CSE 102 Programming Assignment 1

DUE

October 20, 2022, 23:55

Description

- This is an individual assignment. Please do not collaborate
- If you think that this document does not clearly describe the assignment, ask questions before its too late.

This is a `C` Programming assignment. You will write a `C` program according to the following description.

- Your program catches user input, learns and tests a simple classifier. This assignment is about using control statements, performing arithmetic operations and simple input/output.
- You are expected to divide the solution into blocks and use functions.

Program Flow

- Read pre-defined number of points for `class 1`.
- Read pre-defined number of points for `class 2`.
- Find the `average point` for each class.
- Find the line connecting the two `average points`.
- Find the midpoint of the connecting line.
- Find the perpendicular `separating line` which passes through the midpoint.
- Classify any given point and print its label until an unusual input is encountered.

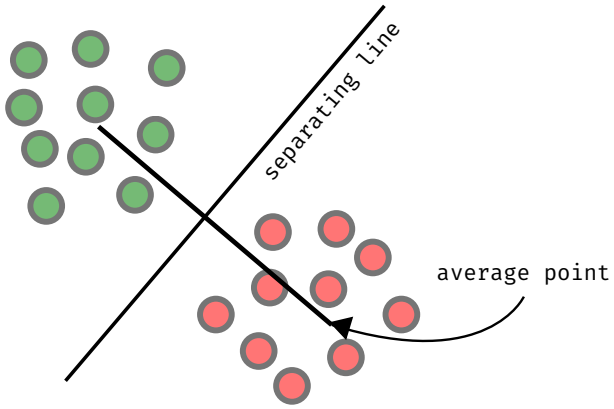

Figure 1: Visualization

Example

- **line numbers and comments will not be printed.** Only the things after `>` sign is seen on terminal window.

```
#         commands                  comments

line 0  > 10.4 34.8                 User enters point coordinates for class 1
line 1  > 3.5 45.9
        > .
        > .
        > .
line 10 > 22.2 12.6                 User enters point coordinates for class 2
line 11 > 62.1 11.0
        > .
        > .
```

```
        > .
line 20 > 10.2 34.9                 User enters a test point
line 21 > class 1                   Program prints its class
line 22 > 10.3 33.8                 User enters a test point
line 23 > class 1                   Program prints its class
        > .
        > .
        > .
line x > q                          User enters something not expected
                                    Program quits.
```

Remarks

- There will be 20 training samples (10 for each class). But, don't make this number hard-coded. Use a macro. Changing the macro should be enough for trying different number of samples. You can assume that the number of training samples for each class will be the same.
- **There is no limit on the number of test points.** In order to stop the infinite loop, you have to check the status returned by `scanf()`. If the status indicates an unexpected input case, your program quits.
- Assume that none of the points will be on the separating line.
- Assume training phase will be error-free.
- Be careful with `divide-by-zero` situations. You can avoid them by perturbing the `0` results with very small numbers. (if the divider is `0`, make it `0 + EPSILON`. Define a macro `EPSILON` and make it a small floating point number. (ex: `0.001`))
- **Do not print anything other than the expected output.**
- Your program either prints `class 1` or `class 2`. Each label should be on a new line. There shouldn't be any empty lines.
- You cannot use pointers and other things which are not covered in class.
- Using input/output redirection is advised. (Do not submit any of the files you used for testing).
- Test your program. Hand-trace the execution. For this, you have to simplify it by decreasing the size of input. For example, instead of accepting 10 numbers for each class, accept 2-3 numbers. And, select numbers so that you can easily calculate the average by hand.

Turn in:

- Source code of a complete `C` program. Name of the file should be in this format: `<full_name>_PA1.c`.
- Example: `van_helsing_PA1.c`. Please do not use any Turkish special characters.
- You don't need to use an IDE for this assignment. Your code will be compiled and run in a command window.
- Your code will be compiled and tested on a Linux machine(Ubuntu). `GCC` will be used.
- Make sure you don't get compile errors when you issue this command : `gcc <full_name>_PA1.c`.
- A script will be used in order to check the correctness of your results. So, be careful not to violate the expected output format.
- Provide comments unless you are not interested in partial credit. (If I cannot easily understand your design, you may loose points.)
- You may not get full credit if your implementation contradicts with the statements in this document.

Late Submission

- Late submission is **NOT** accepted.

Grading (Tentative)

- `Max Grade` : `100`.

All of the followings are possible deductions from `Max Grade`.

- No submission: `-100`. (be consistent in doing this and your overall grade will converge to N/A)
- Compile errors: `-100`.
- Irrelevant code: `-100`.

- Major parts are missing: `-100`.
- Unnecessarily long code: `-30`.
- Using language elements and libraries which are not allowed: `-100`.
- Not caring about the structure and efficiency: `-30`. (avoid using hard-coded values).
- Significant number of compiler warnings: `-10`.
- Not commented enough: `-5`. (Comments are in English)
- Source code encoding is not `UTF-8` and characters are not properly displayed: `-5`. (You can use 'Visual Studio Code', 'Sublime Text', 'Atom' etc... Check the character encoding of your text editor and set it to `UTF-8`)
- Fails at properly catching user input: `-20`.
- Fails during training phase (program crashes): `-70`.
- Fails during test phase (program crashes): `-50`.
- Prints wrong class labels (the number of errors is not important): `-50`.
- Infinite loop: `-90`.
- Prints anything other than the expected: `-30`.
- Submission includes files other than the expected: `-10`.
- Submission does not follow the file naming convention: `-10`.
- Sharing or inheriting code: `-200`.

Note: Some of these items are not independent. So, you cannot expect isolation of many of them. For example, if you cannot catch user input correctly, your tests will fail. Partial grading is not guaranteed.