Daisy Chaussee: dac2183

Anthony Kim: ak3703

Rafael Takasu: rgt2108

Ignacio Torras: it2216

DARN

Language Reference Manual

10/25/16

1. Introduction

DARN is a Java-like matrix manipulation and image processing language. Named after the first initials of our names, *DARN* allows for efficient linear algebra calculations and easy access to rows and columns in matrices, both in 2D and 3D. For example, programmers can use *DARN* to create a program that finds the eigenvalues and eigenvectors of a matrix. Matrices and tuples prove essential to *DARN*'s image processing capabilities. Using geometric operations and matrix manipulation, *DARN* acts like a mini Photoshop, enabling a user to edit images through saturation, desaturation, lightening, color-correcting, cropping, scaling, resizing, and rotating.

2. Types

Type	**Description**
int	32-bit integer data type, represented as binary signed 2's complement bitstring internally
float	single-precision floating point number, floating point constants contain a decimal point or an exponent or both
char	1 byte character data type
bool	1 byte Boolean data type, 0 represents false and 1 represents true internally
string	string data type made up of chars
tuple	An ordered sequence of values
matrix	Matrix data type, all elements of a matrix must be of the same type, only allows int or float values

3. Lexical Conventions

3.1 Identifiers

An identifier can be made up of upper and lower case letters and digits, but the first character must be a lowercase letter. Upper and lowercase letters are distinct, so y and Y are two different

names. The underscore "_" counts as a letter. Keywords like **if, elif, int, float**, etc. may not be used as variable names.

3.2 Keywords

Keyword	**Description**
for	for in a for statement*
if	if in an if statement
elif	elif strings together multiple if statements
else	else in an if-else statement
return	return function value
void	no type
while	while in a while statement
true	Boolean literal value for true
false	Boolean literal value for false
break	Breaks out of a loop

* see section regarding statements

3.3 Comments

Comments are denoted by

```
/* this is a comment */
/*
  this a
  block comment
*/
```

3.4 Operators

Operators	**Description**
=	assignment
*	multiplication

/	division
%	modulus
+	addition
-	subtraction
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
==	equality
!=	inequality
&&	logical AND
| |	logical OR
!	logical NOT
;	statement separator
{ }	block separator
( )	i.e. if (statement)
,	separates list of values, esp. in tuples

Matrix Operators	**Description**
,	list of values
[ ]	encloses an entire matrix
[[ ]]	double brackets for 3D matrix
[ ]	matrix indexing for rows and columns
( )	encloses a tuple within a matrix
+, -, *, /	matrix scalar operations

3.5 Precedence

Precedence	Expressions and Operators
lowest	=
	| |
	&&
	==, !=
	>, <, >=, <=
	+, -
	* , /
	!
highest	function and matrix declarations <func_name>(arg1, arg2…) <matrix_name>[ ] [ ]

4. Syntax

4.1 Expressions

Assignment operators are binary operators with right-to-left associativity. Arithmetic expressions represent mathematical operations with left-to-right associativity.

Matrix Arithmetic Expressions

Matrix scalar multiplication:

M1 = M1 * 3

Matrix multiplication:

M2 = M1 * M1

4.2 Declaration and Initialization

Basic Data Type Declaration and Initialization

All variables must be declared before use. A declaration specifies the variable type and the variable name. A variable may also be initialized in its declaration.

Examples:

type variable_name;

-OR-

type variable_name = literal;

Variables may also be declared in separate lines, such as

type variable_name;
variable_name = literal;

Examples:

int x = 5;
float y = 1.2;
bool b = true;

Tuple Declaration and Initialization

Tuples must be of the same type. Tuples are declared in the format:

type tuple variable_name;

And initialized in the format:

variable_name = (literal, literal,...);

Tuples may also be initialized in the same line as declaration, for example:

type tuple variable_name = (literal, literal, …);

Matrix Declaration and Initialization

Matrices can only be made up of ints or floats, and all elements must be of the same type.

Declaring a matrix:

To declare a matrix, one must use the format:

type matrix variable_name;

To initialize:

variable_name = [1, 2, 3][1, 2, 3];

Or, matrices can be declared and initialized in the same line:

type matrix variable_name = [1, 1, 1][1, 1, 1];

Examples:

/* 2D Matrix */

int matrix m2 = [1, 2, 3][1, 2, 3];

/* 3D Matrix */

```
int matrix m3 = [[(0,0,0), (1,1,1)], [(2,2,2), (3,3,3)]];
```

5. Control Flow

5.1 Statements and Blocks

Statements are always followed by a semicolon.

```
int a = 1;
```

Blocks are surrounded by semicolons { }.

```
if (condition) {
        i++;
}
```

Blocks don't need to be followed by a semicolon

5.2 If-Elif-Else

The syntax for if-elif-else statements:

```
if (condition) {

} elif (condition) {

} else {

}
```

The else statement is not necessary. The if condition will be tried first, then the elif conditions in order, until one condition matches. If none matches, then it will run the block under else.

5.3 Loops

While Loop

```
while (condition) {
        i++;
}
```

A while loop will run the code inside the while block as long as the condition is true. The loop will not start if the condition is not met to begin with. If the condition is something that will always be true like 1==1 then you will have to break out of the loop with “break;” .

For Loop

```
for (expr1;expr2;expr3) {
```

```
        print(i);
}
```

The first and third expression are assignments or function calls, while the second is a relational expression. It calls the first expression then makes sure the second condition is true and then every time the code block is run, the third expression runs. If the second expression is missing then the loop will run forever until broken.

5.4 Break and Continue

Break

```
while ( condition) {
        break;
}
```

The break will immediately exit you from the innermost loop it is in.

Continue

```
for (int i=0;i<10;i++) {
        if (i == 2) {
                continue;
        }
        print(i);
}
```

The continue statement causes the loop to go to the next iteration, for example above the loop will not print the number 2 because it will go to the next iteration before it gets to print.

6. Functions and Program Structure

6.1 Functions and Function Calls

Functions are declared by first stating the return type of the function, and then the function name. The arguments of the function need to have types:

```
<return type> <function name>(<arg1 type> <arg1 name>) {
        return <return value>;
}
```

Examples:

1)

```
        int matrix_average(int[][] matrix) {
```

```
        /* compute average */
        return average;
    }
```

2)

```
    /* function that returns void*/
    void foo(int[] matrix) {
        /* execute */
    }
```

In order to be able to call a function, the function must have been declared and implemented before. The function call will execute using the given arguments and return the value defined as the return type during its declaration. All arguments will be passed by value, so a function can change the values of the parameters within the scope of the function block without affecting the arguments in the function call.

Some functions that DARN will include:

- Scalar: multiply each entry in matrix by some number
- Multiplication: multiply two or more matrices together
- Concatenation of Matrices
- Gaussian Elimination (i.e. row reduction)
- Eigenvalues Calculation

6.2 Scope Rules

The scope of an automatic variable declared at the beginning of a function is the function in which the name is declared. Local variables of the same name in different functions are unrelated.

Examples:

1)

```
    void foo() {
        string s = "i am a string";
        print(string); /* should print "i am a string" */
    }
    void bar() {
        print(string) /* should result in an error */
    }
```

2)

```
void foo() {
        string s = "hello";
        print(s); /* should print "hello" */
}
void bar() {
        string s = "world";
        print(s); /* should print "world" */
}
```

6.3 Block Structure

Variables may be defined in a block-structured fashion within a function. Declarations in blocks hide any variables that are declared outside of the block:

```
void foo() {
        string s = "hello";
        int i = 2;
        if (i == 2) {
                string s = "world";
                print(s); /* should print "world" */
        }
}
```

6.4 Recursion

DARN functions may be used recursively:

```
int factorial( int i ) {
        if ( i < 2) {
                return 1;
        }
        else {
                return i * factorial(i-1);
        }
}
```