**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе № 2**

**по дисциплине «Программирование»**

Тема: Обработка текста

|  |  |  |
| --- | --- | --- |
| Студент гр. 2309 |  | Савин П.А. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2023

**Формулировка задания.**

Дан текст — набор слов, не всегда имеющий смысл. Слова разграничиваются пробелом (или пробелами) и знаками пунктуации. Требуется выделить в тексте все уникальные слова, подсчитать кол-во их вхождений и составить из них таблицу, где слова отсортированы по убыванию кол-ва вхождений.

**Анализ задания.**

По знакам пунктуации — это надмножество знаков препинания, т.е. к привычным знакам, применяемым в письме, добавляются и другие — математические знаки, знак «собака» и прочие. Об этом подробнее в «Особенностях реализации задания на компьютере».

По организации текста — помимо разделения пробелами слова могут разделяться остальными пробельными символами — переход на новую строку и табуляция.

По организации таблицы — отбор регистрочувствительный, т.е. отличие по регистру хоть в один знак уже считается основанием записать новое уникальное слово. Также следует уточнить, что алгоритм сортировки ограничивается только критерием кол-ва вхождений, т.е. мы не сравниваем дальше слова с равным числом вхождений.

**Математическая** **постановка.**

Дано:

Текст — последовательность строк или даже одна строка. Строка — последовательность символов; часть из них являются пробельными или знаками пунктуации — они отделяют непрерывные подпоследовательности друг от друга — слова (не гарантированно).

Найти:

Слова отдельно от знаков пунктуации и пробельных символов, отобрать среди них уникальные, произвести подсчет их вхождений и составить таблицу со статистикой по каждому уникальному слову, отсортированную по убыванию кол-ва вхождений.

Решение:

В качестве примера будем рассматривать двухстрочный текст:

Hello, world!

Goodbye, world!

Первый шаг заключается в том, чтобы преобразовать текст в строки. Изначально текст размещается линейным образом (рис. 1):

C:\Users\DOM\Desktop\struct1.png

рис. 1 — текст в линейном представлении

Здесь видны и пробелы, и переходы на новую строку, и обычные символы, образующие слова. По такому представлению нужно создать линейный массив строк — разбить текст по символу перехода на новую строку, заодно и посчитаем слова заранее. Алгоритм обработки выглядит так (рис. 2):



рис. 2 — алгоритм обработки исходного массива: образование нового массива + подсчет слов

По этому алгоритму получаем следующий массив (рис. 3):

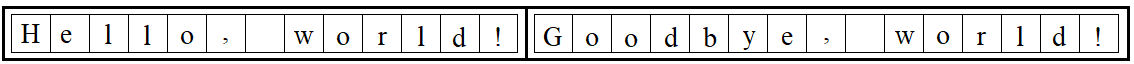


рис. 3 — новый массив строк

Поскольку слова находятся в пределах одной строки, обработка тоже будет построчной, используемый алгоритм уже был применен (рис. 2), но тут доработан под новый массив. Выглядит алгоритм так (рис. 4):

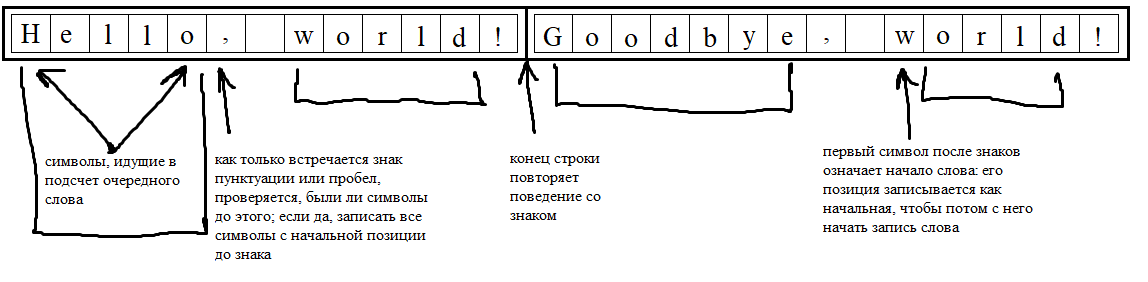


рис. 4 — алгоритм обработки массива строк

Получаем следующий массив (рис. 5):

C:\Users\DOM\Desktop\struct2.png

рис. 5 — новый массив, состоящий только из слов

Далее нужно определить кол-во уникальных слов и сформировать массив-таблицу с ними. Сначала подсчет: алгоритм выглядит так (рис. 6):



рис. 6 — алгоритм обработки массива слов

Здесь же (в конце описания) приведены действия при обнаружении неуникального слова — дополнение для алгоритма создания таблицы. Получается следующая таблица (рис. 7):

C:\Users\DOM\Desktop\table1.png

рис. 7 — таблица уникальных слов в изначальном варианте

Эта структура уже содержит всю необходимую информацию для ответа, но еще не организована. Сортировка происходит по убыванию, для упрощения написания кода была применена шейкерная сортировка (рис. 8) по второму подэлементу элемента таблицы — как раз по кол-ву вхождений. Побочные критерии в учет не брать.

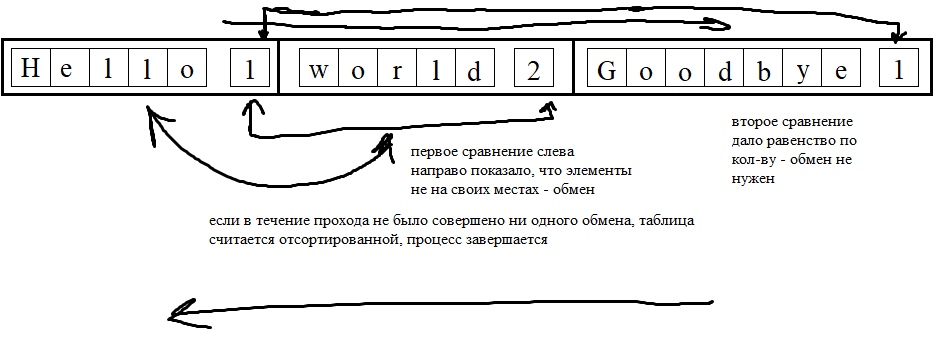


рис. 8 — алгоритм сортировки элементов

В случае с нашей таблицей получим следующее (рис. 9):

C:\Users\DOM\Desktop\table2.png

рис. 9 — итоговая таблица, используемая для вывода ответа

Обмену подверглись только элементы 1 и 2 — только они удовлетворили критерий сортировки. Остается лишь последовательно выписать все элементы слева направо, каждый на новой строке.

Ответ:

world 2

Hello 1

Goodbye 1

ИЛИ

сообщить, что в тексте не найдено слов (если такое возникает в ходе обработки).

**Контрольный пример.**

За пример был взят следующий текст:

get ready; get set; go!

"ready, steady, go!"

В нем специально вставлены хаотично табуляции и переходы на новые строки. Ответ на такой текст следующий:

get 2

ready 2

go 2

set 1

steady 1

**Особенности реализации задания на компьютере.**

По знакам пунктуации — пусть дана стандартная ASCII таблица символов (рис. 10):

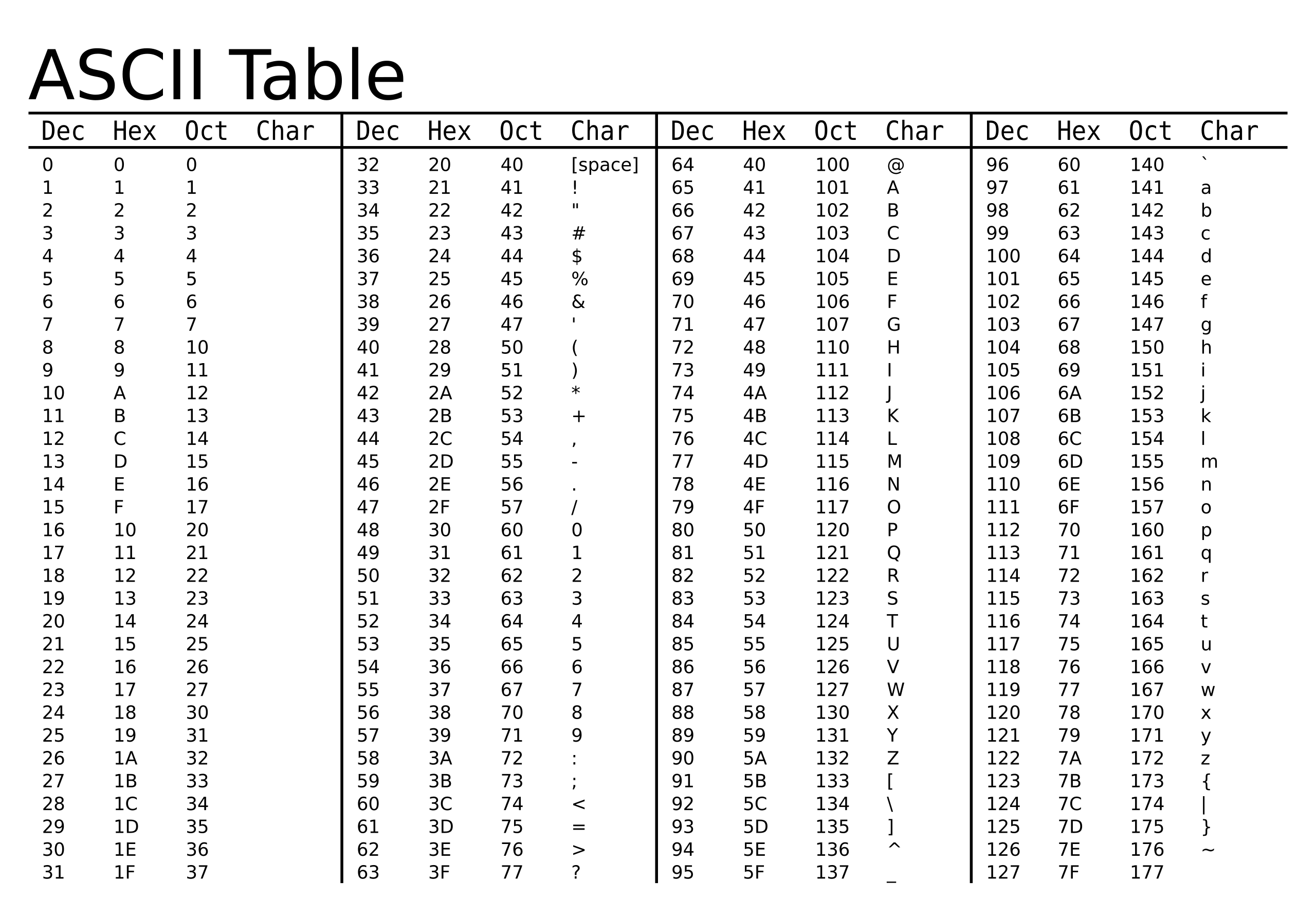


рис. 10 — собственно, ASCII таблица; неотображаемые знаки остались на таблице неотображаемыми

Поскольку правила письменного русского языка изменились довольно сильно, то и список знаков пунктуации расширился. Теперь это не только традиционные знаки препинания, но и операнды математических выражений, и некоторые символы, используемые в компьютерах. Грубо говоря, под отсечение попадают следующие символы (рис. 11):

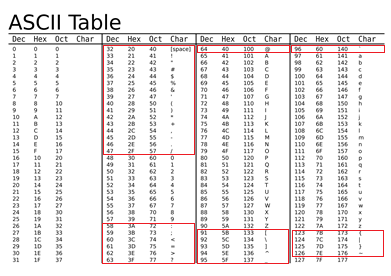


рис. 11 — все та же ASCII таблица, но теперь красным обведены отсекаемые символы

Остается неучтенным знак «№», который не входит в основную таблицу, соответственно, он будет входить в состав слов.

По описанным в «Математической постановке» структурам — структура, организующая хранение слова, остается прежней, модельное представление — привязанная длина слова (рис. 12).

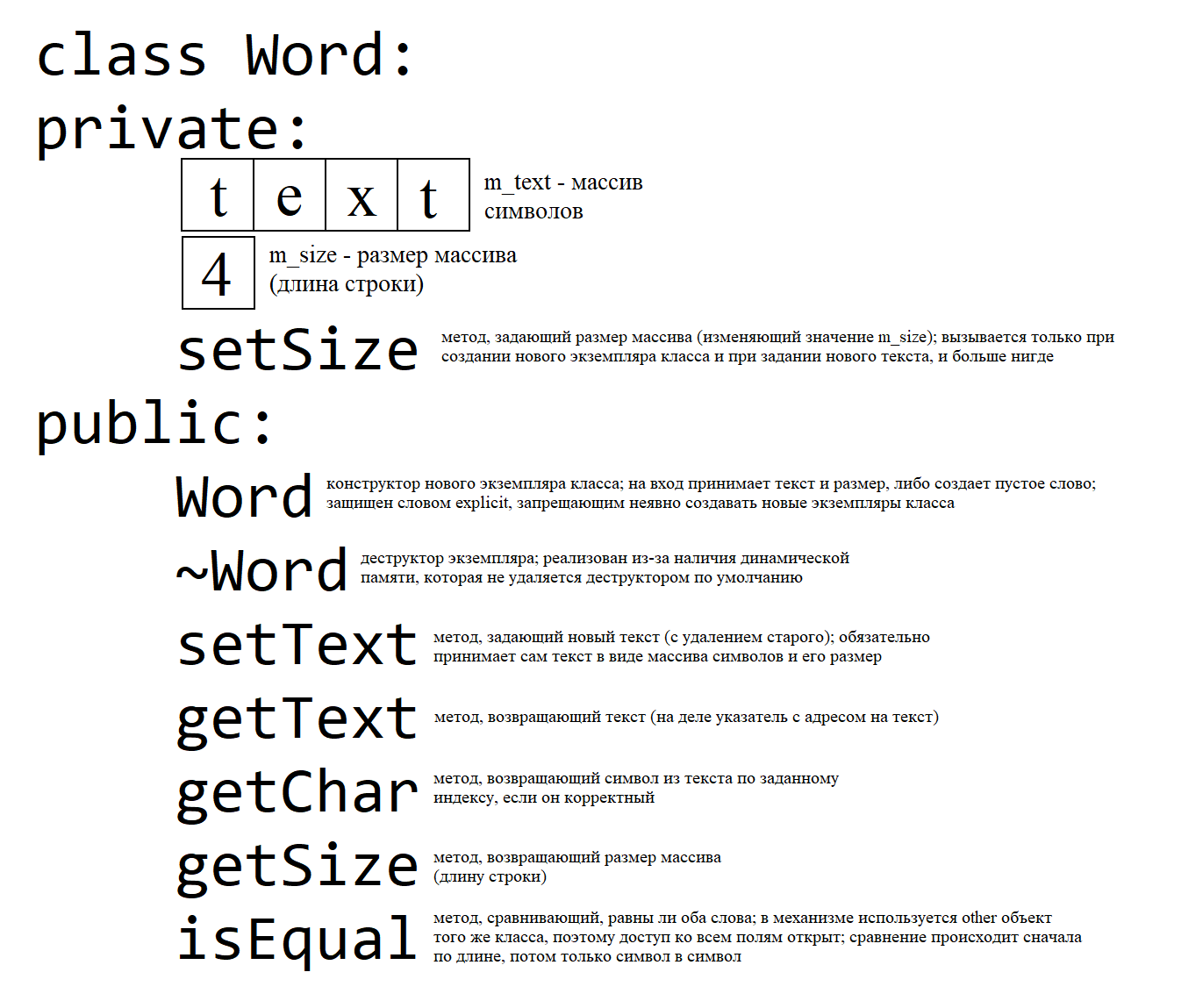


рис. 12 — описание класса Word (базовый класс для всех махинаций)

Эта структура удобна тем, что через нее можно реализовать две последующие структуры — для текста и для слов (рис. 13):

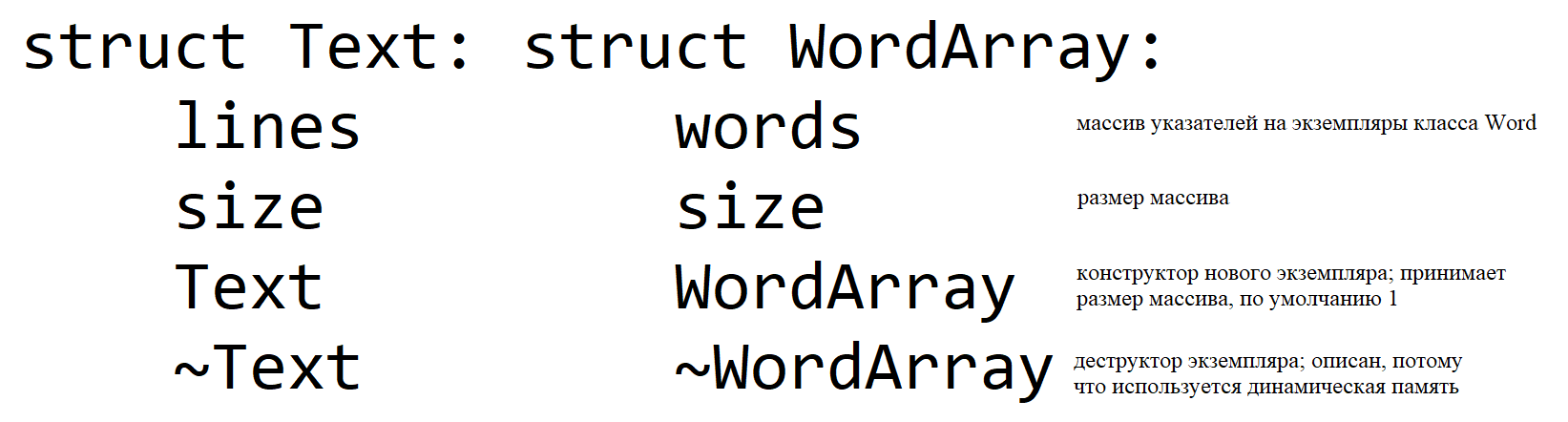


Рис. 13 — описание структур Text и WordArray

А вот со структурой, подготавливающей таблицу, все немного сложнее. Дело в том, что она изначально реализована как массив, но вот какой? Одномерный, хранящий объекты, которые, в свою очередь, будучи структурами, хранят уже 2 элемента — слово и кол-во вхождений. Поэтому появятся сразу 2 новые структуры — для отдельных элементов (рис. 14) и для всей таблицы (рис. 15):

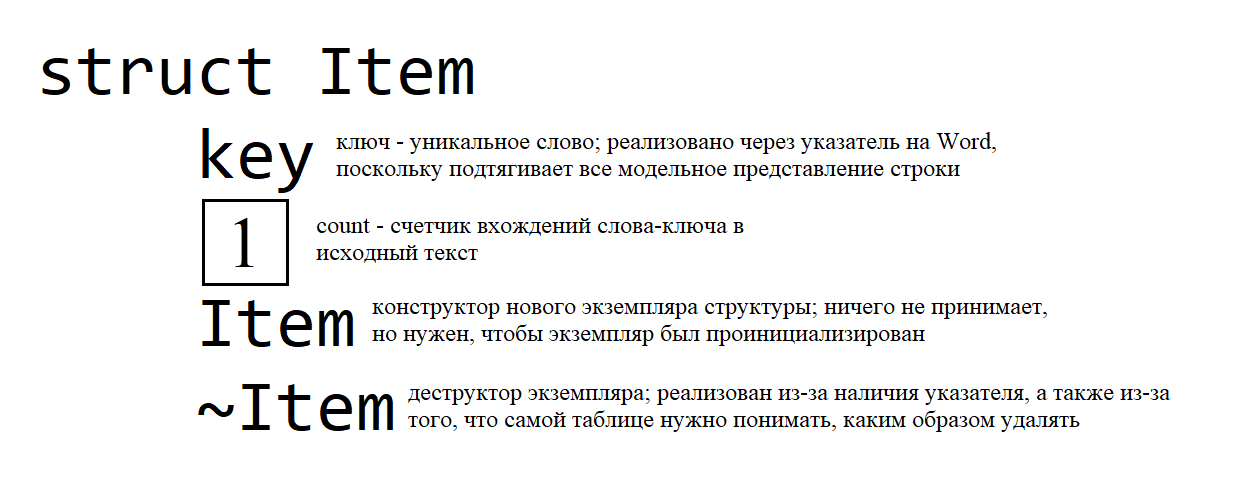


рис. 14 — описание структуры Item (базовый элемент таблицы)

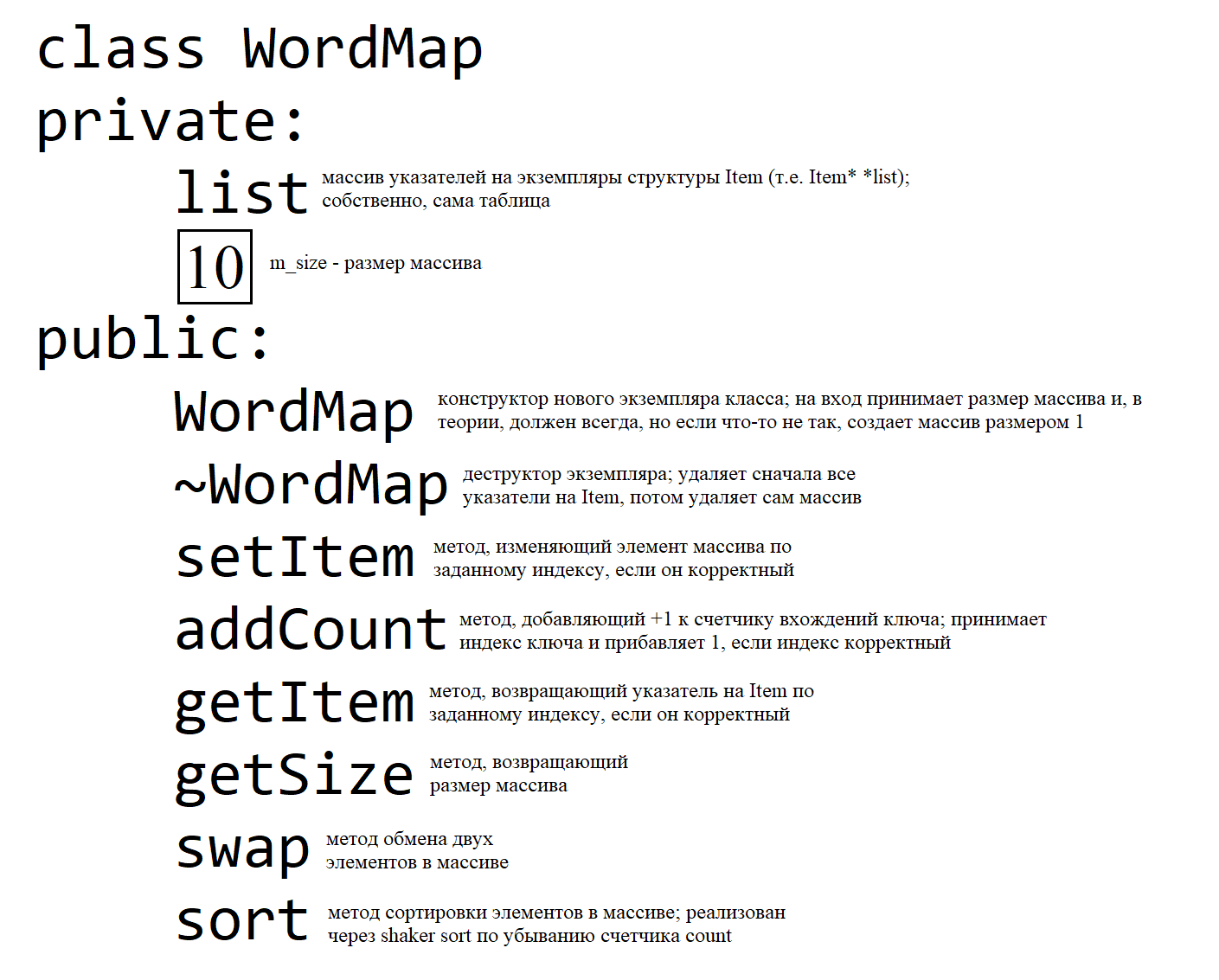


рис. 15 — описание класса WordMap (реализация таблицы)

По организации приложения — теперь пофайловое разбиение обязательно для всех программ. Раньше это просто допускалось в целях распределения кода и снижения визуальной нагрузки, теперь этот факт нужно огласить путем изображения так называемого «дерева подключений». По стародавним соглашениям за подключение модуля отвечает заголовочный файл (расширение .h), но в нем (также по соглашениям и соображениям оптимизации кода времен C) содержатся лишь описания элементов модуля, а описание реализации содержится в одноименном файле источника (source file, расширение .cpp). В итоговом приложении это выглядит так (рис. 16):

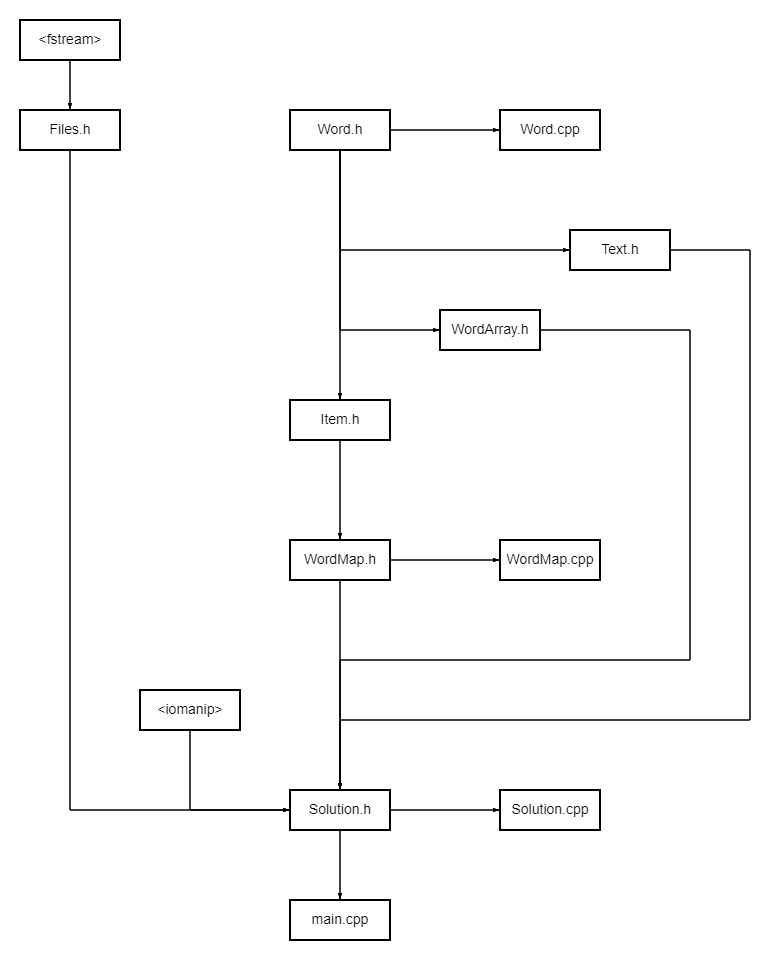


рис. 16 — «дерево подключений» при запуске main

И заканчиваем выводом ответа — таблицу нужно организовать. Если брать вывод из «Контрольного примера»:

get 2

ready 2

go 2

set 1

steady 1

то форматированный вариант будет выглядеть так:

----------------------------------------

|№ |Слово |Кол-во вхождений|

|1 |get |2 |

|2 |ready |2 |

|3 |go |2 |

|4 |set |1 |

|5 |steady |1 |

----------------------------------------

Соответственно, если под колонки номера и кол-ва вхождений места хватает на 4 позиции, то вот колонку со словами надо просчитывать. Именно поэтому к алгоритму, описанному в рис. 2, добавляется еще и определение максимально длинного слова. Если такое длиннее 5 символов, все колонки подстраиваются под него, иначе фиксировано 5 символов.

**Разработка интерфейса пользователя.**

Входной файл:

I1: текст

Выходной файл:

O1: Ошибка: входной файл не найден. (если такое произошло)

O2: Ошибка: входной файл пуст. (если в файле не обнаружено ни одного символа)

O3: Считан следующий текст:

O4: текст из I1

O5: Ошибка: в тексте не обнаружено ни одного слова. (если при подсчете не обнаружено ни одного слова)

O6: Итоговая таблица уникальных слов:

O7: - x (число символов в колонке слов + 25)

|№ |Слово…|Кол-во вхождений| (после «Слово» либо есть символы, либо нет)

Номер слова Слово Число вхождений

- x (число символов в колонке слов + 25)

Лог-файл:

O1: Программа начала свою работу.

O2: Входной файл не найден. Программа завершила свою работу. (если такое произошло)

O3: Входной файл пуст. Программа завершила свою работу. (если в файле не обнаружено ни одного символа)

O4: Считан следующий текст:

O5: текст из I1

O6: В тексте не обнаружено ни одного слова. Программа завершила свою работу. (если при подсчете не обнаружено ни одного слова)

O7: Записаны следующие слова:

O8: cccccc…

O9: Производится запись уникальных слов.

O10: Записаны уникальные слова. Производится сортировка таблицы по убыванию кол-ва вхождений.

O11: Итоговая таблица уникальных слов:

O12: - x (число символов в колонке слов + 25)

|№ |Слово…|Кол-во вхождений| (после «Слово» либо есть символы, либо нет)

Номер слова Слово Число вхождений

- x (число символов в колонке слов + 25)

O13: Программа завершила свою работу.

**Описание используемых данных.**

Структура Files:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| std::ifstream | inFile | public | входной файл |
| std::ofstream | outFile | выходной файл |
| logFile | лог-файл |

Класс Word:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| char\* | m\_text | private | массив символов (слово) |
| int | m\_size | привязанная длина слова |

Структура Text/WordArray:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| Word\*\* | lines/words | public | массив указателей на экземпляры Word (строки/слова) |
| int | size | длина массива |

Структура Item:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| Word\* | key | public | указатель на слово-ключ |
| int | count | привязанное кол-во вхождений |

Класс WordMap:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| Item\*\* | list | private | массив указателей на пары ключ-значение |
| int | m\_size | размер массива |

Класс Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Имя | Модификатор | Назначение |
| Files\* | files | private | указатель на объект Files |

Остальные переменные:

|  |  |  |
| --- | --- | --- |
| Тип | Имя | Назначение |
| Solution\* | solver (main) | указатель на экземпляр Solution для решения задачи |
| bool | flag (Solution::readSolveAndOut) | флаг для фиксирования нового уникального слова |
| swapped (WordMap::sort) | флаг для проверки сортировки |
| int | charsStrCount, charsWordCount (Solution::readSolveAndOut) | счетчик символов в подсчет строк/слов |
| linesCount, wordsCount, uniqueWords (Solution::readSolveAndOut) | счетчик строк/слов/уникальных слов, позже индексация строк/слов |
| startPos (Solution::readSolveAndOut) | начальная позиция считывания с файла/массива |
| maxChars (Solution::readSolveAndOut) | счетчик длины максимально длинного слова |
| curIndex (Solution::readSolveAndOut) | индексация уникальных слов |
| start, end (WordMap::sort) | индексы начала и конца сортировки таблицы |

**Описание используемых функций.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Имя | Файл | Назначение | Параметры | | | |
| входные | выходные | модифицируемые | транзитные |
| Word::Word() | Word.h | конструктор Word (выделяет память и заполняет ее) | text (по умолчанию пустая строка), size (по умолчанию 1) |  |  | size (переходит в setSize) |
| Word::~Word() | деструктор Word (освобождает память) |  |  |  |  |
| Word::setSize | Word.cpp | задание размера объекту | size (из setText или Word()) |  |  |  |
| Word::setText | задание текста объекту | text, size |  |  | size (переходит в setSize) |
| Word::getText | возврат текста у объекта |  |  |  |  |
| Word::getChar | возврат символа из текста объекта по индексу | index |  |  |  |
| Word::getSize | возврат размера текста объекта |  |  |  |  |
| Word::isEqual | сравнение двух объектов Word | other |  |  |  |
| Files::Files() | Files.h | конструктор Files (открывает файлы) | in, out, log |  |  |  |
| Files::~Files() | деструктор Files (закрывает файлы) |  |  |  |  |
| Text::Text() | Text.h | конструктор Text (выделяет память под массив) | size (по умолчанию 1) |  |  |  |
| Text::~Text() | деструктор Text (освобождает память от массива) |  |  |  |  |
| WordArray::WordArray() | WordArray.h | конструктор WordArray (выделяет память под массив) | size (по умолчанию 1) |  |  |  |
| WordArray::~WordArray() | деструктор WordArray (освобождает память от массива) |  |  |  |  |
| Item::Item() | Item.h | конструктор Item (выделяет память под Word\*) |  |  |  |  |
| Item::~Item() | деструктор Item (освобождает память от Word\*) |  |  |  |  |
| WordMap::WordMap() | WordMap.h | конструктор WordMap (выделяет память под массив) | size (по умолчанию 1) |  |  |  |
| WordMap::~WordMap() | деструктор WordMap (освобождает память от массива) |  |  |  |  |
| WordMap::setItem | WordMap.cpp | задание нового элемента по указанному индексу | index |  |  | key (поля переходят в Word::setText) |
| WordMap::addCount | прибавление счетчика | index |  |  |  |
| WordMap::getItem | возврат элемента из массива по указанному индексу | index |  |  |  |
| WordMap::getSize | возврат размера массива |  |  |  |  |
| WordMap::swap | перестановка элементов внутри массива | first, second |  |  |  |
| WordMap::sort | сортировка массива |  |  |  |  |
| Solution::Solution() | Solution.h | конструктор Solution (создать объект Files) |  |  |  |  |
| Solution::~Solution() | деструктор Solution (удалить объект Files) |  |  |  |  |
| Solution::checkFile | Solution.cpp | проверка наличия входного файла |  |  |  |  |
| Solution::readSolveAndOut | решение задачи и вывод ответа |  |  |  |  |

**Организация ввода-вывода.**

|  |  |  |  |
| --- | --- | --- | --- |
| Библиотека | Имя | Команда | Назначение |
| iomanip |  | setw | задание фиксированной ширины |
| setfill | задание заполнителя ширины |
| left | выравнивание по левому краю |
| right | выравнивание по правому краю |
| fstream | inFile | >> | считывание с файла |
| seekg | установка указателя на заданную позицию |
| tellg | возврат текущей позиции указателя |
| eof | проверка на EOF |
| clear | очистка флагов |
| unsetf | снятие флага |
| outFile, logFile | << | запись в файл |

**Представление алгоритма решения задачи.**

1. Проверить, есть ли входной файл

2. Прогнать файл линейно, подсчитать строки и символы

3. Если есть хотя бы один символ, продолжить обработку

4. Запустить запись текста в массив строк по алгоритму из рис. 2

5. Если есть хотя бы одно слово, продолжить обработку

6. Запустить запись слов в массив слов по алгоритму из рис. 4

7. Запустить запись слов в таблицу по алгоритму из рис. 6

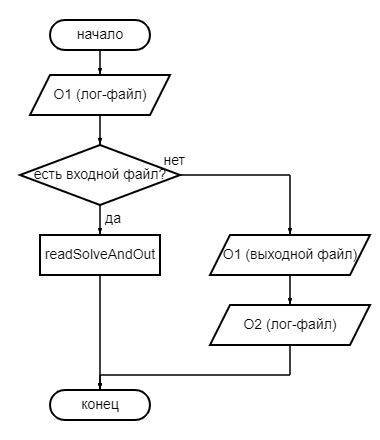
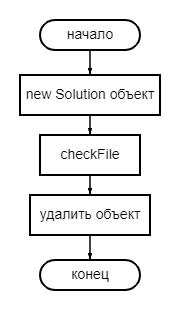
8. Отсортировать таблицу по алгоритму из рис. 8

9. Вывести таблицу по шаблонам

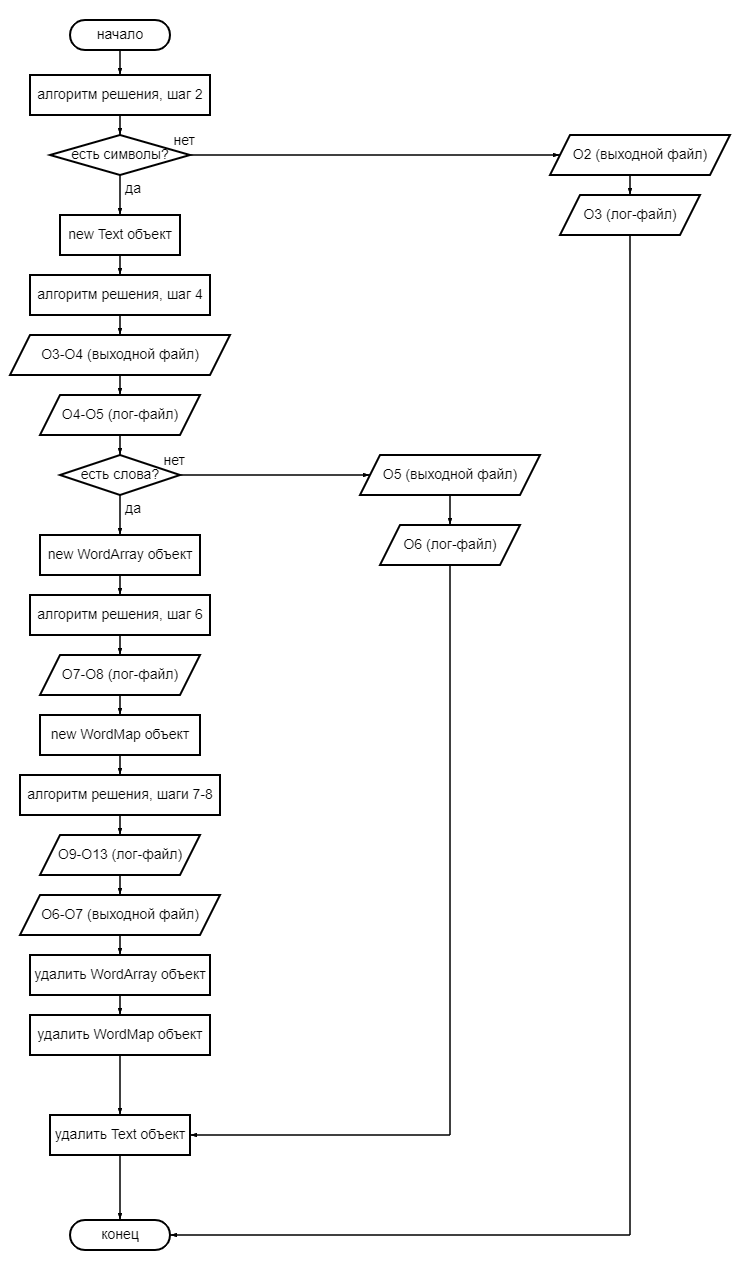
**Блок-схема.**

Описанию подлежит только самые важные части программы.

main и Solution::checkFile:



Solution::readSolveAndOut:



**Текст программы.**

main (main.cpp):

/\*

Программа по исследованию текста на уникальные слова

Автор: Савин Павел Алексеевич, группа 2309. Версия 2.1.2

Даты: Начало: 06.03.2023 Окончание: 20.03.2023

\*/

#include "Solution.h"

int main() {

Solution\* solver = new Solution;

solver->checkFile();

delete solver;

return 0;

}

Files.h (Files):

#pragma once

#include <fstream>

struct Files {

std::ifstream inFile;

std::ofstream outFile;

std::ofstream logFile;

Files(const char\* in = "in.txt", const char\* out = "out.txt", const char\* log = "log.txt") {

this->inFile.open(in);

this->outFile.open(out);

this->logFile.open(log);

}

~Files() {

this->inFile.close();

this->outFile.close();

this->logFile.close();

}

};

Word.h (Word):

#pragma once

class Word {

private:

char\* m\_text;

int m\_size;

void setSize(const int size);

public:

explicit Word(const char\* text = " ", const int size = 1) {

this->setSize(size);

this->m\_text = new char[size];

for (int i = 0; i < size; ++i) m\_text[i] = text[i];

}

~Word() {

delete[] this->m\_text;

}

void setText(const char\* text, const int size);

char\* getText();

char getChar(const int index);

int getSize();

bool isEqual(const Word\* other);

};

Word.cpp (Word):

#include "Word.h"

void Word::setSize(const int size) {

this->m\_size = size;

}

void Word::setText(const char\* text, const int size) {

delete[] this->m\_text;

this->setSize(size);

this->m\_text = new char[size];

for (int i = 0; i < size; ++i) m\_text[i] = text[i];

}

char\* Word::getText() {

return this->m\_text;

}

char Word::getChar(const int index) {

if (index >= 0 && index <= this->m\_size) return this->m\_text[index];

return 0;

}

int Word::getSize() {

return this->m\_size;

}

bool Word::isEqual(const Word\* other) {

if (this->m\_size == other->m\_size) {

for (int i = 0; i < this->m\_size; ++i) if (this->m\_text[i] != other->m\_text[i]) return false;

return true;

}

return false;

}

Text.h (Text):

#pragma once

#include "Word.h"

struct Text {

Word\* \*lines;

int size;

explicit Text(const int size = 1) {

this->lines = new Word\* [size];

for (int i = 0; i < size; ++i) this->lines[i] = new Word;

this->size = size;

}

~Text() {

for (int i = 0; i < this->size; ++i) delete this->lines[i];

delete[] this->lines;

}

};

WordArray.h (WordArray):

#pragma once

#include "Word.h"

struct WordArray {

Word\* \*words;

int size;

WordArray(const int size = 1) {

this->words = new Word\* [size];

for (int i = 0; i < size; ++i) this->words[i] = new Word;

this->size = size;

}

~WordArray() {

for (int i = 0; i < this->size; ++i) delete this->words[i];

delete[] this->words;

}

};

Item.h (Item):

#pragma once

#include "Word.h"

struct Item {

Word\* key;

int count;

Item() {

this->key = new Word;

this->count = 1;

}

~Item() {

delete this->key;

}

};

WordMap.h (WordMap):

#pragma once

#include "Item.h"

class WordMap {

private:

Item\* \*list;

int m\_size;

public:

explicit WordMap(const int size = 1) {

this->list = new Item\* [size];

for (int i = 0; i < size; ++i) this->list[i] = new Item;

this->m\_size = size;

}

~WordMap() {

for (int i = 0; i < this->m\_size; ++i) delete this->list[i];

delete[] this->list;

}

void setItem(Word\* key, const int index);

void addCount(const int index);

Item\* getItem(const int index);

int getSize();

void swap(Item\* first, Item\* second);

void sort();

};

WordMap.cpp (WordMap):

#include "WordMap.h"

void WordMap::setItem(Word\* key, const int index) {

if (index >= 0 && index < this->m\_size) {

delete this->list[index];

this->list[index] = new Item;

this->list[index]->key->setText(key->getText(), key->getSize());

}

}

void WordMap::addCount(const int index) {

if (index >= 0 && index < this->m\_size) this->list[index]->count += 1;

}

Item\* WordMap::getItem(const int index) {

if (index >= 0 && index < this->m\_size) return this->list[index];

}

int WordMap::getSize() {

return this->m\_size;

}

void WordMap::swap(Item\* first, Item\* second) {

Item\* tmp = new Item;

tmp->key->setText(first->key->getText(), first->key->getSize());

tmp->count = first->count;

first->key->setText(second->key->getText(), second->key->getSize());

first->count = second->count;

second->key->setText(tmp->key->getText(), tmp->key->getSize());

second->count = tmp->count;

delete tmp;

}

void WordMap::sort() {

bool swapped = true;

int start = 0;

int end = this->m\_size - 1;

while (swapped) {

swapped = false;

for (int i = start; i < end; ++i) {

if (this->list[i]->count < this->list[i + 1]->count) {

swap(this->list[i], this->list[i + 1]);

swapped = true;

}

}

if (!swapped) break;

swapped = false;

--end;

for (int i = end - 1; i >= start; --i) {

if (this->list[i]->count < this->list[i + 1]->count) {

swap(this->list[i], this->list[i + 1]);

swapped = true;

}

}

++start;

}

}

Solution.h (Solution):

#pragma once

#include "Files.h"

#include "Text.h"

#include "WordArray.h"

#include "WordMap.h"

#include <iomanip>

class Solution {

private:

Files\* files;

void readSolveAndOut();

public:

Solution() {

this->files = new Files;

}

~Solution() {

delete this->files;

}

void checkFile();

};

Solution.cpp (Solution):

#include "Solution.h"

void Solution::checkFile() {

this->files->logFile << "Программа начала свою работу.\n";

if (this->files->inFile.is\_open()) this->readSolveAndOut();

else {

this->files->outFile << "Ошибка: входной файл не найден.";

this->files->logFile << "Входной файл не найден.\nПрограмма завершила свою работу.";

}

}

void Solution::readSolveAndOut() {

this->files->inFile.unsetf(std::ios::skipws);

int charsStrCount = 0, linesCount = 1;

char tmp = 0;

do {

this->files->inFile >> tmp;

if (this->files->inFile.eof()) break;

if (tmp == '\n') ++linesCount;

++charsStrCount;

}

while (true);

if (charsStrCount) {

this->files->outFile << "Считан следующий текст:\n";

this->files->logFile << "Считан следующий текст:\n";

this->files->inFile.clear();

this->files->inFile.seekg(0, std::ios::beg);

Text\* text = new Text(linesCount);

int wordsCount = 0, charsWordCount = 0, maxChars = 0, startPos = 0;

charsStrCount = 0, linesCount = 0;

do {

this->files->inFile >> tmp;

if (!this->files->inFile.eof()) {

this->files->outFile << tmp;

this->files->logFile << tmp;

}

else {

if (charsStrCount) {

this->files->inFile.clear();

this->files->inFile.seekg(startPos, std::ios::beg);

char\* chars = new char[charsStrCount];

for (int i = 0; i < charsStrCount; ++i) this->files->inFile >> chars[i];

text->lines[linesCount]->setText(chars, charsStrCount);

++linesCount;

delete[] chars;

}

else text->lines[linesCount]->setText(" ", 1);

if (charsWordCount) ++wordsCount;

break;

}

if (tmp == '\n') {

if (charsStrCount) {

this->files->inFile.seekg(startPos, std::ios::beg);

char\* chars = new char[charsStrCount];

for (int i = 0; i < charsStrCount; ++i) this->files->inFile >> chars[i];

text->lines[linesCount]->setText(chars, charsStrCount);

++linesCount;

delete[] chars;

this->files->inFile >> tmp;

}

else text->lines[linesCount]->setText(" ", 1);

if (charsWordCount) ++wordsCount;

charsStrCount = 0;

charsWordCount = 0;

startPos = this->files->inFile.tellg();

}

else if (tmp == '\t' || (tmp >= 32 && tmp <= 47) || (tmp >= 58 && tmp <= 64) || (tmp >= 91 && tmp <= 96) || (tmp >= 123 && tmp <= 126)) {

if (charsWordCount) ++wordsCount;

if (charsWordCount > maxChars) maxChars = charsWordCount;

charsWordCount = 0;

++charsStrCount;

}

else {

++charsStrCount;

++charsWordCount;

}

}

while (true);

if (wordsCount) {

this->files->logFile << "\nЗаписаны следующие слова:\n";

WordArray\* words = new WordArray(wordsCount);

wordsCount = 0, startPos = 0, charsWordCount = 0, tmp = 0;

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

for (int j = 0; j <= text->lines[i]->getSize(); ++j) {

if (j != text->lines[i]->getSize()) tmp = text->lines[i]->getChar(j);

if (j == text->lines[i]->getSize() || tmp == '\t' || (tmp >= 32 && tmp <= 47) || (tmp >= 58 && tmp <= 64) || (tmp >= 91 && tmp <= 96) || (tmp >= 123 && tmp <= 126)) {

if (charsWordCount) {

this->files->logFile << '\n';

char\* chars = new char[charsWordCount];

for (int k = startPos; k < startPos + charsWordCount; ++k) {

chars[k - startPos] = text->lines[i]->getChar(k);

}

words->words[wordsCount]->setText(chars, charsWordCount);

delete[] chars;

++wordsCount;

}

charsWordCount = 0;

}

else {

if (!charsWordCount) startPos = j;

++charsWordCount;

this->files->logFile << tmp;

}

}

}

int uniqueWords = 0;

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

bool flag = false;

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

if (words->words[i]->isEqual(words->words[j])) {

flag = true;

break;

}

}

if (!flag) ++uniqueWords;

}

this->files->logFile << "Производится запись уникальных слов.\n";

WordMap\* map = new WordMap(uniqueWords);

int curIndex = 0;

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

bool flag = false;

int mapPos = 0;

for (int j = 0; j < uniqueWords; ++j) {

if (words->words[i]->isEqual(map->getItem(j)->key)) {

mapPos = j;

flag = true;

break;

}

}

if (!flag) {

map->setItem(words->words[i], curIndex);

++curIndex;

}

else map->addCount(mapPos);

}

this->files->logFile << "Записаны уникальные слова. Производится сортировка таблицы по убыванию кол-ва вхождений.\n";

map->sort();

this->files->logFile << "Итоговая таблица уникальных слов:\n";

this->files->outFile << "\nИтоговая таблица уникальных слов:\n";

this->files->logFile << std::setw((maxChars > 5 ? maxChars : 5) + 25) << std::setfill('-') << '\n';

this->files->outFile << std::setw((maxChars > 5 ? maxChars : 5) + 25) << std::setfill('-') << '\n';

this->files->logFile << "|№ |" << std::setw((maxChars > 5 ? maxChars : 5)) << std::setfill(' ') << std::left << "Слово" << "|Кол-во вхождений|\n";

this->files->outFile << "|№ |" << std::setw((maxChars > 5 ? maxChars : 5)) << std::setfill(' ') << std::left << "Слово" << "|Кол-во вхождений|\n";

for (int i = 0; i < map->getSize(); ++i) {

this->files->logFile << '|' << std::setw(4) << std::left << i + 1 << '|';

this->files->outFile << '|' << std::setw(4) << std::left << i + 1 << '|';

for (int j = 0; j < map->getItem(i)->key->getSize(); ++j) {

this->files->logFile << map->getItem(i)->key->getChar(j);

this->files->outFile << map->getItem(i)->key->getChar(j);

}

this->files->logFile << std::setw((maxChars > 5 ? maxChars : 5) - map->getItem(i)->key->getSize() + 1) << std::setfill(' ') << std::right << '|'; this->files->logFile << std::setw(16) << std::left << map->getItem(i)->count; this->files->logFile << "|\n";

this->files->outFile << std::setw((maxChars > 5 ? maxChars : 5) - map->getItem(i)->key->getSize() + 1) << std::setfill(' ') << std::right << '|'; this->files->outFile << std::setw(16) << std::left << map->getItem(i)->count; this->files->outFile << "|\n";

}

this->files->logFile << std::setw((maxChars > 5 ? maxChars : 5) + 24) << std::setfill('-') << '-';

this->files->outFile << std::setw((maxChars > 5 ? maxChars : 5) + 24) << std::setfill('-') << '-';

this->files->logFile << "\nПрограмма завершила свою работу.";

delete words;

delete map;

}

else {

this->files->outFile << "Ошибка: в тексте не обнаружено ни одного слова.";

this->files->logFile << "В тексте не обнаружено ни одного слова.\nПрограмма завершила свою работу.";

}

delete text;

}

else {

this->files->outFile << "Ошибка: входной файл пуст.";

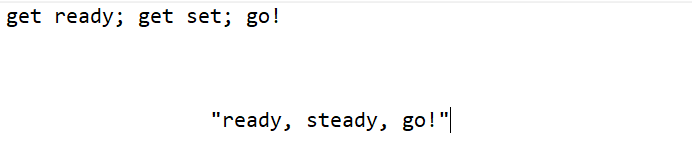
this->files->logFile << "Входной файл пуст.\nПрограмма завершила свою работу.";

}

}

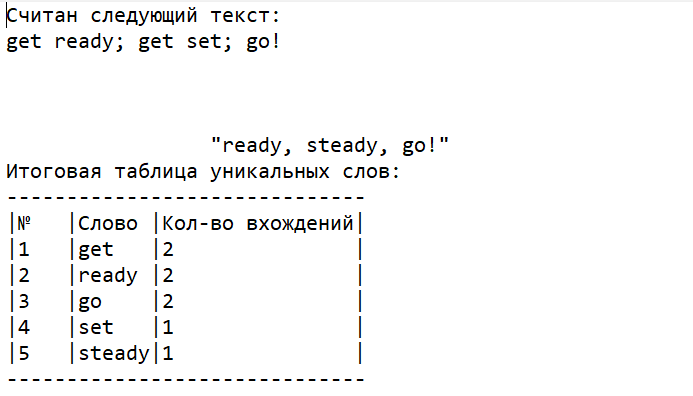
**Результаты работы программы.**

На примере следующего входного файла:

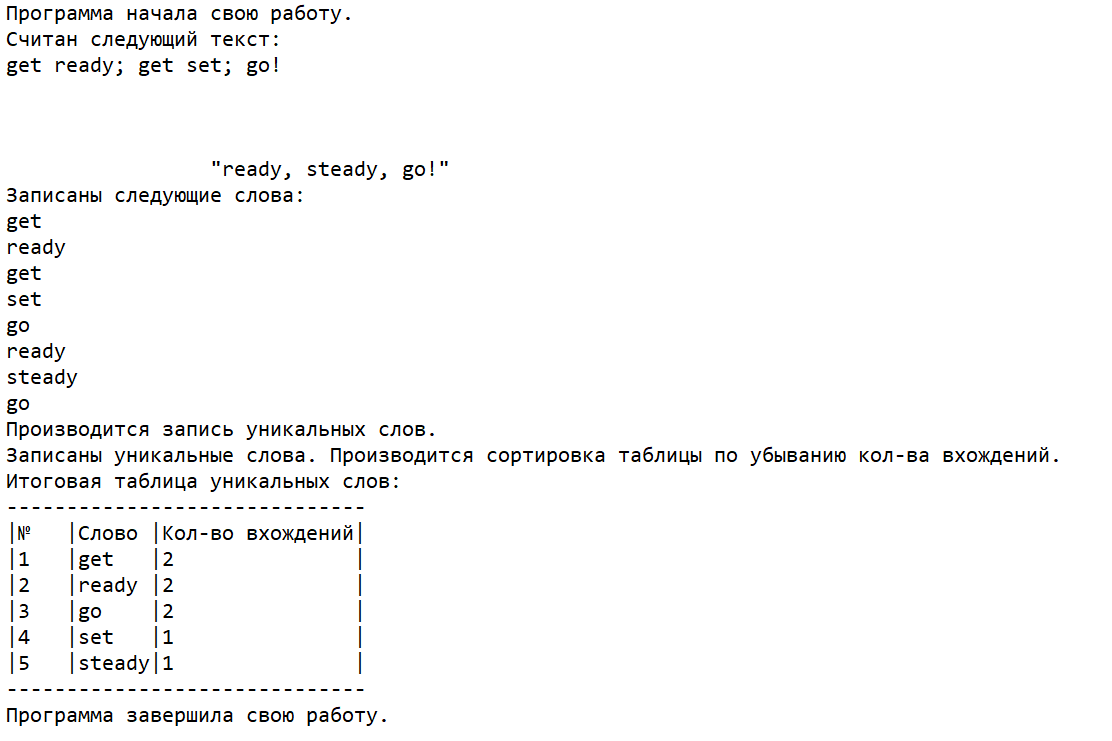


получены следующие результаты:

Выходной файл:



Лог-файл:



**Вывод о проделанной работе.**

Освоена работа с текстом, пофайловое разбиение приложения, также получены начальные знания о популярных структурах данных.