**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР.**

отчет

**по лабораторной работе №3**

**по дисциплине «Программирование»**

Тема: «Кортежи элементов строк и действия с ними».

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1309 |  | Ищенко Д.О. |
| Преподаватель |  | Калмычков В.А. |

Санкт-Петербург

2022

Оглавление

[Формулировка задания. 3](#_Toc98290200)

[Анализ задания. 3](#_Toc98290201)

[Способ внутреннего представления данных. 4](#_Toc98290202)

[Способ реализации ввода-вывода. 5](#_Toc98290203)

[Файловая организация программы. 5](#_Toc98290204)

[Алгоритмы решения задачи. 6](#_Toc98290205)

[Текст программы. 7](#_Toc98290206)

[Результаты работы программы. 13](#_Toc98290207)

[Вывод. 14](#_Toc98290208)

# Формулировка задания.

Задан текст, содержащий произвольное количество строк, в которых отдельные слова могут разделяться одним или несколькими пробелами и знаками пунктуации (переносы слов с одной строки на другую не используются). Сформировать новый текст, в котором каждое слово, встречающееся более одного раза в тексте заменить начиная со второго вхождения на обозначение вида: S(NN), где S - первый знак слова; NN - порядковый номер в тексте первого вхождения слова.

# Анализ задания.

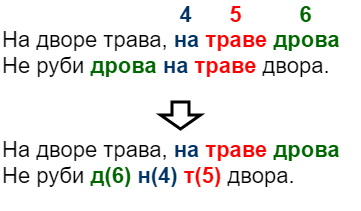


Рис. 1

Результатом работы программы должен являться тот же текст, в котором изменяются только повторяющиеся слова, начиная с каждого второго. В примере на рис. 1 предлог «на» заменяется только во второй части скороговорки. При этом все последующие вхождения этого предлога должны быть заменены таким же обозначением («н(3)»). Нумерация слов происходит с начала текста. Если посреди слова стоит какой-либо знак пунктуации, то это слово делится на два (например: стол()вая - это «стол» и «овая»), исключением являются апострофы и дефисы. К знакам пунктуации относятся:

. , ! ? | / () { } [ ] : ;

Исходное количество пробелов и переносов строки в тексте должно сохраняться.

# Способ внутреннего представления данных.

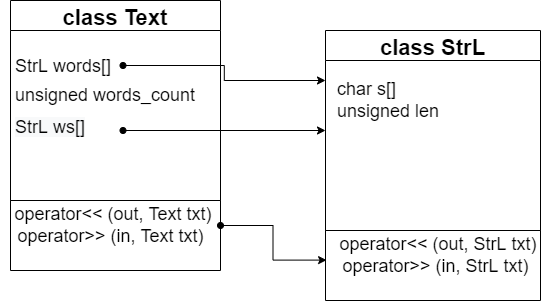


Рис. 2

Внутреннее хранение данных реализовано с использованием двух пользовательских классов: Text и StrL.

StrL является реализацией строкового представления массива символов с хранением длины задействованной части массива.

Text содержит три поля: words[] - массив строк StrL, который хранит слова из текста; words\_count - количество слов в тексте и ws[] - массив строк StrL, который хранит разделители между словами, то есть все что находится между, включая пробелы, переносы строк и знаки пунктуации.

К примеру, такой текст «Один ... :( Два!» будет храниться как words = {“Один”, “Два”}, words\_count = 2, ws = {“”,” ... :( ”,”!”}. Размер массива разделителей всегда на одну позицию больше, для того, чтобы можно было хранить как разделители между словами, так и разделители до и после (всего на 1 больше).

# Способ реализации ввода-вывода.

Для каждого класса перегружены операторы ввода/вывода для работы с файлами, при этом работа операторов с операндами типа Text базируется на работе операторов с операндами ипа StrL. Эти перегрузки позволяют считывать и записывать данные, используя синтаксис «in>>txt» и «out<<txt», где in, out - объекты файлов типа fstream, а txt - объект типа Text.

# Файловая организация программы.

**Main.cpp** содержит определение функций main и process. В функции process реализован алгоритм решения поставленной задачи с использованием описанных выше типов внутреннего хранения данных. См пункт «Алгоритмы решения задачи».

**Globals.h** содержит глобальные константы, такие как пути файлов, символы-разделители и другие.

**StrL.h** содержит объявление класса StrL без реализации методов, их определение и некоторых вспомогательных функций вынесено в файл **StrL.cpp**.

Аналогично, объявление класса Text содержится в **Text.h,** а реализация его методов в **Text.cpp.**

Порядок подключения файлов см на рис. 3. Проблема двойного включения файлов и дублирования кода решается с помощью использования директивы препроцессора pragma\_once.

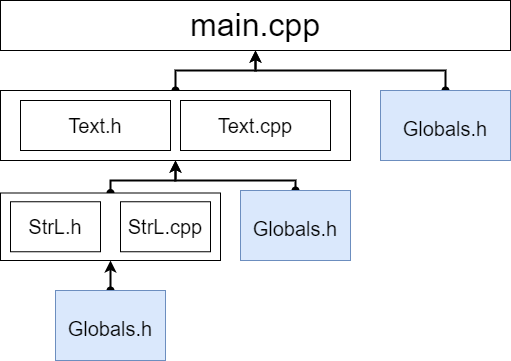
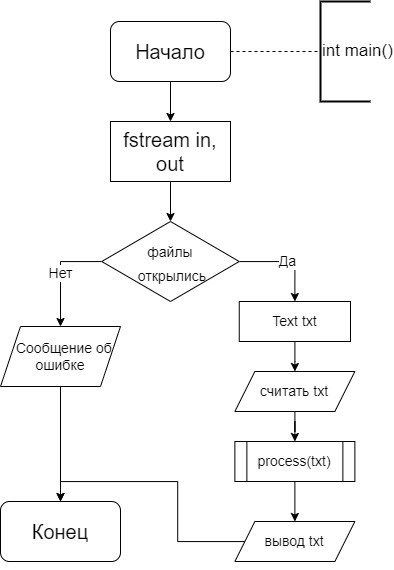
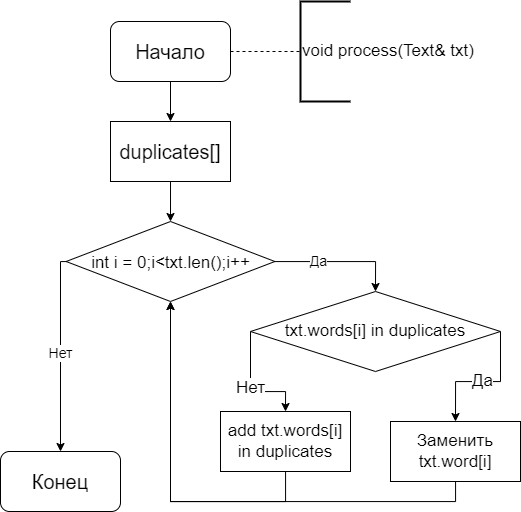
****

Рис. 3

# Алгоритмы решения задачи.

Ниже проиллюстрирован принцип работы функций main и proccess, которые отвечают за выполнение программы в целом и обработку данных соответственно.





# Текст программы.

**Main.cpp:**

#include <iostream>

#include "Text.h"

#include "Globals.h"

void process(Text& txt) {

unsigned txt\_len = txt.get\_count();

StrL\* duplicates = new StrL[txt\_len];

unsigned d\_ptr = 0;

unsigned\* d\_numbers = new unsigned[txt\_len];

StrL word, new\_w;

bool is\_doubles = false;

for (int i = 0; i < txt\_len; i++) {

word = txt.get\_word(i);

for (int j = 0; j < d\_ptr; j++) {

if (word == duplicates[j]) {

char one, two, three, four;

unsigned num = d\_numbers[j];

one = (char)(num / 1000) + 48;

num %= 1000;

two = (char)(num / 100) + 48;

num %= 100;

three = (char)(num / 10) + 48;

num %= 10;

four = (char)(num + 48);

if (one != '0')

new\_w = StrL(word.get\_char(0), '(', one, two,three,four, ')','\0');

else if (two!='0')

new\_w = StrL(word.get\_char(0), '(', two, three, four, ')', '\0');

else if (three != '0')

new\_w = StrL(word.get\_char(0), '(', three, four, ')', '\0');

else

new\_w = StrL(word.get\_char(0), '(', four, ')', '\0');

txt.set\_word(i, new\_w);

is\_doubles = true;

break;}

is\_doubles = false;}

if(!is\_doubles) {

duplicates[d\_ptr] = word;

d\_numbers[d\_ptr] = i+1;

d\_ptr++;}}}

int main() {

std::fstream in(INPUT\_PATH),

out(OUTPUT\_PATH, std::fstream::trunc | std::fstream::out);

if (in.is\_open() && out.is\_open()) {

Text txt;

in >> txt;

process(txt);

out << txt;

}else { std::cout << "files not open";}

in.close(); out.close();}

**Globals.h:**

#pragma once

#include <string>

const unsigned WORDS\_COUNT = 1024;

const unsigned WORD\_LEN = 32;

const std::string INPUT\_PATH = "input.txt";

const std::string OUTPUT\_PATH = "output.txt";

const char SEPARATORS[] = { '.', ',', ' ', '\n', '!', '?', '|', '/', '(', ')', '{', '}', '[', ']', ':', ';'};

**Text.h**

#pragma once

#include "StrL.h"

#include "Globals.h"

class Text{

StrL words[WORDS\_COUNT];

unsigned words\_count = 0;

StrL ws[WORDS\_COUNT + 1];

public:

void set\_word(unsigned word\_n, StrL s);

Text(const StrL\* s = nullptr, unsigned count = 0, const StrL\* ws\_ = nullptr);

const StrL& get\_word(unsigned n) const;

const unsigned get\_count() const;

const StrL& get\_ws(unsigned n) const;

friend std::fstream& operator>>(std::fstream& in, Text& txt);};

std::fstream& operator<<(std::fstream& out, const Text& txt);

**Text.cpp**

#include "Text.h"

Text::Text(const StrL\* s, unsigned count, const StrL\* ws\_) : words\_count(count) {

for (int i = 0; i < count; i++)

words[i] = s[i];

for (int i = 0; i < count+1; i++)

if (count !=0)

ws[i] = ws\_[i];}

const StrL& Text::get\_word(unsigned n) const {

return words[n];}

const unsigned Text::get\_count() const {

return words\_count;}

void Text::set\_word(unsigned word\_n, StrL s) {

words[word\_n] = s;}

const StrL& Text::get\_ws(unsigned n) const {

return ws[n];}

std::fstream& operator<<(std::fstream& out, const Text& txt) {

for (int i = 0; i < txt.get\_count(); i++) {

out << txt.get\_ws(i) << txt.get\_word(i); }

out << txt.get\_ws(txt.get\_count());

return out;}

unsigned get\_words\_count(std::fstream& in) {

unsigned w\_count = 0;

char tmp[WORD\_LEN];

while (!in.eof()) {

in >> tmp;

if (tmp != "\n" && tmp != "\0" && strlen(tmp) != 0) {

w\_count++; }}

in.close();

in.open(INPUT\_PATH);

return w\_count;}

bool is\_sep(char ch) {

for (int i = 0;i<sizeof(SEPARATORS)/sizeof(char); i++)

{if (ch == SEPARATORS[i])

return true;}

return false;}

std::fstream& operator>>(std::fstream& in, Text& txt) {

unsigned w\_count = get\_words\_count(in);

StrL\* words = new StrL[w\_count];

StrL\* ws = new StrL[w\_count + 1];

unsigned ws\_ptr = 0, words\_ptr = 0, w\_ptr = 0;

char t;

char\* tmp = new char[WORD\_LEN]; bool is\_ws = true;

t = in.get();

while (!in.eof()) {

while (is\_sep(t) && !in.eof()) {

if (!is\_ws) {

words[words\_ptr] = StrL(tmp, w\_ptr);

words\_ptr++;

w\_ptr = 0;}

is\_ws = true;

tmp[w\_ptr] = t;

w\_ptr++;

t = in.get();}

while (!is\_sep(t) && !in.eof()) {

if (is\_ws) {

ws[ws\_ptr] = StrL(tmp, w\_ptr);

ws\_ptr++;

w\_ptr = 0;

is\_ws = false;}

tmp[w\_ptr] = t;

w\_ptr++;

t = in.get();}

if (in.eof()) {

if (is\_ws) {ws[ws\_ptr] = StrL(tmp, w\_ptr); }

else {words[words\_ptr] = StrL(tmp, w\_ptr);}}}

txt = Text( words, w\_count, ws);

return in;}

**StrL.h**

#pragma once

#include <fstream>

#include <stdarg.h>

#include "Globals.h"

class StrL{

char s[WORD\_LEN];

unsigned len = 0;

public:

StrL(const char\* s, unsigned len);

explicit StrL(const char ch, ...);

StrL() { }

const char\* get\_s() const { return s; }

const unsigned get\_len() const { return len; }

void set\_word(const char\* str);

void set\_len(unsigned n) { len = n; }

char get\_char(unsigned n) const { if (n < len) return s[n]; }};

std::fstream& operator<<(std::fstream& out, const StrL& s);

std::fstream& operator>>(std::fstream& in, StrL& s);

bool operator == (const StrL& a, const StrL& b);

**StrL.cpp**

#include "StrL.h"

std::fstream& operator<<(std::fstream& out, const StrL& s) {

const char\* word = s.get\_s();

for (int i = 0; i < s.get\_len(); i++) {

out.put(word[i]);}

return out;}

void StrL::set\_word(const char\* str) {

for (int i = 0; i < WORD\_LEN; i++)

s[i] = str[i];}

std::fstream& operator>>(std::fstream& in, StrL& s) {

char word[WORD\_LEN];

in >> word;

s.set\_word(word);

char t = word[0]; unsigned count = 0;

while (t != '\0') {

count++;

t = word[count];}

s.set\_len(count);

return in;}

StrL::StrL(const char ch, ...) {

len++;

s[0] = ch;

va\_list list;

va\_start(list, ch);

while (true) {

char t = va\_arg(list, char);

if (t == '\0')

break;

s[len] = t;

len++;}

va\_end(list);}

StrL::StrL(const char\* s, unsigned len) : len(len) {

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

this->s[i] = s[i];}}

bool operator == (const StrL& a, const StrL& b) {

if (a.get\_len() != b.get\_len())

return false;

for (int i = 0; i < a.get\_len(); i++) {

if (a.get\_char(i) != b.get\_char(i))

return false;}

return true;}

# Результаты работы программы.

**Input.txt:**

The most elementary part of a mesh is the vertex

(vertices plural) which is a single point or position

in 3D space. Vertices are represented in the 3D Viewport

in Edit Mode as small dots. The vertices of an object are

stored as an array of coordinates.

An edge always connects two vertices by a straight line.

The edges are the “wires” you see when you look at a mesh in

wireframe view. They are usually invisible on the rendered

image. They are used to construct faces.

**Output.txt**

The most elementary part of a mesh is the vertex

(vertices plural) which i(8) a(6) single point or position

in 3D space. Vertices are represented i(20) t(9) 3(21) Viewport

i(20) Edit Mode as small dots. T(1) v(11) o(5) an object a(24)

stored a(33) a(39) array o(5) coordinates.

An edge always connects two v(11) by a(6) straight line.

T(1) edges a(24) t(9) “wires” you see when y(63) look at a(6) m(7) i(20)

wireframe view. They a(24) usually invisible on t(9) rendered

image. T(74) a(24) used to construct faces.

# Вывод.

В ходе выполнения лабораторной работы была написана программа, решающая поставленную задачу. Во время её разработки были получены практические навыки работы с различными конструкциями языка с++, такими как директивы препроцессора, перегрузка операторов, отделение объявления от определения методов. А также были получены навыки разбиения программы на модули и реализации пользовательских типов строк.