Отчёт. Лабораторная работа №3.

Программа, поделена на три файла:

- файл lab5.cpp
- файл func.cpp
- файл lab5.h

Файл lab5.cpp

Подключены заголовочные файлы для организации ввода-вывода (<iostream>, <fstream>), а также заголовочный файл lab5.h, содержащий объявления функций из func.cpp и глобальные переменные для func.cpp и lab5.cpp

В lab5.cpp содержится функция int main() с параметрами argc и argv[]. При запуске пользователь может ввести флаги --tofile, --fromfile, а также указать файл, с которого будут считываться данные, и файл, куда будет записан полученные результат. Предусмотрен некорректный ввод флагов, а именно:

- чрезмерное количество флагов

```
else if (argc > 5) {
	std::cout << "ERROR! Too many flags" << std::endl;
	return -1;
}
```

- некорректное имя флага

```
else {
	std::cout << "ERROR! Flags are not correct" << std::endl;
	return -1;
	 }
```

В случае ошибки программа завершается и на экран выводится фраза с объяснением ошибки.

Если все флаги корректны или флаги вовсе не были введены, запускается функция allright(), на вход получающая параметр argv.

Файл func.cpp

Функции, реализующие необходимые действия для решения задачи вынесены в отдельный файл.

int allright(char* argv[])

Основная функция в этом файле – функция int allright(char* argv[]), суть которой в считывании отрезков и их обработки. Отрезки могут считываться как с командной строки, так и с файла.

```
if (flag_read) // значение flag_read присваивается в main() в зависимости от введенных
                                                                                флагов
{
    fin >> b >> e;
 }
else { std::cin >> b >> e; }
```

Рассматриваются несколько возможных случая при считывании данных.

1. Данные некорректны, то есть e<b, где e – конец отрезка, b – начало отрезка

```
if (e <= b) {
                std::cout << "ERROR! Data isn't correct" << std::endl;
                return 0;
 }
```

2. Программа только запустилась и данные еще не были считаны, поэтому e и b просто добавляются в массив (по факту, там два массива, но далее я буду говорить о этих двух массивах, как об одном), содержащие соответственно данные о конце отрезков и о начале.

```
if (k == 0) {
        bm[k] = b;
        em[k] = e;
        ++k;} // k счётчик элементов в массиве
```

3. В массив уже добавлены данные о 10000 отрезках, то есть больше записать невозможно и поэтому дописать невозможно.

```
if (k>=T) {
        std::cout << "ERROR! Too many values" << std::endl;
        return 0;}
```

4. Начало отрезка левее всех отрезков.

```
if (temppoint == -1) { // temppoint = binf(b, bm, k) (binf - функция бинпоска)
        // ЗДЕСЬ ФУНКЦИЯ ПРОВЕРКИ ПЕРЕПОЛНЕНИЯ МАССИВА
        temppoint = 0;
                for (int j = k - 1; j >= temppoint; j--) { //сдвиг вправо
                em[j + 1] = em[j];
                bm[j + 1] = bm[j];
        }
        em[temppoint] = e;
        bm[temppoint] = b;
        k++;
        check();} //эта функция будет описана позже
```

5. Отрезок правее всех или его начало совпадает с началом последнего отрезка

```
else if (temppoint == k) {
        if (b>em[k-1])
        {
                em[temppoint] = e;
                bm[temppoint] = b;
                ++k;
        }
        else {
                em[temppoint - 1] = (em[temppoint - 1] < e) ? e : em[temppoint - 1];
        }
 }
 else if (temppoint == k - 1) {
        em[temppoint] = (em[temppoint] < e) ? e : em[temppoint];}
```

6. Отрезок внутри уже вписанных отрезков (между началом первого отрезка и концом второго)
    a. Если начала двух отрезков (введенного и находящегося в массиве) совпадают, то проверяется, не длиннее ли введенный отрезок отрезка из массива

```
if (bm[temppoint] == b) {
        if (em[temppoint] < e) {
                em[temppoint] = e;
```

```
                check(); //эта функция будет описана позже
        }
}
```

b. В бинпоиске ищется элемент равный или меньший чем введенное значение, поэтому я сделала еще проверку, что найденный элемент меньше, чем введенный. Если при этом em[temppoint]<b, в массив добавляется новый отрезок, в ином случае обновляется отрезок с индексом temppoint и выполняется функция check().

```
else if (bm[temppoint] < b) {
        if (em[temppoint] >= b) {
                if (em[temppoint] < e) {
                        em[temppoint] = e;
                        check();
                }
        }
        else {
                if (k >= T) {
                        std::cout << "ERROR! Too many values" << std::endl;
                        return 0;
                }
                else {
                       for (int j = k - 1; j >= temppoint; j--) {//сдвиг вправо
                                em[j + 1] = em[j];
                                bm[j + 1] = bm[j];
                        }
                temppoint = binf(b, bm, k); //эта функция будет описана позже
                        if (temppoint<T)
                                {
                                em[temppoint] = e;
                                bm[temppoint] = b;
                                ++k;
                        }
                        check();
                }
        }
}
```

И после выполнения программы в зависимости от введенного в начале флага про вывод данных полученные отрезки выводятся либо в файл, либо в консоль.

```
std::cout << ' ' << std::endl;
for (int i = 0; i < k; i++) {
        if (flag_write == 1) { fout << bm[i] << ' ' << em[i] << std::endl; }
        else {
              std::cout << bm[i] << ' ' << em[i] << std::endl; }
 }
```

int check()

Данная функция проверят, чтобы после добавления в массив отрезка не было пересечения отрезков. Она выполняется до тех пор, пока начало отрезка, следующего после добавленного, не станет больше конца добавленного или добавленные массив не станет последним в последовательности отрезков. Реализует сдвиг влево.

```
tp = temppoint + 1; //Индекс отрезка, следующего за введенным
 while (bm[tp] <= em[temppoint] && bm[tp] != END) {
       em[temppoint] = em[temppoint] < em[tp] ? em[tp] : em[temppoint];
       for (int j = tp; j < k - 1; j++) {
             em[j] = em[j + 1];
             bm[j] = bm[j + 1];
       }
       em[k - 1] = END; //END - константа для обозначения конца заполненного массива
       bm[k - 1] = END;
       --k;
 }
```

int binf(float x, float* a, int k)

Данная функция на вход получает отсортированный массив, значение, которое надо найти в этом массиве, и длину массива (имеется в виду заполненность массива). С помощью этого бинпоиска можно найти индекс элемента массива, равного или меньшего данного значения.

```
int l = -1;
int r = k + 1;
int mid;
while (r - l > 1) {
       mid = (int)(l + r) / 2;
       if (a[mid] == x) {
             return mid;
       }
       if (a[mid] > x) r = mid;
       else l = mid;
} //функция возвращает значение левой границы
```

Файл lab5.h

```
#ifndef LAB5
#define LAB5
int binf(float x, float* a, int k);
int allright(char* argv[]);
int check();
extern int flag_write;
extern int flag_read;
#endif
```

В заголовочном файле перечислены функции, используемые в func.cpp и lab5.cpp, а также глобальные переменные flag_write, flag_read, указывающие на необходимость записывания данных в файл и считывания из файла соответственно. Также реализована защита от переопределения.

Makefile

Мейк-файл создан для компиляции программы. Реализованы методы clean и distclean.

```
TR = lab5
CC = g++
OB = func.o lab5.o
$(TR) : $(OB)
       $(CC) func.o lab5.o -o $(TR)

func.o : func.cpp
       $(CC) -c func.cpp -o func.o

lab5.o : lab5.cpp
       $(CC) -c lab5.cpp -o lab5.o

clean :
       rm $(OB)
distclean : clean
       rm $(TR)
```

clean удаляет файлы с расширением .o, a distclean удаляет все файлы.