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**

оставь надежды всяк сие читающий (карин т.а.)

**Отчет по учебной практике**

**«**Разработка общего представления линейного списка**»**

**Выполнил:**

студент группы 381706­1

Карин Тимофей Андреевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Проверил:**

ассистент каф. МОСТ ИИТММ

Лебедев Илья Геннадьевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018.

Содержание

[1. Введение 3](#_Toc536219868)

[3. Руководство пользователя 5](#_Toc536219869)

[4. Руководство программиста 6](#_Toc536219870)

[4.1. Описание структуры программы 6](#_Toc536219871)

[4.2 Описание структур данных 6](#_Toc536219872)

[4.3 Описание алгоритмов 6](#_Toc536219873)

[5. Заключение 7](#_Toc536219874)

[6. Список литературы 8](#_Toc536219875)

1. Введение

Список – абстрактная структура данных, в которой элементы имеют явные связи. Каждый элемент списка кроме самих данных хранит указатель на следующий элемент. Это означает, что элементы могут храниться не в смежных областях памяти. Как же получить нужный элемент в списке. Нужно каждый раз начиная с первого элемента передвигаться по указателю к следующему, пока не дойдём до нужного.

Для чего нужен список? Основная проблема массивов в том, что он имеет ограниченный размер. Для того, чтобы увеличить массив нужно создавать массив большего размера и удалять предыдущий. Разумеется, это не рационально. Список, в отличие от массива, не имеет фиксированной длины, его размер ограничен лишь количеством свободной общей памяти.

Списки могут быть односвязным (когда элемент кроме данных хранит только указатель на следующий элемент списка), и двусвязным (когда элемент помимо прочего хранит указатель на предыдущий элемент списка). В данной работе мы остановимся на односвязных списках.

2. Постановка задачи

Целью данной лабораторной работы является написание библиотеки для работы со списками. Для этого необходимо выполнить следующие задачи:

* Создать класс, объекты которого будут являться элементами списка;
* Создать класс, предназначенный для хранения списка и работы с ним;
* Написать пример использования библиотеки;
* Проверить на работоспособность все методы классов.

3. Руководство пользователя

4. Руководство программиста

4.1. Описание структуры программы

Программа состоит из 3 основных модулей:

* *ListLib –* библиотека для работы со списками. Состоит из файлов *Elem.h* и *List.h*, в которых соответственно находятся шаблонные классы элемента списка (класс *TElem*) и самого списка (класс *TList*).
* *List –* пример использования библиотеки.
* *ListTest –* тесты для проверки работоспособности библиотеки.

4.2 Описание структур данных

Рассмотрим класс *TElem*. Класс шаблонный, *T –* шаблонный тип данных, присваиваемый элементам, которые будут храниться в списке (то есть без учёта указателя на следующий элемент). Рассмотрим поля и методы:

Со спецификатором доступа protected:

* *T A* – данные, которые пользователь будет хранить в списке;
* *TElem <T>\* next* – указатель на следующий элемент списка;

Со спецификатором доступа public:

* *TElem (T \_A, TElem <T>\* n = 0) –* конструктор;
* *TElem (TElem<T> &B) –* конструктор копирования;
* *T Get() –* функция получение пользовательских данных (возвращает значение поля *A*);
* *TElem\* GetNext() –* возвращает указатель на следующий элемент (поле *next*);
* *void Set(T p) –* ввод данных (присваивает полю *A* значение *p*);
* *void SetNext(TElem<T>\* n) –* устанавливает указатель на следующий элемент (присваивает полю *next* значение *n*).

Рассмотрим шаблонный класс *TList.* Рассмотрим поля и методы:

Со спецификатором доступа protected:

* *TElem<T>\* begin –* указатель на начало (на первый элемент списка);

Со спецификатором доступа public:

* *TList()* – конструктор;
* *TList(TList<T> &A) –* конструктор копирования;
* *~TList() –* деструктор;
* *void PutBegin(T A) –* добавить элемент в начало;
* *void PutEnd(T A) –* добавить элемент в конец;
* *void PutElem(T A, int n) –* добавить элемент *A* на позицию *n* (нумерация элементов начинается с 0);
* *T GetBegin() –* получить первый элемент;
* *T GetEnd() –* получить последний элемент.
* *T GetElem(int n) –* получить элемент с порядковым номером *n* (нумерация элементов начинается с 0).

4.3 Описание алгоритмов

Добавление элемента в начало списка:

Если список пустой, то создаём элемент с указателем равным 0. Если список не пустой, то создаём элемент с указателем на элемент, который был первым, то есть равный полю *begin.* В любом случае, после этого полю *begin* нужно присвоить адрес добавленного элемента.

Добавление в конец:

Если список пуст, то добавление в конец ничем не отличается от добавления в начало. Если в списке есть элементы, то создаём новый элемент, кладём в него нужные данные, указатель приравниваем к 0 (т.к. это будет последний элемент списка). Затем от первого элемента списка последовательно доходим до последнего и меняем его указатель на только что созданный элемент.

Добавление в любую позицию:

Прежде всего проверяем корректность полученного параметра (чтобы он не был отрицательным). Затем, если позиция равна 0, то вызываем функцию добавления в начало. Если позиция отлична от нуля, то поочерёдно доходим до элемента, за которым должен следовать добавляемый. Естественно, все эти элементы должны существовать, поэтому если на каком-то этапе мы получим нулевой указатель на элемент (на текущий, не на следующий), то вызываем исключение. У полученного элемента нужно изменить его указатель на следующий элемент: текущий указатель «отдать» добавляемому элементу, а на его место поместить указатель на добавленный элемент.

Получение первого элемента:

Проверяем на пустоту. Берём первый элемент, сохраняем его данные. В качестве первого элемента устанавливаем следующий за удаляемым элемент. Сам элемент теперь можно удалять, а данные возвращать.

Получение последнего:

Проверяем на пустоту. Теперь нужно получить предпоследний элемент, поэтому случай, когда в списке один элемент, нужно разбирать отдельно. Предпоследний элемент станет последним, поэтому его указатель нужно обнулить (не забыв сохранить в другое место). Переходим к последнему. Данные сохраняем, сам элемент удаляем, данные возвращаем. Если в списке один элемент, то его данные сохраняем, его удаляем, указатель на первый элемент списка обнуляем, данные возвращаем.

Получение любого:

Проверяем на пустоту, проверяем параметр на не отрицательность. Если параметр равен нулю, то вызывается функция получения первого элемента. Если параметр больше нуля, то последовательно переходим к элементу, который находится перед удаляемым. При этом, элементы, конечно же, должны существовать, поэтому, если на каком-то этапе указатель на следующий элемент (в данном случае не на сам элемент, а даже и на следующий) равен нулю, вызывается исключение. Теперь у полученного элемента нужно изменить указатель: ставим на его место указатель на элемент, следующий за удаляемым. Теперь можно удалять нужный элемент и возвращать его данные.

5. Заключение

В ходе проделанной работы удалось достичь следующих результатов:

* Разработана библиотека для простой работы с односвязными списками;
* Написан пример использования программы;
* Обработаны основные исключительные ситуации;
* Проверена работоспособность библиотеки с помощью тестов фреймворка Google Test.

6. Список литературы

1. Ахо Альфред В, Хопкрофт Джон Э и Ульман Джеффри Д Структуры данных и алгоритмы [Книга]. - [б.м.] : Вильямс, 2003.
2. Лафоре Роберт Структуры данных и алгоритмы в Java [Книга]. - СПб : Питер, 2013. - 2 : стр. 704.
3. Павловская Т. А. C/C++ Программирование на языке высокого уровня [Книга]. - СПб : Питер, 2003.
4. Страуструп Бьерн Язык программирования C++ Бином, 2004.