**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Мультипарадигменне програмування»

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01 Пашковський Євгеній Сергійович*

**Перевірив**

(прізвище, ім'я, по батькові)

*ас. Очеретяний О.К.*

Київ 2022

1. Завдання лабораторної роботи

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

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

1. Опис використаних технологій

Під час виконання лабораторної роботи було задіяно мову С, що підтримує конструкцію goto. Було використано редактор коду Visual Studio Code.

1. Вихідний код

task1.c:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct

{

  char \*word;

  int n;

} word\_count;

int main(int argc, char \*argv[])

{

  if (argc != 2)

  {

    printf("You need to pass the name of a file to this program:\n./task1.exe [FILENAME]");

    return (1);

  }

  int N = 25;

  char buf[255];

  word\_count \*\*map;

  map = calloc(sizeof(word\_count \*), N);

  FILE \*fp;

  fp = fopen(argv[1], "r");

  if (fp == NULL)

  {

    printf("Error while reading a file... Maybe typo in the name of file?");

    return (1);

  }

  char chunk;

  short i = 0;

  int n = 0;

  short done = 0;

  char \*word;

  short wordOK = 1;

  char \*bannedWords[] = {" ", "in", "the", "for", "and", "or", "of"};

  int bannedWordsCount = sizeof(bannedWords) / sizeof(char \*);

read\_word:

  chunk = fgetc(fp);

  if (!(chunk >= 97 && chunk <= 122) && !(chunk >= 65 && chunk <= 90) && chunk != '\'')

  {

    if (chunk == EOF)

      done = 1;

    goto read\_word\_end;

  }

  buf[i] = chunk;

  i++;

  goto read\_word;

read\_word\_end:

  word = calloc(i, sizeof(char));

  int j = 0;

word\_load:

  if ((buf[j] >= 97 && buf[j] <= 122) || (buf[j] >= 65 && buf[j] <= 90) || buf[j] == '\'')

  {

    word[j] = (buf[j] >= 65 && buf[j] <= 90) ? buf[j] + 32 : buf[j];

  }

  else

  {

    i--;

    if (i < 1) {

      goto next\_loop;

    }

    word = realloc(word, sizeof(char) \* i);

  }

  buf[j] = EOF;

  j++;

  if (j < i)

    goto word\_load;

  word = realloc(word, sizeof(char) \* (i + 1));

  word[i] = '\0';

  j = 0;

  i = 0;

  int g = 0;

checkExisting:

  if (g < n && strcmp(word, map[g]->word) == 0)

  {

    map[g]->n++;

    goto next\_loop;

  }

  g++;

  if (g < n)

  {

    goto checkExisting;

  }

  wordOK = 1;

  int k = 0;

validate\_word:

  if (strcmp(word, bannedWords[k]) == 0)

    wordOK = 0;

  k++;

  if (k < bannedWordsCount && wordOK == 1)

  {

    goto validate\_word;

  }

  if (wordOK == 1)

  {

    word\_count \*w;

    w = malloc(sizeof(word\_count));

    w->word = word;

    w->n = 1;

    map[n++] = w;

    if (n >= N)

    {

      map = realloc(map, sizeof(word\_count \*) \* (n + 1));

    }

  }

  if (n < 2 || wordOK == 0)

    goto next\_loop;

  int h = 0;

outer\_sort:

  h = 0;

  short sorted = 1;

inner\_sort:

  if (map[h]->n < map[h + 1]->n)

  {

    word\_count \*buf;

    buf = map[h];

    map[h] = map[h + 1];

    map[h + 1] = buf;

    sorted = 0;

  }

  h++;

  if (h < n - 1)

    goto inner\_sort;

  if (sorted == 0)

    goto outer\_sort;

next\_loop:

  if (done == 0)

    goto read\_word;

  h = 0;

read\_map:

  printf("%s - %d\n", map[h]->word, map[h]->n);

  h++;

  if (h < (n < N ? n : N))

  {

    goto read\_map;

  }

  fclose(fp);

  return 0;

}

task2.c:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct

{

  char \*word;

  int count;

  int \*pages;

} dict\_item;

short PAGE\_LINES = 45;

short IGNORE\_COUNT = 100;

int main(int argc, char \*argv[])

{

  if (argc != 2)

  {

    printf("You need to pass the name of a file to this program:\n./task2.exe [FILENAME]");

    return (1);

  }

  FILE \*fp;

  fp = fopen(argv[1], "r");

  if (fp == NULL)

  {

    printf("Error while reading a file... Maybe typo in the name of file?");

    return (1);

  }

  dict\_item \*\*index;

  long items\_count = 0;

  index = malloc(0);

  char chunk;

  char \*page;

  page = calloc(sizeof(char), 1);

  page[0] = ' ';

  long page\_count = 0;

  short done = 0;

  int i = 0;

  int n = 0;

read\_page:

  chunk = fgetc(fp);

  if (chunk == EOF)

  {

    done = 1;

    goto read\_page\_end;

  }

  if (chunk == '\n')

    n++;

  page = realloc(page, sizeof(char) \* (i + 1));

  page[i++] = chunk;

  if (n != PAGE\_LINES)

    goto read\_page;

read\_page\_end:

  page = realloc(page, sizeof(char) \* (i + 1));

  page[i++] = '\0';

  page\_count++;

  int chars\_on\_page = i + 1;

  i = 0;

  short j = 0;

  char \*word;

  word = malloc(0);

read\_words:

  if (!(page[i] >= 97 && page[i] <= 122) && !(page[i] >= 65 && page[i] <= 90) && page[i] != '\'')

  {

    word = realloc(word, sizeof(char) \* (j + 1));

    word[j++] = '\0';

    if (word[0] != '\0')

    {

      int \*pages\_list;

      dict\_item \*item;

      n = 0;

      short found = 0;

    find\_in\_index:

      if (strcmp(index[n]->word, word) == 0)

      {

        if (page\_count > index[n]->pages[index[n]->count - 1])

        {

          index[n]->pages = realloc(index[n]->pages, sizeof(int) \* (index[n]->count + 1));

          index[n]->pages[index[n]->count] = page\_count;

          index[n]->count++;

        }

        found = 1;

        goto find\_in\_index\_end;

      }

      n++;

      if (n < items\_count)

        goto find\_in\_index;

    find\_in\_index\_end:

      if (found == 0)

      {

        pages\_list = calloc(sizeof(int), 1);

        pages\_list[0] = page\_count;

        item = malloc(sizeof(dict\_item));

        item->word = word;

        item->pages = pages\_list;

        item->count = 1;

        index = realloc(index, sizeof(dict\_item \*) \* (items\_count + 1));

        index[items\_count++] = item;

        short sorted;

        if (items\_count > 1)

        {

        sort:

          sorted = 1;

          int k = items\_count - 2;

        bubble:

        {

          if (strcmp(index[k]->word, index[k + 1]->word) > 0)

          {

            dict\_item \*buf;

            buf = index[k];

            index[k] = index[k + 1];

            index[k + 1] = buf;

            sorted = 0;

          }

        }

          k--;

          if (k >= 0)

            goto bubble;

          if (sorted == 0)

            goto sort;

        }

      }

    }

    j = 0;

    word = malloc(0);

    i++;

    goto read\_words;

  }

  if ((page[i] >= 97 && page[i] <= 122) || (page[i] >= 65 && page[i] <= 90) || (page[i] == '\'' && j != 0))

  {

    word = realloc(word, sizeof(char) \* (j + 1));

    word[j++] = (page[i] >= 65 && page[i] <= 90) ? page[i] + 32 : page[i];

  }

  i++;

  if (i < chars\_on\_page)

    goto read\_words;

next\_loop:

  i = 0;

  n = 0;

  j = 0;

  if (done == 0)

  {

    goto read\_page;

  }

read\_word\_from\_index:

  if (index[i]->count <= IGNORE\_COUNT)

  {

    printf("%s - ", index[i]->word);

  read\_pages:

    printf("%d", index[i]->pages[j]);

    if (j != index[i]->count - 1)

      printf(", ");

    j++;

    if (j < index[i]->count)

      goto read\_pages;

    printf("\n");

    j = 0;

  }

  i++;

  if (i < items\_count)

    goto read\_word\_from\_index;

  return 0;

}

1. Опис алгоритмів

task1:

1. Визначаємо перелік стоп-слів і максимальну кількість N слів для виводу;
2. Читаємо наступне слово з файлу посимвольно (розділювачем є пробіл) в буфер;
3. Переносимо слово з буферу в окрему змінну, очищаємо буфер;
4. Виконуємо послідовно дії:
   1. Якщо слово є в словнику – інкрементуємо частоту його зустрічі n, переходимо до п.5;
   2. Якщо слово знаходиться у переліку стоп-слів – переходимо до п.5;
   3. Добавляємо слово в кінець словника;
   4. Якщо в словнику менше 2 слів – переходимо до п.5;
   5. Сортуємо словник з кінця за частотою n (з більшого до меншого);
5. Якщо в файлі ще залишились слова – переходимо до п.2;
6. Виводимо перші N елементів словника.

task2:

1. Визначаємо кількість рядків на одній сторінці та кількість входжень, яка є максимальною для відображення у виводі програми;
2. Читаємо наступну сторінку;
3. Читаємо наступне слово з сторінки посимвольно (розділювачем є пробіл);
4. Якщо слово є в індексі – добавляємо номер поточної сторінки (якщо цей номер ще не присутній) до запису з цим словом і переходимо до п. 7;
5. Якщо слова немає в індексі – створюємо запис з цим словом і номером поточної сторінки в кінці індексу;
6. Якщо в індексі принаймні 2 елементи – сортуємо його з кінця за алфавітним порядком;
7. Якщо не останнє слово на сторінці – переходимо до п. 3;
8. Якщо не остання сторінка – переходимо до п.2;
9. Виводимо значення записів індексу, кількість входжень яких не перевищує визначену кількість сторінок.
10. Скріншоти виконання коду



