CS69011: Computing Lab-1
Test-1
August 02, 2023

========== Instructions ==========

1. All Questions carry equal marks.
2. In case of user-input assume only valid values will be passed as input.
3. In Q2, Q3, and Q4 we have specifically mentioned the type of data structure to be used and the algorithm to be implemented. Additionally, we have also suggested an alternative data structure and algorithm to implement the question. Please note that using alternative data structure will fetch a penalty of 50%.
4. **Regarding Submission:** For each question create a separate C file. -> Q1.c, Q2.c, Q3.c, and Q4.c. Create a zip file of all these C files in the name <RollNo>_T1.zip and submit it to moodle.

==========

Q1. You can define a point in two-dimensional space using its x and y coordinate P(x,y). You can also define a circle in two-dimensional space using coordinates of a center and its radius : C[P(x,y), R].

1. Define a structure to store a point and a circle in 2-d space.
2. Given a point P(a,b) check its relation with the circle C[P(x,y),R].
    a. Lies inside, outside or on the circle.
    Hint: compute the value of euclidean distance between the center of the circle and the point -> d.
    If d < R : point is inside circle
    If d=R: point is on the circle
    If d > R: point is outside the circle.
3. Given two circles $C_1[P(x_1,y_1),R_1]$ and $C_2[P(x_2,y_2),R_2]$ find the relationship among the circles.
    a. C_1 lies inside C_2 or vice-versa, or [if euclidean distance between centers < difference of radius]
    b. C_1 and C_2 intersect, or [if euclidean distance of centers < sum of radius]
    c. C_1 and C_2 touch, or [if euclidean distance of centers == sum of radius]
    d. C_1 and C_2 are disjoint [if euclidean distance of centers > sum of radius]

[Test Cases]

For Part 2.:

```
Input a point:
X-coordinate: 2
Y-coordinate: 3

Input point is: P(2,3)

Input the details of circle:
X-coordinate of center: 2
Y-coordinate of center: 5
Radius: 3

Input Circle is: C[P[(2,5),3]
Relation of point with circle is: Point lies inside the circle.
```

For Part 3:

```
Input the details of circle C1:
X-coordinate of center: 2
Y-coordinate of center: 5
```

Radius: 3

Input Circle C1 is: C[P[(2,5),3]

Input the details of circle C2:
X-coordinate of center: 1
Y-coordinate of center: 1
Radius: 2

Input Circle C2 is: C[P[(1,1),2]
The relation between two circles is: C1 and C2 intersect\
++

Q2. Assume we have a single user and a multi-resource setting in a punch card scenario—a user will run multiple jobs given in a setting (the setting is given as input via punch cards) and each job can use exactly one resource. In this scenario, assume that a user can include *at most* 10 jobs in a setting. Naturally, for each job, the user has to specify the unique id of the resource the job will use and the estimated execution time of the job.

For eg: here is what a punch card containing seven jobs will look like:
(Resource id) (Execution Time)

1	5
2	4
3	6
1	8
6	9
7	2
1	6

Now in this environment, a simple scheduler would be a program that just prioritizes the scheduling of job(s) based on the execution time (lower executing time means higher priority). For our example punch card above, The output of this scheduler would be a job execution plan which would look as follows:
[7 2] -> [2 4] -> [1 5] -> [1 6] -> [3 6] -> [1 8] -> [6 9]

Your task is to write a C-program that partially implements the working of scheduler as follows:

1. The program reads the punch card given in a file ‘resourse.txt’, which contains resource-id and execution time (one jon in every line).
2. Store the rows of the file in a 2-d array.
3. Sort this 2-d array in increasing order based on the resource consumption time. In case of a tie in execution time, pick the least resource-id job.
4. Print the 2-d array in a file: ‘Plan.txt’ and also on the terminal. The output should be in the format of the job execution plan as shown above.

If you are not familiar with handling 2-d arrays, use two one-dimensional arrays (one each for resources and execution time). Please make sure to sort these two arrays accordingly.

++

Q3. In a doubly-linked list, we define an element as a local minima if it is minimum to all its N left and N right neighbors (if available). Given a sequence of numbers as user input, enter the numbers in a doubly-linked list. Find all such minima from the list and put them in an array. Subsequently output the largest minima. N is a user given input to the program.

If you face difficulty in implementing a doubly-link list, try implementing the same using an array (will fetch only 50% marks).

[Test Case]

Enter the number [Enter -1 to stop]: 10
Enter the number [Enter -1 to stop]: 2
Enter the number [Enter -1 to stop]: 4
Enter the number [Enter -1 to stop]: 7
Enter the number [Enter -1 to stop]: 6
Enter the number [Enter -1 to stop]: 3
Enter the number [Enter -1 to stop]: 11
Enter the number [Enter -1 to stop]: -1

The entered linked list is: 10->2->4->7->6->3->11->NULL
Enter the value of N (size of neighborhood) : 2

The minimas stored in the arrays are: 2, 3
The maximum of these minimas are: 3

Explanation:
For every element in linked list we have to check its two left and two right neighbors [since N=2]

For 10: The two right neighbors are 2,4 and no left neighbors: It is not a local minima
For 2:The two right neighbors are 4,7 and 10 as the only left neighbor: It is a local minima —store it in array
For 4: The two right neighbors are 7,6 and two left neighbors are 10,2: It is not a local minima.
For 7: The two right neighbors are 6,3 and two left neighbors are 2,4: It is not a local minima.
For 6: Its two right neighbors are 3,11 and two left neighbors are 7,4: It is not a local minima.
For 3: The only right neighbor is 11 and two left neighbors are 6,7: It is a local minima —store it in array
For 11: There is no right neighbor, its two left neighbors are 6,3: It is not a local minima.

So we have two local minima: 2,3 and the maximum is 3.

++

Q4. Implement Quick Sort. Write three functions. Partition, QSort, Main.

```
// Partition the array using an element as the pivot
int partition(...){
}

// The main function that implements QuickSort
void QSort(...){
}
```

In case you cannot implement Quick Sort, try implementing Bubble Sort (this would fetch 50% marks)

++

==========