ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ «НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А. Н. Тихонова

Отчет по Лабороторной работе №3
по предмету «Языки Программирования»

Выполнил: студент группы СКБ221
Нугманов М.И.

Москва 2022 г.

Оглавление

Введение

В программу вводится натуральное число **T** <= 10000, а затем вводятся начала и концы отрезков в количестве **T** штук. Программа должна вывести объединения отрезков в порядке возрастания их начал на числовой прямой.

Вывод:

```
5
1.2 3.4
-1.8 0.1
0.5 1.5
1.3 2.8
-2 -1
```

Вывод:

```
RESULT:

-2 0.1
0.5 3.4
```

Для компиляции программы нужно написать команду make в командную строку. С помощью команды make clean можно удалить объектные файлы и созданный программой файл DefaultFile.txt (в случае выбора флага --tofile, но без выбора конкретного файла для вывода результата), а с помощью make distclean – удалить объектные файлы, файл Default.txt (в случае выбора флага --tofile, но без выбора конкретного файла для вывода результата) и скомпилированную программу.

Ошибки

Член main (int argc, char **argv)

ERROR: flag not found!
ERROR: You must enter the file name for flag '–fromfile'!
ERROR: invalid input!
ERROR: this flag doesn't need another argument!

Список файлов

Файлы

Полный список файлов:

- **functions.cpp (Файл, в котором описаны функции для решения задачи об отрезках)**
- **functions.h**
- **main.cpp (Главный файл, в котором реализован ввод флагов и названий файлов пользователем)**

Файлы

Файл functions.cpp

Файл, в котором описаны функции для решения задачи об отрезках.
Включенные заголовочные файлы:

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

Функции

void **Help** ()

Help() - *функция для вывода инструкции по работе с программой*

int **IsDigit** (char symbol)

IsDigit() - *функция, проверяющая, является ли введенный символ цифрой*

int **IsMinus** (char symbol)

IsMinus() - *функция, проверяющая, является ли введенный символ знаком Минус*

int **IsSpace** (char symbol)

IsSpace() - *функция, проверяющая, является ли введенный символ Пробелом*

int **IsPoint** (char symbol)

IsPoint() - *функция, проверяющая, является ли введенный символ Точкой*

int **IsCorrectSymbol** (char symbol)

IsCorrectSymbol() - *функция, проверяющая, является ли введенный символ корректным для работы программы*

int **IsSegment** (double a, double b)

IsSegment() - *функция, проверяющая, является ли пара введенных вещественных чисел концами отрезка*

double **GetNumber** (int positive)

GetNumber() - *функция, обрабатывающая массив символов входного потока и конвертирующая их в число*

double **GetMax** (double a, double b)

GetMax() - *фукнция, принимающая на вход два числа и возвращающая максимальное из них*

int **GetNumberSegments** ()

GetNumberSegments() - *функция, работающая с массивом stream и возвращающая введенное число отрезков*

void **SortSegments** (int segments_count)

SortSegments() *- функция, сортирующая отрезки по возрастанию их начал. Используется сортировка Insertion Sort.*

void **ConvertToSegments** (int index_segments)

ConvertToSegments() *- функция, преобразующая символы входного потока в числа, обозначающие концы и начала отрезка*

void **SolutionToConsole** (int segments_count)

SolutionToConxole() - функция, в который реализован основной алгоритм поиска обьединенных отрезков

void **SolutionToFile** (const char *nametofile, int segments_count)

SolutionToFile() *- функция, в которой реализован основной алгоритм поиска обьединенных отрезков*

void **GetResultFromConsole** ()

GetResultFromConsole() *- функция, осуществляющая ввод данных пользователем с консоли и вывод также в консоль*

void **GetResultFromConsole** (const char *nametofile, int is_name_file)

GetResultFromConsole() *- перегруженная функция, осуществляющая ввод данных пользователем с консоли и вывод в файл*

void **GetResultFromFile** (const char *namefromfile)

GetResultFromFile() *- функция, осуществляющая ввод данных пользователем с файла и вывод в консоль*

void **GetResultFromFile** (const char *namefromfile, const char *nametofile, int is_name_file)

GetResultFromFile() *- перегруженная функция, осуществляющая ввод данных пользователем с файла и вывод в файл*

Переменные

const int **STREAM_SIZE** = 32

Константная переменная типа Integer, хранящая величину массива stream, в который записываются символы с входного потока.

char **stream** [**STREAM_SIZE**]

Статический массив типа Char, в котором хранится входной поток символов с консоли / файла

double * **starts**

Динамический массив типа Double, в котором хранятся начала отрезков

double * **ends**

Динамический массив типа Double, в котором хранятся концы отрезков

int **index_stream**

Переменная типа Integer, предназначенная для обращения к ячейке массива stream.

char **token**

Переменная типа Char, в который записывается текущий символ входного потока

double **number1** = 0

Переменная типа Double, в котором будет храниться начало отрезка

double **number2** = 0

Переменная типа Double, в котором будет храниться конец файла

Подробное описание

Файл, в котором описаны функции для решения задачи об отрезках.

См. определение в файле **functions.cpp**

Функции

void ConvertToSegments (int *index_segments*)

ConvertToSegments() - функция, преобразующая символы входного потока в числа, обозначающие концы и начала отрезка

Аргументы

index_segments	Текущий инденкс. В соответствующие ячейки массивов starts и ends запишутся начало и конец отрезка соответственно

См. определение в файле **functions.cpp** строка **227**

double GetMax (double *a*, double *b*)

GetMax() - фукнция, принимающая на вход два числа и возвращающая максимальное из них

Аргументы

a	Первое число
b	Второе число

Возвращает

a - если a > b

b - если b >= a

Заметки

В случае равенства чисел нам неважно, какое число возвращать, поэтому по умолчанию возвращается b

См. определение в файле **functions.cpp** строка **169**

double GetNumber (int *positive*)

GetNumber() - функция, обрабатывающая массив символов входного потока и конвертирующая их в число

Аргументы

positive	Переменная, хранящая 0 или 1 и показывающая, является ли возвращаемое число отрицательным или положительным соответственно

Возвращает

положительное число, если positive равен 1

отрицательное число, если positive равен 0

См. определение в файле **functions.cpp** строка **110**

int GetNumberSegments ()

GetNumberSegments() - функция, работающая с массивом stream и возвращающая введенное число отрезков

Возвращает

result - введенное пользователем число отрезков

См. определение в файле **functions.cpp** строка **177**

void GetResultFromConsole ()

GetResultFromConsole() - функция, осуществляющая ввод данных пользователем с консоли и вывод также в консоль

См. определение в файле **functions.cpp** строка **337**

void GetResultFromConsole (const char * ***nametofile*****, int** ***is_name_file*****)**

GetResultFromConsole() - перегруженная функция, осуществляющая ввод данных пользователем с консоли и вывод в файл

Аргументы

nametofile	Название файла, в который будет записан результат выполнения функции
is_name_file	Флаг, обозначающий, ввел ли пользователь название файла или нет

Заметки

Если is_name_file = 1, значит пользователь выбрал конкретный файл, в который будет записан результат выполения функции. Если is_name_file = 0, значит пользователь не вводил название файла ,запись будет осуществленная в файл DefaultFile.txt по умолчанию

См. определение в файле **functions.cpp** строка **358**

void GetResultFromFile (const char * ***namefromfile*****)**

GetResultFromFile() - функция, осуществляющая ввод данных пользователем с файла и вывод в консоль

Аргументы

namefromfile	Название файла, из которого будет производиться чтение данных

См. определение в файле **functions.cpp** строка **397**

void GetResultFromFile (const char * *namefromfile*, const char * *nametofile*, int *is_name_file*)

GetResultFromFile() - перегруженная функция, осуществляющая ввод данных пользователем с файла и вывод в файл

Аргументы

namefromfile	Названия файла, с которого будет производиться чтение данных
nametofile	Название файла, в который будет производиться запись данных
is_name_file	Флаг, обозначающий, ввел ли пользователь название файла для вывода данных или нет

Заметки

Если is_name_file = 1, значит пользователь выбрал конкретный файл, в который будет записан результат выполения функции. Если is_name_file = 0, значит пользователь не вводил название файла ,запись будет осуществленная в файл DefaultFile.txt по умолчанию

См. определение в файле **functions.cpp** строка **430**

void Help ()

Help() - функция для вывода инструкции по работе с программой

См. определение в файле **functions.cpp** строка **28**

int IsCorrectSymbol (char *symbol*)

IsCorrectSymbol() - функция, проверяющая, является ли введенный символ корректным для работы программы

Аргументы

symbol	Переменная, хранящая символ, который проходит проверку на корректность

Возвращает

1 - если символ является корректным для ввода

0 - если символ не является корректным для ввода

См. определение в файле **functions.cpp** строка **89**

int IsDigit (char *symbol*)

IsDigit() - функция, проверяющая, является ли введенный символ цифрой

Аргументы

symbol	Переменная, хранящая символ, который проходит проверку на цифру

Возвращает

1 - если символ является цифрой

0 - если символ не является цифрой

См. определение в файле **functions.cpp** строка **49**

int IsMinus (char *symbol*)

IsMinus() - функция, проверяющая, является ли введенный символ знаком Минус

Аргументы

symbol	Переменная, хранящая символ, который проходит проверку на знак Минус

Возвращает

1 - если символ является знаком Минус

0 - если символ не является знаком Минус

См. определение в файле **functions.cpp** строка **59**

int IsPoint (char *symbol*)

IsPoint() - функция, проверяющая, является ли введенный символ Точкой

Аргументы

symbol	Переменная, хранящая символ, который проходит проверку на Точку

Возвращает

1 - если символ является Точкой

0 - если символ не является Точкой

См. определение в файле **functions.cpp** строка **79**

int IsSegment (double *a*, double *b*)

IsSegment() - функция, проверяющая, является ли пара введенных вещественных чисел концами отрезка

Аргументы

a	Переменная, хранящая вещественное число и являющаяся началом отрезка
b	Переменная, хранящая вещественное число и являющаяся концом отрезка

Возвращает

1 - если конец отрезка строго больше начала, т.е. пара чисел

0 - в противном случае

См. определение в файле **functions.cpp** строка **100**

int IsSpace (char *symbol*)

IsSpace() - функция, проверяющая, является ли введенный символ Пробелом

Аргументы

symbol	Переменная, хранящая символ, который проходит проверку на Пробел

Возвращает

1 - если символ является Пробелом

0 - если символ не является Пробелом

См. определение в файле **functions.cpp** строка **69**

void SolutionToConsole (int *segments_count*)

SolutionToConxole() - функция, в который реализован основной алгоритм поиска обьединенных отрезков

Заметки

Результат функции выводится на консоль

Аргументы

segments_count	Количество введенных пользователем отрезков

См. определение в файле **functions.cpp** строка **281**

void SolutionToFile (const char * *nametofile*, int *segments_count*)

SolutionToFile() - функция, в которой реализован основной алгоритм поиска обьединенных отрезков

Заметки

Результат функции записывается в файл

Аргументы

nametofile	Название файла, куда запишется результат функции
segments_count	Количество введенных пользователем отрезков

См. определение в файле **functions.cpp** строка **310**

void SortSegments (int *segments_count*)

SortSegments() - функция, сортирующая отрезки по возрастанию их начал. Используется сортировка Insertion Sort.

Аргументы

segments_conut	количество отрезков, которые требуется отсортировать

См. определение в файле **functions.cpp** строка **204**

Переменные

double* ends

Динамический массив типа Double, в котором хранятся концы отрезков

См. определение в файле **functions.cpp** строка **16**

int index_stream

Переменная типа Integer, предназначенная для обращения к ячейке массива stream.

См. определение в файле **functions.cpp** строка **18**

double number1 = 0

Переменная типа Double, в котором будет храниться начало отрезка

См. определение в файле **functions.cpp** строка **22**

double number2 = 0

Переменная типа Double, в котором будет храниться конец файла

См. определение в файле **functions.cpp** строка **24**

double* starts

Динамический массив типа Double, в котором хранятся начала отрезков

См. определение в файле **functions.cpp** строка **14**

char stream[STREAM_SIZE]

Статический массив типа Char, в котором хранится входной поток символов с консоли / файла

См. определение в файле **functions.cpp** строка **12**

const int STREAM_SIZE = 32

Константная переменная типа Integer, хранящая величину массива stream, в который записываются символы с входного потока.

См. определение в файле **functions.cpp** строка **10**

char token

Переменная типа Char, в который записывается текущий символ входного потока
См. определение в файле **functions.cpp** строка **20**

functions.cpp

См. документацию.

```
00001
00005 #include <iostream>
00006 #include <fstream>
00007 #include <cstring>
00008 #include <cmath>
00010 const int STREAM_SIZE = 32;
00012 char      stream[STREAM_SIZE];
00014 double *  starts;
00016 double *  ends;
00018 int       index_stream;
00020 char      token;
00022 double    number1 = 0;
00024 double    number2 = 0;
00028 void Help() {
00029     std::cout << "\n";
00030     std::cout <<
"############################################################_INSTRUCTION_##############
##################################################" << std::endl;
00031     std::cout << "You must enter:  [./<program_name>] [*] "
<< std::endl;
00032     std::cout << "-------------------------------------------------------------
-------------------------------------------------------------------------" <<
std::endl;
00033     std::cout << "Instead of the [*], You must enter the following options:"
<< std::endl;
00034     std::cout << "  1) --help .....................................(output of
instructions for working with the program)"                                <<
std::endl;
00035     std::cout << "  2) --default ..................................(console
input and console output)"                                                  <<
std::endl;
00036     std::cout << "  3) --tofile ...................................(console
input and default file 'DefaultFile.txt' output)"                           <<
std::endl;
00037     std::cout << "  4) --tofile <filename> ........................(console
input and user-selected file output)"                                       <<
std::endl;
00038     std::cout << "  5) --fromfile <filename> ......................(user-
selected file input and console output)"                                      <<
std::endl;
00039     std::cout << "  6) --fromfile <filename1> --tofile <filename2> ...(user-
selected file input and user-selected file output, neccesary different files!)" <<
std::endl;
00040     std::cout <<
"################################################################################
##################################################" << std::endl;
00041     std::cout << "\n";
00042 }
00049 int IsDigit(char symbol) {
00050     if(symbol >= '0' && symbol <= '9') return 1;
00051     return 0;
00052 }
00059 int IsMinus(char symbol) {
00060     if(symbol == '-') return 1;
00061     return 0;
00062 }
00069 int IsSpace(char symbol) {
00070     if(symbol == ' ') return 1;
00071     return 0;
00072 }
00079 int IsPoint(char symbol) {
00080     if(symbol == '.') return 1;
00081     return 0;
00082 }
00089 int IsCorrectSymbol(char symbol) {
00090     if(IsDigit(symbol) || IsMinus(symbol) || IsSpace(symbol) || IsPoint(symbol))
return 1;
00091     return 0;
00092 }
00100 int IsSegment(double a, double b) {
00101     if(a < b) return 1;
00102     return 0;
00103 }
```

```
00110 double GetNumber(int positive) {
00115     double whole_part    = 0;
00120     double decimal_part  = 0;
00125     int    count_decimal = 0;
00130     int    empty         = 0;
00131     while(IsDigit(stream[index_stream]))
00132         whole_part = whole_part * 10 + (stream[index_stream++] - '0');
00133     if(IsPoint(stream[index_stream])) {
00134         if(IsDigit(stream[index_stream + 1])) {
00135             index_stream++;
00136             while(IsDigit(stream[index_stream])) {
00137                 empty = empty * 10 + (stream[index_stream++] - '0');
00138                 count_decimal++;
00139                 if(count_decimal > 2) {
00140                     std::cerr << "ERROR: incorrect number of decimal places!" <<
std::endl;
00141                     exit(1);
00142                 }
00143             }
00144         }
00145         else {
00146             std::cerr << "ERROR: decimal part not found!" << std::endl;
00147             exit(1);
00148         }
00149     }
00150     if(IsSpace(stream[index_stream]) || !stream[index_stream])
00151         decimal_part = empty / pow(10, count_decimal);
00152     else {
00153         std::cerr << "ERROR: number must be separated by a space!" << std::endl;
00154         exit(1);
00155     }
00156
00157     if(positive)
00158         return (whole_part + decimal_part);
00159     return -(whole_part + decimal_part);
00160 }
00169 double GetMax(double a, double b) {
00170     if(a > b) return a;
00171     return b;
00172 }
00177 int GetNumberSegments() {
00178     int result;
00179     if(token = stream[index_stream]) {
00180         while(token = stream[index_stream++]) {
00181             if(!IsDigit(token)) {
00182                 std::cerr << "ERROR: invalid characters in the natural number of
segments input!" << std::endl;
00183                 exit(1);
00184             }
00185         }
00186     }
00187     else {
00188         std::cerr << "ERROR: number of segments not entered!" << std::endl;
00189         exit(1);
00190     }
00191     index_stream = 0;
00192     result = int(GetNumber(1));
00193     memset(stream, '\0', sizeof(stream));
00194     if(result <= 0 || result > 10000) {
00195         std::cerr << "ERROR: You must enter a natural number of segments not
exceeding 10000!" << std::endl;
00196         exit(1);
00197     }
00198     return result;
00199 }
00204 void SortSegments(int segments_count) {
00205     for(int i = 0; i < segments_count; i++) {
00206         double max_start = -10000000000;
00207         int index_max = 0;
00208         double empty;
00209         for(int j = 0; j < segments_count - i; j++) {
00210             if(starts[j] > max_start) {
00211                 max_start = starts[j];
00212                 index_max = j;
00213             }
00214         }
00215         empty = starts[segments_count-1-i];
```

```
00216         starts[segments_count-1-i] = starts[index_max];
00217         starts[index_max] = empty;
00218         empty = ends[segments_count-1-i];
00219         ends[segments_count-1-i] = ends[index_max];
00220         ends[index_max] = empty;
00221     }
00222 }
00227 void ConvertToSegments(int index_segments) {
00228     index_stream = 0;
00229     while(token = stream[index_stream++]) {
00230         if(!IsCorrectSymbol(token)) {
00231             std::cerr << "ERROR: invalid character!" << std::endl;
00232             exit(1);
00233         }
00234     }
00235     index_stream = 0;
00236     while(token = stream[index_stream++]) {
00237         if(IsMinus(token)) {
00238             if(IsDigit(stream[index_stream])) {
00239                 number1 = GetNumber(0);
00240             }
00241             else {
00242                 std::cerr << "ERROR: segment enteres incorrectly!" << std::endl;
00243                 exit(1);
00244             }
00245         }
00246         else if(IsDigit(token)) {
00247             index_stream--;
00248             number1 = GetNumber(1);
00249         }
00250         else if(IsSpace(token)) {
00251             if((IsMinus(stream[index_stream]) && IsDigit(stream[index_stream +
1])) || IsDigit(stream[index_stream])) {
00252                 token = stream[index_stream];
00253                 if(IsMinus(token)) {
00254                     index_stream++;
00255                     number2 = GetNumber(0);
00256                 }
00257                 else if(IsDigit(token)) {
00258                     number2 = GetNumber(1);
00259                 }
00260             }
00261             else {
00262                 std::cerr << "ERROR: segment enteres incorrectly!" << std::endl;
00263                 exit(1);
00264             }
00265         }
00266     }
00267     if(!IsSegment(number1, number2)) {
00268         std::cerr << "ERROR: the beginning of the segment must be less than the
end!" << std::endl;
00269         exit(1);
00270     }
00271     else {
00272         starts[index_segments] = number1;
00273         ends[index_segments]   = number2;
00274     }
00275 }
00281 void SolutionToConsole(int segments_count) {
00282     std::cout << "\nRESULT:\n" << std::endl;
00283     double glob start = starts[0];
00284     double glob_end   = ends[0];
00285     int    count      = 1;
00286     if(segments_count == 1) {
00287         std::cout << count << ") " << glob_start << " " << glob_end <<
std::endl;
00288     }
00289     else {
00290         for(int i = 1; i < segments_count; i++) {
00291             if(starts[i] <= glob_end) {
00292                 glob_end = GetMax(glob_end, ends[i]);
00293             }
00294
00295             else {
00296                 std::cout << count++ << ") " << glob_start << " " << glob_end <<
std::endl;
00297                 glob_start = starts[i];
```

```
00298                 glob_end   = ends[i];
00299             }
00300         }
00301         std::cout << count << ") " << glob_start << " " << glob_end <<
std::endl;
00302     }
00303 }
00310 void SolutionToFile(const char * nametofile, int segments_count) {
00311     std::ofstream ToFile(nametofile);
00312     ToFile << "RESULT:\n" << std::endl;
00313     double glob_start = starts[0];
00314     double glob_end   = ends[0];
00315     int    count      = 1;
00316     if(segments_count == 1) {
00317         ToFile << count << ") " << glob_start << " " << glob_end << std::endl;
00318     }
00319     else {
00320         for(int i = 1; i < segments_count; i++) {
00321             if(starts[i] <= glob_end) {
00322                 glob_end = GetMax(glob_end, ends[i]);
00323             }
00324             else {
00325                 ToFile << count++ << ") " << glob_start << " " << glob_end <<
std::endl;
00326                 glob_start = starts[i];
00327                 glob_end   = ends[i];
00328             }
00329         }
00330         ToFile << count << ") " << glob_start << " " << glob_end << std::endl;
00331     }
00332     ToFile.close();
00333 }
00337 void GetResultFromConsole() {  // из консоли в консоль
00338     std::cin.getline(stream, STREAM_SIZE - 1); // водим вол-во отрезков
00339     int NUM_SEGMENTS = GetNumberSegments();
00340     starts = new double[NUM_SEGMENTS];
00341     ends   = new double[NUM_SEGMENTS];
00342     for(int i = 0; i < NUM_SEGMENTS; i++) {
00343         std::cin.getline(stream, STREAM_SIZE - 1);
00344         ConvertToSegments(i);
00345     }
00346     SortSegments(NUM_SEGMENTS);
00347     SolutionToConsole(NUM_SEGMENTS);
00348     delete[] starts;
00349     delete[] ends;
00350 }
00358 void GetResultFromConsole(const char * nametofile, int is_name_file) { // из
консоли в файл
00359     std::cin.getline(stream, STREAM_SIZE - 1); // водим вол-во отрезков
00360     int NUM_SEGMENTS = GetNumberSegments();
00361     starts = new double[NUM_SEGMENTS];
00362     ends   = new double[NUM_SEGMENTS];
00363     for(int i = 0; i < NUM_SEGMENTS; i++) {
00364         std::cin.getline(stream, STREAM_SIZE - 1);
00365         ConvertToSegments(i);
00366     }
00367     SortSegments(NUM_SEGMENTS);
00368
00369     if(is_name_file) {
00370         std::ifstream check(nametofile);
00371         if(check.is_open()) {
00372             check.close();
00373             SolutionToFile(nametofile, NUM_SEGMENTS);
00374             std::cout << "-COMMAND COMPLETED-" << std::endl;
00375             delete[] starts;
00376             delete[] ends;
00377         }
00378         else {
00379             check.close();
00380             delete[] starts;
00381             delete[] ends;
00382             std::cerr << "ERROR: file for output not found, You must create it
or enter right!" << std::endl;
00383             exit(1);
00384         }
00385     }
00386     else {
```

```
00387         SolutionToFile(nametofile, NUM_SEGMENTS);
00388         std::cout << "-COMMAND COMPLETED-" << std::endl;
00389         delete[] starts;
00390         delete[] ends;
00391     }
00392 }
00397 void GetResultFromFile(const char * namefromfile) { // из файла в консоль
00398     std::ifstream FromFile(namefromfile);
00399     if(FromFile.is_open()) {
00400         FromFile.getline(stream, STREAM_SIZE - 1);
00401         int NUM_SEGMENTS = GetNumberSegments();
00402         starts = new double[NUM_SEGMENTS];
00403         ends   = new double[NUM_SEGMENTS];
00404         for(int i = 0; i < NUM_SEGMENTS; i++) {
00405             FromFile.getline(stream, STREAM_SIZE - 1);
00406             ConvertToSegments(i);
00407         }
00408         SortSegments(NUM_SEGMENTS);
00409         SolutionToConsole(NUM_SEGMENTS);
00410         FromFile.close();
00411         delete[] starts;
00412         delete[] ends;
00413     }
00414     else {
00415         FromFile.close();
00416         delete[] starts;
00417         delete[] ends;
00418         std::cout << "ERROR: file for input not found, You must create it or
enter right!" << std::endl;
00419         exit(1);
00420     }
00421 }
00430 void GetResultFromFile(const char * namefromfile, const char * nametofile, int
is_name_file) { // из файла в файл
00431     std::ifstream FromFile(namefromfile);
00432     if(FromFile.is_open()) {
00433         FromFile.getline(stream, STREAM_SIZE - 1);
00434         int NUM_SEGMENTS = GetNumberSegments();
00435         starts = new double[NUM_SEGMENTS];
00436         ends   = new double[NUM_SEGMENTS];
00437         for(int i = 0; i < NUM_SEGMENTS; i++) {
00438             FromFile.getline(stream, STREAM_SIZE - 1);
00439             ConvertToSegments(i);
00440         }
00441         SortSegments(NUM_SEGMENTS);
00442 
00443         if(is_name_file) {
00444             std::ifstream check(nametofile);
00445             if(check.is_open()) {
00446                 check.close();
00447                 SolutionToFile(nametofile, NUM_SEGMENTS);
00448                 std::cout << "-COMMAND COMPLETED-" << std::endl;
00449                 delete[] starts;
00450                 delete[] ends;
00451             }
00452             else {
00453                 check.close();
00454                 delete[] starts;
00455                 delete[] ends;
00456                 std::cerr << "ERROR: file for output not found, You must create
it or enter right!" << std::endl;
00457                 exit(1);
00458             }
00459         }
00460         else {
00461             SolutionToFile(nametofile, NUM_SEGMENTS);
00462             std::cout << "-COMMAND COMPLETED-" << std::endl;
00463             delete[] starts;
00464             delete[] ends;
00465         }
00466     }
00467     else {
00468         FromFile.close();
00469         delete[] starts;
00470         delete[] ends;
00471         std::cerr << "ERROR: file for input not found, You must create it or
enter right!" << std::endl;
```

```
00472         exit(1);
00473     }
00474 }
```

Файл functions.h

Заголовочный файл, в котором объявлены функции, использующиеся в файле **main.cpp**

Функции

void **Help** ()

Help() - *функция для вывода инструкции по работе с программой*

void **GetResultFromConsole** ()

GetResultFromConsole() - *функция, осуществляющая ввод данных пользователем с консоли и вывод также в консоль*

void **GetResultFromConsole** (const char *nametofile, int is_name_file)

GetResultFromConsole() - *перегруженная функция, осуществляющая ввод данных пользователем с консоли и вывод в файл*

void **GetResultFromFile** (const char *namefromfile)

GetResultFromFile() - *функция, осуществляющая ввод данных пользователем с файла и вывод в консоль*

void **GetResultFromFile** (const char *namefromfile, const char *nametofile, int is_name_file)

GetResultFromFile() - *перегруженная функция, осуществляющая ввод данных пользователем с файла и вывод в файл*

Функции

void GetResultFromConsole ()

GetResultFromConsole() - функция, осуществляющая ввод данных пользователем с консоли и вывод также в консоль

См. определение в файле **functions.cpp** строка **337**

void GetResultFromConsole (const char * *nametofile*, int *is_name_file*)

GetResultFromConsole() - перегруженная функция, осуществляющая ввод данных пользователем с консоли и вывод в файл

Аргументы

nametofile	Название файла, в который будет записан результат выполнения функции
is_name_file	Флаг, обозначающий, ввел ли пользователь название файла или нет

Заметки

Если is_name_file = 1, значит пользователь выбрал конкретный файл, в который будет записан результат выполения функции. Если is_name_file = 0, значит пользователь не вводил название файла ,запись будет осуществленная в файл DefaultFile.txt по умолчанию

См. определение в файле **functions.cpp** строка **358**

void GetResultFromFile (const char * *namefromfile*)

GetResultFromFile() - функция, осуществляющая ввод данных пользователем с файла и вывод в консоль

Аргументы

namefromfile	Название файла, из которого будет производиться чтение данных

См. определение в файле **functions.cpp** строка **397**

void GetResultFromFile (const char * *namefromfile*, const char * *nametofile*, int *is_name_file*)

GetResultFromFile() - перегруженная функция, осуществляющая ввод данных пользователем с файла и вывод в файл

Аргументы

namefromfile	Названия файла, с которого будет производиться чтение данных
nametofile	Название файла, в который будет производиться запись данных
is_name_file	Флаг, обозначающий, ввел ли пользователь название файла для вывода данных или нет

Заметки

Если is_name_file = 1, значит пользователь выбрал конкретный файл, в который будет записан результат выполения функции. Если is_name_file = 0, значит пользователь не вводил название файла ,запись будет осуществленная в файл DefaultFile.txt по умолчанию

См. определение в файле **functions.cpp** строка **430**

void Help ()

Help() - функция для вывода инструкции по работе с программой

См. определение в файле **functions.cpp** строка **28**

functions.h

См. документацию.

```
00001
00002 #ifndef FUNCTIONS
00003 #define FUNCTIONS
00004 void Help();
00005 void GetResultFromConsole();
00006 void GetResultFromConsole(const char * nametofile, int is_name_file);
00007 void GetResultFromFile(const char * namefromfile);
00008 void GetResultFromFile(const char * namefromfile, const char * nametofile, int
is_name_file);
00009 #endif
```

Файл main.cpp

Главный файл, в котором реализован ввод флагов и названий файлов пользователем
Включенные заголовочные файлы:

```
#include <iostream>
#include <cstring>
#include "functions.h"
```

Функции

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

main() *- главная функция, принимающая на вход введенные флаги, названия файлов пользователем, и вызывающая на основании этого функции, обьявленные в заголовочном файле* ***functions.h***

Подробное описание

Главный файл, в котором реализован ввод флагов и названий файлов пользователем

См. определение в файле **main.cpp**

Функции

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

main() - главная функция, принимающая на вход введенные флаги, названия файлов пользователем, и вызывающая на основании этого функции, обьявленные в заголовочном файле **functions.h**

Аргументы

argc	Количество параметров, передаваемых в функцию
argv	Массив этих самых параметров

Возвращает

0 - программа выполнена успешно

Константный массив с флагами, ввод которых ожидается от пользователя.

```
const char * flags[5] = {"--help", "--default", "--tofile", "--fromfile",
"DefaultFile.txt"};
```

Если был введен один параметр (название обьектного файла), то выведется ошибка:

Ошибка:

ERROR: flag not found!

```
if(argc == 1) {
  std::cerr << "ERROR: flag not found!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

Если было введено два параметра, то ожидается, что вторым будет введен один из следующий флагов: –help, –default, –tofile.

```
else if(argc == 2) {...}
```

Если пользователь ввёл флаг –help, то с помощью функции **Help()** будет выведена инструкция по использованию программы.

Заметки

Сравнение введенного параметра с конкретным флагом, который хранится в массиве flags, осуществляется с помощью функции strcmp(), которая посимвольно сравнивает введенные в нее параметры, и возвращает 0, если параметры полностью совпадает, 1 - в противном случае. Поэтому мы берем отрицание возвращаемого значения.

```
if(!strcmp(argv[1], flags[0])) {
    Help();
}
```

Если пользователь ввёл флаг –default, то будет выведено сообщение с просьбой ввести Количество отрезков, а затем и сами отрезки. Далее вызывается функция **GetResultFromConsole()**, после чего результат выполения программы выведется на консоль.

```
else if(!strcmp(argv[1], flags[1])) {
  std::cout << "Please enter the number of segments and then the segments:" <<
std::endl;
  GetResultFromConsole();
}
```

Если пользователь ввёл флаг –tofile, то будет выведено сообщение с просьбой ввести Количество отрезков, а затем и сами отрезки. Далее вызывется функция **GetResultFromConsole()** с введенными в нее параметрами: название файла по умолчанию, а также 0 - флаг, показывающий, что было введено название дефолтного файла. Результат будет записан в дефолтный файл.

```
else if(!strcmp(argv[1], flags[2])) {
  std::cout << "Please enter the number of segments and then the segments:" <<
std::endl;
  GetResultFromConsole(flags[4], 0);
}
```

Если пользователь ввел флаг –fromfile, то выведется ошибка:

Ошибка:

ERROR: You must enter the file name for flag '–fromfile'!

```
else if (!strcmp(argv[1], flags[3])) {
  std::cerr << "ERROR: You must enter the file name for flag '--fromfile'!" <<
std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

Были рассмотрены все допустимые исходы при таком количестве введеных параметров, если ничего из вышеперечисленного не сработало, будет выведена ошибка:

Ошибка:

ERROR: invalid input!

```
else {
  std::cerr << "ERROR: invalid input!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
```

```
  exit(1);
}
```

Если пользователь ввёл три параметра, то вторым параметром ожидаются флаги: –fromfile, –tofile; а третьим - название файла.

```
else if(argc == 3) {...}
```

Если пользователь ввёл флаг –tofile, то выведется сообщение с просьбой ввести количество отрезков, а затем и сами отрезки. Далее вызывется перегруженная функция **GetResultFromFile()**, с введенными в нее параметрами: название файла для записи результата выполнения функции, "1" - флаг, означающий, что пользователь выбрал конкретный файл для записи результата.

```
if(!strcmp(argv[1], flags[2])) {
  std::cout << "Please enter the number of segments and then the segments:" <<
std::endl;
  GetResultFromConsole(argv[2], 1);
}
```

Если пользователь ввёл флаг –fromfile, то вызывется функция **GetResultFromFile()**, с введенным в нее параметром: название файла для чтения данных.

```
else if(!strcmp(argv[1], flags[3])) {
  GetResultFromFile(argv[2]);
}
```

Если пользователь ввел флаг –help или –default, то выведется ошибка:

Ошибка:

ERROR: this flag doesn't need another argument!

```
else if(!strcmp(argv[1], flags[0]) || !strcmp(argv[1], flags[1])) {
  std::cerr << "ERROR: this flag doesn't need another argument!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

Были рассмотрены все допустимые исходы при таком количестве введеных параметров, если ничего из вышеперечисленного не сработало, будет выведена ошибка, описанная выше:

```
else {
  std::cerr << "ERROR: invalid input!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

Если пользователь ввёл 4 параметра, то вторым и четвертым ожидаются флаги –fromfile и –tofile соответственно.

```
else if(argc == 4) {...}
```

Если пользователь ввел вторым и четвертым параметрами флаги –fromfile и –tofile соответственно, то вызывется перегруженная функция **GetResultFromFile()**, с введенными в нее параметрами: название файла для чтения данных, название файла, выбранного по умолчанию "DefaultFile.txt", для записи результата выполнения функции, "0" - флаг, означающий, что пользователь не выбирал файл для записи результата.

```
if(!strcmp(argv[1], flags[3]) && !strcmp(argv[3], flags[2])) {
    GetResultFromFile(argv[2], flags[4], 0);
}
```

Были рассмотрены все допустимые исходы при таком количестве введеных параметров, если ничего из вышеперечисленного не сработало, будет выведена ошибка, описанная выше:

```
else {
  std::cerr << "ERROR: invalid input!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

Если пользователь ввел 5 параметров, то вторым и четвертым ожидаются флаги –fromfile и –tofile соответственно, а третьим и пятым - соответствующие названия файлов.

```
else if(argc == 5) {...}
```

Если пользователь ввел вторым и четвертым параметрами флаги –fromfile и –tofile соответственно, то вызывается перегруженная функция **GetResultFromFile()**, с введенными в нее параметрами: название файла, с которого будет произваодиться чтение данных, название файла, выбранного пользователем, в который будет производиться запись результата выполения программы, "1" - флаг, означающий, что пользователь выбрал конкретный файл для записи результата.

```
if(!strcmp(argv[1], flags[3]) && !strcmp(argv[3], flags[2])) {
  GetResultFromFile(argv[2], argv[4], 1);
}
```

Были рассмотрены все допустимые исходы при таком количестве введеных параметров, если ничего из вышеперечисленного не сработало, будет выведена ошибка, описанная выше:

```
else {
  std::cerr << "ERROR: invalid input!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

Были рассмотрены все допустимые исходы при таком количестве введеных параметров, если ничего из вышеперечисленного не сработало, будет выведена ошибка, описанная выше:

```
else {
  std::cerr << "ERROR: invalid input!" << std::endl;
  std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
  exit(1);
}
```

См. определение в файле **main.cpp** строка **14**

main.cpp

См. документацию.

```
00001
00005 #include <iostream>
00006 #include <cstring>
00007 #include "functions.h"
00014 int main(int argc, char ** argv) {
00017     const char * flags[5] = {"--help", "--default", "--tofile", "--fromfile",
"DefaultFile.txt"};
00028     if(argc == 1) {
00029         std::cerr << "ERROR: flag not found!"                          <<
std::endl;
00030         std::cerr << "-You can enter '--help' for instruction output-" <<
std::endl;
00031         exit(1);
00032     }
00036     else if(argc == 2) {
00047         if(!strcmp(argv[1], flags[0])) {
00048             Help();
00049         }
00059         else if(!strcmp(argv[1], flags[1])) {
00060             std::cout << "Please enter the number of segments and then the
segments:" << std::endl;
00061             GetResultFromConsole();
00062         }
00073         else if(!strcmp(argv[1], flags[2])) {
00074             std::cout << "Please enter the number of segments and then the
segments:" << std::endl;
00075             GetResultFromConsole(flags[4], 0);
00076         }
00087         else if (!strcmp(argv[1], flags[3])) {
00088             std::cerr << "ERROR: You must enter the file name for flag '--
fromfile'!" << std::endl;
00089             std::cerr << "-You can enter '--help' for instruction output-"
<< std::endl;
00090             exit(1);
00091         }
00103         else {
00104             std::cerr << "ERROR: invalid input!"                          <<
std::endl;
00105             std::cerr << "-You can enter '--help' for instruction output-" <<
std::endl;
00106             exit(1);
00107         }
00108     }
00112     else if(argc == 3) {
00123         if(!strcmp(argv[1], flags[2])) {
00124             std::cout << "Please enter the number of segments and then the
segments:" << std::endl;
00125             GetResultFromConsole(argv[2], 1);
00126         }
00134         else if(!strcmp(argv[1], flags[3])) {
00135             GetResultFromFile(argv[2]);
00136         }
00147         else if(!strcmp(argv[1], flags[0]) || !strcmp(argv[1], flags[1])) {
00148             std::cerr << "ERROR: this flag doesn't need another argument!" <<
std::endl;
00149             std::cerr << "-You can enter '--help' for instruction output-" <<
std::endl;
00150             exit(1);
00151         }
00162         else {
00163             std::cerr << "ERROR: invalid input!"                          <<
std::endl;
00164             std::cerr << "-You can enter '--help' for instruction output-" <<
std::endl;
00165             exit(1);
00166         }
00167     }
00171     else if(argc == 4) {
00181         if(!strcmp(argv[1], flags[3]) && !strcmp(argv[3], flags[2])) {
00182             GetResultFromFile(argv[2], flags[4], 0);
00183         }
00194         else {
```

```
00195             std::cerr << "ERROR: invalid input!"                          <<
std::endl;
00196             std::cerr << "-You can enter '--help' for instruction output-" <<
std::endl;
00197             exit(1);
00198         }
00199     }
00204     else if(argc == 5) {
00216         if(!strcmp(argv[1], flags[3]) && !strcmp(argv[3], flags[2])) {
00217             GetResultFromFile(argv[2], argv[4], 1);
00218         }
00229         else {
00230             std::cerr << "ERROR: invalid input!" << std::endl;
00231             std::cerr << "-You can enter '--help' for instruction output-" <<
std::endl;
00232             exit(1);
00233         }
00234     }
00245     else {
00246     std::cerr << "ERROR: invalid input!" << std::endl;
00247     std::cerr << "-You can enter '--help' for instruction output-" << std::endl;
00248     exit(1);
00249     }
00250     return 0;
00251 }
```