| Due before: | 11:59 PM on Wed Oct 13 2021 |
|---|---|

# CSE-443/543: High Performance Computing

# Homework #4

## Due: Wed Oct 13 2021 before 11:59 PM

## Email-based help Cutoff: 5:00 PM on Tue Oct 12 2021

Maximum Points: 20

**Submission Instructions**

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your program compiles successfully, without any warnings or style errors. Ensure you have documented the methods. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload just the following via the CODE plug-in:

1. The C++ header and source file modified for this project.

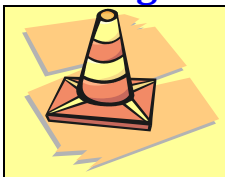
**General Note**: Upload each file associated with homework individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

**Objective**

The objective of this homework is to develop a K-means clustering program to continue to gain familiarity with:

- Work with 1-D and 2-D vectors for matrix.
- Continue to gain familiarity with operator overloading.
- Review reading and writing data to data files.

## Grading Rubric:



The program submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet base case requirements or just skeleton code will be assigned zero score! Program that do not compile, **have even 1 method longer than 25 lines**, or just some skeleton code will be assigned zero score.

- Your program should compile without any warnings.
- **NOTE:** Violating CSE programming style guidelines is a compiler error! Your program should not have any style violations reported.
- **Do not use global variables** – that is not good programming practice.
- Points for operators: stream extraction (3-points), `dot` (3-points). Rest of the operators 2-points each.
- **Formatting & Documentation: 2 points** (if your comments are poor, then you lose points in this category)
- **Delayed submission: Only 80% points:** Submission delayed **by no more than 24-hours** will be accepted for a partial credit of maximum 80% of the points.

# Project overview

In this project you will be developing a simple `Matrix` class that can perform standard matrix operations on a 2-D matrix of `double` values. Most of the matrix operations such as addition, subtraction, multiplication is straightforward and perform the operation on corresponding values as shown in the following pseudocode:

```
Matrix_operation(A, B, ⊗):
Assume dimension of A is (m × n), dimension of B is (m × n)
The operator ⊗ can be +, -, *
Begin
   define C matrix as (m × n)
   for i in range 0 to m - 1, do
      for j in range 0 to n - 1, do
            C[i][j] = (A[i][j] ⊗ B[i][j])
         done
      done
   done
   return C
End
```

The pseudocode for the apply a given method or unary-operator (⊗u) is shown below:

```
Matrix_apply(A, ⊗u):
Assume dimension of A is (m × n)
Begin
   define C matrix as (m × n)
   for i in range 0 to m - 1, do
      for j in range 0 to n - 1, do
            C[i][j] = ⊗u(A[i][j])
         done
      done
   done
   return C
End
```

The matrix-multiplication or dot operator has a different operation and its pseudocode is shown below:

```
Matrix_dot(A, B):
Assume dimension of A is (m × n), dimension of B is (p × q)
Begin
   if n is not same as p, then exit
   otherwise define C matrix as (m × q)
   for i in range 0 to m - 1, do
      for j in range 0 to q - 1, do
         for k in range 0 to p, do
            C[i][j] = C[i][j] + (A[i][k] * B[k][j])
         done
      done
   done
   return C
End
```

| Due before: | 11:59 PM on Wed Oct 13 2021 |
|---|---|

# Project requirements

In this project, you are expected to implement the various operators declared in the supplied `Matrix.h` header file:

- First, you will need to add a dummy body for each method in order to compile the starter code.
- The `apply` method must be implemented in the header file (as this method accepts the unary operator as a templatized parameter for versatility and good performance).
- You may implement short methods (say up to 3- lines) in the `Matrix.h` file.
- Longer methods must be implemented in the `Matrix.cpp` source file.
- The stream extraction operator (`operator >>`) should be able to read values from text files. See supplied `mat3x3.txt` file for example. The input data to the stream extraction operator is supplied in the following format:

```
<rows> WS <cols> WS <num> WS <num> ....
```

Where WS indicates 1 or more white spaces. There are exactly `rows` × `cols` values indicated by `<num>`.

**Starter code**: To aid with this project you are already supplied with data files and starter code in the **`/fs/ess/PMIU0184/cse443/homeworks/homework4`** directory.

# Sample outputs

You are supplied with a `main.cpp` file to help with testing your Matrix method implementations. You can test and debug your program from VS-Code, by setting command-line argument as demonstrated in the `OSC OnDemand & VS-Code demonstrations` page on Canvas. See video titled `Using command-line arguments in VS-Code`.

**Base case #1 (load matrix)** – must function correctly to earn any points:

```
$ ./homework4 '<<' mat3x3.txt
3 3
1 -1 3
-2 4 7
4 2 3
```

**Base case #2 (matrix addition)** – must work to earn any points:

```
$ ./homework4 '+' mat3x3.txt mat3x3_2.txt
3 3
2 0 0
0 0 14
8 9 6
```

**Additional operator (matrix subtraction)**:

```
$ ./homework4 '-' mat3x3.txt mat3x3_2.txt
3 3
0 -2 6
-4 8 0
0 -5 0
```

**Additional operator (multiplication)**:

```
$ ./homework4 '*' mat3x3.txt mat3x3_2.txt
3 3
1 -1 -9
-4 -16 49
16 14 9
```

**Additional operator (matrix dot/multiplication)**:

```
$ ./homework4 'dot' mat2x3.txt mat3x2.txt
2 2
58 64
139 154
```

**Additional operator (matrix transpose)**:

```
$ ./homework4 'trans' mat2x3.txt
3 2
1 4
2 5
3 6
```

**Additional operator (matrix apply)**:

```
$ ./homework4 'sigmoid' mat3x2.txt
3 2
0.999089 0.999665
0.999877 0.999955
0.999983 0.999994
```

# Turn-in:

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure you have tested operations of your program with different test files and command-line arguments. Once you have tested your implementation, upload your header and source file to Canvas.