

We can use static two dimensional array of 4x4 like that int matrix[4][4];

But I choose to work with dynamic array , much more fun

First of all we need to know how to allocate a dynamic multidimensional array:

We do know how to work with static dim array like int matrix[4][4]

But for this exercise we must work with unknown size:

int rows = 3, columns = 3;

/\* initialize rows and columns to the desired value \*/

int \*\*matrix = (int\*\*)malloc(rows\*sizeof(int\*));

for (int i = 0; i<rows; i++)

{

matrix[i] = (int\*)malloc(columns \* sizeof(int));

}

for (int i = 0; i<rows; i++)

{

free(matrix[i]);

}

free(matrix);

Don’t forget to free the internal buffers and the pointers it self.

The stages to create this kind of project are first make sure all our building blocks are working.

I.E, the matrices sum , sun , multiply and more

Once those functions are ready and debugged, we can start with the user interface

Let’s create some helper functions before we start:

int \*\*allocateMatrix(int matrixDim)

{

int \*\*matrix = (int\*\*)malloc(matrixDim\*sizeof(int\*));

for (int i = 0; i<matrixDim; i++)

{

matrix[i] = (int\*)malloc(matrixDim \* sizeof(int));

}

return matrix;

}

int \*\*matrix\_sum(int \*\*matrixA, int \*\*matrixB, int \*\* matOutput , int matrixDim)

{

for (int i = 0; i < matrixDim; i++)

for (int j = 0; j < matrixDim; j++)

{

matOutput[i][j] = matrixA[i][j] + matrixB[i][j];

}

return matOutput;

}

This function just put a value in all the array to help us check the results.

void init\_matrix(int \*\*A, int dimSize, int value)

{

for (int i = 0; i < dimSize; i++)

for (int j = 0; j < dimSize; j++)

{

A[i][j] = value;

}

}

void print\_matrix(int \*\*S, int size)

{

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

printf("%d ", S[i][j]);

}

printf("\n");

}

}

int \*\*matrix\_sub(int \*\*matrixA, int \*\*matrixB, int matrixDim)

{

int \*\* matOutput = allocateMatrix(matrixDim);

for (int i = 0; i < matrixDim; i++)

for (int j = 0; j < matrixDim; j++)

{

matOutput[i][j] = matrixA[i][j] - matrixB[i][j];

}

return matOutput;

}

In all the functions, I choose to work with the array pointer as a input parameter

Out main function:

int main(int argc, char\* argv[])

{

int \*\* A = allocateMatrix(3);

int \*\* B = allocateMatrix(3);

int \*\* S = allocateMatrix(3);

init\_matrix(A, 3, 1);

init\_matrix(B, 3, 1);

S = matrix\_sum(A, B, S, 3);

print\_matrix(S, 3);

matrix\_sub(A, B, S, 3);

print\_matrix(S, 3);

return 0;

}

We can use the return value from the function or just use void, does not matther much

Don’t forget to free all :

void free\_matrix(int \*\*A, int size)

{

for (int i = 0; i < size; i++)

{

free(A[i]);

}

free(A);

}

free\_matrix(A,3);

free\_matrix(B,3);

free\_matrix(S,3);

becaouse we need to follow the instructions we need to declare 6 matrices:

MAT\_A, MAT\_B, MAT\_C, MAT\_D, MAT\_E,MAT\_F

In size of 4x4:

int \*\* MAT\_A = allocateMatrix(4);

int \*\* MAT\_B = allocateMatrix(4);

int \*\* MAT\_C = allocateMatrix(4);

int \*\* MAT\_D = allocateMatrix(4);

int \*\* MAT\_E = allocateMatrix(4);

int \*\* MAT\_F = allocateMatrix(4);

Multiplication

Matrix multiplacation require more work than just add the same I,j

a00 a01 a02 b00 b01 b02

a10 a11 a12 b10 b11 b12

a20 a21 a22 b20 b21 b22

(a00\*b00 + a01 \*b10 + a02\*b20) (a00\*b01 + a01\*b11 + a02\*b21)

(a10\*b00 + a11\*b10 + a12\*b20) (a10\*b01+a11\*b11+a12\*b21)

(a20\*b00 + a21\*b10 + a22\*b20)

And so on..

void matrix\_mult(int \*\*A, int \*\*B, int \*\*S, int size)

{

int count;

for (int i = 0; i < size; i++)

{

count = 0;

for (int r = 0; r < size; r++)

{

S[r][i] = 0;

for (int j = 0; j < size; j++)

S[r][i] += A[r][j] \* B[j][i];

}

}

}

To easy check the mult I created an identity matrix function:

void init\_identity(int \*\* matrix, int size)

{

init\_matrix(matrix, size, 0);

for (int i = 0; i < size; i++)

{

matrix[i][i] = 1;

}

}

AI = A;

Checking:

int \*\*MAT\_A = allocateMatrix(4);

int \*\*MAT\_B = allocateMatrix(4);

int \*\*R = allocateMatrix(4);

read\_mat(MAT\_A, 4);

print\_matrix(MAT\_A, 4);

init\_identity(MAT\_B, 4);

print\_matrix(MAT\_B, 4);

matrix\_mult(MAT\_A, MAT\_B, R, 4);

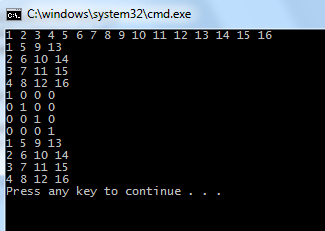
print\_matrix(R, 3);

free\_matrix(MAT\_A, 4);

free\_matrix(MAT\_B, 4);

free\_matrix(R,4);

Input 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16



We can see that the input matrix is equal to the output matrix after multiplication

**read mat**

The function will read values into the matrix , more then 16 values will be omited.

Less then 16 will be subtitue with 0.

It is good that we have internet today , ha? So in order to write a good input with protection that support space we can use the following code from :

<http://stackoverflow.com/questions/1247989/how-do-you-allow-spaces-to-be-entered-using-scanf>

/\* Maximum name size + 1. \*/

#define MAX\_NAME\_SZ 256

/\* Allocate memory and check if okay. \*/

char \*input = malloc(MAX\_NAME\_SZ);

if (input == NULL) {

printf("No memory\n");

return 1;

}

/\* Get the name, with size limit. \*/

fgets(input, MAX\_NAME\_SZ, stdin);

/\* Remove trailing newline, if there. \*/

if ((strlen(input)>0) && (input[strlen(input) - 1] == '\n'))

input[strlen(input) - 1] = '\0';

/\* Free memory and exit. \*/

free(input);

And to get the tokens from the input which are separated by spaces:

Use strtok function

char \* pch;

pch = strtok(input, " ");

while (pch != NULL)

{

printf("%s ", pch);

Fill the matrix here

pch = strtok(NULL, " ,.-");

}

And there for a good place will be in the red space above

We need a check number to see that the input number is valid:

int checkNumber(char \*some\_string, long \*value)

{

char \*endptr;

\*value = strtol(some\_string, &endptr, 10);

if (endptr == some\_string)

{

// Not a valid number at all

return 0;

}

else if (\*endptr != '\0')

{

return 0;

}

else

{

// String is a number

return 1;

}

}

The final read mat with the checking and inserting into the matrix:

void read\_mat(int\*\* matrix, int size)

{

/\* Maximum name size + 1. \*/

#define MAX\_NAME\_SZ 256

char \* pch;

int i = 0;

int j = 0;

long value;

int count = 0;

/\* Allocate memory and check if okay. \*/

char \*input = malloc(MAX\_NAME\_SZ);

if (input == NULL) {

printf("No memory\n");

return 0;

}

init\_matrix(matrix, size, 0);

/\* Get the name, with size limit. \*/

fgets(input, MAX\_NAME\_SZ, stdin);

/\* Remove trailing newline, if there. \*/

if ((strlen(input)>0) && (input[strlen(input) - 1] == '\n'))

input[strlen(input) - 1] = '\0';

pch = strtok(input, " ");

while (pch != NULL)

{

if (count == (size \* size))

break;

//printf("%s ", pch);

if (checkNumber(pch, &value) == 0)

{

printf("Invalid input");

return 0;

}

count++;

// here we need to fill the matrix with those tokens

if (i == size)

{

j++;

i = 0;

matrix[i][j] = (int)value;

}

else {

matrix[i][j] = (int)value;

}

i++;

pch = strtok(NULL, " ,.-");

}

/\* Free memory and exit. \*/

free(input);

return 1;

}

Read user input

int(\*functionParameters[8])(char \*str, int size) = { 0 };

functionParameters[0] = &read\_mat\_input\_parameters;

functionParameters[1] = &add\_mat\_input\_parameters;

functionParameters[2] = &print\_mat\_input\_parameters;

functionParameters[3] = &sub\_mat\_input\_parameters;

functionParameters[4] = &mult\_mat\_input\_parameters;

functionParameters[5] = &transpose\_mat\_input\_parameters;

functionParameters[6] = &mult\_scalar\_input\_parameters;

functionParameters[7] = &stop\_input\_parameters;

int read\_mat\_input\_parameters(char \*str, int size)

{

}

I am using a function pointer to minimize the code size in the parser