Exercises

Test your knowledge on pointers and dynamic memory!

We'll cover the following

Question

Refactor the code from your matrix program (from here) so that the size of a matrix is not fixed to a maximum number of elements. Instead use dynamic memory allocation.

Here are some hints:

Exercise:

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  double *data;
  int nrows;
  int ncols;
} Matrix;

void printmat(Matrix *M);
void matrixmult(Matrix *A, Matrix *B, Matrix *C);
Matrix *createMatrix(int nrows, int ncols);
void destroyMatrix(Matrix *M);

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

  Matrix *A = createMatrix(3, 2);
  //Uncomment the following code when you implement createMatrix
  //otherwise it will give a segmention fault if createMatrix
  //is not implemented correctly
  /*
  A->data[0] = 1.2;
  A->data[1] = 2.3;
  A->data[2] = 3.4;
  A->data[3] = 4.5;
  A->data[4] = 5.6;
  A->data[5] = 6.7;
  printmat(A);



  Matrix *B = createMatrix(2, 3);
  B->data[0] = 5.5;
  B->data[1] = 6.6;
  B->data[2] = 7.7;
  B->data[3] = 1.2;
  B->data[4] = 2.1;
  B->data[5] = 3.3;
  printmat(B);

  Matrix *C = createMatrix(3, 3);
  matrixmult(A, B, C);
  printmat(C);

  destroyMatrix(A);
  destroyMatrix(B);
  destroyMatrix(C);
  */
  return 0;
}

// your code goes below...


Matrix *createMatrix(int nrows, int ncols)
{
  // fill in the code here
}

void destroyMatrix(Matrix *M)
{
  // fill in the code here
}

void printmat(Matrix *M)
{
  // fill in the code here
  printf("so far printmat does nothing\n");
}

void matrixmult(Matrix *A, Matrix *B, Matrix *C)
{
  // fill in the code here
  printf("so far matrixmult does nothing\n");
}
```

Solution:

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

typedef struct {
```

```
typedef struct {
  double *data;
  int nrows;
  int ncols;
} Matrix;

void printmat(Matrix *M);
void matrixmult(Matrix *A, Matrix *B, Matrix *C);
Matrix *createMatrix(int nrows, int ncols);
void destroyMatrix(Matrix *M);

int main(int argc, char *argv[])
{
  Matrix *A = createMatrix(3, 2);
  A->data[0] = 1.2;
  A->data[1] = 2.3;
  A->data[2] = 3.4;
  A->data[3] = 4.5;
  A->data[4] = 5.6;
  A->data[5] = 6.7;
  printmat(A);

  Matrix *B = createMatrix(2, 3);
  B->data[0] = 5.5;
  B->data[1] = 6.6;
  B->data[2] = 7.7;
  B->data[3] = 1.2;
  B->data[4] = 2.1;
  B->data[5] = 3.3;
  printmat(B);

  Matrix *C = createMatrix(3, 3);
  matrixmult(A, B, C);
  printmat(C);

  destroyMatrix(A);
  destroyMatrix(B);
  destroyMatrix(C);
  return 0;
}

// your code goes below...


Matrix *createMatrix(int nrows, int ncols)
{
  Matrix *M = malloc(sizeof(Matrix));
  M->data = malloc(nrows*ncols*sizeof(double));
  M->nrows = nrows;
  M->ncols = ncols;
  return M;
}

void destroyMatrix(Matrix *M)
{
  free(M->data);
  free(M);
}

void printmat(Matrix *M)
{
  // fill in the code here
  // printf("so far printmat does nothing\n");
  int i, j;
  printf("[\n");
  for (i=0; i<M->nrows; i++) {
    for (j=0; j<M->ncols; j++) {
      printf("%6.3f ", M->data[i*M->ncols + j]);
    }
    printf("\n");
  }
  printf("]\n\n");
}

void matrixmult(Matrix *A, Matrix *B, Matrix *C)
{
  // fill in the code here
  // printf("so far matrixmult does nothing\n");
  if (A->ncols != B->nrows) {
    printf("error: ncols of A does not equal nrows of B\n");
  }
  else {
    int i, j, k;
    double count;
    for (i=0; i<A->nrows; i++) {
      for (j=0; j<B->ncols; j++) {
        count = 0.0;
        for (k=0; k<A->ncols; k++) {
          count += A->data[i*A->ncols + k] * B->data[k*B->ncols + j];
        }
        C->data[i*A->nrows + j] = count;
      }
    }
  }
}
```