String **C**omputation Progr**am**

Reference Manual

Scott Pender
scp2135@columbia.edu

COMS4115 Fall 2012

10/31/2012

1 Lexical Conventions

There are four kinds of tokens: identifiers, keywords, expression operators, and other separators.

1.1 Identifiers (Names)

An identifier may consist of a sequence of letters and digits. Special characters are not allowed. Identifiers are used to specify function and variable names. Reserved keywords may not be used as identifiers.

1.2 Keywords

The following keywords are reserved for use as keywords and may not be used otherwise:

FOREACH	int	MAIN
WHILE	str	OPEN
IF	str[]	

1.3 Comments

The ? character denotes a single-line comment. The ? must be used independently, outside of quotation marks. Multi-line comments are not available.

1.4 Line Terminators

The line feed character is used to terminate a line. More simply, there is no formal line termination character; lines are terminated automatically when a new line is fed to the parser.

1.5 Whitespace Characters

As noted previously the line feed character (\n) is used to denote line termination. The tab character (\t) denotes ownership of statements to the last statement that used one less tab character. This is used in cases of control statements and functions where the statements to be executed are grouped with by indentation. Control statements may be nested within functions, but functions may never be nested (only function calls). The space character ‘ ‘ is generally ignored, though it may be required to separate otherwise adjacent identifiers.

```
EX:     myFunction(X,Y)
            IF(1<2)
                @=”Hello world”
            @=”If they are strings, the concatenation of X & Y is ”X Y
```

2 Scope

2.1 Programs

A program may open other program files to utilize their libraries. Aside from native scam libraries, which may never be overridden, the most local definition of a function or variable is the one that will be used.

2.2 Functions

All functions are always accessible from within the defining program and from any program that accesses the defining program. There are no access modifiers.

2.3 Variables

Variables are inherited by nested function calls. If a variable is not overridden with a local variable during a function call, the inherited value is used in any reference to that variable.

3 Programs

Each program in scam consists of a document containing one MAIN function and zero to many helper functions.

A program may open other programs to access their function libraries using the OPEN keyword as follows: OPEN(@./filename.scam)

Arguments in the MAIN function may be accessed using @1, @2, etc.

4 Functions

A function is the definition of a sequence of statements to be executed in a specific order. Functions may accept input variables and return a value. Functions may also be called recursively.

4.1 Function definition

The following example demonstrates a function definition. The method signature is for a function 'myLabel' that accepts parameters X,Y, and Z. Note that Z must be an integer in this definition. Tab is used as the delimiter to denote the definition of a function. Within the function definition, the function label may be set like a variable to indicate a return value. The default return value is "". Since this is set like a variable, it can be used to make recursive calls to itself.

```
Ex:     myLabel(X, Y, int Z)
            ? Impl of myLabel
            myLabel = "optional returnValue"
```

4.2 Function call

Functions may be called from within any other function in a program. The example below demonstrates a function call.

```
Ex:     myLabel (X, Y, Z)
```

Implicit return from function (can return string or integer) can be set as value of another variable or sent to output (console, file

```
Ex:     @ = myLabel (X,Y,Z)
```

5 Variables

Variables are dynamically typed. A variable can be set to point to an integer, array, or string. A variable can later be re-defined to be of a different type. However, variables must still be referenced and defined properly according to the rules for each type. Type reference issues may be solved using static typing through explicit type enforcement. Variables are set using the '=' character with the variable identifier on the left-hand-side and the intended value on the right-hand-side. The following scenarios demonstrate variable definitions and their resulting values.

- X = 2
  - Define variable X as an integer value of 2
- X = "2"
  - Define variable X as string value of "2"
- XY="strinng"
  - Define variable XY as string "strinng"
- XY = []
  - Define XX as an empty array
- |X|
  - Length of value in X.
    - If string, return string length
    - If array, return array length
    - If integer, return integer value
    - Default return value for undefined variables is 0

6 Types

6.1 Strings

Strings, the primary focus of the scam language, may be compiled into an array of strings and may themselves be referenced as an array of (individual character) strings. Strings may be composed of any characters except for quotes themselves (or string formatting characters).

Stings are surrounded by quotes (" ") at all times. Everything between an opening and closing quote is part of that string. There is no need to escape characters within a string as everything within quotes is valid except for string formatting characters. String formatting characters, ~N for new line and ~T for tab character, may not be escaped.

String values may be concatenated using the string concatenation operator.

Whenever a string value is evaluated as null or is yet undefined, the strings value will be "" (empty string).

Since a string is itself an array, a string's length, value at an index, and substring may be referenced in the same manner as an array. The one exception to this rule exists when you reference the length of a single-character string. If a string's length is 1, the ascii decimal value of the character is returned.

6.2 Arrays of Strings

An array of strings is effectively a two-dimensional array since a string is an array itself. However, an array may not be defined as having more than one dimension. Arrays are only defined for string values, numbers are not permitted. Index order must be sequential and starts at 0.

Since array variables may continuously be re-assigned it is possible (though computationally costly) to construct an empty array, create a new array that contains one more element, and assign the original variable containing the array, the value of the new array. This could then be repeated iteratively using the array concatenation operator.

6.2.1 Array Definition

- X = []
  - Empty array of length 0.
- M = [X,Y,Z]
  - Create an array where M[0]=X, M[1]=Y, & M[2]=Z. If more than one array value is specified in this manner, all values MUST be string values.
- R = [x]
  - If x is an integer this will yield an array of size x; if x is a string this will yield a single-celled array containing the string value of x.
- M[0] = “mystring”
  - Explicitly define the value at an index.
- M[0] = X[1]
  - Set the value at an index equal to the value of another array’s cell. The referenced cell appears as a string value.
- M=X
  - Set one array to the value of another array. Overwrite existing array.

6.2.2 Array Reference

- X[indexVal]
  - Returns cell of index value, previously defined as a variable. If referenced cell has no value, return “”.
- X[0:2]
  - Represents the sub-array containing the first 3 cells.
- X
  - The name of the array returns entire array.
- |X|
  - Return the array length.
- Reference to non-existent cell (outside array bounds) also yields “”.

6.3 Integer

Integers [0-9] are used in the context of array manipulation. Mathematical operations are very simplistic, limited to addition and subtraction operations. Boolean true and false are represented by the integer values 1 and 0 respectively.

To obtain the string equivalent of an integer the integer must be surrounded by quotes (“9”).

Since a negative array index or length may not be derived or used, a negative integer value serves no true purpose that coincides with the purpose of scam. As a result, there is no negative (-) unary operator defined for integers.

6.4 Explicit types

Any variable set to the value of a type listed previously may be re-assigned to another type without conflict as long as references to that variable are appropriate to the new type. In some circumstances we would like to avoid the potential conflict while referencing a variable by explicitly setting a variable's type. Note that this qualifier may also be used in function parameters to enforce types for input. The following qualifiers are provided immediately prior to a variable's identifier.

6.4.1 int <var_name>

Integer type.

6.4.2 str <var_name>

String type.

6.4.3 str[]<var_name>

String array type.

7 Expressions

7.1 Generic expression conventions

Notation described here may be used on all types.

7.1.1 (expression)

Parentheses are used to group expressions. Nested parentheses are allowed. Strings and arrays may only use parenthesis with equality operators.

7.1.2 Equality operators

Equality operators group left to right. Equality operators are valid for any type (string, array, or integer) and are used to compare value. Arrays must have the same size and all values in corresponding indices must be equal to be considered equal. Equality operators yield 0 if the specified relation is false and 1 if it is true.

7.1.2.1 expression == expression

7.1.2.2 expression != expression

7.2 Concatenation

The '+' symbol has an entirely different meaning in the context of strings and arrays than it does in the context of integer algebraic operations.

7.2.1 string + string (String Concatenation)

Two or more strings or variables with string values may be appended together. When the resulting string is formed, the input strings are joined from left to right.

Ex: myStr = “Hello”+ “ ” + “World!”
@=myStr
→ # Hello World!

7.2.2 array + array (Array Concatenation)

Two or more arrays or variables with array values may be appended together. When the resulting array is formed, arrays are added from left to right with the leftmost array’s contents starting at the 0 index, effectively following the same pattern as string concatenation.

7.3 Integer Operators

The following operators may only be used when all operands are integers. Note that many of these operators consume or return Boolean values and that in scam Boolean values are represented by the integers 1 and 0.

7.3.1 Unary Operators

Expressions with unary operators group right to left. Unary operators yield 0 if the specified relation is false and 1 if it is true.

7.3.1.1 !expression

Exclamation point denotes the NOT operator. True if the expression is false (0).

7.3.2 Relational operators

The relational operators group left to right. Relational operators yield 0 if the specified relation is false and 1 if it is true.

7.3.2.1 expression < expression

7.3.2.2 expression > expression

7.3.2.3 expression <= expression

7.3.2.4 expression >= expression

7.3.3 expression || expression

A double pipe symbol denotes the OR operator. True if either expression is true (1). If an integer value is anything other than 0 it will evaluate to true.

7.3.4 expression && expression

A double ampersand denotes the AND operator. True if both expressions are true (1). If an integer value is anything other than 0 it will evaluate to true.

7.3.5 Integer Algebra

The following functions have specific behavior defined for integers that differs from the behavior defined for strings and arrays. Scam follows standard conventions for integer algebraic evaluation with operators grouped left to right.

7.3.5.1 expression + expression

7.3.5.2 expression – expression

8 Input/Output

8.1 Output

8.1.1 Output X to console (concatenates all string values)

- @= X

8.1.2 Output to file

- @filepath = "text to print to file"
- @"C:/my folder/textfile" = "text to print to file"

8.2 Input

8.2.1 From console

- X = @

8.2.2 From file

- X = @filepath
- X = @"./my folder/filename.txt"

8.2.3 From argument in MAIN

- X = @1

9 Control Statements

Pre-defined labels using the same notation as a standard function. The evaluated expression will accept 0 as false (unless ! is used). All other values are true.

9.1 IF(X)

```
IF (X)
    Do a bunch of stuff
```

9.2 WHILE(X)

```
WHILE(X)
    Do a bunch of stuff
```

9.3 FOREACH(cellX in myArray)

May only be used with arrays or strings as the indexed object

```
FOREACH(cellX in myArray)
    Do a bunch of stuff with cellX
```