Assignment 5

CSCI 2132: Software Development

Due March 18, 2019

Assignments are due on the due date before 23:59. All assignments must be submitted electronically via the course SVN server. Plagiarism in assignment answers will not be tolerated. By submitting your answers to this assignment, you declare that your answers are your original work and that you did not use any sources for its preparation other than the class notes, the textbook, and ones explicitly acknowledged in your answers. Any suspected act of plagiarism will be reported to the Faculty's Academic Integrity Officer and possibly to the Senate Discipline Committee. The penalty for academic dishonesty may range from failing the course to expulsion from the university, in accordance with Dalhousie University's regulations regarding academic integrity.

General Instructions: How to Submit Your Work

You must submit your assignment answers electronically:

- Change into your subversion directory on bluenose: `cd ~/csci2132/svn/CSID`.
- Create a directory `a5` for the current assignment.
- Change into your assignment directory: `cd a5`.
- Create files inside the `a5` directory as instructed in the questions below and put them under Subversion control using `svn add <filename>`. **Only add the files you are asked to add!**
- Once you are done answering all questions in the assignment (or the ones that you are able to answer—hopefully all), the contents of your `a4` directory should look like this:

```
a5
├── luhn.c
└── matrixmult.c
```

  (You will also have executable programs and potentially some data files in this directory, but you should not add them to SVN.) Submit your work using `svn commit -m"Submit Assignment 5"`.

(Q1) Is Your Credit Card Number Valid? — 10 marks

In 1954, Hans Peter Luhn invented a simple checksum method that can be used to detect simple errors in various identification numbers. Today, this method, known as *Luhn's checksum*, is used to check for the validity of credit card numbers, Canadian social insurance numbers, and numerous others.

Consider a number $x = \sum_{i=0}^{n} x_i \cdot 10^i$, where $x_i \in \{0, \ldots, 9\}$ for all $0 \leq i \leq n$. In other words, $x_n x_{n-1} \ldots x_0$ is the sequence of x's decimal digits. For the number $x = 932$, for example, we have $n = 2$, $x_2 = 9$, $x_1 = 3$, and $x_0 = 2$.

To calculate Luhn's checksum, we transform the sequence $x = \langle x_n, x_{n-1}, \ldots, x_0 \rangle$ into another sequence $y = \langle y_n, y_{n-1}, \ldots, y_0 \rangle$ defined as

$$y_i = \begin{cases} x_i & \text{if } i \text{ is even} \\ 2x_i & \text{if } i \text{ is odd} \end{cases},$$

for all $0 \leq i \leq n$. Next, we construct a sequence $z = \langle z_n, z_{n-1}, \ldots, z_0 \rangle$ from $y = \langle y_n, y_{n-1}, \ldots, y_0 \rangle$ by defining

$$z_i = \begin{cases} y_i & \text{if } y_i < 10 \\ \lfloor y_i / 10 \rfloor + (y_i \bmod 10) & \text{if } y_i \geq 10 \end{cases},$$

for all $0 \leq i \leq n$. Finally, Luhn's checksum of $x = x_n x_{n-1} \ldots x_0$ is $L(x) = \sum_{i=0}^{n} z_i$. The number x is said to be *valid* if $L(x)$ is divisible by 10. Otherwise, x is *invalid*.

As an example, consider the number $x = 2264\,1658\,9022\,6421$. Then

$$\begin{aligned} x &= \langle 2, 2, 6, 4, 1, 6, 5, 8, 9, 0, 2, 2, 6, 4, 2, 1 \rangle, \\ y &= \langle 4, 2, 12, 4, 2, 6, 10, 8, 18, 0, 4, 2, 12, 4, 4, 1 \rangle, \text{ and} \\ z &= \langle 4, 2, 3, 4, 2, 6, 1, 8, 9, 0, 4, 2, 3, 4, 4, 1 \rangle. \end{aligned}$$

This gives $L(x) = 4+2+3+4+2+6+1+8+9+0+4+2+3+4+4+1 = 57$, so this number is invalid.

If we change the number to $x = 2264\,1658\,9022\,6424$, this sets $x_0 = y_0 = z_0 = 4$ and leaves all other z_i-values unchanged. Thus, $L(x) = 60$ in this case, that is, this number is valid.

Your task is to write a program `luhn.c` that reads a sequence of numbers from `stdin`, checks for each whether it is valid, and accordingly prints "valid" or "invalid" to `stdout` for each number.

Input format: The input consists of $n+1$ lines. The first line is the number n of remaining lines in the input. Each of the remaining n lines stores a number to be checked.

The numbers in the input can be arbitrarily long, that is, you must not assume that each number can be stored even in an `unsigned long int`. You may assume, however, that the checksum $L(x)$ of each number x in the input can be represented in an `unsigned long int`.

An example input looks like this:

```
3
00554
999
2264165890226424
```

Output format: For an input with $n+1$ lines (which consists of n numbers preceded by the count n of these numbers), the output your program produces must consist of n lines. The ith line should be "valid" if the ith number in the input is valid, and "invalid" if the ith number in the input is invalid.

For the input above, for example, your program should output:

```
valid
invalid
valid
```

(Q2) Matrix Multiplication **10 marks**

The main focus of this question is for you to practice using multi-dimensional arrays in C. Algorithmically, the problem is fairly straightforward. You should be familiar with the process of multiplying two matrices from an introductory linear algebra course. Given an $\ell \times m$ matrix A and an $m \times n$ matrix B,

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{\ell,1} & a_{\ell,2} & \cdots & a_{\ell,m} \end{pmatrix} \qquad B = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,n} \end{pmatrix},$$

our task is to compute an $\ell \times n$ matrix C,

$$C = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{\ell,1} & c_{\ell,2} & \cdots & c_{\ell,n} \end{pmatrix},$$

where

$$c_{i,j} = \sum_{k=1}^{m} a_{i,k} b_{k,j}, \text{ for all } 1 \leq i \leq \ell \text{ and } 1 \leq j \leq n.$$

Your program should read the matrices A and B from `stdin` and write the matrix C to `stdout`. Implement your program in a file `matrixmult.c` and add it to SVN. The following are the requirements on the format of the input your program must be able to consume and of the output your program must produce.

Input format: The input of your program consists of a file split into three parts. The first and second part are separated by an empty line, as are the second and third part of the input.

The first part consists of a single line containing the three integers ℓ, m, and n, separated by spaces.

The second part represents the matrix A. This part has ℓ lines. Each line has m floating point numbers on it. The jth number on the ith line is the value $a_{i,j}$.

The third part represents the matrix B. This part has m lines. Each line has n floating point numbers on it. The jth number on the ith line is the value $b_{i,j}$.

Each floating point number in Parts 2 and 3 are represented with two digits after the decimal point.

Here's an example input:

```
3 2 4

3.00 4.50
1.25 -1.00
2.00 -3.00

1.00 1.00 -2.00 4.00
5.25 3.25 -2.25 1.00
```

This represents the input matrices

$$A = \begin{pmatrix} 3 & 4.5 \\ 1.25 & -1 \\ 2 & -3 \end{pmatrix} \text{ and } B = \begin{pmatrix} 1 & 1 & -2 & 4 \\ 5.25 & 3.25 & -2.25 & 1 \end{pmatrix}.$$

Output format: The output of your program must consist of two parts, **not** separated by any blank lines. The first part consists of a single line with two integers on it, ℓ and n. The second part consists of ℓ lines. Each line has n floating point numbers on it. The jth number on the ith line is the value $c_{i,j}$. Each value must be represented with two digits after the decimal point.

The output for the above sample input should look like this:

```
3 4
26.63 17.63 -16.12 16.50
-4.00 -2.00 -0.25 4.00
-13.75 -7.75 2.75 5.00
```

This represents the output matrix

$$C = A \times B = \begin{pmatrix} 26.625 & 17.625 & -16.125 & 16.5 \\ -4 & -2 & -0.25 & 4 \\ -13.75 & -7.75 & 2.75 & 5 \end{pmatrix}.$$

(Due to the requirement that each number should be represented with two digits of precision after the decimal point, the first three numbers on the first line are rounded. You do not have to worry about implementing a particular rounding rule. Simply produce the output that `printf`'s `%.2f` format specifier produces.)