UNIVERSITY OF WASHINGTON
Department of Applied Mathematics

AMATH 483/583 **Assigned:** 4/25/2017
Problem Set #Midterm **Due:** 5/1/2017

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 – only with the course instructors via private messages on Piazza or in office hours. You *may* refer to your previous assignments, AMATH 483/583 slides and lectures, and instructor solutions for previous problem sets in this course. 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 5/1**
   The Catalyst Dropbox will electronically close at this time and no further midterms will be accepted.

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*. Your exam will be hand graded, and any clarifying comments may help the graders award partial credit if your code doesn't exhibit the correct behavior.

2 Command Line Matrix-Vector Product

You are asked to create a program for performing matrix-vector product. The program reads its command line arguments to obtain data for the operation to/from files or to/from `cout` /`cin`.

Usage The program is named `matvec`. It is invoked in the following way:

```
% ./matvec matrixFile [ vectorInputFile ] [ vectorOutputFile ]
```

The argument `matrixFile` specifies the plain-text file containing the matrix to be used. The program should accept matrices in both a dense format and a sparse format of your choosing. The arguments `vectorInputFile` and `vectorOutputFile` are optional. If the argument `vectorOutputFile` is not provided, your program should write its answer to `cout`. Similarly, if the arguments `vectorInputFile` and `vectorOutputFile` are both absent, your program should read its vector data from the `cin` and write its vector output to `cout`.

Requirements Your program is expected to

- correctly read a matrix in a dense format that includes a header, trailer, number of columns, and number of rows
- correctly read a matrix in a sparse format that includes the above as well as the number of non-zero entries
- correctly accept a vector in the format used in previous assignments, either from the terminal or from a text file based on the command-line arguments

- compute the matrix vector product between the input matrix and input vector
- correctly directing the result, in the format used in previous assignments, either to the terminal or to a text file based on the command-line arguments
- handle errors as described below

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

- command-line argument errors (if the user includes 1 or 3 arguments, you may assume they are to valid text files)
- 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).

Deliverables Complete source code for the `matvec` program. This includes:

- A `Makefile` that successfully builds your program in response to `make matvec`.
- All C++ source files that make the above `make` command possible.
- A sample input text file `sampleDenseMatrix.txt` that contains the matrix $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.
- A sample input text file `sampleSparseMatrix.txt` that contains a 2 by 2 matrix whose non-zero entries are $A_{0,0} = 1$ and $A_{1,1} = 2$.

You may use files or matrix formats provided previously in this course. If you make any changes to the files, please insert a comment *// BEGIN CHANGE* and *// END CHANGE* to delimit the code you changed.

3 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 AOSCOO matrix that implements the COO sparse matrix as an array of structs. The AOSCOO matrix class can be used to produce a matrix plain-text file that can be used by your `matvec` program above.

Interface The AOSCOO matrix has the following declaration. 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 numRows();
  int numCols();
  int numNonzeros();

  void matvec(const Vector& x, Vector& y) const;
  void streamMatrix(std::ostream& outputFile) 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 Proper implementation of the interface above:

- `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 numRows()` should return the number of rows in the matrix this object represents
- `int numCols()` 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 streamMatrix(std::ostream& outputFile) const` should send the matrix this object represents, **in the format accepted by your `matvec` program above**, to the output stream defined by `outputFile`, which may be either of `std::ofstream` or `std::cout`
- `int iRows, jCols` should denote the number of rows and columns, respectively, of the matrix this object represents
- `std::vector<Element> arrayData` should be an array where `arrayData[k]` is the Element object that represents the `k`-th matrix element (i, j, val)
- `int row, col` should denote the row index and column index of the matrix element (i, j, val) that this Element object represents
- `double val` should denote the entry value of the matrix element (i, j, val) that this Element object represents
- all implementations should placed within the `AOSMatrix class`, as was done with the related functions in `COOMatrix class`

Deliverables Complete source code for the `AOSCOO` sparse matrix class.

- A `Makefile` that sucessfully compiles your class in response to `make AOSCOO.o`
- A C++ source file named `AOSMatrix.hpp` that contains the entire implementation of `class AOSMatrix`
- A `Makefile` that successfully builds a program `aosmatvec` in response to `make aosmatvec`, which has the same requirements as `matvec` above, with the exception that if the input is a COO, rather than creating a `COOMatrix` and calling a COO based matrix-vector product function, your program should construct an `AOSMatrix` and call the corresponding matrix-vector product function.
- All C++ source files that make the above `make` command possible.

You may use files provided previously in this course for `aosmatvec`. If you make any changes to those files, please insert a comment `// BEGIN CHANGE` and `// END CHANGE` to delimit the code you changed.

Extra Credit For extra credit include a table or graphic, with `AOSvsCOO` in the name, that compares the performance of `AOSMatrix` with `COOMatrix` for a variety of matrix sizes. You can plot time or performance in FLOPS. To generate large matrices you may use "piscetize" to fill it in and use "randomize" to generate the vectors.

4 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 `CSCMatrix` class using this approach. As with the `AOSCOO` class above, `CSCMatrix` can be used to produce a matrix plain-text file for use by your `matvec` program.

Interface The `CSCMatrix` class has the following declaration. 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 numRows();
  int numCols();
  int numNonzeros();

  void matvec(const Vector& x, Vector& y) const;
  void streamMatrix(std::ostream& outputFile) const;

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

Requirements Proper implementation of the interface above:

- `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 numRows()` should return the number of rows in the matrix this object represents
- `int numCols()` 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 streamMatrix(std::ostream& outputFile) const` should send the matrix this object represents, **in the format accepted by your `matvec` program**, to the output stream defined by `outputFile`, which may be either of `std::ofstream` or `std::cout`
- `int iRows, jCols` should denote the number of rows and columns, respectively, of the matrix this object represents
- `bool is_open` should indicate whether the object is accepting more elements before compression
- `std::vector<int> rowIndices` should be an array where `rowIndices[k]` is the row index of the `k`-th matrix element $(i, , j, val)$
- `stid::vector<double> arrayData` should be an array where `arrayData[k]` is the entry value of the `k`-th matrix element (i, j, val)
- all implementations should placed within the `CSCMatrix class`, as was done with the related functions in `COOMatrix class`

Deliverables Complete source code for the `CSCMatrix` class.

- A `Makefile` that successfully compiles your class in response to `make CSCMatrix.o`
- A C++ source file named `CSCMatrix.hpp` that contains the entire implementation of `class CSCMatrix`
- A `Makefile` that successfully builds a program `cscmatvec` in response to `make cscmatvec`, which has the same requirements as `matvec` above, with the exception that if the input is a COO, rather than creating a `COOMatrix` and calling a COO based matrix-vector product function, your program should construct an `CSCMatrix` and call the corresponding matrix-vector product function.
- All C++ source files that make the above `make` command possible.

You may use files provided previously in this course for `cscmatvec`. If you make any changes to those files, please insert a comment *`// BEGIN CHANGE`* and *`// END CHANGE`* to delimit the code you changed.

Extra Credit For extra credit include a table or graphic, with `CSCvsCOO` in the name, that compares the performance of `CSCMatrix` with `COOMatrix` for a variety of matrix sizes. You can plot time or performance in FLOPS. To generate large matrices you may use "piscetize" to fill it in and use "randomize" to generate the vectors.

5 What To Turn In

Similar to the homework assignments, you are to turn in a tarball `midterm.tgz`, containing all the deliverables described above, to the CollectIt Dropbox. Unlike the homework, you are not provided with a testing script. 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 all the `make` commands work.