CS 211: Computer Architecture, Summer 2019
Programming Assignment 1: Introduction to C

Instructor: Jay Lim

Introduction

This assignment is designed to give you some initial experience with programming in C, as well as compiling, linking, running, and debugging. Your task is to write 6 C programs. Each of them will test a portion of your knowledge about C programming. They are discussed below. Your program must follow the input-output guidelines listed in each section exactly with no additional or missing output.

No cheating or copying will be tolerated in this class. Your assignments will be automatically checked with plagiarism detection tools that are pretty powerful. Hence, you should not look at your friend's code. See CS department's academic integrity policy at: `http://www.cs.rutgers.edu/academic-integrity/introduction`

First: Array and Sorting

You have to write a program that will read an array from a file and sort the given array. You will return the array sorted with all odd numbers in ascending order at the front followed by all even numbers in descending order.

You can use any sorting algorithm you know or wish to use. However, you cannot use the library sort functions. You should write your own sort function.

Input-Output format: Your program will take a file name as an input. The first line in the file provides the total number of values in the array. The second line will contain a list of numbers separated by tabs. For example a sample input file "file1.txt" is:

```
file1.txt:
6
25 10 1 99 4 2
```

Your output will be a sorted lis of numbers, odd numbers in ascending order and then even numbers in descending order, each separated by tabs.

```
$./first file1.txt
1 25 99 10 4 2
```

We will not give you improperly formatted files. You can assume all your input files will be in proper format as above.

Hint: It may be helpful to move all odd numbers to the front of an array first and then sort them.

Second: Hash table

In this part, you will implement a hash table containing integers. The hash table has 10,000 buckets. An important part of a hash table is collision resolution. In this assignment, we want you to use chaining with a linked list to handle a collision. This means that if there is a collision at a particular bucket then you will maintain a linked list of all values stored at that bucket. For more information about chaining, see `http://research.cs.vt.edu/AVresearch/hashing/openhash.php`.

A hash table can be implemented in many ways in C. You must find a simple way to implement a hash table structure where you have easy access to the buckets through the hash function. As a reminder, a hash table is a structure that has a number of buckets for elements to "hash" into. You will determine where the element falls in the table using a hash function we describe below.

You must not do a linear search of the 10,000 element array. We will not award any credit for $O(n)$ time implementation of searches or insertions in the common case.

For this problem, you will be using the following **hash function**: key modulo the number of buckets.

Input format: This program takes a file name as argument from the command line. The file contains successive lines of input. Each line contains a character, either 'i' or 's', followed by a **tab** and then an integer. For each line that starts with 'i', your program should insert that number in the hash table if it is not present. If the line starts with a 's', your program should search the hash table for that value.

Output format: For each line in the input file, your program should print the status/result of that operation. For an insert, the program should print "inserted" if the value is inserted or "duplicate" if the value is already present. For a search, the program should print "present" or "absent" based on the outcome of the search. You can assume that the program inputs will have proper structure.

Example Execution: Let's assume we have a text file with the following contents:

```
file1.txt:
i 10
i 12
s 10
i 10
s 5
```

Then the result will be:

```
$./second file.txt
inserted
inserted
present
duplicate
absent
```

Third: Bit function

In this exercise, you have to write a program that will read a number followed by a series of bit operations from a file and perform the given oeprations sequentially on the number. The operations are as follows:
`set(x, n, v)`: sets the nth bit of the number x to v
`comp(x, n, v)`: sets the value of the nth bit of x to its complement (1 if 0 and 0 if 1)
`get(x, n, v)`: returns the value of the nth bit of the number x.

The least significant bit (LSB) is considered to be index 0.

For this exercise, you must use logical operations to implement the three functions. You are not allowed to use any arithmetic operations or math library functions.

Input format: Your program will take the file name as input. The first line in the file provides the value of the number x to be manipulated. This number should be considered an **unsigned short**. The following lines will contain the operations to manipulate the number. To simplify parsing, the format of the operations will always be the command name followed by 2 numbers, separated by **tabs**. For the `set(x, n, v)` command, the value of the second input number (v) will always be either 0 or 1. For the `comp(x, n)` and `get(x, n)` commands the value of the second input number will always be 0 and can be ignored. Note that the changes to x are cumulative, rather than each instruction operating independently on the original x.

Output format: Your output for `comp` and `set` commands will be the resulting value of the number x after each opeartion, each on a new line. For `get` commands, the output should be the requested bit's value.

Example Execution: For example, a sample input file `file1.txt` contains the following (except the annotation comments):

```
file1.txt:
5               # x = 5
get   0  0      # get(x, 0), ignoring second value (0)
comp  0  0      # comp(x, 0), ignoring second value (0)
set   1  1      # set(x, 1, 1)
```

The result of the sample run is:

```
$./third file1.txt
```

```
1
4
6
```

Fourth: One-Shot Learning

In this exercise, you will write a C program that implements simple "one-shot" machine learning algorithm for predicting house prices in your area.

There is significant hype and excitement around artificial intelligence (AI) and machine learning. CS 211 students will get a glimpse of AI/ML by implementing a simple machine learning algorithm to predict house prices based on historical data.

For example, the price of the house (y) can depend on certain attributes of the house: number of bedrooms (x_1), total size of the house (x_2), number of baths (x_3), and the year the house was built (x_4). Then, the price of the house can be computed by the following equation:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \tag{EQ1}$$

Given a house, we know the attributes of the house (i.e. x_1, x_2, x_3, x_4). However, we don't know the weights for these attributes: w_0, w_1, w_2, w_3, and w_4. *The goal of the machine learning algorithm in our context is to learn the weights for the attributes of the house from lots of training data.*

Let's say we have N examples in your training data set that provide the values of the attributes and the price. Let's say there are K attributes. We can represent the attributes from all the examples in the training data as a $N \times (K + 1)$ matrix as follows, which we call X:

$$\begin{bmatrix} 1 & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ 1 & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ 1 & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ 1 & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ & & \vdots & & \\ 1 & x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{bmatrix}$$

where n is $N - 1$. We can represent the prices of the house from the examples in the training data as a $N \times 1$ matrix, which we call Y:

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Similarly, we can represent the weights for each attributes as a $(K + 1) \times 1$ matrix, which we call W:

$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix}$$

The goal of our machine learning algorithm is to learn this matrix from the training data. Now, in the matrix notation, entire learning process can be represented by the following equation, where X, Y, W are matrices as described above.

$$XW = Y$$

Using the training data, we can learn the weights using the below equation:

$$W = (X^T X)^{-1} X^T Y$$

where X^T is the transpose of the matrix X and $(X^T X)^{-1}$ is the inverse of the matrix $X^T X$.

Your main task in this part of the assignment is to implement a program to read the training data and learn the weights for each of the attributes. You have to implement functions to multiply matrices, transpose matrices, and compute the inverse of the matrix. You will use the learned weights to predict the house prices for the examples in the test data set.

Want to learn more about One-shot Learning? The theory behind this learning is not important for the purposes of this assignment. The algorithm you are implementing is known as linear regression with least square error as the error measure. The matrix $(X^T X)^{-1} X^T$ is also known as the pseudo-inverse of matrix X. If you are curious, you can learn more about this algorithm at `https://www.youtube.com/watch?v=FIbVs5GbBlQ&hd=1`.

Computing the Inverse using Gauss-Jordan Elimination

To compute the weights above, your program has to compute the inverse of matrix. There are numerous methods to compute the inverse of a matrix. We want you to implement a **specific method for computing the inverse of a matrix known as Gauss-Jordan elimination**, which is described below. If you compute inverse using any other method, you will risk losing all points for ths part.

An inverse of a square matrix A is another square matrix B, since $AB = BA = I$, where I is the identity matrix.

Gauss-Jordan Elimination for computing inverses

Below, we give a sketch of Gauss-Jordan elimination method. Given a matrix A whose inverse needs to be computed, you create a new matrix A_{aug}, which is called the augmented matrix of A, by concatenating identity matrix with A as shown below.

Let's say matrix A, whose inverse you want to compute is shown below:

$$\begin{bmatrix} 1 & 2 & 4 \\ 1 & 6 & 7 \\ 1 & 3 & 2 \end{bmatrix}$$

The augmented matrix (A_{aug}) of A is:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 1 & 6 & 7 & 0 & 1 & 0 \\ 1 & 3 & 2 & 0 & 0 & 1 \end{bmatrix}$$

The augmented matrix essentially has the original matrix and the identity matrix. Next, we perform row operations on the augmented matrix so that the original matrix part of the augmented matrix turns into an identity matrix.

The valid row operations to compute the inverse (for this assignment) are:

- You can divide the entire row by a constant
- You can subtract a row by another row
- You can subtract a row by another row multipled by a constant

However, youa re not allowed to swap the rows. In the more complex Gauss-Jordan elimination method, you are allowed to swap the rows. For simplicity, we do not allow you to swap the rows.

Let's see this method with the above augmented matrix A_{aug}. Again, our goal is to transform A part of A_{aug} into an identity matrix. Since $A_{aug}[1][0] \neq 0$, we will subtract the first row from the second row to make $A_{aug}[1][0] = 0$. Hence, we perform the operation $R_1 = R_1 - R_0$ where R_1 and R_0 represent the second and nfirst row of the augmented matrix:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 4 & 3 & -1 & 1 & 0 \\ 1 & 3 & 2 & 0 & 0 & 1 \end{bmatrix}$$

Now, we want to make $A_{aug}[1][1] = 1$. Hence, we perform the operation $R_1 = R_1/4$. The augmented matrix after this operation is:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 1 & 3 & 2 & 0 & 0 & 1 \end{bmatrix}$$

Next, we want to make $A_{aug}[2][0] = 0$. Hence, we perform the operation $R_2 = R_2 - R_0$. The augmented matrix after this operation is:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 1 & -2 & -1 & 0 & 1 \end{bmatrix}$$

Next, we want to make $A_{aug}[2][1] = 0$. Hence, we perform the operation $R_2 = R_2 - R_1$. The resulting matrix is:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & \frac{11}{4} & -\frac{3}{4} & -\frac{1}{4} & 1 \end{bmatrix}$$

Now, we want to make $A_{aug}[2][2] = 1$. Hence, we perform the operation $R_3 = R_3 \times -\frac{4}{11}$:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & 0 \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{bmatrix}$$

So far we have moved down the matrix and zero'ed the lower triangle matrix to 0's and the diagonals of the original matrix A to 1's. Now, we will move up the matrix and transform the upper triangle portion of the original matrix A to 0.

As a first step of this phase, we want to make $A_{aug}[1][2] = 0$. Hence, we perform the operation $R_1 = R_1 - \frac{3}{4} \times R_2$:

$$\begin{bmatrix} 1 & 2 & 4 & 1 & 0 & 0 \\ 0 & 1 & 0 & -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{bmatrix}$$

Next, we want to make $A_{aug}[0][2] = 0$. Hence, we perform the operation $R_0 = R_0 - 4 \times R_2$:

$$\begin{bmatrix} 1 & 2 & 0 & \frac{1}{11} & -\frac{4}{11} & \frac{16}{11} \\ 0 & 1 & 0 & -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{bmatrix}$$

Next, we want to make $A_{aug}[0][1] = 0$. Hence, we perform the operation $R_0 = R_0 - 2 \times R_1$:

$$\begin{bmatrix} 1 & 2 & 0 & \frac{9}{11} & -\frac{8}{11} & \frac{10}{11} \\ 0 & 1 & 0 & -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ 0 & 0 & 1 & \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{bmatrix}$$

At this time, the A part of the augmented matrix is an identity matrix. The inverse of A matrix is the right half of the augmented matrix, which originally was the identity matrix:

$$\begin{bmatrix} \frac{9}{11} & -\frac{8}{11} & \frac{10}{11} \\ -\frac{5}{11} & \frac{2}{11} & \frac{3}{11} \\ \frac{3}{11} & \frac{1}{11} & -\frac{4}{11} \end{bmatrix}$$

Your program must implement how to compute the inverse of a matrix using Gaussian-Jordan elmination in order to perform one-shot learning.

Input/Output specification

Usage interface

Your program for this part will be executed as follows:

```
$./fourth <train-data-file-name> <test-data-file-name>
```

where `<train-data-file-name>` is the name of the training data file with attributes and price of the house. You can assume that the training data file will exist and that it is well structured. The `<test-data-file-name>` is the name of the test data file with attributes of the house. You have to predict the price of the house for each entry in the test data file.

Input Specification

The input to the program will be a training data file and a test data file.

Structure of the training data file

The first line in the training file will be an integer that provides the number of attributes (K) in the training set. The second line in the training data file will be an integer (N) providing the number of training examples in the training data set. The next N lines represent the N training examples. Each line for the example will be a list of comma-separated **double precision** floating point values. The first double precision value in the line represents the price of the house. The remaining K double precision values represent the values for the attributes of the house.

An example training data file (`train1.txt`) is shown below:

```
4
7
221900.000000,3.000000,1.000000,1180.000000,1955.000000
538000.000000,3.000000,2.250000,2570.000000,1951.000000
180000.000000,2.000000,1.000000,770.000000,1933.000000
604000.000000,4.000000,3.000000,1960.000000,1965.000000
510000.000000,3.000000,2.000000,1680.000000,1987.000000
1230000.000000,4.000000,4.500000,5420.000000,2001.000000
257500.000000,3.000000,2.250000,1715.000000,1995.000000
```

In the example above, there are 4 attributes and 7 training data examples. Each example has the price of the house followed by the values for the attributes. To illustrate, consider the training example below:

```
221900.000000,3.000000,1.000000,1180.000000,1955.000000
```

The price of the house is 221900.000000. The first attribute has value 3.000000, the second attribute has value 1.000000, the third attribute has value 1180.000000, and the fourth attribute has value 1955.000000.

Structure of the test data file

The first line in the training file will be an integer (M) that provides the number of test data points in the file. Each line will have K attributes. The value of K is defined in the training data file. Your goal is predict the price of house for each line in the test data file. The next M lines represent the M test points for which you have to predict the price of the house. Each line will be a list of comma-separated double precision floating point values. There will be K double precision values that represent the values for the attributes of the house.

An example test data file (`test1.txt`) is shown below:

```
2
3.000000,2.500000,3560.000000,1965.000000
2.000000,1.000000,1160.000000,1942.000000
```

It indicates that you have to predict the price of the house using your training data for 2 houses. The attributes of each house is listed in the subsequent lines.

Output specification

Your program should print the price of the house for each line in the test data file. Your program should not produce any additional output. If the price of the house is a fractional value, then your program should round it to the nearest integer, which you can accomplish with the following printf statement:

```
printf("%0.0lf\n", value);
```

where value is the price of the house and its type is **double** in C.

Your program should predict the price of the entry in the test data file by substituting the attributes and the weights (learned from the training data set) using (EQ1).

A sample output of the execution when you execute your program is shown below:

```
$./first train1.txt test1.txt
737861
203060
```

Structure of your submission folder

All files must be included in the pa1 folder. The pa1 directory in your tar file must contain 9 subdirectories, one each for each of the parts. The name of the directories should be named first through fourth (in lower case). Each directory should contain a c source file, a header file (if you use it) and a Makefile. For example, the subdirectory `first` will contain `first.c` `first.h` (if you create one) and `Makefile` (the names are case sensitive).

```
pa1
|- first
   |-- first.c
   |-- first.h (if used)
   |-- Makefile
|- second
   |-- second.c
   |-- second.h (if used)
   |-- Makefile
|- third
   |-- third.c
   |-- third.h (if used)
   |-- Makefile
|- fourth
   |-- fourth.c
   |-- fourth.h (if used)
   |-- Makefile
```

Submission

You have to e-submit the assignment using Sakai. Your submission should be a tar file named `pa1.tar`. To create this file, put everything that you are submitting into a directory (folder) named `pa1`. Then, `cd` into the directory containing `pa1` (that is, `pa1`'s parent directory) and run the following command:

```
tar cvf pa1.tar pa1
```

To check that you have correctly created the tar file, you should copy it (`pa1.tar`) into an empty directory and run the following command:

```
tar xvf pa1.tar
```

This should create a directory named `pa1` in the (previous) e mpty directory.

AutoGrader

We provide the AutoGrader to test your assignment. AutoGrader is provided as autograder.tar. Executing the following command will create the autograder folder.

```
$tar xvf autograder.tar
```

There are two modes available for testing your assignment with the AutoGrader.

First mode

Testing when you are writing code with a `pa1` folder
(1) Lets say you have a `pa1` folder with the directory structure as described in the assignment.
(2) Copy the folder to the directory of the autograder
(3)Run the autograder with the following command:

```
$python auto_grader.py
```

Second mode

This mode is to test your final submission (i.e. `pa1.tar`)
(1) Copy `pa1.tar` to the auto_grader directory
(2) Run the auto_grader with `pa1.tar` as the argument:

```
$python auto_grader.py pa1.tar
```

The auto grader will run your programs and print your score.

Scoring

The autograder will print out information about the compilation and the testing process. At the end, if your assignment is completely correct, the score will look something similar to what is shown below:

```
You scored
10.0 in first
10.0 in second
10.0 in third
20.0 in fourth
Your TOTAL SCORE = 50.0 /50
```

The point distribution may not necessarily be exactly the same as above. Your assigment will be graded for another 50 points with test cases not given you to.

Grading Guidelines

In this class, the most significant part of your grade will be based on programmatic checking of your program. That is, we will build the binary using Makefile and source code taht you submitted, and test the binary for correct functionality against a set of inputs. Thus:

- You should not see or use your friend's code either partially or fully. We will run state of the art plagiarism detectors. We will report everything caught by the tool to Office of Student Conduct.
- You should make sure that we can build your program by just running make.
- You should test your code as thoroughly as you can. For example, programs should not crash with memory errors.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result in up to 100% penalty. Be especially careful to not add extra whitespace or newlines. That means you will probably not get any credit if you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask on the discussion forum.