Data Structure and Algorithm

Homework #0 (updated: 2/22 11am)

Due: 5pm, Thursday, March 1, 2012

TA email: dsa1@csie.ntu.edu.tw

=== Homework submission instructions ===

- Submit your source code, a shell script to compile the source, and a brief documentation to the SVN server (katrina.csie.ntu.edu.tw). You should create a new folder "hw0" and put these three files in it.
- The filenames of the source code, the shell script, and the documentation file should be "mul.c", "compile.sh", and "report.txt", respectively; you will get some penalties in your grade if your submission do not follow the naming rule.
- The documentation file should be in plain text format (.txt file). In the documentation file you should explain how your code works, and anything you would like to convey to the TAs.
- For more information about the SVN server, please see the slide from the course website. (`http://www.csie.ntu.edu.tw/~hsinmu/courses/_media/dsa_12spring:svn_introduction.pdf`)
- You can utilize the sample shell script we provide on the course website to compile the source.
- No late submission of the homework will be given any score (for that portion).

Problem 1. (10% of Homework #1)

Matrix multiplication is an operation that takes 2 matrices A and B as inputs, and outputs another matrix C. If A is an m-by-n matrix and B is an n-by-l matrix, the result of their multiplication is an m-by-l matrix defined only if the number of columns in the first matrix A equals to the number of rows in the second matrix B.

The definition of matrix multiplication can be found on Wikipedia:

`http://en.wikipedia.org/wiki/Matrix_multiplication`

Example 1. $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, AB = \begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$

Write a program to calculate the result of the multiplication, $C = AB$, of the given two matrices A and B. The input has the following format (download a sample of the input, "hw0_input_example", from the course website):

$m \quad n \leftarrow$ Number of row and column of A

$a_{11}\ a_{12}\ \ldots\ \ldots\ a_{1n} \leftarrow$ the n numbers of the first row

$a_{21}\ a_{22}\ \ldots\ \ldots\ a_{2n} \leftarrow$ the n numbers of the second row

$\vdots$

$a_{m1}\ a_{m2}\ \ldots\ \ldots\ a_{mn} \leftarrow$ the n numbers of the m-th row

$r \quad s \leftarrow$ Number of row and column of B

$b_{11}\ b_{12}\ \ldots\ \ldots\ b_{1s} \leftarrow$ the s numbers of the first row

$b_{21}\ b_{22}\ \ldots\ \ldots\ b_{2s} \leftarrow$ the s numbers of the second row

$\vdots$

$b_{r1}\ b_{r2}\ \ldots\ \ldots\ b_{rs} \leftarrow$ the s numbers of the r-th row

If the multiplication result is valid (e.g. $n = r$ in the example above), output the matrix C in the following format:

$c_{11}\ c_{12}\ \ldots\ \ldots\ c_{1s} \leftarrow$ the s numbers of the first row

$c_{21}\ c_{22}\ \ldots\ \ldots\ c_{2s} \leftarrow$ the s numbers of the second row

$\vdots$

$c_{m1}\ c_{m2}\ \ldots\ \ldots\ c_{ms} \leftarrow$ the s numbers of the m-th row

Otherwise, output an one-line message "error".

You can utilize the following assumptions:

1. Every entry of the matrix from input can be stored in a (16-bit) integer. (*Hint: note that there is no similar assumption for the output.*)

2. The dimension m, n, r, s could be any value (but also can be stored in a 32-bit integer). This means you HAVE TO dynamically allocate the memory to store the matrix (*malloc,.....*).

3. If the result of the multiplication is valid, $1 \leq m \times n \times s, m \times n, r \times s \leq 100000000$

4. Take the input from the standard input device (stdin).

Sol:

Implementations Of Two-dimensional Array:

In this problem, you have to dynamically allocate 2-D arrays to store matrices. There are two possible ways to implement a 2-D array with m rows and n columns:

1. Using a 1-D array:

   To implement a 2-D array(A[m][n]), we can allocate a 1-D array with m*n elements (Arr[m*n]). Then, we can access an element in the i-th row and the j-th column by simple arithmetics: the element A[i][j] will be stored in Arr[(i-1)*m+j].

2. By array of arrays:

   At first, we allocate an m-element 1-D array of pointers for the "heads" of each row. Then, we allocate an n-element 1-D array starting for each "head". The code typically looks like the following:

```
DataType** Arr = (DataType**)malloc( m*sizeof(DataType*) );
for(i=0;i<m;++i)
    Arr[i] = (DataType*)malloc( n*sizeof(DataType) );
//Use Arr[i][j] to access A[i][j]
```

Common Problems

- Data type to store the entries of the matrice

  If we do not use a proper data type to store the entries of the matrices, an **overflow** could occur. From the assumptions in this problem, we can roughly estimate the maximum value of an entry in the resulting matrix after a multiplication. Consider

a 1-by-10000000 matrix multiplied by a 10000000-by-1 matrix, the maximum value of the only entry in the result matrix is approximately

$$2^{16} \times 2^{16} \times 100000000 < 2^{64} \quad .$$

Therefore, the `long long` type (64-bit) is sufficient. Note that the following attempt of type conversion will not work:

```
short a, b;
long long c;
...
c = a * b;
```

The reason is that the result of `a * b` is first calculated in the `short` type(so overflow occurs!) and then assigned to `c`. To fix the problem, we should instead use:

```
c = (long long)a * (long long)b;
```

Since we have converted the type of `a` and `b`before the multiplication, the value of `a * b` is calculated in the `long long` type, which is what we want.
In fact, this problem can be avoided if you can ensure that your program can work correctly even with the input which is at the "boundary" of the problem assumption (we usually call this "the boundary case"). **The ability to test the correctness of your program by yourself is a very important skill for all levels of programmers.**

- Be sure that your compile.sh works

  The TAs grade your programs on a Linux system. Please make sure that your program can be compiled by using your compile.sh. You can test it on the 217 workstations. Successful compilation in your IDE guarantees NOTHING! A very common problem is the "for loop initial declaration", such as:

  ```
  for(int i=0;i<n;i++)
  ```

. In this case, you need an additional option `-std=c99` in your compilation command, which is not included in the compile.sh we provided on the course website. (In fact, we strongly discourage having variable declarations in the middle of a function. This is not standard practice of C programming and we recommend that you have all your declarations at the beginning and therefore is not supported by the compiler be default.)

Many CSIE courses that you will take in the future require you to write programs in the UNIX environment (Linux is one of the those), so it is recommended that you get familiar with it as soon as possible.