COP 3502C Programming Assignment # 4
Sorting
Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will use **Merge sort**

What should you submit?
Write all the code in a single file and upload the main.c file Please include the following commented lines in the beginning of your code to declare your authorship of the code:

/* COP 3502C Assignment 4
This program is written by: Your Full Name */

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.
Caution!!!
Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:
See the deadline in Webcourses. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

What to do if you need clarification on the problem?
Write an email to the TA and put the course teacher in the cc for clarification on the requirements. I will also create a discussion thread in webcourses, and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question as you. Also, other students can reply, and you might get your answer faster.

How to get help if you are stuck?
According to the course policy, all the helps should be taken during office hours. There Occasionally, we might reply in email.

Problem: Projector

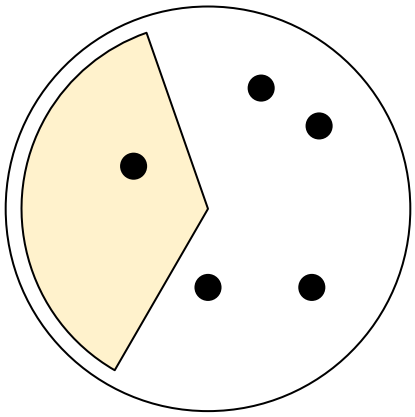
Objective

Give practice with **merge sort in C.**
Practice with compareTo function strategy to compare objects in C while sorting
Practice with floating point comparison strategy in C

Story

Before starting to read the assignment, you can probably watch the recording posted in the announcement where I have briefly discussed the requirements and some hints.

Your movie theater is trying out a new projector. The projection will be in the center of a circular room and will project a movie onto a part of the wall. The screen will project a sector of light at some angle width which comes at some default value, but can be updated based on a setting. The projector can be rotated such that the projection starts at any angle in the room.



However, you have several locations that groups of people have decided to stand at in the room. These groups of people are relatively small compared to the distance they are from the projector and can be treated as points with 0 radius. Although the projection won't be disturbed by our infinitely thin customers, the light emitted from the projector will be harmful to the people in the group.

You want to answer 2 main questions.

1. What is the largest possible angle of projection that can be used such that no group will be standing in the projection, thus will not harm anyone?
2. Find all the ways you can project this largest angle and show the list of pairs of groups that will be closest to the edge of those projections. It means, if there are multiple ways to get that largest angle, you need to list all the pairs of groups that are closest to the projection edge.

Problem

Given the locations and number of people in the groups, determine the largest angle that can be used such that no one is in the projection. Additionally, find all the ways you can project this largest angle and show the list of pairs of groups will be closest to the edge of the projection.

Input (no file i/o. Only standard input)

Input starts with a line containing 2 integers, ***N*** and ***A***, ($1 \le N \le 500{,}000$; $1 \le A \le 359$) representing the number of groups standing in the room and the angle in degrees at which the screen projector initially projects (However, we will change this default setting).

The following N lines will each contain 3 space separated integers, ***x***, ***y***, and ***s***, ($0 \le |x|, |y| \le 1{,}000{,}000$; $1 \le s \le 1{,}000$), representing x and y coordinates respectively of the location of a group and the number of people in that given group. Assume that the projector will be located at location (0, 0). Also, you can consider they are entered in a sequence of group numbers starting from 0 to ending at N-1. It means the first x, y, and s represent the data for group 0, the second one is group 1 and so on. It is also guaranteed that that the position of multiple groups will not be exactly in the same angle in reference to the x-axis. For example, if a group's position is (1,1), then there will not be a group at position (2,2) as both (1,1) and (2,2) are at 45 degree angle from the x-axis.

Note since the customers are so small multiple customers could stand at the same location.

Output

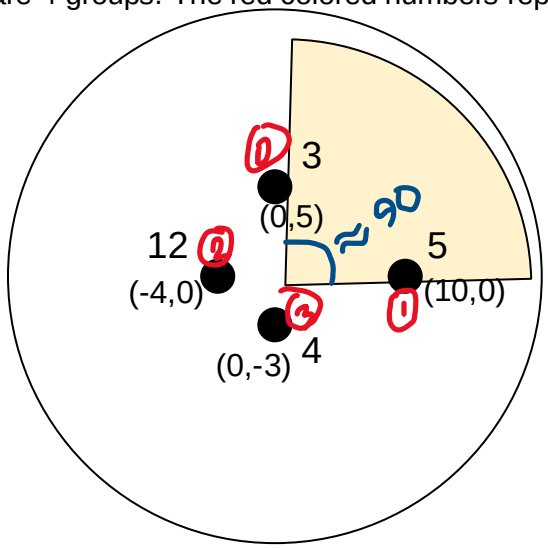
The output should consist of two items:
The first line will contain a floating-point value that specifies the maximum angle in degrees that can be used such that no person will be in the projection rounded to 4 digits after the decimal. Assuming, there are P different ways that maximum angle can be projected, print a list of P pairs, one per line containing the group numbers that will be closest to the projection edge. While printing a pair, the first group number must be smaller than the second group number. Also, the list of pairs must be printed as a sorted list. If multiple pairs starts with same group number, then it must be sorted based on the second group number in the pair.

Sample Input	**Sample Output**
4 91 0 5 3 10 0 5 -4 0 12 0 -3 4	Max projection degree without harming 90.0000 Closest possible group pairs in order: 0 1 0 2 1 3 2 3
3 181 1 1 8 -2 1 5 2 10 10	Max projection degree without harming 251.5651 Closest possible group pairs in order: 0 1

Explanation

Case 1

In the first test case there are 4 groups. The red colored numbers represent the group numbers.



You can use an angle slightly less than 90 degrees to project with no group in the projection. See the example image where these 90 degrees is achieved between group 0 and 1. You can get this 90 degrees in 4 different ways. Between group 0 and 2, between 2 and 3, and between 1 and 3. Also, if you observe the output, it is in ordered based on the group number of the pair. If there is a tie in the first number of the pair, it is sorted base on the second number.

Example:

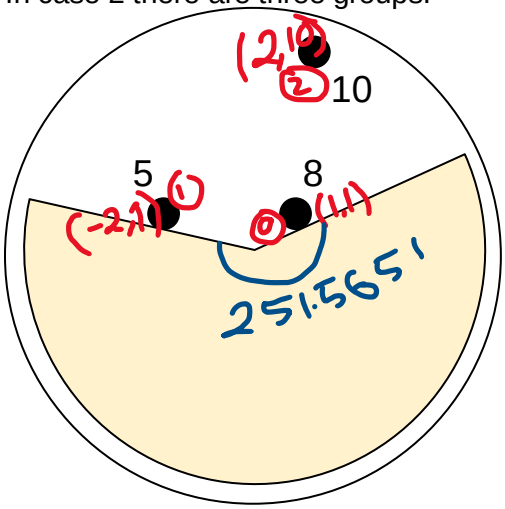
0 1

0 2

(as the first item is zero for both of them, it is ordered by the second item for those cases)

Case 2

In case 2 there are three groups.



The angle can be quite large without projecting on anybody. We can get this angle keeping group pair 0 and 1 close the edge.

Some implementation restrictions and hints:

- You must create two structs, one for group and one for Result. The group struct can contain group size, angle (in radian), and the group number. The result struct contains a pair (two int firstGroup and secondGroup), and angle (in degree). You can modify the content of the struct. But keeping the mentioned structure fields will help you.
- You must use dynamic memory allocation for both all the arrays. As the size of the result array is not known initially, you can allocate it based on the number of groups and use tracker to track how many of them are used.
- Convert each position of a group into radian angle. Explore the tan2 function of math.h library to calculate this. If you need to review what is radian, maybe watch this YouTube video: https://www.youtube.com/watch?v=Aqm8v0fJDzM
- At various places in your code, you may need to convert radian to degree. During that conversion use M_PI constant from math.h library to get a more precise value of π
  The formula would be, *angle in degree = angle in radian * 180 / π*
- **You must sort the array of groups based on their angle (in radian) using merge sort algorithm discussed in the class. Due to a large value of number of groups, the default mid index calculation formula (left+right)/2 may over flow. So, use the following formula during mid index calculation (*mid index = left + (right – left)/2*)**
- After sorting the groups based on angle, you can now iterate through them to find the max angle as part of the answer.
- Based on the answer, you can now build the result array with all the pair based on the sorted groups and their angle. Make sure the group 1 and group 2 of a result struct value is ordered.
    - As you will store only the pairs that has an angle equal to the max angle you found earlier, there is possibility of error in equality comparison as they are floating point numbers
    - For example. See the issue in the following piece of code:

```
double d1 = 13+0.2+0.2+0.1;       /*  13.5 ? */
double d2 = 14-0.2-0.2-0.1;       /*  13.5 ? */
if(d1==d2) {
    printf("They're the same %lf %lf\n",d1,d2);
}
else {
    printf("They're not the same %lf %lf\n",d1,d2);
}
Output:
They're not the same 13.500000 13.500000
```

    - To deal with this problem, we have to use a threshold (epsilon). If the difference of two floating point numbers are within the threshold, we consider them as equal. For example, if a and b are two floating point numbers, we can consider that they are same if the absolute value of their difference is less then epsilon. For this assignment use 0.0001 as the value for epsilon.
        - `fabs(a-b) < epsilon`
    - write a function *doubleCompare(double d1, double d2, double eps)* that returns 1 if d1 and d2 are same based on the epsilon as discussed above. Otherwise, it should

return 0. You **must** use this function whenever you need to compare two double type of data in your assignment for checking whether they are same or not.

- After obtaining the array of results, you must sort the results using merge sort *algorithm (you may need to write another merge sort code for this)*.
    - As during this sorting process, you will need to compare two results based the firstGroup and secondGroup property of the result struct, you must implement and use the following compareTo function.
    - You must write a function compareTo which takes in two pointers, ptrPt1 and ptrPt2, to Result structs and returns a negative integer if the Result pointed by ptrPt1 is smaller than the Result pointed by ptrPt2. Otherwise, it should return a positive number indicating the Result pointed by ptrPt1 is greater than the Result pointed by ptrPt2. As part of this comparison, you should compare the firstGroup property between them fist and if there is a tie, you should compare the secondGroup to make the decision. If both values are exactly same, you should return 0.
- You must free all the memory properly to receive full credit
- The code must compile and work on codegrade platform
- Also, make sure to test your code with more test cases to make sure your code will pass other test cases as well.

Tentative Rubric (Subject to change)

- A code not compiling or creating seg fault without producing any result can get zero. There may or may not be any partial credit. But at least they will not get more than 35% even if it is for a small reason.
- There is no point for well structured, commented and well indented code. ***There will be a deduction of 10% points if the code is not well indented and 5% for not commenting important blocks.***
- There is no grade for just writing the required functions and structs. However, there will be 20% penalty for not writing and using CompareTo() function, 10% penalty for not writing and using *doubleCompare*()function,
- Reading data and loading them into group struct array properly with proper angle calculation: 10 pts
- Sorting the groups based on angle properly using merge sort: 10 pts
- Comparing adjacent sorted groups to find the max angle properly: 10 pts
- Making the Results array properly with the pair and angles: 10 pts
- Soring the Results using merge sort properly: 10 pts
- Programs will be tested on 10 cases
    - 5 points each
    - There may be no ***partial credit for an incorrect case. The output format must match to receive credit for an output.***

Some Steps (if needed) to check your output AUTOMATICALLY in a command line in repl.it or other compiler:

You can run the following commands to check whether your output is exactly matching with the sample output or not.

Step1: Copy the sample output to sample_out.txt file and move it to the server

Step2: compile your code using typical gcc and other commands.

//if you use math.h library, use the -lm option with the gcc command. Also, note that scanf function returns a value depending on the number of inputs. If you do not use the returned value of the scanf, gcc command may show warning to all of the scanf. In that case you can use "-Wno-unused-result" option with the gcc command to ignore those warning. So the command for compiling your code would be:

 # gcc main.c leak_detector_c.c -Wno-unused-result -lm

Step3: Execute your code and pass the sample input file as a input and generate the output into another file with the following command

$ ***./a.out < sample_in.txt > out.txt***

Step4: Run the following command to compare your out.txt file with the sample output file

```
$cmp out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the first mismatched byte with the line number.

Step4(Alternative): Run the following command to compare your out.txt file with the sample output file

```
$diff -y out.txt sample_out.txt
```

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the all the mismatches with more details compared to cmp command.

 # diff -c myout1.txt sample_out1.txt //this command will show ! symbol to the unmatched lines.