Отчёт.

- В начале программы идёт разделение на один из 5 случаев ввода различных флагов, в переменной flag запоминается какие флаги были введены, какие нет. При неправильном вводе выдаёт ошибку и завершает выполнение программы.
- В данном фрагменте кода происходит считывание разными способами (в зависимости от флагов, либо из консоли, либо из файла) количества отрезков

```
int main(int argc, char* argv[]) {
    int n, flag = 0;
    if(argc == 1) {
        std::cin >> n;
        flag = 1;
    } else if (argc == 3 && strcmp("--fromfile", argv[1])== 0){
        std::ifstream file (argv[2]);
        file >> n;
        flag = 2;
    } else if (argc == 5 && strcmp("--fromfile", argv[3])== 0 && strcmp("--
tofile", argv[1])== 0) {
        std::ifstream file(argv[4]);
        file >> n;
        flag = 3;
    } else if ((argc == 3 && strcmp("--tofile", argv[1])== 0)) {
        std::cin >> n;
        flag = 4;
    } else if (argc == 5 && strcmp("--fromfile", argv[1])== 0 && strcmp("--
tofile", argv[3])== 0){
        std::ifstream file (argv[2]);
        file >> n;
        flag = 5;
    } else {
        std::cerr << "Please enter --fromfile or --tofile and the way to
them. \nFor example: --fromfile 1.txt --tofile 2.txt" << std::endl;
        exit(0);
    }
```

- Создание динамического двумерного массива, длина которого равна количеству отрезков, в первой строке хранится начало отрезков, во второй концы отрезков. Также заполнение массива нулями для отлавливания ошибок далее.

```
float** a = new float* [n];
for(int i = 0; i < n; i++)
    a[i] = new float [2];
for(int i = 0; i < n;i++){
    a[i][0] = 0;
    a[i][1] = 0;
}
```

- Заполнение двумерного массива различными способами в зависимости от заданных флагов (либо ввод с консоли, либо считывание из файла)
- Также в каждом случае стоит проверка на ошибки, в случае которой будет выведено сообщение об ошибке и прекращено выполнение программы
- При считывании из файла, после заполнения массива, идет закрытие файла

```
if(flag == 1 || flag == 4) {
    for (int i = 0; i < n; i++) {
        std::cin >> a[i][0] >> a[i][1];
        if (a[i][0] >= a[i][1]) {
            std::cerr << "Error! Cut off incorrectly" << std::endl;
            exit(0);
        }
    }
} else if (flag == 2 || flag == 5){
    std::ifstream file (argv[2]);
    file >> n;
    for (int i = 0; i < n; i++) {
        file >> a[i][0] >> a[i][1];
        if (a[i][0] >= a[i][1]) {
            std::cerr << "Error! Cut off incorrectly" << std::endl;
            exit(0);
        }
    }
    file.close();
} else {
    std::ifstream file(argv[4]);
    file >> n;
    for (int i = 0; i < n; i++) {
        file >> a[i][0] >> a[i][1];
        if (a[i][0] >= a[i][1]) {
            std::cerr << "Error! Cut off incorrectly" << std::endl;
            exit(0);
        }
    }
    file.close();
}
```

- Сортировка двумерного массива пузырьковым методом по началам отрезков от меньшего к большему

```
for(int i = 0 ; i < n;i++){
    for(int j = 0; j < n - 1; j++){
        if (a[j][0] > a[j+1][0]){
            t1 = a[j][0];
            t2 = a[j][1];
            a[j][0] = a[j+1][0];
            a[j][1] = a[j+1][1];
            a[j+1][0] = t1;
            a[j+1][1] = t2;
        }
```

```
    }
}
```

- Проход по массиву в поисках пересечений, count – отвечает за количество найденных объединений, curCount – отвечает за количество отрезков в объединении, оно нужно для того, чтобы отсеивать единичные отрезки. curBegin – начало текущего обрабатываемого объединения, curEnd – конец текущего обрабатываемого объединения.
- Смысл такой: мы проходим по массиву, сравнивая конец текущего обрабатываемого объединения и начало следующего отрезка, если он больше (т.е. не входит в объединение) то записываем найденное объединение в массив с индексом count. Если же это не так, то в curEnd записываем максимальное значение конца из текущего отрезка и curCount.

```
int count = 0, curCount = 0;
float curBegin = a[0][0], curEnd= a[0][1];
for(int i = 0; i < n-1; i++){
    if(a[i+1][0] > curEnd){
        a[count][0] = curBegin;
        a[count][1] = curEnd;
        count++;
        curBegin = a[i+1][0];
        curEnd = a[i+1][1];
        if(curCount == 0){
            count--;
        }
        curCount = 0;
    } else {
        curEnd = std::max(curEnd, a[i+1][1]);
        curCount++;
    }
}
```

- После цикла обрабатываем последний подходящий отрезок

```
a[count][0] = curBegin;
a[count][1] = curEnd;
count++;
if(curCount == 0){
    count--;
}
```

- В зависимости от флагов происходит вывод всех найденных пересечений, либо в консоль, либо в файл.
- После записывания в файл, происходит закрытие файла.

```
if(flag == 1 || flag == 2) {
    std::cout << std::endl;
    if(count == 0){
        std::cout << "NOTHING FOUND" << std::endl;
    } else {
        for (int i = 0; i < count; i++) {
            std::cout << a[i][0] << " " << a[i][1] << std::endl;
        }
    }
} else if (flag == 3 || flag == 4){
    std::ofstream file(argv[2]);
    if(count == 0){
        file << "NOTHING FOUND" << std::endl;
    } else {
        for (int i = 0; i < count; i++) {
            file << a[i][0] << " " << a[i][1] << std::endl;
        }
    }
    file.close();
} else {
    std::ofstream file(argv[4]);
    if(count == 0){
        file << "NOTHING FOUND" << std::endl;
    } else {
        for (int i = 0; i < count; i++) {
            file << a[i][0] << " " << a[i][1] << std::endl;
        }
    }
    file.close();
}
```

- Удаление двумерного массива

```
for (int i = 0; i < n;i++)
    delete [] a[i];
delete [] a;
```