UNIVERSITY OF WASHINGTON
Department of Applied Mathematics

AMATH 483/583 **Assigned:** 5/2/2018
Problem Set #Midterm **Due:** 5/9/2018

1 The Rules

This is a take-home midterm exam for AMATH 483/583. Its purpose is to evaluate what *you* have learned so far in this course. The same rules as for homework assignments apply for the mid-term – *with the following two acceptions*:

1. **The exam may not be discussed with anyone except course instructors**
   Do not discuss it with your classmates, with your friends, with your enemies. You *may* refer to your previous assignments, AMATH 483/583 slides and lectures.You may contact the instructors via private messages on Piazza to clarify questions. The same policies and penalties for plagiarism apply for the mid-term as for the regular assignments.

2. **No midterms will be accepted after 11:59pm PDT on 5/8**
   The submission entry on Canvas will electronically close at this time and no further midterms will be accepted. Grace days cannot be applied to the midterm.

Commenting Your Code

We haven't really had any requirements on commenting your code, either with `//` or the /*, */ pair. Continuing in this regard, general code comments will not be required for the midterm, but are *highly suggested*. Any clarifying comments may help the graders award partial credit if your code doesn't exhibit the correct behavior.

2 Python Grading Script

You will not be provided with python grading script for this midterm. You should still obtain the highest confidence in your code possible, via your own test drivers, before submitting the tarball. I suggest, at the minimum, unpacking the tarball, using `tar -xzf midterm.tgz`, in a new directory and verifying the `make all` command work, and generates the required executables listed in Section 4.
To enforce standard outputs, you are required to use the functions in `amath583IO.hpp` to write your results to files.

3 Exercises

3.1 Array of Structs

In class we discussed that there are two ways to represent a COO sparse matrix – struct of arrays or array of structs. Our implementation thus far has been the former. You are to implement a new `AOSMatrix` **`class`** that implements the COO sparse matrix as an array of structs (AOS).

Interface Below are some declarations of data members and member functions of the `AOSMatrix` class, which can be found in the provided `AOSMatrix.hpp` file. Your job is to fill in the implementation.

```
class AOSMatrix {
public:
  AOSMatrix(int M, int N);

  void push_back(int  i, int  j, double val);
  void clear();
  void reserve(int n);

  int num_rows();
  int num_cols();
  int numNonzeros();
  void readMatrix(std::string& inputfile);

  void matvec(const Vector& x, Vector& y) const;

private:
  class Element {
  public:
    int row, col;
    double val;
  };

  int iRows, jCols;
  std::vector<Element> arrayData;
};
```

Note the difference in how one obtains the row, column, and values for the kth element:

Current		Previous
`arrayData[k].row`	versus	`rowIndices[k]`
`arrayData[k].col`		`colIndices[k]`
`arrayData[k].val`		`arrayData[k]`

Requirements You should implement following interfaces. Look for `/* fix me */` and `/* write me */` sign in the source code.

- `AOSMatrix(int M, int N)` should create a sparse-representation object for an `M` by `N` matrix
- `void push_back(int i, int j, double val)` should add an `Element` to this object, setting the `i`-th column, `j`-th row of the matrix it represents to `val`
- `void clear()` should delete all the matrix elements in this object
- `void reserve(int n)` should reserve space for `n` matrix elements
- `int num_rows()` should return the number of rows in the matrix this object represents
- `int num_cols()` should return the number of columns in the matrix this object represents
- `int numNonzeros()` should return the number of non-zero elements in the matrix this object represents
- `void readMatrix(std::string& inputfile)` should read a matrix in COO format from a text file `inputfile` and store the matrix in this object.
- `void matvec(const Vector& x, Vector& y) const` should multiply the `Vector x` by the matrix this object represents and store the result in `Vector y`

Do not modify anything below (in the private section):

- `int iRows, jCols` denote the number of rows and columns, respectively, of the matrix this object represents

- `std::vector<Element> arrayData` is an array where `arrayData[k]` is the Element object that represents the `k`-th matrix element (i, j, val)
- `int row, col` denote the row index and column index of the matrix element (i, j, val) that this Element object represents
- `double val` denote the entry value of the matrix element (i, j, val) that this Element object represents

These implementations should be placed within the `CSCMatrix class`.
You should also implement the `* operator` in `amath583.cpp`:

- `Vector operator*(const AOSMatrix& A, const Vector& x)`

Code Deliverable Complete source code for the `AOSMatrix` class. Then write a program `aosmatvec.exe` that computes matrix vector product, where the matrix is represented by an `AOSMatrix` object.

Your program should take three command line arguments. The first corresponds to a file that contains a matrix `A` in COO format (see the `Input File Format` section). The second argument corresponds to a file that contains a vector, `x` (in the format as a vector saved using *writeVector()* in `amath583IO.hpp`). The third argument corresponds to a file where you should put the result of matrix vector product between `A` and `x` (namely `A*x`).

Your program should read in a matrix stored in COO format in a file using the `readMatrix` member function of the `AOSMatrix` class, which you are asked to implement. You can read the vector from a file using the function *readVector()* from `amath583IO.hpp`. You must use the function *writeVector()* in `amath583IO.hpp` to write the resulting matrix-vector product to the file `mat_vec_aos.txt`.

Input File Format You are provided with sample input files `inputCOOMatrix.txt` and `inputVector.txt`, which can be used to test both `aosmatvec.exe` and `cscmatvec.exe` (AMATH 583 only). The `Vector` object is stored in `inputVector.txt` with the same format as in `ps1`. The Matrix stored in COO format in `inputCOOMatrix.txt` has the following format:

1. The first line contains "AMATH 583 COOMATRIX"
2. The second line has two numbers representing number of rows and columns in the matrix
3. The third line has one number representing number of non-zero entries in the matrix
4. Each of following lines has three numbers that represent row index, column index and value of one non-zero entry in the matrix
5. The last line contains "THIS IS THE END"

Error Handling Your program should anticipate user errors. A non-complete list of these is

- command-line argument errors
- input data formatting errors
- mismatch between matrix and vector dimensions for multiplication

If an error is detected, the program should print an error message to the terminal and immediately exit with an error code (non-zero integer of your choosing).

3.2 Compressed Sparse Column (AMATH583 Only)

In lecture we derived the compressed sparse column format from coordinate format by first sorting the data based on the row indices stored in COO. By applying run-length encoding we compressed the row indices from a size equal to the number of non-zeros in the matrix down to the number of rows in the matrix (plus one).

A similar process could have been applied had we instead sorted the coordinate data by columns rather than rows. Such a format is known as compressed sparse column (CSC). You are to implement a `CSCMatrixclass` using this approach.

Interface Below are some declarations of data members and member functions of `CSCMatrix` class. Your job is to fill in the implementation.

```
class CSCMatrix {
public:
  CSCMatrix(int M, int N);

  void openForPushBack() { is_open = true};
  void closeForPushBack();
  void push_back(int  i, int  j, double val);
  void clear();
  void reserve(int n);

  int num_rows();
  int num_cols();
  int numNonzeros();
  void readMatrix(std::string& inputfile);

  void matvec(const Vector& x, Vector& y) const;

private:
  int iRows, jCols;
  bool is_open;
  std::vector<int> rowIndices, colIndices;
  std::vector<double> arrayData;
};
```

Requirements You should implement following interfaces. Look for `/* fix me */` and `/* write me */` sign in the source code.

- `CSCMatrix(int M, int N)` should create a CSC-representation object for an `M` by `N` matrix
- `void closeForPushBack()` should update `is_open` and compress the indices in `colIndices`, accordingly
- `void push_back(int i, int j, double val)` should add matrix element to this object, setting the `i`-th column, `j`-th row of the matrix it represents to `val`
- `void clear()` should delete all the matrix elements in this object
- `void reserve(int n)` should reserve space for `n` matrix elements
- `int num_rows()` should return the number of rows in the matrix this object represents
- `int num_cols()` should return the number of columns in the matrix this object represents
- `int numNonzeros()` should return the number of non-zero elements in the matrix this object represents
- `void matvec(const Vector& x, Vector& y) const` should multiply the `Vector x` by the matrix this object represents and store the result in `Vector y`
- `void readMatrix(std::string& inputfile)` should read a matrix in COO format from a text file `inputfile` and store the matrix in this object.

Do not modify anything below (in the private section):

- `int iRows, jCols` denote the number of rows and columns, respectively, of the matrix this object represents

- `bool is_open` indicate whether the object is accepting more elements before compression
- `std::vector<int> rowIndices` is an array where `rowIndices[k]` is the row index of the `k`-th matrix element $(i, , j, val)$
- `stid::vector<double> arrayData` is an array where `arrayData[k]` is the entry value of the `k`-th matrix element (i, j, val)

These implementations should be placed within the `CSCMatrix class`.
You should also implement the `* operator` in `amath583.cpp`:

- `Vector operator*(const CSCMatrix& A, const Vector& x)`

Code Deliverable Complete source code for the `CSCMatrix` class. Then write a program `cscmatvec.exe` that computes matrix vector product, where the matrix is represented by an `CSCMatrix` object.

Your program should take three command line arguments. The first corresponds to a file that contains a matrix `A` in COO format (see the `Input File Format` section above). The second argument corresponds to a file that contains a vector, `x` (in the format as a vector saved using *writeVector()* in `amath583IO.hpp`). The third argument corresponds to a file where you should put the result of matrix vector product between `A` and `x` (namely `A`*`x`). Your program should read in a matrix stored in COO format in a file using the `readMatrix` member function of the `CSCMatrix` class, which you are asked to implement. You can read the vector from a file using the function *readVector()* from `amath583IO.hpp`. You must use the function *writeVector()* in `amath583IO.hpp` to write the resulting matrix-vector product to the file `mat_vec_csc.txt`.

Error Handling Your program should anticipate user errors. A non-complete list of these is

- command-line argument errors
- input data formatting errors
- mismatch between matrix and vector dimensions for multiplication

If an error is detected, the program should print an error message to the terminal and immediately exit with an error code (non-zero integer of your choosing).

4 Turning in The Exercises

All your .cpp, .hpp files, and Makefile, should go in the tarball `midterm.tgz` and submit to Canvas.

AMATH 483: Before submitting homework, check that the command "make all" generates the following executables:

1. aosmatvec.exe [MatrixFilename] [VectorFilename] [outputFile]

AMATH 583: Before submitting homework, check that the command "make all" generates the following executables:

1. aosmatvec.exe [MatrixFilename] [VectorFilename] [outputFile]
2. cscmatvec.exe [MatrixFilename] [VectorFilename] [outputFile]