Exercises

Time to write some code to solve problems.

We'll cover the following

Question

Below is a program that uses a `struct` to encapsulate a two-dimensional matrix. The struct contains the matrix values and the dimensions of the matrix. We assume that the matrix is filled row by row (that is, across columns). Since we haven't covered dynamic allocation of memory yet, for now, we assume a matrix can hold a maximum number of values equal to 1024. We will cover dynamic allocation of memory later.

Your first task is to write a function that prints a matrix to the screen. A function skeleton is provided (`printmat()`).

```
#include <stdio.h>

#define MAXDATA 1024

typedef struct {
  double data[MAXDATA];
  int nrows;
  int ncols;
} Matrix;

void printmat(Matrix M);
Matrix matrixmult(Matrix A, Matrix B);

int main(int argc, char *argv[])
{
  Matrix A = { {1.2, 2.3,
```

```
                3.4, 4.5,
                5.6, 6.7},
               3,
               2};
  Matrix B = { {5.5, 6.6, 7.7,
                1.2, 2.1, 3.3},
               2,
               3};
  printmat(A);
  printmat(B);

  //  Matrix C = matrixmult(A, B);
  //  printmat(C);

  return 0;
}

// your code goes below...

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

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

Exercise:

Exercise | Solution

```
void printmat(Matrix M)
{
  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");
}

int main(){
  Matrix A = { {1.2, 2.3,
                3.4, 4.5,
                5.6, 6.7},
               3,
```

```
                2};
        Matrix B = { {5.5, 6.6, 7.7,
                1.2, 2.1, 3.3},

                2,
                3};

  printmat(A);
  printmat(B);
  return 0;
}
```

Question

Your next task is to write a function that performs matrix multiplication. A function skeleton is provided (`matrixmult()`).

You can check your solution against Wolfram Alpha.

Here is an example of what the output might look like of your finished program:

```
[
 1.200  2.300
 3.400  4.500
 5.600  6.700
]

[
 5.500  6.600  7.700
 1.200  2.100  3.300
]

[
 9.360 12.750 16.830
24.100 31.890 41.030
38.840 51.030 65.230
]
```

Exercise:

Exercise | Solution

```
Matrix matrixmult(Matrix A, Matrix B)
{
  Matrix C;
  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++) {
```

```
      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;
      }
    }
    C.nrows = A.nrows;
    C.ncols = B.ncols;
  }
  return C;
}

int main(int argc, char *argv[])
{
  Matrix A = { {1.2, 2.3,
                3.4, 4.5,
                5.6, 6.7},
               3,
               2};
  Matrix B = { {5.5, 6.6, 7.7,
                1.2, 2.1, 3.3},
               2,
               3};

  Matrix C = matrixmult(A, B);
  printmat(C);

  return 0;
}
```

Question

We saw in this section how to use command-line arguments to your C programs. Modify the code from the Functions section (the Fibonacci function that you wrote here) so that it prints the nth Fibonacci number, passed through as a command line argument. Your program should be able to be run like this:

```
$ ./go 10
fib(10)=55

$ ./go 12
fib(12)=144

... and so on
```

Exercise:

```
#include <stdio.h>

int fib(int n);
```

```
//You can use this main function in your own environment.
//For now use the main function given below. You can give different
//values in hard coded form to the function "fib".

/*int main(int argc, char *argv[])
{
  if (argc < 2) {
    printf("error: please pass a number\n");
    return 1;
  }
  else {
    int n = atoi(argv[1]);
    if (n<0) {
      printf("error: n must be >= 0\n");
      return 1;
    }
    else {
      printf("fib(%d)=%d\n", n, fib(n));
      return 0;
    }
  }
}*/

int main()
{
  int n = 10;  //Change the value of "n" to whatever you want.
  printf("fib(%d)=%d\n", n, fib(n));
}

int fib(int n)
{
  int i;
  if ((n==0) | (n==1)) return n;
  int a = 0;
  int b = 1;
  int tmp;
  for (i=0; i<n; i++) {
    tmp = b;
    b = a+b;
    a = tmp;
  }
  return a;
}
```