**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.

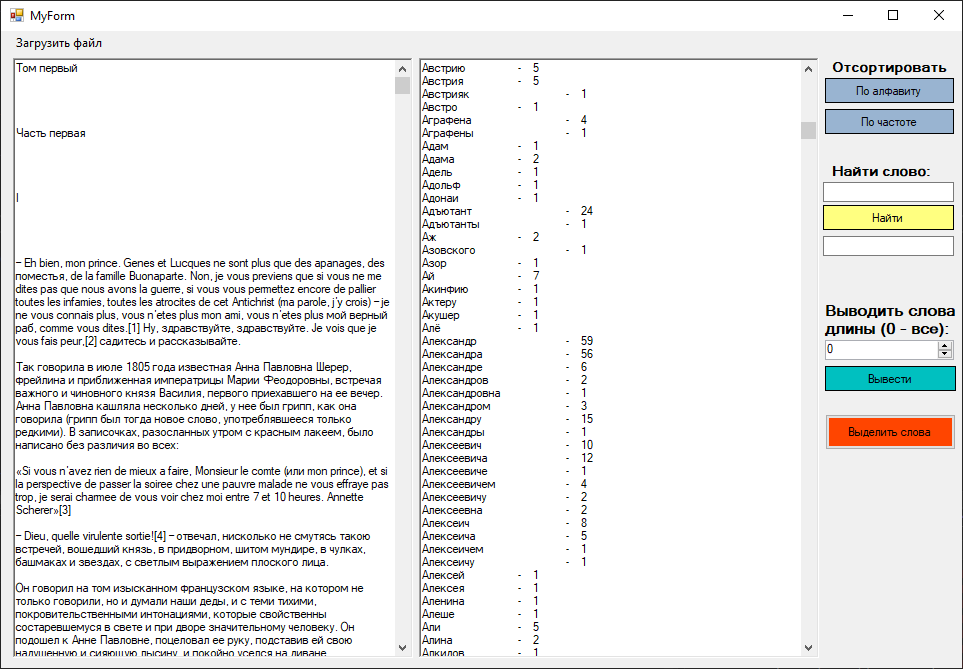
4. Выводить частотный словарь слов, отсортированный в порядке частоты их употребления.

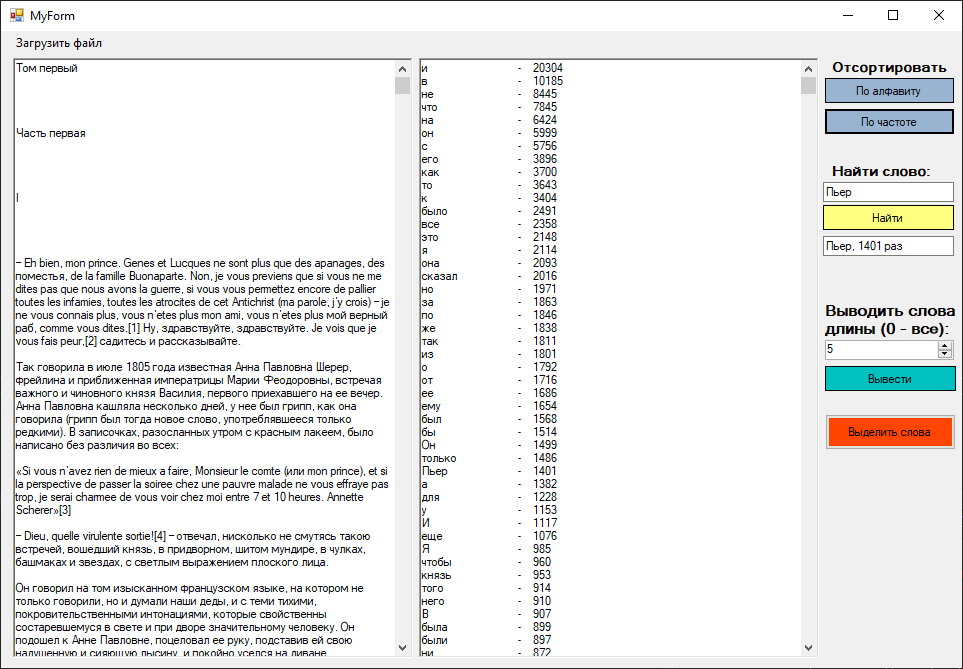
5. Производить поиск заданного слова.

6. Производить фильтрацию, оставляющую в словаре слова определенной длины.

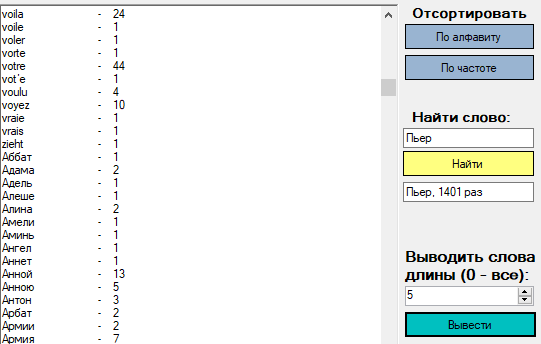
Алгоритм

* Бинарное дерево: нода (слово, число повторений слова, ссылка на левое и правое поддерево, статическая переменная общего размера дерева), функция записи слова в дерево, вывода дерева симметричным обходом, удаление дерева
* Загрузка файла через диалоговое окно
* Создание дерева: разделение текста на слова, запись слов в массив, запись слов в дерево поиска, запись дерева в файл с частотой слов (построчно, и слово, и частота), запись дерева в алфавитном порядке (симметричным обходом, построчно слово вместе с частотой), запись дерева в порядке частоты встречаемости слов, вывод дерева в алфавитном порядке
* Сортировка по частоте: запись адресов всех элементов дерева в массив, сортировка массива методом Шелла, запись отсортированного дерева в файл (построчно слово и его частота)
* Поиск слова: проход по файлу с деревом, пока не будет найдено слово, вывод слова с его частотой в случае успеха, иначе вывод сообщения «Слово не найдено»
* Вывод слов определённой длины: задание длины счетчиком, функция поиска слова: чтение слова из файла с деревом, сравнение его длины с счетчиком, в случае равенства запись слова с его частотой в файл, вывод содержания файла









#include "MyForm.h"

#include <iostream>

#include <msclr\marshal\_cppstd.h>

using namespace System;

using namespace System::Windows::Forms;

using namespace System::IO;

[STAThreadAttribute]

int main(array<String^>^ args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Lab4::MyForm form;

Application::Run(% form);

}

//создание дерева

class Node {

public:

std::string word;

int count;

Node\* left, \* right;

static int size;

};

int Node::size = 0;

//запись слов в дерево

void insert(Node\*& root, std::string\* word) {

if (root == NULL) {

root = new Node;

root->word = \*word;

root->count = 1;

root->right = 0;

root->left = 0;

Node::size++;

}

else if (\*word < root->word)

insert(root->left, word);

else if (\*word > root->word)

insert(root->right, word);

else root->count++;

}

//вывод дерева по алфавиту (симметричный обход)

void printTree2(Node\* node, StreamWriter^% file, StreamWriter^% file\_tree) {

if (node != NULL) {

printTree2(node->left, file, file\_tree);

file->Write(gcnew String(node->word.c\_str()) + "\t\t- " + node->count + "\n");//преобразование из std::string в String^

file\_tree->WriteLine(gcnew String(node->word.c\_str()));

file\_tree->WriteLine(node->count);

printTree2(node->right, file, file\_tree);

}

}

//метод Шелла

void shell(Node\*\* mass, int n) {

int step = n / 2;

while (step > 0) {

for (int i = 0; i < n - step; i++) {

int j = i;

while (j >= 0 && mass[j]->count < mass[j + step]->count) {

std::swap(mass[j], mass[j + step]);

j -= step;

}

}

step /= 2;

}

}

//заполнение массива адресов дерева

void searchTree(Node\* node, Node\*\* mass, int\* counter) {

if (node != NULL) {

mass[\*counter] = node;

\*counter = \*counter + 1;

searchTree(node->left, mass, counter);

searchTree(node->right, mass, counter);

}

}

//сортировка по частоте

void printFrequency(Node\* node, StreamWriter^% file) {

Node\*\* mass{ new Node \* [node->size] };

int counter = 0;

searchTree(node, mass, &counter);

shell(mass, counter);

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

{

file->Write(gcnew String(mass[i]->word.c\_str()) + "\t\t- " + mass[i]->count + "\n");//System::Convert::ToString(node->size)

}

//file->Write(System::Convert::ToString(counter));

delete[] mass;

}

//удаление дерева

void deleteTree(Node\* node) {

if (node != NULL) {

deleteTree(node->left);

deleteTree(node->right);

delete node;

}

}

void search\_word(StreamReader^% file\_tree, int\* counter, String^% word1) {

while (file\_tree->Peek() > -1) {

if (file\_tree->ReadLine() == word1) {

\*counter = Convert::ToInt32(file\_tree->ReadLine());

break;

}

else file\_tree->ReadLine();

}

}

System::Void Lab4::MyForm::загрузитьФайлToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e)

{

OpenFileDialog^ openFile1 = gcnew OpenFileDialog;

openFile1->DefaultExt = "\*.txt";

openFile1->Filter = "Текстовые файлы (\*.txt)|\*.txt|Все файлы (\*.\*)|\*.\*";

if (openFile1->ShowDialog() == System::Windows::Forms::DialogResult::OK && openFile1->FileName->Length > 0)

richTextBox1->Text = System::IO::File::ReadAllText(openFile1->FileName, System::Text::Encoding::Default);

}

System::Void Lab4::MyForm::start\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (richTextBox1->Text != "") {

Node\* tree = NULL;//дерево поиска

int row = 0;//количество слов в таблице

String^ word = "";//слово

array <String^>^ seps = { "\t", "\n", "\r", " ", ".", ",", "!", "?", "-", ":", ";" , "–", "\_" , "…", "[", "]", "(", ")", "/", "\*", "\"", "»", "«" };

array <String^>^ text = richTextBox1->Text->Split(seps, StringSplitOptions::RemoveEmptyEntries);

for (int i = 0; i < text->Length; i++) {

word = text[i];

//запись слов в дерево поиска

std::string word\_std = msclr::interop::marshal\_as<std::string>(word);//преобразование из String^ в std::string

insert(tree, &word\_std);

word = "";

}

//запись дерева с частотой

StreamWriter^ file\_tree = gcnew StreamWriter("tree.txt", false, System::Text::Encoding::Unicode);

//запись дерева в алфавитном порядке

StreamWriter^ file0 = gcnew StreamWriter("tree\_alphabet.txt", false, System::Text::Encoding::Unicode);

printTree2(tree, file0, file\_tree);

file0->Close();

file\_tree->Close();

//запись дерева в порядке частоты употребления слов

StreamWriter^ file1 = gcnew StreamWriter("tree\_frequency.txt", false, System::Text::Encoding::Unicode);

printFrequency(tree, file1);

file1->Close();

//вывод дерева в алфавином порядке

StreamReader^ fout = gcnew StreamReader("tree\_alphabet.txt");

richTextBox2->Text = fout->ReadToEnd();

fout->Close();

deleteTree(tree);//удаление работает

MessageBox::Show(L"Результаты записаны в файл 'tree\_alphabet'", L"Успех!", MessageBoxButtons::OK, MessageBoxIcon::Asterisk);

}

else MessageBox::Show(L"Пустая входная строка", L"Ошибка!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

//сортировка по алфавиту

System::Void Lab4::MyForm::sort\_alphabetically\_Click(System::Object^ sender, System::EventArgs^ e)

{

StreamReader^ fout = gcnew StreamReader("tree\_alphabet.txt");

richTextBox2->Text = fout->ReadToEnd();

fout->Close();

}

//сортировка по частоте

System::Void Lab4::MyForm::sort\_frequency\_Click(System::Object^ sender, System::EventArgs^ e)

{

StreamReader^ fout = gcnew StreamReader("tree\_frequency.txt");

richTextBox2->Text = fout->ReadToEnd();

fout->Close();

}

//найти слово

System::Void Lab4::MyForm::find\_Click(System::Object^ sender, System::EventArgs^ e)

{

StreamReader^ file\_tree = gcnew StreamReader("tree.txt");

int counter = -1;

String^ word = textBox1->Text;

search\_word(file\_tree, &counter, word);

if (counter > 0) textBox2->Text = word + ", " + counter + " раз";

else textBox2->Text = "Слово не найдено";

file\_tree->Close();

}

//вывести слова определённой длины

System::Void Lab4::MyForm::find\_long\_Click(System::Object^ sender, System::EventArgs^ e)

{

if (numericUpDown1->Value > 0) {

String^ word;

StreamReader^ file\_tree = gcnew StreamReader("tree.txt");

StreamWriter^ file = gcnew StreamWriter("long.txt", false, System::Text::Encoding::Unicode);

while (file\_tree->Peek() > -1) {

word = file\_tree->ReadLine();

if (word->Length == numericUpDown1->Value) {

file->Write(word + "\t\t- " + file\_tree->ReadLine() + "\n");

}

else file\_tree->ReadLine();

}

file->Close();

file\_tree->Close();

StreamReader^ file1 = gcnew StreamReader("long.txt");

richTextBox2->Text = file1->ReadToEnd();

file1->Close();

}

else {

StreamReader^ fout = gcnew StreamReader("tree\_alphabet.txt");

richTextBox2->Text = fout->ReadToEnd();

fout->Close();

}

}