The Problem

- Similar to the first assignment, this assignment is about operations on sparse matrices. The only difference is that you are going to use a linked representation of sparse matrices, which is the one introduced in Chapter 4.
- You can use the existing definitions of the matrices and the matrix nodes in the textbook. You have to provide the implementation as well as new functionalities.
  - You can choose whether to use **`union`** or not.
- Always do the necessary range/size checking of the input arguments.

The Problem

- The list of functionalities to add:
  - Constructor:
    - `SparseMatrix(int rows, int cols);`
    - Initialize to an empty matrix (no non-zero terms).
  - Destructor:
    - `~SparseMatrix();`
    - You have to do this carefully so that it will not crash and there will be no memory leak.
  - Retrieve an item given its row/column index:
    - `float get(int row, int col);`
    - Return zero if the term (node) is not found.

The Problem

- The list of functionalities to add (continued):
  - Set the value of a term:
    - `void set(int row, int col, float value);`
    - Delete the term (node) if `value` is zero.
    - Add a new term (node) if necessary.
  - List the matrix in its normal (rectangular) matrix form:
    - `ostream& operator<<(ostream& os, const Matrix& m);`
    - Make this a friend function of the `Matrix` and the `MatrixNode` classes.
    - Other conditions are the same as in assignment #1.

The Problem

- The list of functionalities to add (continued):
  - Matrix addition:
    - **`SparseMatrix Add(const SparseMatrix &b);`**
    - Return **`*this + b`** as a new **`SparseMatrix`** object.
  - Matrix multiplication:
    - **`SparseMatrix Multiply(const SparseMatrix &b);`**
    - Return **`*this * b`** as a new **`SparseMatrix`** object.
  - Matrix transpose:
    - **`void Transpose();`**
    - Matrix **`*this`** is transposed; no new matrix created.

Additional Analysis

- For this part, you need to submit a separate file (Microsoft Word of PDF format).
- In this file, write your analysis about the following:
  - The time complexity for these added operations: **`get`**, **`set`**, **`Add`**, **`Multiply`**, **`Transpose`**.
    - Clearly indicate your instance characteristics.
    - Explain how you derive your complexities.
    - Compare your complexities in the first assignment. What are the differences? Give brief explanations.
  - Consider a slightly different implementation here: replacing the linked list of row/column header nodes with an array.
    - Will this change affect the complexities?
    - Discuss which one you will recommend, and why.

The Guidelines (Programming Part)

- Allowed programming environment: VS2015 only.
- For simplicity of submission, put the whole class, including the implementation, in **a single header file**.
- You need to write your own **`main`** function to test your code. You do not need to include this **`main`** function in your submission.
- No usage of STL class templates allowed.
- Include documentation; this will be part of your grade.
- Demo: Only a randomly selected subset of students; the list will be announced separately after the due date.

Submission

- Use E3 only.
- For the code, submit it under "**Assignment #1 – Programming Part**". Name your code **`P2_xxxxxx.h`**, where **`xxxxxx`** is your ID. **Do not** submit your **`main`** function or any file that is not your code (such as the *.sln file). No compressed file (*.zip, *.rar, etc.) accepted.
- For the analysis, submit it under "**Assignment #1 – Analysis Part**". Name your file **`P2_xxxxxx.docx`** or **`P2_xxxxxx.pdf`**.
- Due date: **11/4/2016**. There's a grace period of 3 days with 10% deduction per day. (The deduction kicks in only when you have accumulated more than three days of delay during the semester.)