COMS 4115: NumNum Language Reference Manual

Programming Languages and Translators

COMS 4115 W Section 1

Prof. Edwards

October 15, 2017

Sharon Chen	syc2138	Tester
Kaustubh Gopal Chiplunkar	kc3148	Language Guru
Paul Czopowik	pc2550	Manager
David Tofu	dat2149	Tester
Art Zuks	az2487	System Architect

Table of Contents

Introduction

NumNum is a programming language which is based on C and Python languages. It is designed to be domain specific matrix and array manipulation language. NumNum differs in syntax and encapsulates the best of C and Python and some other common languages to deliver a fun and easier programming experience for a user.

Lexical Conventions

White space

White space is used to separate tokens in the language and is otherwise ignored. The programmer is free to use space, tab or newline characters to make code more readable.

Comments

The character `/*` marks the start of a string and the character `*/` marks its end.

Identifiers for functions and variables:

An identifier is a sequence of letters and digits and the first character must be alphabetic. The underscore _ counts as alphabetic. Upper and lower case letters are considered different.
Declared more formally as :

```
['a'-'z']['a'-'z' 'A'-'Z' '0'-'9' '_'  ]*
```

Keywords:

- `int`
- `float`
- `str`
- `while`
- `for`
- `if`
- `elsif`
- `else`
- `print`
- `void`
- `shape`
- `dims` (# dimensions)
- `func`
- `continue`
- `break`
- `return`

Constants

The language contains the following constants:

- integer
- floating point number
- string

Integer Constants

An integer constant consists of a sequence of digits. The language recognizes decimal numbers only and does not recognize binary, octal, hexadecimal or other number systems. Integer constants are signed by default. To represent a negative integer, the minus sign is used. Leading zeros are ignored.

Example:
```
int a = 456
int b = -12
```

Floating Point Constants

Floating point constants consist of the integral part in form of a sequence of digits, a period and a fractional part which is also a sequence of digits. The language recognizes decimal numbers only and does not recognize binary, octal, hexadecimal or other number systems. For the integral part, leading zeroes are ignored and the number can be signed with a minus sign.

Example:
```
float a = 456.789
float b = -12.0
```

String Constants

A string constant is a sequence of characters enclosed by double quotes “” and terminated by a null byte \0 to indicate the end of the string. Strings are not parsed for comments and The backslash \ is used for escaping characters in the string.

Escape Characters:

- \ - Escape Character
- \n - newline Character
- \t - Tab Character
- \\ - Backslash
- \” - Quote

Example:

```
str name = “John Doe”;
str x = “10 \t 20 \”Inch\””;
str example = "example string /* this is not a comment */ \" still in
the string"
```

Syntax

The semicolon `;` is a statement terminator.

```
print (“Hello, world!”);
```

Code Blocks

Code blocks are enclosed by curly braces `{ }`

Functions:

Function has a return type and has arguments. A function cannot return a matrix but can return other data types. Matrices can only be passed by reference in a function. A function is defined by calling the keyword 'func' before it is declared.

Syntax:

```
/* Function Declaration */
func type name (list of parameters){
      statement;
      return statement;
}

/* Function Call */
name (list of parameters);
```

Example:

```
func int add(int a, int b){
      return (a + b);
}
```

Control Flow

Control flow is achieved by loops and conditional statements. Loops can also use special control flow statements `continue` or `break`.

Loops

There is are two ways to implement loops, a for loop and a while loop:

For Loop

Syntax:

```
for (expression; condition expression; increment expression) {
      statement;
}
```

While Loop

Syntax:

```
while (condition expression) {
      statement;
}
```

Continue

The `continue` statement is used to skip the remaining statements in the current iteration of the loop and begin the next iteration. Can be used for for and while loops.

Break

The `break` statement is used to stop the execution of the loop. Can be used for for and while loops.

Conditional Statements

Conditional statements are handled by using if, elsif and else.

Syntax:

```
if (expression){
      expression;
}elsif (expression){
      expression;
}else {
      expression;
}
```

Operators

Binary Operators

+	Addition of two 32-bit integers/ two 32-bit floats
-	Subtraction of two 32-bit integers/ two 32-bit floats
/	Division two 32-bit integers/ two 32-bit floats
*	Two 32-bit integers/ two 32-bit floats
%	Modulus of two 32-bit integers/ two 32-bit floats
==	Equality Check
!=	Inequality Check
>	Greater Than Operator
<	Less Than Operator
>=	Greater Than or Equal Operator
<=	Less Than or Equal Operator
&&	Logical And
||	Logical Or

Unary Operators

-	Written before in int/float to make it negative
!	Logical Not

Assignment Operators

=	Assignes the right hand value to the variable on the left

Operator Precedence

[ ],{}	Highest
!	
* , / ,%	
+, -	
>,<,<=,>=	
== , !=	
&&	
||	
=	Lowest

Matrices

Matrices are not are a primitive datatype. Each matrix has attributes `dtype`, `size`, `shape`, and `dimension.` Matrices are stored in the memory sequentially allowing easy index-based access.

Declaration:

```
int mat [dim1][dim2];
float mat [2][3] = [[1,2],[2,3],[3,4]];
```

Standard Matrix Library

Some built-in functions in the matrix library

```
print(expression);
```

Prints the expression as a string to standard output. Accepts strings.

```
sort(type, algorithm);
```

Returns the sorted array type by reference. Uses the sorting algorithm as denoted by the input field “Algorithm”. If the requested sorting algorithm is not implemented in the library, uses merge sort.

```
dim(expression)
```

Returns an integer of the dimensions of the input expression.

```
shape(expression)
```

Returns an array with the size of each dimension.
Eg:

```
     int u [2][2] = [(1,2),(3,4)];
     dim(u);  /* Returns 2 */
     shape(u); /*Returns [2,2]*/
```

```
matrix(dimensions,type)
```

Returns a matrix of the needed dimensions and data type which is initialized by zeros. Dimensions have to be constants and not identifiers themselves.

Example:

```
     float mat [3][4] = matrix(3,4,float);
```