**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Основы информатики»

Отчет по лабораторной работе №9

«Ввод, сортировка и двоичный поиск в массиве структур.»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-13Б |  | преподаватель каф. ИУ5 |
| Поляков Данила |  | Аксёнова М.В. |
| Подпись и дата |  | Подпись и дата: |

Москва, 2021 г.

**Постановка задачи**

Англо-русский словарь построен в виде массива структур Dictionary. Структура содержит английское слово и соответствующее ему русское слово. Максимальный размер словаря – 100 пар слов.

Разработать программу, которая:

• обеспечивает формирование словаря (добавление и удаление записей);

• записывает словарь, отсортированный по английским значениям слов, в файл;

• обеспечивает просмотр словаря;

• выполняет перевод слов с английского на русский, используя для поиска слова в словаре метод двоичного поиска в отсортированном массиве;

• выполняет перевод слов с русского на английский, используя для поиска слов в словаре метод перебора;

Программа должна обеспечивать диалог с помощью меню.

Начальное число слов в словаре равно 10.

Для исключения проблем, связанных с вводом кириллицы, вводите русские слова латинскими буквами, например: kot (cat), sobaka(dog) и т.п.

Шаги разработки программы:

I. Определение состава и способа представления исходных данных, результатов и промежуточных данных.

Исходные данные. Так как максимальный размер словаря по условию задачи ограничен, а фактическое число записей в словаре может изменяться , то для хранения словаря в оперативной памяти (ОП) можно использовать нединамический массив из 100 элементов типа Dictionary.

const int max\_size = 100;

const int l\_word = 31;

struct Dictionary {

char engl[l\_word]; // слово по-английски

char rus[l\_word]; // слово по-русски

};

Максимальная длина русских и английских слов l\_word = 30 символам. Словарь хранится в текстовом файле. Элементы массива структур Dictionary будем записывать в файл последовательно, начиная с нулевого. При этом поля структуры записываются в виде отдельных строк, т.е. каждое слово должно заканчиваться символом ‘\0’.

**Разработка алгоритма**

**Описание алгоритма**

Данная программа позволяет выполнять несколько действий со словами из словаря в формате “Englis\_word Русское\_слово”, записывая и считывая данный набор из файла. Во время работы программа использует оперативную память, в которой хранит считанный словарь. Все изменения структуры словаря происходят в ней, и по просьбе пользователя сохраняются в файле.  
При добавлении слова в словарь оно автоматически вставляется на необходимое место по алфавиту. Весь словарь отсортирован по алфавиту для английских слов, поэтому для нахождения необходимого места в нём для вставки нового слова используется бинарный поиск. После определения места для вставки, всё что ниже этой позиции сдвигается, освобождая место под новое слово.

Бинарный поиск так же используется для нахождения соответствующего английского слова для его дальнейшего перевода на русский язык.  
Поиск русского слова происходит по O(n), обычным проходом по массиву.

После удаления массив так же смещается к тому месту, с которого было удалена запись.

**Описание используемых переменных**

enum MenuItem {

ShowDictionary = 1,

AddRecord,

DeleteRecord,

TranslateEtoR,

TranslateRtoE,

SaveDictionary,

Exit

}; - константы пунктов меню

struct Word {

char English[maxWordLength];

char Russian[maxWordLength];

}; - структура слова в словаре

void menu(Word \*, size\_t &) – функция печати меню, ввода и выполнения(обработки) данного пункта.

size\_t getDictionary(Word \*) – функция получения словаря из файла, возвращает длину словаря.

void showDictionary(Word \*, const size\_t) – функция вывода словаря из памяти на экран

void addRecord(Word \*, size\_t &) - добавление записи в словарь

void deleteRecord(Word \*, size\_t &) – удаление записи из словаря

void saveDictionary(Word \*, const size\_t) – сохранение словаря в файл

void translateEnglishToRussian(Word \*, const size\_t) – вывод слова из словаря в формате English\_word – Русское\_слово по запросу английского.

void translateRussianToEnglish(Word \*, const size\_t) – вывод слова из словаря в формате English\_word – Русское\_слово по запросу русского.

int findEnglishWord(Word \*dictionary, const size\_t sizeOfDictionary, Word wordToFind) – функция, находящая позицию искомого слова, если не найдено возвращает -1.

Word \*dictionary – словарь в памяти

size\_t &sizeOfDictionary – текущий размер словаря, некоторые функции изменяют это значение(add, delete, get)

**Текст программы**

**Листинг кода программы:**

**main.cpp**

#include "functions.h"

int main() {

auto dictionary = new Word[dictionarySize];

size\_t size = getDictionary(dictionary);

menu(dictionary, size);

return 0;

}

**functions.h**

#ifndef CODE\_FUNCTIONS\_H

#define CODE\_FUNCTIONS\_H

#include <iostream>

#include <stdlib.h>

#include <cstring>

#include <iomanip>

#include <fstream>

#define dictionaryFileName "dictionary.qqq"

#define maxWordLength 33

#define dictionarySize 100

enum MenuItem {

ShowDictionary = 1,

AddRecord,

DeleteRecord,

TranslateEtoR,

TranslateRtoE,

SaveDictionary,

Exit

};

struct Word {

char English[maxWordLength];

char Russian[maxWordLength];

};

void printMenu();

void menu(Word \*, size\_t &);

size\_t getDictionary(Word \*);

void showDictionary(Word \*, const size\_t);

void addRecord(Word \*, size\_t &);

void deleteRecord(Word \*, size\_t &);

void saveDictionary(Word \*, const size\_t);

void translateEnglishToRussian(Word \*, const size\_t);

void translateRussianToEnglish(Word \*, const size\_t);

#endif //CODE\_FUNCTIONS\_H

**functions.cpp**

#include "functions.h"

void printMenu() {

const char \*menu[7] = {"Show dictionary",

"Add record to dictionary",

"Delete record from dictionary",

"Translate word from English to Russian",

"Translate word from Russian to English",

"Save dictionary",

"Exit"};

for (size\_t i = 0; i < 7; ++i) {

std::cout << i + 1 << ". " << menu[i] << '\n';

}

std::cout << "Enter needed number:\n";

}

void menu(Word \*dictionary, size\_t &sizeOfDictionary) {

int menuPoint;

while (true) {

printMenu();

std::cin >> menuPoint;

switch (menuPoint) {

case ShowDictionary:

showDictionary(dictionary, sizeOfDictionary);

break;

case AddRecord:

addRecord(dictionary, sizeOfDictionary);

break;

case DeleteRecord:

deleteRecord(dictionary, sizeOfDictionary);

break;

case TranslateEtoR:

translateEnglishToRussian(dictionary, sizeOfDictionary);

break;

case TranslateRtoE:

translateRussianToEnglish(dictionary, sizeOfDictionary);

break;

case SaveDictionary:

saveDictionary(dictionary, sizeOfDictionary);

break;

case Exit:

return;

default:

break;

}

}

}

size\_t getDictionary(Word \*dictionary) {

std::ifstream dictionaryFile(dictionaryFileName);

size\_t i = 0;

while (!dictionaryFile.eof()) {

dictionaryFile >> dictionary[i].English >> dictionary[i].Russian;

i++;

}

dictionaryFile.close();

return i;

}

int findEnglishWord(Word \*dictionary, const size\_t sizeOfDictionary, Word wordToFind) {

bool translated = false;

int begin = 0;

int end = sizeOfDictionary - 1;

int current\_position = (end - begin) / 2;

while (!translated) {

if (strcmp(dictionary[current\_position].English, wordToFind.English) == 0)

translated = true;

else if (strcmp(dictionary[current\_position].English, wordToFind.English) > 0) {

end = current\_position;

current\_position -= (end - begin) / 2;

if (end - begin == 1) {

if (begin == 0) {

current\_position--;

if (strcmp(dictionary[current\_position].English, wordToFind.English) == 0) translated = true;

else break;

} else break;

}

} else {

begin = current\_position;

current\_position += (end - begin) / 2;

if (end - begin == 1) {

if (end == sizeOfDictionary - 1) {

current\_position++;

if (strcmp(dictionary[current\_position].English, wordToFind.English) == 0) translated = true;

else break;

} else break;

}

}

}

if (translated)

return current\_position;

else

return -1;

}

int findRussianWord(Word \*dictionary, const size\_t sizeOfDictionary, Word wordToFind) {

int pos = -1;

for (size\_t i = 0; i < sizeOfDictionary; ++i) {

if (strcmp(dictionary[i].Russian,wordToFind.Russian)==0)

{

pos = i;

break;

}

}

return pos;

}

void saveDictionary(Word \*dictionary, const size\_t sizeOfDictionary) {

std::ofstream dictionaryFile(dictionaryFileName, std::ios::out | std::ios::trunc);

for (size\_t i = 0; i < sizeOfDictionary - 1; ++i) {

dictionaryFile << dictionary[i].English << ' ' << dictionary[i].Russian << std::endl;

}

dictionaryFile << dictionary[sizeOfDictionary - 1].English << ' ' << dictionary[sizeOfDictionary - 1].Russian;

dictionaryFile.close();

}

void showDictionary(Word \*dictionary, const size\_t sizeOfDictionary) {

if (sizeOfDictionary == 0) {

std::cout << "Dictionary is clear!" << std::endl;

} else

for (size\_t i = 0; i < sizeOfDictionary; ++i) {

std::cout << std::setw(31) << std::right << dictionary[i].English << " — " << std::setw(31) << std::left

<< dictionary[i].Russian

<< std::endl;

}

}

void addRecord(Word \*dictionary, size\_t &sizeOfDictionary) {

std::cout << "Enter word in format \"English Russian\":" << std::endl;

Word tempWord = {"", ""};

std::cin >> tempWord.English >> tempWord.Russian;

size\_t pos = sizeOfDictionary;

for (size\_t i = 0; i < sizeOfDictionary; ++i) {

if (strcmp(dictionary[i].English, tempWord.English) > 0) {

pos = i;

break;

}

}

sizeOfDictionary++;

for (size\_t i = sizeOfDictionary; i > pos; --i) {

dictionary[i] = dictionary[i - 1];

}

dictionary[pos] = tempWord;

std::cout << "Word has been added" << std::endl;

}

void deleteRecord(Word \*dictionary, size\_t &sizeOfDictionary) {

std::cout << "Enter word to delete in English:" << std::endl;

Word tempWord = {"", ""};

std::cin >> tempWord.English;

int pos = findEnglishWord(dictionary, sizeOfDictionary, tempWord);

if (pos == -1) {

std::cout << "Word not found!" << std::endl;

} else {

for (size\_t i = pos; i < sizeOfDictionary; ++i) {

dictionary[i] = dictionary[i + 1];

}

sizeOfDictionary--;

std::cout << "Word has been deleted" << std::endl;

}

}

void translateEnglishToRussian(Word \*dictionary, const size\_t sizeOfDictionary) {

std::cout << "Enter word to translate in English:" << std::endl;

Word tempWord = {"", ""};

std::cin >> tempWord.English;

int pos = findEnglishWord(dictionary, sizeOfDictionary, tempWord);

if (pos == -1) {

std::cout << "Word not found!" << std::endl;

} else {

std::cout << dictionary[pos].English << " — " << dictionary[pos].Russian << std::endl;

}

}

void translateRussianToEnglish(Word \*dictionary, const size\_t sizeOfDictionary) {

std::cout << "Enter word to translate in Russian:" << std::endl;

Word tempWord = {"", ""};

std::cin >> tempWord.Russian;

int pos = findRussianWord(dictionary, sizeOfDictionary, tempWord);

if (pos == -1) {

std::cout << "Word not found!" << std::endl;

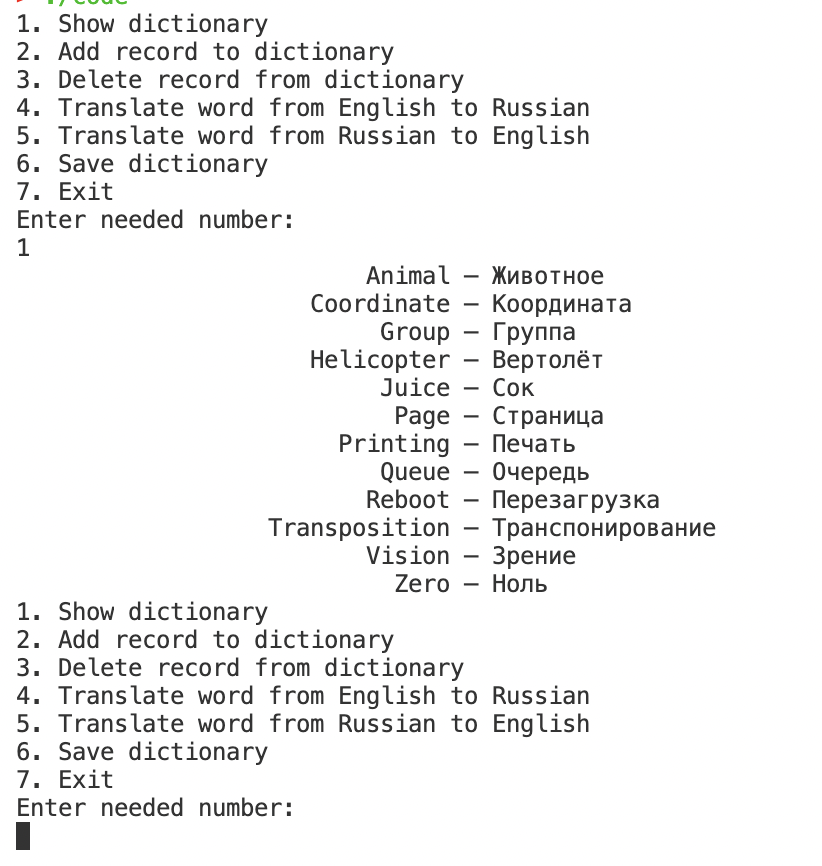
} else {

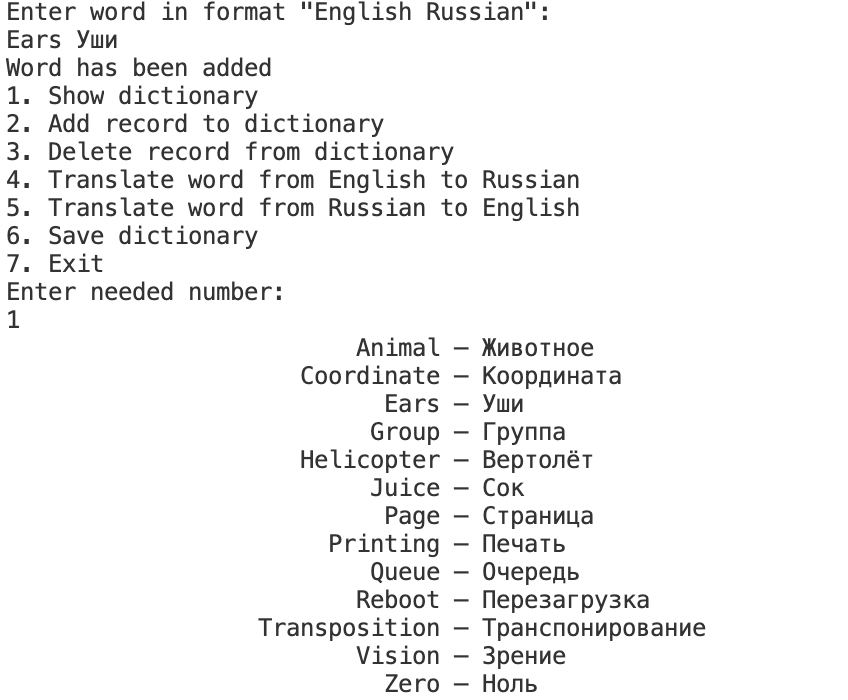
std::cout << dictionary[pos].English << " — " << dictionary[pos].Russian << std::endl;

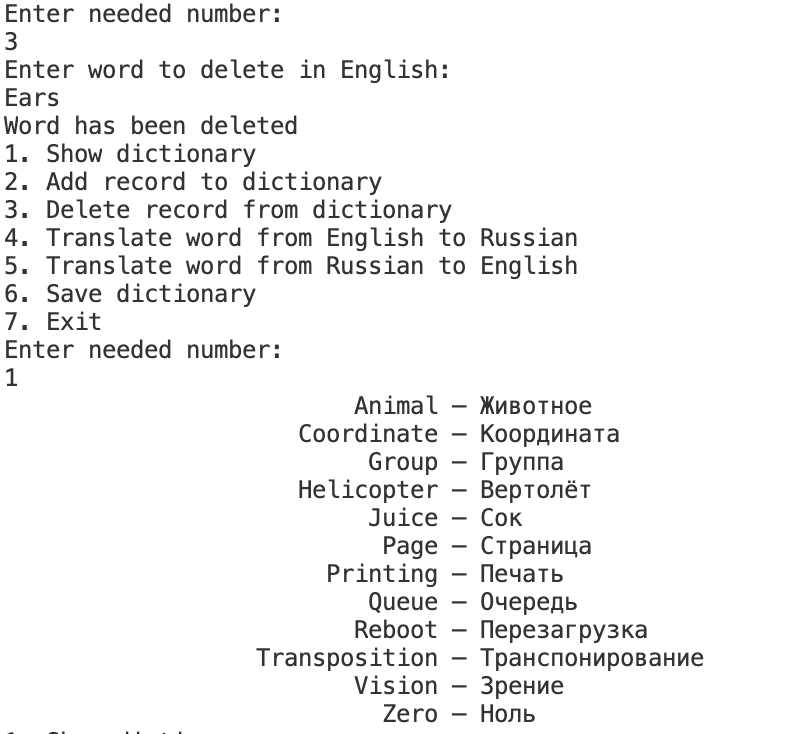
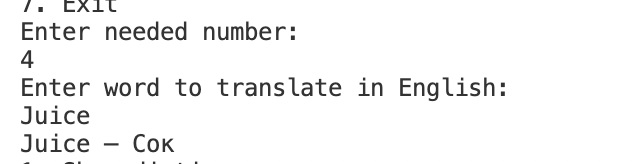
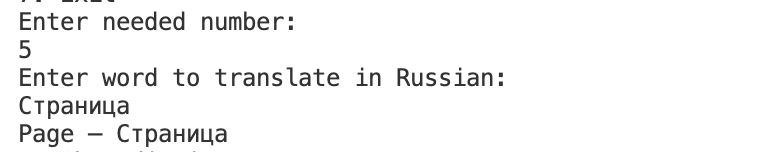
}

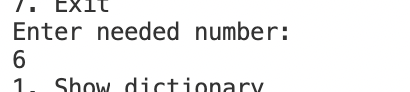
}

**Анализ результатов**







Файл dictionary.qqq:

