STePL (Simple Text Processing Language)

Nandan Naik (nan2118)

1. Introduction

STePL is a simple language for text processing that allows retrieval of text from files, regular expression matching with support for multiple match regions and persistence of processed text to files. STePL is inspired from AWK. It is offers a subset of the capabilities offered by AWK via a syntax that is similar to the Ruby Language.

2. Token Types and Interpretation

The following types of tokens can be present in a STePL program:

- Identifiers
- Constants (Integer and String)
- Expression Operators
- Keywords

Tokens are interpreted in a “greedy” manner. That is, a valid token is the longest string of characters in the input stream that could be grouped to form the token.

3. Whitespace

In general spaces, tabs and newlines are ignored. A space must be used to separate adjacent Identifiers and Constants.

4. Comments

Comments are indicated by a string of characters that follow a “#” character. Comments cannot be nested. Any characters found in a line after the initial “#” character will be ignored. A comment must occupy a line by itself and cannot be preceded by any tokens.

5. Expressions

An expression is defined to be of one of the following forms:

- L-Value
- L-Value Operator L-Value
- L-Value Operator StringConstant
- L-Value Operator IntegerConstant
- Expression and Expression
- Expression or Expression

The terms L-Value, Operator, StringConstant and IntegerConstant are defined below.

The last two rules above apply to the following cases:

- Where multiple expressions are combined with an “and” indicating that all of the constituent expressions must yield a value of 0 for the final result to yield a 0.
- Where multiple expressions are combined with an “or”, indicating that any one of the constituent expressions must yield a value of 0 for the final result to yield a 0.

6. Identifiers

Identifiers can consist of alphabets only. They can consist of any alphabet in the ASCII character set. Identifiers are case sensitive. Therefore an identifier called “MyVariable” is different from one called “myVariable”. The length of a valid Identifier must be greater than or equal to one.

7. Object Types, Objects and L-Values

There are four types of objects in STePL:

- Integers
- Integer Arrays
- Strings
- String Arrays

An Object is a location in memory capable of storing values. It can be of one of the four types listed above. An Identifier is used to denote the name of the object. Objects cannot be initialized at definition. All integer objects and individual elements of integer arrays are set to 0 by default. All string objects and individual elements of string arrays are set to the empty string “” by default.

Elements of the String and Integer Arrays are accessed via an index starting at 0 until (ArraySize – 1).

L-Values in STePL refer to one of two cases:

- For Integer and String Objects, they are the same as the Identifiers that are used to name the objects.
- For Integer and String Array Objects, they are the Identifier used to name the Array plus an index which is an integer constant provided in square brackets.
  Ex. MyIntegerArray[5]
  Ex. MyStringArray[5]

Object can only be defined outside of functions (objects have global scope). They can, however, only be assigned inside functions.

8. String Constants

String constants must consist of characters in the ASCII character set. They must be delimited by double-quotes. Ex. “12345”.

A double-quote at the start of a String constant immediately followed by a double quote to end it signifies an empty string (“”).

Any double-quote characters that are part of a string constant must be escaped using a preceding double-quote.

Ex. “The actor said, “”Hello””.” Here the word Hello is prefixed and suffixed by two double quotes.

The following escape sequences are used to denote whitespace: “\n” for newline, “\t” for a tab, “\r” for a carriage return.

STePL does not have the need for a “NULL” value and is thus not supported in the language.

9. Integer Constants

Integer constants can only consist of the following characters: 0,1,2,3,4,5,6,7,8,9. Negative numbers are not supported in STePL. Only decimal integers are supported.

STePL does not have the need for a “NULL” value and is thus not supported in the language.

10. Keywords

Keywords are case sensitive. The following keywords are available in STePL:

Keyword	Description
appendFile	Appends the specified string to the end of the file pointed to by FilePath. Returns 0 on success and a 1 on failure. **Usage:** intReturnValue = appendFile FilePath AppendString **Conditions:** FilePath must be a string constant or a string l-value. AppendString must be a string constant or a string l-value. intReturnValue must be an integer l-value.
callFunction	Makes the flow of control jump to the function name specified by FunctionName. The flow of control returns to the line after "callFunction" when the function returns. **Usage:** callFunction FunctionName **Conditions:** FunctionName must be an identifier. The function specified by FunctionName must be defined before being called.

else	Used in an if-then-else code block. If the condition for the if statement evaluates to a non-zero value, the statements associated with else are executed.
	
	Usage:
	if SomeInt == 1 then
	# Some STePL Statement(s)
	else
	# Some STePL Statement(s)
	endIf

endFunction	Used to denote the end of a function. **Usage:** function FunctionName # Some STePL Statement(s) endFunction **Conditions:** FunctionName must be an identifier.

endIf	Used in an if-then or an if-then-else code block to denote the end of the block.
	
	Usage:
	if SomeInt == 1 **then**
	  # Some STePL Statement(s)
	else
	  # Some STePL Statement(s)
	endIf

endWhile	Used to denote the end of a while-endWhile block.
	
	Usage:
	while counter < 0 **then**
	  # Some STePL Statement(s)
	endWhile

function	Used to denote the beginning of a named code block. Control can be transferred to this named block of code through **callFunction**. Functions in STePL do not take any parameters and do not have a return value. All variables have global scope and can thus be accessed inside functions. **Usage:** **function** FunctionName # Some STePL Statement(s) **endFunction** **Conditions:** FunctionName must be an identifier.
getLine	Returns a line of text from the file pointed to by the FilePath at the line number specified by LineNumber. Returns an empty string if there is not content available at the specified LineNumber or if the content cannot be read for any other reason (such as the file not being present). **Usage:** StringVariable = getLine FilePath LineNumber **Conditions:** FilePath must be a string constant or a string l-value. LineNumber must be an integer or an integer l-value. StringVariable must be a string l-value.
int	Used to indicate that an object can store an integer. **Usage:** **int** MyInteger **Conditions:** MyInteger must be an identifier. Objects cannot be assigned when they are declared. They can only be assigned inside functions.

	By default all integer objects are set to 0.
intArray	Used to indicate that an object can store an array of integers. **Usage:** **intArray** MyIntArray ArraySize **Conditions:** MyIntArray must be an identifier. ArraySize must be an integer constant. **Usage:** MyIntArray [index] = 12345 **Conditions:** index must be an integer constant. Objects cannot be assigned when they are declared. They can only be assigned inside functions. By default all elements of integer array objects are set to 0.

if	Used to denote the start of an if-then-else code block. Must be followed by an expression that yields an integer value.
	
	Usage:
	if SomeInt == 1 **then**
	# Some STePL Statement(s)
	else
	# Some STePL Statement(s)
	endIf

main	Used to denote the block of code at which the flow of control is started in a STePL program. **Usage:** **function main** # Some STePL Statement(s) **endFunction** **Conditions:** The main function must always be present in a STePL program.
regexMatch	Returns a stringArray which matches the regular expressions in MatchRegExArray. The returned stringArray must have the same size as the MatchRegExArray. An element in the MatchRegExArray corresponds to an element at the same index in the returned stringArray, if a match was found. If no match was found, the element in the returned stringArray is an empty string “”. **Usage:** MyStringArray = **regexMatch** Content MatchRegExArray **Conditions:**

	Content must be an l-value or an string constant. MatchRegExArray must be an identifier. MyStringArray must be an indentifier.
regexReplace	Returns a string after replacing any occurrence of the regular expressions in MatchRegExArray with corresponding strings in the ReplaceStringArray. The MatchRegExArray must have the same size as the ReplaceStringArray. For every element in the MatchRegExArray, an element at the same index in theReplaceStringArray is used as the text for replacement, if a match was found. If no match was found, the element in the ReplaceStringArray is ignored. **Usage:** MyString = regexReplace Content MatchRegExArray ReplaceStringArray **Conditions:** Content must be an l-value or an string constant. MatchRegExArray must be an identifier. ReplaceStringArray must be an indentifier. MyString must be an identifier.
string	Used to indicate that an object can store a string. **Usage:** **string** MyString **Conditions:** MyString must be an identifier. Objects cannot be assigned when they are declared. They can only be assigned inside functions. By default all elements of string objects are set to “”.
stringArray	Used to indicate an object that can store an array of strings. **Usage:** **stringArray** MyStringArray ArraySize **Conditions:** MyStringArray must be an identifier. ArraySize must be an integer constant. **Usage:** MyStringArray [index] = “Value” **Conditions:** index must be an integer constant. Objects cannot be assigned when they are declared. They can only be assigned inside functions. By default all elements of string array objects are set to “”.
strReplace	Returns a string after replacing any occurrence of the strings in MatchStringArray with corresponding strings in the ReplaceStringArray. The MatchStringArray must

	have the same size as the ReplaceStringArray. For every element in the MatchStringArray , an element at the same index in theReplaceStringArray is used as the text for replacement, if a match was found. If no match was found, the element in the ReplaceStringArray is ignored. **Usage:** MyString = strReplace Content MatchStringArray ReplaceStringArray **Conditions:** Content must be an l-value or an string constant. MatchStringArray must be an identifier. ReplaceStringArray must be an indentifier. MyString must be an identifier.

while	Used to denote the start of a while-then-endWhile block. Must be followed by an expression that yields an integer value.
	
	Usage:
	while counter < 0 **then**
	# Some STePL Statement(s)
	endWhile

then	Used to denote the end of the condition section of a an if-then, if-then-else or a while-then code block. Denotes the start of the section of the if or while block which runsif the condition evaluates to a zero value.
	
	Usage:
	while counter < 0 **then**
	# Some STePL Statement(s)
	endWhile

11. Operators

All operators in STePL associate from left to right.

Additive Operators

Operator	Usage	Description
+	Expression1 + Expression2	Adds two expressions that both yield integer values. Yields an integer value.
-	Expression1 – Expression2	Subtracts Expression 2 from Expression1 where both expressions yield integer values. Yields an integer value.
&	Expression1 & Expression2	Concatenates two expressions that both yield string values. Yields a string value.

Unary Operators

There are no unary operators in STePL. STePL does not explicitly allow the initialization of an integer object to a negative number.

Other Operators

Operator	Usage	Description
[and]	[integerConstant]	Indicates which element in an Array we are operating on.

Relational Operators

Operator	Usage	Description
==	Expression1 == Expression2	Checks if two expressions that both yield integer values are equal. Yields 0 when they are equal and 1 when they are not.
!=	Expression1 != Expression2	Checks if two expressions that both yield integer values are not equal. Yields 0 when they are not equal and 1 when they are.
eq	Expression1 eq Expression2	Checks if two expressions that both yield string values are equal. Yields 0 when they are equal and 1 when they are not.
neq	Expression1 eq Expression2	Checks if two expressions that both yield string values are not equal. Yields 0 when they are not equal and 1 when they are.
<	Expression1 < Expression2	Yields 0 when Expression1 is less than Expression2 and 1 when it is not so. Both Expression1 and Expression2 must yield integer values.
<=	Expression1 <= Expression2	Yields 0 when Expression1 is less than or equal to Expression2 and 1 when it is not so. Both Expression1 and Expression2 must yield integer values.
>	Expression1 > Expression2	Yields 0 when Expression1 is greater than Expression2 and 1 when it is not so. Both Expression1 and Expression2 must yield integer values.
>=	Expression1 >= Expression2	Yields 0 when Expression1 is greater than or equal to Expression2 and 1 when it is not so. Both Expression1 and Expression2 must yield integer values.
and	Expression1 and Expression2	Yields 0 when both Expression1 and Expression2 are equal to 0. Both Expression1 and Expression2 must yield integer values.
or	Expression1 or Expression2	Yields 0 when either Expression1 or Expression2 is equal to 0. Both Expression1 and Expression2 must yield integer values.

Assignment Operators

Operator	Usage	Description
=	L-Value = Expression	Assigns the value of Expression to L-Value. The final value of Expression must match the Object Type of the L-Value. **Conditions:** An assignment to an L-Value can only occur inside of functions. Variables cannot be simultaneously declared and assigned.

Multiplicative Operators

There are no multiplicative operators in STePL.

12. Storage Classes and Scope

All variables in STePL belong to the global storage class. This entails that they are initiliazed when a STePL program is started and accessible across every function in the program.

There is no support for STePL programs to be split into multiple files. Thus, any variables in a STePL program are by default available in any function in that program.

13. Example STePL Program

```
# Many Html Pages have a meta tag in the head section to communicate to search engines what keywords are most relevant to the contents of the page. These keywords can be useful to categorize an HTML page, or categorize a web-site in general (based on the most frequent keywords in all its HTML pages). This program written in STePL fetches an HTML page from disk, reads the keywords in the meta tag for the page, and writes it to a file on disk.

# All variables are global and must be declared outside functions.
string HtmlFile
string OutputFile

string HtmlLine

string MetaKeywords

stringArray MatchReqExStringArray 1
stringArray StringArray 1

stringArray MatchStringArray 1
stringArray ReplaceStringArray 1

int counter

# This is a user defined function to get the keywords from a meta tag in the head section.
# All functions must be defined before their use. Functions do not have any input or output parameters since all variables have global scope.
function getMetaKeywords
        MatchReqExStringArray[0] = "<meta name=""keywords"" content=""[.]*"" />"
        StringArray = regexMatch HtmlLine MatchReqExStringArray

        MatchReqExStringArray[0] = "content=""[a-zA-Z,0-9]*"""
        StringArray = regexMatch StringArray[0] MatchReqExStringArray

        MatchStringArray[0] = "content="""
        ReplaceStringArray[0] = ""
        StringArray = strReplace StringArray[0] MatchStringArray ReplaceStringArray

        MatchStringArray[0] = """"
        MetaKeywords = strReplace StringArray[0] MatchStringArray ReplaceStringArray
endFunction
```

```
# The main function must always be present. It is the entry point for a STePL program.
function main
	HtmlFile = “c:\HtmlFile.html”
	OutputFile = “c:\keywords.txt”

	counter = 1
	HtmlLine = getLine HtmlFile counter
	while  HtmlLine neq “”  then
		callFunction getMetaKeywords
		appendFile OutputFile  MetaKeywords

		counter = counter + 1
		HtmlLine = getLine HtmlFile counter
	endWhile
endFunction
```