TND012/2019: Dugga 3

Attention

This is an **individual** home dugga. Thus, any form of cooperation with other students (e.g. LiTHehack) is seen as academic dishonesty.

Books, notes, and internet material can be used.

If you find any open questions in the presented exercise then feel free to make your own decisions. However, your decisions should be reasonable and must not contradict any of the conditions explicitly written for the exercise. Please, write comments in the program that clarify your assumptions/decisions, if any.

Remember that you should not ignore compiler warnings, since they are usually an indication of serious problems in the code. Remember to set the warning level of your compiler. For Visual Studio, you can follow the instructions given in this file. The file also shows how to set Visual Studio compiler to compile C++17.

Remember also to include in your program all the libraries used in your code (e.g. `#include <cmath>`).

Course goals

This is an assessment of whether you can

- structure a program with the help of functions and
- define new data types.

All the points above have been discussed in lectures 1 to 10 and lessons 1 to 8.

Requirements for grade G

- Readable and well indented code.
- Use of good programming practices.
- Global variables cannot be used.
- Respect for submission instructions.
- Code copied from a web page should be clearly indicated through a commented line containing the web page address used as source, with exception for the course material posted on the course website (e.g. code posted for lectures).
- Use of statements that are not part of the ISO C++ leads to automatically failing the dugga.
- Your programs must pass all test examples shown in this paper.
- Other criteria are listed in the feedback report document available from course website.

Note that there is no guarantee that your code is correct just because it passes all given test examples. Thus, you should always test your code with extra examples, you come up with.

Deadline

14 of October, 12:00.

Submission instructions

1. Submit **one** source code (`.cpp`) file for the exercise through Lisam. The file must be named with your LiU id (e.g. `ninek007.cpp`).
2. The submitted source code file must have your name and personal number at the top of the file.
3. The submitted source code (`.cpp`) file cannot be compressed (e.g. neither `.zip` nor `.rar` files are accepted).

Remember that you should deliver only the source code (`.cpp`) file. Moreover, answers to the dugga exercises sent by email are ignored.

Duggor solutions submitted not accordingly the submission instructions are ignored.

Questions

The aim of the dugga is that you solve the exercise without help of other people. However, we give you the possibility to send us questions about the problem in this dugga by email. If you are a MT student then you should email Aida Nordman (`Aida.Vitoria@liu.se`). If you are an ED or KTS student then you should email Martin Falk (`Martin.Falk@liu.se`).

Only emails received from 17:00 on Friday to 12:00 on Saturday will be answered. Emails received after 12:00 on Saturday are not answered. The emails will be answered until Saturday at 20:00.

Be brief and concrete. Emails can be written either in Swedish or English.

Note that you should not send emails related to Lisam system.

User support for Lisam

Help and support for Lisam system is available at helpdesk@student.liu.se and by calling the phone number 013-28 58 98.

Login and password to access the course material

All course material, like lecture slides and code examples, are available through the course website (**`http://weber.itn.liu.se/~aidvi05/courses/10/index.html`**). However, the material is password protected. Use the following login and password to access the material.

login: **`TND012`** password: **`TND012ht2_12`**

Lycka till!!

Exercise 1

Write a C++ program that starts by requesting a straight line L and a list of straight lines. The program should then indicate which lines in the list are parallel with line L and which intersect line L.

A straight line L is defined by the equation $y = mx + b$, where values m and b are real numbers (**`double`**s in C++). Note that vertical lines are not considered in this exercise because the slop m would then be infinite.

Consider two lines[1]: $L_1 : y = m_1x + b_1$ and $L_2 : y = m_2x + b_2$.

- Lines L_1 and L_2 are parallel, if they have the same slope, i.e. $m_1 = m_2$.
- Line L_1 intersects line L_2 at point (p_x, p_y), if $p_y = m_1 \times p_x + b_1$ and $p_y = m_2 \times p_x + b_2$. Note that non-parallel lines always intersect each other at one point.

Your program should perform the following steps, by the indicated order.

1. Request to the user to enter a straight line L. The user should enter two values for m and b.
2. Request to the user the number of lines n in the list of lines, at most 100. The number of lines in the list should be validated and if it is not correct then the program should request the user to re-enter n again.
3. Read n straight lines, entered by the user.
4. List all lines in the user given list that are parallel with line L, if any. Otherwise, the message "*No parallel lines found!!*" should be displayed.
5. List all lines in the user given list which intersect line L, if any. For each line that intersects line L, the program should also display the intersection point between the lines. If there are no lines that intersect L then the message "*No intersection lines found!!*" should be displayed.

All values should be displayed with two digits after the decimal point.

Your program must satisfy the following coding requirements.

- Define a new data type `Line` to represent a straight line. Obviously, `Line` should also be used in the program.
- The program must be structured with the help of small functions.
- The `main` function must not contain any loops.
- There must be a separate function, named `display_line`, that writes a given line L to `cout`. The equation of the line must be displayed as $y = mx + b$, if $b \geq 0$. Otherwise, the line must be displayed as $y = mx - b$.

[1] This webpage has easy to read information about intersection of two straight lines and how to find the intersection point.

- There must be a separate function, named `is_parallel`, that tests whether two given lines are parallel to each other. This function should not do any keyboard input (i.e. read from `cin`) nor screen output (i.e. write to `cout`).
- There must be a separate function, named `find_intersection_point`, that returns the intersection point between two given lines. This function should not do any keyboard input nor screen output.

Note that you decide which arguments the functions need and what they should return, without going against the given instructions. Feel free to add any other functions, besides the ones requested above.

Assume that the user always enters numeric values as input to the program.

Three examples are given below. Values in green color are entered by the user.

Example 1

```
Enter a line L (m followed by b): 3 -3  ◄··················  y = 3x − 3
Enter number of lines: 5
Enter 5 lines:
2.3 4    ◄··················  y = 2.3x + 4
3 10     ◄··················  y = 3x + 10
12 8.8
0 12
3 2.8

Lines parallel with line y = 3.00x - 3.00
y = 3.00x + 10.00
y = 3.00x + 2.80

Lines that intersect with line y = 3.00x - 3.00
y = 2.30x + 4.00        Intersection point: (10.00, 27.00)
y = 12.00x + 8.80       Intersection point: (-1.31, -6.93)
y = 0.00x + 12.00       Intersection point: (5.00, 12.00)
```

Example 2

```
Enter a line L (m followed by b): 8 19
Enter number of lines: -1
Invalid number of lines!!

Enter number of lines: 1
Enter 1 lines:
8 -2.2

Lines parallel with line y = 8.00x + 19.00
y = 8.00x - 2.20

Lines that intersect with line y = 8.00x + 19.00
No intersection lines found!!
```

Example 3

```
Enter a line L (m followed by b): -3.5 -6.5
Enter number of lines: 6
Enter 6 lines:
1 1
-1 -1
1 -1
-1 1
0 6.6
0 -6.6

Lines parallel with line y = -3.50x - 6.50
No parallel lines found!!

Lines that intersect with line y = -3.50x - 6.50
y = 1.00x + 1.00        Intersection point: (-1.67, -0.67)
y = -1.00x - 1.00       Intersection point: (-2.20, 1.20)
y = 1.00x - 1.00        Intersection point: (-1.22, -2.22)
y = -1.00x + 1.00       Intersection point: (-3.00, 4.00)
y = 0.00x + 6.60        Intersection point: (-3.74, 6.60)
y = 0.00x - 6.60        Intersection point: (0.03, -6.60)
```