CS19003: Programming & Data Structure Lab, Section 15
Autumn 2020-21
Lab Test 2, February 18, 2021

PART - 1

Time: 9-10 am to 10-30 am

Instructions (Read carefully)

1. **Your C programs must first contain comments with your name, roll no., and Labtest no. (=2), as done in class.**
2. **Name your C file LT2_1_<your roll no>.c for (For example LT2_1_20ME30006.c)**
3. **Submit through the links (Intermediate and Final) for PART 1 in moodle. MAKE SURE TO VERIFY YOUR SUBMISSION after final submission.**
4. **(Very Important) There are two sets of problems. Problem set 1 is for 100% marks and covers the entire syllabus of what is covered in the lab. Problem set 2 is for 60% marks and covers only up to arrays and functions. You are free to choose which set you will attempt. However, there cannot be any mixing between the sets, you can answer only one of the sets. Submit only one C file for one set.**
5. **All other instructions regarding lab test sent earlier in the slides to be followed strictly.**

PROBLEM SET 1 (FOR 100% (= 10) MARKS)

An interval on the real line can be defined by two points on the line, i.e., by two floating point numbers, indicating the left and the right endpoint of the interval respectively. The interval is assumed to include the two endpoints that define it (i.e., the interval is closed), and all points between them. Two intervals are said to overlap if they include at least one common point (including any common endpoint). For example, the intervals [2.5, 3] overlaps with all of the intervals [2.7, 3.4], [1, 2.7], [2, 4], and [3, 4], but does not overlap with the interval [3.2, 5]. Note that we say [2.5, 3] overlaps with [3, 4] since they have a common endpoint (right endpoint of the first interval is same as left endpoint of the second).

Define a C structure named **INTERVALSET** that will hold **a set of intervals** on the real line. The structure will contain an integer **n** that will store the number of intervals, and two float pointers **L** and **R** that will be pointers to float arrays that will store the left and the right endpoints respectively of the **n** intervals. The **INTERVALSET** structure should be able to store any number of intervals, i.e., there is no upper bound given for the value of **n**.

Write a C function

void MaxOverlap (struct INTERVALSET P, …..)

that takes an interval set **P** as parameter. For each interval **x** in **P**, the function computes the number of other intervals in **P** that **x** overlaps with. The function returns the left and the right endpoints of the interval with the maximum number of overlaps, and the number of overlaps of this interval, through appropriate parameters that you have to add in the function prototype (you can add them any way as long as you can return these three values). If there are more than one interval with same maximum number of overlaps, you can return any one. You cannot change the return type of the function from **void**. Also, you cannot use any additional arrays.

Write a C function

DeleteInterval(…)

that will take an interval set **P** as parameter, and do the following. The function will delete all intervals that overlap with at least one other interval in **P** from the interval set (i.e., finally **P** will contain only intervals that overlapped with no other interval in **P** at the beginning before any deletion). Put in appropriate parameters and return values for the function prototype (you have to decide whether you want to take **P** directly as a parameter, or take a pointer to it as a parameter to do what you have to do in the function). For full marks in this part, you cannot use any additional array. If you cannot think of how to do it without using any additional array, you can use at most one additional array with a 30% penalty for this part.

Write a **main( )** function that does the following:

1. Define a variable to store an interval set **P**.
2. Read in **n**, the number of intervals to be stored in the interval set **P**, malloc an appropriate length array in **L** and **R**, and then read in the endpoints of the intervals one by one in **L** and **R**.
3. Print the endpoints of the intervals read nicely.
4. Call the function **MaxOverlap( )** to find and print the endpoints of the interval with the maximum number of overlaps and its number of overlaps.
5. Call the function **DeleteInterval( )** to delete intervals from **P** as described above. Print the intervals in the interval set **P** after returning from the function.

Example:

n = 7
Set of intervals: [2.1, 4], [5.1, 6], [2.5, 5.5], [9.2, 11.5], [7.2, 8.1], [12, 13], [12.5, 13.5]

Your program should output (exact messages can vary):

The intervals are [2.1, 4], [5.1, 6], [2.5, 5.5], [9.2, 11.5], [7.2, 8.1], [12, 13], [12.5, 13.5]
The interval with the max overlap is [2.5, 5.5], with 2 overlaps
The intervals after deletion are [7.2, 8.1], [9.2, 11.5]

<u>PROBLEM SET 2 (FOR 60% (= 6) MARKS)</u>

Consider a set of **n** points in a 2-d plane, with the x-coordinates of the points stored in an array **X** and the y-coordinates of the points stored in an array **Y** (so coordinates of the first point are (**X**[0], **Y**[0]), that of the second point are (**X**[1], **Y**[1]) and so on). Assume that the coordinates are all integers.

Write a C function

int CountNonHV(int X[ ], int Y[ ], int n)

that takes as parameters the **n** points with their coordinates in the arrays **X** and **Y** as described above, and returns the number of pairs of points with the property that the line segment joining the two points in the pair is neither parallel to x-axis nor parallel to y-axis (i.e. the number of pairs of points which differ in both x and y coordinates).

Write a C function

int FindArea (int X[ ], int Y[ ], int n)

that returns the area of the smallest axis-parallel rectangle that contains all the points. A rectangle is axis-parallel if its sides are parallel to the x and y-axis. A point is said to be contained by the rectangle if it lies inside or on one of the sides of the rectangle. Note that the corners of the smallest rectangle may or may not be any of the points given.

Write a C function

int MaxArea (int X[ ], int Y[ ], int n)

that returns the area of the largest triangle that can be formed with any 3 points in the set.

Write a **main( )** function that does the following:

1. Read in an integer **n** (< 100) .
2. Read in the coordinates of **n** points in two arrays **X** and **Y.**
3. Print the coordinates of the points read nicely.
4. Call the function **CountNonHV( )** to find and print the number of pairs of points with the property that the line segment joining them is neither parallel to x-axis nor parallel to y-axis. (Just print the number of such pairs, the actual pairs need not be printed)
5. Call the function **FindArea( )** to find and print the area of the smallest **<u>axis-parallel</u>** rectangle containing all the points.
6. Call the function **MaxArea( )** to find and print the area of the largest triangle that can be formed with any 3 points in the set.

Example:

n = 6
Set of points: (3, 5), (4, 5), (4, 7), (8, 7), (3, 7), (1, 6)

Your program should output (exact messages can vary):

The points are (3, 5), (4, 5), (4, 7), (8, 7), (3, 7), (1, 6)
The number of pairs is 9
The area of the smallest rectangle is 14
The area of the largest triangle is 5.0

Note that the actual 9 pairs are {(3, 5), (4,7)}, {(3, 5), (8,7)}, {(3, 5), (1,6)}, {(4, 5), (8,7)}, {(4, 5), (3,7)}, {(4, 5), (1,6)}, {(4, 7), (1,6)}, {(8, 7), (1,6)}, and {(3, 7), (1,6)}. But you need to keep track of **only the count,** the exact pairs are not needed.