ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ» Московский институт электроники и математики им. А. Н. Тихонова

Отчет по лабораторной работе №3 по дисциплине «Языки программирования»

Выполнила: студентка группы СКБ221
Коновалова Александра Романовна

Москва, 2022

Оглавление

Список файлов

Файлы

Полный список файлов.

Файлы

Файл main.cpp

```
#include <iostream>
#include <cstring>
#include <fstream>
```

Функции

- void **Menu** ()
  Функция для вывода подсказок для пользователя

- void **ToFile** (char *fname)
  ToFile - Функция, при которой мы считываем из входного потока символы, считываем их во временный файл по предложения на каждую строку Затем по строкам считываем в 1ый массив предложения, во 2 массив считываем длины, сортируем 2 массив, одновременно сортируя первый и выводим файл Функция принимает параметр в виде названия выходного файла

- void **FromFile** (char *fname)
  FromFile - Функция, при которой мы считываем из входного файла символы, считываем их во временный файл по предложения на каждую строку Затем по строкам считываем в 1ый массив предложения, во 2 массив считываем длины, сортируем 2 массив, одновременно сортируя первый и выводим в консоль Функция принимает параметр в виде названия входного файла

- void **FromAndToFile** (char *in_file, char *to_file)
  FromAndToFile - Функция, при которой мы считываем из входного файла символы, считываем их во временный файл по предложения на каждую строку Затем по строкам считываем в 1ый массив предложения, во 2 массив считываем длины, сортируем 2 массив, одновременно сортируя первый и результат записываем в файл Функция принимает параметры в виде названия входного и выходного файлов

- void **Get_NINEFILE** ()
  Get_NINEFILE - функция, которая работает в ToFile, она в самом длинном предложении переворачивает слова и выводит их в файл nine.txt.

- int **main** (int argc, char *argv[])
  ф-ии main задаем параметры, первый это количество аргументов, второй это массив, состоящий из аргументов, введенных в консоль при запуске программы

Переменные

- const char * **flags** [2] = {"--fromfile","--tofile"}
  создаем массив строк, где будем содержать то, что может вводить пользователь

Функции

void FromAndToFile (char * *in_file*, char * *to_file*)

FromAndToFile - Функция, при которой мы считываем из входного файла символы, считываем их во временный файл по предложения на каждую строку Затем по строкам считываем в 1ый массив предложения, во 2 массив считываем длины, сортируем 2 массив, одновременно сортируя первый и результат записываем в файл Функция принимает параметры в виде названия входного и выходного файлов

См. определение в файле **main.cpp** строка **155**

void FromFile (char * *fname*)

FromFile - Функция, при которой мы считываем из входного файла символы, считываем их во временный файл по предложения на каждую строку Затем по строкам считываем в 1ый массив предложения, во 2 массив считываем длины, сортируем 2 массив, одновременно сортируя первый и выводим в консоль Функция принимает параметр в виде названия входного файла

См. определение в файле **main.cpp** строка **88**

void Get_NINEFILE ()

Get_NINEFILE - функция, которая работает в ToFile, она в самом длинном предложении переворачивает слова и выводит их в файл nine.txt.

См. определение в файле **main.cpp** строка **219**

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

ф-ии main задаем параметры, первый это количество аргументов, второй это массив, состоящий из аргументов, введенных в консоль при запуске программы

См. определение в файле **main.cpp** строка **278**

void Menu ()

Функция для вывода подсказок для пользователя

См. определение в файле **main.cpp** строка **7**

void ToFile (char * *fname*)

ToFile - Функция, при которой мы считываем из входного потока символы, считываем их во временный файл по предложения на каждую строку Затем по строкам считываем в 1ый массив предложения, во 2 массив считываем длины, сортируем 2 массив, одновременно сортируя первый и выводим файл Функция принимает параметр в виде названия выходного файла

См. определение в файле **main.cpp** строка **19**

Переменные

const char* flags[2] = {"--fromfile","--tofile"}

создаем массив строк, где будем содержать то, что может вводить пользователь
См. определение в файле **main.cpp** строка **273**

main.cpp

См. документацию.
```
00001 #include <iostream>
00002 #include <cstring>
00003 #include <fstream>
00007 void Menu(){
00008     std::cout << "Enter as follows:" << std::endl;
00009     std::cout << "To output from a file, enter: ./prog --fromfile <name.txt>" <<
std::endl;
00010     std::cout << "To enter the file, enter: ./prog --tofile <name.txt>" <<
std::endl;
00011     std::cout << "For reading from one file and out to another file enter:
./nameprog --fromfile --tofile <name1.txt> <name2.txt>" << std::endl;
00012
00013 }
00019 void ToFile(char *fname){
00020     std::ofstream LongestSentence ("LongestSentence.txt");   //Файл для ввода в
него самого длинного предложения
00021     std::ofstream temp_out("var.txt");   // Файл для предварительного чтения из
потока cin
00022     std::ofstream result(fname);  // Файл для ввода в него отсортированных
предложений
00023     char   previous token   =   'a'; //иниц. переменной, отвеч. за хранение
предыдущего считанного элемента
00024     int    length     =     0; //иниц переменной, отвеч. за хранение текущей
длины предложения
00025     int   max length   =   0; //иниц. переменной, отвеч. за хранение макс.длины
00026     int   count sentences   =   0; // иниц. переменной, отвеч. за хранения кол-
ва всех предложений
00027     char   current_token   =   '0'; // иниц. переменной для хранения текущего
элемента
00028     while (std::cin.get(current token) && current token != '\n') { //пока
считываем токен и он не переход на новую строку //пока считываем токен и он не переход
на новую строку
00029         if (current_token == '.') {     // если токен - точка
00030             count_sentences++;  //увел. кол-во предложений и длину и обновляем
макс.длину, текущую обнулим
00031             length++;
00032             if (length > max length) max length = length;
00033             length = 0;
00034             previous_token = '.'; // в предыдущий запишем эту самую точку и в
файле сделаем переход на новую строку и поставим точку в конце
00035             temp out << ".\n";
00036         }
00037     else if  (current token == ' ' && previous token == '.') continue; // если
встретился пробел после точки, то продолжаем
00038         else   {                // иначе считываем символ, в предыдущий
присваиваем его и увеличиваем длину
00039             temp out << current token;
00040             previous token = current token;
00041             length++;
00042         }
00043     }
00044     temp_out.close(); //закроем временный файл
00045     max length++; // т.к всё закончилось, то увеличим макс.длину на 1
00046     char array sent[count sentences][max length];  //двум.массив, хранящий
предложения(по сути это матрица m на n, где m - макс.длина, n - число предл.
00047     int array_len[count_sentences];
00048     std::ifstream in temp file ("var.txt"); // теперь наш предыдущий файл для
ввода в него делаем таким, чтобы из него можно было считать символы
00049     for (int i = 0; i < count sentences; i++) { //пройдясь 2-ным циклом по
массиву, перепишем данные из файла в него, как раз в каждом элементе будет предложение
00050         for (int j = 0; j < max length; j++) {
00051             in_temp_file.get(current_token);
00052             if (current token == '\n') {
00053                 array sent[i][j] = '\0';
00054                 array len[i] = j;
00055                 break;
00056             }
00057             array_sent[i][j] = current_token;
00058         }
00059     }
00060     in temp file.close(); //вновь закроем временный файл
00061   int var = 0;   //временная переменная для пузырьковой сортировки
```

```
00062   char buffer[max length]; // буфер в качестве массива, который будет хранить
временно предложения в сортировке
00063     for (int i = 0; i < count sentences;i++){
00064         for (int j = 0; j < count sentences - i - 1;j++){
00065             if (array_len[j]> array_len[j+1]){
00066                     var = array_len[j];
00067                     array len[j] = array len[j+1];
00068                     array len[j+1] = var;
00069                     strcpy(buffer,array sent[j]);
00070                     strcpy(array sent[j],array sent[j+1]);
00071                     strcpy(array_sent[j+1],buffer);
00072             }
00073         } //проводим сортировку
00074     }
00075     for (int i = 0; i < count sentences; i++)
00076
00077
00078     result << array_sent[i] <<  std::endl; // выводим предложения на каждую
строку в наш результирующий файл
00079     LongestSentence << array sent[count sentences-1] << std::endl; // в другой
файл выводим самое длинное предложение
00080     result.close(); // закрываем файлы
00081     LongestSentence.close();
00082 }
00088 void FromFile(char *fname) {
00089     std::ofstream temp out ("var.txt"); // Файл для временного хранения того что
считали с введеного файла и разбиения на предложения
00090     std::ifstream infile (fname); //файл с которого мы считаем символы
00091     char   previous_token   =   'a'; //иниц. переменной, отвеч. за хранение
предыдущего считанного элемента
00092     int    length     =     0; //иниц переменной, отвеч. за хранение текущей
длины предложения
00093     int   max length   =   0; //иниц. переменной, отвеч. за хранение макс.длины
00094     int   count_sentences   =   0; // иниц. переменной, отвеч. за хранения кол-
ва всех предложений
00095     char   current token   =   '0'; // иниц. переменной для хранения текущего
элемента
00096     if (infile.is open()){ // если файл открылся
00097         while (infile.get(current_token) && current_token != '\n') { //пока
считываем токен и он не переход на новую строку
00098             if (current_token == '.') {     // если токен - точка
00099                 count sentences++;  //увел. кол-во предложений и длину и
обновляем макс.длину, текущую обнулим
00100                 length++;
00101                 if (length > max_length) max_length = length;
00102                 length = 0;
00103                 previous_token = '.'; // в предыдущий запишем эту самую точку и
в файле сделаем переход на новую строку и поставим точку в конце
00104                 temp out << ".\n";
00105       }
00106         else if  (current_token == ' ' && previous_token == '.') continue; //
если встретился пробел после точки, то продолжаем
00107         else   {                // иначе считываем символ, в предыдущий
присваиваем его и увеличиваем длину
00108             temp out << current token;
00109             previous_token = current_token;
00110             length++;
00111       }
00112     }
00113   }
00114     else std::cerr << "The file is not found" << std::endl; //если не открылся,
то выводим ошибку
00115     infile.close();
00116     temp_out.close(); //закроем временный файл
00117     max length++; // т.к всё закончилось, то увеличим макс.длину на 1
00118     char array sent[count sentences][max length];  //двум.массив, хранящий
предложения(по сути это матрица m на n, где m - макс.длина, n - число предл.
00119     int array_len[count_sentences];
00120     std::ifstream in_temp_file ("var.txt"); // теперь наш предыдущий файл для
ввода в него делаем таким, чтобы из него можно было считать символы
00121     for (int i = 0; i < count sentences; i++) { //пройдясь 2-ным циклом по
массиву, перепишем данные из файла в него, как раз в каждом элементе будет предложение
00122         for (int j = 0; j < max_length; j++) {
00123             in_temp_file.get(current_token);
00124             if (current_token == '\n') {
00125                 array sent[i][j] = '\0';
00126                 array_len[i] = j;
```

```
00127                 break;
00128             }
00129             array sent[i][j] = current token;
00130         }
00131     }
00132     in_temp_file.close(); //вновь закроем временный файл
00133     int var = 0;   //временная переменная для пузырьковой сортировки
00134     char buffer[max length]; // буфер в качестве массива, который будет хранить
временно предложения в сортировке
00135     for (int i = 0; i < count sentences;i++){
00136         for (int j = 0; j < count_sentences - i - 1;j++){
00137             if (array_len[j]> array_len[j+1]){
00138                     var = array len[j];
00139                     array len[j] = array len[j+1];
00140                     array len[j+1] = var;
00141                     strcpy(buffer,array sent[j]);
00142                     strcpy(array_sent[j],array_sent[j+1]);
00143                     strcpy(array_sent[j+1],buffer);
00144             }
00145         } //проводим сортировку
00146     }
00147     for (int i = 0; i < count sentences; ++i)
00148         std::cout << array_sent[i]  << std::endl; // Вывод отсортированных
предложений
00149 }
00155 void FromAndToFile(char *in file, char *to file){
00156     std::ofstream temp out("var.txt");
00157     std::ifstream infile(in_file);
00158     std::ofstream result(to_file);
00159     char   previous token   =   'a';
00160     int    length     =     0;
00161     int   max length   =   0;
00162     int   count sentences   =   0;
00163     char   current_token   =   '0';
00164     while (infile.get(current_token) && current_token != '\n') { //пока
считываем токен и он не переход на новую строку
00165         if (current token == '.') {     // если токен - точка
00166             count sentences++;  //увел. кол-во предложений и длину и обновляем
макс.длину, текущую обнулим
00167             length++;
00168             if (length > max_length) max_length = length;
00169             length = 0;
00170             previous token = '.'; // в предыдущий запишем эту самую точку и в
файле сделаем переход на новую строку и поставим точку в конце
00171             temp_out << ".\n";
00172     }
00173         else if  (current_token == ' ' && previous_token == '.') continue; //
если встретился пробел после точки, то продолжаем
00174         else   {                // иначе считываем символ, в предыдущий
присваиваем его и увеличиваем длину
00175             temp_out << current_token;
00176             previous_token = current_token;
00177             length++;
00178         }
00179     }
00180     temp_out.close();
00181     infile.close();
00182     temp out.close(); //закроем временный файл
00183     max length++; // т.к всё закончилось, то увеличим макс.длину на 1
00184     char array sent[count sentences][max length];  //двум.массив, хранящий
предложения(по сути это матрица m на n, где m - макс.длина, n - число предл.
00185     int array_len[count_sentences];
00186     std::ifstream in_temp_file ("var.txt"); // теперь наш предыдущий файл для
ввода в него делаем таким, чтобы из него можно было считать символы
00187     for (int i = 0; i < count sentences; i++) { //пройдясь 2-ным циклом по
массиву, перепишем данные из файла в него, как раз в каждом элементе будет предложение
00188         for (int j = 0; j < max length; j++) {
00189             in_temp_file.get(current_token);
00190             if (current_token == '\n') {
00191                 array_sent[i][j] = '\0';
00192                 array len[i] = j;
00193                 break;
00194             }
00195             array_sent[i][j] = current_token;
00196         }
00197     }
00198     in_temp_file.close(); //вновь закроем временный файл
```

```
00199     int var = 0;   //временная переменная для пузырьковой сортировки
00200     char buffer[max length]; // буфер в качестве массива, который будет хранить
временно предложения в сортировке
00201     for (int i = 0; i < count sentences;i++){
00202         for (int j = 0; j < count_sentences - i - 1;j++){
00203             if (array_len[j]> array_len[j+1]){
00204                     var = array len[j];
00205                     array len[j] = array len[j+1];
00206                     array len[j+1] = var;
00207                     strcpy(buffer,array sent[j]);
00208                     strcpy(array_sent[j],array_sent[j+1]);
00209                     strcpy(array_sent[j+1],buffer);
00210             }
00211         } //проводим сортировку
00212     }
00213     for (int i = 0; i < count sentences; i++)
00214         result << array_sent[i]  << std::endl; // Записываем предложения в файл
00215 }
00219 void Get NINEFILE(){
00220     std::ifstream infile("LongestSentence.txt"); //Откроем файл для чтения
наибольшего предложения
00221     std::ofstream var("tempfile.txt"); //Временный файл для обработки
предложения
00222     char current token = '0';
00223     char previous token = 'a';
00224     int length = 0;
00225     int max length = 0;
00226     int count_words = 1;
00227     while (infile.get(current_token)){ // Пока считываем токен
00228         if (current token == ' '){   //если он является пробелом, то увеличиваем
количество слов и обновляем макс.длину, и затем текущую обнуляем
00229             count words+=1;
00230             max length = std::max(++length, max length);
00231             length = 0;
00232             var << "\n";
00233         }
00234         else { // В другом случае просто считаем символы, увеличивая длину
00235             var << current token;
00236             length++;
00237     }
00238     }
00239     infile.close();
00240     var.close(); // Закрываем файлы
00241     max length++;
00242     char words[count_words][max_length]; //Массив слов
00243     std::ifstream temp("tempfile.txt");
00244     std::ofstream res_file("nine.txt"); //Файл, куда запишем результат
переворота предложения
00245     for (int i = 0; i< count words;i++){ //Пройдясь циклом по массиву, если
считанный символ - переход на новую строку, то
00246         for (int j = 0; j< max_length;j++){
00247             temp.get(current_token);
00248
00249
00250             if (current token == '\n') { // то текущим элементом будет пробел
00251                     words[i][j] = '\0';
00252                     break; //выходим из цикла
00253                 }
00254             if (current token!='.'){ //если это не точка, то записываем символ в
массив
00255
00256                 words[i][j] = current_token;
00257                 }
00258     }
00259     }
00260     char token;
00261     for (int i = count words-1;i>=0;i--){ //Проходимся обратным циклом по i, и
пока токен это элемент массива, мы его считываем в файл
00262         int j = 0;
00263         while (token = words[i][j++]){
00264             res file << token;
00265         }
00266         res_file << " ";  //при окончании слова ставим пробел в файл в качестве
разделителя
00267     }
00268     temp.close();  //закрываем временный файл
00269 }
```

```
00273 const char* flags[2] = {"--fromfile","--tofile"}; //создаем массив строк, где
будем содержать то, что может вводить пользователь
00278 int main(int argc, char*argv[]){ //ф-ии main задаем параметры, первый это
количество аргументов,
00279 //второй это массив, состоящий из аргументов, введенных в консоль при запуске
программы
00280     if (argc == 1){ //если было введено только название out файла, то выводим
ошибку
00281         std::cerr << "ERROR: flag(s) is not found" << std::endl;
00282         Menu();
00283     }
00284     else if (argc == 2){ // если ввели что-то еще:
00285         if (!strcmp(argv[1],flags[1])){//если ввели --tofile, то выводим ошибку
о требовании ввести название выходного файла
00286             std::cerr << "ERROR: name file is not found" << std::endl; //и
выводим меню для --tofile
00287             Menu();
00288         }
00289         else if (!strcmp(argv[1],flags[0])){ //если ввели --fromfile, то выводим
ошибку о требовании ввести название входного файла
00290             std::cerr << "ERROR: name file is not found" << std::endl; //и
выводим меню для --fromfile
00291             Menu();
00292         }
00293         else { //если было введено что-то другое, то выводим ошибку и выводим
общее меню
00294             std::cerr << "ERROR: incorrect flag" << std::endl;
00295             Menu();
00296         }
00297     }
00298     else if (argc == 3){ // если было введено всего 3 флага, включая название
программы
00299         char *fname = argv[2]; // в массиве будем хранить название 3 флага,
подразумевается, что это название файла
00300         if (!strcmp(argv[1],flags[1]))  { //если это флаг --tofile, то выполняем
tofile и Get NINEFILE(на 9 баллов)
00301             ToFile(fname);
00302             Get NINEFILE();
00303         }
00304         else if (((!strcmp(argv[1],flags[0])) && (!strcmp(argv[2],flags[1]))) ||
//иначе если мы ввели --tofile и --fromfile, то
00305                 ((!strcmp(argv[1],flags[1])) && (!strcmp(argv[2],flags[0])))){
00306                 std::cerr << "ERROR: you need to enter namefiles" << std::endl;;
//потребовать ввести файлы для флагов с пом. меню
00307                 Menu();
00308                 }
00309         else if (!strcmp(argv[1],flags[0]))     FromFile(fname); //если введен -
-tofile, то выполнить FromFile
00310         else {
00311             std::cerr << "ERROR:incorrect flag" << std::endl; // если введено
что-то другое, то пишем ошибку
00312             Menu();
00313     }
00314     else if (argc == 5){ //если аргументов было введено 5(включая файл)
00315         char *in_file = argv[2]; //пусть названиями файлов будет (2+1)ый и
(4+1)ый аргумент
00316         char *to_file = argv[4];
00317         if ((!strcmp(argv[1],flags[0])) && (!strcmp(argv[3], flags[1]))) {
//если 2 аргумент - это fromfile , а 4ый это tofile
00318             FromAndToFile(in_file,to_file); // то выполняем FromAndToFile
00319         }
00320         else {
00321         std::cerr << "ERROR: incorrect input" << std::endl;
00322         FromAndToMenu(); // а иначе вывести ошибку с выводом меню для выполнения
одновременно обоих флагов
00323         }
00324     }
00325     else { // если же флагов было введено неверное кол-во, то вывести ошибку и
меню
00326         std::cerr << "ERROR: unnormal count of flags" << std::endl;
00327         Menu();
00328 }
00329 }
```

Алфавитный указатель

INDEX