МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего

образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

**ОТЧЕТ**

**по Лабораторной работе №5**

по дисциплине

Технология программирования

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Багиров М.Б.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Вихрова С.С

(подпись) (фамилия, и.,о.)

18-ИСТ-4

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород   
2020

Оглавление

[Вариант №4 3](#_Toc37505861)

[Листинг программы 3](#_Toc37505862)

[Результат работы программы 7](#_Toc37505863)

# Вариант №4

Задается словарь. Найти в нем все анаграммы (слова, составленные из одних и тех же букв).

# Листинг программы

**-- Helpers.p –**

#ifndef HELPERS\_H

#define HELPERS\_H

#include <string>

#include <vector>

#include <fstream>

class FstreamHelper {

public:

/\* Вспомогательная функция для чтения данных из файла и записи в матрицу.

Возвращает true если считывание прошло

успешно. \*/

static bool ReadFileToStrings(std::string& theFilename, std::vector<std::string>& theStrings) {

std::ifstream aFile(theFilename);

if (!aFile.is\_open()) { return false;

}

std::string aFileLineString;

// Считываем по одной строчке

while (getline (aFile, aFileLineString)) {

theStrings.push\_back (aFileLineString); }

aFile.close();

return true;

}

};

#endif // HELPERS\_H

**--Main.cpp—**

#include <iostream>

#include <string>

#include <vector>

#include <wchar.h>

#include "helpers.h"

class InsertSort {

public:

/\* Функция для сортировки вставками. \*/

static void Sort(int\* const theMassive, const int theSize) {

int aSortIndex = 0;

while (aSortIndex < theSize - 1)

{ if (theMassive[aSortIndex] > theMassive[aSortIndex + 1])

{ int anIndexToInsert = FindInsertIndex(theMassive, theSize, theMassive[aSortIndex + 1]);

Insert(theMassive, anIndexToInsert, aSortIndex + 1);

}

++aSortIndex; }

}

private:

/\* Вставляет значение из theMassive[thePreviousIndex]

в theMassive[theIndexToInsert] и выполняет сдвиг вправо.

\*/

static void Insert(int\* const theMassive, const int theIndexToInsert, const int thePreviousIndex) {

int aTemp = theMassive[theIndexToInsert];

theMassive[theIndexToInsert] = theMassive[thePreviousIndex];

for (int i = thePreviousIndex; i > theIndexToInsert + 1; --i)

{ theMassive[i] = theMassive[i - 1]; }

theMassive[theIndexToInsert + 1] = aTemp;

}

/\* Ищет место для вставки. \*/

static int FindInsertIndex(const int\* const theMassive, const int theSize, const int theValueToInsert)

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

if (theValueToInsert < theMassive[i]) { return i; } }

std::cout << "Error" << std::endl;

return -1; }

};

class Anagram {

public: Anagram(std::string& theWord) : myWord(theWord) { Sort(); }

std::string GetWord() { return myWord; }

std::string SortedSymbols() { return mySortedSymbols; }

private:

/\* Сортируем символы и записываем полученную строку в поле mySortedSymbols. Для использования функции

\* InsertSort::Sort создаем промежуточный сырой массив целых чисел.

\*/

void Sort() {

const int aMassiveSize = static\_cast<int> (myWord.size());

int \*aMassive = new int[aMassiveSize];

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

int anIndex = static\_cast<int> (i); // приводим к числовому типу

aMassive[anIndex] = myWord[i]; }

InsertSort::Sort (aMassive, aMassiveSize);

// Сортируем числа

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

mySortedSymbols.push\_back(static\_cast<char> (aMassive[i]));

// Приводим обратно к символьному типу

} }

std::string myWord;

std::string mySortedSymbols;

};

/\* Класс для поиска анаграмм. \*/

class AnagramFounder {

public: /\* Функция создает объекты Anagram из строк переданного вектора theStrings.

\* Объекты помещаются в собственный контейнер myAnagrams.

\*/

void Init(std::vector<std::string>& theStrings) {

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

{ myAnagrams.push\_back(Anagram(theStrings[i]));

} }

/\* Функция вывода всех анаграмм. Использует массив флагов для избежания повторений.

\* Для поиска анаграмм использует функцию Anagramm::SortedSymbols().

\* Если отсортированный набор символов совпадает, то выводим слова используя функцию Anagramm::GetWord().

\*/

void Print() {

const int aMassiveSize = myAnagrams.size();

bool \*anAlsoDisplayed = new bool[aMassiveSize];

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

anAlsoDisplayed[i] = false;

}

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

if (anAlsoDisplayed[i] == false) {

anAlsoDisplayed[i] = true;

size\_t anIndex = static\_cast<size\_t> (i);

std::cout << myAnagrams[anIndex].GetWord() << std::endl;

for (size\_t j = 0; j < myAnagrams.size(); ++j) {

// Сравниваем отсортированные строки

if (!anAlsoDisplayed[j] && myAnagrams[anIndex].SortedSymbols() == myAnagrams[j].SortedSymbols()) {

anAlsoDisplayed[j] = true;

std::cout << myAnagrams[j].GetWord() << std::endl;

}

}

std::cout << "----------------------" << std::endl;

}

}

}

private:

std::vector<Anagram> myAnagrams;

};

int main() {

std::string aFilename = "dictionary.txt";

std::vector<std::string> aStrings;

bool aResult = FstreamHelper::ReadFileToStrings(aFilename, aStrings);

if (!aResult) {

std::cout << "Error reding from file. Check that file " << aFilename << " exist." << std::endl;

}

AnagramFounder aFounder;

aFounder.Init(aStrings);

aFounder.Print();

std::cout << "\_\_\_\_\_" << std::endl;

system("pause");

return 0;

}

**-- dictionary.txt—**

tar

arc

car

elbow

taste

cider

angel

cried

dusty

state

study

night

thing

inch

below

brag

grab

cat

chin

act

bored

save

robed

vase

glean

stressed

rat

desserts

# Результат работы программы

