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
Дурдинця Олександра Тиберійовича

Викладач:
ас. Очеретяний О.К.

Київ – 2022

Завдання 1:

Обчислювальна задача тут тривіальна: для текстового файлу ми хочемо відобразити N (наприклад, 25) найчастіших слів і відповідну частоту їх повторення, упорядковано за зменшенням. Слід обов'язково нормалізувати використання великих літер і ігнорувати стоп-слова, як «the», «for» тощо. Щоб все було просто, ми не піклуємося про порядок слів з однаковою частотою повторень. Ця обчислювальна задача відома як **term frequency**.

Ось такий вигляд матимуть ввід і відповідно вивід результату програми:

Input:

```
White tigers live mostly in India

Wild lions live mostly in Africa
```

Output:

```
live - 2

mostly - 2

africa - 1

india - 1

lions - 1

tigers - 1

white - 1

wild - 1
```

Програмна реалізація:

main.c task1.c

GoToHell

```
1   #pragma once
2   #pragma warning(disable : 4996)
3
4   #include <stdio.h>
5   #include <stdlib.h>
6
7   #define WORD_SIZE 25
8   #define WORDS_TO_IGNORE_COUNT 3
9
10  inline void wrapper()
11  {
12      char c;
13      char filename[FILENAME_MAX];
14      FILE* file;
15
16      struct Word
17      {
18          char* word;
19          int count;
20      } w;
21
22      struct WordList;
23      struct WordList {
24          struct Word data;
25          struct WordList* next;
26      };
27
28      struct WordList* words = 0;
29      struct WordList* current_word = 0;
30      struct WordList** current_word_ptr = 0;
31
32      char* words_to_ignore[WORDS_TO_IGNORE_COUNT] = {
33          "the",
34          "a",
35          "an"
36      };
37
38  task_1:
39      //открыть файл
40      printf("Enter filename\n ");
41      gets(filename);
42      file = fopen(filename, "r");
43      if (file == NULL)
44      {
45          printf("Invalid filename\n ");
46          goto task_1;
47      }
48
```

```
48
49      get_word:
50          w.count = 0;
51          w.word = malloc(WORD_SIZE * sizeof(char));
52
53          int char_position = 0;
54
55      get_char:
56          //вылетаем, если слова какие-то длинные
57          if (char_position == WORD_SIZE)
58          {
59              printf("Words in text is longer than I expected 0_0\n");
60              goto end;
61          }
62          //выходим если файл закончился
63          if ((c = fgetc(file)) == EOF)
64          {
65              w.word[char_position] = '\0';
66              goto process_word;
67          }
68          //переход на следующее слово
69          if (c == ' ' )
70          {
71              if (char_position == 0)
72                  goto get_char;
73              w.word[char_position] = '\0';
74              goto process_word;
75          }
76          //игнорируем символы, которые не буквы
77          else if (!(c >= 'A' && c <= 'Z') && !(c >= 'a' && c <= 'z'))
78          {
79              goto get_char;
80          }
81          //обрабатываем буквы
82          else
83          {
84              //to lower
85              if (c >= 'A' && c <= 'Z')
86              {
87                  c -= 'A' - 'a';
88              }
89              //add char to word
90              w.word[char_position] = c;
91              ++char_position;
92              goto get_char;
93          }
```

```
process_word:
    current_word_ptr = &words;
    char index = 0;
    char index2 = 0;

compare_with_stop_words:
    if (index == WORDS_TO_IGNORE_COUNT)
        goto compare_with_word;
    else if (w.word[index2] != words_to_ignore[index][index2])
    {
        index++;
        index2 = 0;
        goto compare_with_stop_words;
    }
    else if (w.word[index2] == '\0')
        goto end_processing_word;
    else
    {
        index2++;
        goto compare_with_stop_words;
    }

compare_with_word:
    if (*current_word_ptr == 0)
    {
        *current_word_ptr = malloc(sizeof(struct WordList));
        (**current_word_ptr).data.word = w.word;
        (**current_word_ptr).data.count = 1;
        (**current_word_ptr).next = 0;
    }
    else
    {
        index = 0;

    compare_chars:
        if (w.word[index] != (**current_word_ptr).data.word[index])
        {
            current_word_ptr = &((**current_word_ptr).next);
            goto compare_with_word;
        }
        else if(w.word[index] == '\0')
        {
            ++(**current_word_ptr).data.count;
            goto get_word;
        }
        else
        {
            index++;
            goto compare_chars;
        }
    }
```

```
end_processing_word:
    if(!feof(file))
        goto get_word;

sort_words:

    //selection sort))))))))))
    //O = n^2 in any case
    //so cool

    if (words == 0 || (*words).next == 0)
        goto out;

    current_word = words;
    struct WordList* another_word = (*words).next;
    struct Word buff;

sort_iteration:
    if (current_word == 0)
        goto out;
    struct WordList* best_word = current_word;
    another_word = current_word;

select_max:
    another_word = (*another_word).next;
    if (another_word == 0)
        goto swap;
    if ((*another_word).data.count > (*best_word).data.count)
        best_word = another_word;
    goto select_max;
swap:
    buff = (*best_word).data;
    (*best_word).data = (*current_word).data;
    (*current_word).data = buff;

    current_word = (*current_word).next;
    goto sort_iteration;

out:
    current_word = words;

out_next:
    if (current_word != 0)
    {
        printf("%s - %i\n", (*current_word).data.word, (*current_word).data.count);
        current_word = (*current_word).next;
        goto out_next;
    }
```

```
197     end:
198         current_word = words;
199         if (current_word != 0)
200         {
201             current_word = (*current_word).next;
202             free(words);
203             words = current_word;
204             goto end;
205         }
206         return;
207     }
```

Результат:

```
Enter filename
 test1.txt
live - 2
mostly - 2
in - 2
tigers - 1
white - 1
india - 1
wild - 1
lions - 1
africa - 1
```

Опис алгоритму:

1. Оголошення змінних, структур даних, списку слів для ігнорування
2. Введення назви файла, відкриття файла
3. Початок введення слова. Виділення пам'яті
4. Зчитування символу з файла
    4.1 Якщо кінець файлу, перейти до 5
    4.2 Якщо символ ' ', перейти до 5.
    4.3 Якщо символ - не буква, перейти до 4
    4.5 Якщо символ - uppercase letter, замінити на lower case. Додати символ до слова. Перейти до 4
5. Перевірка на співпадіння з списком стоп-слів.
    5.1 Оголошення двох ітераторів(наприклад i, j)
    5.2 Якщо i більше ніж розмір масиву стоп-слів, перейти до 6
    5.3 Якщо символи з індексом j в слові та стоп-слові різні, i++, перейти до 5.1
    5.4 інакше, якщо символи є кінцем рядку, перейти до 7
    5.5 j++, перейти до 5.1
6. Аналогічно перевіряємо на співпадіння з словами, які вже є в списку слів
    6.1 Якщо співпадінь немає, створити нове слово. Інакше, збільшити кількість входжень існуючого слова. Перейти до 7
7. Якщо кінець файлу, перейти до 8. Інакше, перейти до 4
8. Сортування слів. Оголошення i, j, max
    8.1 j = i + 1. Пошук слова з максимальною кількістю входжень, починаючи з j.
    8.2 swap i with max.
    8.3 якщо i - останній елемент, перейти до 9. Інакше перейти до 8.1

9. Вивести слова
10. Очистити пам'ять

Завдання 2:

Тепер, нам потрібно виконати задачу, що називається словниковим індексуванням. Для текстового файлу виведіть усі слова в алфавітному порядку разом із номерами сторінок, на яких Ці слова знаходяться. Ігноруйте всі слова, які зустрічаються більше 100 разів. Припустимо, що сторінка являє собою послідовність із 45 рядків. Наприклад, якщо взяти книгу Pride and Prejudice, перші кілька записів індексу будуть:

```
abatement - 89

abhorrence - 101, 145, 152, 241, 274, 281

abhorrent - 253

abide - 158, 292
```

Програмна реалізація:

```
#pragma once
#pragma warning(disable : 4996)

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define PAGE_SIZE 45
#define WORD_SIZE 12
#define WORDS_TO_IGNORE_COUNT 3

inline void wrapper()
{
    char c;
    char filename[FILENAME_MAX];
    FILE* file;

    struct PageList
    {
        int page;
        struct PageList* next;
    };

    struct Word
    {
        char* word;
        long long int heuristic;
        int count;
        struct PageList* pages;
    } w;

    //struct WordList;
    struct WordList {
        struct Word data;
        struct WordList* next;
    };
```

```c
    //struct WordList;
    struct WordList {
        struct Word data;
        struct WordList* next;
    };

    struct WordList* words = 0;
    struct WordList* current_word = 0;
    struct WordList** current_word_ptr = 0;

    int current_page = 0;
    int current_row = 0;

task_1:
    //открыть файл
    printf("Enter filename\n ");
    gets(filename);
    file = fopen(filename, "r");
    if (file == NULL)
    {
        printf("Invalid filename\n ");
        goto task_1;
    }

get_word:
    w.count = 0;
    w.heuristic = 0;
    w.word = malloc(WORD_SIZE * sizeof(char));

    int char_position = 0;

get_char:
    //игнорируем часть слова, которая не влазит
    if (char_position == WORD_SIZE - 1)
    {
        w.word[char_position] = '\0';
        w.heuristic *= pow('z' - 'a', WORD_SIZE - char_position);
    getc: if ((c = fgetc(file)) != ' ' && c != '\n' && c != EOF) goto getc;
```

```
get_char:
    //игнорируем часть слова, которая не влазит
    if (char_position == WORD_SIZE - 1)
    {
        w.word[char_position] = '\0';
        w.heuristic *= pow('z' - 'a', WORD_SIZE - char_position);
    getc: if ((c = fgetc(file)) != ' ' && c != '\n' && c != EOF) goto getc;
        goto process_word;
    }
    //выходим если файл закончился
    if ((c = fgetc(file)) == EOF)
    {
        w.word[char_position] = '\0';
        w.heuristic *= pow('z' - 'a', WORD_SIZE - char_position);
        goto process_word;
    }
    //переход на следующее слово
    if (c == ' ')
    {
        if (char_position == 0)
            goto get_char;
        w.word[char_position] = '\0';
        w.heuristic *= pow('z' - 'a', WORD_SIZE - char_position);
        goto process_word;
    }
    if (c == '\n')
    {
        w.word[char_position] = '\0';
        w.heuristic *= pow('z' - 'a', WORD_SIZE - char_position);
        current_row++;
        if (current_row == PAGE_SIZE)
        {
            current_row = 0;
            current_page++;
        }
        goto process_word;
    }
    //игнорируем символы, которые не буквы
    else if (!(c >= 'A' && c <= 'Z') && !(c >= 'a' && c <= 'z'))
    {
        goto get_char;
    }
```

```
105         //обрабатываем буквы
106         else
107         {
108             //to lower
109             if (c >= 'A' && c <= 'Z')
110             {
111                 c -= 'A' - 'a';
112             }
113             //add char to word
114             w.word[char_position] = c;
115             w.heuristic *= 'z' - 'a';
116             w.heuristic += c - 'a' + 1;
117             ++char_position;
118             goto get_char;
119         }
120 
121 
122     process_word:
123         current_word_ptr = &words;
124         char index = 0;
125 
126         if (w.word[0] == 0)
127             goto end_processing_word;
128 
129     compare_with_word:
130         if (*current_word_ptr == 0)
131         {
132             *current_word_ptr = malloc(sizeof(struct WordList));
133             (**current_word_ptr).data.word = w.word;
134             (**current_word_ptr).data.count = 1;
135             (**current_word_ptr).data.heuristic = w.heuristic;
136             (**current_word_ptr).data.pages = malloc(sizeof(struct PageList));
137             (*(**current_word_ptr).data.pages).page = current_page;
138             (*(**current_word_ptr).data.pages).next = 0;
139             (**current_word_ptr).next = 0;
140         }
141         else
142         {
143             index = 0;
144 
145         compare_chars:
146             if (w.word[index] != (**current_word_ptr).data.word[index])
147             {
148                 current_word_ptr = &((**current_word_ptr).next);
149                 goto compare_with_word;
150             }
151             else if (w.word[index] == '\0')
152             {
153                 ++(**current_word_ptr).data.count;
154                 struct PageList** it = &((**current_word_ptr).data.pages);
155                 add_page:
```

```
155             add_page:
156                 if (*it == 0)
157                 {
158                     (*it) = malloc(sizeof(struct PageList));
159                     (**it).page = current_page;
160                     (**it).next = 0;
161                 }
162                 else
163                 {
164                     if ((**it).page == current_page)
165                         goto get_word;
166                     it = &((**it).next);
167                     goto add_page;
168                 }
169                 goto get_word;
170             }
171             else
172             {
173                 index++;
174                 goto compare_chars;
175             }
176         }
177
178     end_processing_word:
179         if (!feof(file))
180             goto get_word;
181
182     sort_words:
183
184         //selection sort))))))))))))
185         //O = n^2 in any case
186         //so cool
187
188         if (words == 0 || (*words).next == 0)
189             goto out;
190
191         current_word = words;
192         struct WordList* another_word = (*words).next;
193         struct Word buff;
194
195     sort_iteration:
196         if (current_word == 0)
197             goto out;
198         struct WordList* best_word = current_word;
199         another_word = current_word;
200
```

```
select_max:
    another_word = (*another_word).next;
    if (another_word == 0)
        goto swap;
    if ((*another_word).data.heuristic < (*best_word).data.heuristic)
        best_word = another_word;
    goto select_max;
swap:
    buff = (*best_word).data;
    (*best_word).data = (*current_word).data;
    (*current_word).data = buff;

    current_word = (*current_word).next;
    goto sort_iteration;

out:
    current_word = words;

out_next:
    if (current_word != 0)
    {
        if ((*current_word).data.count < 100)
        {
            printf("%s - ", (*current_word).data.word);
            struct PageList* it = (*current_word).data.pages;
        print_page:
            if (it == 0)
                printf("\n");
            else
            {
                printf("%i ", (*it).page);
                it = (*it).next;
                goto print_page;
            }
        }
        current_word = (*current_word).next;
        goto out_next;
    }

end:
    current_word = words;
    if (current_word != 0)
    {
        current_word = (*current_word).next;
        free(words);
        words = current_word;
        goto end;
    }
    return;
}
```

Опис алгоритму:

1. Оголошення змінних, структур даних
2. Введення назви файла, відкриття файла
3. Початок введення слова. Виділення пам'яті
4. Зчитування символу з файла
    4.1 Якщо кінець файлу, перейти до 5
    4.2 Якщо символ ' ', перейти до 5.
    4.3 Якщо кінець рядка, збільшити лічильник рядків. Якщо лічильник дорівнює кількості рядків на сторінці, обнулити лічильник та збільшити лічильник сторінок
    4.4 Якщо символ - не буква, перейти до 4
    4.5 Якщо символ - uppercase letter, замінити на lower case. Додати символ до слова. Перейти до 4
5. Перевірка на співпадіння з існуючими словами
    5.1 Оголошення двох ітераторів(наприклад i, j)
    5.2 Якщо список слів закінчився, створити нове слово та додати його до списку.
    5.3 Якщо символи з індексом j в словах різні, i++, перейти до 5.1
    5.4 інакше, якщо символи є кінцем рядку, збільшити лічильник слова з списку та додати поточну сторінку до списку сторінок, де зустрічається слово, перейти до 6.
    5.5 j++, перейти до 5.1
6. Якщо кінець файлу, перейти до 7. Інакше, перейти до 4
7. Сортування слів. Оголошення i, j, max
    7.1 j = i + 1. Пошук слова, що знаходиться першим в алфавітному порядку, починаючи з j.
    7.2 поміняти місцями i та знайдене слово.
    7.3 якщо i - останній елемент, перейти до 8. Інакше перейти до 7.1

8. Вивести слова
9. Очистити пам'ять

Висновки:

В межах даної роботи необхідно було вивчити імперативне програмування та виконати два завдання. Для виконання необхідно було дотримуватися імперативної парадигми програмування та не використовувати функції, цикли. Для виконання я обрав мову C.