Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий
Высшая школа программной инженерии

РАСЧЕТНО-ГРАФИЧЕСКАЯ РАБОТА

Алгоритмы работы со словарями
по дисциплине **«Алгоритмы и структуры данных»**

Выполнил
студент гр.
3530904/00004 Почернин Владислав Сергеевич

Руководитель Червинский Алексей Петрович

«___» _____ 2021 г.

Санкт-Петербург
2021 г

Содержание

Введение. Общая постановка задачи

Тема: Алгоритмы 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 1.4.4

Частотный словарь. Хеш-таблица

Разработать и реализовать алгоритм формирования частотного словаря:

- Определить понятие слово.
- Прочитать текст и сформировать набор слов вместе с информацией о частоте их встречаемости.
- Определить три чаще всего встречающихся слова.

Для реализации задания использовать **хеш-таблицу с разрешением коллизий с помощью цепочек.**

Ключом хеш-функции должно быть слово.

Основная часть работы

1. Описание алгоритма решения и используемых структур данных

Хеш-таблица представляет собой эффективную структуру данных для реализации словарей (ключ – значение), в которой среднее время всех базовых словарных операций (а именно вставки, поиска и удаления элементов) составляет O(1) [1].

По своей сути хеш-таблица обобщает обычный массив, который хранит данные. Прямая индексация элементов массива обеспечивает доступ к любому элементу за постоянное время. Хитрость заключается в том, что для каждой пары (ключ – значение) индекс в массиве вычисляется по значению ключа с помощью хеш-функции.

Хеш-функция – специальная функция, которая определённым образом отображает конкретный ключ на конкретное значение индекса массива. Она может быть реализована по-разному. В написанной программе используется метод деления.

- Хеш-функция обязана давать одно и то же значение для одного и того же ключа.
- Хорошая хеш-функция обеспечивает равномерное распределение индексов массива, давая разные значения от разных ключей.

Имея массив, который позволяет за постоянное время обратиться к любому его элементу и хеш-функцию, которая генерирует нам конкретное значение индекса массива по конкретному значению ключа, мы можем создать хеш-таблицу:

- Добавление какого-либо элемента будет осуществляться не в очередную ячейку массива, а именно в ту, значение которой даст нам хеш-функция.
- Для поиска элемента нам не потребуется пробегаться по всему массиву, мы точно будем знать, какой индекс должен быть у искомого элемента с помощью хеш-функции.
- Удаление элемента также не потребует его поиска.

Однако, мы сталкиваемся со следующей проблемой: количество элементов, которые мы захотим добавить, а, следовательно, количество

ключей может оказаться больше, чем размер создаваемого нами массива. Кроме того, сама хеш-функция может оказаться не идеальной и независимо от количества добавляемых элементов дать одинаковое значение для двух разных ключей. Все это приводит к тому, что два или более ключа могут быть хешированы в одну и ту же ячейку. Такая ситуация называется **коллизией** [1].

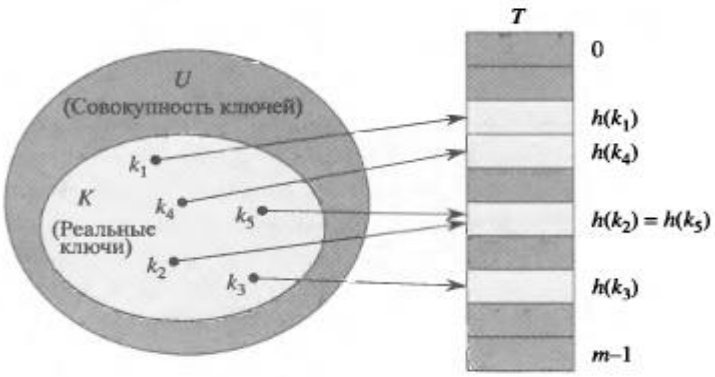

Рис. 1 Коллизия

Существуют разные методы борьбы с коллизиями. Одним из них является **метод цепочек**. В нём мы помещаем все элементы, ключи которых (с помощью хеш-функций) дали нам одинаковые значения в связанный список. Соответственно, каждая ячейка массива будет хранить не конкретные данные по данному ключу, а указатель на список всех элементов, хеш-значение ключа которых равно соответствующему индексу. Если же в какой-то ячейке массива элементов нет – она хранит указатель на nullptr [1].

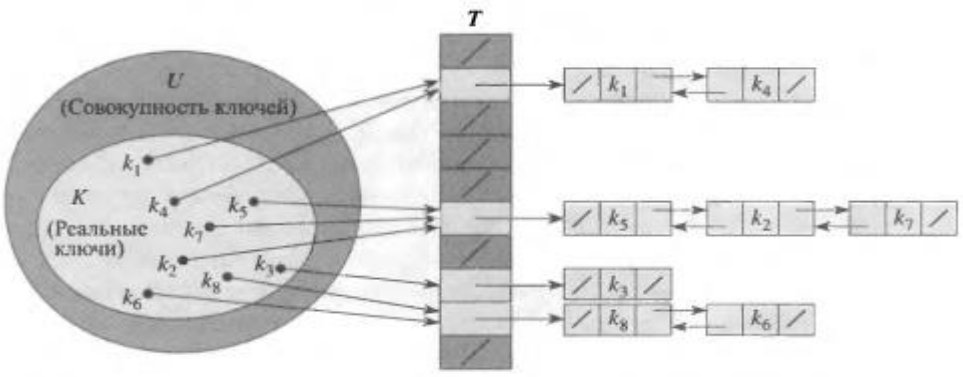

Рис. 2 Разрешение коллизий с помощью цепочек

При этом в скорости работы алгоритма возникнут некоторые нюансы. Подробнее этот момент будет рассмотрен в анализе алгоритма (2-й пункт).

Рассмотрим конкретную реализацию хеш-таблицы. В написанной программе используется две структуры данных

Двусвязный список

- Используется для разрешения коллизий с помощью цепочек.
- Каждый элемент представлен структурой **Node**:
  - **key_** - ключ (слово).
  - **value_** - значение (количество вхождений слова).
  - **next_** - указатель на следующий элемент списка.
  - **prev_** - указатель на предшествующий элемент списка.
- Реализованы следующие методы:
  - **insertItem()** – добавление узла с заданным ключем (или увеличение счетчика вхождений).
  - **searchItem()** – поиск количества вхождений слова.
  - **deleteItem() –** удаление узла с заданным ключем.
  - **clear() –** очищение списка.
  - **count() –** количество элементов списка.
  - **print() –** вывод содержимого списка в поток вывода.
  - **fillVector()** – добавление к поступаемому на вход вектору элементов из списка.
  - **fillThreeMost()** – заполнение вектора трех чаще всего встречающихся слов.

Хеш-таблица количества вхождений слова

- Хранит в себе массив двусвязных списков.
- Реализованы следующие методы:
  - **insertWord() –** добавление слова или увеличение счетчика его вхождений.
  - **searchWord() –** поиск количества вхождений слова.
  - **deleteWord() –** удаление слова.
  - **readFile() –** Чтение слов из файла в хеш-таблицу.
  - **getThreeMost() –** получение вектора пар трех чаще всего встречающихся слов.
  - **printUnsorted() –** Печать хеш-таблицы в поток вывода в неотсортированном виде.

    - **printSorted() –** Печать хеш-таблицы в поток вывода в отсортированном виде.
    - **printThreeMost() –** Печать трех чаще всего встречающихся слов в поток вывода.
    - **clear() –** очищение словаря.
    - **size() –** размер словаре.
    - **count() –** количество записей в словаре.
    - **collisions() –** количество коллизий в словаре.

Также были написаны вспомогательные функции.

Описание вспомогательных функций

- **getCharCode() –** получение кода символа.
    - Код русских букв может получаться отрицательным, данная функция приводит его к виду расширенной таблицы ASCII.
- **hashByDivision() –** хеш-функция методом деления.
    - Построение хеш-функции методом деления состоит в отображении ключа в одну из ячеек путем получения остатка от деления ключа на количество ячеек массива.
    - При этом хороший результат равномерного распределения хеш-значений можно получить, выбирая в качестве размера массива простое число, достаточно далекое от степени двойки [1].
- **isRussianLetter() –** проверка, является ли символ русской буквой.
- **isEnglishLetter() –** проверка, является ли символ английской буквой.
- **clearWord() –** очищение слова от символов, не являющихся буквами.
- **isWord() –** проверка, является ли строка словом.
- **toLower() –** перевод слова в нижний регистр.
- **quicksort() –** быстрая сортировка вектора пар (ключ – значение)
- **printVector() –** вывод вектора пар в поток вывода.
- Также были написаны технические функции интерфейса программы, однако приводить их в отчете – излишне.

2. Анализ алгоритма

Чтобы определить сильные и слабые стороны написанной программы, проведем анализ эффективности выполнения трёх основных операций: вставки, поиска и удаления элементов.

Двусвязный список

- Пусть n – количество элементов в двусвязном списке.
- **Вставка** в двусвязный список в любом случае будет осуществляться за O(n), поскольку, чтобы вставить элемент, нам сначала надо проверить, а не находится ли он уже в списке (в таком случае нужно будет просто увеличить количество вхождений слова), а, следовательно, найти его.
- **Поиск** имеет сложность O(n) в любом случае, потому что, чтобы найти элемент, необходимо пройтись по всему списку.
- **Удаление** из двусвязного списка также будет осуществляться за O(n) в любом случае, так как удаляемый элемент необходимо найти.
- Итак, все основные операции двусвязного списка осуществляются в любом случае за O(n).

Хеш-таблица

- Пусть n – количество элементом в соответствующем двусвязном списке.
- Определим такое понятие, как коэффициент заполнения. **Коэффициентом заполнения** будем называть отношение количества элементов в хеш-таблице ко всему размеру хеш-таблицы.
- Назовем средним случаем ситуацию, в которой мы имеем:
    - Хорошую хеш-функцию, которая равномерно отображает разные значения ключей на разные индексы массива.
    - Низкий коэффициент заполнения < 1.
- Назовем худшим случаем ситуацию, в которой мы имеем:
    - Плохую хеш-функцию, которая часто дает одни и те же значения индексов массива.
    - Высокий коэффициент заполнения > 1.

- **Вставка** в хеш-таблицу в среднем случае будет происходить за O(1), поскольку мы сразу получим индекс массива, куда нужно записать соответствующий ключ и запишем элемент в пустой список. В худшем случае эта операция будет происходить за O(n), поскольку нужно будет определить наличие элемента в списке.
- **Поиск** также будет осуществляться за O(1) в среднем случае и O(n) в худшем. Мы либо сразу определим наличие (или отсутствие) элемента в соответствующей ячейке массива, либо нам придется проходиться по списку для поиска.
- **Удаление** подразумевает поиск элемента из-за чего его скорость будет O(1) в среднем случае и O(n) в худшем.

Исходя из анализа алгоритма, можно заметить, что хеш-таблица является крайне эффективной структурой данных, позволяя выполнять все основные операции за постоянное время. Но для этого нужно:

- Выбирать такой размер массива, при котором коэффициент заполнения будет меньше единицы.
- Выбирать такую хеш-функцию, которая будет равномерно распределять разные ключи на разные ячейки массива.

В противном случае хеш-таблица будет работать со скоростью двусвязного списка, а именно за O(n).

Исходя из вышесказанного, можно определить оптимальный размер массива для нашей ситуации. Исходить будем из двух фактов:

- Средний словарный запас носителей русского языка – 80 тысяч слов.
- Хеш-функция методом деления дает наилучший результат при размере массива равном простому числу, далекому от степени двойки.

Следовательно, возьмем простое число, которое будет лежать между $2^{16} = 65\,536$ и $2^{17} = 131\,072$, а именно **98299.**

При таком размере массива по умолчанию, мы сможем минимизировать количество коллизий.

Быстрая сортировка

Из всех остальных функций следует выделить выбор алгоритма сортировки хеш-таблицы (для вывода в удобном для чтении виде).

Так как мы имеем дело с множеством слов, верхняя граница которых не превышает 150 000 для русского языка (согласно Большому академическому словарю) и 400 000 для английского (согласно Оксфордскому словарю), оптимальным вариантом является быстрая сортировка. Она имеет в среднем случае скорость O(nlog(n)) и будет эффективно работать на наших данных.

3. Описание спецификации программы (детальные требования)

1. При запуске программы пользователю предлагаются 3 варианта работы, которые он может выбрать, введя в консоль соответствующую команду.
   1.1. При вводе «**1**» начинается работа непосредственно с частотным словарем.
   1.2. При вводе «**2**» программа пропускается через набор тестов, далее происходит выход из программы.
   1.3. При вводе «**exit**» происходит выход из программы.
   1.4. При вводе некорректной команды или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.
2. В случае, если пользователь введёт «1», пользователю будут представлены полный список команд и приглашение ввода.
   2.1. «**help**» - посмотреть полный список команд.
      2.1.1. В консоль выводится полный список команд, далее следует приглашение ввода команды.
   2.2. «**exit**» - выйти из программы.
      2.2.1. Происходит выход из программы.
   2.3. «**insert**» - добавить слово.
      2.3.1. Выводится приглашение ввода слова, которое пользователь собирается добавить.
      2.3.2. При вводе корректного слова (определение корректного слова будет дано в описании программы (4-й пункт)) происходит перевод слова в нижний регистр, добавление его в хеш-таблицу и вывод сообщения об этом. Далее следует приглашение ввода команды.
      2.3.3. При вводе некорректного слова или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.
   2.4. «**search**» - найти слово.
      2.4.1. Выводится приглашение ввода слова, которое пользователь собирается найти.
      2.4.2. При вводе корректного слова происходит поиск в хеш-таблице, выводится сообщение с информацией о количестве встречаемости слова в таблице. Далее следует приглашение ввода команды.
      2.4.3. При вводе некорректного слова или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.

2.5. «**delete**» - удалить слово.

2.5.1. Выводится приглашение ввода слова, которое пользователь собирается удалить.

2.5.2. При вводе корректного слова, которое было в таблице, происходит его удаление и сообщение об этом. Далее следует приглашение ввода команды.

2.5.3. При вводе корректного слова, которого не было в таблице, выводится сообщение о его отсутствии. Далее следует приглашение ввода команды.

2.5.4. При вводе некорректного слова или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.

2.6. «**readFile**» - добавить все слова из файла.

2.6.1. Выводится приглашение ввода имени файла, из которого пользователь собирается прочитать слова.

2.6.2. При вводе корректного имени файла, который удается прочитать, происходит чтение слов из файла и вывод сообщения об этом. Далее следует приглашение ввода команды.

2.6.2.1. Все слова считываемого файла очищаются от символов, которые не являются буквами и переводятся в нижний регистр.

2.6.2.2. В случае, если после этого очередное слово является некорректным, оно просто игнорируется.

2.6.3. При вводе корректного имени файла, который не удается прочитать, выводится сообщение об ошибке и приглашение ввода команды.

2.6.4. При вводе некорректного имени фала или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.

2.7. «**print**» - вывести весь словарь.

2.7.1. В консоль выводится весь словарь в отсортированном виде в формате строк “«ключ»: «значение»”, далее следует приглашение ввода команды.

2.8. «**printThreeMost**» - вывести три чаще всего встречающихся слова.

2.8.1. В консоль выводятся три чаще всего встречающихся слова в формате строк “«ключ»: «значение»”, далее следует приглашение ввода команды.

2.9. «**writeFile**»

2.9.1. Выводится приглашение ввода имени файла, в который пользователь собирается прочитать слова.

2.9.2. При вводе корректного имени файла, который удается открыть и изменять, происходит запись слов в отсортированном виде в файл в формате строк “«ключ»: «значение»”, далее следует приглашение ввода команды.

2.9.3. При вводе корректного имени файла, который не удается открыть и изменять, выводится сообщение об ошибке и приглашение ввода команды.

2.9.4. При вводе некорректного имени файла или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.

2.10. «**clear**» - очистить словарь.

2.10.1. Происходит очистка словаря, выводится сообщение об этом. Далее следует приглашение ввода команды.

2.11. При вводе некорректной команды или любой другой строки выводится сообщение об ошибке и приглашение ввода команды.

4. Описание программы (структура программы, форматы входных и выходных данных)

Всю представленную программу можно разделить на 4 части:

- **double-linked-list.h** и **double-linked-list.cpp** – реализация двусвязного списка.
- **functions.h** и **functions.cpp** – вспомогательные функции.
- **frequency-dictionary.h** и **frequency-dictionary.cpp** – реализация самой хеш-таблицы (частотного словаря).
- **main.cpp** – точка входа программы, главное меню пользовательского интерфейса.

Всевозможные комбинации ввода и вывода описаны в спецификации программы (3-й пункт). Для полного её понимание следует дать определение понятию «слово».

Определение понятия «слово».

- Словом является строка, содержащая набор символов.
- Слово абсолютно не чувствительно к регистру. Все буквы верхнего регистра автоматически переводятся в нижний.
- В каждом слове должны быть только буквы, при наличии посторонних символов слово считается некорректным.
- Каждое слово должно состоять из букв либо русского, либо английского алфавита. При наличии в слове букв обоих алфавитов, слово считается некорректным.

Проблема русских символов.

При работе с русскими символами программа пользуется расширенной таблицей ASCII. В связи с этим корректная работа программы возможно только в кодировке Windows-1251.

- В консоли Linux следует работать только со словами из английских букв, русские буквы не будут правильно введены и отображены.
- Файлы с русскими словами следует создавать в кодировке Windows-1251, иначе они будут неправильно считаны.

Заключение

В течение курса «Алгоритмы и структуры данных» мной был получен пласт теоретических знаний и практических навыков в различных областях. В частности:

- О быстрой сортировке массива
- О написании связанных списков
- О хеш-функциях и хеш-таблицах

Всё это помогло мне написать свою собственную хеш-таблицу, проанализировать её сильные и слабые стороны и реализовать с её помощью частотный словарь.

Подводя итог, можно сказать, что хеш-таблица является незаменимой структурой данных в современном программировании, которая при должном внимании к размеру массива и хеш-функции позволяет создать наиэффективнейший способ быстрого получения значения по ключу. Именно поэтому её еще называют ассоциативным массивом.

Поставленная мне задача была выполнена в полном объёме, тщательно оттестирована и готова к практическому использованию.

Список использованных источников

1. Алгоритмы: построение и анализ: / Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. – М.: Вильямс, 2013. – 1328с.

Приложения

Приложение 1. Текст программы (по стандарту кодирования)

double-linked-list.hpp

```
#ifndef DOUBLE_LINKED_LIST_HPP
#define DOUBLE_LINKED_LIST_HPP

#include <iostream>
#include <string>
#include <vector>

// Дважды связный список пар ключ(слово) - значение(частота встречаемости слова).
class DoubleLinkedList
{
public:
  DoubleLinkedList(); // Конструктор "по умолчанию" - создание пустого списка.

  virtual ~DoubleLinkedList(); // Деструктор списка.

  bool insertItem(const std::string& key); // Добавление узла с заданным ключом (или
увеличение счетчика вхождений).
  size_t searchItem(const std::string& key) const; // Поиск количества вхождений слова.
  bool deleteItem(const std::string& key); // Удаление узла с заданным ключом.
  void clear();  // Очищение списка.
  size_t count() const; // Количество элементов списка.
  void print(std::ostream& out) const; // Вывести содержимое списка в out.
  void fillVector(std::vector< std::pair< std::string, size_t > >& vec) const; //
Добавить к поступаемому на вход вектору пары из списка.
  void fillThreeMost(std::vector< std::pair< std::string, size_t > >& vec) const; //
Заполнить вектор трех чаще всего встречающихся слов.

private:
  // Тип Node используется для описания элемента цепочки, связанного со
  // следующим с помощью поля next_ и предшествующим с помощью поле prev_.
  struct Node
  {
    std::string key_; // Ключ (слово).
    size_t value_;    // Значение (количество вхождений слова).
    Node* next_;      // Указатель на следующий элемент списка.
    Node* prev_;      // Указатель на предшествующий элемент списка.

    // Конструктор для создания нового элемента списка.
    Node(std::string key, size_t value, Node* next = nullptr, Node* prev = nullptr):
      key_(std::move(key)),
      value_(value),
      next_(next),
      prev_(prev)
    {}
  };

  size_t count_;      // Счетчик числа элементов списка.
  Node* head_;        // Указатель на первый элемент списка.
  Node* tail_;        // Указатель на последний элемент списка.

  void insertNode(Node* x); // Вставить сформированный узел в список.
  Node* searchNode(const std::string& key) const; // Поиск узла (адрес) с заданным
значением ключа.
  void deleteNode(Node* x); // Удаление заданного узла.
};

#endif
```

double-linked-list.cpp

```
#include "double-linked-list.hpp"

DoubleLinkedList::DoubleLinkedList():
  count_(0),
  head_(nullptr),
  tail_(nullptr)
{}

DoubleLinkedList::~DoubleLinkedList()
{
  clear();
}

bool DoubleLinkedList::insertItem(const std::string& key)
{
  Node* current = head_; // Указатель на текущий элемент
  while (current != nullptr) // Если в списке уже есть элемент с таким ключем - просто
увеличим его value_.
  {
    if (current->key_ == key)
    {
      current->value_++;
      return false; // Добавления не было.
    }
    current = current->next_;
  }
  // Сейчас мы уверены в том, что в списке нет элемента с ключем key.
  // Поэтому просто добавим в список элемент с ключем key и значением 1.
  insertNode(new Node(key, 1));
  return true; // Добавление было.
}

size_t DoubleLinkedList::searchItem(const std::string& key) const
{
  Node* result = searchNode(key);
  if (result == nullptr)
  {
    return 0;
  }
  else
  {
    return result->value_;
  }
}

bool DoubleLinkedList::deleteItem(const std::string& key)
{
  Node* current = head_;
  while (current != nullptr)
  {
    if (current->key_ == key)
    {
      deleteNode(current);
      return true; // Удаление было.
    }
    current = current->next_;
  }
  return false; // Удаления не было (так как элемент с таким ключом отсутствовал в
списке).
}

void DoubleLinkedList::clear()
```

```
{
  Node* current = nullptr; // Указатель на элемент, подлежащий удалению.
  Node* next = head_; // Указатель на следующий элемент.
  while (next != nullptr) // Пока в списке есть элементы.
  {
    current = next;
    next = next->next_; // Переход к следующему элементу.
    delete current; // Освобождение памяти.
  }
  count_ = 0;
  head_ = nullptr;
  tail_ = nullptr;
}

void DoubleLinkedList::print(std::ostream& out) const
{
  Node* current = head_;
  while (current != nullptr)
  {
    out << current->key_ <<": " << current->value_ << "\n"; // Выводим элемент.
    current = current->next_; // Переходим к следующему.
  }
}

void DoubleLinkedList::fillVector(std::vector < std::pair< std::string, size_t > > &vec)
const
{
  Node* current = head_;
  while (current != nullptr) // Просто записываем все пары ключ - значение в вектор.
  {
    std::pair< std::string, size_t > pair;
    pair.first = current->key_;
    pair.second = current->value_;
    vec.push_back(pair);
    current = current->next_;
  }
}

void DoubleLinkedList::fillThreeMost(std::vector< std::pair< std::string, size_t > >&
vec) const
{
  if (vec.size() != 3) // Вектор должен быть размером 3.
  {
    throw (std::invalid_argument("Некорректный размер подаваемого вектора!"));
  }

  Node* current = head_;
  while (current != nullptr) // Каждую пару ключ - значение ставим на свое место в
векторе (если она должна там быть).
  {
    std::pair< std::string, size_t > pair;
    pair.first = current->key_;
    pair.second = current->value_;
    if (pair.second > vec[0].second) // Если пара должна оказаться на первом месте...
    {
      vec[2] = vec[1]; // ...смещаем уже стоявшие элементы.
      vec[1] = vec[0];
      vec[0] = pair; // И ставим пару на первое место.
    }
    else if (pair.second > vec[1].second)
    {
      vec[2] = vec[1];
      vec[1] = pair;
```

```
    }
    else if (pair.second > vec[2].second)
    {
      vec[2] = pair;
    }
    current = current->next_;
  }
}

size_t DoubleLinkedList::count() const
{
  return count_;
}

void DoubleLinkedList::insertNode(Node* x)
{
  // Так как на этом этапе мы уверены, что элемента с таким ключем в списке нет - просто
добавим в начало списка.
  // Мы уверены в этом, потому что метод вызывается из insertItem() после проверки всего
списка.
  x->next_ = head_;
  if (head_ != nullptr)
  {
    // Список был НЕ пуст - предыдущий первый элемент должен указывать на вставляемый.
    head_->prev_ = x;
  }
  else
  {
    // Список был пуст - новый элемент будет и первым, и последним.
    tail_ = x;
  }
  head_ = x; // Теперь x - первый элемент списка.
  count_++; // Число элементов списка увеличилось.
}

DoubleLinkedList::Node* DoubleLinkedList::searchNode(const std::string& key) const
{
  Node* current = head_;
  while ((current != nullptr) && (current->key_ != key)) // Пока не прошли весь список
или не нашли нужный узел...
  {
    current = current->next_; // ...ищем дальше.
  }
  return current;
}

void DoubleLinkedList::deleteNode(Node* x)
{
  if (x == nullptr)
  {
    throw std::invalid_argument("DoubleLinkedList::deleteNode - неверно задан адрес
удаляемого узла");
  }

  if (x->prev_ != nullptr)
  {
    // Удаляется НЕ голова списка.
    // Значит с предыдущего элемента переносим указатель на следующий за удаляемым
элемент.
    (x->prev_)->next_ = x->next_;
  }
  else
  {
```

```
    // Удаляется голова списка.
    // Значит следующий за удаляемым элемент становится первым.
    head_ = x->next_;
  }

  if (x->next_ != nullptr)
  {
    // Удаляется НЕ хвост списка.
    // Значит со следующего элемента переносим указатель на предыдущий перед удаляемым
элемент.
    (x->next_)->prev_ = x->prev_;
  }
  else
  {
    // Удаляется хвост.
    // Значит предыдущий перед удаляемым элемент становится последним.
    tail_ = x->prev_;
  }
  delete x; // Чистим память.
  count_--; // Число элементов списка уменьшилось на 1.
}
```

frequency-dictionary.hpp

```
#ifndef FREQUENCY_DICTIONARY_HPP
#define FREQUENCY_DICTIONARY_HPP

#include <string>
#include <iostream>
#include <vector>

#include "double-linked-list.hpp"
#include "functions.hpp"

class FrequencyDictionary
{
public:
  FrequencyDictionary(); // Конструктор "по умолчанию" - создание словаря с размером "по
умолчанию".

  explicit FrequencyDictionary(size_t size); // Консутрктор словаря с конкретным
размером.

  ~FrequencyDictionary(); // Деструктор словаря.

  void insertWord(const std::string& str); // Добавление слова или увеличение счетчика
его вхождений.
  size_t searchWord(const std::string& str) const; // Поиск количества вхождений слова.
  bool deleteWord(const std::string& str); // Удаление слова.
  void readFile(const std::string& fileName); // Чтение слова из файла в хеш-таблицу.
  void fillVector(std::vector< std::pair< std::string, size_t > >& vec) const; //
Заполнение поступающего на вход вектора пар хеш-таблицей.
  std::vector< std::pair < std::string, size_t > > getThreeMost() const; // Получение
вектора пар трех чаще всего встречающихся слов.
  void printUnsorted(std::ostream& out) const; // Печать хеш-таблицы в out (в
неосортированном виде).
  void printSorted(std::ostream& out) const; // Печать хеш-таблицы в out (в
отсортированном виде).
  void printThreeMost(std::ostream& out) const; // Печать трех чаще всего встречающихся
слова в out.
  void clear(); // Очищение словаря.
  size_t size(); // Размер словаря.
  size_t count(); // Количество записей в словаре.
```

```
  size_t collisions(); // Количество коллизий в словаре.

private:
  size_t size_; // Размер массива двусвязных списков (размер хеш-таблицы).
  size_t count_; // Количество записей (слов, записанных в хеш-таблицу).
  DoubleLinkedList* data_; // Массив двусвязных списков (сама хеш-таблица).
};

#endif
```

frequency-dictionary.cpp

```
#include "frequency-dictionary.hpp"

#include <fstream>

const size_t DEFAULT_SIZE = 98299; // Рандомное простое число.

FrequencyDictionary::FrequencyDictionary():
  size_(DEFAULT_SIZE),
  count_(0),
  data_(new DoubleLinkedList[DEFAULT_SIZE])
{}

FrequencyDictionary::FrequencyDictionary(size_t size)
{
  if (size == 0)
  {
    throw (std::invalid_argument("Размер хеш таблицы должен быть положительным
числом!"));
  }
  size_ = size;
  count_ = 0;
  data_ = new DoubleLinkedList[size];
}

FrequencyDictionary::~FrequencyDictionary()
{
  delete[] data_;
}

void FrequencyDictionary::insertWord(const std::string& str)
{
  std::string word = str; // Слово, которое непосредственно будем добавлять.
  if (isWord(word)) // Проверяем на соответствие слову.
  {
    toLower(word); // Переводим слово в нижний регистр.
    // Одновременно добавляем элемент в таблицу
    // и проверяем, произошло добавление или просто увеличение счетчика.
    if (data_[hashByDivision(word, size_)].insertItem(word))
    {
      count_++;
    }
  }
  else
  {
    throw (std::invalid_argument("Некоорректное слово!"));
  }
}

size_t FrequencyDictionary::searchWord(const std::string& str) const
{
```

```
  if (isWord(str))
  {
    // Ищем значение по ключу str в сответствующем двусвязном списке.
    // Если слова в хеш таблице нет - просто получим 0.
    return data_[hashByDivision(str, size_)].searchItem(str);
  }
  else
  {
    throw(std::invalid_argument("Некорректное слово!"));
  }
}

bool FrequencyDictionary::deleteWord(const std::string& str)
{
  if (isWord(str))
  {
    // Одновременно производим попытку удаления
    // и уменьшение счетчика, если удаление произошло.
    if (data_[hashByDivision(str, size_)].deleteItem(str))
    {
      count_--;
      return true;
    }
  }
  else
  {
    throw(std::invalid_argument("Некорректное слово!"));
  }
  return false;
}

void FrequencyDictionary::readFile(const std::string& fileName)
{
  std::ifstream fin(fileName);
  if (!fin)
  {
    throw(std::runtime_error("Ошибка открытия файла!"));
  }

  std::string str = "";
  while (!fin.eof())
  {
    fin >> str; // Считываем очередную строку из файла.
    clearWord(str); // Очищаем строку от не букв.
    try
    {
      insertWord(str);
    }
    catch (const std::invalid_argument& error) // Пропускаем ошибки о некорректных словах
(пропускаем эти слова).
    {}
  }
  fin.close();
}

void FrequencyDictionary::printUnsorted(std::ostream& out) const
{
  for (size_t i = 0; i < size_; i++)
  {
    data_[i].print(out);
  }
}
```

```
void FrequencyDictionary::fillVector(std::vector < std::pair< std::string, size_t > >
&vec) const
{
  vec.clear();
  for (size_t i = 0; i < size_; i++)
  {
    data_[i].fillVector(vec);
  }
}

void FrequencyDictionary::printSorted(std::ostream &out) const
{
  std::vector< std::pair< std::string, size_t > > vec;
  fillVector(vec);
  quickSort(vec, 0, vec.size() - 1);
  printVector(vec, out);
}

std::vector< std::pair< std::string, size_t > > FrequencyDictionary::getThreeMost() const
{
  std::vector< std::pair< std::string, size_t > > vec;
  std::pair< std::string, size_t > emptyPair;
  emptyPair.first = "";
  emptyPair.second = 0;
  for (size_t i = 0; i < 3; i++) // Заполняем созданный вектор тремя пустыми парами.
  {
    vec.push_back(emptyPair);
  }

  for (size_t i = 0; i < size_; i++) // Заполняем вектор трех чаще всего встречающихся
слов.
  {
    data_[i].fillThreeMost(vec);
  }

  return vec;
}

void FrequencyDictionary::printThreeMost(std::ostream& out) const
{
  std::vector< std::pair< std::string, size_t > > vec = getThreeMost();
  printVector(vec, out);
}

void FrequencyDictionary::clear()
{
  for (size_t i = 0; i < size_; i++)
  {
    data_[i].clear(); // Очищаем каждый двусвязный список хеш-таблицы.
  }
  count_ = 0; // Не забываем обнулить количество слов в словаре.
}

size_t FrequencyDictionary::size()
{
  return size_;
}

size_t FrequencyDictionary::count()
{
  return count_;
}
```

```
size_t FrequencyDictionary::collisions()
{
  size_t result = 0;
  for (size_t i = 0; i < size_; i++)
  {
    if (data_[i].count() > 1) // Если в двусвязном списке больше 1 слова - это коллизия.
    {
      result++;
    }
  }
  return result;
}
```

functions.hpp

```
#ifndef FUNCTIONS_HPP
#define FUNCTIONS_HPP

#include <string>
#include <vector>

class FrequencyDictionary; // Говорим, что такой класс существует, чтобы использовать его
в сигнатурах.

size_t getCharCode(char ch); // Получение кода символа (код русских букв может получаться
как "код - 256").
size_t hashByDivision(const std::string& str, size_t size); // Хеш-функция методом
деления.

bool isRussianLetter(char ch); // true - если символ - русская буква.
bool isEnglishLetter(char ch); // true - если символ - английская буква.
void clearWord(std::string& str); // Очищение слова от не букв.
bool isWord(const std::string& str); // true - если строка - подходящее нашему
определению слово.
void toLower(std::string& str); // Перевод слова в нижний регистр.

void quickSort(std::vector< std::pair< std::string, size_t > >& vec, int l, int r); //
Быстрая сортировка вектора пар.
void printVector(const std::vector< std::pair< std::string, size_t > >& vec,
std::ostream& out); // Вывод вектора пар в out.

void startProgram(); // Функция выполнения основной программы.
void showHelp(); // Вывод списка команд.
void doInsert(FrequencyDictionary& dictionary); // Добавление слова.
void doSearch(const FrequencyDictionary& dictionary); // Поиск слова.
void doDelete(FrequencyDictionary& dictionary); // Удаление слова.
void doReadFile(FrequencyDictionary& dictionary); // Добавление слова из файла.
void doPrint(FrequencyDictionary& dictionary); // Вывод словаря в консоль.
void doPrintThreeMost(const FrequencyDictionary& dictionary); // Печать трех чаще всего
встречающихся слов.
void doWriteFile(const FrequencyDictionary& dictionary); // Запись в файл слов по
убыванию их встречаемости.
void doClear(FrequencyDictionary& dictionary); // Очищение словаря.

void testProgram(); // Функция выполнения тестов.
void doDoubleLinkedListTest(); // Тестирование двусвязного списка.
void doFunctionsTest(); // Тестирование вспомогательных функций.
void doFrequencyDictionaryTest(); // Тестирование частотного словаря.

#endif
```

functions.cpp

```
#include "functions.hpp"

#include <cassert> // Для assert.
#include <ctime> // Для рандомизации.
#include <iostream>
#include <fstream>
#include <iomanip> // Для красивого вывода.

#include "frequency-dictionary.hpp"

size_t getCharCode(char ch)
{
  int temp = static_cast< int >(ch);
  if (temp >= 0)
  {
    return static_cast< size_t >(temp);
  }
  else
  {
    assert(temp > -256); // Если код <= -256 - значит что-то не так. Так не должно быть
впринципе.
    return static_cast<size_t >(256 + temp);
  }
}

size_t hashByDivision(const std::string& str, size_t size)
{
  unsigned long long k = 0; // Число, в которое мы переведем нашу строку.
  unsigned long long factor = 1; // Степени 4-ки (на них будем умножать код каждого
символа строки.
  for (int i = static_cast< int >(str.size()) - 1; i >= 0; i--) // Преобразуем строку в
число, умножая каждый символ на соответсвующую степень 4-ки.
  {
    k += getCharCode(str[i]) * factor;
    factor *= 4;
  }
  return k % size; // Берем остаток от деления получившегося числа на размер хеш-таблицы.
}

void clearWord(std::string& str)
{
  std::string::iterator i = str.begin(); // Итератор на начало слова.
  while (i != str.end()) // Пока мы не дошли до конца слова...
  {
    if (!(isRussianLetter(*i) || isEnglishLetter(*i))) // Если очередной символ - не
буква...
    {
      i = str.erase(i); // ...удаляем его, присваивая итератору следующее положение.
    }
    else
    {
      i++; // Иначе просто передвигаем итератор вперед.
    }
  }
}

bool isRussianLetter(char ch)
{
  size_t firstUpperCode = 192; // Код буквы 'А'.
  size_t lastUpperCode = 223; // Код буквы 'Я'.
  size_t firstLowerCode = 224; // Код буквы 'а'.
  size_t lastLowerCode = 255; // Код буквы 'я'.
```

```
  size_t additionalUpperCode = 168; // Код буквы 'Ё'.
  size_t additionalLowerCode = 184; // Код буквы 'ё'.
  size_t code = getCharCode(ch); // Код подаваемой на вход буквы.
  return (((code >= firstUpperCode) && (code <= lastUpperCode)) ||
          ((code >= firstLowerCode) && (code <= lastLowerCode)) ||
          (code == additionalUpperCode) ||
          (code == additionalLowerCode));
}

bool isEnglishLetter(char ch)
{
  size_t firstUpperCode = 65; // Код буквы 'A'.
  size_t lastUpperCode = 90; // Код буквы 'Z'.
  size_t firstLowerCode = 97; // Код буквы 'a'.
  size_t lastLowerCode = 122; // Код буквы 'z'.
  size_t code = getCharCode(ch); // Код подаваемой на вход буквы.
  return (((code >= firstUpperCode) && (code <= lastUpperCode)) || ((code >=
firstLowerCode) && (code <= lastLowerCode)));
}

bool isWord(const std::string& str)
{
  if (str.empty())
  {
    return false; // Пустая строка - не слово.
  }

  bool isRussian = true; // true - слово русское, false - английское.
  if (isRussianLetter(str[0]))
  {
    isRussian = true; // Первая буква русская => все слово должно быть русским.
  }
  else if (isEnglishLetter(str[0]))
  {
    isRussian = false; // Первая буква английская => все слово должно быть английским.
  }
  else
  {
    return false; // Если первая буква не прошла две предыдущие проверки, значит там не
буква из алфавита.
  }

  // Проверка, чтобы все буквы слова были того же языка, что и первая.
  // При этом, автоматически фильтруются лишние символы.
  for (size_t i = 1; i < str.size(); i++)
  {
    if (isRussian)
    {
      if (!isRussianLetter(str[i]))
      {
        return false;
      }
    }
    else
    {
      if (!isEnglishLetter(str[i]))
      {
        return false;
      }
    }
  }

  return true; // true - так как пройдены все проверки.
```

```
}

void toLower(std::string& str)
{
  if (!isWord(str)) // На вход должно поступать русское/английское слово.
  {
    throw (std::invalid_argument("На вход поступило не слово!"));
  }

  if (isRussianLetter(str[0])) // Если слово русское...
  {
    size_t firstUpperCode = 192; // Код буквы 'А'.
    size_t lastUpperCode = 223; // Код буквы 'Я'.
    size_t firstLowerCode = 224; // Код буквы 'а'.
    size_t additionalUpperCode = 168; // Код буквы 'Ё'.
    size_t additionalLowerCode = 184; // Код буквы 'ё'.
    for (size_t i = 0; i < str.size(); i++)
    {
      size_t code = getCharCode(str[i]);
      if ((code >= firstUpperCode) && (code <= lastUpperCode))
      {
        // ...заменяем каждую заглавную русскую букву прописной.
        str[i] = static_cast< char >(code + (firstLowerCode - firstUpperCode));
      }
      else if (code == additionalUpperCode) // Не забываем про букву 'Ё'.
      {
        str[i] = static_cast < char >(additionalLowerCode);
      }
    }
  }
  else // Если слово английское...
  {
    size_t firstUpperCode = 65; // Код буквы 'A'.
    size_t lastUpperCode = 90; // Код буквы 'Z'.
    size_t firstLowerCode = 97; // Код буквы 'a'.
    for (size_t i = 0; i < str.size(); i ++)
    {
      size_t code = getCharCode(str[i]);
      if ((code >= firstUpperCode) && (code <= lastUpperCode))
      {
        // ...заменяем каждую заглавную английскую букву прописной.
        str[i] = static_cast< char >(code + (firstLowerCode - firstUpperCode));
      }
    }
  }
}

void quickSort(std::vector< std::pair< std::string, size_t > >& vec, int l, int r)
{
  srand(time(0)); // Рандомизация.

  if (l < r) // Массив из 1 или 0 элементов уже отсортирован.
  {
    int pivotIndex = (rand() % (r - l + 1)) + l; // Выбираем случайный опорный элемент.
    std::pair< std::string, size_t > pivot = vec[pivotIndex];
    std::swap(vec[pivotIndex], vec[r]); // Ставим опорный элемент на последнее место.
    int i = l - 1; // Вспомогательная переменная, чтобы следовать по элементам, которые
больше опорного.
                   // Изначально она указывает за левую границу сортируемой части
массива.
    for (int j = l; j <= r - 1; j++) // Цикл, который поставит элементы, которые больше
или равны опорному в начало.
    {
```

```
      if (vec[j].second >= pivot.second) // Если очередной элемент больше или равен
опорному...
      {
        i++; // ...увеличиваем вспомогтельню переменную. Теперь она в том месте, где
должен стоять этот элемент.
        std::swap(vec[i], vec[j]); // Меняем местами найдный элемент и элемент, на
который указывает i.
      }
    }

    // В итоге, в начале массива будут элементы больше или равные опорному.
    // Причем i будет указывать на последний такой элемент.
    std::swap(vec[i + 1], vec[r]); // Переносим опорный элемент на своё место (за
последним большим или равным ему).
    pivotIndex = i + 1; // Его индекс, соответственно, стал i + 1.
    quickSort(vec, l, pivotIndex - 1); // Сортируем левую от опорного элемента часть
массива.
    quickSort(vec, pivotIndex + 1, r); // Сортируем правую от опорного элемента часть
массива.
  }
}

void printVector(const std::vector< std::pair< std::string, size_t > >& vec,
std::ostream& out)
{
  for (size_t i = 0; i < vec.size(); i++)
  {
    out << vec[i].first << ": " << vec[i].second << "\n";
  }
}

void startProgram()
{
  FrequencyDictionary dictionary;
  std::cout << "-------------------------------------------------------\n"
            << "Началось выполнение программы\n"
            << "\"help\" - посмотреть полный список команд\n"
            << "\"exit\" - выйти из программы\n"
            << "\"insert\" - добавить слово\n"
            << "\"search\" - найти слово\n"
            << "\"delete\" - удалить слово\n"
            << "\"readFile\" - добавить все слова из файла\n"
            << "\"print\" - вывести весь словарь\n"
            << "\"printThreeMost\" - вывести три чаще всего встречающихся слова\n"
            << "\"writeFile\" - записать в файл слова по убыванию их встречаемости\n"
            << "\"clear\" - очистить словарь\n"
            << "-------------------------------------------------------\n";
  std::string input = "";
  while (true)
  {
    try
    {
      std::cout << "Введите команду: ";
      std::cin >> input;
      if (std::cin.fail() || std::cin.peek() != 10)
      {
        std::cin.clear();
        std::cin.ignore(32767, '\n');
        throw (std::invalid_argument("Некорректная команда!"));
      }
      if (input == "help")
      {
        showHelp();
```

```
      }
      else if (input == "insert")
      {
        doInsert(dictionary);
      }
      else if (input == "search")
      {
        doSearch(dictionary);
      }
      else if (input == "delete")
      {
        doDelete(dictionary);
      }
      else if (input == "readFile")
      {
        doReadFile(dictionary);
      }
      else if (input == "print")
      {
        doPrint(dictionary);
      }
      else if (input == "printThreeMost")
      {
        doPrintThreeMost(dictionary);
      }
      else if (input == "writeFile")
      {
        doWriteFile(dictionary);
      }
      else if (input == "clear")
      {
        doClear(dictionary);
      }
      else if (input == "exit")
      {
        break;
      }
      else
      {
        throw (std::invalid_argument("Некорректная команда!"));
      }
    }
    catch (const std::exception& error)
    {
      std::cout << "Ошибка: " << error.what() << "\n";
    }
  }
}

void showHelp()
{
  std::cout << "-------------------------------------------------------\n"
            << "Список команд:\n"
            << "\"help\" - посмотреть полный список команд\n"
            << "\"exit\" - выйти из программы\n"
            << "\"insert\" - добавить слово\n"
            << "\"search\" - найти слово\n"
            << "\"delete\" - удалить слово\n"
            << "\"readFile\" - добавить все слова из файла\n"
            << "\"print\" - вывести весь словарь\n"
            << "\"printThreeMost\" - вывести три чаще всего встречающихся слова\n"
            << "\"writeFile\" - записать в файл слова по убыванию их встречаемости\n"
            << "\"clear\" - очистить словарь\n"
```

```
            << "-----------------------------------------------------\n";
}
void doInsert(FrequencyDictionary& dictionary)
{
  std::string str;
  std::cout << "Введите слово, которое хотите добавить: ";
  std::cin >> str;
  if (std::cin.fail() || std::cin.peek() != 10)
  {
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    throw (std::invalid_argument("Некорректное слово!"));
  }
  dictionary.insertWord(str);
  std::cout << "Успешно.\n";
}

void doSearch(const FrequencyDictionary& dictionary)
{
  std::string str;
  std::cout << "Введите слово, которое хотите найти: ";
  std::cin >> str;
  if (std::cin.fail() || std::cin.peek() != 10)
  {
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    throw (std::invalid_argument("Некорректное слово!"));
  }
  size_t count = dictionary.searchWord(str);
  std::cout << "Слово \"" << str << "\" встречается " << count << " раз.\n";
}

void doDelete(FrequencyDictionary& dictionary)
{
  std::string str;
  std::cout << "Введите слово, которое хотите удалить: \n";
  std::cin >> str;
  if (std::cin.fail() || std::cin.peek() != 10)
  {
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    throw (std::invalid_argument("Некорректное слово!"));
  }
  if (dictionary.deleteWord(str))
  {
    std::cout << "Слово успешно удалено.\n";
  }
  else
  {
    std::cout << "Удаление не произошло, слова в словаре не было.\n";
  }
}

void doReadFile(FrequencyDictionary& dictionary)
{
  std::string str;
  std::cout << "Введите имя файла, который хотите прочитать: ";
  std::cin >> str;
  if (std::cin.fail() || std::cin.peek() != 10)
  {
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    throw (std::invalid_argument("Некорректное имя файла!"));
```

```
  }
  dictionary.readFile(str);
  std::cout << "Успешно.\n";
}

void doPrint(FrequencyDictionary& dictionary)
{
  dictionary.printSorted(std::cout);
}

void doPrintThreeMost(const FrequencyDictionary& dictionary)
{
  std::cout << "Три чаще всего встречающихся слова:\n";
  dictionary.printThreeMost(std::cout);
}

void doWriteFile(const FrequencyDictionary& dictionary)
{
  std::string str;
  std::cout << "Введите имя файла, в который хотите записать: ";
  std::cin >> str;
  if (std::cin.fail() || std::cin.peek() != 10)
  {
    std::cin.clear();
    std::cin.ignore(32767, '\n');
    throw (std::invalid_argument("Некорректное имя файла!"));
  }
  std::ofstream fout(str);
  if (!fout)
  {
    throw(std::runtime_error("Ошибка открытия файла!"));
  }
  dictionary.printSorted(fout);
  std::cout << "Успешно.\n";
  fout.close();
}

void doClear(FrequencyDictionary& dictionary)
{
  dictionary.clear();
  std::cout << "Словарь успешно очищен.\n";
}

void testProgram()
{
  doDoubleLinkedListTest();
  doFunctionsTest();
  doFrequencyDictionaryTest();
}

void doDoubleLinkedListTest()
{
  std::cout << "---------------------------------ТЕСТИРОВАНИЕ ДВУСВЯЗНОГО СПИСКА---------
-------------------------\n";
  std::cout << "-----------------ТЕСТ 1: ПУСТОЙ СПИСОК-----------------\n";
  DoubleLinkedList list;
  std::cout << "Количество элементов: " << list.count() << "\n";
  std::cout << "Сам список: \n";
  list.print(std::cout);

  std::cout << "Слово \"test\" встречается в списке " << list.searchItem("test") << "
раз.\n";
```

```
  std::cout << "Удаление слова \"test\": " << (list.deleteItem("test") ? "произошло." :
"не произошло.") << "\n";
  std::cout << "Количество элементов: " << list.count() << "\n";
  std::cout << "Сам список: \n";

  list.print(std::cout);
  std::vector< std::pair< std::string, size_t > > vec;
  list.fillVector(vec);
  std::cout << "Вектор, заполненный пустым списокм:\n";
  printVector(vec, std::cout);

  std::vector< std::pair< std::string, size_t > > vec3;
  std::pair< std::string, size_t > emptyPair;
  emptyPair.first = "";
  emptyPair.second = 0;
  for (size_t i = 0; i < 3; i++)
  {
    vec3.push_back(emptyPair);
  }
  list.fillThreeMost(vec3);
  std::cout << "Вектор трех чаще всего встречающихся слов, заполненный пустым
списокм:\n";
  printVector(vec3, std::cout);
  std::cout << "-----------------------------------------------------\n";

  std::cout << "-----------------ТЕСТ 2: ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ И ПОИСК--\n";
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("programming");
  list.insertItem("programming");
  list.insertItem("student");
  list.insertItem("student");
  list.insertItem("student");
  list.insertItem("teacher");
  std::cout << "Количество элементов: " << list.count() << "\n";
  std::cout << "Сам список: \n";
  list.print(std::cout);

  std::cout << "Слово \"test\" встречается в списке " << list.searchItem("test") << "
раз.\n";
  std::cout << "Слово \"deadline\" встречается в списке " << list.searchItem("deadline")
<< " раз.\n";
  std::cout << "-----------------------------------------------------\n";

  std::cout << "-----------------ТЕСТ 3: УДАЛЕНИЕ ЭЛЕМЕНТОВ-----------\n";
  std::cout << "Список до удаления элементов:\n";
  std::cout << "Количество элементов: " << list.count() << "\n";
  list.print(std::cout);

  std::cout << "Удаление слова \"test\": " << (list.deleteItem("test") ? "произошло." :
"не произошло.") << "\n";
  std::cout << "Удаление слова \"deadline\": " << (list.deleteItem("deadline") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "Удаление слова \"student\": " << (list.deleteItem("student") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "Удаление слова \"test\": " << (list.deleteItem("test") ? "произошло." :
"не произошло.") << "\n";

  std::cout << "Список после удаления элементов:\n";
  std::cout << "Количество элементов: " << list.count() << "\n";
```

```
  list.print(std::cout);
  std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 4: ЗАПОЛНЕНИЕ ВЕКТОРОВ------------\n";
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("test");
  list.insertItem("language");
  list.insertItem("language");
  list.insertItem("language");
  list.insertItem("language");
  list.insertItem("language");
  list.insertItem("law");
  std::cout << "Сам список: \n";
  std::cout << "Количество элементов: " << list.count() << "\n";
  list.print(std::cout);

  list.fillVector(vec);
  std::cout << "Вектор, заполненный списком:\n";
  printVector(vec, std::cout);

  list.fillThreeMost(vec3);
  std::cout << "Вектор трех чаще всего встречающихся слов:\n";
  printVector(vec3, std::cout);

  vec.clear();
  std::cout << "Вектор, заполненный пустым списокм:\n";
  printVector(vec, std::cout);
  std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 5: ОЧИСТКА СПИСКА-----------------\n";
  std::cout << "Список до очкстки: \n";
  std::cout << "Количество элементов: " << list.count() << "\n";
  list.print(std::cout);

  list.clear();
  std::cout << "Список после очкстки: \n";
  std::cout << "Количество элементов: " << list.count() << "\n";
  list.print(std::cout);
  std::cout << "-----------------------------------------------------\n";
  std::cout << "-------------------------------------------------------------------------
------------------------\n";
}

void doFunctionsTest()
{
  std::cout << "---------------------------------ТЕСТИРОВАНИЕ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ----
----------------------------\n";
  std::cout << "----------------ТЕСТ 1: getCharCode()-----------------\n";
  std::ifstream finRussianLetters("russianLetters.txt"); // Берём их из файла, потому что
должна быть другая кодировка (в IDE UTF-8).
  std::string russianLetters = "";
  finRussianLetters >> russianLetters;
  finRussianLetters.close();
  std::string englishLetters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
  std::string randomSymbols = "!@#$%^&*()_-+=";

  std::cout << "Русские буквы:\n";
  for (size_t i = 0; i < russianLetters.size(); i++)
  {
```

```
    std::cout << russianLetters[i] << ": " << std::setw(4) <<
getCharCode(russianLetters[i]) << ", ";
    if ((i != 0) && (i % 7 == 0))
    {
      std::cout << "\n";
    }
  }

  std::cout << "Английские буквы:\n";
  for (size_t i = 0; i < englishLetters.size(); i++)
  {
    std::cout << englishLetters[i] << ": " << std::setw(4) <<
getCharCode(englishLetters[i]) << ", ";
    if ((i != 0) && (i % 7 == 0))
    {
      std::cout << "\n";
    }
  }

std::cout << "Случайные символы:\n";
  for (size_t i = 0; i < randomSymbols.size(); i++)
  {
    std::cout << randomSymbols[i] << ": " << std::setw(4) <<
getCharCode(randomSymbols[i]) << ", ";
    if ((i != 0) && (i % 7 == 0))
    {
      std::cout << "\n";
    }
  }
  std::cout << "\n";
  std::cout << "-------------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 2: hashByDivision()---------------\n";
  std::cout << "Пусть размер таблицы - простое число 23.\n";
  std::cout << "Слово \"test\": " << hashByDivision("test", 23) << "\n";
  std::cout << "Слово \"programmer\": " << hashByDivision("programmer", 23) << "\n";
  std::cout << "Слово \"deadline\": " << hashByDivision("deadline", 23) << "\n";
  std::cout << "Слово \"student\": " << hashByDivision("student", 23) << "\n";
  std::cout << "Слово \"teacher\": " << hashByDivision("teacher", 23) << "\n";
  std::cout << "Слово \"language\": " << hashByDivision("language", 23) << "\n";

  std::cout << "Еще раз (чтобы проверить неизменность хешей).\n";
  std::cout << "Слово \"test\": " << hashByDivision("test", 23) << "\n";
  std::cout << "Слово \"programmer\": " << hashByDivision("programmer", 23) << "\n";
  std::cout << "Слово \"deadline\": " << hashByDivision("deadline", 23) << "\n";
  std::cout << "Слово \"student\": " << hashByDivision("student", 23) << "\n";
  std::cout << "Слово \"teacher\": " << hashByDivision("teacher", 23) << "\n";
  std::cout << "Слово \"language\": " << hashByDivision("language", 23) << "\n";

  std::cout << "А теперь пусть размер таблицы - простое число 137.\n";
  std::cout << "Слово \"test\": " << hashByDivision("test", 137) << "\n";
  std::cout << "Слово \"programmer\": " << hashByDivision("programmer", 137) << "\n";
  std::cout << "Слово \"deadline\": " << hashByDivision("deadline", 137) << "\n";
  std::cout << "Слово \"student\": " << hashByDivision("student", 137) << "\n";
  std::cout << "Слово \"teacher\": " << hashByDivision("teacher", 137) << "\n";
  std::cout << "Слово \"language\": " << hashByDivision("language", 137) << "\n";

  std::cout << "Еще раз (чтобы проверить неизменность хешей).\n";
  std::cout << "Слово \"test\": " << hashByDivision("test", 137) << "\n";
  std::cout << "Слово \"programmer\": " << hashByDivision("programmer", 137) << "\n";
  std::cout << "Слово \"deadline\": " << hashByDivision("deadline", 137) << "\n";
  std::cout << "Слово \"student\": " << hashByDivision("student", 137) << "\n";
  std::cout << "Слово \"teacher\": " << hashByDivision("teacher", 137) << "\n";
```

```
  std::cout << "Слово \"language\": " << hashByDivision("language", 137) << "\n";
  std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 3: isRussianLetter()-------------\n";
  std::cout << "Пройдемся по всем русским буквам.\n";
  for (size_t i = 0; i < russianLetters.size(); i++)
  {
    std::cout << russianLetters[i] << ": " << std::setw(4) <<
(isRussianLetter(russianLetters[i]) ? "Да" : "Нет") << ", ";
    if ((i != 0) && (i % 7 == 0))
    {
      std::cout << "\n";
    }
  }
  std::cout << "\n";
  std::cout << "Q" << ": " << std::setw(4) << (isRussianLetter('Q') ? "Да" : "Нет") <<
"\n";
  std::cout << "r" << ": " << std::setw(4) << (isRussianLetter('r') ? "Да" : "Нет") <<
"\n";
  std::cout << "!" << ": " << std::setw(4) << (isRussianLetter('!') ? "Да" : "Нет") <<
"\n";
  std::cout << " " << ": " << std::setw(4) << (isRussianLetter(' ') ? "Да" : "Нет") <<
"\n";
  std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 4: isEnglishLetter()-------------\n";
  std::cout << "Пройдемся по всем английским буквам.\n";
  for (size_t i = 0; i < englishLetters.size(); i++)
  {
    std::cout << englishLetters[i] << ": " << std::setw(4) <<
(isEnglishLetter(englishLetters[i]) ? "Да" : "Нет") << ", ";
    if ((i != 0) && (i % 7 == 0))
    {
      std::cout << "\n";
    }
  }
  std::cout << "\n";
  std::cout << ";" << ": " << std::setw(4) << (isEnglishLetter(';') ? "Да" : "Нет") <<
"\n";
  std::cout << "-" << ": " << std::setw(4) << (isEnglishLetter('-') ? "Да" : "Нет") <<
"\n";
  std::cout << "!" << ": " << std::setw(4) << (isEnglishLetter('!') ? "Да" : "Нет") <<
"\n";
  std::cout << " " << ": " << std::setw(4) << (isEnglishLetter(' ') ? "Да" : "Нет") <<
"\n";
  std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 5: clearWord()--------------------\n";
  std::string word = "Hello,";
  std::cout << "\"" << word << "\": ";
  clearWord(word);
  std::cout << "\"" << word << "\"\n";

  word = "@#@q&^w&^&*e&^&*r^%&t^&%&^y#@#";
  std::cout << "\"" << word << "\": ";
  clearWord(word);
  std::cout << "\"" << word << "\"\n";

  word = "O0neMo0reTest";
  std::cout << "\"" << word << "\": ";
  clearWord(word);
  std::cout << "\"" << word << "\"\n";
  std::cout << "-----------------------------------------------------\n";
```

```
  std::cout << "----------------TECT 6: isWord()----------------------\n";
  std::cout << "Сначала пройдемся по случайным русским словам.\n";
  std::ifstream finRussianWords("russianWords.txt"); // Берём их из файла, потому что
должна быть другая кодировка (в IDE UTF-8).
  while (!finRussianWords.eof())
  {
    finRussianWords >> word;
    std::cout << "\"" << word << "\": " << (isWord(word) ? "Да\n" : "Нет\n");
  }
  finRussianWords.close();

  std::cout << "Теперь по английским.\n";
  std::cout << "\"" << "dog" << "\": " << (isWord("dog") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "function" << "\": " << (isWord("function") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "quality" << "\": " << (isWord("quality") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "Member" << "\": " << (isWord("Member") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "WeATHeR" << "\": " << (isWord("WeATHeR") ? "Да\n" : "Нет\n");

  std::cout << "Теперь по НЕ словам.\n";
  std::cout << "\"" << "do!g" << "\": " << (isWord("do!g") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "@function" << "\": " << (isWord("@function") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "quality#" << "\": " << (isWord("quality#") ? "Да\n" : "Нет\n");
  std::cout << "\"" << "$@$@$@" << "\": " << (isWord("$@$@$@") ? "Да\n" : "Нет\n");
  std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------TECT 7: toLower()---------------------\n";
  std::cout << "Сначала пройдемся по русским словам.\n";
  finRussianWords.open("russianWords.txt");
  while(!finRussianWords.eof())
  {
    finRussianWords >> word;
    std::cout << "\"" << word << "\": ";
    toLower(word);
    std::cout << "\"" << word << "\"\n";
  }
  finRussianWords.close();

  std::cout << "Теперь по английским.\n";
  word = "DOG";
  std::cout << "\"" << word << "\": ";
  toLower(word);
  std::cout << "\"" << word << "\"\n";

  word = "fUnCtIoN";
  std::cout << "\"" << word << "\": ";
  toLower(word);
  std::cout << "\"" << word << "\"\n";

  word = "QuAlItY";
  std::cout << "\"" << word << "\": ";
  toLower(word);
  std::cout << "\"" << word << "\"\n";

  std::cout <<"Теперь проверим возникновение исключеня, если передать функции не
слово.\n";
  word = "Hel!@@#!lo";
  std::cout << "\"" << word << "\": ";
  try
  {
    toLower(word);
    std::cout << "\"" << word << "\"\n";
  }
```

```
  catch (const std::exception& error)
  {
    std::cout << error.what() << "\n";
  }

std::cout << "-----------------------------------------------------\n";

  std::cout << "----------------ТЕСТ 8: quickSort() и printVector()----\n";
  std::vector< std::pair< std::string, size_t > > vec;
  std::cout << "Пустой вектор до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Пустой вектор после сортировки:\n";
  printVector(vec, std::cout);

  vec.push_back(std::make_pair("test", 1));
  std::cout << "Вектор с одним элементом до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Вектор с одним элементом после сортировки:\n";
  printVector(vec, std::cout);

vec.push_back(std::make_pair("programmer", 42));
  std::cout << "Вектор с двумя элементами до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Вектор с двумя элементами после сортировки:\n";
  printVector(vec, std::cout);

  vec.push_back(std::make_pair("deadline", 422));
  std::cout << "Вектор с тремя элементами до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Вектор с тремя элементами после сортировки:\n";
  printVector(vec, std::cout);

  vec.clear();
  vec.push_back(std::make_pair("test", 422));
  vec.push_back(std::make_pair("student", 300));
  vec.push_back(std::make_pair("programmer", 150));
  vec.push_back(std::make_pair("teacher", 43));
  vec.push_back(std::make_pair("language", 2));
  std::cout << "Лучший случай до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Лучший случай после сортировки:\n";
  printVector(vec, std::cout);

  vec.clear();
  vec.push_back(std::make_pair("test", 2));
  vec.push_back(std::make_pair("student", 42));
  vec.push_back(std::make_pair("programmer", 34));
  vec.push_back(std::make_pair("teacher", 150));
  vec.push_back(std::make_pair("language", 1));
  std::cout << "Средний случай до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Средний случай после сортировки:\n";
  printVector(vec, std::cout);

  vec.clear();
  vec.push_back(std::make_pair("test", 1));
  vec.push_back(std::make_pair("student", 23));
```

```
  vec.push_back(std::make_pair("programmer", 242));
  vec.push_back(std::make_pair("teacher", 1234));
  vec.push_back(std::make_pair("language", 4444));
  std::cout << "Худший случай до сортировки:\n";
  printVector(vec, std::cout);
  quickSort(vec, 0, vec.size() - 1);
  std::cout << "Худший случай после сортировки:\n";
  printVector(vec, std::cout);
  std::cout << "-----------------------------------------------------\n";
  std::cout << "-------------------------------------------------------------------------------
----------------------------\n";
}

void doFrequencyDictionaryTest()
{
  std::cout << "--------------------------------ТЕСТИРОВАНИЕ ЧАСТОТНОГО СЛОВАРЯ---------
----------------------------\n";
  std::cout << "Перед началом тестов хочу отметить, что функции fillVector() и
getThreeMost()\n"
            << "тестируются неявно, так как они являются ключевыми в printSorted() и
printThreeMost().\n";
  std::cout << "------------ТЕСТ 1: ТАБЛИЦА С НУЛЕВЫМ РАЗМЕРОМ--------\n";
  try
  {
    FrequencyDictionary zeroSizedDictionary(0);
  }
  catch(const std::exception& error)
  {
    std::cout << error.what() << "\n";
  }
  std::cout << "-----------------------------------------------------\n";

  std::cout << "------------ТЕСТ 2: ПУСТОЙ СЛОВАРЬ---------------------\n";
  FrequencyDictionary dictionary;
  std::cout << "Размер пустого словаря: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в пустом словаре: " << dictionary.count() << "\n";
  std::cout << "Пустой словарь в неотсортированном виде:\n";
  dictionary.printUnsorted(std::cout);
  std::cout << "Пустой словарь в отсортированном виде:\n";
  dictionary.printSorted(std::cout);
  std::cout << "Три чаще всего встречающихся слова пустого словаря:\n";
  dictionary.printThreeMost(std::cout);

  std::cout << "Слово \"test\" встречается в списке " << dictionary.searchWord("test") <<
" раз.\n";

  std::cout << "Удаление слова \"test\": " << (dictionary.deleteWord("test") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "-----------------------------------------------------\n";

  std::cout << "------------ТЕСТ 3: ДОБАВЛЕНИЕ, ВЫВОД И ПОИСК----------\n";
  std::cout << "Вручную я буду добавлять только английские слова из-за кодировки IDE.\n";
  dictionary.insertWord("test");
  dictionary.insertWord("test");
  dictionary.insertWord("student");
  dictionary.insertWord("student");
  dictionary.insertWord("student");
  dictionary.insertWord("student");
  dictionary.insertWord("teacher");
  dictionary.insertWord("teacher");
  dictionary.insertWord("teacher");
  dictionary.insertWord("teacher");
  dictionary.insertWord("teacher");
```

```
  dictionary.insertWord("deadline");
  std::cout << "Размер словаря: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре: " << dictionary.count() << "\n";
  std::cout << "Словарь в неотсортированном виде:\n";
  dictionary.printUnsorted(std::cout);
  std::cout << "Словарь в отсортированном виде:\n";
  dictionary.printSorted(std::cout);
  std::cout << "Три чаще всего встречающихся слова:\n";
  dictionary.printThreeMost(std::cout);
  std::cout << "Слово \"test\" встречается в списке " << dictionary.searchWord("test") <<
" раз.\n";
  std::cout << "Слово \"teacher\" встречается в списке " <<
dictionary.searchWord("teacher") << " раз.\n";
  std::cout << "------------------------------------------------------\n";

  std::cout << "------------ТЕСТ 4: УДАЛЕНИЕ СЛОВ----------------------\n";
  std::cout << "Размер словаря до удаления: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре до удаления: " << dictionary.count() <<
"\n";
  std::cout << "Словарь в отсортированном виде до удаления:\n";
  dictionary.printSorted(std::cout);

  std::cout << "Слово \"test\" встречается в списке " << dictionary.searchWord("test") <<
" раз.\n";
  std::cout << "Удаление слова \"test\": " << (dictionary.deleteWord("test") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "Удаление слова \"deadline\": " << (dictionary.deleteWord("deadline") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "Слово \"test\" встречается в списке " << dictionary.searchWord("test") <<
" раз.\n";

  std::cout << "Словарь в отсортированном виде после удаления:\n";
  dictionary.printSorted(std::cout);
  std::cout << "Размер словаря после удаления: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре после удаления: " << dictionary.count() <<
"\n";
  std::cout << "Удаление слова \"student\": " << (dictionary.deleteWord("student") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "Удаление слова \"teacher\": " << (dictionary.deleteWord("teacher") ?
"произошло." : "не произошло.") << "\n";
  std::cout << "Словарь в отсортированном виде после удаления:\n";
  dictionary.printSorted(std::cout);
  std::cout << "Размер словаря после удаления: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре после удаления: " << dictionary.count() <<
"\n";
  std::cout << "------------------------------------------------------\n";

std::cout << "------------ТЕСТ 5: ЧТЕНИЕ ФАЙЛА----------------------\n";
  std::cout << "В качестве примера будет прочитана первая глава Капитанской дочки
Пушкина.\n";
  std::cout << "Скорее всего, тесты будут запускаться на linux терминале, где не видна
кодировка русских слов словаря.\n";
  std::cout << "В связи с этим на консоль я выведу только три чаще всего встречающихся
слова, "
            << "а весь словарь в отсортированном виде выведу в файл sampleOutput.txt.\n";
  dictionary.readFile("sampleInput.txt");
  std::cout << "Размер словаря: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре: " << dictionary.count() << "\n";
  std::cout << "Три чаще всего встречающихся слова:\n";
  dictionary.printThreeMost(std::cout);
  std::ofstream fout("sampleOutput.txt");
  dictionary.printSorted(fout);
  fout.close();
```

```
  std::cout << "Словарь в отсортированном виде напечатан в файл sampleOutput.txt\n";

  std::cout << "Теперь проверим возникновение исключения при попытке открыть
несуществующий файл.\n";
  try
  {
    dictionary.readFile("awdadadwa.awdawdadaw");
  }
  catch(const std::exception& error)
  {
    std::cout << error.what() << "\n";
  }
  std::cout << "-----------------------------------------------------\n";

  std::cout << "-----------ТЕСТ 6: ОЧИЩЕНИЕ СЛОВАРЯ-----------------\n";
  std::cout << "Размер словаря до очищения: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре до очищения: " << dictionary.count() <<
"\n";
  std::cout << "Очищение словаря.\n";
  dictionary.clear();
  std::cout << "Размер словаря после очищения: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре после очищения: " << dictionary.count() <<
"\n";
  std::cout << "-----------------------------------------------------\n";
  std::cout << "-----------ТЕСТ 7: КОЛЛИЗИИ-------------------------\n";
  std::cout << "Проверим возникновение коллизий в словаре\n";
  std::cout << "Изначально имеем пустой словарь.\n";
  std::cout << "Размер словаря: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре: " << dictionary.count() << "\n";
  std::cout << "Количество коллизий в словаре: " << dictionary.collisions() << "\n";
  std::cout << "Добавим в словарь 1 элемент.\n";
  dictionary.insertWord("test");
  std::cout << "Размер словаря: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре: " << dictionary.count() << "\n";
  std::cout << "Количество коллизий в словаре: " << dictionary.collisions() << "\n";
  std::cout << "Прочитаем в словарь наш файл\n";
  dictionary.readFile("sampleInput.txt");
  std::cout << "Размер словаря: " << dictionary.size() << "\n";
  std::cout << "Количество элементов в словаре: " << dictionary.count() << "\n";
  std::cout << "Количество коллизий в словаре: " << dictionary.collisions() << "\n";
  std::cout << "Создадим словарь размера 1000 и прочитаем в него наш файл\n";
  FrequencyDictionary bigDictionary(1000);
  bigDictionary.readFile("sampleInput.txt");
  std::cout << "Размер словаря: " << bigDictionary.size() << "\n";
  std::cout << "Количество элементов в словаре: " << bigDictionary.count() << "\n";
  std::cout << "Количество коллизий в словаре: " << bigDictionary.collisions() << "\n";
  std::cout << "-----------------------------------------------------\n";
  std::cout << "-----------------------------------------------------------------------
----------------------------\n";
}
```

main.cpp

```
#include <iostream>

#include "frequency-dictionary.hpp"

int main()
{
  std::cout << "-----------------------------------------------------\n"
            << "Вы в главном меню программы, выберите действие:\n"
            << "\"1\" - начать работу с программой\n"
            << "\"2\" - прогнать программу по тестам\n"
            << "\"exit\" - выйти из программы\n"
```

```
            << "-----------------------------------------------------\n";
  std::string input = "";
  while (true)
  {
    std::cin >> input;
    if (input == "1")
    {
      startProgram();
      break;
    }
    else if (input == "2")
    {
      testProgram();
      break;
    }
    else if (input == "exit")
    {
      break;
    }
    else
    {
      std::cout << "Некорректная команда, попробуйте еще раз.\n";
    }
  }

  return 0;
}
```

russianLetters.txt

```
АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюя
```

russianWords.txt

```
тест
прогРАММист
ДЕДлайН
стуДент
преподаватель
ЯЗЫК
```

sampleInput.txt

Представлен в файлах программы.

sampleOutput.txt

Представлен в файлах программы.

Приложение 2. Протоколы отладки

```
./app
-----------------------------------------------------
Вы в главном меню программы, выберите действие:
"1" - начать работу с программой
"2" - прогнать программу по тестам
"exit" - выйти из программы
-----------------------------------------------------
```

Тесты

```
-----------------------------------------------------
Вы в главном меню программы, выберите действие:
"1" - начать работу с программой
"2" - прогнать программу по тестам
"exit" - выйти из программы
-----------------------------------------------------
2
---------------------------------ТЕСТИРОВАНИЕ ДВУСВЯЗНОГО СПИСКА---------------------------------
-----------------ТЕСТ 1: ПУСТОЙ СПИСОК-----------------
Количество элементов: 0
Сам список:
Слово "test" встречается в списке 0 раз.
Удаление слова "test": не произошло.
Количество элементов: 0
Сам список:
Вектор, заполненный пустым списокм:
Вектор трех чаще всего встречающихся слов, заполненный пустым списокм:
: 0
: 0
: 0
-----------------------------------------------------
----------------ТЕСТ 2: ДОБАВЛЕНИЕ ЭЛЕМЕНТОВ И ПОИСК--
Количество элементов: 4
Сам список:
teacher: 1
student: 3
programming: 2
test: 5
```

```
test: 5
Слово "test" встречается в списке 5 раз.
Слово "deadline" встречается в списке 0 раз.
------------------------------------------------------
-----------------ТЕСТ 3: УДАЛЕНИЕ ЭЛЕМЕНТОВ------------
Список до удаления элементов:
Количество элементов: 4
teacher: 1
student: 3
programming: 2
test: 5
Удаление слова "test": произошло.
Удаление слова "deadline": не произошло.
Удаление слова "student": произошло.
Удаление слова "test": не произошло.
Список после удаления элементов:
Количество элементов: 2
teacher: 1
programming: 2
------------------------------------------------------
-----------------ТЕСТ 4: ЗАПОЛНЕНИЕ ВЕКТОРОВ------------
Сам список:
Количество элементов: 5
law: 1
language: 5
test: 5
teacher: 1
programming: 2
Вектор, заполненный списком:
law: 1
```

```
law: 1
language: 5
test: 5
teacher: 1
programming: 2
Вектор, заполненный списком:
law: 1
language: 5
test: 5
teacher: 1
programming: 2
Вектор трех чаще всего встречающихся слов:
language: 5
test: 5
programming: 2
Вектор, заполненный пустым списокм:
------------------------------------------------------
----------------ТЕСТ 5: ОЧИСТКА СПИСКА-----------------
Список до очкстки:
Количество элементов: 5
law: 1
language: 5
test: 5
teacher: 1
programming: 2
Список после очкстки:
Количество элементов: 0
------------------------------------------------------
-------------------------------------------------------------------------------------------------
--------------------------------ТЕСТИРОВАНИЕ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ--------------------------------
```

```
--------------------------------ТЕСТИРОВАНИЕ ВСПОМОГАТЕЛЬНЫХ ФУНКЦИЙ--------------------------------
---------------ТЕСТ 1: getCharCode()----------------
Русские буквы:
а:  192, б:  193, в:  194, г:  195, д:  196, е:  197, ё:  168, ж:  198,
з:  199, и:  200, й:  201, к:  202, л:  203, м:  204, н:  205,
о:  206, п:  207, р:  208, с:  209, т:  210, у:  211, ф:  212,
х:  213, ц:  214, ч:  215, ш:  216, щ:  217, ъ:  218, ы:  219,
ь:  220, э:  221, ю:  222, я:  223, А:  224, Б:  225, В:  226,
Г:  227, Д:  228, Е:  229, Ж:  184, З:  230, И:  231, Й:  232,
К:  233, Л:  234, М:  235, Н:  236, О:  237, П:  238, Р:  239,
С:  240, Т:  241, У:  242, Ф:  243, Х:  244, Ц:  245, Ч:  246,
Ш:  247, Щ:  248, Ъ:  249, Ы:  250, Ь:  251, Э:  252, Ю:  253,
Я:  254, Ё:  255, Английские буквы:
A:   65, B:   66, C:   67, D:   68, E:   69, F:   70, G:   71, H:   72,
I:   73, J:   74, K:   75, L:   76, M:   77, N:   78, O:   79,
P:   80, Q:   81, R:   82, S:   83, T:   84, U:   85, V:   86,
W:   87, X:   88, Y:   89, Z:   90, a:   97, b:   98, c:   99,
d:  100, e:  101, f:  102, g:  103, h:  104, i:  105, j:  106,
k:  107, l:  108, m:  109, n:  110, o:  111, p:  112, q:  113,
r:  114, s:  115, t:  116, u:  117, v:  118, w:  119, x:  120,
y:  121, z:  122, Случайные символы:
!:   33, @:   64, #:   35, $:   36, %:   37, ^:   94, &:   38, *:   42,
(:   40, ):   41, _:   95, -:   45, +:   43, =:   61,
------------------------------------------------------
---------------ТЕСТ 2: hashByDivision()--------------
Пусть размер таблицы - простое число 23.
Слово "test": 2
Слово "programmer": 6
Слово "deadline": 7
Слово "student": 11
```

```
Слово "student": 11
Слово "teacher": 18
Слово "language": 4
Еще раз (чтобы проверить неизменность хешей).
Слово "test": 2
Слово "programmer": 6
Слово "deadline": 7
Слово "student": 11
Слово "teacher": 18
Слово "language": 4
А теперь пусть размер таблицы - простое число 137.
Слово "test": 26
Слово "programmer": 135
Слово "deadline": 48
Слово "student": 66
Слово "teacher": 68
Слово "language": 45
Еще раз (чтобы проверить неизменность хешей).
Слово "test": 26
Слово "programmer": 135
Слово "deadline": 48
Слово "student": 66
Слово "teacher": 68
Слово "language": 45
------------------------------------------------------
---------------ТЕСТ 3: isRussianLetter()--------------
Пройдемся по всем русским буквам.
а: Да, б: Да, в: Да, г: Да, д: Да, е: Да, ё: Да, ж: Да,
з: Да, и: Да, й: Да, к: Да, л: Да, м: Да, н: Да,
о: Да, п: Да, р: Да, с: Да, т: Да, у: Да, ф: Да,
```

```
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да, ◆: Да,
◆: Да, ◆: Да,
Q: Нет
r: Нет
!: Нет
 : Нет
-------------------------------------------------------
----------------ТЕСТ 4: isEnglishLetter()-------------
Пройдемся по всем английским буквам.
A: Да, B: Да, C: Да, D: Да, E: Да, F: Да, G: Да, H: Да,
I: Да, J: Да, K: Да, L: Да, M: Да, N: Да, O: Да,
P: Да, Q: Да, R: Да, S: Да, T: Да, U: Да, V: Да,
W: Да, X: Да, Y: Да, Z: Да, a: Да, b: Да, c: Да,
d: Да, e: Да, f: Да, g: Да, h: Да, i: Да, j: Да,
k: Да, l: Да, m: Да, n: Да, o: Да, p: Да, q: Да,
r: Да, s: Да, t: Да, u: Да, v: Да, w: Да, x: Да,
y: Да, z: Да,
;: Нет
-: Нет
!: Нет
 : Нет
-------------------------------------------------------
----------------ТЕСТ 5: clearWord()--------------------
```

```
----------------ТЕСТ 5: clearWord()--------------------
"Hello,": "Hello"
"@#@q&^w&^&*e&^&*r^%&t^&%&^y#@#": "qwerty"
"O0neMo0reTest": "OneMoreTest"
-------------------------------------------------------
----------------ТЕСТ 6: isWord()-----------------------
Сначала пройдемся по случайным русским словам.
"◆◆◆◆": Да
"◆◆◆◆◆◆◆◆◆◆◆": Да
"◆◆◆◆◆◆◆◆": Да
"◆◆◆◆◆◆◆◆": Да
"◆◆◆◆◆◆◆◆◆◆◆◆◆": Да
"◆◆◆◆": Да
Теперь по английским.
"dog": Да
"function": Да
"quality": Да
"Member": Да
"WeATHeR": Да
Теперь по НЕ словам.
"do!g": Нет
"@function": Нет
"quality#": Нет
"$@$@$@": Нет
-------------------------------------------------------
----------------ТЕСТ 7: toLower()----------------------
Сначала пройдемся по русским словам.
"◆◆◆◆": "◆◆◆◆"
"◆◆◆◆◆◆◆◆◆◆◆": "◆◆◆◆◆◆◆◆◆◆◆"
"◆◆◆◆◆◆◆◆": "◆◆◆◆◆◆◆◆"
```

```
"◆◆◆◆◆◆◆": "◆◆◆◆◆◆◆"
"◆◆◆◆◆◆◆": "◆◆◆◆◆◆◆"
"◆◆◆◆◆◆◆◆◆◆◆◆◆": "◆◆◆◆◆◆◆◆◆◆◆◆◆"
"◆◆◆◆": "◆◆◆◆"
Теперь по английским.
"DOG": "dog"
"fUnCtIoN": "function"
"QuAlItY": "quality"
Теперь проверим возникновение исключеня, если передать функции не слово.
"Hel!@@#!lo": На вход поступило не слово!
-----------------------------------------------------
----------------ТЕСТ 8: quickSort() и printVector()----
Пустой вектор до сортировки:
Пустой вектор после сортировки:
Вектор с одним элементом до сортировки:
test: 1
Вектор с одним элементом после сортировки:
test: 1
Вектор с двумя элементами до сортировки:
test: 1
programmer: 42
Вектор с двумя элементами после сортировки:
programmer: 42
test: 1
Вектор с тремя элементами до сортировки:
programmer: 42
test: 1
deadline: 422
Вектор с тремя элементами после сортировки:
deadline: 422
```

```
deadline: 422
programmer: 42
test: 1
Лучший случай до сортировки:
test: 422
student: 300
programmer: 150
teacher: 43
language: 2
Лучший случай после сортировки:
test: 422
student: 300
programmer: 150
teacher: 43
language: 2
Средний случай до сортировки:
test: 2
student: 42
programmer: 34
teacher: 150
language: 1
Средний случай после сортировки:
teacher: 150
student: 42
programmer: 34
test: 2
language: 1
Худший случай до сортировки:
test: 1
student: 23
```

```
student: 23
test: 1
-------------------------------------------------------
-------------------------------------------------------------------------------------------------------
--------------------------------ТЕСТИРОВАНИЕ ЧАСТОТНОГО СЛОВАРЯ--------------------------------
Перед началом тестов хочу отметить, что функции fillVector() и getThreeMost()
тестируются неявно, так как они являются ключевыми в printSorted() и printThreeMost().
------------ТЕСТ 1: ТАБЛИЦА С НУЛЕВЫМ РАЗМЕРОМ--------
Размер хеш таблицы должен быть положительным числом!
-------------------------------------------------------
------------ТЕСТ 2: ПУСТОЙ СЛОВАРЬ---------------------
Размер пустого словаря: 98299
Количество элементов в пустом словаре: 0
Пустой словарь в неотсортированном виде:
Пустой словарь в отсортированном виде:
Три чаще всего встречающихся слова пустого словаря:
: 0
: 0
: 0
Слово "test" встречается в списке 0 раз.
Удаление слова "test": не произошло.
-------------------------------------------------------
------------ТЕСТ 3: ДОБАВЛЕНИЕ, ВЫВОД И ПОИСК----------
Вручную я буду добавлять только английские слова из-за кодировки IDE.
Размер словаря: 98299
Количество элементов в словаре: 4
Словарь в неотсортированном виде:
test: 2
teacher: 5
deadline: 1
```

```
deadline: 1
student: 4
Словарь в отсортированном виде:
teacher: 5
student: 4
test: 2
deadline: 1
Три чаще всего встречающихся слова:
teacher: 5
student: 4
test: 2
Слово "test" встречается в списке 2 раз.
Слово "teacher" встречается в списке 5 раз.
-------------------------------------------------------
------------ТЕСТ 4: УДАЛЕНИЕ СЛОВ----------------------
Размер словаря до удаления: 98299
Количество элементов в словаре до удаления: 4
Словарь в отсортированном виде до удаления:
teacher: 5
student: 4
test: 2
deadline: 1
Слово "test" встречается в списке 2 раз.
Удаление слова "test": произошло.
Удаление слова "deadline": произошло.
Слово "test" встречается в списке 0 раз.
Словарь в отсортированном виде после удаления:
teacher: 5
student: 4
Размер словаря после удаления: 98299
```

```
Размер словаря после удаления: 98299
Количество элементов в словаре после удаления: 2
Удаление слова "student": произошло.
Удаление слова "teacher": произошло.
Словарь в отсортированном виде после удаления:
Размер словаря после удаления: 98299
Количество элементов в словаре после удаления: 0
------------------------------------------------------
------------ТЕСТ 5: ЧТЕНИЕ ФАЙЛА----------------------
В качестве примера будет прочитана первая глава Капитанской дочки Пушкина.
Скорее всего, тесты будут запускаться на linux терминале, где не видна кодировка русских слов словаря.
В связи с этим на консоль я выведу только три чаще всего встречающихся слова, а весь словарь в отсортированном виде выве
ду в файл sampleOutput.txt.
Размер словаря: 98299
Количество элементов в словаре: 1170
Три чаще всего встречающихся слова:
◆: 93
◆: 72
◆: 53
Словарь в отсортированном виде напечатан в файл sampleOutput.txt
Теперь проверим возникновение исключения при попытке открыть несуществующий файл.
Ошибка открытия файла!
------------------------------------------------------
------------ТЕСТ 6: ОЧИЩЕНИЕ СЛОВАРЯ------------------
Размер словаря до очищения: 98299
Количество элементов в словаре до очищения: 1170
Очищение словаря.
Размер словаря после очищения: 98299
Количество элементов в словаре после очищения: 0
------------------------------------------------------
```

```
Ошибка открытия файла!
------------------------------------------------------
------------ТЕСТ 6: ОЧИЩЕНИЕ СЛОВАРЯ------------------
Размер словаря до очищения: 98299
Количество элементов в словаре до очищения: 1170
Очищение словаря.
Размер словаря после очищения: 98299
Количество элементов в словаре после очищения: 0
------------------------------------------------------
------------ТЕСТ 7: КОЛЛИЗИИ--------------------------
Проверим возникновение коллизий в словаре
Изначально имеем пустой словарь.
Размер словаря: 98299
Количество элементов в словаре: 0
Количество коллизий в словаре: 0
Добавим в словарь 1 элемент.
Размер словаря: 98299
Количество элементов в словаре: 1
Количество коллизий в словаре: 0
Прочитаем в словарь наш файл
Размер словаря: 98299
Количество элементов в словаре: 1171
Количество коллизий в словаре: 25
Создадим словарь размера 1000 и прочитаем в него наш файл
Размер словаря: 1000
Количество элементов в словаре: 1170
Количество коллизий в словаре: 328
------------------------------------------------------
------------------------------------------------------------------------------------------------------
```

Сама программа

```
-----------------------------------------------------
Вы в главном меню программы, выберите действие:
"1" - начать работу с программой
"2" - прогнать программу по тестам
"exit" - выйти из программы
-----------------------------------------------------
1
-----------------------------------------------------
Началось выполнение программы
"help" - посмотреть полный список команд
"exit" - выйти из программы
"insert" - добавить слово
"search" - найти слово
"delete" - удалить слово
"readFile" - добавить все слова из файла
"print" - вывести весь словарь
"printThreeMost" - вывести три чаще всего встречающихся слова
"writeFile" - записать в файл слова по убыванию их встречаемости
"clear" - очистить словарь
-----------------------------------------------------
Введите команду:
```

```
-----------------------------------------------------
Введите команду: help
-----------------------------------------------------
Список команд:
"help" - посмотреть полный список команд
"exit" - выйти из программы
"insert" - добавить слово
"search" - найти слово
"delete" - удалить слово
"readFile" - добавить все слова из файла
"print" - вывести весь словарь
"printThreeMost" - вывести три чаще всего встречающихся слова
"writeFile" - записать в файл слова по убыванию их встречаемости
"clear" - очистить словарь
-----------------------------------------------------
Введите команду:
```

```
Введите команду: insert
Введите слово, которое хотите добавить: test
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: test
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: test
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: student
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: double
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: double
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: qwe2e^&* dsa786*
Ошибка: Некорректное слово!
Введите команду:
```

```
Введите команду: search
Введите слово, которое хотите найти: test
Слово "test" встречается 3 раз.
Введите команду: search
Введите слово, которое хотите найти: empty
Слово "empty" встречается 0 раз.
Введите команду: search
Введите слово, которое хотите найти: qwe
Слово "qwe" встречается 0 раз.
Введите команду:
```

```
Введите команду: search
Введите слово, которое хотите найти: test
Слово "test" встречается 3 раз.
Введите команду: delete
Введите слово, которое хотите удалить:
test
Слово успешно удалено.
Введите команду: search
Введите слово, которое хотите найти: test
Слово "test" встречается 0 раз.
Введите команду: delete 7^DS^7d^AD&S*
Ошибка: Некорректная команда!
Введите команду:
```

```
--------------------------------------------------
Введите команду: print
double: 2
student: 1
Введите команду:
```

```
Введите команду: insert
Введите слово, которое хотите добавить: qwe
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: qwe
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: qwe
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: qwe
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: abc
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: abc
Успешно.
Введите команду: insert
Введите слово, которое хотите добавить: abc
Успешно.
Введите команду: print
qwe: 4
abc: 3
double: 2
student: 1
Введите команду:
```

```
Введите команду: printThreeMost
Три чаще всего встречающихся слова:
qwe: 4
abc: 3
double: 2
Введите команду:
```

```
Введите команду: clear
Словарь успешно очищен.
Введите команду: print
Введите команду:
```

```
Введите команду: readFile
Введите имя файла, который хотите прочитать: sampleInput.txt
Успешно.
Введите команду: writeFile
Введите имя файла, в который хотите записать: sampleOutput.txt
Успешно.
Введите команду:
```