REPORT

Data Structures

Student ID : 1771008
Name : Minjeong Kim

Homework2_1

변수(variable) 분석

main()함수에서 먼저 SparseMatrix 타입의 mat 변수를 설정한다.
SparseMatrix 타입의 구조체로 되어있으면 Sparse Matrix 타입안에 있는 변수는 총 4가지로 [표 1]과 같다.
이때 rows cols terms는 각각 matrix하나의 row수, column의 수를 저장하고, terms 변수에는 matrix에서 0이 아닌 변수들의 수를 저장한다.
element 타입의 array에는 matrix에 들어가는 수들의 정보를 담으며, element 타입의 구조체 안에 있는 변수는 총 3가지로 [표 2]와 같다.
matrix에 들어가는 수에 대한 정보로 해당하는 element의 위치를 row와 col에 저장하고 그 값을 value에 저장한다. 예를들어, row의 index3에 위치하고, column의 index2에 위치하는 수 50을 저장하고 싶다면
element타입안에 row = 3; col = 2; value =50; 으로 저장하면 된다.
따라서 SparseMatrix안에있는 elementarray의 배열 숫자만큼, Matrix안에 들어가는 숫자를 저장할 수 있다.

SparseMatrix [표1]	
type	**name**
element array	data
int	rows
int	cols
int	terms

element [표2]	
type	**name**
int	row
int	col
int	value

정리하자면, SparseMatrix 타입을 선언하면 해당하는 변수에는
row, column의 개수를 저장하고, element타입에 들어갈 수의개수를 terms에 저장한다.
그리고 Matrix에 들어가는 수에 대한 정보는 element array 형식으로 들어가는데, 해당 배열의 각 요소에는 Matrix에 들어갈 수의 위치를 row, col 변수에 저장해고 그 값을 value에 저장한다.

함수(function) 분석

1. matrixInput() (return: SparseMatrix)

초기 값으로 SparseMatrix 타입의 지역변수 m을 설정하고,
scanf로 row, col, terms에 해당하는 정보를 각각 입력받아 변수 m의 멤버인 rows, cols terms에 각각 저장한다.
terms의 값은 해당 matrix에 들어갈 수의 개수이므로, SparseMatrix의 멤버인 element array의 최대 값인 10을 넘지 않게 하기 위해 printf문으로 주의를 준다.
예시) >> Input row size: **6**
>> Input col size: **6**
>> Input terms(maximum is 10): **1**
이경우 6X6인 matrix가 생성된 것이고, 0이 아닌 숫자는 1개임을 의미한다.

이후, matrix에 들어갈 수의 위치와 값을 입력 받기위해 terms의 값만큼 반복해서 input을 받는다.
이때 input의 형태는 row col value로 한 줄씩 받는다.
예시) >> Input numbers by format row col value : **3 2 30**
이경우 matrix의 index3의 row, index2의 column에 위치한 값 30을 m의 data array의 index0에 저장한 것이다.

즉, 해당 함수로 matrix 숫자에 대한 input을 받아, 다음 matrix가 생성이 되었다.
마지막으로 m을 return해서 사용자 input을 받아 완성된 SparseMatirx타입의 변수를 main함수의 변수에 저장할 수 있다.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	**30**	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

2. transposMatrix(SparseMatrix m) (return: SparseMatrix)

초기 값으로 SparseMatrix 타입의 지역변수 trans를 설정하고, parameter인 SparseMatrix타입의 변수 m의 Matrix를 transpose operation한 결과를 저장한다.
transpose Matrix는 해당 Matrix의 주 대각선을 기준으로하여 뒤집어 얻을 수 있다.
즉, row와 column을 교환하여 얻는 matrix이다.
따라서 M*N matrix가 N*M 형태가 되고, 한 element의 row와 col의 값은 뒤바뀐다.
따라서 trans의 row는 m의 col로, col은 m의 row로 설정한다. (M*N -> N*M)
그리고 각 element에 접근하여, element의 위치값도 col과 row를 바꾸어서 저장한다 (B[i][j] -> B[j][i])
이때 매트릭스에 0이아닌 숫자만 바꿔주기 위해 반복은 m의 terms의 값만큼한다.
결국, trans변수는 변수m의 row와 col이 뒤집힌 결과인 transpose operation을 한 결과를 담은 matrix 가되고, trans를 return해서, transpose연산이 완료된 SparseMatrix타입의 변수를 main함수의 변수에 저장할 수 있다.

3. printMat(SparseMatrix m) (return: void)

SparseMatrix타입의 변수m의 값을 받아서, 0을 포함한 dense matrix형태로 출력하기 위한 함수이다.
dense matrix형태로 출력하기 위해서는 M*N에 대한 2차원 배열을 선언하고, 모든 값을 0으로 초기화 시킨 후, sparseMatrix에 저장되어있는 element들을 matrix의 알맞은 위치에 해당 값으로 설정한다.
따라서 malloc을 이용해서 m의 rows cols의 값만큼 M*N의 배열인 array변수를 만들어주고, for문을 이용해서 배열의 각요소에 접근해서 모든 값을 0으로 초기화한다.
이후, m.terms의 값만큼 반복을 해서 m의 element array에 저장된 element의 값들을 위에 만든 배열에 element의 위치값 (row, col)에 따라 value를 저장한다.
예시) m의 data의 index4의 위치에있는 element의 값이 row =3 col=2 value =30이라면
array[3][2] = 30으로 저장되게 한다. array[m.data[4].row][m.data[4].col] = m.data[4].value;
마지막으로, for문을 이용해서 printf로 array변수안에 있는 모든 값을 출력한다.

Homework2_2

변수(variable) 분석

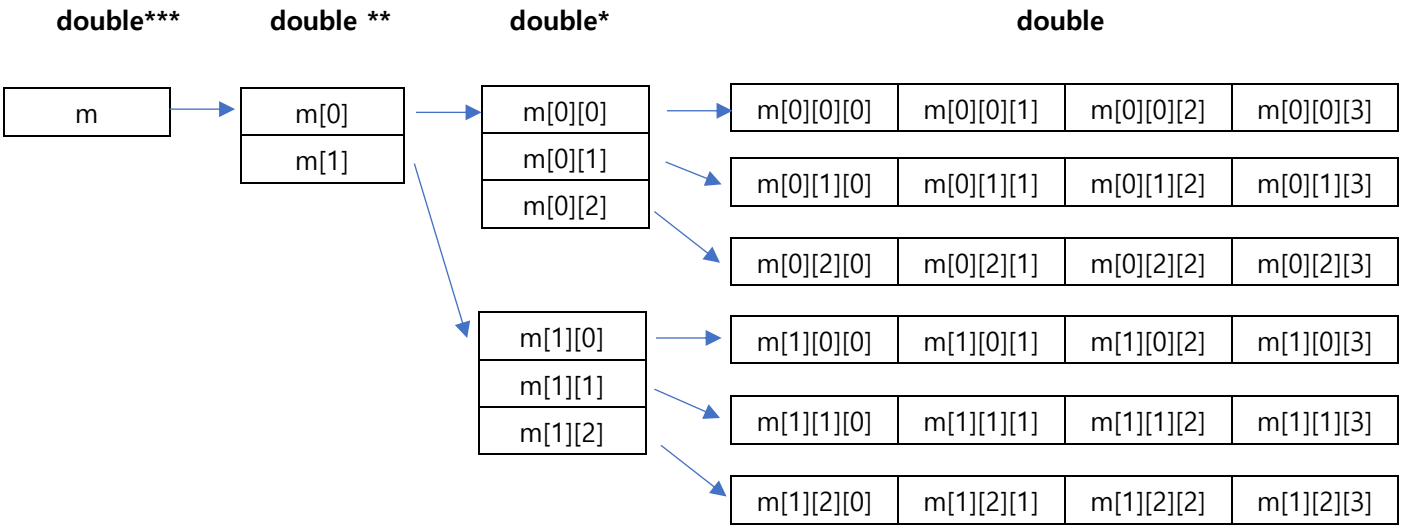
3차원 배열은 X*Y*Z 의 형태를 갖는다 따라서 배열의 크기를 설정하기위해 사용자에게 x, y, z 값을 입력 받는다.

예제) input x, y, z (format = x y z): **2 3 4**

이 결과 x=2, y=3 , z=4로 할당된다.

함수(function) 분석

1. mem_alloc_3D_double()



3차원 배열을 만들기 위한 함수 mem_alloc_3D_double()에서 나타나는 메모리 할당을 위처럼 표로 정리한 것이다.

이경우 배열의 크기는 2X3X4인 3차원 배열로 x=2 , y=3, z=4의 값을 갖는다.

m은 3중 포인터로 address of address of address를 의미하고,

m[]은 2중 포인터로 address of address를 의미하며,

m[][]은 포인터로 address를 의미한다.

즉, 배열의 m[][][]꼴의 값에 접근하기 위한 address는 m[][]이고,

m[][]의 address에 접근하기위한 address는 m[]이고 (address of address)

m[]의 address에 접근하기위한 address는 m이다. (address of address of address)

따라서, malloc을 이용해서 주소를 할당해주는데,

double*** 타입인 m은 m[0] m[1]의 두개의 double**타입의 공간을 필요로하므로, (x=2)

double ***m = (double***)malloc(sizeof(double**)*x); 로 메모리 할당을 해준다.

이때, (double***)을 붙이는 이유는, malloc은 void값의 형태로 return되어 데이터형을 맞춰주기 위해서이다.

마찬가지로 m[i]는 double **타입으로, m[i][0] , m[i][1] , m[i][2]의 세개의 double* 타입의 공간을 필요로하므로, (y=3)

m[i] = (double**)malloc(sizeof(double*)*y); 로 메모리 할당을 해준다.

m[i][j]는 double* 타입으로, m[i][j][0], m[i][j][1], m[i][j][2], m[i][j][3]의 네개의 double 타입의 공간을 필요로해, (z=4)

m[i][j] = (double**)malloc(sizeof(double)*z); 로 메모리 할당을 해준다.

이 함수는 double*** 타입으로, address of address of address를 의미한다. 따라서 해당 의미를 갖는 변수인 m을 return하여, main함수의 double***타입의 변수 A, B, C에 각각 할당을 한다.

2. assignMatrix(double ***m)

for문을 이용하여, 3차원 배열 m의 모든 요소에 사용자의 입력값을 할당한다.
입력의 형식은 matrix[i][j][q] = input 이다.
예시) matrix[0][0][1] = **2**
따라서 1부터 24까지의 숫자를 사용자가 input으로 assignMatrix()함수를 이용해서 입력했다고 했을 때,
3차원 배열의 값은 아래 표와 같다.

m[0][0][0] = 1	m[0][0][1] = 2	m[0][0][2] = 3	m[0][0][3] = 4
m[0][1][0] = 5	m[0][1][1] = 6	m[0][1][2] = 7	m[0][1][3] = 8
m[0][2][0] = 9	m[0][2][1]= 10	m[0][2][2] = 11	m[0][2][3] = 12
m[1][0][0] = 13	m[1][0][1] = 14	m[1][0][2] = 15	m[1][0][3] = 16
m[1][1][0] = 17	m[1][1][1] = 18	m[1][1][2] = 19	m[1][1][3] = 20
m[1][2][0] = 21	m[1][2][1] = 22	m[1][2][2] = 23	m[1][2][3] = 24

이 함수는 void 타입으로 return값이 없어도 된다.

3. addition_3D(double ***input1, double ***input2, double ***output)

할당이 완료된 double***타입의 세개의 변수를 addition_3D함수의 parameter로 사용한다.
main함수에서 addition_3D(A,B,C);로 입력하였는데, 이는 A matrix와 B matrix를 더하는 연산을 한 결과를 C matrix에 저장하겠다는 의미이다. 이때, 변수 A,B,C는 모두 double***타입 (address of address of address)이므로 &나* 같은 부가적인 기호를 붙이지 않고 바로 인자로 넣으면 된다.
matrix의 덧셈은 같은 자리에 있는 element들끼리의 합이기 때문에, output[i][j][q]를 이용해서 해당위치에 있는 값에 접근하다. 따라서 output[i][j][q] = input1[i][j][q] + input2[i][j][q]; 로 input의 [i][j][q]위치에 있는 값들을 더하여 output의 [i][j][q]위치에 저장한다는 의미가 된다.
그리고 덧셈이 잘 되었는지 확인하기 위해 printf를 이용하여 확인한다.

4. deallocate_3D_double(double*m)**

할당된 메모리를 해제할 때에는, 할당과 반대의 순서로 메모리를 해제해야한다.
메모리 할당의 과정이 m -> m[i] -> m[i][j]였다면
메모리 해제의 과정은 m[i][j] -> m[i] -> m이다.
따라서 중첩 반복문을 이용하여 먼저 free(m[i][j])를 해주고, 그 밖에 반복문에서 free(m[i]) 그리고 반복문의 밖에서 free(m)으로 3차원 배열의 메모리 공간을 모두 해제한다. 결국 변수 m은 메모리가 할당되지 않은 상태, 즉 아무것도 가리키지 않는 NULL포인터로 만들어야 하므로 m=NULL; 을 마지막줄에 넣어준다.