Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Кафедра компьютерной безопасности

Отчёт

к лабораторной работе №3

по дисциплине

«Языки программирования»

Работу выполнила

студентка группы СКБ221

В. А. Кинаш

Москва 2022

Постановка задачи:

Вводится последовательность предложений, оканчивающихся точкой. Предложения могут занимать более одной строки. Вывести все предложения в порядке возрастания длин ровно по одному на строку, заменяя перенос строки в исходном вводе пробельным символом. Код программы должен поддерживать ввод с консоли флагов --tofile и --fromfile. Первый говорит нам, что вывод будет задан в заданный пользователем в командной строке файл (в случае отсутствия файла создает его), а второй говорит о считывании текста из файла, заданного пользователем.

Необходимо предусмотреть возможность исполнения обоих флагов. Учесть возможность некорректного ввода и обработать исключения.

Создать Makefile с возможностью сборки, clean и distclean.

Алгоритм выполнения задачи

1) В функции main (int argc, char** argv) в начале происходит обработка флагов, в которой мы определям, откуда производится ввод, и куда производить вывод (основываясь на возможном наличие флагов –fromfile, --tofile). Если в main.cpp передано больше аргументов, или после флага не указано название файла, то выдается ошибка.

2) Если указан флаг –fromfile, то с помощью fgets мы считываем предложения из файла, название которого хранится в переменной from_file, сохраняя их в массив char *s, пока не встретим EOF, означающий конец файла.

3) Если в main не был передан флаг –fromfile, то есть file_name = “-1”, то с помощью метода getline построчно считываются предложения из терминала, пока не будет встречен знак «@», означающий конец ввода.

4) Далее с помощью функции cords_of_points я сохранила индексы всех точек, разделяющих предложения в массив points и вернула их количество в переменную points_cnt.

5) С помощью функции lens_of_sentens я записала длины всех предложений в изначальном порядке в массив lens.

6) С помощью сортировки «пузырьком» я отсортировала массивы lens и order_of_sentences по возрастанию длин предложений. В order_of_sentences хранятся номера предложений в нужном нам порядке.

7) Далее я выводила предложения либо в файл to_file, либо на экран, если нужный флаг не был передан, зная порядок предложений и индексы точек, обозначающие их «границы». В момент вывода я сохранила в переменные max_start и max_end границы самого большого по длине предложения, чтобы дальше его перевернуть нужным образом.

8) Зная границы максимального предложения для удобства я записала его в массив max_sent, одновременно с этим считая количество пробелов в предложении.

9) С помощью функции coord_of_spaces я получаю индексы пробелов внутри предложения. Далее вывожу в файл «NINE.txt» слова в обратном порядке, идя по массиву с индексами проблов в обратном порядке. в массив длин. При этом элементмассива длин соответствовал элементу массива координат точек.

Код программы

main.cpp

```
#include <iostream>
#include <cstring>
#include <fstream>
#include "fun.h"

#define MAX_SIZE 10000

using std::cout;
using std::cin;


int main(int argc, char ** argv){
    //std::cout << "You can use the --from file or/and --tofile flags
if you want to use files for input or/and output. There should be a
file name after each flag.\n" ;
    const char* from_file = "-1"; //названия файлов для ввода и вывода
    const char* to_file = "-1";

    if (argc % 2 == 0 || argc > 5) { //если четное число аргументов,
или их больше, чем возможное число
        std::cerr << "Incorrect format for flags";
        return EXIT_FAILURE;
    }
    else if (argc == 3) {
        if (std::strcmp(argv[1], "--fromfile") == 0){ //если флаг
чтения из файла
            from_file = argv[2];
        }
        else if (std::strcmp(argv[1], "--tofile") == 0){
            to_file = argv[2]; //если флаг вывода в файл
        }
        else {
            std::cerr << "Incorrect format for flags";
            return EXIT_FAILURE;
        }
    }
    else if (argc == 5) {
        // если чтение из файла и ввод в файл
        if (std::strcmp(argv[1], "--fromfile") == 0 and
std::strcmp(argv[3], "--tofile") == 0){
            from_file = argv[2];
            to_file = argv[4];
        }
        else if (std::strcmp(argv[1], "--tofile") == 0 and
std::strcmp(argv[3], "--fromfile") == 0 ){
            from_file = argv[4];
            to_file = argv[2];
        }
        else {
            std::cerr << "Incorrect format for flags";
```

```
            return EXIT_FAILURE;
        }
    }


    char token = ' ';
    int SZ = 0, dots_cnt = 0;
    char *s = new char [MAX_SIZE]; //все предложения
    int *points = new int [MAX_SIZE]; //индексы точек в тексте

    if (strcmp(from_file, "-1") != 0) { //если чтение из файла
        FILE* fp;
        int j = 0;
        fp = fopen(from_file, "r");
        while ((s[j] = fgetc(fp)) != EOF) {
            j++; //читаем, пока не дойдем до конца файла
        }
        s[j] = '\0'; //элемент конца си-й строки
        fclose(fp);
    }

    cout << "Please write a space after the points at the end of
sentences if there are further sentences in the same line.\n";

    if (strcmp(from_file, "-1") == 0) {
        std::cout << "Enter text. \nTo mark the end of input, type '@'
and press Enter.\n" << std::endl;
        std::cin.getline(s, 1024, '@'); //считываем из терминала и
записываем в массив с предложениями, пока не встретим @
    }

    int points_cnt = coord_of_points(s, points); //получаем количество
точек

    int *lens = lens_of_sentens(points, points_cnt); //массив,
содержащий длины всех предложений в изначальном порядке

    int *order_of_sentences = new int [points_cnt]; //массив, в
котором будут хранится номера предложений по увеличению их длины

    for (int i = 0; i < points_cnt; ++i) {
        order_of_sentences[i] = i;
    }
   //сортировка пузырьком по длине предложений. теперь мы знаем
порядок предложений по увеличению их длин
    for (int i = 0; i < points_cnt; i++) {
        for (int j = 0; j < points_cnt - 1; j++) {
            if (lens[j] > lens[j + 1]) {
                std::swap(lens[j], lens[j + 1]);
                std::swap(order_of_sentences[j], order_of_sentences[j
+ 1]);
            }
        }
    }
```

```
    int max_start, max_end;

    if (strcmp(to_file, "-1") == 0) { //если требуется вывод на экран
        for (int i = 0; i < points_cnt; ++i) { //проходимся по всем
точкам
            int start = points[order_of_sentences[i] - 1] + 2, end =
points[order_of_sentences[i]]; //предложение находится между двумя
точками
            if (order_of_sentences[i] == 0) { //если оно первое, то
начинается не с точки, а просто с начала файла
                start = 0;
                end = points[0];
            }
            if (i == points_cnt - 1) { //запоминаем местонахождение
максимального по длине предложения, чтобы потом его перевернуть
                max_start = start;
                max_end = end;
            }
            for (int j = start; j <= end; ++j) { //выводим предложение
                cout << s[j];
            }
            cout << '\n';
        }
    }
    else {
        //аналогично выводу на экран, только в данном случае вывод
происходит в заданный файл
        std::ofstream fout;
        fout.open(to_file);

        for (int i = 0; i < points_cnt; ++i) {
            int start = points[order_of_sentences[i] - 1] + 2, end =
points[order_of_sentences[i]];
            if (order_of_sentences[i] == 0) {
                start = 0;
                end = points[0];
            }
            if (i == points_cnt - 1) {
                max_start = start;
                max_end = end;
            }
            cout << start << ' ' << end << '\n';
            for (int j = start; j <= end; ++j) {
                fout << s[j];
            }
            fout << '\n';
        }
        fout.close();
    }

    int max_sz = max_end - max_start + 1; //размер самого длинного
предложения
    char * max_sent = new char [max_sz]; //

    //записываем нужное предложение в max_sent и считаем количество
```

```
пробелов в нем
    int spaces_cnt = 0;
    for (int i = max_start; i <= max_end; ++i) {
        max_sent[i - max_start] = s[i];
        if (s[i] == ' ') {
            ++spaces_cnt;
        }
    }
    //получаем массив с индексами пробелов
    int *spaces = coord_of_spaces(max_sent, spaces_cnt, max_sz);

    //выводим в нужном формате это предложение в NINE.txt
    std::ofstream fout;
    fout.open("NINE.txt");

    for (int i = spaces_cnt - 1; i >= 0; --i) { //проходимся по всем
пробелам. слово лежит между двумя пробелами
        int temp_end, j;
        if (i == spaces_cnt - 1) { //если слово последнее, то конец
его там же, где и конец массива с предложением
            temp_end = max_sz - 1;
            j = spaces[i] + 1;
        }
        else {
            temp_end = spaces[i + 1]; //начало и конец слова
            j = spaces[i];
        }

        for (j; j < temp_end; ++j) {
            fout << max_sent[j]; //выводим слово
        }
    }
    //для первого слова нельзя получить индекс пробела, после которого
он стоит, поэтому выводим его отдельно
    fout << ' ';
    for (int i = 0; i < spaces[0]; ++i) {
        fout << max_sent[i];
    }
    //очищаем память
    delete [] s;
    delete [] points;
    delete [] lens;
    delete [] order_of_sentences;
    delete [] spaces;
    delete [] max_sent;
    return 0;
}
```

fun.cpp

```
#include "fun.h"
#include <iostream>

int coord_of_points(char *s, int *points)
{ //получаем индексы точек, которые
разделяют предложения
    int ind = 0, sz = std::strlen(s);
    for (int i = 0; i < sz; ++i) {
        if (s[i] == '.') {
            points[ind] = i;
            ++ind;
        }
    }
    return ind;
}

int* lens_of_sentens(int *points, int sz)
{ //считаем размер всех предложений, зная
индексы точек, окружающие его
    int *lens = new int [sz];
    if (sz > 0) {
        lens[0] = points[0];
        for (int i = 1; i < sz; ++i) {
            lens[i] = (points[i] -
points[i - 1] - 2);
        }
    }
    return lens;
}

int* coord_of_spaces(char *max_sent, int
spaces_cnt, int sz) { //получаем индексы
пробелов в максимально длинном
предложении
    int *spaces = new int [spaces_cnt];
    int temp_cnt = 0;
    for (int i = 0; i < sz; ++i) {
        if (max_sent[i] == ' ') {
            spaces[temp_cnt] = i;
            ++temp_cnt;
        }
```

```
    }
    return spaces;
}
```

fun.h

```
int coord_of_points(char *, int *);

int* lens_of_sentens(int *, int);

int* coord_of_spaces(char *, int, int);
```

makefile

```
out : main.o fun.o
     g++ main.o fun.o -o out

main.o : main.cpp
     g++ -c main.cpp -o main.o

fun.o : fun.cpp
     g++ -c fun.cpp -o fun.o

clean :
     rm out *.o

distclean :
     rm out *.o *.txt
```

Вывод

В ходе данной лабораторной работы я изучила новые способы ввода текста из файла, вспомнила реализацию сортировки пузырьком и узнала о новой цели для makefile “distclean”.