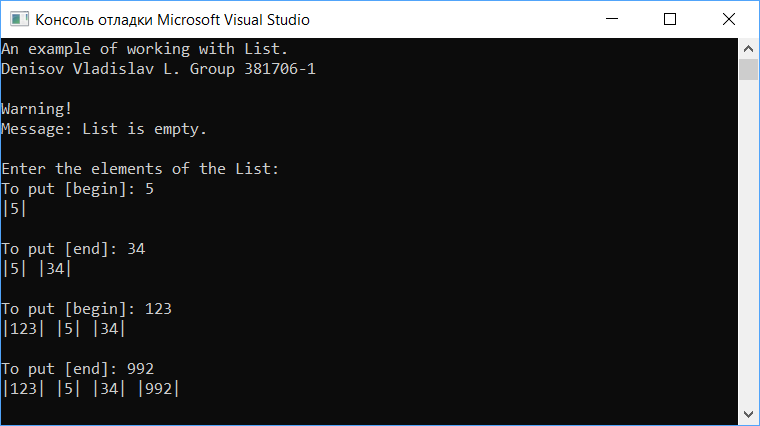
* добавления в начало нового звена списка,
* добавления нового звена в указанную позицию,
* добавления в конец нового звена списка,
* извлечения звена из начала списка (с удалением),
* извлечения звена из указанной позиции (с удалением),
* извлечения звена из конца списка (с удалением),
* проверка списка на полноту/пустоту.

Программное решение будет выглядеть следующим образом:

1. Класс списка – TList.
2. Вспомогательный класс TElem – звено (элемент) списка.
3. Класс для обработки исключений – TException, которые могут возникнуть при выполнении различных операций.
4. Программа, демонстрирующая работу класса TList.
5. Набор автоматических тестов с использованием Google C++ Testing Framework.

# Руководство пользователя

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 1 Пример работы демонстрационной программы.*

Затем выполняется изъятие элементов из конца и начала списка. А также предпринимается попытка извлечь элемент, когда список уже пуст.

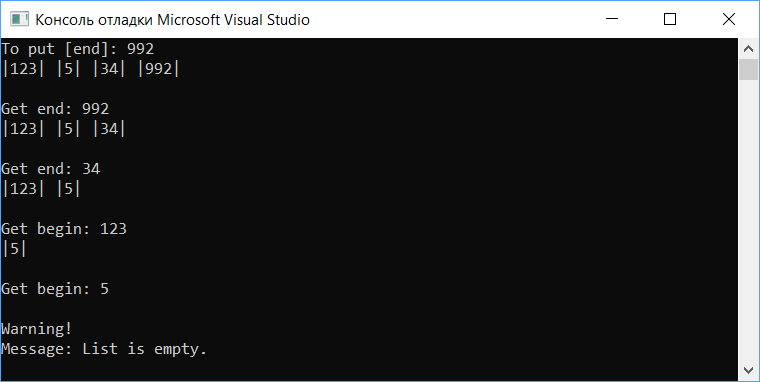


Рисунок 2 Пример работы демонстрационной программы

После этих дейсвий демонстрируется добавление элементов в список, который стал пустым в результате работы с ним. А также выполняется копирование списка и отображение соответствующих старому и новому спискам адресов в памяти.

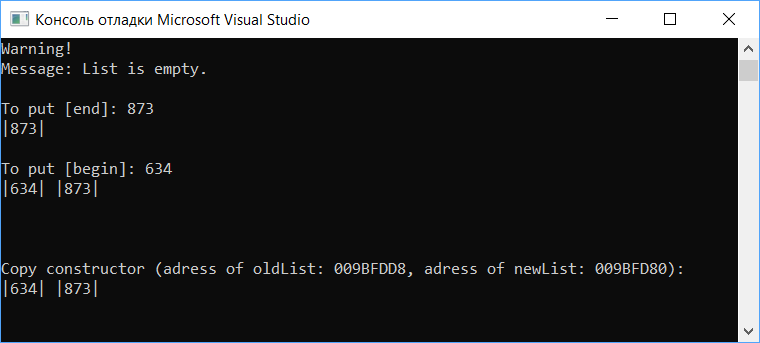


Рисунок 3 Пример работы демонстрационной программы

# 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 10 тестов, описанных в файле *ListTest.cpp* и разработанных с помощью использования Google C++ Testing Framework.
* Модуль ExceptionLib – библиотека, позволяющая создавать собственные исключения.

## Описание структур данных

#### Класс TException – класс исключений.

Класс содержит 1 **private** поле *std::string msg* – переменная, хранящая сообщение об ошибке в виде строки.

И содержит 2 **public** элемента:

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

*void Print()* – метод отображения ошибки на консоль.

#### Класс TElem – класс звена списка

Рассмотрим класс *TElem* подробно.

template <class T> class TElem {…} – класс звена списка является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

T data – хранилище данных в списке.

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

**Элементы класса, объявленные со спецификатором public:**

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

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

virtual ~TElem() – деструктор.

TElem\* GetNext() – получить указатель на следующее звено.

T GetData() – получить значение из звена списка.

void Set(T e) – установить значение в звено списка.

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

#### Класс TList – класс списка.

Рассмотрим класс *TList* подробно.

template <class T> class TList {…} – класс списка является шаблонным, что позволяет использовать его для хранения данных любого типа.

**Элементы класса, объявленные со спецификатором protected:**

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

**Элементы класса, объявленные со спецификатором public:**

TList() – конструктор по умолчанию.

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

virtual ~TList() – деструктор.

void PutBegin(T a) – положить элемент в начало списка.

void Put(int \_n, T a) – положить элемент в указанную позицию.

void PutEnd(T a) – положить элемент в конец списка.

T GetBegin() – взять элемент с удалением из начала.

T Get() – взять элемент из указанной позиции.

T GetEnd() – взять элемент списка с удалением из конца.

bool IsEmpty() – метод проверки списка на пустоту.

void Print() – метод вывода списка на консоль.

## Описание алгоритмов

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

**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 список, 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, путем изменения значения *a.* 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 нуля, 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.

## Сравнение производительности.

|  |  |  |
| --- | --- | --- |
| 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 | 16 |

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

Изучив результаты эксперимента, можно сделать вывод о том, что метод *PutBegin(),* работает значительно быстрее метода *PutEnd()*. Объясняется это тем, что при добавлении элемента в начало списка, к памяти мы обращаемся всего 1 раз, таким образом, сложность этого метода равняется O(1). При добавлении элемента в конец списка, мы должны пройти по всем n элементам 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 результате лабораторной работы была разработана библиотека, реализующая шаблонный класс списка. Она позволяет создать объект класса списка и выполнить с ним такие операции, как:

* добавления в начало нового звена списка,
* добавления в указанную позицию нового звена,
* добавления в конец нового звена списка,
* извлечения звена из начала списка (с удалением),
* извлечение звена из указанной позиции (с удалением),
* извлечения звена из конца списка (с удалением),
* проверка списка на полноту/пустоту.

Были разработаны и доведены до успешного выполнения тесты, созданные для данного программного проекта с использованием Google C++ Testing Framework.

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

# Литература

1. Википедия: свободная электронная энциклопедия: на русском языке [Электронный ресурс] // URL: https://ru.wikipedia.org/wiki/Список\_(информатика) (дата обращения: 17.12.2018)