The Matcat Language Proposal

Davit Barblishvili, Mariam Khmaladze, James Ryan, Sarah Huang

1 Overview of Language - Sarah

The Matcat language is an imperative, mathematically-inclined language whose goal is to aid in the writing and computation of linear algebra operations. The implementation of these functions will occur in similar ways to the Wolfram language (see https://reference.wolfram.com/language/guide/MatricesAndLinearAlgebra.html). Matcat aims to mimic functionality from Java and C.

Our primary goals are:

- To simplify operations on matrices. Examples of these include row reduction, computing transformations, finding eigenvalues/eigenvectors, diagonalizing/transposing a matrix, performing the Gram-Schmidt process to find orthogonal bases, and others.
- To achieve Java-like syntax
- To allow users to write readable code; matrix/vector structures and the operators must mirror mathematical symbols and read intuitively
- Compile time optimizations (static typing)

2 Language Details - Davit and James

- Data Types and Operations

Matcat will include the core data types of Java - int, char, float, string, boolean. In addition, the data types matrix and vector will be implemented in order to help programmers perform linear algebra and optimize their operations at compile time. See table below for description of operations. In line with Java, Matcat will be explicitly typed.

Data Type	**Description**	**Operations**	**Examples**
int	4 byte integer	=, ==, !=, +, -, *, /, %, ++, –, >>, <<, +=, -=, <, >, =<, >=	4+5 → 9 9 == 9 → True 9 =< 1 → False 2 << 1 → 4 int x = 16;
char	1 byte data type holding character	=, ==, !=, +, ++, –, <, >, =<, >=	‘A’ == ‘A’ → True ‘A’ + 32 → ‘a’
boolean	1 byte, evaluates to True/False	=, ==, !=, !, &&, ||	True || False → True True && False → False
string	Array of chars	=, ==, !=, <, >, =<, >=, +	“Hello” > “World” → True “a” + “b” → “ab”
float	A 8-byte floating	=, ==, !=, +, -, *, /, %, ++, –,	3 + 4.0 = 7.0

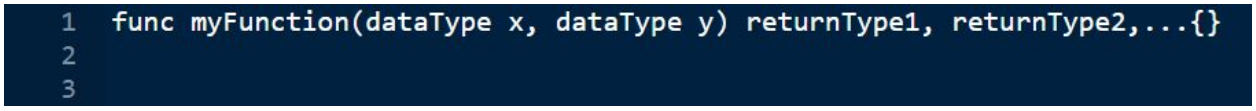
	point number bytes	+=, - =, <, >, =<, >=	
vector	Analogous to array - holds a list of int or float; immutable size once instantiated, but indexes can be reassigned	+ Vector addition - Vector subtraction * Vector Multiplication . Dot product x Cross Product *. Product with a scalar +. Addition by a scalar == Vector equality	[1, 0] + [0, 1] → [1 1] [2, 3] . [3, 4] → 18 [1, 2, 3] x [4, 5, 6] → [-3, 6, -3] 5 + [10, 5] → [15, 10] Indexing v[0]
matrix	Essentially a 2D array, composed of vectors; immutable dimensions after declaration	* Product of two matrices * Product with a vector *. Product with a scalar +. Addition by a scalar + Matrix addition - Matrix subtraction == Matrix equality	[[1, 2], [3, 4]] * [[5, 6], [19, 22]] → [[19, 22], [43, 50]] [1, 2] *v [[6, 7], [2, 3]] → [20, 8] m[0] → returns vector

- Keywords

The following words are reserved in Matcat and can not be used for any other purpose outside of their defined functionality → {while, for, if, else, return, true, false, int, float, char, bool, string, new, print}. In addition to these typically reserved words, we are going to add a few new keywords that would be useful to efficiently manipulate the matrices → {inverse, transpose, scalar, dim, det, rref}

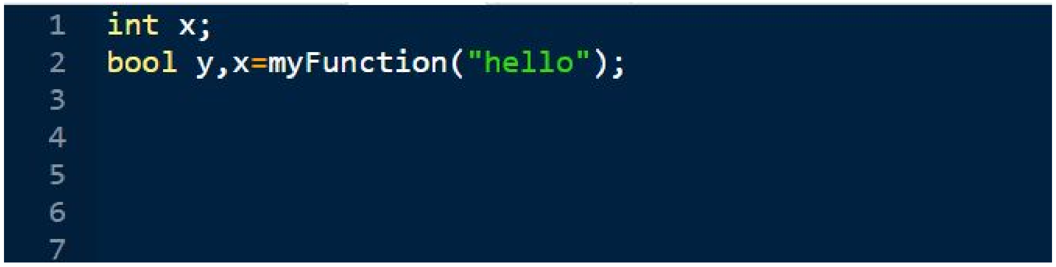
Keyword	**Description**	**Example**
inverse	A function that is going to take a matrix data type and return a new matrix in inverse form.	inverse(matrix1) → matrix2;
transpose	A function that is going to take a matrix data type and return a new matrix in its transpose form.	transpose(matrix1) → matrix2;
scalar	A function that is going to take two arguments - a scalar, and matrix - and perform scalar multiplication and return a new matrix.	scalar(matrix1, 3) → matrix2;
dim	A function that takes one argument - a matrix - and returns a dimension of it.	dim(matrix1) → 2X3.
det	A function that takes one argument - a matrix - and returns a determinant value of it.	det(matrix1) → -4;
rref	A function that takes one argument - a matrix - and returns boolean value {true/false}. rref determines whether or not a matrix is in rref form.	rref(matrix1) → True;

- Control flow
  - The following keywords are reserved for control flow, and work like in Java: if...else, while, for().
- Function/function declaration
  - Matcat supports function declaration in the following syntax:



```
func myFunction(dataType x, dataType y) returnType1, returnType2,...{}
```

  - We are going to allow the returning multiple data types; explicit order of the return types is going to be in the function declaration. Means of capturing multiple return types:



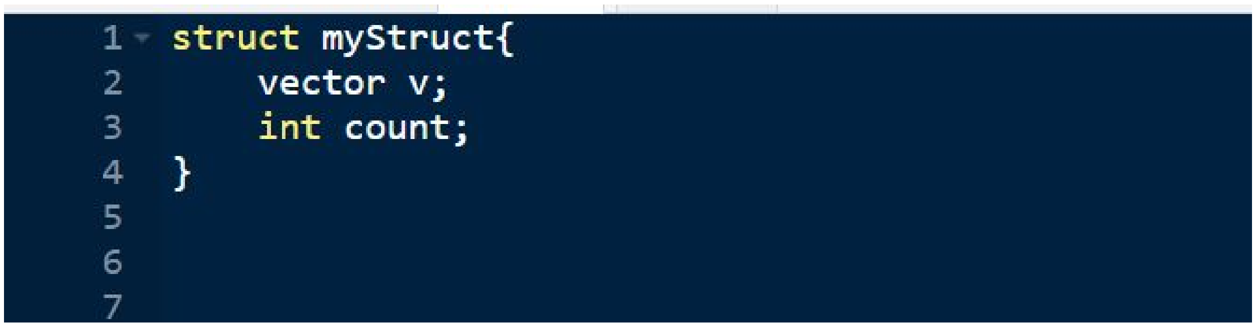
```
int x;
bool y,x=myFunction("hello");
```

- Previously declared x can be assigned, while y must be explicitly typed. This order must match that of the function declaration.

- Comments
  - Single line comment → // This is a comment line
  - Multi line comment → /* this is a comment line but over multiple lines */
- Memory

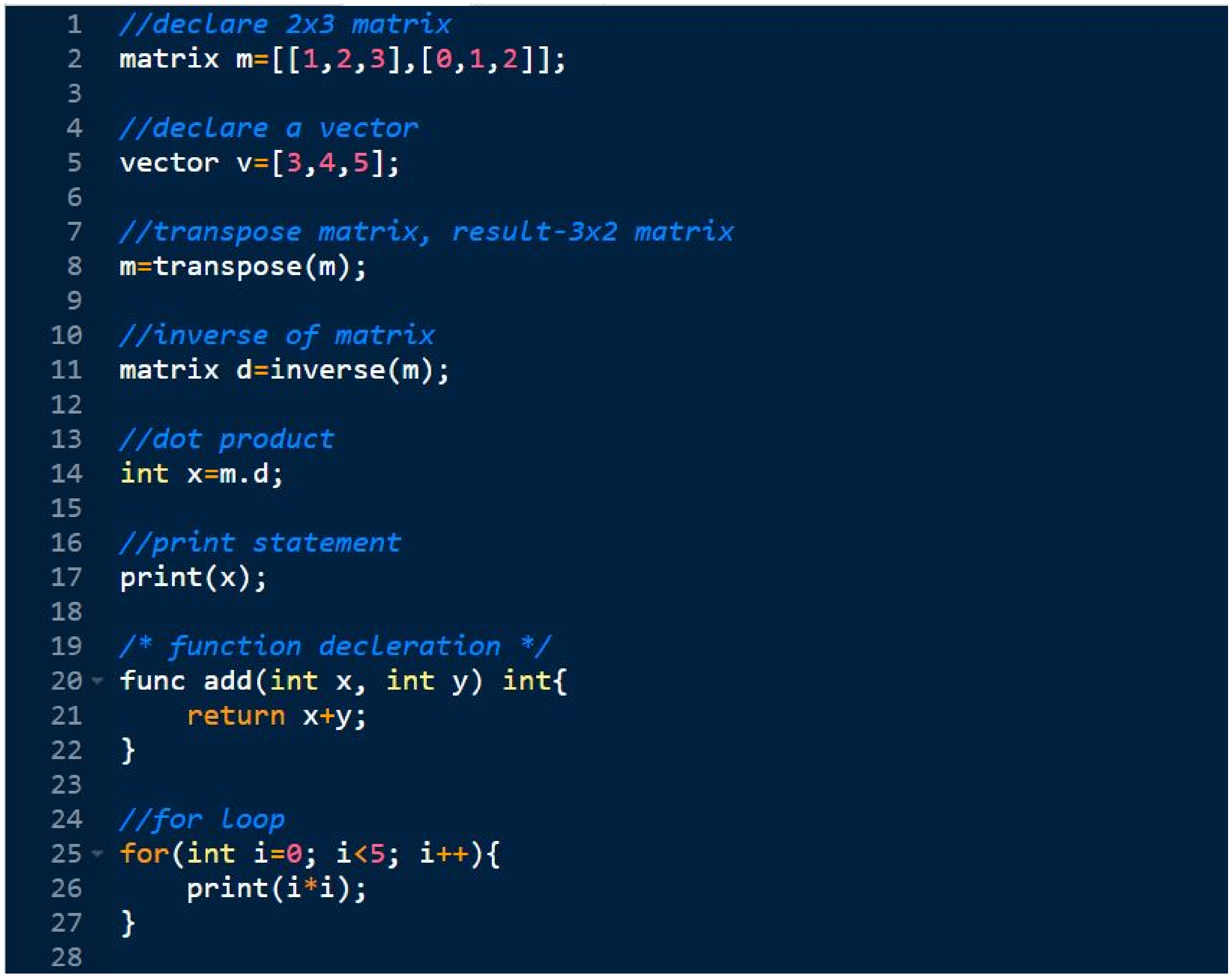
- The Matcat language will be ”pass by reference,” as in Java, and we are going to implement a memory management system that will be handled internally by a simple garbage collector (just like in Java).
- To allocate data on the heap, we are going to use a keyword → new.

- User Defined Types
    - Matcat will allow for lightweight structs, similar to C. Example:



```
struct myStruct{
    vector v;
    int count;
}
```

3 Examples - Mariam



```
//declare 2x3 matrix
matrix m=[[1,2,3],[0,1,2]];

//declare a vector
vector v=[3,4,5];

//transpose matrix, result-3x2 matrix
m=transpose(m);

//inverse of matrix
matrix d=inverse(m);

//dot product
int x=m.d;

//print statement
print(x);

/* function decleration */
func add(int x, int y) int{
    return x+y;
}

//for loop
for(int i=0; i<5; i++){
    print(i*i);
}

```