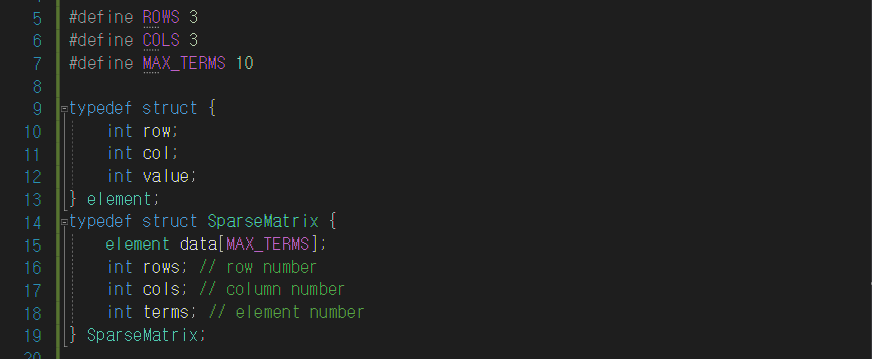
Homework 1

Theoretical explanation of the codes

**variable**

수업자료 ‘DS-Lec03-Array\_pointer.pdf’ p.26에 있는 자료구조를 활용하여 SparseMatrix 구조체를 구현하였습니다.



|  |  |
| --- | --- |
| **element 구조체** | |
| **data type** | **name** |
| int | row |
| int | col |
| int | value |

|  |  |
| --- | --- |
| **SparseMatrix 구조체** | |
| **data type** | **name** |
| element 배열 | data |
| int | rows |
| int | cols |
| int | terms |

1. **SparseMatrix 구조체**

* data : matrix에 있는 모든 non-zero element의 위치와 값을 저장한 배열
* rows : 전체 matrix 의 행 크기
* cols : 전체 matrix의 열 크기
* terms : non-zero element의 개수

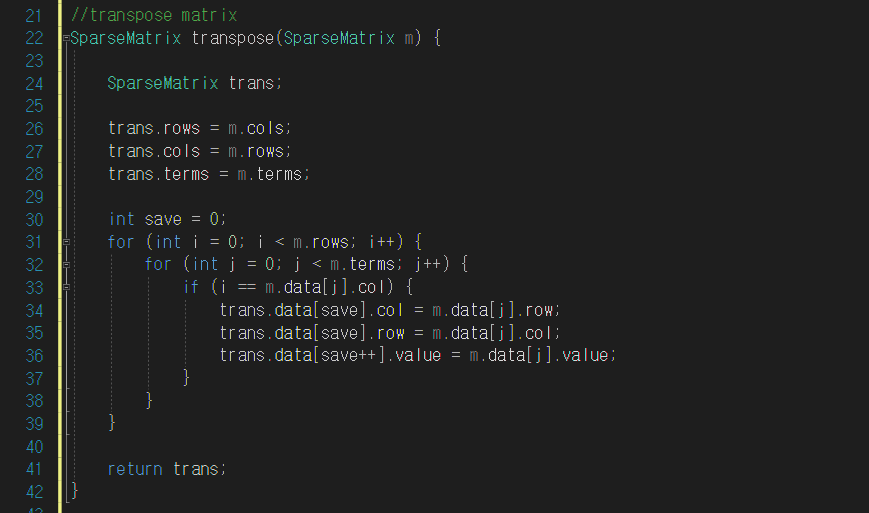
1. **element 구조체**

* row : non-zero element의 행 위치
* col : non-zero element의 열 위치
* value : non-zero element의 값

**function**

1. **SparseMatrix transpose(SparseMatrix m)**

: 인자로 전달된 SparseMatrix 구조체의 전치행렬(transpose matrix)을 구해주는 함수

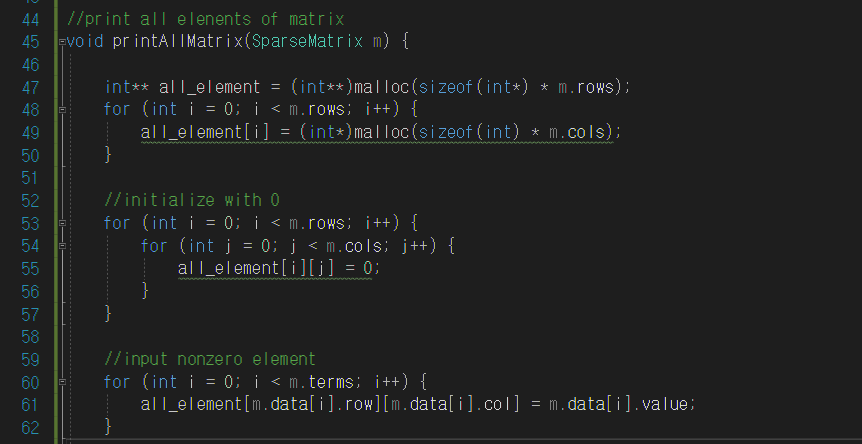


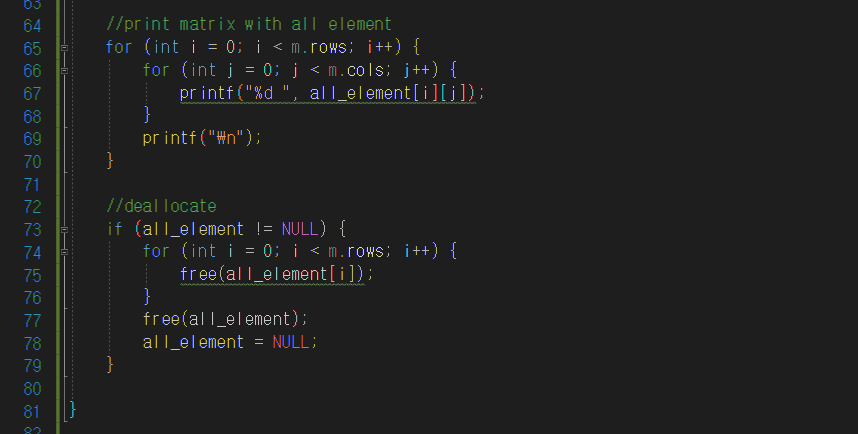
대칭을 수행할 matrix를 인자로 전달받습니다. 전달받은 matrix는 변수 m에 저장된다. transpose된 matrix를 저장하기 위해, 새로운 SparseMatrix 구조체 trans를 생성한다. 행렬을 전치하면 행과 열의 크기가 바뀌므로, trans의 rows값에는 기존 매트릭스 m의 cols값을, trans의 cols 값에는 기존 매트릭스 m의 row값을 넣어준다. terms의 값은 non- zero element의 개수는 동일하게 유지되므로 기존 매트릭스 m의 값과 동일하게 유지한다.

이 때, 전치된 행렬에서도 저장된 element들의 순서가 row값을 기준으로 정렬되어야 한다. 그러기 위해서 data배열 상에서 저장할 위치를 알려주는 save 변수를 따로 선언하고, 초기값은 0으로 설정한다. (배열의 0번째 index부터 저장할 것이기 때문이다.) 반복문을 통해 i 값이 0부터 행렬의 rows 값의 크기까지 1씩 증가하면서, 변수 i의 값과 기존 행렬의 element 배열의 col값들과 비교하여 두 값이 동일할 때만 row와 col값을 서로 바꿔서 trans 매트릭스의 element 배열에 저장해준다. 결과적으로 기존 행렬의 element 중에서 col값이 작은 순서대로 저장된다. 한 번 값이 저장될 때 마다, 배열의 그 다음 칸에 저장이 이루어져야하므로 저장할 위치를 의미하는 save 변수를 1 증가시킨다.

1. **void printAllMatrix(SparseMatrix m)**

: non-zero element만 저장된 SparseMatrix를 dense matrix 형태로 출력해주는 함수





2차원 배열 all\_element를 선언하고, 인자로 전달된 matrix의 멤버 rows와 cols 값의 크기만큼 메모리를 할당한다. 모든 원소의 값을 0으로 초기화시킨다. 그 후, 전달된 SparseMatrix 변수 m의 멤버인 data에 저장된 값을 이용하여 non-zero element 값만 배열의 적절한 위치에 저장한다. 반복문을 돌면서 모든 원소를 출력한 후, 메모리 할당을 해제한다.

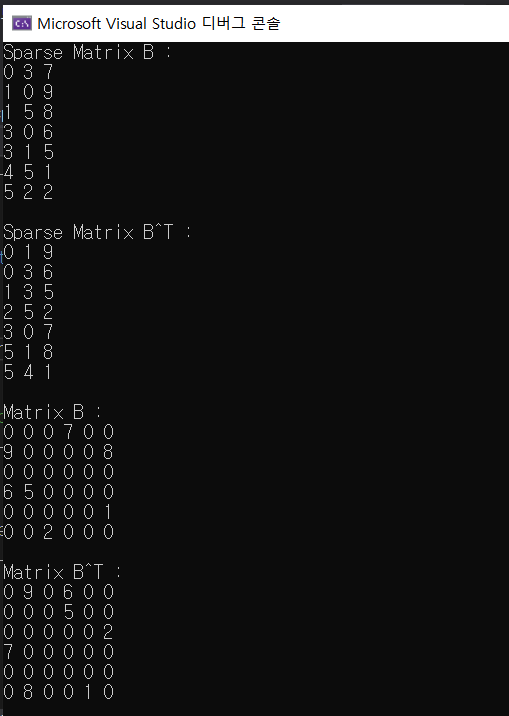
1. **void main()**

전체 프로그램의 실행 순서 :

SparseMatrix 구조체 B에 값을 저장 -> 전치행렬을 구하는 함수 transpose()를 실행하고 return된값을 변수 transposed\_B에 저장 -> 원래 행렬과 전치된 행렬의 sparse matrix form과 dense matrix form을 모두 출력.

experimental results

예시로 나와있는 matrix B값을 입력했을 때 출력되는 결과입니다.



Homework2

Theoretical explanation of the codes

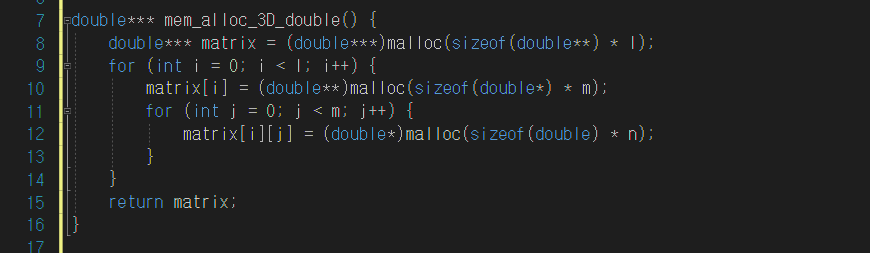
**variable**

3차원 배열의 크기 l,m,n은 모든 함수에서 공통적으로 사용되므로 전역변수로 선언하였다.

**Function**

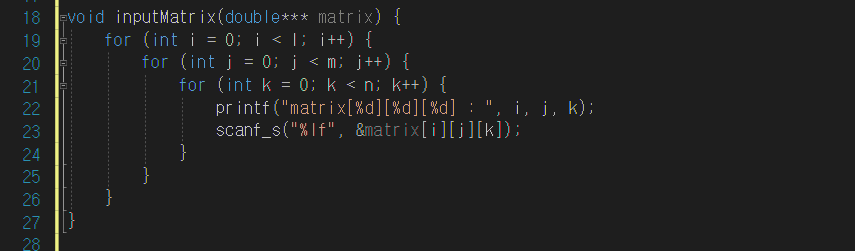
1. **double\*\*\* mem\_alloc\_3D\_double()**

: 3차원 배열의 크기(l\*m\*n)만큼 메모리를 할당하는 함수



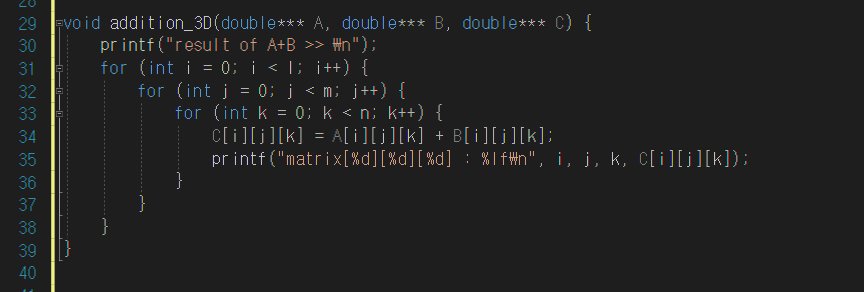
1. **void inputMatrix(double\*\*\* matrix)**

: 사용자에게 3차원 배열의 값을 차례로 입력 받아 저장하는 함수



1. **void addition\_3D(double\*\*\* A, double\*\*\* B, double\*\*\* C)**

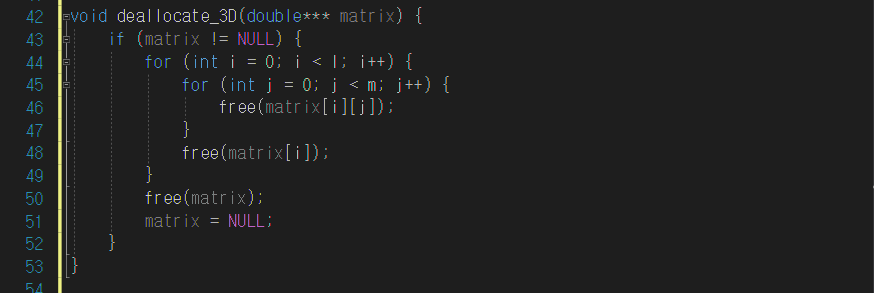
: 3차원 배열 A와 B의 덧셈을 수행하여 C에 저장한 후, 출력하는 함수



반복문을 돌며 A와 B matrix의 동일한 위치에 저장된 값끼리 더하는 연산을 수행한다. 그 후 matrix C의 동일한 위치에 저장하고, 차례로 출력한다.

1. **void deallocate\_3D(double\*\*\* matrix)**

**: 3차원 배열에 할당했던 메모리를 해제하는 함수**



메모리 해제는 할당과 정 반대의 순서로 이루어진다. 전달받은 배열 포인터 변수 matrix를이용하여 matrix[i][j] -> matrix[i] -> matrix의 순서로 반복문을 돌면서 할당한 메모리를 전부 해제한다.

1. **int main()**

전체 프로그램의 실행 순서 :

3차원 배열의 크기를 입력받아 l,m,n에 저장 -> 3차원 배열의 크기만큼 메모리를 할당 -> 덧셈을 수행할 행렬 A,B에 값을 저장 -> 덧셈을 수행하고 그 결과를 출력하는 함수 addition\_3D()를 호출 -> deallocate\_3D()를 호출하여 사용이 끝난 메모리 자원을 반납.

experimental results

