ECE/CS 5510 Multiprocessor Programming Fall 2018
Homework 1

Part I: Problems [50 points]:

Learning objective: Reinforcing understanding of the problem of mutual exclusion, its definition, concurrent reasoning and coordination, Amdahl's Law.

Solve the following problems [Problems 1 to 5 = 8 points, Problem 6 = 10 points.]:

1. For each of the following, state whether it is a safety property or a liveness property. Identify the bad or good thing of interest.
    a. Two processes with critical sections *c0* and *c1* may not be in their critical sections at the same time.
    b. Clients may not retain resources forever.
    c. It is always cloudy before it rains.
    d. Only one person can sit on the driver's seat in a car.
    e. A red traffic light (stop) will turn green (proceed) sometime in future.
    f. Cars entering a roundabout yield to cars already in the roundabout.
    g. A car in a roundabout eventually leaves the roundabout.
    h. At a four-way intersection, no more than one direction/side should be green at any given time
2. A programmer has two computer systems available to run his/her application. The first system has a single-core processor that executes *N*X* instructions per second. The second system has *2*N*-core processor where each core executes *X* instructions per second. Using Amdahl's law, find the lower bound on application parallelism, so that it is advantageous to run the application on the second system from a performance standpoint.
3. An application takes *T'* seconds to execute on an *N*-core processor. When the sequential part of the application is optimized and made 10x faster, the program takes *T'/4* seconds to execute on an *N*-core processor. What portion of the overall execution time (i.e., equivalent execution on a single-core processor) did the sequential part account for before the optimization? (as a function of *N*). Also find the lower bound on the number of cores *N* for which this scenario holds true.
4. An application has a last level cache miss rate of 50%; i.e., 50% of the executed memory instructions go to the DRAM to fetch data. Assume that the execution of 30% of such instructions is stalled because of the refresh mechanism in DRAM. What is the expected performance benefit for this application if the penalty incurred by DRAM refresh is fully eliminated?

5. In the following code snippet, can the inner loop be parallelized? Can the outer loop be parallelized? Give reason. Assume matrix `a` is already initialized and has appropriate dimension for the code snippet to work.

```
for(i = 1; i < n; ++i) {
    for(j = 0; j < n; ++j) {
      a[j][i] += a[j][i+1] - a[j][i-1];
    }
}
```

6. There are 12 students taking an exam. Each student has their own examination room. Each student is individually led to a common room, only one at a time. During this time, each of the remaining 11 students is still in their respective examination room. The common room contains two table lamps, that can be set to an ON or OFF position. The only way for the students to pass is by going into the room and stating that all other students have been in the room. If the statement is true, then all students will pass. If the statement is false, all students will fail. The only way for students to communicate state information is using the state of table lamps (on or off). That is, students can only use their own memory and the state of the table lamps to decide if all of the other students have been in the room. Students can be taken to the room infinitely many times, and in no particular order. The important thing to note, is that the students know about the exam ahead of time and can coordinate a plan beforehand to pass. Suggest a plan that ensures all students succeed in the examination.

Part II: Programming assignment [50 points]

Learning objective: Getting started programming in Java, reinforcing Java concepts, getting used to reading the Java documentation, setting up IntelliJ IDE.

You will write a (single-threaded) console-based inventory manager application, called Inventory. You may need to use the following Java classes: Date, DateFormat, String, ArrayList, Scanner/BufferedReader/FileReader, Matcher, and others.

Inventory will manage an in-memory database of items in a shop. It will operate upon this database by reading and executing commands from an input file named **in.txt**. The program will also produce results, which will be written to a file named **out.txt**. For this program, it is sufficient to store the records in a simple in-memory Java collection in the order they are added (e.g., an ArrayList would be a good choice).

Inventory must support the following commands. <X> represents an argument passed to the command, e.g., 5, i7-8700K, 18/2/1995, “Intel Corp”. Arguments will be given without angle brackets and will be separated by whitespace; arguments that contain whitespace must be double-quoted. For example, a company name that has a space. Arguments will not contain escaped whitespace (e.g. '\ ') or escaped quotes (e.g. \").

LOAD <filename.csv>

- Loads the contents of database from the named file, replacing the existing contents.
- The file is in a comma-separated value (CSV) format, with one entry per line.
- Each entry matches the following format exactly.
  <Name>,<Company>,<ReleaseDate>,<Quantity>
- Name and Company are strings. They may not contain commas or double quotes.
- ReleaseDate is given as MM/DD/YYYY.
- Quantity is a number.
- At the end of the file, there is exactly one blank line (i.e., the text of all entries is followed by a new-line character \n).
- You can assume the CSV file is always given in the correct format. Example of an entry in the CSV file:
  ```
  i7-8700K,Intel Corp,05/02/2017,12
  ```
- The following line should be written to the output file, where N is the number of records loaded:
  LOAD: OK <N>

STORE <filename.csv>

- Stores the contents of the database in the named file. Overwrites if the file already exists.
- The format must be exactly as described above.
- The following line should be written to the output file (not the csv file!), where N is the number of records stored:
  STORE: OK <N>

CLEAR

- Clears the contents of the database.
- The following line should be written to the output file:
  CLEAR: OK

ADD <Name> <Company> <ReleaseDate>

- Add the given entry to the database.
- Quantity is initialized to 0. An entry for an item must be added before BUY or SELL.
- Duplicate entries are not allowed, i.e., adding the same NAME/COMPANY combo twice results in an error. (This also applies if the items were loaded from a CSV file.)

- The following line should be written to the output file:
  ADD: OK <Name> <Company>

STATUS
- Lists all added items.
- STATUS will write the following to the output file, where M is the number of items.
- STATUS: OK <M>
  <Name1>,<Company1>,<ReleaseDate1>,<Quantity1>
  <Name2>,<Company2>,<ReleaseDate2>,<Quantity2>
  …
  <NameM>,<CompanyM>,<ReleaseDateM>,<QuantityM>

BUY <Name> <Company> <Quantity>
- Buys an item. BUY searches for the record identified by Name Company and adds the specified quantity to the old quantity.
- Quantity should be >= 1.
- The following line should be written to the output file:
  BUY: OK <Name> <Company> <UpdatedQuantity>

SELL <Name> <Company> <Quantity>
- Sells an item. SELL searches for the record identified by Name Company and subtracts the specified quantity to the old quantity.
- <Quantity> should be >= 1.
- Cannot sell a quantity more than the quantity in the inventory.
- The following line should be written to the output file:
  SELL: OK <Name> <Company> <UpdatedQuantity>

QUAN GREATER <Quantity>
QUAN FEWER <Quantity>
QUAN BETWEEN <Quantity1> <Quantity2>
- These commands list all items that are greater than, fewer than or in between the specified quantities.
- Note: GREATER means >, not >=. Similarly implement FEWER and BETWEEN by ignoring the equivalent case.
- The following will be written to the output file, where M is the number of items matching the request:
  QUAN: OK <M>
  <Name1>,<Company1>,<ReleaseDate1>,<Quantity1>
  <Name2>,<Company2>,<ReleaseDate2>,<Quantity2>
  …
  <NameM>,<CompanyM>,<ReleaseDateM>,<QuantityM>

SEARCH <Needle>

- Search for Needle as a substring in either Name or Company.
- The following lines will be written to the output file. The format for each record is the CSV format as described for LOAD. M is the number of matching records:
  SEARCH: OK <M>
  <Name1>,<Company1>,<ReleaseDate1>,<Quantity1>
  <Name2>,<Company2>,<ReleaseDate2>,<Quantity2>
  …
  <NameM>,<CompanyM>,<ReleaseDateM>,<QuantityM>

Error handling:

In case of errors for an individual command, Inventory will write the following line instead of OK. The command will not be executed.

<CMDNAME>: ERROR <ERRCODE>

For example:

ADD: ERROR DUPLICATE_ENTRY

You must handle at least the following errors (error code shown first):

1. INVALID_DATE: Invalid dates by value or format, for example: 11/31/2011, 14/23/2008, 11-31-2011, 5/5, applesauce.
2. WRONG_ARGUMENT_COUNT: Too few or too many arguments were given to a command.
   For example:
   ADD Intel 11/11/2011
   STATUS 5 10 15
   BUY "AMD" 2
3. CANNOT_BUY_BEFORE_ADD: When attempting to buy an item before adding it.
4. CANNOT_SELL_BEFORE_ADD: When attempting to sell an item before adding it.
5. INVALID_QUANTITY: When the quantity for BUY/SELL command is 0 or negative or doesn’t match the format.
   For example:
   BUY "Ryzen 5 2600X" AMD -44
   BUY "Ryzen 5 2600X" AMD 05/02/2017
6. CANNOT_SELL_QUANTITY_MORE_THAN_STOCK: When attempting to sell a greater quantity of an item than what’s available in the inventory.
7. FILE_NOT_FOUND: When LOAD command tries to load a .csv file that does not exist.
8. DUPLICATE_ENTRY: When an ADD command results in duplicate entries.

Example, adding the same item twice.
ADD "Ryzen 7 2700" AMD 05/06/2018
ADD "Ryzen 7 2700" AMD 05/09/2018
(Note: ReleaseDate is ignored for this check.)

9. UNKNOWN_COMMAND: When the command is invalid.
   For example:
   PING kernel.org
   QUAN LESS 5
   QUAN BTWEN 5
   QUAN BTWEN 5 10

Example execution:

```
in.txt                                  out.txt

ADD Intel 11/1/2011                     ADD: ERROR WRONG_ARGUMENT_COUNT
STATUS 5 10 15                          STATUS: ERROR WRONG_ARGUMENT_COUNT
BUY "Table Lamp" 2                      BUY: ERROR WRONG_ARGUMENT_COUNT
ADD "Ryzen 7 2700X" AMD  05/06/2018     ADD: OK "Ryzen 7 2700X" AMD
ADD "Ryzen 7 2700" AMD  05/06/2018      ADD: OK "Ryzen 7 2700" AMD
ADD "Ryzen 7 2700" AMD  05/06/2018      ADD: ERROR DUPLICATE_ENTRY
ADD "Ryzen 5 2600X" AMD 05/04/2018      ADD: OK "Ryzen 5 2600X" AMD
ADD "Ryzen 5 2400G" AMD 05/04/2018      ADD: OK "Ryzen 5 2400G" AMD
ADD i7-8700K "Intel Corp" 05/02/2017    ADD: OK i7-8700K "Intel Corp"
ADD i5-8600K "Intel Corp" 05/02/2017    ADD: OK i5-8600K "Intel Corp"
ADD i3-8350K "Intel Corp" 05/02/2017    ADD: OK i3-8350K "Intel Corp"
STATUS                                  STATUS: OK 7
                                        Ryzen 7 2700X,AMD,05/06/2018,0
                                        Ryzen 7 2700,AMD,05/06/2018,0
                                        Ryzen 5 2600X,AMD,05/04/2018,0
                                        Ryzen 5 2400G,AMD,05/04/2018,0
                                        i7-8700K,Intel Corp,05/02/2017,0
                                        i5-8600K,Intel Corp,05/02/2017,0
                                        i3-8350K,Intel Corp,05/02/2017,0
BUY "Ryzen 5 2600X" AMD -44             BUY: ERROR INVALID_QUANTITY
BUY "Ryzen 5 2600X" AMD 05/02/2017      BUY: ERROR INVALID_QUANTITY
BUY "Ryzen 5 2600X" AMD 55              BUY: OK "Ryzen 5 2600X" AMD 55
BUY "Ryzen 5 2600X" AMD 22              BUY: OK "Ryzen 5 2600X" AMD 77
SELL "Ryzen 5 2600X" AMD 22             SELL: OK "Ryzen 5 2600X" AMD 55
SELL "Ryzen 5 2600X" AMD 56             SELL: ERROR CANNOT_SELL_QUANTITY_MORE_THAN_STOCK
SELL "Ryzen 5 2600X" “AMD” 55           SELL: OK "Ryzen 5 2600X" AMD 0
BUY i7-8700K "Intel Corp" 12            BUY: OK i7-8700K "Intel Corp" 12
BUY i5-8600K "Intel Corp" 17            BUY: OK i5-8600K "Intel Corp" 17
BUY i3-8350K "Intel Corp" 24            BUY: OK i3-8350K "Intel Corp" 24
```

```
STATUS                        STATUS: OK 7
                              Ryzen 7 2700X,AMD,05/06/2018,0
                              Ryzen 7 2700,AMD,05/06/2018,0
                              Ryzen 5 2600X,AMD,05/04/2018,0
                              Ryzen 5 2400G,AMD,05/04/2018,0
                              i7-8700K,Intel Corp,05/02/2017,12
                              i5-8600K,Intel Corp,05/02/2017,17
                              i3-8350K,Intel Corp,05/02/2017,24
QUAN GREATER 5                QUAN: OK 3
                              i7-8700K,Intel Corp,05/02/2017,12
                              i5-8600K,Intel Corp,05/02/2017,17
                              i3-8350K,Intel Corp,05/02/2017,24
QUAN FEWER 15                 QUAN: OK 5
                              Ryzen 7 2700X,AMD,05/06/2018,0
                              Ryzen 7 2700,AMD,05/06/2018,0
                              Ryzen 5 2600X,AMD,05/04/2018,0
                              Ryzen 5 2400G,AMD,05/04/2018,0
                              i7-8700K,Intel Corp,05/02/2017,12
QUAN BETWEEN 10 18            QUAN: OK 2
                              i7-8700K,Intel Corp,05/02/2017,12
                              i5-8600K,Intel Corp,05/02/2017,17
STORE out.csv                 STORE: OK 7
```

out.csv contents in the above example:

```
Ryzen 7 2700X,AMD,05/06/2018,0
Ryzen 7 2700,AMD,05/06/2018,0
Ryzen 5 2600X,AMD,05/04/2018,0
Ryzen 5 2400G,AMD,05/04/2018,0
i7-8700K,Intel Corp,05/02/2017,12
i5-8600K,Intel Corp,05/02/2017,17
i3-8350K,Intel Corp,05/02/2017,24
```

Submission instructions:

Your Java files should be all grouped in a single root folder called hw1. Make sure to give your program a package name (which must start with hw1). For example, your files may be organized as follows:

hw1/	
hw1/Inventory.java	package hw1;
hw1/Database.java	package hw1;
hw1/util/Errors.java	package hw1.util;
hw1/util/CSV.java	package hw1.util;

Your code should be compilable and executable on a GNU/Linux Ubuntu system with Java 1.8 as follows (according to https://stackoverflow.com/questions/6623161/javac-option-to-compile-all-java-files-under-a-given-directory-recursively/8769536#8769536):

```
find -name "*.java" > sources.txt

javac @sources.txt

java hw1.Inventory
```

Note that the `java` command must be run from the folder containing your source package(s); if using the default IntelliJ project layout, that would be the `src` folder, which is also where **in.txt** needs to be. (When running your program through IntelliJ, **in.txt** will need to be in the folder one level above; that is, the folder containing src/ and out/. Note: IntelliJ license is free for students.)

Zip up the PDF from Part I, together with the source code for Part II, and save it in a file called **hw1_<myPID>.zip**. Submit the archive as Homework 1 on the Canvas page before the deadline.