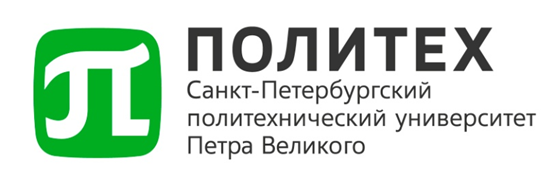
**Санкт-Петербургский политехнический университет Петра Великого**

**Институт компьютерных наук и технологий**

**Высшая школа программной инженерии**

****

**ЛАБОРАТОРНАЯ РАБОТА**

**Алгоритмы работы со словарями**

по дисциплине «**Практикум по программированию**»

Выполнил

студент гр.3530904/00004  Мурзаканов И.М.

Руководитель Павлов Е.А.

«\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2021

Санкт-Петербург

2021

# Содержание

[Содержание 2](#_Toc73196955)

[Введение. Общая постановка задачи: 3](#_Toc73196956)

[Основная часть работы 3](#_Toc73196957)

[Описание алгоритма решения и используемых структур данных 3](#_Toc73196958)

[Анализ алгоритма 5](#_Toc73196959)

[*Поиск элемента, отсутствующего в таблице* 5](#_Toc73196960)

[*Поиск элемента, существующего в таблице* 5](#_Toc73196961)

[*Работа с вектором* 5](#_Toc73196962)

[Описание спецификации программы 6](#_Toc73196963)

[*Список главных терминов* 6](#_Toc73196964)

[*Спецификация* 6](#_Toc73196965)

[Описание программы 6](#_Toc73196966)

[*Форматы данных* 6](#_Toc73196967)

[*Структура программы* 7](#_Toc73196968)

[Заключение 10](#_Toc73196969)

[Список использованных источников 11](#_Toc73196970)

[Приложение 1 12](#_Toc73196971)

[Приложение 2 13](#_Toc73196972)

# Введение. Общая постановка задачи:

**Тема: Алгоритм работы со словарями**

1. Для разрабатываемого словаря реализовать основные операции:
   * INSERT (ключ, значение) – добавить запись с указанным ключом и значением
   * SEARCH (ключ)- найти запись с указанным ключом
   * DELETE (ключ)- удалить запись с указанным ключом
2. Предусмотреть обработку и инициализацию исключительных ситуаций, связанных, например, с проверкой значения полей перед инициализацией и присваиванием.
3. Программа должна быть написана в соответствии со стилем программирования: C++ Programming Style Guidelines (<http://geosoft.no/development/cppstyle.html>).
4. Тесты должны учитывать как допустимые, так и не допустимые последовательности входных данных.

**Вариант 1.1.5.**

**Англо-русский словарь. Хеш - таблица**

Ячейка хеш-таблицы должна содержать:

* Ключ – английское слово,
* Информационная часть – ссылка на список, содержащий переводы английского слова, отсортированные по алфавиту (переводов слова может быть несколько).

# Основная часть работы

## Описание алгоритма решения и используемых структур данных

Чтобы реализовать данный алгоритм, было принято решение использовать собственные структуры данных. Такой выбор дает право создавать структуры, похожие на стандартные, но с особыми свойствами, которые помогают разработать программу для выполнения конкретных задачей, делают ее более ориентированной.

Мой Англо-русский словарь построен на основе структур, которые взаимодействуют между собой. К ним относится:

1. Двусвязный список (class DoubleLinkedList)
2. Слово (class Word)
3. Словарь на основе хеш-таблицы (class HashTable)

Хеш-таблица обобщает обычный массив, который содержит в себе требуемый набор данных. В ячейке таблицы хранится ключ и указатель на список переводов, которые сортируются по алфавиту. В качестве ключа и переводов выступают строки. Возможность индексации обеспечивает быстрый доступ к элементам хеш-таблицы, которые располагаются в определенном порядке. Их порядок непосредственно определяется хеш-функцией. В общем понимании хеш-функция – математическое уравнение или алгоритм, предназначенный и позволяющий превратить входящий информационный поток неограниченного объема в лаконичную строчку с заданным количеством парных символов. Наша хеш-функция будет вычислять место в массиве для некоторого английского термина по первой букве. Так как в английском языке нет таких букв, с которых бы не могло начаться слово, то мы гарантированно сможем внести или найти, удалить слово (если таковое имеется в таблице). Чтобы осуществлять поиск, вставку или удаление будем использовать хеш-функцию, вычисляющую хеш-значение из кодов ASCII символов слова. Используемая форма имеет следующий вид:

Где hash-хеш-значение; englishWord[i] - код i-ой буквы из таблицы ASCII; tableSize\_ – количество ячеек хеш-таблицы.

После вычисления хеш-значения и успешного поиска происходит непосредственное взаимодействие с элементом хеш-таблицы и со списком переводов слов.

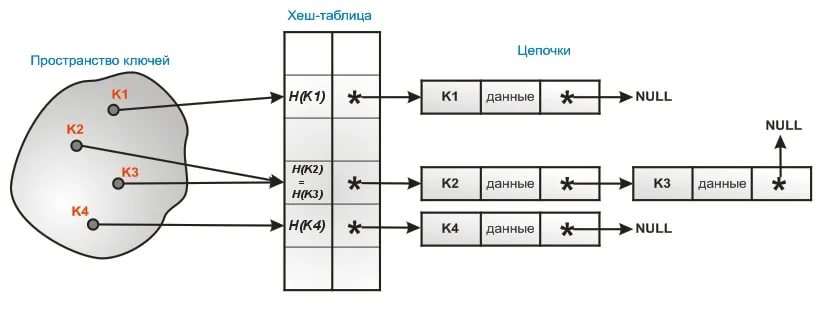


Рисунок 1

## Анализ алгоритма

Рассмотрим работу алгоритма хеширования открытой адресацией со списком и оценим его сложность.

### *Поиск элемента, отсутствующего в таблице*

Алгоритм хеширования имеет сложность О(n), где n – длина нашего слова. В среднем случае слово настолько короткое, что можно считать O(n) = O(1), т.е. алгоритм хеширования имеет постоянную временную сложность. Если в нашей таблице количество коллизий не столько велико, то поиск элемента, отсутствующего в таблице будет равно сложности алгоритма хеширования О(1). В случае коллизии сложность будет зависеть от поиска элемента в списке слов – О(k), где k – длина списка.

### *Поиск элемента, существующего в таблице*

Аналогично с предыдущим случаем, будем считать, что сложность алгоритма хеширования О(1). Тогда в случае, если у нас нет коллизий, сложность алгоритма будет О(1), так как нам придется только проверить, не пуст ли у нас список. Если же у нас произошла коллизия(и не одна), нам придется сравнить текущее слово со всеми английскими словами в списке из n элементов. В таком случае сложность поиска элемента будет равняться O(n).

### *Работа со списком*

Конечно, работа нашего алгоритма также связана с работой с вектором, где хранятся переводы слова. Однако его временная сложность не так велика, потому что при работе с вектором добавление перевода выполняется с конца. Временная сложность такой операции O(1). Что же касается сортировки вектора, то она происходит по первым буквам слова в их целочисленном представлении. Сложность сортировки составляет в среднем случае O(n log n), где n – количество переводов.

## Описание спецификации программы

### *Список главных терминов*

Чтобы подробно разобраться в самой спецификации, необходимо знать определение следующих терминов:

1. Английское слово — это термин, который записывается на английском языке.
2. Русское слово — это термин, который записывается на русском языке.
3. Слово – это английский термин (английское слово) и все его переводы (русские слова), которые записываются в одну строку в определенном формате.
4. Код команды – это число, определяющее поведение словаря.

### *Спецификация*

1. Чтобы начать работу со словарем, пользователь должен ввести несколько слов и их переводов. Для завершения работы пользователю необходимо ввести 0.
2. Словарь должен предоставлять пользователю следующий интерфейс:

* Вставка слова в словарь
* Поиск слова в словаре
* Поиск переводов слова в словаре
* Удаление слова из словаря

1. Чтобы использовать интерфейс программы, пользователь должен ввести верный код команды.

* 1 - внести слово в словарь
* 2 - удалить слово из словаря
* 3 - получить перевод слова
* 4 - найти слово
* 0 - завершить программу

В случае неверного ввода кода команды, т.е. был введен другой код, выдается сообщение о неверном вводе.

В случае ошибки формата ввода кода команды, словарь завершает свою работу.

1. Чтобы работа со словарем проходила успешно, необходимо, чтобы формат вводимых слов и терминов был правильной. В противном случае будут сгенерированы исключения.
2. Чтобы осуществить поиск нужных слов, переводов или чтобы удалить слово, достаточно просто ввести английский термин.
3. Чтобы осуществить внесение слова в словарь, необходимо записать его в нужном формате.
4. Программа абсолютно не чувствительна к регистрам.
5. Словарь игнорирует все символы, кроме символов, обеспечивающих нужный формат, а также английских и русских букв.
6. Словарь не следит за правилами орфографии и пунктуации.
7. Переполнение словаря невозможно, так как решение коллизий производится с помощью метода цепочек.
8. При внесении слова, не содержащего перевод, в словарь должно быть сгенерировано исключение.
9. При поиске/удалении слова, которое отсутствует в словаре или которое было введено с ошибкой генерируется исключение.
10. Программа должна сортировать все переводы термина по алфавиту.
11. Пользователь сам должен указать размер словаря.

## Описание программы

### *Форматы данных*

Чтобы работать со словарем, необходимо придерживаться некоторого формата входных данных.

Для вставки слова в словарь необходимо, чтобы оно было записано в формате: <Английское слово> = <Перевод 1>|<Перевод 2>…|<Перевод n>

Пример: help=помогать|помощь|поддержка

Для поиска/удаления слова из словаря необходимо просто ввести слово правильно с точки зрения орфографии.

Пример: help

При получении перевода пользователь получает русское слово в стандартном формате.

Пример:

Помогать

Помощь

Поддержка

При поиске слова пользователь получает уведомление о том, содержится ли данное слово в словаре.

Пример: help

Вывод:

Dictionary contains this word

ИЛИ

Dictionary does not contain this word

В случае возникновения ошибки словарь должен уведомить об этом пользователя с помощью сообщения.

В начале работы пользователь видит следующие сообщение:

Enter the command number

1 - insert the word in dictionary

2 - remove the word from dictionary

3 - show the translation

4 - find the word

0 - close the program

После набора команды программа также взаимодействует с пользователем.

Пример: «Input English word and translations like this»

### *Структура программы*

template < class T >

class DoubleLinkedList

{

private:

template < typename C >

struct Node // -***узел списка***

{

C item\_; // ключ узла

Node\* next\_; // указатель на следующий элемент списка

Node\* prev\_; // указатель на предшествующий элемент списка

// Конструктор для создания нового элемента списка.

Node(C item, Node\* next = nullptr, Node\* prev = nullptr):

item\_(item),

next\_(next),

prev\_(prev)

{}

};

int count\_; // счетчик числа элементов

Node< T >\* head\_; // первый элемент списка

Node< T >\* tail\_; // последний элемент списка

// Доступ к головному узлу списка

Node< T >\* head() const

{

return head\_;

}

// Доступ к хвостовому узлу списка

Node< T >\* tail() const

{

return tail\_;

}

// Всавить сформированный узел в хвост списка

void insertTail(Node< T >\* x);

// Вставить сформированный узел в начало списка

void insertHead(Node< T >\* x); // (int item);

// Удаление заданного узла

void deleteNode(Node< T >\* x);

// Поиск узла (адрес) с заданным значением

Node< T >\* searchNode(T item);

// Замена информации узла на новое

Node< T >\* replaceNode(Node< T >\* x, T item);

public:

// Конструктор "по умолчанию" - создание пустого списка

DoubleLinkedList();

// количество элементов списка

int count() const;

// Доступ к информации головного узла списка

T headItem() const;

T& headItem();

// Доступ к информации хвостового узла списка

T tailItem() const;

T& tailItem();

// Вставить элемент в голову списка

void insertHead(T item);

// Вставить элемент в хвост списка

void insertTail(T item);

// Удалить элемент с головы списка

bool deleteHead();

// Удалить элемент из хвоста списка

bool deleteTail();

// Удаление узла с заданным значением

bool deleteItem(const T item);

// Поиск записи с заданным значением

Node< T >\* searchItem(T item);

// Замена информации узла на новое

bool replaceItem(T itemOld, T itemNew);

// Вывод элементов списка в текстовом виде в стандартный выходной поток

void outAll();

//оператор копирующего присваивания

DoubleLinkedList& operator =(const DoubleLinkedList& src);

//swap

void swap(DoubleLinkedList& src) noexcept;

//перегруженный оператор вывода

//friend std::ostream& operator<< (std::ostream& out, DoubleLinkedList& src);

//Проверка на empty

bool empty() const;

// Деструктор списка

virtual ~DoubleLinkedList();

};

class HashTable

{

public:

HashTable(size\_t size); - конструктор с размером словаря

void insert(std::istream& in); - метод вставки нового перевода в словарь

void search(std::istream& in, std::ostream& out); - метод поиска слова в словаре

void deleteWord(std::istream& in); - удаление перевода из словаря

void show(std::istream& in , std::ostream& out); - вывод списка перевода для слова

private:

std::vector< DoubleLinkedList< Word > > array\_; - вектор списков(для разрешения коллизий методом цепочек) переводов

size\_t tableSize\_; - размер словаря

int numOfCollsions\_; - количество коллизий

int getHash(const std::string& englishWord); - функция получения хеша

};

bool checkWordIsCorrect(const std::string& word); - проверка на правильность введения слова

void mergeVectors(std::vector< std::string >& v1, std::vector< std::string >& v2); - функция слияния двух векторов

void doStandartString(std::string& str); - функция приведения слова в стандартный вид

void myToLower(std::string& str); - функция перевода всех букв слова в нижний регистр

class Word

{

public:

Word(); - конструктор по умолчанию для переводов

Word(std::string englishWord); - конструктор с английским словом

void insertWord(std::istream& in); - метод вставки перевода

std::string getEnglishWord(); - метод возврата английского слова

std::vector< std::string >& getVect(); - метод получения вектора переводов

void show(); - метод вывода переводов

void clear(); - метод удаления переводов

bool operator==(const Word& rhs); - оператор сравнения переводов

bool operator!=(const Word& rhs);

private:

std::string englishWord\_; - английское слово

std::vector< std::string > translations\_; - вектор, хранящий переводы английского слова

};

# Заключение

В результате выполнения задания были получены знания о хеш-таблицах, о том, в чем отличие хеш-таблиц от массивов, каков смысл в применении хеш-таблиц. Кроме того, была оценена алгоритмическая сложность изученного алгоритма, которая показывает особенности хеш-таблицы. В результате, мной была реализована собственная простейшая хеш-таблица для хранения английских слов и списка переводов, которые отсортированы по алфавиту.

# Список использованных источников

1. Кормен Т. и др. Алгоритмы: построение и анализ: М. [и др.]: Вильямс, 2011.
2. Седжвик Р., Алгоритмы на C++: Анализ, структуры данных, сортировка, поиск, алгоритмы на графах: М.: Вильямс, 2016. – 1056 с.
3. Что такое хеш-таблицы и как они работают. URL: https://ruhighload.com/Что+такое+хеш-таблицы+и+как+они+работают
4. Структуры данных и хеширование. Хеш таблицы. URL: http://algolist.manual.ru/ds/s\_has.php

# Приложение 1

Первая таблица – тест-план для команды INSERT

|  |  |  |
| --- | --- | --- |
| Входные данные | Сгенерированное исключение | Вывод |
| hold on=держаться|продержаться |  | Держаться продержаться держать |
| hold on = держаться|продержаться |  | Держаться продержаться держать |
| = держаться|продержаться | std::invalid\_argument | Invalid input |
| hold onдержаться|продержаться | std::invalid\_argument | Invalid input |
| hold on= | std::invalid\_argument | Invalid input |

Вторая таблица – тест-план для команды DELETE(remove)

|  |  |  |
| --- | --- | --- |
| Входные данные | Сгенерированное исключение | Вывод |
| Слово, не содержащееся в словаре | std::runtime\_error | Dictionary does not contain this word |
| Слово, содержащееся в словаре  hold on |  | Successful deletion |

Третья таблица – тест-план для команды show

|  |  |  |
| --- | --- | --- |
| Входные данные | Сгенерированное исключение | Вывод |
| Слово, не содержащееся в словаре | std::runtime\_error | Dictionary does not contain this word |
| Слово, содержащееся в словаре  hold on |  | Держаться продержаться держать |

Четвертая таблица – тест-план для команды SEARCH

|  |  |  |
| --- | --- | --- |
| Входные данные | Сгенерированное исключение | Вывод |
| Слово, не содержащееся в словаре |  | Dictionary does not contain this word |
| Слово, содержащееся в словаре  hold on |  | Dictionary contain this word |

# Приложение 2

Ниже приведен полный листинг кода программы

DoubleLinkedList.hpp

#ifndef DOUBLE\_LINKED\_LIST\_HPP

#define DOUBLE\_LINKED\_LIST\_HPP

#include<iostream>

// DoubleLinkedList.h - Дважды связный список целых чисел

//

template **<** class T **>**

class DoubleLinkedList

**{**

private**:**

// Тип Node используется для описания элемента списка, связанного со

// следующим с помощью поля next\_ и предшествующим с помощью поле prev\_

template **<** typename C **>**

struct Node // может использоваться только в классе DoubleLinkedList

**{**

C item\_**;** // значение элемента списка

Node**\*** next\_**;** // указатель на следующий элемент списка

Node**\*** prev\_**;** // указатель на предшествующий элемент списка

// Конструктор для создания нового элемента списка.

Node**(**C item**,** Node**\*** next **=** **nullptr,** Node**\*** prev **=** **nullptr):**

item\_**(**item**),**

next\_**(**next**),**

prev\_**(**prev**)**

**{}**

**};**

int count\_**;** // счетчик числа элементов

Node**<** T **>\*** head\_**;** // первый элемент списка

Node**<** T **>\*** tail\_**;** // последний элемент списка

// Доступ к головному узлу списка

Node**<** T **>\*** head**()** const

**{**

**return** head\_**;**

**}**

// Доступ к хвостовому узлу списка

Node**<** T **>\*** tail**()** const

**{**

**return** tail\_**;**

**}**

// Всавить сформированный узел в хвост списка

void insertTail**(**Node**<** T **>\*** x**);**

// Вставить сформированный узел в начало списка

void insertHead**(**Node**<** T **>\*** x**);** // (int item);

// Удаление заданного узла

void deleteNode**(**Node**<** T **>\*** x**);**

// Поиск узла (адрес) с заданным значением

Node**<** T **>\*** searchNode**(**T item**);**

// Замена информации узла на новое

Node**<** T **>\*** replaceNode**(**Node**<** T **>\*** x**,** T item**);**

public**:**

// Конструктор "по умолчанию" - создание пустого списка

DoubleLinkedList**():**

count\_**(**0**),**

head\_**(nullptr),**

tail\_**(nullptr)**

**{}**

// Конструктор "копирования" – создание копии имеющегося списка

DoubleLinkedList**(**const DoubleLinkedList**&** src**);**

// Конструктор перемещения

DoubleLinkedList**(**DoubleLinkedList**&&** src**)**noexcept**;**

//

DoubleLinkedList**&** **operator=(**DoubleLinkedList**&&** x**)** noexcept**;**

// количество элементов списка

int count**()** const**;**

// Доступ к информации головного узла списка

T headItem**()** const**;**

T**&** headItem**();**

// Доступ к информации хвостового узла списка

T tailItem**()** const**;**

T**&** tailItem**();**

// Вставить элемент в голову списка

void insertHead**(**T item**);**

// Вставить элемент в хвост списка

void insertTail**(**T item**);**

// Удалить элемент с головы списка

bool deleteHead**();**

// Удалить элемент из хвоста списка

bool deleteTail**();**

// Удаление узла с заданным значением

bool deleteItem**(**const T item**);**

// Поиск записи с заданным значением

Node**<** T **>\*** searchItem**(**T item**);**

// Замена информации узла на новое

bool replaceItem**(**T itemOld**,** T itemNew**);**

// Вывод элементов списка в текстовом виде в стандартный выходной поток

void outAll**();**

//оператор копирующего присваивания

DoubleLinkedList**&** **operator** **=(**const DoubleLinkedList**&** src**);**

//swap

void swap**(**DoubleLinkedList**&** src**)** noexcept**;**

//перегруженный оператор вывода

//friend std::ostream& operator<< (std::ostream& out, DoubleLinkedList& src);

//Проверка на empty

bool empty**()** const**;**

// Деструктор списка

virtual **~**DoubleLinkedList**();**

**};**

template **<** typename T **>**

bool DoubleLinkedList**<** T **>::**empty**()** const

**{**

**return** **!**count\_**;**

**}**

template**<** typename T **>**

DoubleLinkedList**<** T **>::**DoubleLinkedList**(**const DoubleLinkedList**&** src**)** **:**

head\_**(nullptr),**

tail\_**(nullptr),**

count\_**(**0**)**

**{**

Node**<** T **>\*** currentNode **=** src**.**head\_**;**

**while** **(**currentNode **!=** **nullptr)**

**{**

**this->**insertTail**(**currentNode**->**item\_**);**

currentNode **=** currentNode**->**next\_**;**

**}**

**}**

// Вставить сформированный узел в хвост списка

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**insertTail**(**Node**<** T **>\*** x**)**

**{**

x**->**prev\_ **=** tail\_**;**

**if** **(**head\_ **!=** **nullptr)** **{**

tail\_**->**next\_ **=** x**;**

**}**

**else**

**{**

head\_ **=** x**;**

**}**

tail\_ **=** x**;**

count\_**++;**

**}**

// Вставить сформированный узел в начало списка

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**insertHead**(**Node**<** T **>\*** x**)**

**{**

x**->**next\_ **=** head\_**;**

**if** **(**head\_ **!=** **nullptr)** **{**

// список был НЕ пуст – новый элемент будет и первым, и последним

head\_**->**prev\_ **=** x**;**

**}**

**else** **{**

// список был пуст – новый элемент будет и первым, и последним

tail\_ **=** x**;**

**}**

head\_ **=** x**;**

count\_**++;** // число элементов списка увеличилось

**}**

// Удаление заданного узла

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**deleteNode**(**Node**<** T **>\*** x**)**

**{**

**if** **(**x **==** **nullptr)** **{**

**throw** **(**"DoubleLinkedList::deleteNode - неверно задан адрес удаляемого узла"**);**

**}**

**if** **(**x**->**prev\_ **!=** **nullptr)**

**{**

// удаляется НЕ голова списка

**(**x**->**prev\_**)->**next\_ **=** x**->**next\_**;**

**}**

**else**

**{**

// удаляется голова списка, второй элемент становится первым

head\_ **=** x**->**next\_**;**

**}**

**if** **(**x**->**next\_ **!=** **nullptr)** **{**

// удаляется НЕ хвост

**(**x**->**next\_**)->**prev\_ **=** x**->**prev\_**;**

**}**

**else** **{**

// удаляется хвост

tail\_ **=** x**->**prev\_**;**

**}**

**delete** x**;** //

count\_**--;** // число элементов списка уменьшилось

**}**

// Поиск узла (адрес) с заданным значением

template **<** typename T **>**

DoubleLinkedList**<** T **>::**Node**<** T **>\*** DoubleLinkedList**<** T **>::**searchNode**(**T item**)**

**{**

Node**<** T **>\*** x **=** head\_**;**

**while** **(**x **!=** **nullptr** **&&** x**->**item\_ **!=** item**)** **{**

x **=** x**->**next\_**;**

**}**

**return** x**;**

**}**

// Замена информации узла на новое

template **<** typename T **>**

DoubleLinkedList**<** T **>::**Node**<** T **>\*** DoubleLinkedList**<** T **>::**replaceNode**(**DoubleLinkedList**<** T **>::**Node**<** T **>\*** x**,** T item**)**

**{**

x**->**item\_ **=** item**;**

**return** x**;**

**}**

//количество элементов списка

template **<** typename T **>**

int DoubleLinkedList**<** T **>::**count**()**const **{** **return** count\_**;** **}**

// Доступ к информации головного узла списка

template **<** typename T **>**

T DoubleLinkedList**<** T **>::**headItem**()** const

**{**

**if** **(**head\_ **!=** **nullptr)** **{**

**return** head\_**->**item\_**;**

**}**

**throw** **(**"headItem - список пуст!"**);**

**}**

template **<** typename T **>**

T**&** DoubleLinkedList**<** T **>::**headItem**()**

**{**

**if** **(**head\_ **!=** **nullptr)** **{**

**return** head\_**->**item\_**;**

**}**

**throw** **(**"headItem - список пуст!"**);**

**}**

// Доступ к информации хвостового узла списка

template **<** typename T **>**

T DoubleLinkedList**<** T **>::**tailItem**()** const

**{**

**if** **(**tail\_ **!=** **nullptr)** **{**

**return** tail\_**->**item\_**;**

**}**

**throw** **(**"tailItem - список пуст!"**);**

**}**

template **<** typename T **>**

T**&** DoubleLinkedList**<** T **>::**tailItem**()**

**{**

**if** **(**tail\_ **!=** **nullptr)** **{**

**return** tail\_**->**item\_**;**

**}**

**throw** **(**"tailItem - список пуст!"**);**

**}**

// Вставить элемент в голову списка

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**insertHead**(**T item**)**

**{** // создаем новый элемент списка и добавляем в голову

insertHead**(new** Node**<** T **>(**item**));**

**}**

// Вставить элемент в хвост списка

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**insertTail**(**T item**)**

**{** // создаем новый элемент списка и добавляем в хвост

insertTail**(new** Node**<** T **>(**item**));**

**}**

// Удалить элемент с головы списка

template **<** typename T **>**

bool DoubleLinkedList**<** T **>::**deleteHead**()**

**{**

**if** **(**head\_ **==** **nullptr)** **{**

**return** 0**;** // список пуст, удалений нет

**}**

deleteNode**(**head\_**);**

**return** 1**;** // список был НЕ пуст, удаление головы

**}**

// Удалить элемент из хвоста списка

template **<** typename T **>**

bool DoubleLinkedList**<** T **>::**deleteTail**()**

**{**

**if** **(**tail\_ **==** **nullptr)** **{**

**return** **false;**

**}**

deleteNode**(**tail\_**);**

**return** 1**;**

**}**

// Удаление узла с заданным значением

template **<** typename T **>**

bool DoubleLinkedList**<** T **>::**deleteItem**(**const T item**)**

**{**

bool flag **=** 0**;**

Node**<** T **>\*** temp **=** head\_**;**

**while** **(**temp **!=** **nullptr)**

**{**

**if** **(**temp**->**item\_ **==** item**)**

**{**

temp **=** temp**->**next\_**;**

deleteNode**(**temp**->**prev\_**);**

flag **=** 1**;**

**}**

**else**

**{**

temp **=** temp**->**next\_**;**

**}**

**}**

**return** flag**;**

**}**

// Поиск записи с заданным значением

template **<** typename T **>**

DoubleLinkedList**<** T **>::**Node**<** T **>\*** DoubleLinkedList**<** T **>::**searchItem**(**T item**)**

**{** // возвращаем объект

**return** **(**searchNode**(**item**));**

**}**

// Замена информации узла на новое

template **<** typename T **>**

bool DoubleLinkedList**<** T **>::**replaceItem**(**T itemOld**,** T itemNew**)**

**{**

bool isReplace **=** 0**;**

Node**<** T **>\*** x **=** head\_**;**

**while** **(**x **!=** **nullptr)** **{**

**if** **(**x**->**item\_ **==** itemOld**)**

**{**

x**->**item\_ **=** itemNew**;**

isReplace **=** **true;**

**}**

x **=** x**->**next\_**;**

**}**

**return** isReplace**;**

**}**

// Вывод элементов списка в текстовом виде в стандартный выходной поток

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**outAll**()**

**{**

Node**<** T **>\*** current **=** head\_**;** // Указатель на элемент

**while** **(**current **!=** **nullptr)**

**{**

std**::**cout **<<** current**->**item\_ **<<** ' '**;**

current **=** current**->**next\_**;** // Переход к следующему элементу

**}**

std**::**cout **<<** std**::**endl**;**

**}**

// Деструктор списка

template **<** typename T **>**

DoubleLinkedList**<** T **>::~**DoubleLinkedList**()**

**{**

Node**<** T **>\*** current **=** **nullptr;** // указатель на элемент, подлежащий удалению

Node**<** T **>\*** next **=** head\_**;** // указатель на следующий элемент

**while** **(**next **!=** **nullptr)** **{** // пока есть еще элементы в списке

current **=** next**;**

next **=** next**->**next\_**;** // переход к следующему элементу

**delete** current**;** // освобождение памяти

**}**

**}**

//оператор копирующего присваивания

template **<** typename T **>**

DoubleLinkedList**<** T **>&** DoubleLinkedList**<** T **>::operator=(**const DoubleLinkedList**<** T **>&** src**)**

**{**

**if** **(this** **!=** **&**src**)**

**{**

DoubleLinkedList**(**src**).**swap**(\*this);**

**}**

**return** **\*this;**

**}**

//swap

template **<** typename T **>**

void DoubleLinkedList**<** T **>::**swap**(**DoubleLinkedList**&** src**)** noexcept

**{**

Node**<** T **>\*** temp **=** **nullptr;**

temp **=** head\_**;**

head\_ **=** src**.**head\_**;**

src**.**head\_ **=** temp**;**

temp **=** tail\_**;**

tail\_ **=** src**.**tail\_**;**

src**.**tail\_ **=** temp**;**

temp **=** **nullptr;**

std**::**swap**(**count\_**,** src**.**count\_**);**

**}**

//конструктор перемещения

template **<** typename T **>**

DoubleLinkedList**<** T **>::**DoubleLinkedList**(**DoubleLinkedList**<** T **>&&** src**)** noexcept

**{**

head\_ **=** src**.**head\_**;**

tail\_ **=** src**.**tail\_**;**

count\_ **=** src**.**count\_**;**

src**.**head\_ **=** **nullptr;**

src**.**tail\_ **=** **nullptr;**

src**.**count\_ **=** 0**;**

**}**

//оператор присвающего перемещения

template **<** typename T **>**

DoubleLinkedList**<** T **>&** DoubleLinkedList**<** T **>::operator=(**DoubleLinkedList**<** T **>&&** x**)** noexcept

**{**

**if** **(&**x **==** **this)**

**return** **\*this;**

**while** **(**head\_ **!=** **nullptr)**

**{**

deleteHead**();**

**}**

head\_ **=** x**.**head\_**;**

tail\_ **=** x**.**tail\_**;**

count\_ **=** x**.**count\_**;**

x**.**head\_ **=** **nullptr;**

x**.**tail\_ **=** **nullptr;**

x**.**count\_ **=** 0**;**

**return** **\*this;**

**}**

#endif

hashTable.hpp

#ifndef HASH\_TABLE\_HPP

#define HASH\_TABLE\_HPP

#include <string>

#include <iosfwd>

#include "word.h"

#include "DoubleLinkedList.hpp"

class HashTable

**{**

public**:**

HashTable**(**size\_t size**);**

void insert**(**std**::**istream**&** in**);**

void search**(**std**::**istream**&** in**,** std**::**ostream**&** out**);**

void deleteWord**(**std**::**istream**&** in**);**

void show**(**std**::**istream**&** in**,** std**::**ostream**&** out**);**

private**:**

std**::**vector**<** DoubleLinkedList**<** Word **>** **>** array\_**;**

size\_t tableSize\_**;**

int numOfCollsions\_**;**

int getHash**(**const std**::**string**&** englishWord**);**

**};**

#endif

dictionary.hpp

#ifndef HASH\_TABLE\_HPP

#define HASH\_TABLE\_HPP

#include <string>

#include <iosfwd>

#include "word.h"

#include "DoubleLinkedList.hpp"

class Dictionary

**{**

public**:**

Dictionary**(**size\_t size**);**

void insert**(**std**::**istream**&** in**);**

void search**(**std**::**istream**&** in**,** std**::**ostream**&** out**);**

void deleteWord**(**std**::**istream**&** in**);**

void show**(**std**::**istream**&** in**,** std**::**ostream**&** out**);**

private**:**

std**::**vector**<** DoubleLinkedList**<** Word **>** **>** array\_**;**

size\_t tableSize\_**;**

int numOfCollsions\_**;**

int getHash**(**const std**::**string**&** englishWord**);**

**};**

#endif

dictionary.cpp

#include "dictionary.hpp"

#include <string>

#include "word.h"

#include "tools.h"

inline int Dictionary**::**getHash**(**const std**::**string**&** englishWord**)**

**{**

int hash **=** 0**;**

**for** **(**size\_t i **=** 0**;** i **<** englishWord**.**length**();** i**++)**

**{**

hash **+=** englishWord**[**i**];**

**}**

hash **=** hash **%** tableSize\_**;**

**return** hash**;**

**}**

Dictionary**::**Dictionary**(**size\_t size**):**

tableSize\_**(**size**),**

numOfCollsions\_**(**0**)**

**{**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

array\_**.**push\_back**(**DoubleLinkedList**<** Word **>());**

**}**

**}**

void Dictionary**::**insert**(**std**::**istream**&** in**)**

**{**

Word word**;**

word**.**insertWord**(**in**);**

int index **=** getHash**(**word**.**getEnglishWord**());**

**if** **(!**array\_**[**index**].**empty**())**

**{**

**if** **(**array\_**[**index**].**searchItem**(**word**))**

**{**

mergeVectors**(**array\_**[**index**].**searchItem**(**word**)->**item\_**.**getVect**(),** word**.**getVect**());**

**}**

**else**

**{**

numOfCollsions\_**++;**

array\_**[**index**].**insertHead**(**word**);**

**}**

**}**

**else**

**{**

array\_**[**index**].**insertHead**(**word**);**

**}**

**}**

void Dictionary**::**search**(**std**::**istream**&** in**,** std**::**ostream**&** out**)**

**{**

std**::**string englishWord**;**

**do**

**{**

getline**(**in**,** englishWord**);**

**}** **while** **(**englishWord**.**empty**());**

**if** **(!**in**)**

**{**

**return;**

**}**

Word engWord**(**englishWord**);**

int index **=** getHash**(**englishWord**);**

**if** **(**array\_**[**index**].**searchItem**(**engWord**))**// O(n)

**{**

out **<<** "Dictionary contains this word\n"**;**

**}**

**else**

**{**

out **<<** "Dictionary does not contain this word\n"**;**

**}**

**}**

void Dictionary**::**deleteWord**(**std**::**istream**&** in**)**

**{**

std**::**string englishWord**;**

**do**

**{**

getline**(**in**,** englishWord**);**

**}** **while** **(!**in**.**eof**()** **&&** englishWord**.**empty**());**

int index **=** getHash**(**englishWord**);**

Word engWord**(**englishWord**);**

**if** **(**array\_**[**index**].**searchItem**(**engWord**))**

**{**

array\_**[**index**].**searchItem**(**engWord**)->**item\_**.**clear**();**

**}**

**else**

**{**

**throw** std**::**runtime\_error**(**"Dictionary does not contains this word\n"**);**

**}**

**}**

void Dictionary**::**show**(**std**::**istream**&** in**,** std**::**ostream**&** out**)**

**{**

std**::**string englishWord**;**

**do**

**{**

getline**(**in**,** englishWord**);**

**}** **while** **(!**in**.**eof**()** **&&** englishWord**.**empty**());**

Word engWord**(**englishWord**);**

int index **=** getHash**(**englishWord**);**

**if** **(**array\_**[**index**].**searchItem**(**engWord**))**

**{**

array\_**[**index**].**searchItem**(**engWord**)->**item\_**.**show**(**out**);**

out **<<** "\n"**;**

**}**

**else**

**{**

**throw** std**::**runtime\_error**(**"Dictionary does not contain this word\n"**);**

**}**

**}**

**word.hpp**

#ifndef WORD\_H

#define WORD\_H

#include <vector>

#include <iosfwd>

#include "DoubleLinkedList.hpp"

class Word

**{**

public**:**

Word**();**

Word**(**std**::**string englishWord**);**

void insertWord**(**std**::**istream**&** in**);**

std**::**string getEnglishWord**();**

std**::**vector**<** std**::**string **>&** getVect**();**

void show**(**std**::**ostream**&** out**);**

void clear**();**

bool **operator==(**const Word**&** rhs**);**

bool **operator!=(**const Word**&** rhs**);**

private**:**

std**::**string englishWord\_**;**

std**::**vector**<** std**::**string **>** translations\_**;**

**};**

void myToLower**(**std**::**string**&** str**);**

#endif

**word.cpp**

#include "word.h"

#include <iostream>

#include <string>

#include <sstream>

#include <algorithm>

#include <iterator>

#include "tools.h"

Word**::**Word**():**

englishWord\_**(**"<EMPTY>"**)**

**{**

**}**

Word**::**Word**(**std**::**string englishWord**):**

englishWord\_**(**englishWord**)**

**{**

**}**

void Word**::**insertWord**(**std**::**istream**&** in**)**

**{**

std**::**string line**;**

**do**

**{**

getline**(**in**,** line**);**

**}** **while** **(**line**.**empty**()** **&&** **!**in**.**eof**());**

**if** **(**line**.**empty**()** **&&** **!**in**.**eof**())**

**{**

**throw** std**::**invalid\_argument**(**"Empty input"**);**

**}**

std**::**stringstream sin**(**line**);**

char current **=** sin**.**get**();**

englishWord\_**.**clear**();**

sin **>>** std**::**ws**;**

**while** **(**current **!=** '=' **&&** **!**sin**.**fail**())**

**{**

englishWord\_ **+=** current**;**

current **=** sin**.**get**();**

**}**

**if** **(**englishWord\_**.**empty**()** **||** current **!=** '='**)**

**{**

**throw** std**::**invalid\_argument**(**"Invalid input\n"**);**

**}**

myToLower**(**englishWord\_**);**

doStandartString**(**englishWord\_**);**

**if** **(!**checkWordIsCorrect**(**englishWord\_**))**

**{**

**throw** std**::**invalid\_argument**(**"Invalid english word\n"**);**

**}**

bool flag **=** **true;**

**while** **(**flag**)**

**{**

std**::**string translation**;**

sin **>>** std**::**ws**;**

current **=** sin**.**get**();**

**while** **(!**sin**.**fail**()** **&&** current **!=** '|'**)**

**{**

translation **+=** current**;**

current **=** sin**.**get**();**

**}**

**if** **(**translation**.**empty**())**

**{**

**throw** std**::**invalid\_argument**(**"Invalid input\n"**);**

**}**

**if** **(**sin**.**eof**())**

**{**

flag **=** **false;**

**}**

doStandartString**(**translation**);**

myToLower**(**translation**);**

**if** **(!**checkWordIsCorrect**(**translation**))**

**{**

**throw** std**::**invalid\_argument**(**"Invalid translation\n"**);**

**}**

translations\_**.**push\_back**(**translation**);**

**}**

std**::**sort**(**translations\_**.**begin**(),** translations\_**.**end**());**

**}**

std**::**string Word**::**getEnglishWord**()**

**{**

**return** englishWord\_**;**

**}**

std**::**vector**<**std**::**string**>&** Word**::**getVect**()**

**{**

**return** translations\_**;**

**}**

void Word**::**show**(**std**::**ostream**&** out**)**

**{**

std**::**copy**(**translations\_**.**begin**(),** translations\_**.**end**(),** std**::**ostream\_iterator**<** std**::**string **>(**out**,** " "**));**

**}**

void Word**::**clear**()**

**{**

englishWord\_**.**clear**();**

translations\_**.**clear**();**

**}**

bool Word**::operator==(**const Word**&** rhs**)**

**{**

**return** englishWord\_ **==** rhs**.**englishWord\_**;**

**}**

bool Word**::operator!=(**const Word**&** rhs**)**

**{**

**return** englishWord\_ **!=** rhs**.**englishWord\_**;**

**}**

tools.h

#ifndef TOOLS\_H

#define TOOLS\_H

#include <string>

#include <vector>

bool checkWordIsCorrect**(**const std**::**string**&** word**);**

void mergeVectors**(**std**::**vector**<** std**::**string **>&** v1**,** std**::**vector**<** std**::**string **>&** v2**);**

void doStandartString**(**std**::**string**&** str**);**

void myToLower**(**std**::**string**&** str**);**

#endif

tools.cpp

#include "tools.h"

#include <locale>

#include <algorithm>

bool checkWordIsCorrect**(**const std**::**string**&** word**)**

**{**

**for** **(**size\_t i **=** 0**;** i **<** word**.**length**();** i**++)**

**{**

std**::**locale locRu**(**"ru-RU"**);**

std**::**locale locUS**(**"en-US"**);**

**if** **(!**std**::**isalpha**(**word**[**i**],** locRu**)** **&&** **!**std**::**isalpha**(**word**[**i**],** locUS**)** **&&** **!**std**::**isspace**(**word**[**i**]))**

**{**

**return** **false;**

**}**

**}**

**return** **true;**

**}**

void mergeVectors**(**std**::**vector**<**std**::**string**>&** v1**,** std**::**vector**<**std**::**string**>&** v2**)**

**{**

v1**.**insert**(**v1**.**end**(),** v2**.**begin**(),** v2**.**end**());**

std**::**sort**(**v1**.**begin**(),** v1**.**end**());**

v1**.**erase**(**std**::**unique**(**v1**.**begin**(),** v1**.**end**()),** v1**.**end**());**

**}**

void doStandartString**(**std**::**string**&** str**)**

**{**

**while** **(**str**.**back**()** **==** ' '**)**

**{**

str**.**pop\_back**();**

**}**

**}**

void myToLower**(**std**::**string**&** str**)**

**{**

**for** **(**size\_t i **=** 0**;** i **<** str**.**length**();** i**++)**

**{**

**if** **(**str**[**i**]** **>=** 'A' **&&** str**[**i**]** **<=** 'Z'**)**

**{**

str**[**i**]** **+=** 'z' **-** 'Z'**;**

**}**

**if** **(**str**[**i**]** **>=** '' **&&** str**[**i**]** **<=** 'ߧ**)**

**{**

str**[**i**]** **+=** '' **-** 'ߧ**;**

**}**

**}**

**}**