Exercise 7: Matrices

R Ram Kaushik

April 3, 2018

1 Two-dimensional arrays

- Declare and initialize a 2×3 two-dimensional array of integers.
- Initialize the 2d array.
- The size of first dimenstion is optional. Initialize the 2d array, leaving out the first dimension.

1.1 Specification

A function `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array.

1.2 Prototype

```
void mat_print(int a[][10], int m, int n)
```

1.3 Program Design

The program consists of a function `mat_print(int a[][10], int m, int n)` which prints the matrix, and `main()`, which calls the function.

1.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
```

1.5 Source Code

```
#include<stdio.h>
void mat_print(int a[][10], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ",a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
int main(){
  int a[2][3]={{1,3,5},{7,9,11}};
  int b[][3]={{1,2,3},{4,5,6},{7,8,9}};
  mat_print(a,2,3);
  mat_print(b,3,3);
  return 0;
}
```

1.6 Output

```
1  3   5
7  9  11

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

2 Print a matrix on stdout

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()`.

2.1 Specification

A function `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array.

2.2 Prototype

```
void mat_print(int a[][10], int m, int n)
```

2.3 Program Design

The program consists of a function `mat_print(int a[][10], int m, int n)` which prints the matrix on `stdout`, and `main()`,which gets the input from `stdin` and calls the function.

2.4 Algorithm

```
def mat_print(a,m,n):
    for i in range(m):
        for j in range(n):
            print(a[i][j])
```

2.5 Source Code

```
#include<stdio.h>
void mat_print(int a[][10], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
}
int main(){
  int a[10][10],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);
}
```

2.6 Test Input

```
4 3
```

```
7 2 5 3 1 10 9 6 12 8 4 0
```

2.7 Output

$$\begin{array}{rrr}
7 & 2 & 5 \\
3 & 1 & 10 \\
9 & 6 & 12 \\
8 & 4 & 0
\end{array}$$

3 Read a matrix from stdin

Define an input format for matrix. The first line specifies the number of rows m and columns n of the matrix. This is followed by m lines. Each of these m lines has n numbers. After m lines, the data for another matrix may follow. For example, a 3×4 matrix and a 4×3 may be formatted in stdin as follows. Test the function from `main()`.

```
3 4
10 20 30 40
50 60 70 80
90 100 110 120
4 3
1 1 1
2 2 2
3 3 3
4 4 4
```

Define a function `mat_read()` for reading a matrix in this format. It has there results: a matrix and the shape of the matrix. The shape variables are passed by reference. Since matrix is a 2-d array, it is already passed by reference. `a` is a constat pointer to an integer.

3.1 Specification

2 functions `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array, and `mat_read()` which gets the input from `stdin`.

3.2 Prototype

```
void mat_print(int a[][20], int m, int n)
int mat_read(int a[][20], int* m, int* n)
```

3.3 Program Design

The program consists of 2 functions `mat_print(int a[][20], int m, int n)` which prints the matrix, `mat_read(int a[][20],int* m, int* n)` which reads the matrix and `main()`, which calls the function.

3.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
def mat_read(a,*m,*n):
   if(input(m,n)!=EOF):
      for i in range(m):
         for j in range(n):
            input(a[i][j])
```

3.5 Source Code

```
#include<stdio.h>
void mat_print(int a[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
int mat_read(int a[][20], int* m, int* n){
  if(scanf("%d%d",m,n)!=EOF){
    for(int i=0;i<*m;i++){
      for(int j=0;j<*n;j++){
scanf("%d",&a[i][j]);
      }
    }
    return 1;
  }
  else
    return 0;
}
int main(){
  int a[20][20],m,n;
  while(mat_read(a,&m,&n)!=0){
    mat_print(a,m,n);
  }
}
```

3.6 Test Input

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

3.7 Output

```
12 11 10
 9  8  7
 6  5  4
 3  2  1

13 14
15 16
```

4 Matrix addition

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()`.

4.1 Specification

3 functions `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array, `mat_read()` which gets the input from `stdin`, and `mat_add()` which adds the elements of 2 matrices and stores it in a new matrix.

4.2 Prototype

```
void mat_print(int a[][10], int m, int n)
void mat_read(int a[][20], int m, int n)
void mat_add(int a[][20], int b[][20], int c[][20], int m, int n)
```

4.3 Program Design

The program consists of 3 functions `mat_print(int a[][20], int m, int n)` which prints the matrix, `mat_read(int a[][20],int* m, int* n)` which reads the matrix, `mat_add(int a[][20], int b[][20], int c[][20], int m, int n)` which adds the elements of the matrices and `main()`, which calls the functions.

4.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
def mat_read(a,m,n):
   for i in range(m):
      for j in range(n):
         input(a[i][j])
def mat_add(a,b,c,m,n):
   for i in range(m):
      for j in range(n):
         c[i][j]=a[i][j]+b[i][j]
```

4.5 Source Code

```
#include<stdio.h>
void mat_read(int a[][20], 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[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
void mat_add(int a[][20], int b[][20], int c[][20], 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];
    }
  }
  mat_print(c,m,n);
}
int main(){
  int a[20][20],b[20][20],c[20][20],m,n;
  scanf("%d%d",&m,&n);
  mat_read(a,m,n);
  mat_read(b,m,n);
  mat_add(a,b,c,m,n);
}
```

4.6 Test Input

```
3 3
1 2 3 4 5 6 7 8 9
10 11 12 13 14 15 16 17 18
```

4.7 Output

```
11 13 15
17 19 21
23 25 27
```

5 Matrix copy

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

5.1 Specification

3 functions `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array, `mat_read()` which gets the input from `stdin`, and `mat_copy()` which copies the elements of 1 matrix and stores it in a new matrix.

5.2 Prototype

```
void mat_print(int a[][10], int m, int n)
void mat_read(int a[][20], int m, int n)
void mat_copy(int a[][20], int b[][20], int m, int n)
```

5.3 Program Design

The program consists of 3 functions `mat_print(int a[][20], int m, int n)` which prints the matrix, `mat_read(int a[][20],int* m, int* n)` which reads the matrix, `mat_copy(int a[][20], int b[][20], int m, int n)` which copies the elements of the matrix and `main()`, which calls the functions.

5.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
def mat_read(a,m,n):
   for i in range(m):
      for j in range(n):
         input(a[i][j])
def mat_copy(a,b,m,n):
   for i in range(m):
      for j in range(n):
         b[i][j]=a[i][j]
```

5.5 Source Code

```
#include<stdio.h>
void mat_read(int a[][20], 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[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
void mat_copy(int a[][20], int b[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      b[i][j]=a[i][j];
    }
  }
  mat_print(b,m,n);
}
int main(){
  int a[20][20],b[20][20],m,n;
  scanf("%d%d",&m,&n);
  mat_read(a,m,n);
  mat_copy(a,b,m,n);
}
```

5.6 Test Input

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

5.7 Output

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

6 Matrix scale

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 it by a factor `f` and assignes the result to a matrix `b`.

```
mat_scale (a, f, b)
  for i := 0 to m-1
    for j := 0 to n-1
       b[j,i] := f * a[i,j]
```

6.1 Specification

3 functions `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array, `mat_read()` which gets the input from `stdin`, and `mat_scale()` which multiplies the elements of 1 matrix and stores it in a new matrix.

6.2 Prototype

```
void mat_print(int a[][10], int m, int n)
void mat_read(int a[][20], int m, int n)
void mat_scale(int a[][20], int b[][20], int m, int n)
```

6.3 Program Design

The program consists of 3 functions `mat_print(int a[][20], int m, int n)` which prints the matrix, `mat_read(int a[][20],int* m, int* n)` which reads the matrix, `mat_scale(int a[][20], int b[][20], int m, int n)` which multiplies the elements of the matrix and `main()`, which calls the functions.

6.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
def mat_read(a,m,n):
   for i in range(m):
      for j in range(n):
         input(a[i][j])
def mat_scale(a,b,m,n,f):
   for i in range(m):
      for j in range(n):
         b[i][j]=f*a[i][j]
```

6.5 Source Code

```
#include<stdio.h>
void mat_read(int a[][20], 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[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
void mat_scale(int a[][20], int b[][20], int m, int n, int f){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      b[i][j]=f*a[i][j];
    }
  }
  mat_print(b,m,n);
}
int main(){
  int a[20][20],b[20][20],m,n,f;
  scanf("%d%d",&m,&n);
  mat_read(a,m,n);
  scanf("%d",&f);
  mat_scale(a,b,m,n,f);

}
```

6.6 Test Input

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

6.7 Output

```
 4  12  20
 8  16  24
36  28  32
```

7 Matrix transpose

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

The algorithm for transposing a matrix is

```
mat_transpose (a, b)
  for i := 0 to m-1
    for j := 0 to n-1
       b[j,i] := a[i,j]
```

It takes two parameters: an input matrix `a` and an output matrix `b` in which the result is stored. Thus, the function intends to use `a` as a read parameter and `b` as a write paratemeter. However, since arrays are passed by reference, actually both `a` and `b` are writeable. If someone calls the function as

```
mat_transpose (a, a, m, n)
```

in which `a` is read and written, the specification will not be satisfied. 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.

7.1 Specification

3 functions `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array, `mat_read()` which gets the input from `stdin`, and `mat_transpose()` which transposes the elements of 1 matrix and stores it in a new matrix.

7.2 Prototype

```
void mat_print(int a[][10], int m, int n)
void mat_read(int a[][20], int m, int n)
void mat_transpose(int a[][20], int b[][20], int m, int n)
```

7.3 Program Design

The program consists of 3 functions `mat_print(int a[][20], int m, int n)` which prints the matrix, `mat_read(int a[][20],int* m, int* n)` which reads the matrix, `mat_transpose(int a[][20], int b[][20], int m, int n)` which transposes the elements of the matrix and `main()`, which calls the functions.

7.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
def mat_read(a,m,n):
   for i in range(m):
      for j in range(n):
         input(a[i][j])
def mat_transpose(a,b,m,n):
   for i in range(m):
      for j in range(n):
         b[j][i]=a[i][j]
```

7.5 Source Code

```
#include<stdio.h>
void mat_read(int a[][20], 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[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
void mat_transpose(int a[][20], int b[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      b[j][i]=a[i][j];
    }
  }
  mat_print(b,m,n);
}
int main(){
  int a[20][20],b[20][20],m,n;
  scanf("%d%d",&m,&n);
  mat_read(a,m,n);
  mat_transpose(a,b,m,n);

}
```

7.6 Test Input

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

7.7 Ouput

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

8 Matrix multiplication

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`.

The algorithm for matrix multiplication is as follows.

```
matrix_add (a, b, c)
  for i := 0 to m-1
    for j := 0 to p-1
       // dot product of row i and column j
       c[i,j] := 0;
       for k := 0 to n-1:
          c[i,j] := a[i,k] + b [k,j]
```

To avoid writing `a` or `b`, produce the result in a temporary array `d`, and after the result is completely produced, save it in `c`.

8.1 Specification

3 functions `mat_print()`, which takes an 2-D array `a[][]`, and its dimensions `m`, `n` as inputs and prints the array, `mat_read()` which gets the input from `stdin`, and `mat_multiplication()` which multiplies the elements of 2 matrices and stores it in a new matrix.

8.2 Prototype

```
void mat_print(int a[][10], int m, int n)
void mat_read(int a[][20], int m, int n)
void mat_multiplication(int a[][20], int b[][20], int c[][20], int m, int n, int
```

8.3 Program Design

The program consists of 3 functions `mat_print(int a[][20], int m, int n)` which prints the matrix, `mat_read(int a[][20],int* m, int* n)` which reads the matrix, `mat_multiplication(int a[][20], int b[][20],int c[][20] int m, int n, int p)` which multiplies the elements of 2 matrices and stores it in a new matrix, and `main()`, which calls the functions.

8.4 Algorithm

```
def mat_print(a,m,n):
   for i in range(m):
      for j in range(n):
         print(a[i][j])
def mat_read(a,m,n):
   for i in range(m):
      for j in range(n):
         input(a[i][j])
def mat_multiplication(a,b,c,m,n,p):
   for i in range(m):
      for j in range(p):
         c[i][j]=0
         for k in range(n)
            c[i][j]+=a[i][k]*b[k][j]
```

8.5 Source Code

```
#include<stdio.h>
void mat_read(int a[][20], 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[][20], int m, int n){
  for(int i=0;i<m;i++){
    for(int j=0;j<n;j++){
      printf("%d ", a[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
void mat_multiplication(int a[][20], int b[][20], int c[][20], int m, int n, int
  for(int i=0;i<m;i++){
    for(int j=0;j<p;j++){
      c[i][j]=0;
      for(int k=0;k<n;k++){
c[i][j]+=a[i][k]*b[k][j];
      }
    }
  }
  mat_print(c,m,n);
}
```

```
int main(){
  int a[20][20],b[20][20],c[20][20],m,n,p;
  scanf("%d%d%d",&m,&n,&p);
  mat_read(a,m,n);
  mat_read(b,n,p);
  mat_multiplication(a,b,c,m,n,p);
}
```

8.6 Test Input

```
4 3 3
1 2 3 4 5 6 7 8 9 10 11 12
9 8 7 6 5 4 3 2 1
```

8.7 Output

```
 30   24   18
 84   69   54
138  114   90
192  159  126
```