Rule	Snippet	Description

params_list_opt	test() {
	 return 0;
	}
		We can always choose to pass no arguments

params_list	a a, b a, b, c	Single argument in a list. Multiple arguments in a list. And so on.
return	return 0.0; return a+b;	All programs end with a return statement. It specifies the result of the program. The return value is always of type float.
statements_opt		A program may have zero or more statements.

statement: ID ASSIGN expr SEMI	a = 5;
	b = 6;
	c = (a+b)/2.0;
		There are two kinds of statements: assignments and matrix declarations.
		
		Assignments mostly work as you expect. Variable names need not be defined in advance. c is a new variable set equal to the average of a and b. You are not required to reserve memory for c. It can simply be kept in a register.
		
		a and b are assigned integers, but we assume that integers are always converted to float type. So before assigning an INT to a or b, they are converted to float.

dim: LBRACKET INT X INT RBRACKET	[2 x 2]	Dimensions of a matrix. Always 2D only. First number is rows, second number is columns. The number of rows and columns will always be

		literals. *Only support for 2x2 square matrices.*
matrix_rows: matrix_row | matrix_rows COMMA matrix_row ;	{[1,2,3], [3,4,5], [a,b,1+1]}	A matrix_row is a bracket delimited list of expressions. matrix_rows is multiple matrix_row delimited by curly braces. The example shows a 3x3 matrix initialized with constants, arguments, or expressions.
statement: ID ASSIGN MATRIX dim LBRACE matrix_rows RBRACE SEMI	m = matrix [2 x 2] { [1,0], [0,1] };	This statement lets us declare a matrix. The matrix keyword lets us know we are creating a matrix, the dim rule tells us its dimensions (always 2x2), and the matrix_rows tells us how to initialize it.
expr		The expression rule performs all the computations. Each expr computes a result that is either a matrix or a float.
expr: ID	a = x;	In this case, x is the expr and it will be assigned to a. When we see x, we must look-up the value previously assigned. If x is a matrix, a becomes the same matrix, too. If x is a float, then a becomes the same float.
expr: FLOAT	3.5	A literal floating-point number in the program.
expr: INT	3	A literal int. To match expr, it should be converted into a float.

expr: expr PLUS expr	// add two floats
	
	c = a + b;
	
	// add two matrices
		Let a,b, and c be floats, and let m1, m2, and m3 be matrices previously assigned.
		
		We can add two floats or two matrices. But, we do not allow a

	m3 = m1 + m2;	mixed addition between a float and a matrix.

expr: expr MINUS expr	c = 1.0 - 5.0;
	
	m1 = m2 - m3;
		Subtract two floats or two matrices. But, we do not allow a mixed subtraction between a float and a matrix.

expr: expr MUL expr	c = a*b;
	
	m3 = m1 * m2;
		Multiply two floats or two matrices, there is no mix of matrix and float operations.
		
		Let m1, m2, m3 be matrices. a,b, and c have float type. In this case, c will be the product of two floats.
		
		m3 will be a matrix in which it is the 2x2 product of m1 and m2. Since we know the dimensions, we can verify that the multiplication is legal at compile time.

expr: expr DIV expr	c = a / b;	Divide two floats.

expr: MINUS expr	a = -d;
	
	m1 = -m2;
		Take the negation of a float or a matrix. In a matrix, each element is negated.

expr: DET LPAREN expr RPAREN	d = det (m1);	Calculate the determinant of a matrix, m1. The result is a single float value *Only 2x2 matrices.*
expr: ID LBRACKET INT COMMA INT RBRACKET	a = m1[i,j];	Look-up at the element in matrix m1 at row i and column j and assign it to a.
expr: REDUCE LPAREN expr RPAREN	sum = reduce(m1);	Add up all of the elements in m1 and assign it to sum.