Sabanci University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Summer 2020-2021

Take-Home Exam 4 –Childhood's End
Due: 23 August 2020 11.55pm (Sharp Deadline)

DISCLAIMER:

Your program should be a robust one such that you have to consider all relevant user mistakes and extreme cases; you are expected to take actions accordingly!

Only checking the sample run cases might not be sufficient as your solution will be checked against a variety of samples different from the provided samples; however checking these cases is highly encouraged and recommended.

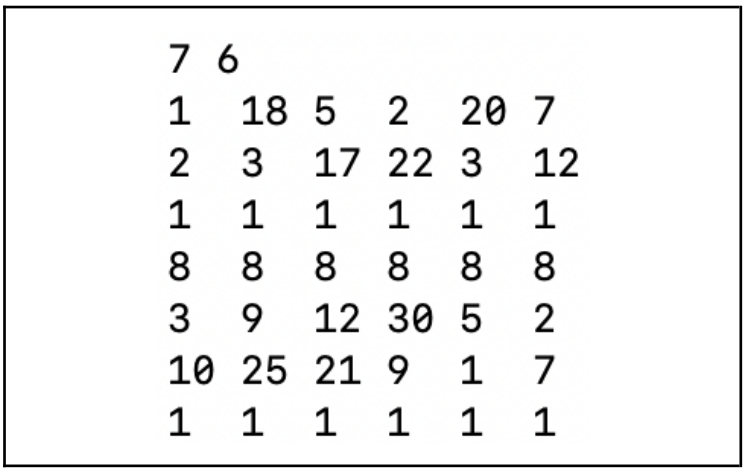
You must NOT collaborate with your friends and discuss your solutions with each other. You have to write down the code on your own. Plagiarism will not be tolerated AND cooperation is not an excuse!

Introduction

The aim of this assignment is to practice multi-threading. You will write a program which updates the cells of a matrix such that the neighbouring cells of the matrix do not have too unbalanced values.

Inputs to Your Program

Your program should prompt for an input file name and read the file name from the standard input (cin). This file will include an integer matrix, structure of which will be in the first line, as exemplified below.



```
7 6
1  18 5  2  20 7
2  3  17 22 3  12
1  1  1  1  1  1
8  8  8  8  8  8
3  9  12 30 5  2
10 25 21 9  1  7
1  1  1  1  1  1
```

matrix1.txt

If the input file cannot be opened successfully, then your program should give an error message and terminate.

While reading the file, **you must store this matrix in a truly implemented 2D dynamic array** (covered in week-2, you may check the slide set 2.2-pointers-linkkedlists).

Format of the Inputs

In the first line of the data file containing the matrix, there will be the dimensions of the respective matrix, i.e. the number of rows and the number of columns, in this given order, separated by some space characters.

In each line of this file, the integers will be separated by some space characters, and the rows will be separated by a newline ('\n') character, as seen above in *matrix1.txt*. Also, there will not be any empty lines in this file. You may assume that this file won't contain any characters other than digits, spaces, newlines and EOF. You may also assume that the row and column count given at the beginning of the file will always be correct.

Program Flow and Operations

Your program should start by asking the filename for the matrix file. If the file can be successfully opened, then your program should read the file and store the numbers in a truly implemented 2D dynamic array.

After reading the matrix, the main algorithm will start to run. This algorithm will try to spot the cells which are too low as compared to its main neighbours and fix this issue by taking from the richest neighbour and giving to the poor one (itself) :) However, we do not want to transfer much in one take, in order not to disturb the general balance. Due to this, one round is not sufficient to settle the balance. Therefore, we do this several times until no changes are done in a single round. The resulting matrix will be a much more balanced matrix, with the same summation of numbers as the original one. The single-threaded version of this algorithm is given below in the form of pseudocode.

```
while increments/decrements were done in the previous round:
     for each cell in matrix:
          mx = maximum of the main neighbouring cells
          if mx >= 2 * cell:
               increment cell by ceiling(cell / 3)
               decrement mx by ceiling(cell / 3)
```

Please be reminded that the above-given information is only a pseudocode of the algorithm and many implementation details are hidden. Please also be reminded that each cell has at most 4 main neighbours: the other 4 neighbours who lay at the corner of the cells are not **main** neighbours.

Throughout the execution of the algorithm, the program should inform the user about the starts and ends of the rounds, the current updated version of the matrix, and also about which operations were done. The detailed format can be found in the Sample Runs section.

Multi-threading

As this assignment is a multi-threading assignment, the previously given version of the algorithm is only a hint of what you are going to really build. Rather than forming a loop on the cells and doing the operations one by one, you will spawn one thread for each cell and let the threads do the operation on that specific cell, on their own. After all the threads are **join()**ed, the main thread will collect the results, decide whether to go on with another round or not, print the current matrix and the current situation. If another round happens, the same thread creation and joining process will repeat.

The tasks that threads accomplish are critical. As many operations occur at the same time, it will cause problems that multiple threads access the same data at the same time. In order to make sure that no data read/write will clash between threads, a thread should access its designated cell and the 4 neighbouring cells in isolation, meaning that no other thread may access these 5 cells during that time frame. In order to accomplish this, you should use mutexes.

There should exist one mutex for each cell, representing the occupancy of this cell. A thread must **try_lock()** all 5 mutexes before doing any calculations/operations. If a thread fails to lock any of the 5 mutexes of interest, it should unlock all the mutexes it has already locked, **yield()** its execution, and start **try_lock()**ing those mutexes from scratch. Your threads really should leave all the locks in case of failing to obtain one of the locks first, for the sake of avoiding deadlocks.

In case of success to obtain all the 5 locks, the thread should calculate the maximum of the neighbours, check if this maximum cell is as big as the double of the cell. If so, it should do the change between these two cells and print a summary of this operation. Afterwards, whether a change is done or not, the thread will leave the mutex locks and terminate.

The mutex operations explained above are very important in terms of data consistency. As our computers have more and more cores every year, multi-threaded programs are a must. And data consistency is a problem that should be solved in multi-threaded programs. So, we place great value in proper use of mutexes and avoiding improper waits between the threads in this assignment. **Failing to use mutexes properly, faking to use mutexes or multi-threading, making threads wait for each other unnecessarily and causing deadlocks will decrease your grade harshly.**

One last use of mutex is the following: If multiple threads try to print to the console at the same time, the messages may mess up. So, make sure that you have a separate single mutex for display purposes. Lock it before cout operations on thread functions and unlock right afterwards. This will make your cout operations consistent on the console.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are the standard input (cin) taken from the user (i.e., like ***this***). You should display the required information with the same words/spaces as here to easen the process of grading. You can use setw(5) from the <iomanip> library to print the matrix with proper spacing.

You will **not** be using GradeChecker for this assignment and we will grade your submissions manually. It is totally okay to have the order of the operations and the resulting matrix different from the sample run. We will check the consistency of the operations and a correctly balanced resulting matrix, as well as your code itself for confirming the correct use of multi-threading and locking mechanisms.

Sample Run 1

```
Welcome to the last assignment :(
Please enter a file name for the matrix: matrix1.txt
-----
Printing the original matrix:
    1   18    5    2   20    7
    2    3   17   22    3   12
    1    1    1    1    1    1
    8    8    8    8    8    8
    3    9   12   30    5    2
   10   25   21    9    1    7
    1    1    1    1    1    1
-----
A new round starts
Row-0,Col-0 (1) is incremented by 1 by stealing from the cell to the
right (18).
Row-0,Col-2 (5) is incremented by 2 by stealing from the cell below
(17).
```

Row-0,Col-3 (2) is incremented by 1 by stealing from the cell below (22).
Row-0,Col-5 (7) is incremented by 3 by stealing from the cell to the left (20).
Row-1,Col-4 (3) is incremented by 1 by stealing from the cell to the left (21).
Row-2,Col-0 (1) is incremented by 1 by stealing from the cell below (8).
Row-2,Col-2 (1) is incremented by 1 by stealing from the cell above (15).
Row-1,Col-1 (3) is incremented by 1 by stealing from the cell above (17).
Row-2,Col-4 (1) is incremented by 1 by stealing from the cell below (8).
Row-2,Col-1 (1) is incremented by 1 by stealing from the cell below (8).
Row-4,Col-0 (3) is incremented by 1 by stealing from the cell below (10).
Row-4,Col-5 (2) is incremented by 1 by stealing from the cell above (8).
Row-3,Col-3 (8) is incremented by 3 by stealing from the cell below (30).
Row-4,Col-1 (9) is incremented by 3 by stealing from the cell below (25).
Row-5,Col-3 (9) is incremented by 3 by stealing from the cell above (27).
Row-5,Col-4 (1) is incremented by 1 by stealing from the cell to the left (11).
Row-5,Col-0 (9) is incremented by 3 by stealing from the cell to the right (22).
Row-2,Col-3 (1) is incremented by 1 by stealing from the cell above (20).
Row-4,Col-2 (12) is incremented by 4 by stealing from the cell to the right (24).
Row-6,Col-3 (1) is incremented by 1 by stealing from the cell above (12).
Row-6,Col-0 (1) is incremented by 1 by stealing from the cell above (12).
Row-6,Col-5 (1) is incremented by 1 by stealing from the cell above (7).
Row-6,Col-2 (1) is incremented by 1 by stealing from the cell above (21).
Row-6,Col-4 (1) is incremented by 1 by stealing from the cell to the left (2).
Row-2,Col-5 (1) is incremented by 1 by stealing from the cell above (12).
Row-4,Col-4 (5) is incremented by 2 by stealing from the cell to the left (20).

```
Row-6,Col-1 (1) is incremented by 1 by stealing from the cell above
(19).
The round ends with updates.
Printing the matrix after the updates:
    2   16    7    3   17   10
    2    4   14   19    4   11
    2    2    2    2    2    2
    7    7    8   11    7    7
    4   12   16   18    7    3
   11   18   20   10    2    6
    2    2    2    1    2    2
-----
A new round starts
Row-0,Col-0 (2) is incremented by 1 by stealing from the cell to the
right (16).
Row-0,Col-2 (7) is incremented by 3 by stealing from the cell to the
left (15).
Row-0,Col-3 (3) is incremented by 1 by stealing from the cell below
(19).
Row-1,Col-0 (2) is incremented by 1 by stealing from the cell to the
right (4).
Row-1,Col-4 (4) is incremented by 2 by stealing from the cell to the
left (18).
Row-2,Col-0 (2) is incremented by 1 by stealing from the cell below
(7).
Row-2,Col-2 (2) is incremented by 1 by stealing from the cell above
(14).
Row-1,Col-1 (3) is incremented by 1 by stealing from the cell to the
right (13).
Row-2,Col-4 (2) is incremented by 1 by stealing from the cell below
(7).
Row-2,Col-3 (2) is incremented by 1 by stealing from the cell above
(16).
Row-4,Col-0 (4) is incremented by 2 by stealing from the cell to the
right (12).
Row-3,Col-2 (8) is incremented by 3 by stealing from the cell below
(16).
Row-2,Col-5 (2) is incremented by 1 by stealing from the cell above
(11).
Row-2,Col-1 (2) is incremented by 1 by stealing from the cell below
(7).
Row-4,Col-4 (7) is incremented by 3 by stealing from the cell to the
left (18).
Row-4,Col-5 (3) is incremented by 1 by stealing from the cell to the
left (10).
Row-5,Col-3 (10) is incremented by 4 by stealing from the cell to
the left (20).
Row-6,Col-2 (2) is incremented by 1 by stealing from the cell above
```

(16).
Row-5,Col-4 (2) is incremented by 1 by stealing from the cell to the left (14).
Row-6,Col-0 (2) is incremented by 1 by stealing from the cell above (11).
Row-6,Col-5 (2) is incremented by 1 by stealing from the cell above (6).
Row-6,Col-3 (1) is incremented by 1 by stealing from the cell above (13).
Row-6,Col-1 (2) is incremented by 1 by stealing from the cell above (18).
The round ends with updates.
Printing the matrix after the updates:

3	12	10	4	17	10
3	4	12	15	6	10
3	3	3	3	3	3
6	6	11	11	6	7
6	10	13	15	9	4
10	17	15	12	3	5
3	3	3	2	2	3

A new round starts
Row-0,Col-0 (3) is incremented by 1 by stealing from the cell to the right (12).
Row-0,Col-3 (4) is incremented by 2 by stealing from the cell to the right (17).
Row-2,Col-1 (3) is incremented by 1 by stealing from the cell below (6).
Row-2,Col-2 (3) is incremented by 1 by stealing from the cell above (12).
Row-1,Col-4 (6) is incremented by 2 by stealing from the cell above (15).
Row-1,Col-1 (4) is incremented by 2 by stealing from the cell above (11).
Row-2,Col-4 (3) is incremented by 1 by stealing from the cell above (8).
Row-2,Col-0 (3) is incremented by 1 by stealing from the cell below (6).
Row-2,Col-3 (3) is incremented by 1 by stealing from the cell above (15).
Row-2,Col-5 (3) is incremented by 1 by stealing from the cell above (10).
Row-3,Col-1 (5) is incremented by 2 by stealing from the cell to the right (11).
Row-5,Col-4 (3) is incremented by 1 by stealing from the cell to the left (12).
Row-6,Col-1 (3) is incremented by 1 by stealing from the cell above (17).

```
Row-4,Col-5 (4) is incremented by 2 by stealing from the cell to the
left (9).
Row-6,Col-0 (3) is incremented by 1 by stealing from the cell above
(10).
Row-6,Col-2 (3) is incremented by 1 by stealing from the cell above
(15).
Row-6,Col-4 (2) is incremented by 1 by stealing from the cell above
(4).
Row-6,Col-3 (2) is incremented by 1 by stealing from the cell above
(11).
The round ends with updates.
Printing the matrix after the updates:
    4    9   10    6   13   10
    3    6   11   14    7    9
    4    4    4    4    4    4
    5    7    9   11    6    7
    6   10   13   15    7    6
    9   16   14   10    3    5
    4    4    4    3    3    3
-----
A new round starts
Row-0,Col-3 (6) is incremented by 2 by stealing from the cell below
(14).
Row-0,Col-0 (4) is incremented by 2 by stealing from the cell to the
right (9).
Row-1,Col-0 (3) is incremented by 1 by stealing from the cell above
(6).
Row-2,Col-2 (4) is incremented by 2 by stealing from the cell above
(11).
Row-2,Col-5 (4) is incremented by 2 by stealing from the cell above
(9).
Row-2,Col-3 (4) is incremented by 2 by stealing from the cell above
(12).
Row-4,Col-4 (7) is incremented by 3 by stealing from the cell to the
left (15).
Row-5,Col-4 (3) is incremented by 1 by stealing from the cell above
(10).
Row-6,Col-0 (4) is incremented by 2 by stealing from the cell above
(9).
Row-6,Col-3 (3) is incremented by 1 by stealing from the cell above
(10).
Row-6,Col-1 (4) is incremented by 2 by stealing from the cell above
(16).
Row-6,Col-2 (4) is incremented by 2 by stealing from the cell above
(14).
The round ends with updates.
Printing the matrix after the updates:
    5    7   10    8   13   10
```

```
    4    6    9   10    7    7
    4    4    6    6    4    6
    5    7    9   11    6    7
    6   10   13   12    9    6
    7   14   12    9    4    5
    6    6    6    4    3    3
-----
A new round starts
Row-5,Col-0 (7) is incremented by 3 by stealing from the cell to the
right (14).
Row-5,Col-4 (4) is incremented by 2 by stealing from the cell above
(9).
Row-6,Col-2 (6) is incremented by 2 by stealing from the cell above
(12).
Row-6,Col-4 (3) is incremented by 1 by stealing from the cell above
(6).
Row-6,Col-3 (4) is incremented by 2 by stealing from the cell above
(9).
The round ends with updates.
Printing the matrix after the updates:
    5    7   10    8   13   10
    4    6    9   10    7    7
    4    4    6    6    4    6
    5    7    9   11    6    7
    6   10   13   12    7    6
   10   11   10    7    5    5
    6    6    8    6    4    3
-----
A new round starts
The round ends without updates.
-----
Program is exiting...
```

Some Important Rules

Although some of the information is given below, please also read THE submission and grading policies from the lecture notes of the first week. In order to get a full credit, your program must be efficient, modular (with the use of functions), well commented and indented. Besides, you also have to use understandable identifier names. Presence of any redundant computation, bad indentation, meaningless identifiers or missing/irrelevant comments may decrease your grade in case that we detect them.

When we grade your submissions, we pay attention to these issues. Moreover, in order to observe the real performance of your code, we are going to run your programs in Release mode and **we may test your programs with very large test cases**. Hence, take into

consideration the efficiency of your algorithms other than correctness.

How to get help?

You may ask your questions to TAs or to the instructor. Information regarding the office hours of the TAs and the instructor are available at SUCourse+.

YOU CANNOT USE GRADE CHECKER FOR THIS ASSIGNMENT!

Due to the irregular nature of threads, it is impossible to produce a consistent output and order of operations in multithreaded programs. As GradeChecker is a tool which works on the output of the programs, we cannot offer it for testing this assignment.

Your programs will be graded manually. This means, the assistants who grade your THE will inspect your codes thoroughly. If the mutex mechanisms and multithreaded programming techniques are misused or worked-around, we will be detecting such cases easily. Your grade might be lowered significantly in such situations. In an extreme case, if you submit a single-threaded program offering the same solution, your grade will be zero.

Submit via SUCourse+ ONLY!

The internal clock of SUCourse+ might be a couple of minutes skewed, so make sure you do not leave the submission to the last minute. In the case of failing to submit your THE on time:

"No successful submission on SUCourse on time = A grade of 0 directly."

What and where to submit (PLEASE READ, IMPORTANT)

It'd be a good idea to write your name and lastname in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your full name is "Duygu Karaoğlan Altop", and if you want to write it as comment; then you must type it as follows:

// Duygu Karaoglan Altop

Submission guidelines are below. You are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be *lowered*.

- Name your cpp file that contains your program as follows:

"SUCourseUserName_the4.cpp"

Your SUCourse user name is actually your SUNet username which is used for checking sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SU e-mail address is **atam@sabanciuniv.edu**, then the file name must be: **"atam_the4.cpp"**

- Please make sure that this file is the latest version of your THE program.
- You should upload all the .txt files and class files to SUCourse+ as well.
- Do not zip any of the documents but upload them as separate files only.
- Submit your work **through SUCourse+ only**!

Plagiarism

Plagiarism is checked by automated tools and we are very capable of detecting such cases. Be careful with that...

Exchange of abstract ideas are totally okay but once you start sharing the code with each other, it is very probable to get caught by plagiarism. So, do NOT send any part of your code to your friends by any means or you might be charged as well, although you have done your THE by yourself. THEs are to be done personally and you have to submit your own work. **Cooperation will NOT be counted as an excuse.**

In case of plagiarism, the rules on the Syllabus apply.

Good Luck!
Duygu K. Altop