Assignment 3

Takunda Takaindisa

March 2024

1 Appendix

1.1 functions.c

Listing 1: functions.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
// Function to generate a random matrix
void generateMatrix(int rows, int cols, double matrix[rows][cols]) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            // Generate random numbers between -10 and 10
            matrix[i][j] = -10.0 + ((double)rand() / RAND_MAX) * 20.0;
        }
    }
}
void printMatrix(int rows, int cols, double matrix[rows][cols]) {
    for (int i = 0; i < rows; i++) { //iterate through the rows of a matrix
        for (int j = 0; j < cols; j++) { //iterate through the columns of a matrix
            printf("%.2f\t", matrix[i][j]); //print the value at that specific row/
                column index with 2 decimal places
        }
        printf("\n");
    }
}
void addMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2][M2
    ], double answer[N1][M1]) {
    //two matrices must have the same dimensions fot them to be added
    //rows of marix 1 must equal rows of matrix 2  and likewise for columns
    if (N1 != N2 || M1 != M2){
        printf("Addition between these two matrices can't be done as they don't
            have the same dimensions.\n");
    }else{

    for (int row=0; row < N1; row++){ //iterate through rows
        for (int column =0; column< M1; column ++){ //iterate through columns
            answer[row][column]=A[row][column]+B[row][column]; //add corresponding
                row,column values between the two matrices
        }
    }
}
}
```

```
void subtractMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2
    ][M2], double answer[N1][M1]) {
    //two matrices must have the same dimensions fot them to be subtracted
    //rows of marix 1 must equal rows of matrix 2  and likewise for columns
    if (N1 != N2 || M1 !=M2){
        printf("Subtraction between these two matrices can't be done as they don't
           have the same dimensions.\n");
    }else{

    for (int row=0; row < N1; row++){ //iterate throgh rows
        for (int column =0; column< M1; column ++){ //iterate through columns
            answer[row][column]=A[row][column]-B[row][column]; //subtract
               corresponding row,column values between the two matrices
        }
    }
}
}
void multiplyMatrices(int N1, int M1, double A[N1][M1], int N2, int M2, double B[N2
    ][M2], double answer[N1][M2]) {
    if ( M1 != N2){
        //rule for multiplication:only when (m*n)*(n*p)
        printf("Matrix multiplication can't be done between these two matrices. The
           number of columns in matrix A must equal the number of rows in matrix B
           .\n");
    }else{

    for (int row =0; row<N1; row++){ //iterate through rows
        for (int column = 0; column <M2; column ++){ //iterate through columns
            answer[row][column]=0;
            for (int index = 0; index<M1; index++){ //iterate over common index
                when row of A= column of B
                answer[row][column] += A[row][index]*B[index][column];
            }
        }
    }
}
}
// Function to solve a system of linear equations Ax = B using Gaussian elimination
//The following code was inspired by chatGPT
void solveLinearSystem(int N1, int M1, double A[N1][M1], int N2, int M2, double B[
    N2][M2], double x[N1][M2])
{
    // Make the coefficient matrix using A and B
    double coefficient[N1][M1+1];
    //coefficient matric is made by combining rows of A dn B
    for (int row = 0; row < N1; row++)
    {
        //Ccopyelements from A to coefficient
        for (int rowvar = 0; rowvar < M1; rowvar++)
        {
            coefficient[row][rowvar] = A[row][rowvar];
        }
    }
    // same for B
    for (int row = 0; row < N1; row++)
    {
        coefficient[row][M1] = B[row][0];
    }

    // diagonals can't be 0, will lead to div by 0 error
    for(int row = 0 ; row < N1; row++)
    {
        // Check if the diagonal element is zero
        double buffer = coefficient[row][row];
        for(int rowvar = 0; rowvar < M1+1; rowvar++)
        {
            if(buffer == 0)
            {
                // swap rows with row below if diag element =0
                for(int colvar = row+1; colvar < N1; colvar++)
                {
                    if(coefficient[colvar][row] != 0)
                    {
                        // Swap rows
                        for(int swap = 0; swap < M1+1; swap++)
                        {
                            double holder = coefficient[row][swap];
                            coefficient[row][swap] = coefficient[colvar][swap];
                            coefficient[colvar][swap] = holder;
                        }
                        break;
                    }
                }
            }
        }
    }

    // make coefficient upper triangular
    for(int row = 0; row < N1; row++)
    {
        for(int rowvar = 0; rowvar < N1; rowvar++)
        {
            if(row != rowvar)
            {
                // Compute the factor which will subtract rows
                double scale = coefficient[rowvar][row] / coefficient[row][row];
                // Subtract current row multiplied by factor from other rows
                for(int colvar = 0; colvar <= M1; colvar++)
                {
                    coefficient[rowvar][colvar] -= scale * coefficient[row][colvar
                        ];
                }
            }
        }

        // Normalize pivot row/divide by pivot
        double pRow = coefficient[row][row];
        for(int colvar = 0; colvar <= M1; colvar++)
        {
            coefficient[row][colvar] /= pRow;
        }
    }
    // solution is last column in coefficient
    for(int row = 0; row < N1; row++)
    {
        x[row][0] = coefficient[row][M1];
    }
}
```

1.2 math_matrix.c

```
// CODE: include necessary library(s)
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include "functions.h"

int main(int argc, char *argv[]) {
    srand(time(NULL)+3);
    // Check if the number of arguments is correct
    if (argc < 6 || argc > 7) {
        printf("Usage: %s nrows_A ncols_A nrows_B ncols_B <operation> [print]\n",
            argv[0]);
        return 1;
    }

    //  Terminal arguments
    //first arg is the executable file itsel
    int nrows_A=atoi(argv[1]); //second arg (no.rows of A)
    int ncols_A=atoi(argv[2]); //third arg (no.cols of A)
    int nrows_B=atoi(argv[3]); //fourth arg (no.rows of B)
    int ncols_B=atoi(argv[4]); //fifth arg (no.cols of B)
    char *operation = argv[5]; //6th arg (operation), array of string so we have to
         use pointer
    /*
    if no.arguments =7 and 7th argument is 'print' then set print to true,
        otherwise false*/
   int print = 0; // Initialize print variable to 0 (false), meaning not print

    if (argc == 7 && strcmp(argv[6], "print") == 0) {
        print = 1; // Set print to 1 (true) if count is 7 and argv[6](item at 7) is
             "print" meaning the answer should beprinted
    }
    double A[nrows_A][ncols_A];
    double B[nrows_B][ncols_B];
    double answer[nrows_A][ncols_B];
    double x[nrows_A][ncols_B];

   srand(time(NULL));
   generateMatrix(nrows_A, ncols_A, A);

// offset seed fot B so matrice produced arent the same
   srand(time(NULL)+63636);
   generateMatrix(nrows_B, ncols_B, B);
```

```
  if (strcmp(operation, "add") == 0) { //compare string arg to add
 if (nrows_A != nrows_B || ncols_A != ncols_B) {
     printf("Addition between these two matrices can't be done as they don't
         have the same dimensions.\n"); //print add answer if T, message if F
     return 1;
 }
 addMatrices(nrows_A, ncols_A, A, nrows_B, ncols_B, B, answer);
 }   else if (strcmp(operation, "subtract") == 0) { //same for subtract
 if (nrows_A != nrows_B || ncols_A != ncols_B) {
     printf("Subtraction between these two matrices can't be done as they don't
         have the same dimensions.\n");
     return 1;
 }
 subtractMatrices(nrows_A, ncols_A, A, nrows_B, ncols_B, B, answer);
 } else if (strcmp(operation, "multiply") == 0) { //same for mult
 if (ncols_A != nrows_B) {
     printf("Multiplication between these two matrices can't be done as the
         number of columns of the first matrix doesn't match the number of rows
         of the second matrix.\n");
     return 1;
 }
 multiplyMatrices(nrows_A, ncols_A, A, nrows_B, ncols_B, B, answer);
 } else if (strcmp(operation, "solve") == 0) { //same for solve
     solveLinearSystem(nrows_A, ncols_A, A, nrows_B, ncols_B, B, x);
 } else {
 printf("That operation isn't recognised, please use add, subtract, multiply, or
     solve.\n");
 return 1;
}


 // print matrices and the answer if print is at the end of the terminal inpt
 if (print) {
     printf("Matrix A:\n");
     printMatrix(nrows_A, ncols_A, A);
     printf("Matrix B:\n");
     printMatrix(nrows_B, ncols_B, B);
    if (strcmp(operation, "add") == 0) { //string compare to add true
     printf("Result of A + B:\n");
     } else if (strcmp(operation, "subtract") == 0) { //string compare to
         subtract true
         printf("Result of A - B:\n");
     } else if (strcmp(operation, "multiply") == 0) { //string compare to
         multiply true
         printf("Result of A * B:\n");
     } else if (strcmp(operation, "solve") == 0) { //string compare to solve
         true
         printf("Result of x=B/A:\n");
     } else {
         printf("That operation isn't recognised, please use add , subtract ,
             multiply or solve.\n");
         return 1;
     }

     // Print the actual answer based on chosen operation: //var for solve is x,
         but for others it's answer
```

```
        if (strcmp(operation,"solve")==0){
            printMatrix(nrows_A, ncols_B, x);
        }
        else{
             printMatrix(nrows_A, ncols_B, answer);
        }
        return 0;
}}
```

1.3 makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99
TARGET = math_matrix
SRCS = math_matrix.c functions.c
OBJS = $(SRCS:.c=.o)

$(TARGET): $(OBJS)
        $(CC) $(CFLAGS) -o $@ $^

%.o: %.c
        $(CC) $(CFLAGS) -c $< -o $@

clean:
        rm -f $(TARGET) $(OBJS)
```

2 Solve implementation

The implementation of the function was inspired by ChatGPT.The way it works is that function first starts by making the augmented matrix by combining the matrices A and B using for loops to copy the row elements from A into a new matrix called coefficient and then doing the same for B, as the elements of B are copied to the last column of coefficient. If any of the diagonal elements are, then eventually all the other elements will be divided by 0 when making the first non-zero element in each row, this will then be impossible to solve so instead, if a diagonal element is zero, it swaps with a row below it that doesn't have its diagonal element as 0. The program then iterates over the rows of coefficient and subtracts the current row multiplied by a scale from other rows to eliminate the doubles in the lower triangle, i.e turn them to 0. After the reduced row operations, the function divides all elements of the row by the pivot element to make sure that the pivot element is actually 1. After all these operations, the solution to the homogenous system is the last column vector the solved augmented matrix, so the program will assign this vector as x and that concludes the operations.

3 Segmentation Fault

I used lldb to debug my code after using a large value of 1000 for all dimensions of the matrices. The code was as follows:
gcc -g -o debug math_matrix.c functions.c
lldb ./debug
and then I set up breakpoints at lines 22, 30,31,32,33 and 34. I then typed in run 1000 1000 1000 1000 add
The first stop was at line 22 which was: char *operation = argv[5];
The reason it stopped was due to a breakpoint I set: '* thread 1, queue = 'com.apple.main-thread',stop reason = breakpoint 1.1 frame 0: debug'main(argc=6, argv=) at math_matrix.c:22:23'
As this wasnt due to an error, I then typed in continue and the program next stopped at line 30, which was: double A[nrows_A][ncols_A];
The reason it stopped was also due to a breakpoint so I continued again and it stopped at line 31. The stop reason was EXC_BAD_ACCESS, which means that the error occured at line 31 which was:

double B[nrows_B][ncols_B];.
This means that the breakpoint didn't trigger and the segmentation fault is being caused by the initialization of the matrix B. When I run ulimit -s in my terminal, it reveals that my stack size is 8192 kB which is the same as 8Mb, however the memory required to store a matrix of doubles with dimensions 1000 by 1000 is 8,000,000 bytes which is slightly less that 8 Mb, which means that there is enough memory in my stack to store the first matrix A, but there is not enough memory left over to store the second matrix B, which then results in a stack overflow. This led to the segmentation fault being displayed when I tried to do the add operation with very large matrices.