НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт до лабораторної роботи №1

з курсу

«Мультипарадигменне програмування»

студента 2 курсу
групи ІТ-01
Салимоненка Вадима Олександровича

Викладач:
Очеретяний О. К.

Київ – 2022

Завдання:

Практична робота складається із трьох завдань, які самі по собі є досить простими. Але, оскільки задача - зрозуміти, як писали код наші славні пращури у 1950-х, ми введемо кілька обмежень:

- Заборонено використовувати функції
- Заборонено використовувати цикли
- Для виконання потрібно взяти мову, що підтримує конструкцію GOTO.

Програмна реалізація:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>


#define TASK 2

struct main_object{
        char* word;
        int word_count;
};

int main(){
   char input_str[15000];
        char stop_words_array[7][10] = {"but", "the", "is", "are", "any", "a", "tazasho"};
        int stop_words_number = 7;
        char non_letter_symbols[] = ",. \n-\"\()1234567890][%;/!?";
        struct main_object *main_storage;
        int storage_size = 0;
        char **word_list;
   int word_count = 0;
        int word_count_new = 0;
        int *num1;



        char is_last_letter = 0;
        char is_CurSym_letter = 0;
        int i = 0;
        int check_letter_i = 0;
        int line_max_num = 0;
```

```
        int CurWord_length = 0;
        int CurWord_start_point = 0;
        int words_list_count = 0;
        int local_iterator = 0;
        int local_iterator2 = 0;
        int remove_iterator = 0;
        int local_iterator33 = 0;
        int local_iterator311 = 0;


   char * filename = "./test.txt";
   char read_str[256];
   FILE *fp;


   if((fp= fopen(filename, "r"))==NULL)
   {
      perror("Error occured while opening file");
      return 1;
   }
        local_iterator2 = 0;
loop1_read111:
   if((fgets(read_str, 1024, fp))==NULL)goto loop1_read1112;

loop1_read1:
        if(read_str[local_iterator] == '\n' ){
                line_max_num++;
                input_str[local_iterator2] =  read_str[local_iterator];
                local_iterator = 0;
                local_iterator2++;
                goto loop1_read111;
        }
        if(read_str[local_iterator] != '&'){
                input_str[local_iterator2] =  read_str[local_iterator];
        }else{
                input_str[local_iterator2] =  read_str[local_iterator];
                goto loop1_read1112;
                }
        local_iterator2++;
        local_iterator++;
        goto loop1_read1;
loop1_read2:

        goto loop1_read111;
loop1_read1112:
```

```
   fclose(fp);
        local_iterator2 = 0;
        local_iterator = 0;




smal_letters:
        if(i  == strlen(input_str)){i = 0;        goto temp;}
        if(input_str[i] >= 65 && input_str[i] <= 90) input_str[i] += 32;
        i++;
        goto smal_letters;
temp:

loop1:   // считаем количество слов
        goto check_not_letter;
loop1_1:
        if((is_last_letter == 1 && is_CurSym_letter == 0) || (is_CurSym_letter == 1 && i ==
strlen(input_str) -1)){
                word_count++;
                is_last_letter = 0;
                i++;
                goto loop1;
        }else{is_last_letter = is_CurSym_letter;}
        i++;
        if(i < (strlen(input_str) ) ) goto loop1;

        goto creating_words_list;

check_not_letter:  // проверяем является ли символ не буквой
        is_CurSym_letter = 1;
        check_letter_i = 0;
local_check_loop:
        if(input_str[i] == non_letter_symbols[check_letter_i]){
                is_CurSym_letter = 0;
                goto loop1_1;
        }
        check_letter_i++;
        if(check_letter_i < strlen(non_letter_symbols)) goto local_check_loop;
        goto loop1_1;



//-------------------
```

```
creating_words_list:
	word_count -= 1;
	word_list = (char**) malloc(sizeof(char*) * (word_count + 1));
	is_last_letter = 1;
	is_CurSym_letter = 0;
	i = 0;
//--------------------------


loop2:
	goto check_not_letter2;
loop2_1:
	if(is_CurSym_letter == 1){ CurWord_length++;}
	if(input_str[i] == '&'){
		i++;
		is_last_letter = 0;
		goto loop2;
	}
	if((is_last_letter == 1 && is_CurSym_letter == 0) || (is_CurSym_letter == 1 && i ==
strlen(input_str) -1) ){
		word_list[words_list_count] = (char*) malloc(sizeof(char) * (CurWord_length+1)
);
		CurWord_start_point = i - CurWord_length;
		if((is_CurSym_letter == 1 && i == strlen(input_str) -1))CurWord_start_point++;
		goto add_str;
continue_adding_words:

		is_last_letter = 0;
		words_list_count++;
		CurWord_length = 0;
		i++;
		is_last_letter;
		if(i < strlen(input_str) )goto loop2;
	}else{is_last_letter = is_CurSym_letter;}
	i++;
	if(i < (strlen(input_str) ) ) goto loop2;
	remove_iterator = 0;
	check_letter_i = 0;
	goto remove_stopwords;



check_not_letter2:
	is_CurSym_letter = 1;
	check_letter_i = 0;
local_check_loop2:
```

```
        if(input_str[i] == non_letter_symbols[check_letter_i]){
                is_CurSym_letter = 0;
                goto loop2_1;
        }
        check_letter_i++;
        if(check_letter_i < strlen(non_letter_symbols)) goto local_check_loop2;
        goto loop2_1;

add_str:

        word_list[words_list_count][local_iterator] = input_str[CurWord_start_point];

        CurWord_length--;
        CurWord_start_point++;
        local_iterator++;
        if(CurWord_length > 0) goto add_str;
        word_list[words_list_count][local_iterator] = '\0';
        local_iterator = 0;
        goto continue_adding_words;


remove_stopwords:

 remove_loop1:
        if(local_iterator >= word_count) {goto creation_map;}
        if(local_iterator2 >= stop_words_number){local_iterator2 = 0; local_iterator++; goto
remove_loop1;}
        if(strcmp(word_list[local_iterator], stop_words_array[local_iterator2] ) == 0){
                local_iterator2 = 0;
                local_iterator++;
                remove_iterator = local_iterator;
                goto remove_loop2;
        }
        local_iterator2++;
        goto remove_loop1;
remove_loop2:
        check_letter_i = remove_iterator-1;
        remove_iterator--;
        free(word_list[remove_iterator]);
recovery1:
        word_list[check_letter_i] = word_list[check_letter_i+1];
        check_letter_i++;
if(check_letter_i < word_count) goto recovery1;
        word_count--;
        local_iterator--;
goto remove_stopwords;
```

```
print_start:

int max_words;
int local_iterator31;
int general_print_count = 0;
int print_iterator = 0;
num1 = malloc( (storage_size+1) * sizeof(int));
print_start2:
	max_words = 0;
	local_iterator31 = 0;
	print_iterator = 0;
print_main_word:
	if(main_storage[print_iterator].word_count >= max_words){
		max_words = main_storage[print_iterator].word_count;
		local_iterator31 = print_iterator;
	}
	print_iterator++;
if(print_iterator < storage_size) goto print_main_word;
	if(TASK == 1)printf("%d - %d - %s;\n", general_print_count+1,
main_storage[local_iterator31].word_count,  main_storage[local_iterator31].word);
	if(main_storage[local_iterator31].word_count < 30) word_count_new++;
	num1[local_iterator31] = main_storage[local_iterator31].word_count;

	main_storage[local_iterator31].word_count = 0;
	general_print_count++;
if(general_print_count < storage_size && general_print_count < storage_size) goto print_start2;

	if(TASK == 2)goto dropping_words1;
	int clear_memory2 = 0;
clear22:
	free(word_list[clear_memory2]);
	clear_memory2++;
if(clear_memory2 < word_count) goto clear22;
	free(word_list);
	free(main_storage);
	return 0;




creation_map:

	main_storage = malloc( (word_count+1) * sizeof(struct main_object));
	int words_iterator = 0;
```

```
	char* storage_word;
	char flag_is_in_storage;
	int is_in_storage_number;

count111:
	storage_word = word_list[words_iterator];
	flag_is_in_storage = 0;
	is_in_storage_number = 0;
is_word_present:
	if(is_in_storage_number >= storage_size) goto count122;
	if(strcmp(main_storage[is_in_storage_number].word, storage_word) == 0) {
		main_storage[is_in_storage_number].word_count += 1;
		flag_is_in_storage = 1;
	}
	is_in_storage_number++;
if(is_in_storage_number < storage_size) goto is_word_present;
count122:
	if(flag_is_in_storage == 0){
			main_storage[storage_size] = (struct main_object){.word=storage_word,
.word_count=1};
			storage_size++;
	}
	words_iterator++;
	if(words_iterator < word_count) goto count111;
	goto print_start;




dropping_words1:

local_iterator = 0;
local_iterator33 = 0;
char word_list_new[word_count_new][100];


dropping_words:
	if(local_iterator >= word_count_new)goto dropping_words_end;
	if(num1[local_iterator] < 30){
		goto word_write1;
word_write2:
		local_iterator33++;
	}
	local_iterator++;
```

```
        goto dropping_words;


word_write1:
        local_iterator2 = 0;
word_write1_1:

        if(main_storage[local_iterator].word[local_iterator2] == '\0'){
                word_list_new[local_iterator33][local_iterator2] = '\0'; goto word_write2;
        }
        word_list_new[local_iterator33][local_iterator2] =
main_storage[local_iterator].word[local_iterator2];
        local_iterator2++;
        goto word_write1_1;



dropping_words_end:
   i = 0;
        local_iterator = 0;
remove_01:
        if(input_str[i] == '&')goto remove_02;
remove_011:
        if(local_iterator >= strlen(non_letter_symbols)) goto remove_012;
                if(input_str[i] == non_letter_symbols[local_iterator] && input_str[i] != '\n')
input_str[i] = ' ';
        local_iterator++;
        goto remove_011;
remove_012:
        local_iterator = 0;
        i++;
        goto remove_01;
remove_02:

 i = 0;
 char str[100];
 int i11 = 1;
 int j = 0;

word_count_new11:
        if(i11 >= word_count_new) goto word_count_new12;
word_count_new111:
        if(j >= word_count_new - i11) goto word_count_new112;
       if(strcmp(word_list_new[j], word_list_new[j+1]) > 0){
          strcpy(str, word_list_new[j]);
          strcpy(word_list_new[j], word_list_new[j+1]);
```

```
            strcpy(word_list_new[j+1], str);
        }
            j++;
            goto word_count_new111;
word_count_new112:
            j = 0;
            i11++;
            goto word_count_new11;
word_count_new12:
            i11 = 2;
            int line_num = 0;
            int page_num = 0;
            int last_page_num = 999;
            int flag11 = 0;
            char *last_word;
            i = 0;
loop_print_1:
    if(i11 >= word_count_new-2)goto loop_print_2;
        printf(" %s  -- ", word_list_new[i11]);
loop_page1:
                    if(strcmp(last_word, word_list_new[i11]) != 0) last_page_num = 999;
                    page_num = line_num/45 + 1;
                    if(line_num >= line_max_num){line_num = 0; i = 0;page_num = 0; goto
loop_page2;}
                    if(input_str[i] == '\n'){line_num++;}
                    if(input_str[i] == word_list_new[i11][0])goto check_word_in_page1;
check_word_in_page2:
                    if(flag11 == 1 && last_page_num != page_num){
                            last_page_num = page_num;
                            printf("%d, ",page_num);
                            last_word = word_list_new[i11];
                    }
                    i++;
                    goto loop_page1;
loop_page2:
            printf("\n", word_list_new[i11]);
            i11++;
            goto loop_print_1;

check_word_in_page1:
                    flag11 = 1;
                    local_iterator = i;
                    local_iterator2 = 0;
check_word_in_page1_1:
            if(word_list_new[i11][local_iterator2] == '\0' ){goto check_word_in_page2;}
            if(word_list_new[i11][local_iterator2] != input_str[local_iterator]){
```

```
                flag11 = 0;goto check_word_in_page2;

                }
        local_iterator2++;
        local_iterator++;
        goto check_word_in_page1_1;

loop_print_2:
        int clear_memory = 0;
clear2:
        free(word_list[clear_memory]);
        clear_memory++;
if(clear_memory < word_count) goto clear2;
        free(word_list);
        free(main_storage);
        return 0;

}
```

Наведена вище реалізація виконує відразу два завдання, що можна вибрати змінивши define TASK. Спочатку програма дістає інформацію з файла, що містить початковий текст. У ньому всі великі літери заміняються на малі та рахується кількість слів для виділення пам’яті під допоміжні структури. На цьому етапі викидаються стоп слова, що не потрібні у фінальному результаті.

Далі для різних завдань різний алгоритм. У першому ми створюємо структуру, що містить слово та поле для кількості повторювань у тексті. Після заповнення структури вона виводеться користовучу. У другому випадку створюється структура з словами та сортується за абеткою. Потім для кожного слова виводяться сторінки на яких воно зустрічається.

Висновок:

У ході даної роботи я розглянув імперативний підхід до програмування, що використовувся на початку зародження програмування. Він значно поступається сучасним підходам як у функціоналі так і у зручності та може бути використаний лише в програмуванні незначних завдань в умовах обмеженої пам’яті носія.