**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по курсовой работе**

**по дисциплине «Программирование»**

**Тема: «Обработка текстовой информации»**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
|  |  |  |
| Преподаватель |  | Чайка К.В. |

Санкт-Петербург, 2021

**Цель работы.**

Цель работы заключается в написании программы, выполняющей разнообразную обработку текста, сохраненного в динамический массив.

**Задание. Вариант 23:**

Программе на вход подается текст (текст представляет собой предложения, разделенные точкой. Предложения - набор слов, разделенные пробелом или запятой, слова - набор латинских букв и цифр. Длина текста и каждого предложения заранее не известна.

Программа должна сохранить этот текст в динамический массив строк и оперировать далее только с ним.

Программа должна найти и удалить все повторно встречающиеся предложения (сравнивать их следует посимвольно, но без учета регистра).

Далее, программа должна запрашивать у пользователя одно из следующих доступных действий (программа должна печатать для этого подсказку. Также следует предусмотреть возможность выхода из программы):

1. Удалить все четные по счету предложения в которых четное количество слов.
2. Отсортировать все слова в предложениях по возрастанию количества букв в верхнем регистре в слове.
3. Заменить все слова в тексте длина которых не более 3 символов на подстроку “Less Then 3”.
4. Найти в каждом предложении строку максимальной длины, которая начинается и заканчивается цифрой. Вывести найденные подстроки по убыванию длины подстроки.

Все сортировки должны осуществляться с использованием функции стандартной библиотеки. Использование собственных функций, при наличии аналога среди функций стандартной библиотеки, запрещается.

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

**Выполнение работы.**

Создаются структуры *struct sentence* и *struct text.*

*Struct sentence* имеет 3 поля:

* *char\* str —* вводимое предложение.
* *int size —* размер предложения в байтах.
* *int n —* количество символов в предложении.

*Struct text* имеет 3 поля:

* *struct sentence\*\* line\_pile —* массив, состоящий из *struct sentence.*
* *Int size —* размер структуры в байтах.
* *Int n —* количество элементов типа *struct sentence* в структуре.

Функция *struct sentence\* readstring()* посимвольно считывает предложения, пока не встретит признак конца ввода данного предложения, динамически выделяя память под сохраняемое предложение. После этого функция создает переменную типа *struct sentence* и заполняет ее данными, полученными в ходе считывания предложения. Функция возвращает ссылку на данную структуру в качестве результата.

Функция *struct text readtext()* динамически выделяет память под поле *line\_pile,* после чего вызывает функцию *readstring(),* пока не встретит на вводе два подряд идущих символа переноса строки (*„\n“).* Функция возвращает элемент типа *struct text.*

Функция *void deleting\_spaces(struct text txt)* принимает на вход элемент типа *struct text* и удаляет пробелы в начале предложения с помощью сдвига памяти, если таковые имеются.

Функция *void sentence\_checker(struct text\* txt)* принимает на вход ссылку на элемент типа *struct text* и проверяет на наличие повторяющихся предложений. Если таковые имеются, функция удаляет повторы, оставляя первое появление предложения в тексте. В случае, если предложение удаляется, то поле *n* у поданной на вход *struct text* уменьшается на количество удаленных предложений.

Функция *void even\_with\_even(struct text\* txt)* соответствует задаче из пункта 1 условия и удаляет четные по счету предложения, в которых четное количество слов. Для проверки предложения на четность количества слов сначала создается копия исходной строки, к которой позднее применяется функция *strtok(),* с каждой итерацией которой счетчик увеличивается на 1. Если после окончания цикла с *strtok()* счетчик оказался четным числом, мы удаляем текущее предложение из текста и перевыделяем память, выделенную под текст, по количеству оставшихся предложений.

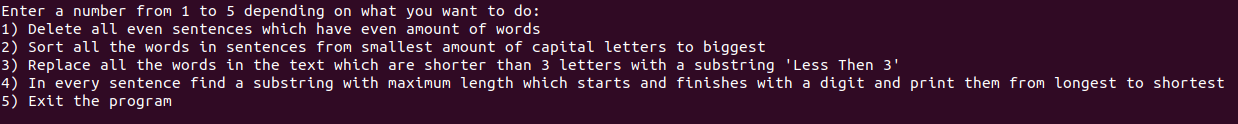
Функция *void sorter\_for\_uppercase(struct sentence\* sent)* соответствует задаче из пункта 2 условия и сортирует слова в предложении по возрастанию количества заглавных букв. Для этого определяется количество слов в предложении, после чего оно разбивается на массив из слов с помощью *strtok().* Далее этот массив сортируется с помощью *qsort()* с функцией-компаратором *compar\_for\_words.* Далее с помощью команд *strcpy()* и *strcat()* отсортированный массив собирается в предложение, которое заменяет собой изначально поданное в функцию.

Функция *void shorts\_replacer(struct text\* txt)* соответствует задаче из пункта 3 условия и заменяет в предложениях слова, длина которых меньше или равна 3-м символам, на подстроку «Less then 3». Для этого аналогично функции *sorter\_for\_uppercase()* создается массив со словами из предложений, которые проверяются на длину, и если слово нужно заменить, то во время сборки нового предложения ему довыделяется память, соразмерная заменяющей подстроке.

Функция *void finding\_a\_substring(struct text\* txt)* соответствует задаче из пункта 4 условия и находит в каждом предложении подстроку максимальной длины, которая начинается и заканчивается цифрой. Для этого выделяется память под двумерный массив, в который будут помещены найденные подстроки. Переменным, означающим индексы первой и последней цифры в подстроке присваивается значение -1, после чего два раза по предложению проходит цикл (один раз с начала предложения, второй — с конца) и находит первое вхождение цифры в цикле. Если индексы оказались равны друг другу (1 цифра в предложении) или оба индекса остались равны -1, то функция переходит к следующему предложению из текста, иначе в массив помещается копия предложения начиная с первого вхождения цифры в предложении и заканчивая последним. Далее полученный массив сортируется с помощью функции *qsort()*, в который передается *compar\_for\_substrings()* как компаратор.

После сортировки полученные подстроки выводятся на экран с новой строки в порядке убывания длины.

В функции *main()* производится ввод текста путем создания *struct text test,* которой присваивается результат функции *readtext().* Далее вызываются команды *deleting\_spaces(text)* и *sentence\_checker(&test),* которые убирают пробелы в начале предложений и повторяющиеся предложения, после чего выводится меню программы:



Далее из строки ввода считывается один символ (цифра от 1 до 5), который определяет, что будет делать программа. Нумерация команд соответствует номерам подзадач из условия. Если ввести 5 — программа завершится и сообщит об этом:



**Результаты тестирования.**

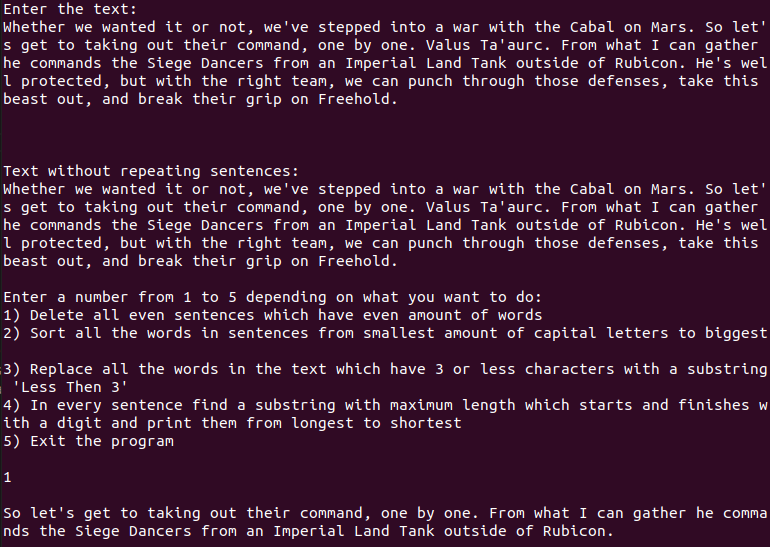
1. Команда «1».

Входные данные:

Whether we wanted it or not we've stepped into a war with the Cabal on Mars. So let's get to taking out their command one by one. Valus Ta'aurc. From what I can gather he commands the Siege Dancers from an Imperial Land Tank outside of Rubicon. He's well protected but with the right team we can punch through those defenses take this beast out and break their grip on Freehold.

На выходе получаем:

So let's get to taking out their command one by one. From what I can gather he commands the Siege Dancers from an Imperial Land Tank outside of Rubicon.



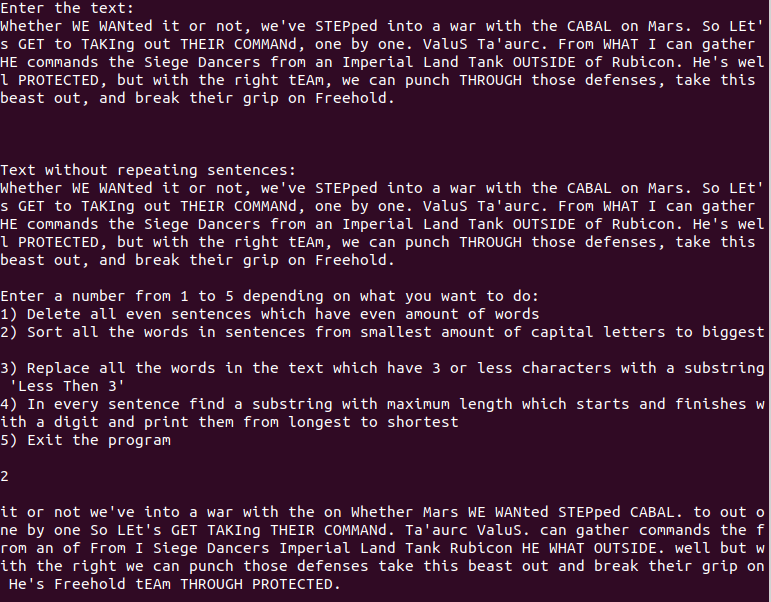
1. Команда «2»

Входные данные:

Whether WE WANted it or not we've STEPped into a war with the CABAL on Mars. So LEt's GET to TAKIng out THEIR COMMANd one by one. ValuS Ta'aurc. From WHAT I can gather HE commands the Siege Dancers from an Imperial Land Tank OUTSIDE of Rubicon. He's well PROTECTED but with the right tEAm we can punch THROUGH those defenses take this beast out and break their grip on Freehold.

На выходе получаем:

it or not we've into a war with the on Whether Mars WE WANted STEPped CABAL. to out one by one So LEt's GET TAKIng THEIR COMMANd. Ta'aurc ValuS. can gather commands the from an of From I Siege Dancers Imperial Land Tank Rubicon HE WHAT OUTSIDE. well but with the right we can punch those defenses take this beast out and break their grip on He's Freehold tEAm THROUGH PROTECTED.



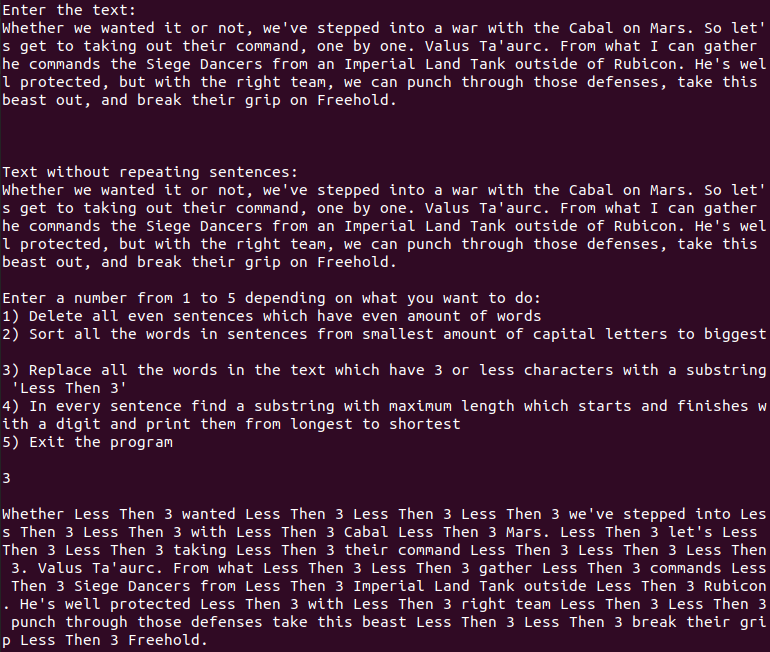
1. Команда «3»

Входные данные:

Whether we wanted it or not we've stepped into a war with the Cabal on Mars. So let's get to taking out their command one by one. Valus Ta'aurc. From what I can gather he commands the Siege Dancers from an Imperial Land Tank outside of Rubicon. He's well protected but with the right team we can punch through those defenses take this beast out and break their grip on Freehold.

На выходе получаем:

Whether Less Then 3 wanted Less Then 3 Less Then 3 Less Then 3 we've stepped into Less Then 3 Less Then 3 with Less Then 3 Cabal Less Then 3 Mars. Less Then 3 let's Less Then 3 Less Then 3 taking Less Then 3 their command Less Then 3 Less Then 3 Less Then 3. Valus Ta'aurc. From what Less Then 3 Less Then 3 gather Less Then 3 commands Less Then 3 Siege Dancers from Less Then 3 Imperial Land Tank outside Less Then 3 Rubicon. He's well protected Less Then 3 with Less Then 3 right team Less Then 3 Less Then 3 punch through those defenses take this beast Less Then 3 Less Then 3 break their grip Less Then 3 Freehold.



1. Команда «4»

Входные данные:

6 brave guardians went down to the Vault of Glass, but only 3 of them succeed. It was a great challenge afterall. There were 5 noble deeds they performed in less than 2 hours. But they've killed only 3 harpys out of 6, what a shame. During the oracles there were 7 sequences and brave guardians have completed only 4 of them. At the end, they've defeated the mighty Atheon.

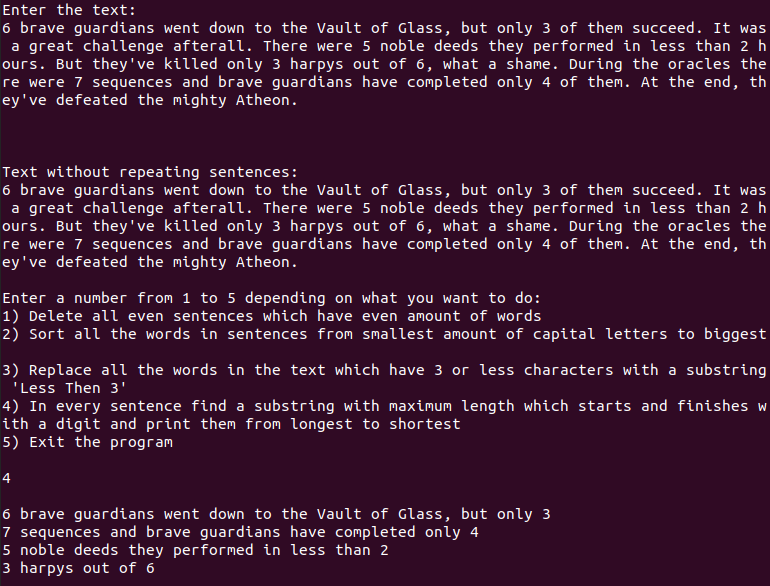
На выходе получаем:

6 brave guardians went down to the Vault of Glass, but only 3

7 sequences and brave guardians have completed only 4

5 noble deeds they performed in less than 2

3 harpys out of 6



**Выводы.**

Используя знания, полученные во время изучения дисциплины «Программирование», средствами языка «Си» удалось написать программу, выполняющую разнообразную обработку получаемого на вход текста.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: course\_work.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MEM\_STEP 100

struct sentence{

char \*str;

int size; // mem size

int n; // strlen

};

struct text{

struct sentence \*\*line\_pile; // pointer to array of sentences

int size;

int n; //count of sentences

};

struct sentence\* readstring();

struct text readtext(){

int size = MEM\_STEP\*sizeof(struct text);

struct sentence\*\* text= malloc(size\*sizeof(struct sentence\*));

int n = 0;

struct sentence\* temp;

int nlcount = 0;

do{

if (n >= size-2){

struct sentence\*\* t= realloc(text, (size + MEM\_STEP)\*sizeof(struct sentence\*));

if(!t){

puts("Reallocation error. Further result does not represent the intended one.");

}

size += MEM\_STEP\*sizeof(struct sentence);

text = t;

};

temp = readstring();

if(temp->str[0] == '\n')

nlcount++;

else{

nlcount = 0;

text[n] = temp;

n++;

}

}while(nlcount < 2);

struct text txt;

txt.size = size;

txt.line\_pile = text;

txt.n = n;

return txt;

};

struct sentence\* readstring(){

int size = MEM\_STEP;

char \*buf = malloc (size\*sizeof(char));

char temp;

int n = 0;

do{

if (n >= size-2){

char \*t = realloc(buf, size + MEM\_STEP);

if(!t){

puts("Reallocation error. Further result does not represent the intended one.");

}

size += MEM\_STEP;

buf = t;

};

temp = getchar();

buf[n] = temp;

n++;

}while(temp != '\n' && temp != '.' && temp != '!' && temp != '?');

buf[n] = '\0';

struct sentence \*sentence = malloc(sizeof(struct sentence));

sentence->str = buf;

sentence->size = size;

sentence->n = n-1;

return sentence;

};

void deleting\_spaces(struct text txt){

for (int i = 0; i < txt.n; i++){

int curr = 0;

char\* curr\_char;

for(curr\_char = txt.line\_pile[i]->str; \*curr\_char == ' ' || \*curr\_char == '\t'; curr\_char++ ){

curr++;

}

memmove(txt.line\_pile[i]->str, curr\_char, strlen(txt.line\_pile[i]->str));

};

}

void sentence\_checker(struct text \*txt){

int first\_index = 0;

while (first\_index < txt->n-1){

char\* comp1 = malloc(strlen(txt->line\_pile[first\_index]->str)\*sizeof(char));

strcpy(comp1, txt->line\_pile[first\_index]->str);

for (int i = 0; i < strlen(comp1); i++){

if (isupper(comp1[i]) != 0){

comp1[i] = tolower(comp1[i]);

}

}

int second\_index = first\_index + 1;

while (second\_index < txt->n){

char\* comp2 = malloc(strlen(txt->line\_pile[second\_index]->str)\*sizeof(char));

strcpy(comp2, txt->line\_pile[second\_index]->str);

for (int i = 0; i < strlen(comp2); i++){

if (isupper(comp2[i]) != 0){

comp2[i] = tolower(comp2[i]);

}

}

if (strcmp(comp1, comp2) == 0){

for (int i = second\_index; i < txt->n; ++i){

txt->line\_pile[i] = txt->line\_pile[i+1];

}

second\_index--;

txt->n--;

free(comp2);

}

second\_index++;

}

first\_index++;

free(comp1);

}

struct sentence\*\* t= realloc(txt->line\_pile, txt->n\*sizeof(struct sentence\*));

if(!t){

puts("Reallocation error. Further result does not represent the intended one.");

}

txt->line\_pile = t;

}

void even\_with\_even(struct text \*txt){

int flag = 0;

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

if (i % 2 == 0 && flag == 0 || i % 2 == 1 && flag == 1){

char\* temp\_string = malloc(strlen(txt->line\_pile[i]->str)\*sizeof(char));

int word\_count = 0;

strcpy(temp\_string, txt->line\_pile[i]->str);

char\* word\_token = strtok(temp\_string, " ,");

while(word\_token != NULL){

word\_count++;

word\_token = strtok(NULL, " ,");

}

if (word\_count % 2 == 0){

for (int j = i; j < txt->n-1; j++){

txt->line\_pile[j] = txt->line\_pile[j+1];

}

txt->n--;

if (flag == 0){

flag = 1;

}else{

flag = 0;

}

i--;

}

free(temp\_string);

}

}

struct sentence\*\* t= realloc(txt->line\_pile, txt->n\*sizeof(struct sentence\*));

if(!t){

puts("Reallocation error. Further result does not represent the intended one.");

}

txt->line\_pile = t;

}

int compar\_for\_words (const void \*p1, const void \*p2){

int uppers\_in\_p1 = 0;

int uppers\_in\_p2 = 0;

char\* p11 = \*(char\*\*)p1;

char\* p22 = \*(char\*\*)p2;

for (int i = 0; i < strlen(p11); i++){

if (isupper(p11[i]) != 0){

uppers\_in\_p1++;

}

}for (int i = 0; i < strlen(p22); i++){

if (isupper(p22[i]) != 0){

uppers\_in\_p2++;

}

}

if (uppers\_in\_p1 < uppers\_in\_p2){

return -1;

}

if (uppers\_in\_p1 > uppers\_in\_p2){

return 1;

}

if (uppers\_in\_p1 == uppers\_in\_p2){

return 0;

}

}

int compar\_for\_substrings (const void \*p1, const void \*p2){

if (strlen(\*(char\*\*)p1) > (strlen(\*(char\*\*)p2))){

return -1;

}

if (strlen(\*(char\*\*)p1) < (strlen(\*(char\*\*)p2))){

return 1;

}

if (strlen(\*(char\*\*)p1) == (strlen(\*(char\*\*)p2))){

return 0;

}

}

void sorter\_for\_uppercase(struct sentence\* sent){

int k = 1;

for(int i = 0; sent->str[i]; i++){

if(isspace(sent->str[i])){

k++;

}

}

char \*new\_sentence = malloc(strlen(sent->str)\*sizeof(char));

char \*\*splitted\_sentences = malloc(k\*sizeof(char\*));

char \*temp;

temp = strtok(sent->str, " ,.");

int i = 0;

while(temp != NULL){

splitted\_sentences[i]=temp;

i++;

temp = strtok(NULL, " ,.");

}

qsort(splitted\_sentences, k, sizeof(char\*), compar\_for\_words);

strcpy(new\_sentence, splitted\_sentences[0]);

if (i == 1){

strcat(new\_sentence, “.”);

}else{

strcat(new\_sentence, " ");

for (int j = 1; j < k; j++){

strcat(new\_sentence, splitted\_sentences[j]);

if( j !=k-1){

strcat(new\_sentence, " ");

}

else{

strcat(new\_sentence, ".");

}

}

}

strcpy(sent->str, new\_sentence);

free(new\_sentence);

free(splitted\_sentences);

}

void shorts\_replacer(struct text\* txt){ // FREE ERROR

char\*\*\* splitted\_sentences = malloc(txt->n\*sizeof(char\*\*));

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

char\* copy\_of\_sentence = malloc(strlen(txt->line\_pile[i]->str)\*sizeof(char));

strcpy(copy\_of\_sentence, txt->line\_pile[i]->str);

int amount\_of\_words = MEM\_STEP;

splitted\_sentences[i] = malloc (amount\_of\_words\*sizeof(char\*));

int n = 0;

char\* word\_token = strtok(copy\_of\_sentence, " ,.");

do{

if (n >= amount\_of\_words-2){

char\*\* t = realloc (splitted\_sentences[i], (amount\_of\_words+MEM\_STEP)\*sizeof(char\*));

if (!t){

puts("Reallocation error. Further result does not represent the intended one.");

};

amount\_of\_words += MEM\_STEP;

splitted\_sentences[i] = t;

}

splitted\_sentences[i][n] = malloc((strlen(word\_token)+1)\*sizeof(char));

splitted\_sentences[i][n] = word\_token;

word\_token = strtok(NULL, " ,.");

n++;

}while (word\_token != NULL);

char\* new\_sentence = malloc ((strlen(txt->line\_pile[i]->str)+MEM\_STEP)\*sizeof(char));

int size\_of\_new\_sentence = strlen(txt->line\_pile[i]->str)\*sizeof(char);

memset(new\_sentence,'\0', sizeof(new\_sentence));

int newsent\_len = 0;

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

if (j == 0){

if (strlen(splitted\_sentences[i][j]) <= 3){

char \*t = realloc(new\_sentence, size\_of\_new\_sentence + strlen("Less Then 3")\*sizeof(char));

if(!t){

puts("Reallocation error. Further result does not represent the intended one.");

}

size\_of\_new\_sentence += strlen("Less Then 3")\*sizeof(char);

new\_sentence = t;

memcpy(new\_sentence, "Less Then 3", strlen("Less Then 3")\*sizeof(char));

newsent\_len = strlen("Less Then 3");

}else{

memcpy(new\_sentence, splitted\_sentences[i][j], strlen(splitted\_sentences[i][j])\*sizeof(char));

newsent\_len = strlen(splitted\_sentences[i][j]);

}

memcpy(new\_sentence+newsent\_len, " ", sizeof(char));

newsent\_len++;

}else{

if (strlen(splitted\_sentences[i][j]) <= 3){

char \*t = realloc(new\_sentence, size\_of\_new\_sentence + strlen("Less Then 3"));

if(!t){

puts("Reallocation error. Further result does not represent the intended one.");

}

size\_of\_new\_sentence += strlen("Less Then 3");

new\_sentence = t;

strcat(new\_sentence, "Less Then 3");

newsent\_len += strlen("Less Then 3");

if (j != n-1){

strcat(new\_sentence, " ");

newsent\_len++;

}else{

strcat(new\_sentence, ".");

newsent\_len++;

}

}else{

strcat(new\_sentence, splitted\_sentences[i][j]);

newsent\_len += strlen(splitted\_sentences[i][j]);

if (j != n-1){

strcat(new\_sentence, " ");

newsent\_len++;

}else{

strcat(new\_sentence, ".");

newsent\_len++;

}

}

}

memcpy(new\_sentence+newsent\_len, "\0", sizeof(char));

txt->line\_pile[i]->str = new\_sentence;

}

}

free(splitted\_sentences);

}

void finding\_a\_substring(struct text\* txt){

char\*\* substrings = malloc (txt->n\*sizeof(char\*));

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

substrings[i] = malloc (strlen(txt->line\_pile[i]->str)\*sizeof(char));

memset(substrings[i], '\0', strlen(txt->line\_pile[i]->str)\*sizeof(char));

int ind\_of\_first = -1;

int ind\_of\_last = -1;

int iteration = 0;

while (ind\_of\_first < 0 && iteration < strlen(txt->line\_pile[i]->str)){

if (isdigit(txt->line\_pile[i]->str[iteration])){

ind\_of\_first = iteration;

}

iteration++;

};

iteration = strlen(txt->line\_pile[i]->str);

while (ind\_of\_last < 0 && iteration > -1){

if (isdigit(txt->line\_pile[i]->str[iteration])){

ind\_of\_last = iteration;

}

iteration--;

};

if (ind\_of\_first != ind\_of\_last){

strncpy(substrings[i], (txt->line\_pile[i]->str+ind\_of\_first), ind\_of\_last-ind\_of\_first+1);

}

}

int zero\_counter = 0;

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

if (strlen(substrings[i]) == 0){

zero\_counter++;

}

}

if (zero\_counter == txt->n){

puts("Text has no substrings!");

}else{

qsort(substrings, txt->n, sizeof(char\*), compar\_for\_substrings);

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

puts(substrings[i]);

}

}

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

free(substrings[i]);

}

free(substrings);

}

void text\_printer(struct text test){

for (int i = 0; i < test.n; i++){

for (int j = 0; j < strlen(test.line\_pile[i]->str); j++){

printf("%c", test.line\_pile[i]->str[j]);

};

printf(" ");

}

printf("\n\n");

}

int main(){

puts("Enter the text:");

struct text test = readtext();

deleting\_spaces(test);

sentence\_checker(&test);

puts("\n");

puts("Text without repeating sentences:");

text\_printer(test);

printf("Enter a number from 1 to 5 depending on what you want to do: \n");

printf("1) Delete all even sentences which have even amount of words \n");

printf("2) Sort all the words in sentences from smallest amount of capital letters to biggest \n");

printf("3) Replace all the words in the text which have 3 or less characters with a substring 'Less Then 3' \n");

printf("4) In every sentence find a substring with maximum length which starts and finishes with a digit and print them from longest to shortest \n");

printf("5) Exit the program \n \n");

char input =' ';

scanf("%c", &input);

while (input != '5'){

printf("\n");

switch (input) {

case '1':

even\_with\_even(&test);

text\_printer(test);

break;

case '2':

for(int i = 0; i < test.n; i++){

sorter\_for\_uppercase(test.line\_pile[i]);

}

text\_printer(test);

break;

case '3':

shorts\_replacer(&test);

text\_printer(test);

break;

case '4':

finding\_a\_substring(&test);

break;

default:

printf("You've either chose something or entered the wrong number. Choose what to do from the list.\n");

break;

};

scanf("%c", &input);

}

puts("The programm is finished!");

for (int i = 0; i <test.n; i++){

free(test.line\_pile[i]);

}

return 0;

}