Project 2

Due November 3, 2020 at 9:00 PM

Overview

This project specification is subject to change at any time for clarification. For this project you will be writing a Java program to parse language X. Language X is similar to the C language, but is simpler and slightly different. For this program you will be reading in an X language file and converting it to a syntax highlighted XHTML version. A CSV file will describe the font and color settings for the XHTML output file. The CSV file will have a .csv extension and the X language file will have a .x extension. You will be creating several classes that will solve the various portions of the problem, and reusing the classes developed in Project 1.

X Programming Language

The following EBNF describes the X language. The Non-terminals on the right are identified by italics. Literal values are specified in bold. The operators not in bold or italics describe the options of the EBNF. These operators are {} for repetition, [] for optional, () for grouping, and | for or.

EBNF Rules:

```
Program := {Declaration} MainDeclaration {FunctionDefinition}
Declaration := DeclarationType (VariableDeclaration | FunctionDeclaration)
MainDeclaration := void main ( ) Block
FunctionDefinition := DeclarationType ParameterBlock Block
DeclarationType := DataType Identifier
VariableDeclaration := [= Constant] ;
FunctionDeclaration := ParameterBlock ;
Block := { {Declaration} {Statement} {FunctionDefinition} }
ParameterBlock := ( [Parameter {, Parameter}] )
DataType := IntegerType | FloatType
Constant := IntConstant | FloatConstant
Statement := Assignment | WhileLoop | IfStatement | ReturnStatement |
   (Expression ;)
Parameter := DataType Identifier
IntegerType := [unsigned] ( char | short | int | long )
FloatType := float | double
Assignment := Identifier = {Identifier =} Expression ;
WhileLoop := while ( Expression ) Block
IfStatement := if ( Expression ) Block
ReturnStatement := return Expression ;
Expression := SimpleExpression [ RelationOperator SimpleExpression ]
SimpleExpression := Term { AddOperator Term }
Term := Factor { MultOperator Factor }
Factor := ( ( Expression ) ) | Constant | (Identifier [ ( [ Expression {,
   Expression}] ) ] )
RelationOperator := ( == ) | < | > | ( <= ) | ( >= ) | ( != )
AddOperator := + | -
MultOperator := * | /
```

Token Rules:

```
Identifier := ( _ | Alpha ) { ( _ | Digit | Alpha )  }
IntConstant := [ - ] Digit { Digit }
FloatConstant := [ - ] Digit { Digit } [ . Digit { Digit } ]
Digit := 0 – 9
Alpha := A – Z | a - z
```

The X programming language allows for nested function declarations. Variables and functions must be declared before they are referenced or called. Identifiers have the same scoping rules as C. An identifier may only be declared once per block but may be declared more than once per file.

Output XHTML

The output XHTML will output the X language file with fonts and colors specified by the CSV file. All token types that are not specified in the CSV file will be assumed to be the same as the browser defaults. The token classes are IntConstants, FloatConstants, Keywords, Operators, Variables, and Functions. The keywords of language X are: **`unsigned`**, **`char`**, **`short`**, **`int`**, **`long`**, **`float`**, **`double`**, **`while`**, **`if`**, **`return`, `void`**, and **`main`**. The operators of language X are: **`(`**, **`,`** , **`)`** , **`{`**, **`}`** , **`=`**, **`==`**, **`<`**, **`>`**, **`<=`**, **`>=`**, **`!=`**, **`+`**, **`-`**, **`*`**, **`/`,** and `;`. All Variable and Function references must link back to the declaration of the Variable or Function. Each nested block should be indented another level, the amount of indention will 4 spaces.

Input CSV File

The input CSV description file describes the formatting to be used for the XHTML file. The default will be specified by the DEFAULT ELEMENT_TYPE. The attribute types are: BACKGROUND, FOREGROUND, STYLE, and INDENT. The DEFAULT BACKGROUND, FOREGROUND, and STYLE attributes set the global background color, foreground color and font face for the output XHTML. The indention is 4 spaces for each nested block. The FOREGROUND and STYLE may be specified for the token types: FUNCTION, VARIABLE, FLOAT_CONSTANT, INT_CONSTANT, OPERATOR, and KEYWORD. Below is an example of CSV formatting file.

```
ELEMENT_TYPE,BACKGROUND,FOREGROUND,STYLE,FONT
DEFAULT,navy,yellow,,"Courier New"
FUNCTION,,orange,,
VARIABLE,,yellow,,
FLOAT_CONSTANT,,aqua,b,
INT_CONSTANT,,aqua,b,
OPERATOR,,white,b,
KEYWORD,,white,b,
```

Suggested Approach

1. Write a class that will open and read the .csv file as specified on the command line. You should use your CSVParser class from Project 1. Make sure you can determine the formatting for each element type.
2. Use your Scanner class from Project 1 to write a recursive descent parser class that will parse the .x file by making repeated calls to the `getNextToken()` method. (You may need to use `peekNextToken()` to look ahead in some instances.) This class should implement one method per EBNF rule. It should also be able to validate that the .x file does conform to the grammar specified. If an error is found in the .x file, the line number of the error and the grammar rule where the error was found should be printed with an appropriate message. A working example is available on the CSIF to test against.
3. Using the classes written in steps 1 and 2, write a program that reads in a .x source file and a .csv file and outputs the XHTML formatting. The colors and styles of tokens in the XHTML file should match the specification in the .csv file. In addition, references to variables and functions in the source file should be hyperlinks back to their respective declarations. Do not worry about matching overloaded functions with appropriate call, just link back to the function that would be in current scope as if it were a variable. The XHTML should be a valid HTML that can be loaded into any web browser and viewed. You will probably need to begin your XHTML with:`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`
   You may also need to have attributes in your html element. So it should look like: `<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">`

Examples of valid input files .x and .csv files are in `/home/cjnitta/ecs140a/proj2` on the CSIF. Scripts that run the example classes can be found there too. All scripts take one argument except for `XFormatter.sh` which takes two arguments, the format CSV file and then the X file to format. Your code will be tested on the CSIF and is expected to compile and run on the CSIF.

You **must** submit the source file(s), a Makefile, and README.txt file, in a tgz archive. Do a `make clean` prior to zipping up your files so the size will be smaller. You will want to be in the parent directory of the project directory when creating the tgz archive. You can tar gzip a directory with the command:

```
tar -zcvf archive-name.tgz directory-name
```

You should avoid using existing source code as a primer that is currently available on the Internet. You **must** specify in your readme file any sources of code that you have viewed to help you complete this project. You must also provide the URL any code sources in comments of your source code. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.

Helpful Hints

- The use of `peekNextToken()` will be helpful in some instances where it isn't clear what the next rule should be parsed. The language is designed so that only a single look ahead is necessary.
- Obviously, a hash table of some form should be used for the symbol table, but there are two clear options for handling of the scope blocks. Either a stack (or list) of hash tables can be used for the blocks, or a hash table list values could be used. Using a stack of hash tables makes the cleaning up of the scope easy when the block ends but requires traversing through the outer scope hash tables if the symbol is not found in the inner table. Using a hash table of list values makes finding the correct symbol easy but complicates the scope cleanup when the block ends.
- In order to implement the X formatter there are two general approaches. One is to use an observer pattern https://en.wikipedia.org/wiki/Observer_pattern that observes the parsing of your class developed in step 2. You can create an observer interface that has methods to notify of beginning/ending of parsing and rules, as well as parsing of a token and if there is an error. The formatter would then just need to implement the observer interface. Another option is to subclass the X parser and to call the equivalent super method after any pre work is completed, and any post work can be done after the method returns. There are advantages/disadvantages to both approaches.