Sophie Reese-Wirpsa (smr2225)
Diego Prado (dtp2118)
Cindy Espinosa (cje2127)
Desu Imudia (aii2003)
Emily Ringel (edr2124)

PLT Project Proposal
Spring 2021

Language Name: MatrixMania (MM)

Description of language:

The goal of our language is to ease the manipulation of matrices. A few features of our mathematical language are built-in matrix addition, subtraction, and multiplication as well as array programming.[1] Array programming would allow the user to apply operations of a primitive against an entire matrix, an entire row of a matrix, or an entire column of a matrix. Our language will rely primarily on a matrix data structure, and to complement this we will replace strings and arrays with 1-dimensional matrices. We plan to use python-like syntax and static typing and will also be enabling type-hinting in order to better indicate the type values within our code. Our language will be derived from Java.

Sorts of programs:

Our language will be used to write programs to manipulate and solve matrices. These programs could include algorithms to invert an n-by-n matrix or to put a matrix into reduced row echelon form by Gauss-Jordan Elimination. These types of algorithms would benefit from many of the features in our language, such as the built in ability to multiply a matrix or part of a matrix by a primitive type value (by array programming) or syntax to make indexing more readable. Array programming, for example, would simplify the steps of Gauss-Jordan elimination where it is necessary to multiply a row by a scalar by avoiding the use of a loop. Our language will also

replicate the basic operations and types from Java, which will make it possible to run most basic algorithms, including GCD and Hello World.

Parts of the Language:

Our language is not object-oriented nor does it use generics.

Garbage Collection:

Our language does not have automatic garbage collection. When making new matrices, the user must call a method, createMatrix(), that will in turn call malloc() and allocate space on the stack. When the user is done with the matrix, they must call freeMatrix() which will call free() and release the allocated space.

Typing:

Our language will be statically scoped with a weak type strength and be statically typed. This will let us change between primitives more easily.
Type-hinting will be a part of our language. We will be adding type information to functions and methods for annotating both the arguments and the return value.

Primitives:

Newly declared variables do not have to have a type before them. Our language has weak type strength allowing variables to be declared and switch types. Our primitive types will be int, float, bool, and char.

Matrices:

The basic building block of our language. The indices will start at 1. Matrices can only contain elements of the same type. If not, the elements will be cast to the same type. Matrices will be able to be sliced like python/numpy arrays. Since our language does not use arrays or strings, one row matrices will be used instead.

Control Flow:

Control flow statements resemble their counterparts in Python. Examples include:

Conditionals:

if *expression*:

statement

elif *expression:*

statement

elif *expression:*

statement

.

.

.

else:

statement

Loops:

while *expression*:

statement

for *target* in *iterable:*

statement

Built in Operations/Functions:

Normal arithmetic operations between number types: +, -, *, /, %

Matrix operations: +, - are normal matrix operations. * is matrix multiplication.

Array Programming:

Array programming would allow the user to apply operations of a primitive against an entire matrix, an entire row of a matrix, or an entire column of a matrix. This reduces the logical complexity of many programs.

Source Code:

Inverting a Matrix:

The sample code below includes a main executable function that creates a matrix and prints out the resulting matrix after inverting the original. Using the function *getMatrixInverse()*, we can see through the type hinting that it will take in an original matrix m and return an inverted matrix.

```
class SourceCode:
     '''Given a 2x2 matrix of integers m,
     Return the inverse of the original matrix m.'''

     def main(args[]):
          matrix m = createMatrix([1, 2; 3, 4])
          print(getMatrixInverse(m))
          freeMatrix(m)

     def getMatrixDeterminant(m: matrix[int]) -> int:
          return (m[1,1] * m[2,2]) - (m[1,2] * m[2,1])

     def getMatrixInverse(m: matrix[int]) -> matrix[float]:
          int determinant = getMatrixDeterminant(m)
          if determinant == 0:
               return [-1]
          return 1/determinant * [m[2,2], -1*m[1,2];
                                  -1*m[2,1],  m[1,1]]
```