###### Information and Computer Science Department

**ICS 103: Computer Programming in C**

**Second Semester 2018-2019 (182)  
Homework No. 1  
[Posted: Thursday 31 January 2019]  
[Due Date: Monday 11 February 2019 @ 11:59 PM (Before Midnight)]**

**Submission Guidelines:**

Submit a zipped file containing the following files:

* HW1.c (C source file) containing your answer to the programming question.

**PLEASE DO NOT INCLUDE .EXE FILES IN YOUR SUBMISSION**

The zipped file should be named as follows:

**HW\_1\_XXXXXXXXX\_YourFamilyName\_Lecture\_Section\_No.zip**

where:

XXXXXXXXX is your 9 digit KFUPM ID.

YourFamilyName is your family name

Lecture\_Section\_No is the number of your ICS 103 lecture section

Submission should be made through your ICS 103 Lecture section Blackboard course page under **HW\_1 Assignment** submission link.

**Important Notes:**

* Submission of the homework solution should be in a zipped filed with the format specified above. **Any different formatting/naming will result in reducing the total homework score by half!**
* **Submitting exe file only without the source file leads to 0 grade**
* **Cheating is taken seriously**. Any cheating attempt will result in 0 for this homework and the remaining ones.
* **EACH STUDENT IS REQUIRED TO DO THE HOMEWORK ALONE**. COPYING FROM ANY SOURCE IS REGARDED AS CHEATING.
* **No late submissions are allowed**.
* **Submissions via email are not accepted and will be simply ignored**.
* **You must use proper indentation and meaningful variable names in your programs.**

**Problem Statement**

The *line intersection problem* is a very fundamental problem in game programming. For example, in order to decide if a car is going to hit a wall in a game, the intersection point between the two lines representing the wall and path of motion for the car must be found. This process is referred to as collision detection. In this homework, you are going to write a program for computing the intersection between two finite-length line segments in two-dimensional (2D) space. To test for intersection between two lines, there are five cases:

1. The two line segments are collinear and overlapping,
2. The two line segments are collinear but not overlapping,
3. The two line segments are parallel,
4. The two line segments intersect at one point, and
5. The two line segments do not intersect.

Write a program that reads the start and end points of two line segments and print the intersection point, if it exists. The x- and y-coordinates of all the four points are positive (positive quadrant).

**Background**

P2 = (x2, y2)

P4 = (x4, y4)

P1 = (x1, y1)

P3 = (x3, y3)

L1

L2

* The two line segments lie in the positive quadrant. Therefore, the x- and y-coordinates of their end points are all positive. In the example above, the two line segments L1 and L2 can mathematically characterized using the following two parametric equations, respectively:
* L1(x, y) = (x1, y1) + t\*(x2 – x1, y2 – y1)
* L2(x, y) = (x3, y3) + u\*(x4 – x3, y4 – y3)

where . The points P1(x1, y1) and P2(x2, y2) are the end points for line 1 (L1). The points P3(x3, y3) and P4(x4, y4) are the end points for line 2 (L2).

* A 2D vector is defined by two points. For example, a vector **v** whose tail is at the origin and its head at point (5, 7) is represented as follows:

y

* + **v** = (5, 7)

(5, 7)

(0, 0)

x

* In this specific case (2D space), all vectors exist on the same plane. Hence, a vector *v* is defined by its x- and y-coordinates and cross-product can mathematically computed as follows:
* v1 x v2 = v1.x \* v2.y – v1.y \* v2.x

Where V1.x represents the x coordinate of vector V1 and V1.y represents the y coordinate of vector V1. Similar meaning for V2.x and V2.y.

* Vector multiplication is defined as follows:
* v1 \* v2 = v1.x \* v2.x + v1.y \* v2.y
* The values of t and u in the parametric equations of the line segments can be computed as follows:
* t = q x s / r x s
* u = q x r / r x s

where the operator *x* indicates the cross-product operation as defined above. **q**, **s**, and **r** are auxiliary vectors whose x- and y-coordinates are computed as follows:

* r = (x2 – x1, y2 – y1) represents the vector P1P2
* s = (x4 – x3, y4 – y3) represents the vector P3P4
* q = (x3 – x1, y3 – y1) represents the vector P3P1

**Algorithm**

* + Read x1, y1, x2, y2
  + Read x3, y3, x4, y4
  + Compute vectors **r**, **s**, and **q**
  + Compute cross-products **q**x**s, q**x**r**, and **r**x**s**
  + Compute auxiliary values: **q**\***r**, **r**\***r**, **q**\***s**, and **s**\***s**
  + If **r**x**s** is zero and **q**x**r** is zero, then
    - The two lines are collinear
      * If 0 <= **q**\***r** <= **r**\***r** or 0 <= **q**\***s** <= **s**\***s**
        + The two lines are overlapping
      * else
        + The two lines are disjoint
  + If **r**x**s** is zero and **q**x**r** is not zero
    - The two lines are parallel and not intersecting
  + If **r**x**s** is not zero , compute t and u
    - If 0 <= t <= 1 and 0 <= u <= 1
      * The two lines intersect at P1+t\***r**
    - else
      * The two line segments do not intersect

**Sample Runs: For simplicity of clarification, the input values used in the sample runs below are integer. But in your program, all coordinates should be double.**

|  |  |  |
| --- | --- | --- |
| **Case** | **I/O** | **Illustration** |
| 1 | **> prog.exe**  Enter the end points of the first line segment (x1, y1) followed by (x2, y2):  1 2 7 2  Enter the end points of the second line segment (x3, y3) followed by (x4, y4):  3 2 5 2  => The two lines are collinear and overlapping => Intersecting | L1: {(1,2), (7,2)}  L2: {(3,2), (5,2)}  (1, 2)  (3, 2)  (5, 2)  (7, 2) |
| 2 | **> prog.exe**  Enter the end points of the first line segment (x1, y1) followed by (x2, y2):  1 2 3 2  Enter the end points of the second line segment (x3, y3) followed by (x4, y4):  5 2 7 2  => The two lines are collinear and disjoint => Not Intersecting | L1: {(1,2), (3,2)}  L2: {(5,2), (7,2)}  (1, 2)  (3, 2)  (5, 2)  (7, 2) |
| 3 | **> prog.exe**  Enter the end points of the first line segment (x1, y1) followed by (x2, y2):  1 4 3 4  Enter the end points of the second line segment (x3, y3) followed by (x4, y4):  1 2 3 2  => The two lines are parallel => Not Intersecting | L1: {(1,4), (3,4)}  L2: {(1,2), (3,2)}  (3, 2)  (1, 2)  (4, 4)  (1, 4) |
| 4 | **> prog.exe**  Enter the end points of the first line segment (x1, y1) followed by (x2, y2):  0 0 4 5  Enter the end points of the second line segment (x3, y3) followed by (x4, y4):  1 5 4 2  => The two lines intersect at (2.67, 3.33) | L1: {(1,5), (4,2)}  L2: {(0,0), (4,5)}  (4, 5)  (1, 5)  (4, 2)  (0, 0) |
| 5 | **> prog.exe**  Enter the end points of the first line segment (x1, y1) followed by (x2, y2):  2 2 4 4  Enter the end points of the second line segment (x3, y3) followed by (x4, y4):  6 7 9 2  => The two lines do not intersect | L1: {(2,2), (4,4)}  L2: {(6,7), (9,2)}  (9, 2)  (6, 7)  (4, 4)  (2, 2) |