ICSI 311 Assignment 6 – Finish the Parser

This assignment is extremely important – (nearly) every assignment after this one uses this one!

If you have bugs or missing features in this, you will need to fix them before you can continue on to new assignments. This is very typical in software development outside of school.

You must submit .java files. Any other file type will be ignored. Especially ".class" files.

You must not zip or otherwise compress your assignment. Brightspace will allow you to submit multiple files.

You must submit every file for every assignment.

You must submit buildable .java files for credit.

Introduction

We will finish the parser by adding the remaining statements plus one important feature – labels.

At the end of this assignment, we should be able to parse and see the results of any BASIC program (for the dialect that we are working with).

Details

A statement could start with a label. The lexer already accepts labels. Modify the statement() method in the parser to look for a label before calling any of the specific statement methods. Create a "LabeledStatementNode" (with all of the usual rules, derived from StatementNode) that includes a member (Java string) of the label and a reference to StatementNode (which will hold the single statement that is labeled).

Add statements for GOSUB and RETURN. GOSUB requires a single IDENTIFIER, RETURN is alone on a line.

Add statements for FOR and NEXT. Note that the STEP and increment on FOR is optional. Without a step, the increment should be set to 1.

Addd a statement for END. It takes no parameters.

Add statement for IF. Note that you will need to create a Boolean expression parser method. We are going to keep this super simple – a Boolean expression must follow the pattern: Expression operator Expression

Operator is >, >=, <, <=, <>, =

Add a statement for WHILE. It will hold a boolean expression and the end label.

Finally, we need to deal with functions. Add the function names to the lexer. Functions are an additional option for EXPRESSION. Remember that we had EXPRESSION as

EXPRESSION = TERM { (plus or minus) TERM}

It now should be:

EXPRESSION = TERM { (plus or minus) TERM} or functionInvocation

Make a new parser method for functionInvocation that looks for the appropriate keywords and returns a FunctionNode or NULL. A function follows this pattern:

FUNCTIONNAME LPAREN parameterList RPAREN

A parameterList is empty OR a list (1 or more) of EXPRESSION or STRING

Rubric	Poor	OK	Good	Great
Code Style	Few comments, bad names (0)	Some good naming, some necessary comments (3)	Mostly good naming, most necessary comments (6)	Good naming, non-trivial methods well commented, static only when necessary, private members (10)
Unit Tests	Don't exist (0)	At least one (3)	Missing tests (6)	All functionality tested (10)
Create the AST classes	None (0)		All classes present, some methods missing (5)	All classes and methods (10)
Handle Labels	None (0)			Correct (5)
Handle END	None (0)			Correct (5)
Handle For/Next	None (0)		Significantly Attempted (5)	Correct (10)
Handle Gosub	None (0)		Significantly Attempted (5)	Correct (10)
Handle Boolean expressions	None (0)		Significantly Attempted (5)	Correct (10)
Handle If	None (0)		Significantly Attempted (5)	Correct (10)
Handle Functions	None (0)		Significantly Attempted (5)	Correct (10)
Handle While	None (0)		Significantly Attempted (5)	Correct (10)