COL216: Assignment 1

Devanshi Khatsuriya 2019CS10344 | Ishita Chaudhary 2019CS10360

Input format:

All inputs must be integers (as specified in the problem statement). The first line takes the input "n" as number of data points. The next "2n" lines alternatively takes x and y co-ordinate of the point. The input has x co-ordinates in non-decreasing order (the program raises an error in case this is not followed).

Design and Implementation:

Algorithm:

We calculate the total area as sum of areas bounded by each pair of consecutive points. For every pair of points (x1, y1) and (x2, y2), we check if the product of y co-ordinates of both points is positive or negative.

If it's non-negative (y1y2>=0), then the area is calculated using the formula for a trapezium, i.e.,

$$\frac{(x2 - x1)|y1 + y2|}{2}$$

Where x1<x2

Else, if it's negative (y1y2<0), the area is calculated as sum of areas of two triangles. The point of intersection with x-axis is calculated using equation of line with two given points and setting y=0 in the equation. This x co-ordinate obtained is put into the area expression and is simplified to:

$$(|y2|y2 - |y1|y1)\frac{(x2 - x1)}{2(y2 - y1)}$$

Where x1<x2

A "mainLoop" is made to repeat the above algorithm for "n-1" iterations. Number of iterations completed are updated after area calculation of every pair of data points.

After the first iteration, in each iteration, the pair of data points is updated to consist of the latter point of previous pair and the next point in input.

Design choices:

- Most of the registers are taken as double type instead of integer type or float type, for maximum accuracy.
- Various conditional control instructions have been used to do case-wise computations for each pair of points and also to jump across various parts of

the code. There are various labels like iter, pos, nega, errorOverflow, exit, etc. to facilitate this.

- Various prompts and error messages have been incorporated to facilitate the user.
- It is assumed that the inputs will be 32-bit integers each entered in a new line as the prompt suggests followed by pressing the enter key. The program may truncate the numbers that are not integers, may read only the first integer if more than one is entered in a line and may face overflow for integers larger than 32-bits, any of which may lead to misleading results.
- The program can handle the case if the input x coordinates are not in non-decreasing order. It prints message that conveys this error.
- If the input has points with same x coordinates, the area is bounded by them is considered 0. In such a case, the area between the last of such points and the next point (with different x coordinate) is considered instead of with any other points of the sequence.
- In both the cases (positive and negative product of y coordinates), the area calculated is not divided by 2 at each iteration. The final answer is divided by 2 in the end to minimize the number of instructions used.
- For area calculations, the product of 32-bit integers done using the mult instruction which stores the product as 64-bit integer in the $hi and $lo registers, which are then copied into two consecutives $t registers. After relevant calculations, they are also converted to double before adding them to the sum register of type double (sum register stores the twice the cumulative area).

Testing Strategy and Test Cases considered:

The program incorporates as well as handles all the cases, including border cases, effectively. The testing strategy followed is as below, with rigorous test cases of each type.

Test Case 1: Negative values of “n” would be invalid for the given program. Also, if n=0 or 1, i.e., area under no point or one point would still be invalid. Hence, the program prints an error message when n<2.

Test Case 2: This test will have input x coordinates not in non-decreasing order, n=4 and data points (2, 4) (3, -3) (-1, 0) (3, 4). This prints an error message to the console.

Test Case 3: This test would be the example given in the problem statement, with n=5 and data points (1, 1) (3, 4) (5, 3) (6, 7) (9, 5). This would check our positive calculation block of program. The area comes out to be correct, i.e., 35.

Test Case 4: This test would be the example given in the problem statement, but with a sign change in all the y co-ordinates, i.e., n=5 and data points (1, -1) (3, -4) (5, -3) (6, -7) (9, -5). This would check our positive calculation block of program which should output the same result as above. The area comes out to be correct, i.e., 35.

Test Case 5: This case checks the negative calculation block of program. Take n=4 and data points (0, 1) (2, -1) (3, 1) (6, -2). The area comes out to be 4 and not -1.5, which is correct.

Test Case 6: This test would be a bigger input which will test both the positive and negative calculation blocks of the program. Taking n=6 and data points as (0, 2) (2, -2) (6, 6) (8, 8) (10, -16) (12, 0), we get area to be 55.33, which is correct.

Test Case 7: This test case checks for inputs with fractional area. Taking n=5 and data points as (4, 4) (5, 2) (10, -12) (12, 6) (15, 0) gives area 42.428, which is correct.

Test Case 8: This test case verifies the program for large areas. Taking n=10 and data points as (0, 100) (20, 50) (25, 500) (65, -400) (70, 1000) (500, 5000) (600, 10000) (10000, 12000) (20000, 20000) (50000) (50000), we get an area of 131545057.539.

Test Case 9: This test case checks data overflow. If the area >> 1, such that the registers cannot handle the given input values of x and y, the program raises an error of data overflow. For example, n=2 with (0,1) (999999999999,1). Here a message saying an overflow error has occurred will be displayed.