EE 599 Lab3
Spring 2019 Nazarian 100 Points

Student ID: _____ Name: _____

Assigned: Wednesday Jan. 23

Due: Wednesday Jan. 30 at 11:59pm

Late submissions will be accepted for two days after the deadline with a maximum penalty of 15% per day. For each day, submissions between 12am and 1am: 2% penalty, between 1 and 2am: 4%, 2-3am: 8%, and after 3am: 15%. No submissions accepted Friday April 1st, after 11:59pm.

Notes:

This assignment is based on individual work. No collaboration is allowed (Discussing the technical details of each problem with other students before submitting your work, copying from students of current or previous semesters is not permitted in our lab and HW assignments). You may post your questions on the discussion forums and/or use the office hours. We may pick some students in random to demonstrate their design and simulations. Please refer to the syllabus and the first lecture regarding the USC and our policies, including the penalties for any violation. If you have any doubts about what is allowed or prohibited in this course, please contact the instructor.

What You Will Practice

In this lab, you are going to practice OOP, multidimensional array manipulations and sorting **in C++**.

Problem Description - "3D Burnt Pancake Sorting"

A chef for a catering event is sloppy, and when he prepares a bunch of pancakes they come out all different sizes. The pancakes are burnt on one side; also they are randomly flipped with the burnt at the top or bottom side. There are M pancakes, such that M = X*Y*Z, where X, Y and Z are known. They are placed in X*Y piles in a rectangular pan, where there are Z pancakes in each pile. This means there are X rows and Y columns of the pancake piles.

Part 1: Implement Solutions to Pancake Sorting Problems (40%)

1. The Original Pancake Sorting Problem (20%)

You would like to rearrange each pile of the pancakes so that the the sizes of them are in ascending order, i.e. the smallest winds up on top, and so on, down to the largest at the bottom. The only operation you are allowed to perform is grabbing several pancakes from the

top and flipping them over, repeating this (varying the number you flip) as many times as necessary.

2. Burnt Pancake Sorting Problem (20%)

The pancakes are burnt on one side, so that in addition to ordering them correctly by size, they must also be placed with the burnt side face down. Initially, the burnt side of a pancake could face either up or down.

Part 2: Rearrange the 3D Pancake Volume (30%)

1. Sort the Interior Piles (15%)

Run your Burnt Pancake Sorting solution for every "interior" pile of pancakes such that they are all sorted **ascendingly**, meaning the largest pancake is placed at the bottom of each pile and the smallest at the top. An "interior" column is defined as a pile (of size Z pancakes) with the coordinate variable x as neither zero nor X-1, and also the coordinate variable y as neither zero, nor Y-1; with x iterating from 0 to X-1 and y from 0 to Y-1. Also for each pile the burnt side of each pancake should be **down** (e.g., from the top view, one would see the non-burnt side of the pancakes for all interior columns. Note that you should do the sorting for (X-2)*(Y-2) piles of pancakes.

2. Sort the Exterior Piles (15%)

Run your Burnt Pancake Sorting solution for every "exterior" pile such that they are sorted **descendingly** and the burnt side for each pancake faces **up**. An exterior pile is defined as a pile (of size Z pancakes) such that its x or y coordinates follow one of these values: x=0, or x=X-1, or y=0, or y=Y-1.

OOP Requirements (30%)

You should Implement the following three classes:
1) Class *Pancake* that includes an integer variable *Size* and a bool variable *Burnt*. Burnt =1 means the top side is burnt (and 0 mean the down side is burnt). A method *flip_pancake()* should be included that flips the pancake, i.e., inverts Burnt.
2) Class *PancakePile* should be defined. It should include an array of size Z of pancake objects of type Pancake. The array can be maintained based on a pointer, or statically sized (of size Z). This class should include four methods: a) *pancake_sort_ascending()*; b) *pancake_sort_descending()*; c) *pancake_sort_ascending_burnt_down()*; d) *pancake_sort_descending_burnt_up()*.
3) Class *MPancakePiles* which must include X*Y objects of type class *PancakePile*. This class should include two methods: a) *sort_interior()*; b) *sort_exterior().*

Part 3: Implement Optimized Pancake Sorting Algorithms (Extra Credit: 30%)

In the Part 1, the solution you implemented takes at most $2n - 3$ flipping movements to leave the pancakes perfectly ordered, where n is the number of pancakes in the pile. Bill Gates, the one who founded Microsoft, proposed an optimized algorithm that gave an upper bound of *(5n+5)/3* in his only academic paper titled *Bounds for Sorting by Prefix Reversal* [1] in 1979. 30 years later, a team of researchers from the University of Texas at Dallas, led by Professor Hal Sudborough proposed a further improved algorithm to establish that upper limit at *18n/11* [2]. Pick one of these two algorithms to implement a solution to the original pancake sorting problem. Compare its runtime with the one you implement in Part 1.

Input File Format

You will be given a few text files named "input.txt" one at a time. This file contains two pieces of information. (i) size of the 3D pancake volume, i.e. number of rows (X), number of columns (Y), and height of each pile (Z), separated by a space character; (ii) size and burt side position of each pancake, separated by comma. The format of input files is defined: The first line is the size of the pancake volume. Starting from second line, each line represents a pancake pile, and it consists of Z pairs of integer numbers representing the properties of each pancake, Size followed by Burnt. Burnt=1 means the top side is burnt (and 0 mean the down side is burnt). Pancakes within each pancake pile is separated by a space character. The pancake piles are listed row by row in the X-Y dimension. Here is an example of "input.txt". The first line tells us X=3, Y=4 and Z=5. Line 2 to Line 5 represent the first row of 4 piles of pancakes, where each pile has 5 pancakes.

In "input.txt" file:

```
3 4 5
4,1 20,1 65,1 95,1 36,0
30,0 19,0 46,1 63,1 5,0
45,1 18,0 79,1 82,0 86,1
5,1 60,1 13,0 47,0 10,0
29,0 33,1 56,0 23,0 20,1
43,1 65,1 42,1 75,1 85,1
12,0 95,0 61,1 50,0 90,0
24,1 33,1 3,0 10,1 88,0
30,0 46,1 58,1 1,1 87,1
69,0 91,1 1,1 41,0 51,0
```

77,1 99,0 99,0 70,0 51,1
55,0 41,1 37,0 41,1 81,0

Output File

After obtaining the result of rearranging the pancakes, you should write your results to a text files with name "**output.txt**". The format should be the same as the input file. You should write your result to files instead of printing in terminal. For example, the desired output for above example will be as follows:

---

3 4 5
95,1 65,1 36,1 20,1 4,1
63,1 46,1 30,1 19,1 5,1
86,1 82,1 79,1 45,1 18,1
60,1 47,1 13,1 10,1 5,1
56,1 33,1 29,1 23,1 20,1
42,0 43,0 65,0 75,0 85,0
12,0 50,0 61,0 90,0 95,0
88,1 33,1 24,1 10,1 3,1
87,1 58,1 46,1 30,1 1,1
91,1 69,1 51,1 41,1 1,1
99,1 99,1 77,1 70,1 51,1
81,1 55,1 41,1 41,1 37,1

Part 4: Optional: Statistical Analysis of the Pancake Piles

Design a Python script that performs statistical analysis of piles, in terms the variance of their sizes, various correlations between different piles, such as cross-correlation, anti-correlation, etc. Please be reminded that this parts has been changed to optional, hence no submission is required for this part.

3. Notes

- X>1, Y>1, and Z>=1. The value of X, Y, or Z will be at most 512.
- You can assume that all the pancake sizes are positive integers.
- Your code should be written in only **ONE** file named "**EE599_Lab3_<STUDENT-ID>.cpp**" where you should implement all the classes and the main driver function. Please replace

<STUDENT-ID> with your own 10 digit USC-ID, the name of the file would be something like: EE599_Lab3_1234567890.cpp

- An "input.txt" and correct "output_golden.txt" files are included in the starter repo for your reference. You can use them to test your code, but there will be other inputs for grading your code.

Summary

- You are **NOT** allowed to use STL in this lab. You can only use standard libraries, such as iostream, fstream, string, etc. For other unsure libraries, please ask in discussion forum first before you use it.
- You have to follow the exact formats and syntax for generating the required results. Even adding an extra white line in the beginning of your output file or an extra space, etc., will result in losing the whole score of this lab.
- There should be only one code file "**EE599_Lab3_<STUDENT-ID>.cpp**", where you implement all the classes and the main driver function. Compiling and executing it will generate one output file.
- You can use the given shell script "check_results.sh" to check whether your result is same as the golden answer, which is similar to what you do in Lab1.
- You should also write a README file. You can write a .txt file or .md file. A README file typically contains information of author's name and email, a brief program summary, references and instructions. Besides, include any information that you think the course staff, especially the grader should know while grading your assignment: references, any non-working part, any concerns, etc.
    1) Any non-working part should be clearly stated.
    2) The citations should be done carefully and clearly, e.g.: "to write my code, lines 27 to 65, I used the Diijkstra's shortest path algorithm c++ code from the following website: www.SampleWebsite.com/ ..."
    3) The Readme file content of labs cannot be hand-written and should only be typed.

Submission

Submit your code files and a README file using GitHub. Please replace <STUDENT-ID> with your own 10 digit USC-ID, the name of the file would be something like: EE599_Lab3_1234567890.cpp

References

[1] Gates, William H., and Christos H. Papadimitriou. "Bounds for sorting by prefix reversal." *Discrete Mathematics* 27.1 (1979): 47-57.

[2] Chitturi, Bhadrachalam, et al. "An (18/11) n upper bound for sorting by prefix reversals." *Theoretical Computer Science* 410.36 (2009): 3372-3390.