# ICSI 311 Assignment 8 – Adding other 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.***

## Background

It might seem hard to believe, but we are nearly done! Calling function is the hardest part.

We need to handle assignment statements. Since we already have the code in place to do calculations (which really handles the right side of an assignment statement), all we have to do is call that code and take the result and populate the variable (which is, remember, in a hash map).

We need to deal with the boolean operators (<, >, etc). Once we can interpret those and get a Java boolean back, we can implement if, while and repeat.

That leaves us with the “for” statement, which is also fairly simple.

## Work on the interpreter

Start with adding assignment statements. In “InterpretBlock” we need to look at the type of AST node that we are running. We previously only worked on FunctionCallNodes. Now we will extend that to AssignmentNode. Remember that an assignment has an expression and a target. The target is a variable and is, therefore, in our hash map of variables. Look it up and find its data type (int or float). That will tell you how to resolve the expression. Call the Resolve function for the “expression” side; take the result and put it into the variable’s Value.

Next we need to deal with boolean expressions. These are simpler than math operations since there are no “chained” operators or differences in precedence. There is a subtle “gotcha” though – we don’t know the data type of either side of the expression. We know that we have

Expression boolOp Expression

But Expression could be int or float. There are a few ways to resolve this; be creative here. Create “EvaluateBooleanExpression” which returns a boolean. You will need to pass in the BooleanExpressionNode and the current variables.

Once you have this working, you can easily implement while, repeat and if. Remember, though, that if has a linked list chain, so you will need to follow that chain to find the first true condition (or else).

For support is, similarly not too hard. Evaluate the “from” and “to” nodes to get the range. Create a loop in your interpreter that sets the index variable to the values in the range of from ->to and call InterpretBlock once for each value. It is reasonable to expect that the FOR block will use integers and to throw an exception if not.

## Testing

This is where we get to have some fun!

You should be able to write any program that you want, so long as it only uses int and float plus the built-in functions. Create a bunch of programs to test your interpreter. Share them with your friends (just the test programs). See if you can break them!

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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) |
| Assignments | None (0) |  | Handles either int or float (7) | Handles both int and float (15) |
| BooleanExpression | None (0) | Handles only some boolean operations (5) | Handles all boolean operations, but only for one data type (10) | Handles all boolean operations for int and float (15) |
| If | None (0) |  | Works for a single if condition (10) | Works for chained if blocks, including else (20) |
| While | None (0) |  | Attempted (5) | Working (10) |
| Repeat | None (0) |  | Attempted (5) | Working (10) |
| For | None (0) |  | Attempted (5) | Working (10) |