CCDSALG/GDDASGO Term 1, AY 2024 – 2025
Project 1 – Convex Hull (Application of Stack Data Structure & Sorting Algorithms)

Groupings	: At most 3 members in a group
Deadline	: Refer to the Canvas Submission Page
Percentage	: 20% of the Final Grade
Programming Language:	C Language

INTRODUCTION

Let's apply what you learned in stack data structures and sorting algorithms.

For this project, you will implement a program that will compute the **convex hull** of a given set of 2D points using the **Graham's Scan algorithm**. The original algorithm was developed by Ronald L. Graham, and it was published in "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set," *Information Processing Letters*, **1**(4), 1972 pp. 132–133. doi:10.1016/0020-0190(72)90045-2).

Please see also the following online resources:

- https://youtu.be/B2AJoQSZf4M?si=X2BOmHcTCRJYNcm-
- Graham scan - Wikipedia

There are several implementations of said Graham's scan algorithm that you can find on the internet, for example,

- Convex hull - Rosetta Code
- Convex Hull using Graham Scan - GeeksforGeeks

There is no problem if you read, study and learn from these publicly available codes. However, please take note that for this MCO1, you are **NOT** allowed to copy a Graham's Scan algorithm source code written by another person. Non-compliance will result in cheating punishable with a final grade of 0 for this course. You'll have to implement this on your own (i.e., with your groupmates) using the C programming language as specified in the list of **TASKS** described in **Section III** below.

I. INPUT

The input is a text file containing the following data. The 1st line encodes *n* which represents the input/problem size, i.e., the number of points. Thereafter, it is followed by *n* lines of data where each line contains two **double** data type values representing the *x* and *y* coordinates of the points. One or more blank spaces separate the *x* and *y* coordinates. The coordinates are NOT sorted in any manner. Refer to the accompanying example input text files named **`sample-input.txt`** (which contains collinear points) and **`input-circle.txt`** (where all points are actually points on a circle that has a radius of 2, and whose centroid is at the origin 0, 0.)

STRICT REQUIREMENT #1: make sure that your program will be able to store and process a maximum problem size of n = 2^15 = 32768 elements.

II. OUTPUT

The output is another text file containing data about the convex hull. The 1st line encodes *m* which is the number of points in the convex hull. Thereafter, it is followed by *m* lines where each line contains two **double** data type values representing the *x* and *y* coordinates of the points making up the convex hull. One or more blank spaces separate the *x* and *y* coordinates.

STRICT REQUIREMENT #2:

a. **The x, y coordinate values should be formatted such that there are 6 digits after the decimal point.**
b. **The points should be listed in the output text file such that the first is the anchor, followed by the succeeding points such that when the points are connected, they trace the convex in counterclockwise direction.**

Study the accompanying example output text files **`sample-output.txt`** and **`output-circle.txt`**.

III. TASKS

Task 1. Design and implement your stack data structure.

- Implement a stack data structure with the following operations (1) **`CREATE(S)`**, (2) **`PUSH(S, elem)`**, (3) **`elem = POP(S)`**, (4) **`elem = TOP(S)`**, (5) **`ISFULL(S)`** and (6) **`ISEMPTY(S)`**.
- **VERY IMPORTANT: Implement as an additional stack operation `elem = NEXT-TO-TOP(S)` which will return the element below the top element. For example: create a stack S. Then push 10, then push 20, and then push 30. `TOP(S)` will return 30. `NEXT-TO-TOP(S)` will return 20.**
- You have to decide on your own if you would like to represent the stack using an array or a linked list.
- **STRICT REQUIREMENT #3: make sure that your stack can store up to a maximum of n = 2^15 = 32768 elements.**
- Practice modular programming. That is, compartmentalize the data structures and operations by storing the implementation codes in separate source code files. For example, the codes for stack data structure are stored in the source files **`stack.h`** and **`stack.c`**.

Task 2. Implement TWO sorting algorithms.

- Sorting is a crucial step in Graham's Scan algorithm. It is the step that contributes the most to its time complexity (performance).
- For this project, you will explore and compare the performance of a relatively "slow" sorting algorithm versus a "fast" sorting algorithm (the algorithms will be assigned to you randomly):
  - "Slow": one of bubble sort, selection sort, insertion sort (quadratic algorithms)
  - "Fast": one of merge sort, heapsort, quicksort (linear log algorithms)
- **VERY IMPORTANT: sorting is based on the polar angle made by the anchor point P_0 with another point, say P_i. The anchor point is the point with the lowest (minimum y coordinate). If there are several points with the same minimum y coordinate, choose the point with the lower x coordinate. Refer to Figure 1 below. Read also about polar coordinates in** Polar and Cartesian Coordinates (mathsisfun.com)
- Implement the two sorting algorithms. Encode also any helper functions that you need to accomplish the sorting task. Depending on how you implement the Graham Scan algorithm, you may need to implement helper functions for computing the polar angle.
- Practice modular programming. Compartmentalize the two sorting algorithm implementations in the source files **`sort.h`** and **`sort.c`**.

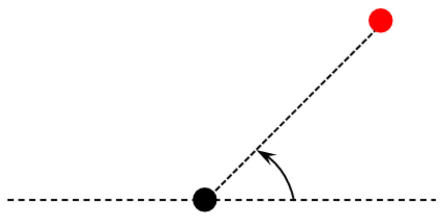

Figure 1: the polar angle is the angle, measured in radians, made by the line segment from the anchor point (black circle) to another point (red circle) with respect to the line parallel to the x-axis.

Task 3. Implement two versions of Graham's Scan algorithm.

- Implement Graham's Scan algorithm (which should call the functions for sorting, and the necessary stack operations).
- There should be two versions. The first version uses a "slow" sorting algorithm, and the second version uses a "fast" sorting algorithm.
- **STRICT REQUIREMENT #4: Both versions should have a printf() statement that displays the elapsed time (in milliseconds) between the start and end of the algorithm implementation execution.** Read, understand and learn from the accompanying C program `empirical-time.c` on how to compute the elapsed time using the `clock()` library function.
- Practice modular programming by compartmentalizing your solution into two separate source files named as `graham_scan1.c` (for the "slow" version), and `graham_scan2.c` (for the "fast" version).

Task 4. Implement two driver programs to test the two versions.

- Each driver program should have its own `main()` function that will do the following sequence of steps:
  1. ask the user to input the filename and extension of the input text file
  2. ask the user to input the filename and extension of the output text file
  3. open the input text file using `fopen()` library function
  4. read the input text file contents using the `fscanf()` library function
  5. call the appropriate version of the Graham scan algorithm implementation
  6. write the *m* output points comprising the convex hull into the output text file using `fprintf()` library function
  7. close the input and output text files using the `fclose()` library function.

- Practice modular programming by compartmentalizing your solution in two separate source files named as `main1.c` (for the "slow" version) and `main2.c` (for the "fast" version).

Task 5. Test your solution.

- Perform exhaustive testing.
- Create at least 5 test case text files with different values of *n* with a minimum of *n = 2^6 = 64 points*. Set the values of *n* such that they are expressed using 2 as exponent (for example *n = 2^8 = 256*).
- Run the two versions of your program for each test case.
- Tabulate the elapsed time for each test case – this will be used in the **Analysis** section of the project documentation.
- **VERY IMPORTANT: We will stress test your codes/solutions! Your 5th (last) test case text file should contain n = 2^15 = 32768 points, i.e., (x, y) coordinates.**

Task 6. Document your implementation and the test results.

- Give a brief description of how you implemented the data structures. For example, did you use arrays? Did you use linked lists (dynamic memory allocation)?
- Disclose in detail what is not working in your submission.
- Discussion of Results – compare the performance of the "slow" versus the "fast" version. Support the discussion with a graph (showing *n* in the *x* axis, and corresponding elapsed time in the *y* axis). The graph will show the relative performance of the "slow" versus the "fast" version as *n* increases.

IV. DELIVERABLES

Submit via Canvas a ZIP file named **`GROUPNUMBER.ZIP`** (for example 01.ZIP for group number 1) which contains:

- Source files (.h and .c files) of the modules as specified above.
- Input text files (.txt) that you used in testing.
- Corresponding output text files (.txt) using the fast version. [Actually, the slow version should produce the same output.]
- Documentation named **`GROUPNUMBER.PDF`** (for example 01.PDF for group number 1). Use the accompanying template **`Documentation-Template.DOCX`** file.

Do NOT include any EXEcutable file in your submission.

V. RUBRIC FOR GRADING

Criteria	**RATINGS**		
1. Stack	**COMPLETE [25 pts]** Correctly implemented the stack data structure.	**INCOMPLETE [5 to 15 pts]** Implementation is not entirely correct.	**NO MARKS [0 pt]** Not implemented.
2. Sorting algorithms	**COMPLETE [15 pts]** Correctly implemented the slow and fast sorting algorithms.	**INCOMPLETE [5 to 10 pts]** Implementation is not entirely correct.	**NO MARKS [0 pt]** Not implemented.
3. Graham's Scan algorithm	**COMPLETE [40 pts]** The implementation Graham's Scan algorithm worked correctly on ALL test data used during the project demo.	**INCOMPLETE [5 to 25 pts]** The implementation Graham's Scan algorithm worked correctly on some but not all test data used during the project demo.	**NO MARKS [0 pt]** Not implemented.
4. Documentation and Analysis	**COMPLETE [15 pts]** Required details were covered. Take note of the following point distributions: a. Comparison Table [4 pts] b. Line Graphs [4 pts] c. Analysis [4 pts] d. Remainder of the document [3 points]	**INCOMPLETE [1 to 10]** Some required details were not discussed.	**NO MARKS [0 pt]** No documentation.
5. Compliance with Instructions	**COMPLIANT [5 pts]** All instructions were complied with.	**NON-COMPLIANCE [0 to 4]** Deduction of 1 point for every instruction not properly complied with. Examples: included an EXE file in the ZIP file, file naming not followed.	**NO MARKS [0 pt]** More than 4 instructions not complied with.
			Maximum Total Points: 100

VI. WORKING WITH GROUPMATES

You are to accomplish this project in collaboration with your fellow students. Form a group of at least 2 to at most 3 members. The group may be composed of students from different sections. Make sure that each member of the group has approximately the same amount of contribution for the project. Problems with groupmates must be discussed internally within the group, and if needed, with the lecturer.

VII. HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honest policy applies. You are encouraged to read and gather information you need to implement the project. **However, please take note that you are NOT allowed to copy-and-paste -- in full or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by other people that are not online). You should develop your own codes from scratch by yourselves, i.e., in cooperation with your groupmates.** According to the handbook (5.2.4.2), *"faculty members have the right to demand the presentation of a student's ID, to give a grade of 0.0, and to deny admission to class of any student caught cheating under Sec. 5.3.1.1 to Sec. 5.3.1.1.6. The student should immediately be informed of his/her grade and barred from further attending his/her classes."*

Question? Please post it in the Canvas Discussion thread. Thank you for your cooperation.

サルバドール・フロランテ