CMAT Language - Project Proposal
COMS 4115

Language Guru: Michael Berkowitz (meb2235)
Project Manager: Frank Cabada (fc2452)
System Architect: Marissa Ojeda (mgo2111)
Tester: Daniel Rojas (dhr2119)

1. Introduction and Motivation

Our team's shared interest in mathematics led us to decide on creating a language centered around matrix manipulation. Matrices are widely used to represent data and mathematical equations in many academic fields ranging from robotics to linear algebra. Our language, CMAT, aims to make matrix manipulations and other such linear algebra operations easier for applications. CMAT is inspired by C and MATLAB, taking the best parts of both to produce a language with high versatility.

Ideally, we want to allow easy, efficient computation and matrix operations without sacrificing the structure of a full programming language. Some other potential applications of our language include finding eigenvalues and eigenvectors, finding the inverse of a matrix, performing linear transformations on vectors, and solving numerical methods.

2. CMAT Program Types

As our introduction hinted, the types of programs we are targeting with this language are similar to those best written in MATLAB. The main ones that come to mind are programs related to linear algebra problems which involve working extensively with matrices. We want to make it simple for users to solve these problems with CMAT while also allowing them to create more complex programs with the basic tools we provide. CMAT could also be used for programs to plot statistical data if paired with a proper graphical interface package that could take advantage of our matrix structure. In addition, the simplicity of CMAT will not restrict users to specific mathematical problems but instead will allow them to also solve diverse problems as they would in other languages such as C or Java.

3. Language Overview

Primitive Data Types

Name	Description
`int`	Integer
`char`	Character
`bool`	True and False
`float`	32-bit floating point number
`double`	64-bit floating point number
`null`	Absence of data
`void`	Used for functions that do not return anything

Supported Data Types

Name	Description
`String`	An array of chars
`[]`	Define a matrix

Basic Keywords

Name	Description
`for`	Iteration until condition not met
`while`	Loops until condition is not met
`if`	Dynamic if accepts 1/0/True/False/null
`else`	Paired with an ‘if’ statement
`/* */`	Block comments
`//`	Single line comments
`main`	First function executed in a program
`return`	Returns a value of a function

Operators

Name	Description
`=`	Assignment operator
`+, - , *, /`	Arithmetic operators
`++`	Increment operator
`--`	Decrement operator
`>`	Greater than operator
`<`	Less than operator
`>=`	Greater than or equal to operator
`<=`	Less than or equal to operator
`==`	Returns 1 if values are equal, else returns 0
`!=`	Returns 0 if values are equal, else returns 1
`&&`	Logical AND operator
`||`	Logical OR operator
`!`	Logical NOT operator

Matrix Operators

Name	Description
`+, - , *, /`	Matrix arithmetic operations and scalar arithmetic operations
`[x:y:z]`	Make a 1-by-n matrix from x to z with a delimiter of y
`[x,:]`	Access specific row of a matrix
`[:,y]`	Access specific column of matrix
`<,<=,>,>=`	If 2 matrices have the same dimensions, these operators compare element by element

4.) Examples and Sample Program

```
//Variable declaration
int x = 1;
char letter = 'A';
bool flag = false;

//Matrix declaration for 3x3 matrix literal
Matrix m = [1 2 3; 4 5 6; 7 8 9]

//Create matrix
Matrix x = [1 : 2 : 10]; // x = [1 3 5 7 9]
Matrix Test = [1 2 3; 4 5 6; 7 8 9];
Test[0,:]; //Access row 0 (first row)
Test[:,2]; //Access column 2
```

Example Determinant Function:

```
int determinant(int squ_mat[])
{
    int sum = 0;
    int rows = r_size(squ_mat);
    int cols = c_size(squ_mat);

    // check that squ_mat is an n x n matrix with n > 0
    if(rows != cols || rows == 0 || cols == 0) return -1;
    // Base case
    if(rows == 1) return squ_mat[0,0];
    // Recursion
    else
    {
        for int i = 0:(r_size(squ_mat)-1) {
            sum += parity*m[0,i]*determinant(rm_c(i, rm_r(0,m)));
            parity = (-1)*parity;
        }
        return sum;
    }
}

void main(){
      print(determinant([ 1 2 ; 3 4 ]));  // -2
}
```