Лабораторная работа №5

Flags.cpp

В данном файле используются аргументы argc и argv для ввода и считывания флагов, которые указывают вывод результата в файл и ввод из файла, --tofile и --fromfile соответственно. Задаём две переменные логического типа данных:

```
bool tofile=false;
bool fromfile=false;
```

Затем с помощью операторов if-else и else if задаём условие, которое проверяет количество аргументов в командной строке и осуществляет лексикографическую проверку строк. Если условие выполняется, то флаги приравниваем к true и выполняется функция func(argv):

```
if (argc==3 && !strcmp(argv[1],"--tofile"))
   {
      tofile=true;
      func(argv);
   }
   else if (argc==3 && !strcmp(argv[1],"--fromfile"))
   {
      fromfile=true;
      func(argv);
   }
   else if (argc==5 && !strcmp(argv[1],"--fromfile") && !strcmp(argv[3],"--tofile"))
   {
      tofile=true;
      fromfile=true;
      func(argv);
```

Header.h

В заголовочном файле содержатся объявления функции, которая определяет объединение отрезков, и флагов с ключевым словом **extern**, которое позволяет использовать их в файле otrezki.cpp.

otrezki.cpp

В данном файле осуществляется функция, которая определяет объединение отрезков. Для начала объявляем переменную n, которая отвечает за количество отрезков, и двумерный массив (список одномерных массивов), который отвечает за значения концов отрезков. Чтобы найти объединение надо отсортировать массив, для этого используется пузырьковая сортировка. Сортировать будем по первому элементу в массиве. Принцип сортировки таков: мы проходимся по всему массиву и сравниваем между собой соседнии ячейки, если значение ячейки i<i-1, то меняем местами отрезки. Задаем цикл, который проходится по i (начало отрезка=первый элемент в массиве), если начало первого отрезка больше начала второго отрезка, то мы меняем местами эти отрезки:

```
   bool b=true;
   while(b)
   {
      b=false;
      for(int i=1;i<n;i++)
      {
```

```
            if (arr[i][0]<arr[i-1][0])
            {
                double p=arr[i][0];
                double p2=arr[i][1];
                arr[i][0]=arr[i-1][0];
                arr[i][1]=arr[i-1][1];
                arr[i-1][0]=p;
                arr[i-1][1]=p2;
                b=true;
            }
        }
    }
```

Как только прошла сортировка по возрастанию, переходим к определению объединения данных отрезков. Сначала объявим, что начало первого отрезка равно переменной nach с типом данных с плавающей точкой, а конец первого отрезка равно переменной kon с тем же типом данных. Если начало второго отрезка < конца первого отрезка, то конец пересекаемого отрезка будет максимум из конца второго отрезка и конца первого, а начало будет совпадать с началом первого отрезка. Если отрезков больше нет, то программа выводит объединение двух этих отрезков, а если происходит разрыв между отрезками, то программа выводит объединение двух отрезков и обновляет начало и конец отрезка, то есть переходит к следующему.

```
    for(int i=1;i<n;i++){
        if (arr[i][0]<kon)
            {
                kon=std::max(arr[i][1],kon);
                nichego=false;
                flag=false;
            }
        else
        {
            if (flag!=true && nichego==false){
                if (tofile==true)
                    fout<<nach<<" "<<kon<<" "<<std::endl;
                else if (fromfile==true)
                    std::cout<<nach<<" "<<kon<<" "<<std::endl;
                nach=arr[i][0];
                kon=arr[i][1];
            }
```

Последний отрезок из объединения записываем вне цикла, поскольку цикл не обрабатывает последнюю строку:

```
         if (flag==false)
    {
        if (tofile==true)
            fout<<nach<<" "<<kon<<std::endl; //не обрабат последняя строку пишем
вручную
        else if (fromfile==true)
            std::cout<<nach<<" "<<kon<<std::endl;
        return 0;
    }
```

Также программа осуществляет проверку на единичные отрезки и разрыв после ещё одного разрыва (т.е. когда между отрезками имеется 2 разрыва, например, 1 2 3 4 5 6). Для этого понадобятся флаги: `bool flag=false; //прыжок`

```
bool nichego=true; // проверка единичных отрезков
```
Если программа находит хотя бы одно объединение из двух отрезков, то флаг nichego приравнивается к false. Если же объединение не нашлось, то:
```
 if (nichego==true)
{
   std::cout<<"Nothing was found";
   return 1;
}
```
Если происходит разрыв после ещё одного разрыва, то программа переходит к следующему отрезку, обновляет концы отрезков и приравнивает flag к true. Если flag==true, то программа не записывает единичный отрезок в объединение.
```
            if (arr[i][1]<arr[i+1][0] && i!=n-1)
      {
         i++;
         nach=arr[i][0];
         kon=arr[i][1];
         flag=true;
      }
```

Работа с –tofile и –fromfile (файловый ввод-вывод)

Подключаем заголовочный файл <fstream>. Создаём объекты, привязанные к файлам:
```
std::ofstream fout;
std::ifstream fin;
```
Если в flags.cpp tofile=true, то программа открывает для записи файл, заданный пользователем, и записывает вывод в этот файл:
```
if (tofile==true && fromfile==false){
      fout.open(argv[2]);
        //...
       if (tofile==true)
      fout<<nach<<" "<<kon<<" "<<std::endl;
        //...
```
Чтобы закрыть файл, используем метод close():
```
fout.close();
```
Если в flags.cpp fromfile=true, то программа открывает для чтения файл, заданный пользователем, и записывает вывод в консоль:
```
else if (fromfile==true && tofile==false){
      fin.open(argv[2]);
      fin>>n;
      for(int i=0;i<n;i++)
      {
         fin>>arr[i][0];
         fin>>arr[i][1];
      }
   }
        //...

else if (fromfile==true)
       std::cout<<nach<<" "<<kon<<std::endl;
           return 0;
```

Makefile

Makefile собирает программный проект и очищает от временных файлов. Distclean - удалит не только файлы, которые удаляются при исполнении цели 'clean', но также и файлы 'TAGS', 'Makefile' и 'config.status'.