Sample Solution to Assignment 1, Problem 4

COURSE HOME
SYLLABUS
CALENDAR
GETTING STARTED
LECTURE NOTES
ASSIGNMENTS
RELATED RESOURCES
DOWNLOAD COURSE MATERIALS

« Back to Assignments

```
/*

PROG: matrix2
LANG: C
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct Matrix_s {
  size_t R, C;
  int *index;
} Matrix;

Matrix* allocate_matrix( size_t R, size_t C ) {
  Matrix *matrix = malloc( sizeof( Matrix ) );
  matrix->R = R;
  matrix->C = C;
  matrix->index = malloc( R * C * sizeof( int ) );
  return matrix;
}

void destroy_matrix( Matrix *matrix ) {
  free( matrix->index );
  free( matrix );
}

typedef enum {
  REGULAR = 0,
  TRANSPOSE = 1
} Transpose;
```

```c
// Allowing reading a matrix in as either regular or transposed
Matrix* read_matrix( FILE *input, Transpose orient ) {
  size_t R, C;
  fscanf( input, "%zu %zu", &R, &C );

  Matrix *matrix = NULL;

  if( orient == REGULAR ) {
    matrix = allocate_matrix( R, C );
    for( size_t r = 0; r < matrix->R; ++r ) {
      for( size_t c = 0; c < matrix->C; ++c ) {
        fscanf( input, "%d", &matrix->index[c + r * C] );
      }
    }
  } else if( orient == TRANSPOSE ) {
    matrix = allocate_matrix( C, R );
    for( size_t r = 0; r < matrix->C; ++r ) {
      for( size_t c = 0; c < matrix->R; ++c ) {
        fscanf( input, "%d", &matrix->index[r + c * R] );
      }
    }
  } else {
    fprintf( stderr, "Error: unknown orientation %d.\n", orient );
    exit( EXIT_FAILURE );
  }

  return matrix;
}

void print_matrix( FILE *output, Matrix *matrix ) {
  fprintf( output, "%zu %zu\n", matrix->R, matrix->C );
  for( size_t r = 0; r < matrix->R; ++r ) {
    for( size_t c = 0; c < matrix->C - 1; ++c ) {
      fprintf( output, "%d ", matrix->index[c + r * matrix->C] );
    }
    fprintf( output, "%d\n", matrix->index[matrix->C - 1 + r * matrix->C] );
  }
}

Matrix* product_matrix( Matrix *a, Matrix *b ) {
  if( a->C != b->C ) {
    printf( "Error: tried to multiply (%zux%zu)x(%zux%zu)\n", a->R, a->C, b->C, b->R );
    exit( EXIT_FAILURE );
  }

  Matrix *prod = allocate_matrix( a->R, b->R );
  size_t nRows = prod->R, nCols = prod->C, nInner = a->C;

  for( size_t r = 0; r < nRows; ++r ) {
```

```
    for( size_t c = 0; c < nCols; ++c ) {
      prod->index[c + r * nCols] = 0;
      for( size_t i = 0; i < nInner; ++i ) {
        prod->index[c + r * nCols] += a->index[i + r * nInner] * b->index[i + c * nInner];
      }
    }
  }

  return prod;
}

int main(void) {
  FILE *fin = fopen( "matrix2.in", "r" );

  if( fin == NULL ) {
    printf( "Error: could not open matrix2.in\n" );
    exit( EXIT_FAILURE );
  }

  Matrix *a = read_matrix( fin, REGULAR );
  Matrix *b = read_matrix( fin, TRANSPOSE );
  fclose( fin );

  Matrix *c = product_matrix( a, b );

  FILE *output = fopen( "matrix2.out", "w" );

  if( output == NULL ) {
    printf( "Error: could not open matrix2.out\n" );
    exit( EXIT_FAILURE );
  }

  print_matrix( output, c );
  fclose( output );

  destroy_matrix( a );
  destroy_matrix( b );
  destroy_matrix( c );

  return 0;
}
```

Below is the output using the test data:

matrix2:

```
 1: OK [0.006 seconds]

 2: OK [0.007 seconds]
```

```
 3: OK [0.007 seconds]
 4: OK [0.019 seconds]
 5: OK [0.017 seconds]
 6: OK [0.109 seconds]
 7: OK [0.178 seconds]
 8: OK [0.480 seconds]
 9: OK [0.791 seconds]
10: OK [1.236 seconds]
11: OK [2.088 seconds]
```

« *Back to Assignments*