# ICSI 311 Assignment 5 – More statements

**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. Blackboard will allow you to submit multiple files.**

***You must submit buildable .java files for credit.***

In this assignment, we will fill in the rest of the core functionality in the lexer and parser to give our language a “real” language feeling – calling functions, loops, and conditionals. We will also tweak our expression handling to allow us to **use** variables and we will also add modulo.

## Lexer changes

We need to add several new keywords to our lexer. This is, remember, as easy as making new tokens and adding them to our hash map. You will need:

if, then, else, elsif, for, from, to, while, repeat, until, mod

We also need the boolean comparators (tokens and lexer changes):

>, <, >=, <=, =, <> (not equal)

## AST Nodes

We need to make ASTNodes for:

BooleanExpression (has a left expression and a right expression and a condition)

While (booleanExpression and collection of statementNodes)

Repeat (booleanExpression and collection of statementNodes)

For (variableReference, start ASTNode, end astNode, collection of statementNodes)

If is a little bit interesting. How do we model that unlimited length chain of “elsif”?

A linked list: If (booleanExpression, collection of statementNodes, ifNode)

For else, you can either use a null boolean expression **or** make an ElseNode that derives from IfNode. One is more work now, one is more work later.

Add modulo to MathOpNode.

## Parser Changes

Our simple boolean expressions are left-to-right, no precedence, so we can implement them without the expression-term-factor pattern. They also don’t chain (1<b<2 style) so we don’t need any recursion. Implement a booleanExpression method in the parser that generates a booleanExpressionNode.

Previously, the only statement type we had was assignments. **Now, we will add the loops and conditionals (if).** The key thing to notice is that every statement is one of these. Make new functions for while, for and if, following the pattern that we have seen:

1. Look for keywords to discern if this function is applicable
2. If not, make sure that we haven’t taken any tokens and return null.

Add modulo to term, much the same as multiplication or division.

Right now, we can’t make an expression that references a variable. For example:

c:=b+1

We can add this by making factor accept an identifier (and creating a variableReferenceNode).

## Testing

We can now test with real programs, so long as they don’t call functions (hint – that’s next). Use some of the test programs we wrote in our first assignment but remove the function calls. Make sure that they parse correctly.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Rubric | Poor | OK | Good | Great |
| Comments | None/Excessive (0) | “What” not “Why”, few (5) | Some “what” comments or missing some (7) | Anything not obvious has reasoning (10) |
| Variable/Function naming | Single letters everywhere (0) | Lots of abbreviations (5) | Full words most of the time (8) | Full words, descriptive (10) |
| ASTNodes – BooleanExpression | None (0) |  | Class present, some methods/members missing (3) | All methods/members (5) |
| ASTNodes – While | None (0) |  | Class present, some methods/members missing (3) | All methods/members (5) |
| ASTNodes – For | None (0) |  | Class present, some methods/members missing (3) | All methods/members (5) |
| ASTNodes – Repeat | None (0) |  | Class present, some methods/members missing (3) | All methods/members (5) |
| ASTNodes – If | None (0) |  | Class present, some methods/members missing (3) | All methods/members (5) |
| Add Modulo | None (0) |  | Token created (3) | Token created and added to AST (5) |
| Parser – Boolean Expressions | None (0) | Attempted (3) |  | Boolean expressions added to AST (5) |
| Parser – While | None (0) | Attempted (5) |  | While AST nodes created (10) |
| Parser – If | None (0) | Attempted (5) |  | If AST nodes created and chained correctly (10) |
| Parser – For | None (0) | Attempted (5) |  | For AST nodes created (10) |
| Parser - Repeat | None (0) | Attempted (5) |  | Repeat AST nodes created (10) |
| Parser – Variable References | None (0) | Attempted (3) |  | Variables can be used as sources for assignment (5) |