CSCI 2500 — Computer Organization
Homework 1 (document version 1.0)
Matrix Multiplication using C

- This homework is due in Submitty by 11:59PM EST on Thursday, January 27, 2022
- You can use at most three late days on this assignment
- This homework is to be done individually, so **do not share your code with anyone else**
- You **must** use C for this assignment, and all submitted code **must** successfully compile via `gcc` with no warning messages when the `-Wall` (i.e., warn all) compiler option is used; we will also use `-Werror`, which will treat all warnings as critical errors
- All submitted code **must** successfully compile and run on Submitty, which uses Ubuntu v20.04.3 LTS and `gcc` version 9.3.0 (`Ubuntu 9.3.0-17ubuntu1~20.04`)

Homework specifications

In this first homework, you will implement matrix multiplication in C. If you need a refresher in how matrix multiplication works, look in a math textbook or check out Wikipedia. Note that we will revisit this topic in the future, but for now you will implement your algorithm in C.

Command-line arguments

Four command-line arguments are required (i.e., `argv[1]` through `argv[4]`). The first two command-line arguments specify how many rows and columns there are in the first matrix, while the second two command-line arguments specify how many rows and columns there are in the second matrix. To properly translate the command-line arguments, use the `atoi()` function, which you can learn more about by reading the `man` page.

Once the pair of matrix dimensions are identified (and validated), prompt the user to enter the non-negative integer values for each matrix. Use `scanf()` to read in these values. And to ensure only non-negative values are used, abort program execution with an error if a negative value is encountered in the input.

Given the two matrices, implement your matrix multiplication algorithm to produce a third matrix, which you display to the user, then exit.

Static vs dynamic memory allocation

For this assignment, you are certainly allowed to use static memory allocation for primitive data types and pointers; however, all arrays must be dynamically allocated. More specifically, the matrices that you work with must be dynamically allocated via `calloc()`. You are therefore given starter code (on page 4) that includes a complete reusable `matrix_alloc()` function.

Required program output

Example program executions are shown on the next page and should help you better understand how your program should work, as well as how you can begin to test your code.

Be sure to match the output formatting exactly as shown. In particular, when displaying a matrix, each line must start with `'['` and end with `']'` (as shown below and on the next page). Be careful not to include any extraneous spaces after the `']'` character. Further, right justify and vertically line up your columns as illustrated below, always using two spaces to separate each column.

```
[12  34  55567]
[ 8   9    123]
[45  67      8]
[ 9  10     11]
```

To achieve the above formatting, creating a reusable function or set of functions is highly recommended. Also, use the formatting features of `printf()`.

Finally, note that your program must return either `EXIT_SUCCESS` or `EXIT_FAILURE`, which you can verify via `echo $?` in the terminal.

Example program executions

Example program executions are shown below. Note that input values may have any number of spaces, tabs, and/or newline characters between them, so you should use `scanf()` in a loop to read in each value.

```
bash$ ./a.out 2 3 3 1
Please enter non-negative integer values for the first matrix (2x3):
1 2 3
4 5 6
Please enter non-negative integer values for the second matrix (3x1):
7
8
9

RESULTS:
[1  2  3] multiplied by [7]
[4  5  6]               [8]
                        [9]
equals [ 50]
       [122]
bash$ echo $?
0
bash$ ./a.out 4 5 6 7
ERROR: Invalid inputs!
bash$ echo $?
1
bash$ ./a.out 2 2 2 2
Please enter non-negative values for the first matrix (2x2):
1 2
3 4
Please enter non-negative values for the second matrix (2x2):
5 6
7 8

RESULTS:
[1  2] multiplied by [5  6]
[3  4]               [7  8]
equals [19  22]
       [43  50]
bash$ echo $?
0
```

Code Template

Below is a good starting point for your code:

```
/* hw1.c */
/* NAME: <your-name-here> */
/* COMPILE: gcc -Wall -Werror -Wvla hw1.c -lm */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>

/* dynamically allocate memory for rows x cols matrix of integers */
int ** matrix_alloc( int rows, int cols )
{
  int ** matrix = calloc( rows, sizeof( int * ) );

  for ( int r = 0 ; r < rows ; r++ )
  {
    matrix[r] = calloc( cols, sizeof( int ) );
  }

  return matrix;
}

int main( int argc, char * argv[] )
{
  /* Ensure we have the correct number of command-line arguments */
  if ( argc != 5 )
  {
    fprintf( stderr, "ERROR: Invalid inputs!\n" );
    return EXIT_FAILURE;
  }




  return EXIT_SUCCESS;
}
```

Error handling

In general, if an error is encountered, display a meaningful error message on `stderr` by using either `perror()` or `fprintf( stderr, "..." )`, then aborting further program execution. Only use `perror()` if the given library or system call sets the global `errno` variable.

Error messages must be one line only and use the following format:

```
ERROR: <error-text-here>
```

Submission instructions

Before you submit your code, be sure that you have clearly commented your code—this should not be an afterthought! Further, your code should have a clear and logical organization. In general, each function should easily fit within a single window and have a clear (and clearly documented) purpose; if not, separate into multiple functions. Variable names and function names should be intuitive and meaningful. Be consistent in your indentation.

To submit your assignment (and also perform final testing of your code), please use Submitty.

Note that this assignment will be available on Submitty a minimum of three days before the due date. There will be hidden test cases.

Also note that output to standard output (`stdout`) is buffered. To disable buffered output on Submitty, use `setvbuf()` as follows:

```
setvbuf( stdout, NULL, _IONBF, 0 );
```

You would not generally do this in practice, as this can substantially slow down your program, but to ensure you see output even on a segmentation fault on Submitty, this is a good technique to use.