Exercise 7: Matrix

Nanda H Krishna

15 March 2018

1 Print a Matrix on `stdout`

Problem description: Define a function `mat_print()` that prints a matrix. The function is passed three parameters: matrix `a[M][N]`, and two shape parameters `m` and `n` (number of rows and number of columns). The size of the first dimension in `a[M][N]` is optional. Test the function from `main()`.

Specification: The function `mat_print()` takes the matrix (2-D Array), number of rows `m` and number of columns `n` of the matrix as inputs and displays the matrix on `stdout`.

Prototype:

```
void mat_print(int A[M][N], int m, int n)
```

Program design: The program consists of 2 functions, `mat_print(int A[M][N], int m, int n)` to print the matrix on `stdout` and `main()` reads a matrix and calls the function to test it.

Algorithm: The algorithm to print a matrix on `stdout` is as follows:

```
mat_print(A[M][N], m, n):
  for i in range(m):
    for j in range(n):
      print A[i][j]
      if j < n-1 print ',' else print '\n'
```

Program:

```
#include<stdio.h>
#define M 1000
#define N 1000
void mat_print(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%s", a[i][j], (j < n - 1) ? ", " : "\n");
      // Comma separation followed by new line at end
    }
```

```
  }
}
int main()
{
  int a[M][N], m, n;
  scanf("%d %d", &m, &n);
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
  mat_print(a, m, n);
  return 0;
}
```

Test Input:

```
2 4
0 7 0 3
1 9 7 3
```

Output:

```
0 7 0 3
1 9 7 3
```

2 Read a Matrix from `stdin`

Problem description: Define a function to read a matrix from `stdin`. The first line specifies the number of rows `m` and columns `n` of the matrix. This is followed by `m` lines, each having `n` numbers.

Specification: The function `mat_read()` takes the matrix (2-D Array), number of rows `m` and number of columns `n` of the matrix as inputs, with the shape parameters `m` and `n` being passed by reference. The matrix represented as a 2-D Array is automatically passed by reference. Outputs are m and n given by reference, and the matrix.

Prototype:

```
void mat_read(int A[M][N], int* m, int* n)
```

Program design: The program consists of `mat_read(int A[M][N], int* m, int* n)` which gets `m` and `n` and the matrix elements from `stdin`, and `mat_print(int A[M][N], int m, int n)` which is used to display the matrix on `stdout`. To test the functions, we have `main()`.

Algorithm: The algorithm to read a matrix from `stdin` is as follows:

```
mat_read(A[M][N], m, n):
  input m, n
```

```
  for i in range(m):
    input line
    for j in range(n):
      read A[i][j] from line
```

Program:

```
#include<stdio.h>
#define MAXLINE 1000
#define SIZE 100
void mat_read (int a[][SIZE], int* m, int* n)
{
  char buffer[MAXLINE];
  char *line = buffer; // If line were an array we cannot change it
  int nbytes;
  int i, j;

  fgets(line, MAXLINE, stdin);
  sscanf (line, "%d%d", m, n);

  for (i = 0; i < *m; i++) {
    fgets(line, MAXLINE, stdin);
    for (j = 0; j < *n; j++) {
      sscanf (line, "%d%n", &a[i][j], &nbytes);
      line += nbytes; // Shift pointer to next element
    }
  }
}
void mat_print(int a[][SIZE], int m, int n)
{
  for(int i = 0; i < m; i++)
    for(int j = 0; j < n; j++)
      printf("%d%s", a[i][j], j < n - 1 ? " " : "\n");
}
int main()
{
  int m, n, a[SIZE][SIZE];
  mat_read(a, &m, &n);
  mat_print(a, m, n);
  return 0;
}
```

Test Input:

```
3 3
2 3 4
5 6 7
8 9 1
```

Output:

```
2 3 4
5 6 7
8 9 1
```

3 Addition of 2 Matrices

Problem description: Write a function `mat_add(a, b, c, m, n)` to add two matrices `a` and `b` of shape `m` x `n`, and leave the result in matrix `c`. Test this function and all the subsequent functions from main().

Specification: The function `mat_add()` takes the matrices `a`, `b`, `c` (2-D Arrays), number of rows `m` and number of columns `n` of the matrices as inputs, with the output being the sum of `a` and `b` stored in `c`.

Prototype:

```
void mat_add(int a[M][N], int b[M][N], int c[M][N], int m, int n)
```

Program design: The program consists of `mat_add(int a[M][N], int b[M][N], int c[M][N], int m, int n)` which adds `a` and `b` into `c`, `mat_print(int A[M][N], int m, int n)` which is used to display the matrices on `stdout`, and `mat_read(int A[M][N], int m, int n)` to read the matrices `a` and `b`. The function `main()` is used for testing.

Algorithm: The algorithm to add 2 matrices is as follows:

```
mat_add(int a[M][N], int b[M][N], int C[M][N], int m, int n):
  for i in range(m):
    for j in range(n):
      c[i][j] = a[i][j] + b[i][j]
```

Program:

```
#include<stdio.h>
#define M 100
#define N 100
void mat_add(int a[M][N], int b[M][N], int c[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      c[i][j] = a[i][j] + b[i][j];
      // The i,j element of c is the sum of the i,j elements of a and b
    }
  }
}
void mat_read(int a[M][N], int m, int n)
{
```

```
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
}
void mat_print(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%s ", a[i][j], (j < n - 1) ? " " : "\n");
    }
  }
  printf("\n");
}
int main()
{
  int a[M][N], b[M][N], c[M][N], m, n;
  scanf("%d %d", &m, &n);
  mat_read(a, m, n);
  mat_read(b, m, n);
  mat_add(a, b, c, m, n);
  mat_print(c, m, n);
  return 0;
}
```

Test Input:

```
2 2
1 2
3 4
5 6
7 8
```

Output:

```
6  8
10 12
```

4 Copy a Matrix

Problem description: Define a function `mat_copy(a, b, m, n)` that copies `m` x `n` matrix `a` to matrix `b` of the same shape.

Specification: The function `mat_copy()` takes the matrices `a` and `b` (2-D Arrays), number of rows `m` and number of columns `n` of the matrices as inputs and copies elements of `a` to `b`.

Prototype:

```
void mat_copy(int a[M][N], int b[M][N], int m, int n)
```

Program design: The program consists of `mat_copy(int a[M][N], int b[M][N], int m, int n)` to copy `a` to `b`, and functions to read and print matrices. The `main()` function calls the others for testing.

Algorithm: The algorithm to copy a matrix to another is as follows:

```
mat_copy(a[M][N], b[M][N], m, n):
  for i in range(m):
    for j in range(n):
      b[i][j] = a[i][j]
```

Program:

```
#include<stdio.h>
#define M 100
#define N 100
void mat_copy(int a[M][N], int b[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      b[i][j] = a[i][j];
      // Copy i,j element of a to i,j position in b
    }
  }
}
void mat_read(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
}
void mat_print(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%s ", a[i][j], (j < n - 1) ? " " : "\n");
    }
  }
  printf("\n");
}
int main()
{
  int a[M][N], b[M][N], m, n;
```

```
  scanf("%d %d", &m, &n);
  mat_read(a, m, n);
  mat_copy(a, b, m, n);
  mat_print(b, m, n);
  return 0;
}
```

Test Input:

```
2 3
1 2 3
0 5 0
```

Output:

```
1 2 3
0 5 0
```

5 Scale a Matrix

Problem description: Write a function `mat_scale(a, b, m, n, f)` that maps every item of a `m` x `n` matrix `a` by multiplying it by a factor `f` and assigns the result to a matrix `b`.

Specification: The function `mat_scale()` takes the matrices `a` and `b` (2-D Arrays), number of rows `m` and number of columns `n` of the matrices and the factor of scale `f` as inputs and the output or scaled matrix is stored in `b`.

Prototype:

```
void mat_scale(int a[M][N], int b[M][N], int m, int n, int f)
```

Program design: The program consists of `mat_scale(int a[M][N], int b[M][N], int m, int n, int f)` to scale each element of `a` by `f` and copy to `b`, and functions to read and print matrices. The `main()` function calls the others for testing.

Algorithm: The algorithm to scale a matrix is as follows:

```
mat_scale(a[M][N], b[M][N], m, n, f):
  for i in range(m):
    for j in range(n):
      b[i][j] = a[i][j] * f
```

Program:

```
#include<stdio.h>
#define M 100
#define N 100
void mat_scale(int a[M][N], int b[M][N], int m, int n, int f)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
```

```
      b[i][j] = a[i][j] * f;
      // Scale i,j element of a and store in i,j position of b
    }
  }
}
void mat_read(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
}
void mat_print(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%s ", a[i][j], (j < n - 1) ? " " : "\n");
    }
  }
  printf("\n");
}
int main()
{
  int a[M][N], b[M][N], m, n, f;
  scanf("%d %d %d", &m, &n, &f);
  mat_read(a, m, n);
  mat_scale(a, b, m, n, f);
  mat_print(b, m, n);
  return 0;
}
```

Test Input:

```
2 2 4
3 4
5 6
```

Output:

```
12 16
20 24
```

6 Transpose of a Matrix

Problem description: Define a function `mat_transpose(a, b, m, n)` that assigns the transpose of a `m` x `n` matrix `a` to matrix `b`.

Specification: Function `mat_transpose()` takes two parameters, an input matrix `a` and an output matrix `b` in which the result is stored.

Prototype:

```
void mat_transpose(int a[M][N], int b[M][N], int m, int n)
```

Program design: The program consists of `mat_transpose(int a[M][N], int b[M][N], int m, int n)` to store the transpose of `a` in `b`, and functions to read and print matrices. To avoid `a` being used for read and write simultaneously, we have to use a temporary matrix to store the transpose, and after the transpose is constructed completely, copy it in the output array. The `main()` function calls the others for testing.

Algorithm: The algorithm to transpose a matrix is as follows:

```
mat_transpose(a, b, m, n):
  for i in range(m):
    for j in range(n):
       b[j][i] = a[i][j]
```

Program:

```
#include<stdio.h>
#define M 100
#define N 100
void mat_transpose(int a[M][N], int b[M][N], int m, int n)
{
  int c[M][N], i, j;
  for(i = 0; i < m; i++) {
    for(j = 0; j < n; j++) {
      c[j][i] = a[i][j];
      // Avoid reading and writing a simultaneously
    }
  }
  for(i = 0; i < n; i++) {
    for(j = 0; j < m; j++) {
      b[i][j] = c[i][j];
    }
  }
}
void mat_read(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
}
void mat_print(int a[M][N], int m, int n)
{
```

```
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%s ", a[i][j], (j < n - 1) ? " " : "\n");
    }
  }
  printf("\n");
}
int main()
{
  int a[M][N], b[M][N], m, n;
  scanf("%d %d", &m, &n);
  mat_read(a, m, n);
  mat_transpose(a, b, m, n);
  mat_print(b, n, m);
  return 0;
}
```

Test Input:

```
3 2
6 4
7 3
5 5
```

Output:

```
6 7 5
4 3 5
```

7 Multiplication of 2 Matrices

Problem description: Define a function `mat_mul(a, b, c, m, n, p)` that multplies an `m` x `n` matrix `a` and an `n` x `p` matrix `b` and assigns the result to a `m` x `p` matrix `c`.

Specification: Function `mat_mul()` takes parameters `a` (`m` x `n`), `b` (`n` x `p`) and `c` (`m` x `p`) matrices (2-D Arrays), and their shape parameters `m`, `n` and `p`. The output which is the product of `a` and `b` is stored in `c`.

Prototype:

```
void mat_mul(int a[M][N], int b[M][N], int c[M][N], int m, int n, int p)
```

Program design: The program consists of `mat_mul(int a[M][N], int b[M][N], int c[M][N], int m, int n, int p)` to multiply `a` and `b` and store the product in `c`, and functions to read and print matrices. The `main()` function calls the others for testing.

Algorithm: The algorithm to multiply 2 matrices is as follows:

```
matrix_mul(a, b, c, m, n, p):
```

```
  for i in range(m):
    for j in range(p):
       // dot product of row i and column j
       c[i,j] = 0
       for k in range(n):
          c[i,j] += a[i,k] * b [k,j]
```

Program:

```
#include<stdio.h>
#define M 100
#define N 100
void mat_mul(int a[M][N], int b[M][N], int c[M][N], int m, int n, int p)
{
  int i, j;
  for(i = 0; i < m; i++) {
    for(j = 0; j < p; j++) {
      for(int k = 0; k < n; k++) {
c[i][j] += a[i][k] * b[k][j];
      }
    }
  }
}
void mat_read(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      scanf("%d", &a[i][j]);
    }
  }
}
void mat_print(int a[M][N], int m, int n)
{
  for(int i = 0; i < m; i++) {
    for(int j = 0; j < n; j++) {
      printf("%d%s ", a[i][j], (j < n - 1) ? " " : "\n");
    }
  }
  printf("\n");
}
int main()
{
  int a[M][N], b[M][N], c[M][N] = { {0} }, m, n, p;
  scanf("%d %d %d", &m, &n, &p);
  mat_read(a, m, n);
  mat_read(b, n, p);
  mat_mul(a, b, c, m, n, p);
```

```
  mat_print(c, m, p);
  return 0;
}
```

Test Input:

```
2 3 2
1 2 1
0 2 0
1 2
0 1
1 0
```

Output:

```
2 4
0 2
```