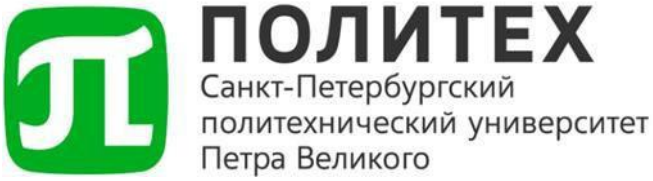
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии



К У Р С О В А Я Р А Б О Т А

Алгоритмы работы со словарями

по дисциплине «**Алгоритмы и структуры данных**»

Выполнил Рулев Г. А.
студент гр. 3530904–00006

Руководитель Павлов Е. А.

«___» _____ 2021

Санкт-Петербург

2021 г

Оглавление

Постановка задачи 3
1. Описание алгоритма решения и используемых структур данных 4
Подробнее: 4
2. Анализ алгоритма 9
3. Описание спецификации программы (детальные требования) 10
4. Описание программы (структура программы, форматы входных и выходных данных) 12
Заключение 14
Список литературы 15
Приложение 1. Текст программы (по стандарту кодирования) 16
Приложение 2. Протоколы отладки 39

Англо-русский словарь. Бинарное дерево поиска

Постановка задачи

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

Предусмотреть обработку и инициализацию исключительных ситуаций связанных, например, с проверкой значения полей перед инициализацией и присваиванием.

Программа должна быть написана в соответствии со стандартом программирования: C++ ProgrammingStyleGuidelines(http://geosoft.no/development/cppstyle.html).

Тесты должны учитывать как допустимые, так и не допустимые последовательности входных данных.

Разработать и реализовать алгоритм работы с англо-русским словарем,реализованным как бинарное дерево поиска.

Узел бинарного дерева поиска должен содержать:
Ключ –английское слово,
Информационная часть –ссылка на список, содержащий переводы английского слова, отсортированные по алфавиту (переводов слова может быть несколько).

1. Описание алгоритма решения и используемых структур данных

Словесное описание алгоритма решения:

Разработан **класс List** со специальным набором методов, реализует двусвязный список для информационной части (ссылка на список, содержащий переводы английского слова, отсортированные по алфавиту(переводов слова может быть несколько))

Разработан **класс EnglishRussianDictionary** со специальным набором методов, реализующий словарь на основе бинарного дерева поиска с ключевой структурой.

Подробнее:

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше,то в левом, если равно, то значение найдено и поиск прекращается.

Добавление узла. Есть два случая:

Дерево пустое.

Дерево не пустое.

Если дерево пустое, мы просто создаем новый узел и добавляем его в дерево. Во втором случае мы сравниваем переданное значение со значением в узле, начиная от корня. Если добавляемое значение меньше значения рассматриваемого узла, повторяем ту же процедуру для левого поддерева. В противном случае — для правого.

Поиск узла в дереве.

Допустим, у вас есть построенное дерево. Что бы найти элемент с ключом key нужно последовательно двигаться от корня вниз по дереву и сравнивать значение key с ключом очередного узла: если key меньше, чем ключ очередного узла, то перейти к левому потомку узла, если больше — к правому, если ключи равны — искомый узел найден.

Удаление узла из дерева. Алгоритм удаления элемента выглядит так:

Найти узел, который надо удалить.

Удалить его.

После того, как мы нашли узел, который необходимо удалить, у нас возможны три случая.

Случай 1: У удаляемого узла нет правого потомка.

В этом случае мы просто перемещаем левого потомка (при его наличии) наместо удаляемого узла.

Случай 2: У удаляемого узла есть только правый потомок, у которого, в своюочередь нет левого потомка.

В этом случае нам надо переместить правого потомка удаляемого узла на егоместо.

Случай 3: У удаляемого узла есть первый потомок, у которого есть левый потомок.

В этом случае место удаляемого узла занимает крайний левый потомок правого потомка удаляемого узла.

Разработано **консольное приложение** на основе специальных методов класса EnglishRussianDictionary, предоставляющее пользователю набор из 6 функций для работы со словарем.

Работа составлена с помощью ранее разработанных и тщательно протестированных классов.

Описание используемых структур данных:

Вспомогательный **класс List** – реализует двусвязный список для занесенияслов-переводов и включает в себя:

Подкласс Node – реализует узел списка и включает в себя:

Поле std::string data_ - хранит слово-перевод;

Поле Node* next_ и Node* prev_ - для связи узлов списка;Конструктор по умолчанию.

Поле Node* head_ - массив узлов;

Поле std::size_t size_ - количество узлов списка для подсчета слов-переводов;Конструктор по умолчанию;

Деструктор;

Метод std::size_t getSize() const – возвращает кол-во слов-переводов;

Метод List& deleteNode(const std::string data) – для удаления слова-перевода;

Метод List& operator+=(const std::string data) – для добавления слова-перевода, сортировка автоматическая по алфавиту, повторные не добавляются;

Метод friend std::ostream& operator<<(std::ostream& out, const List& List) – для вывода списка переводов;

Скрытый метод Node* searchNodeAndReturn(const std::string data) const – дляпоиска нужного узла.

Пример списков с переводами (подчеркнут один список)

```
call - звать, навещать, называть
ahead - вперёд, впереди
actually - на самом деле, фактически
alone - один, одинокий
break - ломать, разбивать
breakfast - завтрак
brother - брат
fuss - волноваться, суетиться
chilly - зябко, прохладный, холодно
calm - спокойный
fortnight - две недели
furious - взбешённый
wrap up - кутаться
wonderful - замечательный
zebra - зебра
```

Класс EnglishRussianDictionary – реализует англо-русский словарь на основе бинарного дерева поиска и включает в себя:

Подкласс Word – вершина дерева, реализует контейнер для хранения информации об английском слове и включает в себя:

Поле std::string key_ - для хранения английского слова;Поле List translation_ - для хранения списка переводов;

Поля Word* left_, Word* right_ и Word* p_ - для связи вершин дерева;

Конструктор по умолчанию.

Поле Word* root_ - массив вершин;

Поле std::size_t size_ - для объема словаря;Конструктор по умолчанию;

Деструктор;

Метод void insertWord(const std::string key, const std::string translation) – для добавления слова с переводом (при повторном добавлении с новым переводом лист пополняется еще одним переводом, повторные переводы не добавляются);

Метод void printWord(const std::string key) const - для вывода слова с переводом;

Метод void deleteWord(const std::string key) - для удаления слова со всей информацией о нем;

Метод void deleteTranslation(const std::string key, const std::string translation) - для удаления конкретного перевода у конкретного слова;

Метод void print() const - для вывода всего словаря;

Метод bool checkWord(const std::string word, const std::string language) const –для проверки слова на соответствие английскому/русскому языку (при наличии символов, цифр, букв из другого языка проверка не проходит);

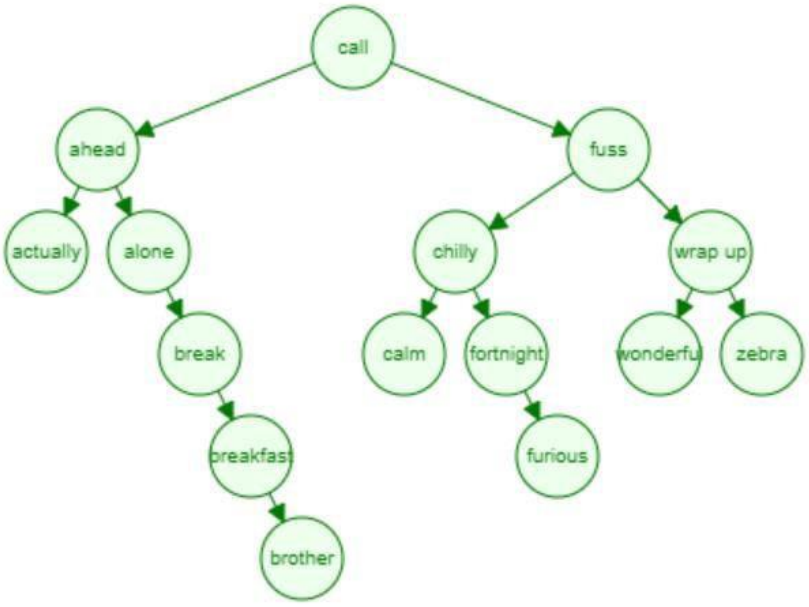
Скрытый метод Word* searchWordAndReturn(const std::string key) const – для поиска нужной вершины дерева, то есть контейнера в словаре;

Скрытый метод Word* searchSuccessorAndReturn(const std::string key) const - из логики дерева, для поиска преемника у вершины;

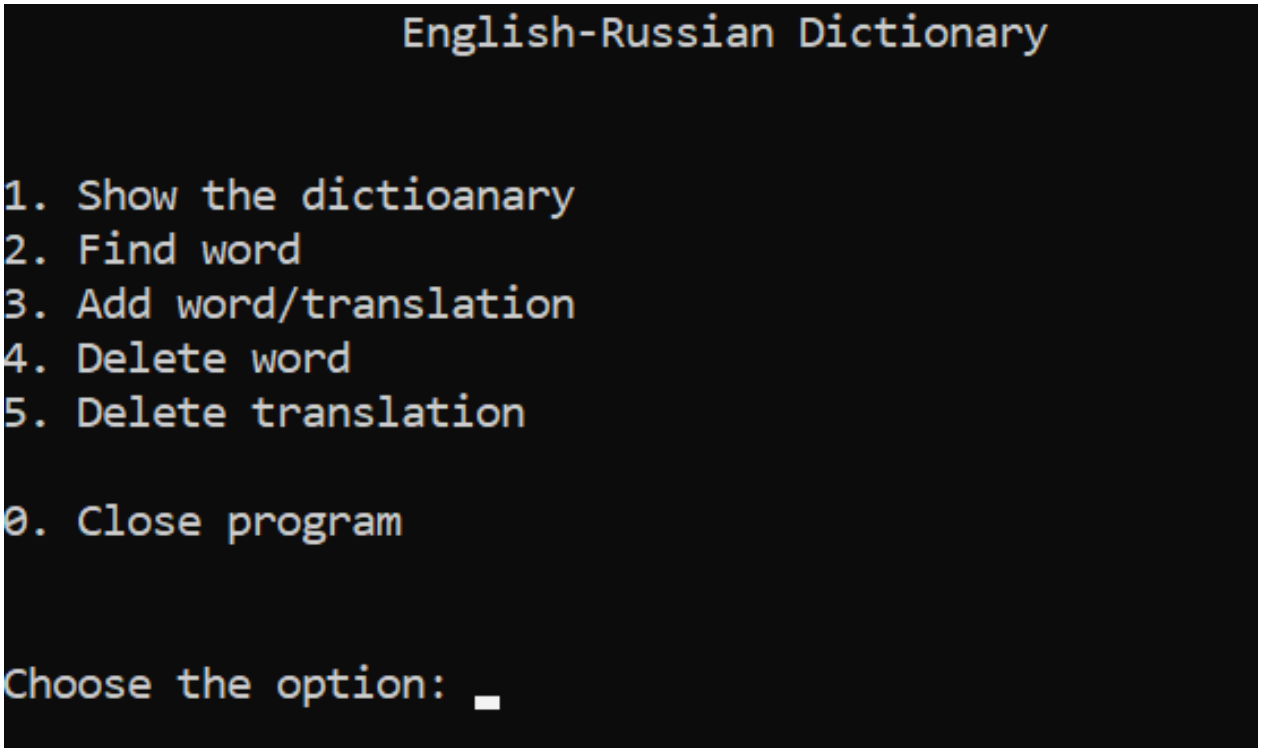
Скрытый метод void deleteDictionary(Word* word) - удаление словаря-дерева;

Скрытый метод void printWordsStartingWith(Word* word) const - вывод частисловаря от определенной вершины дерева.

Пример дерева-словаря:



Консольное приложение



Все возможные исключения учтены (удаление несуществующего слова/перевода, удаление единственного перевода, печать несуществующего слова, добавление несуществующему слову перевода, ввод некорректных слов и т.д.)

2. Анализ алгоритма

EnglishRussianDictionary::Word* EnglishRussianDictionary::searchWordAndReturn(const std::string key) const - сложность поиска O(log(n));

Эффективность поиска O(log(n)), причем логарифм по основанию 2. Если в дереве n элементов, то это значит, что будет log(n) по основанию 2 уровней дерева. А в поиске, за один шаг цикла, мы спускаемся на один уровень.

void EnglishRussianDictionary::insertWord(const std::string key, const std::stringtranslation) - сложность вставки такая же как и поиска - O(log(n));

oid EnglishRussianDictionary::deleteWord(const std::string key) - сложность удаления аппроксимирована к O(log(n)); [1]

3. Описание спецификации программы (детальные требования)

При запуске консольного приложения появляется стартовое меню,пользователь выбирает функцию под определенным номером

В случае ввода 1/2/3/4/5/0 срабатывает определенный алгоритм;

В случае ввода других символов программа возвращается в прежний вид.

Функция <1. Show the dictioanary>

Выводит весь словарь в формате <word_1 - перевод_1, перевод_2

word_2 - перевод

.....>;

В случае пустого словаря ничего не выводится.

Функция <2. Find word>

В случае ввода слова, которое есть в словаре, выводится “Result: <word - перевод_1, перевод_2>”;

В случае ввода слова, которого нет в словаре, выводится “Result: The word is
not found :(/ Слово не найдено :(“;

В случае ввода некорректного слова выводится ошибка “ Word must becorrect!\n(void EnglishRussianDictionary::printWord(const std::string key) const)”.

Функция <3. Add word/translation>

В случае ввода нового слова и перевода все добавляется в словарь ивыводится “Word and translation added!”;

В случае ввода старого слова и нового перевода, в словарь заносится ещеодин перевод данного слова и выводится "Word and translation added!";

В случае ввода старого слова и старого перевода ничего не меняется выводится “Word and translation added!”;

В случае ввода некорректного слова или перевода выводится ошибка “ Allwords must be correct!\n(void EnglishRussianDictionary::insertWord(const std::string key, const std::string translation))”.

Функция <4. Delete word>

В случае ввода слова, которое есть в словаре, данное слово удаляется изсловаря и выводится “Word deleted!”;

В случае ввода слова, которого нет в словаре, выводится ошибка “Word not found! (void English_Russian_Dictionary::delete_word(const std::string word))”;

В случае ввода некорректного слова выводится ошибка “ Word must be correct!\n(void EnglishRussianDictionary::deleteWord(const std::string key))”.

Функция <5. Delete translation>

В случае ввода слова, которое есть в словаре, и перевода, который есть у данного слова и является не единственным, удаляется данный перевод и выводится "Translation deleted!";

В случае ввода слова, которого нет в словаре, и любого перевода, генерируется исключение и выводится ошибка "Word not found! (void EnglishRussianDictionary::deleteTranslation(const std::string key, const std::string translation))";

В случае ввода слова, которое есть в словаре, и перевода, которого нет в словаре, генерируется исключение и выводится ошибка "Translation not found! (void English_Russian_Dictionary::delete_translation(const std::stringword, const std::string translation))";

В случае ввода слова, которое есть в словаре, и перевода, который есть у данного слова, но который является единственным, генерируется исключениеи выводится ошибка "This word has one translation! (void English_Russian_Dictionary::delete_translation(const std::string word, const std::string translation))";

В случае ввода некорректного слова или перевода выводится ошибка “ All words must be correct!\n(void EnglishRussianDictionary::deleteTranslation(conststd::string key, const std::string translation))”.

Функция <0. Close program>

Завершает программу и закрывает окно.

После выполнения функций 1–5 срабатывает пауза, и после нажатия любой клавиши программа возвращается в стартовое меню словаря.

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

Все функции программы исполняются на основе класса EnglishRussianDictionary и его методов. Выбор функции обрабатывается с помощью оператора switch и функции system, а полностью стартовое меню циклично выводится на экран. Все данные вводятся и выводятся в консоль; Корректные данные должны быть типа char (номер функции) или std::string (корректные/некорректные английские или русские слова).

Функция <1. Show the dictioanary> (входные данные из стартового меню: 1)основана на методе void EnglishRussianDictionary::print() const, в результате работы которого выводится текст вида: <word_1 - перевод_1, перевод_2

word_2 - перевод

.....>;

Функция <2. Find word> (входные данные из стартового меню: 2) выводит на экран текст "Enter a search word: ". После ввода слова для поиска (входныеданные вида: <word>) выводится "Result: " и результат обращения к методу void EnglishRussianDictionary::printWord(const std::string key) const - текст вида: <word - перевод_1, перевод_2> или "The word is not found :(/ Слово не найдено :(", либо ошибка “Word must be correct!\n(void EnglishRussianDictionary::printWord(const std::string key) const)”.

Функция <3. Add word/translation> (входные данные из стартового меню: 3) выводит на экран текст "Enter a word to add: ". После ввода слова для добавления (входные данные вида: <word>) выводится текст "Enter the translation: ". После ввода слова- перевода (входные данные вида: <перевод>) срабатывает метод void EnglishRussianDictionary::insertWord(const std::string key, const std::string translation), в результате работы которого либо добавляется новое слово с переводом, либо добавляется перевод к уже существующему в словаре слову, либо выводится ошибка “ All words must be correct!\n(void EnglishRussianDictionary::insertWord(const std::string key, const std::string translation))”. В конце работы функции выводится текст "Word and translation added!";

Функция <4. Delete word> (входные данные из стартового меню: 4) выводитна экран текст "Enter a word to delete: ". После ввода слова для уждаления (входные данные вида: <word>) срабатывает метод void EnglishRussianDictionary::deleteWord(const std::string key) выводится либо "Word deleted!", либо ошибка "Word not found! (void English_Russian_Dictionary::delete_word(const std::string word))", либо ошибка “Word must be correct!\n(void EnglishRussianDictionary::deleteWord(const std::string key))”;

Функция <5. Delete translation> (входные данные из стартового меню: 5) выводит на экран текст "Enter a word: ". После ввода слова, у которого удаляется перевод (входные данные вида: <word>) появляется сообщение "Enter the translation: ". После ввода слова-перевода (входные данные вида: <перевод>) срабатывает метод void EnglishRussianDictionary::deleteTranslation(const std::string key, const std::string translation) и выводится "Translation deleted!", либо ошибка "Word not found!\n(void EnglishRussianDictionary::deleteTranslation(const std::string key, const std::string translation))", либо ошибка "This word has one translation!\n(void English_Russian_Dictionary::delete_translation(const std::string word, const std::string translation))", либо ошибка "Translation not found!\n(void English_Russian_Dictionary::delete_translation(const std::string word, const std::string translation))", либо ошибка “All words must be correct!\n(void EnglishRussianDictionary::deleteTranslation(const std::string key, const std::string translation))”.

Функция <0. Close program> завершает программу с кодом 0 и закрываетокно.

Все ошибки выводятся в результате генерации и обработки исключений.

После окончания работы функций 1–5 срабатывает пауза, и после любогонажатия клавиши возврат в стартовое меню.

Заключение

В ходе работы мною были подробно изучены и реализованы в коде следующие понятия: двусвязный линейный список, бинарное дерево поиска, словарь. С помощью данной теории мною была разработана программа, реализующая англо-русский словарь.

Список литературы

1. Бинарные деревья поиска – URL: https://sohabr.net/habr/post/442352/ . – (дата обращения 19.05.2021).
2. Самочадина Т.Н. Алгоритмы и структуры данных: Лекция №3: Деревья. Основные понятия и определения. Бинарные деревья поиска (дата обращения 19.05.2021).

Приложение 1. Текст программы (по стандарту кодирования)

Файл main.cpp

```
#include <iostream>
#include <windows.h>
#include "en_ru_dict.hpp"
#include <string>
#include <sstream>


int main()
{

  setlocale(LC_ALL, "rus");
  system("chcp 1251 >> null");
  EnglishRussianDictionary ERD;

  try
  {
        createDictionary(ERD);

  }

  catch (const std::invalid_argument& ia)
  {

        std::cerr << ia.what(); return -1;
  }

  char option;

  while (true)
  {

        std::cout << "\t\tEnglish-Russian Dictionary\n\n\n1. Show the dictioanary\n2.
Find word\n3. Add word/translation\n4. Delete word\n5. Delete translation\n\n0.
Close program\n\n\nChoose the option: ";
        std::cin >> option;
        system("cls");

        switch (option)
        {
        case '1':
        {
          std::ofstream outFile;
```

```
  outFile.open("out.txt");
  ERD.print(outFile);
  outFile.close();
  system("pause >> void");
  system("cls");
  break;
}

case '2':
{

  std::string search_word;
  std::cout << "Enter a search word: ";
  std::cin >> search_word;

  std::cout << "\n\Check File! ";


  try
  {
        std::ofstream outFile;
        outFile.open("out.txt");
        ERD.printWord(search_word, outFile);
        outFile.close();
  }

  catch (const std::invalid_argument& ia1)
  {

        std::cerr << '\n' << ia1.what();
        system("pause >> void");
        system("cls");
        break;
  }


  system("pause >> void");
  system("cls");
  break;
}

case '3':
{

  std::string word_to_add;
  std::cout << "Enter a word to add: ";
  std::cin >> word_to_add;
```

```
  std::string translation;
  std::cout << "\n\nEnter the translation: ";
  getline(std::cin >> std::ws, translation);

  try
  {
        std::stringstream temp(translation);

        while (temp.peek() != EOF)
        {
          std::string word;
          temp >> word;
          if (word[word.size() - 1] == ',')
          {
                word.resize(word.size() - 1);
          }
          ERD.insertWord(word_to_add, word);
        }

  }

  catch (const std::invalid_argument& ia2)
  {

        std::cerr << '\n' << ia2.what();
        system("pause >> void");
        system("cls");
        break;
  }


  std::cout << "\n\nWord and translation added!";
  system("pause >> void");
  system("cls"); break;


}
case '4':
{

  std::string word_to_delete;
  std::cout << "Enter a word to delete: ";
  std::cin >> word_to_delete;

  try
  {

        ERD.deleteWord(word_to_delete);
```

```
  }

  catch (const std::invalid_argument& ia3)
  {

        std::cerr << '\n' << ia3.what();
        system("pause >> void");
        system("cls");
        break;
  }


  std::cout <<"\n\n Word /" << word_to_delete << "/ deleted!";
  system("pause >> void");
  system("cls");
  break;
}

case '5':
{

  std::string word;
  std::cout << "Enter a word: ";
  std::cin >> word;


  std::string translation_to_delete;
  std::cout << "\n\nEnter the translation to delete: ";
  std::cin >> translation_to_delete;

  try
  {

        ERD.deleteTranslation(word, translation_to_delete);
  }

  catch (const std::invalid_argument& ia4)
  {

        std::cerr << '\n' << ia4.what();
        system("pause >> void");
        system("cls");
        break;
  }


  std::cout << "\n\n Translation /" << translation_to_delete << "/ deleted!";
  system("pause >> void");
```

```
            system("cls");
            break;
          }

          case '0':
          {

            return 0;
          }

          default:
          {
            break;
          }
          }


  }


  return 0;
}
```

Файл en_ru_dict.hpp

```
#ifndef BINARY_SEARCH_TREE_HPP
#define BINARY_SEARCH_TREE_HPP

#include <iostream>
#include "double_direction_list.hpp"
#include<fstream>

class EnglishRussianDictionary
{
  class Word
  {
  public:
          std::string key_;
          List translation_;
          Word* left_;
          Word* right_;
          Word* p_;

          Word() :
            key_(""),
```

```
            translation_(List()),
            left_(nullptr),
            right_(nullptr),
            p_(nullptr)
          {
          }
  };

  Word* root_;
  std::size_t size_;

public:
  EnglishRussianDictionary();
  ~EnglishRussianDictionary();
  void insertWord(std::string key, std::string translation);
  void printWord(const std::string key, std::ostream& out) const;
  void deleteWord(const std::string key);
  void deleteTranslation(const std::string key, const std::string translation);
  void print(std::ostream& out) const;
  bool checkWord(const std::string word, const std::string language) const;
  friend void createDictionary(EnglishRussianDictionary& ERD);
  std::istream& skipSpaces(std::fstream& input);

private:
  Word* searchWordAndReturn(const std::string key) const;
  Word* searchSuccessorAndReturn(const std::string key) const;
  void deleteDictionary(Word* word);
  void printWordsStartingWith(Word* word, std::ostream& out) const;
};

#endif
```

Файл en_rus_dict.cpp

```
#include "en_ru_dict.hpp"
#include "double_direction_list.hpp"
#include <stdexcept>
#include<algorithm>
#include<iostream>
#include<string>
#include<fstream>
EnglishRussianDictionary::EnglishRussianDictionary():
  root_(nullptr),
  size_(0)
{
}
```

```
EnglishRussianDictionary::~EnglishRussianDictionary()
{
  deleteDictionary(root_);
}

void EnglishRussianDictionary::insertWord(std::string key, std::string translation)
{
  if ((checkWord(key, "English") && checkWord(translation, "Russian")) == false)
  {
        throw std::invalid_argument("All words must be correct!\n(void
EnglishRussianDictionary::insertWord(const std::string key, const std::string
translation))");
  }
  std::transform(key.begin(), key.end(), key.begin(), tolower);
  if (root_ == nullptr)
  {
        root_ = new Word();
        root_->key_ = key;
        root_->translation_ += translation;
  }
  else
  {
        Word* current_word = root_;
        while (true)
        {

          if (key < current_word->key_)
          {

                if (current_word->left_)
                {
                  current_word = current_word->left_;
                }
                else
                {
                  Word* new_word = new Word();
                  new_word->key_ = key;
                  new_word->translation_ += translation;
                  current_word->left_ = new_word;
                  new_word->p_ = current_word;
                  break;

                }
          }

          else if (key > current_word->key_)
          {
```

```
                if (current_word->right_)
                {

                  current_word = current_word->right_;
                }
                else
                {
                  Word* new_word = new Word();
                  new_word->key_ = key;
                  new_word->translation_ += translation;
                  current_word->right_ = new_word;
                  new_word->p_ = current_word;

                  break;

                }
          }

          else if (key == current_word->key_)
          {
                current_word->translation_ += translation;
                return;
          }
         }
  }
}


void EnglishRussianDictionary::printWord(const std::string key, std::ostream& out)
const
{
  if (checkWord(key, "English") == false)
  {
        throw std::invalid_argument("Word must be correct!\n(void
EnglishRussianDictionary::printWord(const std::string key) const)");
  }
  Word* current_word = searchWordAndReturn(key);
  if (current_word)
  {
        out << current_word->key_ << " - " << current_word -> translation_ << '\n';
  }
  else
  {
        out << "The word is not found :( / Слово не найдено :(";
  }

}
```

```
void EnglishRussianDictionary::deleteWord(const std::string key)
{

  if (checkWord(key, "English") == false)
  {
        throw std::invalid_argument("Word must be correct!\n(void
EnglishRussianDictionary::deleteWord(const std::string key))");
  }

  Word* current_word = searchWordAndReturn(key);

  if (current_word == nullptr)
  {
        throw std::invalid_argument("Word not found!\n(void
English_Russian_Dictionary::delete_word(const std::string word))");
  }

  //удаление листа
  if ((current_word->left_ == nullptr) && (current_word->right_ == nullptr))
  {
        if (current_word == root_)
        {
          delete root_;
          root_ = nullptr; //дает возможность работать с пустым деревом
        }
        else
        {
          if (current_word == current_word->p_->left_)
          {
                current_word->p_->left_ = nullptr;
          }

          if (current_word == current_word->p_->right_)
          {
                current_word->p_->right_ = nullptr;
          }

          delete current_word;
        }
  }
  //удаление узла с двумя потомками
  else if (current_word->left_ && current_word->right_)
  {
        Word* successor = searchSuccessorAndReturn(key);

        //разрыв связи преемника со своим старым родителем
        if (successor == successor->p_->left_)
```

```
{
  //передача детей преемника его родителю
  if (successor->right_)
  {
        successor->right_->p_ = successor->p_;
        successor->p_->left_ = successor->right_;
  }
  else if (successor->left_)
  {
        successor->left_->p_ = successor->p_;
        successor->p_->left_ = successor->left_;
  }
  else
  {
        successor->p_->left_ = nullptr;
  }
}

if (successor == successor->p_->right_)
{
  //передача детей преемника его родителю
  if (successor->right_)
  {
        successor->right_->p_ = successor->p_;
        successor->p_->right_ = successor->right_;
  }
  else if (successor->left_)
  {
        successor->left_->p_ = successor->p_;
        successor->p_->right_ = successor->left_;
  }
  else
  {
        successor->p_->right_ = nullptr;
  }
}

//новый родитель
successor->p_ = current_word->p_;

//новые дети будут дальше

//определение положения toDelete
if (current_word == root_)
{
  root_ = successor;
}
else
```

```
        {
          if (current_word == current_word->p_->left_)
          {
                current_word->p_->left_ = successor;
          }

          if (current_word == current_word->p_->right_)
          {
                current_word->p_->right_ = successor;
          }
        }

        //перенос потомков от toDelete к преемнику
        if (current_word->left_)
        {
          successor->left_ = current_word->left_;
          current_word->left_->p_ = successor;
        }

        if (current_word->right_)
        {
          successor->right_ = current_word->right_;
          current_word->right_->p_ = successor;
        }

        delete current_word;
}
//удаление узла с одним потомком
else if (current_word->left_)
{
        if (current_word == root_)
        {
          root_ = current_word->left_;
          root_->p_ = nullptr;
        }
        else
        {
          if (current_word == current_word->p_->left_)
          {
                current_word->p_->left_ = current_word->left_;
                current_word->left_->p_ = current_word->p_;
          }

          if (current_word == current_word->p_->right_)
          {
                current_word->p_->right_ = current_word->left_;
                current_word->left_->p_ = current_word->p_;
          }
```

```
        }

        delete current_word;
  }
  else if (current_word->right_)
  {
        if (current_word == root_)
        {
          root_ = current_word->right_;
          root_->p_ = nullptr;
        }
        else
        {
          if (current_word == current_word->p_->left_)
          {
                current_word->p_->left_ = current_word->right_;
                current_word->right_->p_ = current_word->p_;
          }

          if (current_word == current_word->p_->right_)
          {
                current_word->p_->right_ = current_word->right_;
                current_word->right_->p_ = current_word->p_;
          }
        }

        delete current_word;
  }
}

void EnglishRussianDictionary::deleteTranslation(const std::string key, const
std::string translation)
{
  if ((checkWord(key, "English") && checkWord(translation, "Russian")) == false)
  {
        throw std::invalid_argument("All words must be correct!\n(void
EnglishRussianDictionary::deleteTranslation(const std::string key, const std::string
translation))");
  }

  Word* current_word = searchWordAndReturn(key);
  if (current_word == nullptr)
  {
        throw std::invalid_argument("Word not found!\n(void
EnglishRussianDictionary::deleteTranslation(const std::string key, const std::string
translation))");
  }
```

```
  if (current_word->translation_.getSize() == 1)
  {
        throw std::invalid_argument("This word has one translation!\n(void
English_Russian_Dictionary::delete_translation(const std::string word, const
std::string translation))");
  }

  try
  {
        current_word->translation_.deleteNode(translation);
  }
  catch (...)
  {
        throw std::invalid_argument("Translation not found!\n(void
English_Russian_Dictionary::delete_translation(const std::string word, const
std::string translation))");
  }
}

void EnglishRussianDictionary::print(std::ostream& out) const
{
  printWordsStartingWith(root_, out);
}

bool EnglishRussianDictionary::checkWord(const std::string word, const std::string
language) const
{
  if (word.empty())
  {
        return false;
  }
  bool no_symbols = word.find_first_of("!@#№$;%:^&?*()-+=[]{}|/'.,") ==
std::string::npos;
  bool no_numbers = word.find_first_of("1234567890") == std::string::npos;
  if (language == "English")
  {
        bool ok_eng =
(word.find_first_of("абвгдеёжзийклмнопрстуфхцчшщъыьэюяАБВГДЕЁЖЗИЙК
ЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ") == std::string::npos);
        return no_symbols && no_numbers && ok_eng;
  }
  else if (language == "Russian")
  {
        bool ok_rus =
(word.find_first_of("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTU
VWXYZ") == std::string::npos);
        return no_symbols && no_numbers && ok_rus;
  }
```

```
  else
  {
        throw std::invalid_argument("I don't know that language :(\n(bool
EnglishRussianDictionary::checkWord(const std::string word, const std::string
language) const)");
  }

}

EnglishRussianDictionary::Word*
EnglishRussianDictionary::searchWordAndReturn(const std::string key) const
{

  Word* current_word = root_;

  if (current_word == nullptr)
  {
        return nullptr;
  }

  while (key != current_word->key_)
  {
        if (key < current_word->key_)
        {
          if (current_word->left_)
          {
                current_word = current_word->left_;
          }
          else
          {
                return nullptr;
          }
        }
        else if (key > current_word->key_)
        {
          if (current_word->right_)
          {
                current_word = current_word->right_;
          }
          else
          {
                return nullptr;
          }
        }
  }

  return current_word;
}
```

```
EnglishRussianDictionary::Word*
EnglishRussianDictionary::searchSuccessorAndReturn(const std::string key) const
{
  Word* current_word = searchWordAndReturn(key);

  if (current_word == nullptr)
  {
        throw std::invalid_argument("Argument can not be
nullptr!\n(English_Russian_Dictionary::Word*
English_Russian_Dictionary::search_successor_and_return(const std::string word)
const)");
  }

  if (current_word->right_)
  {
        current_word = current_word->right_;

        while (current_word->left_)
        {
          current_word = current_word->left_;
        }

        return current_word;
  }

  while (current_word->p_)
  {
        if (current_word == current_word->p_->left_)
        {
          return current_word->p_;
        }
        else
        {
          current_word = current_word->p_;
        }
  }

  return nullptr;
}

void EnglishRussianDictionary::deleteDictionary(Word* word)
{
  if (word)
  {
        if (word->right_)
        {
          deleteDictionary(word->right_);
```

```
        }

        if (word->left_)
        {
          deleteDictionary(word->left_);
        }
  }

  delete word;
}

void EnglishRussianDictionary::printWordsStartingWith(Word* word, std::ostream&
output) const
{
  if (word)
  {
        output << word->key_ << " - " << word->translation_ << '\n';

        printWordsStartingWith(word->left_, output);
        printWordsStartingWith(word->right_, output);

  }
}

std::istream& skipSpaces(std::ifstream& input)
{
  while (isblank(input.peek()))
  {
        input.get();
  }

  return input;
}
void createDictionary(EnglishRussianDictionary& ERD)
{
  std::string path = "data.txt";
  std::ifstream fin;

  fin.open(path);

  if (!fin.is_open())
  {
    std::cout << "Wrong path!\n";
  }

  std::string strInsert;

  while (std::getline(fin, strInsert))
```

```
 {
        int a = strInsert.find(" ");
        std::string engWord, rusWord;
        for (size_t i = 0; i < strInsert.find(" "); i++)
        {
          engWord.push_back(strInsert[i]);
        }
        rusWord = strInsert.substr(a++);

        ERD.insertWord(engWord, rusWord);

 }
 fin.close();
}
```

Файл double_direction_list.hpp

```
#ifndef DOUBLE_DIRECTION_LIST_HPP
#define DOUBLE_DIRECTION_LIST_HPP

#include <iostream>

class List
{
        class Node
        {
        public:
                std::string data_;
                Node* next_;
                Node* prev_;

                Node() :
                        data_(""),
                        next_(nullptr),
                        prev_(nullptr)
                {
                }
```

```
    };

    Node* head_;
    std::size_t size_;

public:
    List();
    ~List();
    std::size_t getSize() const;
    List& deleteNode(const std::string data);
    List& operator+=(const std::string data);
    friend std::ostream& operator<<(std::ostream& out, const List& List);

private:
    Node* searchNodeAndReturn(const std::string data) const;
};

#endif
```

Файл double_direction_list.cpp

```
#include "double_direction_list.hpp"
#include <stdexcept>

List::List() :
    head_(nullptr),
    size_(0)
{
}

List::~List()
{
```

```
	while (size_)
	{
		Node* old_head = head_;
		head_ = head_->next_;
		size_--;
		delete old_head;
	}
}

std::size_t List::getSize() const
{
	return size_;
}

List& List::deleteNode(const std::string data)
{
	Node* current = searchNodeAndReturn(data);

	if (current == nullptr)
	{
		throw std::invalid_argument("Node not found!\n(List&
List::delete_node(const std::string data))");
	}

	//один узел
	if ((current == head_) && (current->next_ == nullptr))
	{
		delete current;
		head_ = nullptr;
		return *this;
	}
	//голова
```

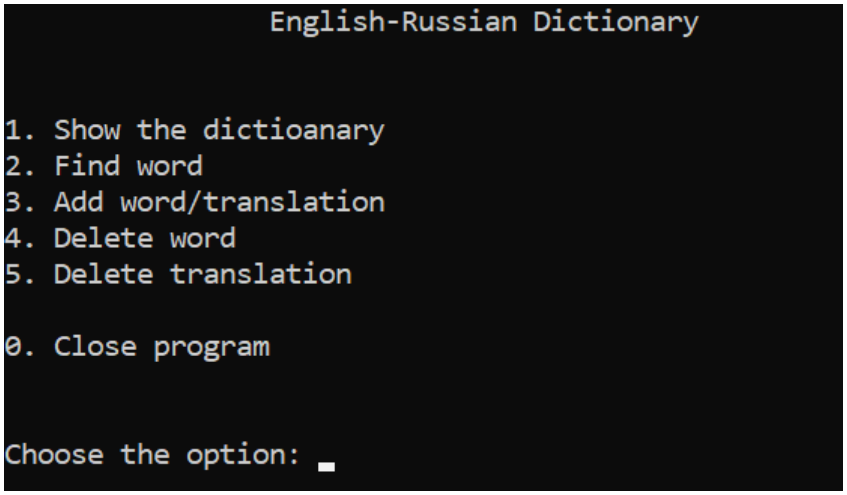
```
        else if (current == head_)
        {
                head_ = head_->next_;
                head_->prev_ = nullptr;

        }
        //хвост
        else if (current->next_ == nullptr)
        {
                current->prev_->next_ = nullptr;
        }
        //середина
        else
        {
                current->prev_->next_ = current->next_;
                current->next_->prev_ = current->prev_;
        }

        size_--;
        delete current;
        return *this;
}

List& List::operator+=(const std::string data)
{
        if (head_ == nullptr)
        {
                head_ = new Node;
                head_->data_ = data;
        }
        else
        {
                Node* current = head_;
```

```
while (data >= current->data_)

{
	if (data == current->data_)
		return *this;

	if (current->next_)
		current = current->next_;
	else
		break;
}


if (current == head_ && data < current->data_)
{
	Node* old_head = head_;
	head_ = new Node;
	head_->data_ = data;
	head_->next_ = old_head;
	old_head->prev_ = head_;
}
else if (current->next_ == nullptr && data > current->data_)
{
	Node* new_node = new Node;
	new_node->data_ = data;
	new_node->prev_ = current;
	current->next_ = new_node;
}
else
{
	Node* new_node = new Node;
	new_node->data_ = data;
	current->prev_->next_ = new_node;
```

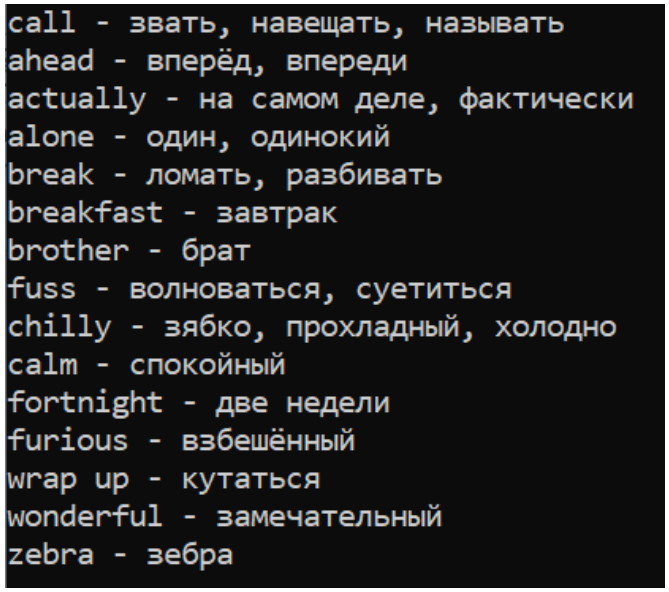
```
                new_node->prev_ = current->prev_;
                current->prev_ = new_node;
                new_node->next_ = current;
            }
        }

        size_++;
        return *this;
}

List::Node* List::searchNodeAndReturn(const std::string data) const
{
        List::Node* current = head_;

        while (current)
        {
            if (data == current->data_)
            {
                return current;
            }

            current = current->next_;
        }

        return nullptr;
}

std::ostream& operator<<(std::ostream& out, const List& List)
{
        if (List.head_)
        {
            List::Node* current = List.head_;
```

```
            while (current->next_)
            {
                out << current->data_ << ", ";
                current = current->next_;
            }

            out << current->data_;
        }

        return out;
}
```

Приложение 2. Протоколы отладки



Стартовое меню.



Функция 1. Show the dictionary

```
Enter a search word: call

Result: call - звать, навещать, называть
```

Функция 2. Find word

```
Enter a word to add: red

Enter the translation: красный

Word and translation added!
```

Функция 3. Add word/translation

```
Enter a word to delete: red

Word deleted!
```

Функция 4. Delete word.

```
Enter a word: call

Enter the translation to delete: звать

Translation deleted!
```

Функция 5. Delete translation