**Лабораторная работа №4**

**Дерево поиска. Частотный словарь**

Составить программу, реализующую частотный словарь текста.

Программа должна производить следующие действия:

1. Выделять в выбранном текстовом файле отдельные слова.
2. Помещать слова в дерево поиска.
3. Выводить из дерева частотный словарь слов в алфавитном порядке.
4. Выводить частотный словарь слов, отсортированный в порядке частоты их употребления.
5. Производить поиск заданного слова.
6. Производить фильтрацию, оставляющую в словаре слова определенной длины.

**Текст программы**

#include <iostream>

#include <fstream>

#include <string>

#include <windows.h>

using namespace std;

struct node {

string word; //слово

int k; //сколько раз встретилось слово

node\* left; //указатель на левое поддерево

node\* right; //указатель на правое поддерево

};

void insert(node\*& root, string word, int& count) { //построение дерева

if (root == NULL) {

root = new node;

root->word = word;

root->k = 1;

count++;

root->left = NULL;

root->right = NULL;

}

else if (word < root->word) {

insert(root->left, word, count);

}

else if (word > root->word) {

insert(root->right, word, count);

}

else root->k++;

}

void printTree(node\* root, int count, ofstream& result) { //вывод дерева в алфавитном порядке

if (root != NULL) {

printTree(root->left, count, result);

if (count <= 50) cout << root->word << " - " << root->k << endl;

else result << root->word << " - " << root->k << endl;

printTree(root->right, count, result);

}

}

void arrayTree(node\* root, node\*\*& nodes, int& index, int& capacity) { //запись дерева в массив

if (root != NULL) {

arrayTree(root->left, nodes, index, capacity);

if (index == capacity) {

capacity \*= 2;

node\*\* temp = new node \* [capacity];

for (int i = 0; i < index; ++i) {

temp[i] = nodes[i];

}

delete[] nodes;

nodes = temp;

}

nodes[index++] = root;

arrayTree(root->right, nodes, index, capacity);

}

}

void sortTree(node\*\* nodes, int size) { //сортировка массива по частоте

for (int i = 0; i < size - 1; ++i) {

for (int j = i + 1; j < size; ++j) {

if (nodes[i]->k < nodes[j]->k) {

node\* temp = nodes[i];

nodes[i] = nodes[j];

nodes[j] = temp;

}

}

}

}

void printSortTree(node\*\* nodes, node\* root, int size, int count, ofstream& result) { //вывод дерева в частотном порядке

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

if (count <= 50) cout << nodes[i]->word << " - " << nodes[i]->k << endl;

else result << nodes[i]->word << " - " << nodes[i]->k << endl;

}

}

node\* search(node\* root, string word) { //поиск

if (root == NULL || root->word == word) return root;

if (word < root->word) return search(root->left, word);

else return search(root->right, word);

}

void lengthSearch(node\* root, int length, int count, ofstream& result) { //фильтрация по количеству букв

if (root != NULL) {

lengthSearch(root->left, length, count, result);

if (root->word.length() == length) {

if (count <= 50) cout << root->word << " - " << root->k << endl;

else result << root->word << " - " << root->k << endl;

}

lengthSearch(root->right, length, count, result);

}

}

bool symbol(char c) { //проверка символа

return (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= 'а' && c <= 'я') || (c >= 'А' && c <= 'Я');

}

int main() {

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Программа строит дерево для файла. Пожалуйста, подождите..." << endl;

ifstream f("input.txt");

if (!f.is\_open()) {

cout << "Ошибка открытия файла" << endl;

return 0;

}

node\* root = NULL;

string word;

char c;

int capacity = 1000, count = 0;

node\*\* nodes = new node\* [capacity];

int index = 0;

while (f.get(c)) {

if (symbol(c)) {

//c = tolower(c); //убрать символ комментария, чтобы не учитывать регистр

word += c;

}

else {

if (!word.empty()) {

insert(root, word, count);

word.clear();

}

}

}

if (!word.empty()) {

insert(root, word, count);

}

if (word.empty() && root == NULL) {

cout << "Файл не содержит слов" << endl;

return 0;

}

int n;

cout << "Выберите режим работы программы:" << endl;

cout << "1 - вывод словаря в алфавитном порядке" << endl;

cout << "2 - вывод словаря в частотном порядке" << endl;

cout << "3 - поиск по слову" << endl;

cout << "4 - поиск по количеству букв" << endl;

cin >> n;

cin.ignore();

ofstream result;

result.open("result.txt");

switch (n) {

case 1: {

if (count > 50) cout << "Результаты записываются в файл. Пожалуйста, подождите..." << endl;

printTree(root, count, result);

if (count > 50) cout << "Результаты записаны в файл result.txt" << endl;

}

break;

case 2: {

if (count > 50) cout << "Результаты записываются в файл. Пожалуйста, подождите..." << endl;

arrayTree(root, nodes, index, capacity);

sortTree(nodes, index);

printSortTree(nodes, root, index, count, result);

if (count > 50) cout << "Результаты записаны в файл result.txt" << endl;

}

break;

case 3: {

string searchWord;

cout << "Введите слово для поиска: ";

cin >> searchWord;

node\* foundNode = search(root, searchWord);

if (foundNode != NULL) {

cout << "Слово \"" << searchWord << "\" встречается в файле " << foundNode->k << " раз" << endl;

}

else {

cout << "Слово \"" << searchWord << "\" не найдено" << endl;

}

}

break;

case 4: {

int wordLength;

cout << "Введите длину слова: ";

cin >> wordLength;

if (count > 50) cout << "Результаты записываются в файл. Пожалуйста, подождите..." << endl;

lengthSearch(root, wordLength, count, result);

if (count > 50) cout << "Результаты записаны в файл result.txt" << endl;

}

break;

default: cout << "Введен неверный номер" << endl;

break;

}

delete[] nodes;

f.close();

result.close();

system("pause");

}

**Описание программы**

Структура **node** содержит следующие поля:

* **word** – строка типа **string**, слово, записываемое в дерево
* **k** – число типа **int**, количество вхождений слова **word** в файле
* **left** и **right** – указатели на левое и правое поддеревья

Функция **insert** – строит дерево:

* Если дерево пустое (**root == NULL**), создается новый узел, а значение переменной **count** увеличивается на **1**
* Если слова еще нет в дереве, и оно меньше текущего узла, вызывается функция **insert** для левого поддерева
* Если слова еще нет в дереве, и оно больше текущего узла, вызывается функция **insert** для левого поддерева
* Если слово уже есть в дереве, то для него значение **k** увеличивается на **1**

Функция **printTree** – выводит дерево в алфавитном порядке:

* Функция рекурсивно обходит дерево и выводит слова и количество их вхождений на экран (если значение переменной **count** меньше или равно **50**) или в файл (если больше **50**)

Функция **arrayTree** – записывает указатели на узлы дерева в массив

* Функция рекурсивно обходит дерево и записывает указатели на узлы дерева в массив **nodes**
* Если массив заполняется полностью, его размер увеличивается в 2 раза путем копирования всех данных во временный массив **temp** и затем обратно в увеличенный массив **nodes**

Функция **sortTree** – сортирует дерево по частоте вхождения слов (от наибольшей к наименьшей)

* Сортировка производится по значению переменной **k**

Функция **printSortTree** – выводит дерево в частотном порядке

* Массив **nodes** выводится на экран (если значение переменной **count** меньше или равно **50**) или в файл (если больше 50)

Функция **search** – поиск слова

* Программа обходит дерево и сравнивает значение поля **word** структуры **node** с введенной пользователем строкой **word**
* Функция имеет тип **node\*** и возвращает указатель на корень **root**

Функция **lengthSearch** – поиск слов по количеству букв

* Программа обходит дерево и сравнивает длину поля **word** со введенным пользователем значением длины **length** и выводит только слова подходящей длины на экран (если значение переменной **count** меньше или равно **50**) или в файл (если больше 50)

Функция **symbol** – проверка символов

* Функция типа **bool**, проверяет, является ли символ, прочитанный из файла, заглавной или строчной буквой русского или английского алфавита. Если да – возвращает значение **1**, если нет, то **0**

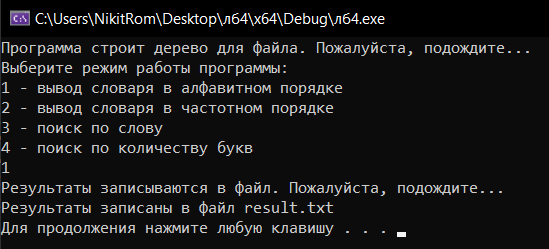
Содержание функции **main** – основной части программы:

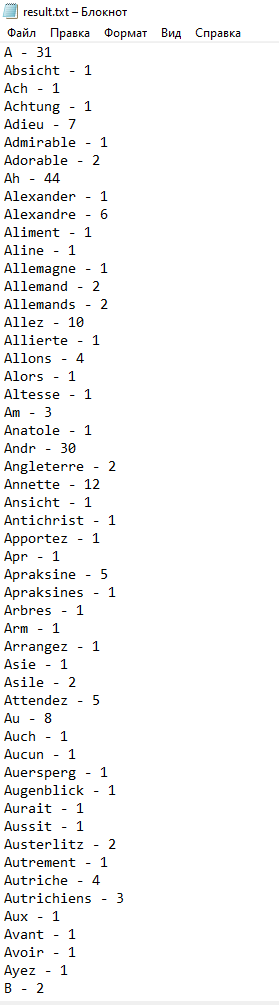
1. Программа открывает файл **f**
2. Программа проверяет, получилось ли открыть файл. Если нет – выводится надпись *«Ошибка открытия файла»*, программа завершается
3. Полю **root** структуры **node** присваивается значение **NULL**
4. Инициализируются переменные: строка **word** типа **string**, символ c типа **char**, числа типа **int**: **capacity** с начальным значением **1000** и **index** с начальным значением **0**
5. Создается динамический массив **nodes** указателей на структуру **node** размером **capacity**
6. Программа посимвольно считывает файл и проверяет каждый символ на то, является ли он буквой, с помощью функции **symbol**. Если да – символ прибавляется к строке **word**. Если нет – получившаяся строка **word** записывается в дерево с помощью функции **insert**. Затем строка очищается и запись слова начинается заново
7. Если корень **root** равен **NULL**, а строка **word** пуста, значит файл либо пуст, либо содержит любые символы, кроме букв. В таком случае выводится фраза *«Файл не содержит слов»* и программа завершается
8. Программа предлагает пользователю выбрать один из четырех вариантов работы: вывод словаря в алфавитном порядке (**1**), вывод словаря в частотном порядке (**2**), поиск по слову (**3**), поиск по количеству букв (**4**). Пользователю необходимо ввести значение переменной **n** – режим работы программы. Выбор режима реализован с помощью оператора **switch**
9. Открывается файл **result**
10. Если выбран режим №1 – выполняется функция **printTree**
11. Если выбран режим №2 – по очереди выполняются функции **arrayTree**, **sortTree**, **printSortTree**
12. Если выбран режим №3 – программа запрашивает у пользователя слово, которое необходимо найти, оно записывается в строку **searchWord**. Выполняется функция **search**, результат которой записывается в переменную **foundNode** типа **node\***. Если слово найдено (т. е. **foundNode != NULL**), на экран выводится надпись *«Слово \*\*\*\*\* встречается в файле \*\*\* раз»*. Если слово не найдено (**foundNode == NULL**), выводится надпись *«Слово \*\*\*\*\* не найдено»*
13. Если выбран режим №4 – программа запрашивает у пользователя длину слова и записывает его в переменную **wordLength**. Затем выполняется функция **lengthSearch**
14. Если пользователь ввел что-то другое, а не число от **1** до **4**, тогда на экран выводится надпись *«Введен неверный номер»*
15. Удаляется массив **nodes**
16. Закрываются файлы **f** и **result**

**Результаты тестов**

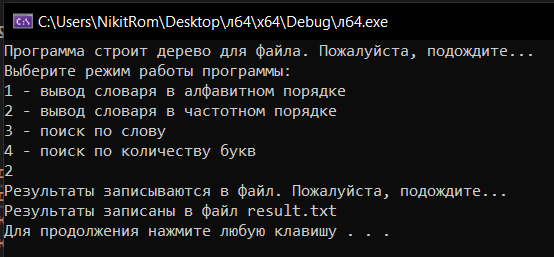
Исходный файл – «Война и мир»

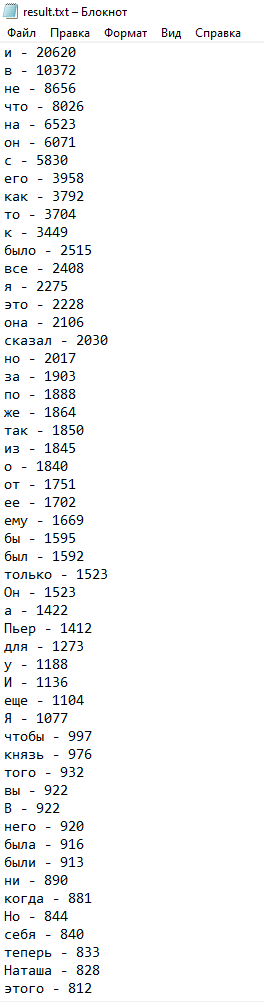
1. Алфавитный порядок



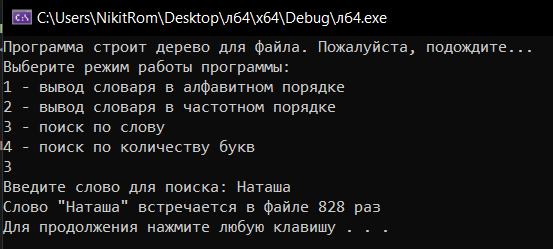


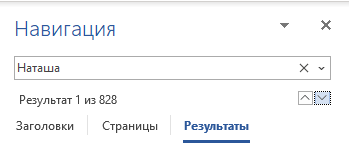
1. Частотный порядок

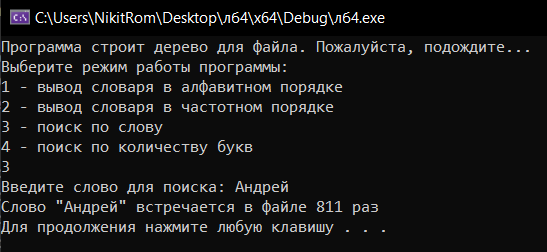


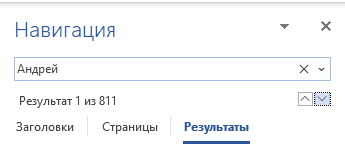


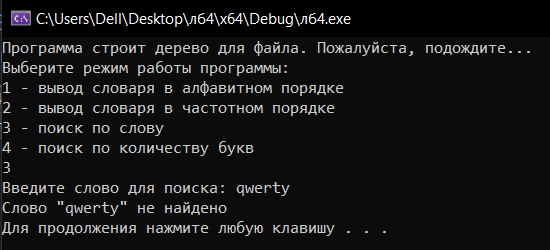
1. Поиск

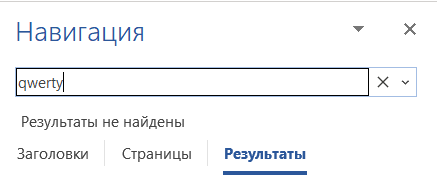




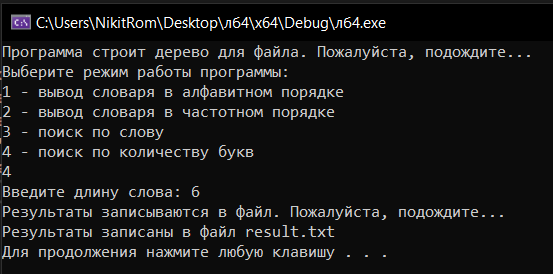




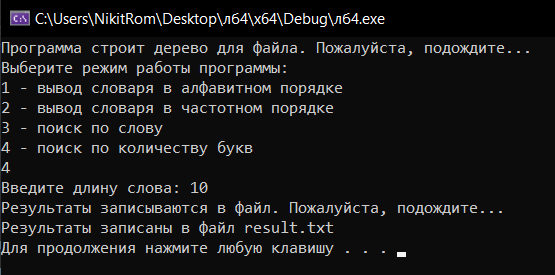


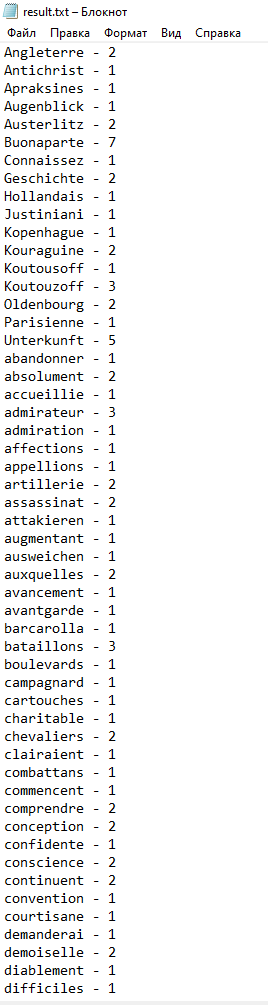


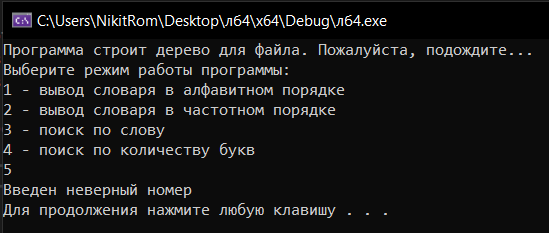
1. Поиск по количеству букв





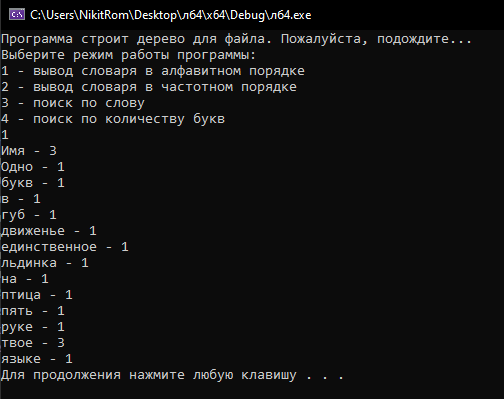




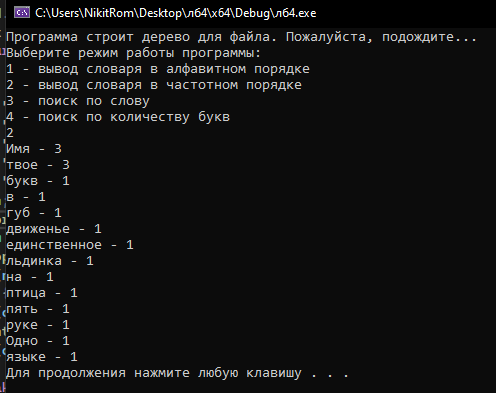


Марина Цветаева «Имя твое — птица в руке...»

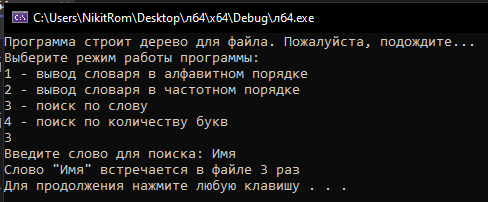
1. Алфавитный порядок



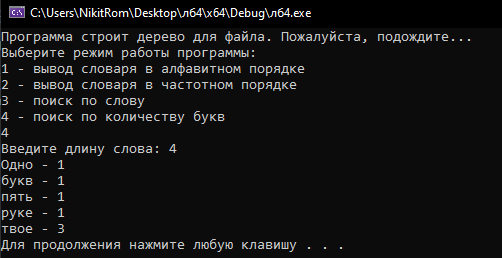
1. Частотный порядок



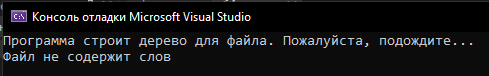
1. Поиск



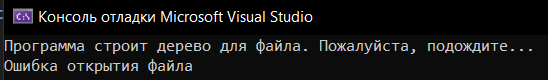
1. Поиск по количеству букв



Пустой файл



Отсутствие файла



**Вывод**

В ходе выполнения лабораторной работы была написана программа, реализующая частотный словарь текста.

Программа выделяет в текстовом файле отдельные слова, помещает их в дерево поиска, выводит из дерева частотный словарь слов в алфавитном и частотном порядке, а также производит поиск заданного слова и фильтрацию, оставляющую в словаре слова определенной длины.