# 3290 A3

Sol Scobie – c3302821

When I was told the class size was odd and that I would be doing the assignments alone I had hoped that the experience of attempting this course the previous year would make this all easier. While that was true, once again attempting part 3 was an overwhelming workload.

At time of writing Semantic Checking is unable to correctly determine that assignment operations are legal, unable to enforce typing for variables and fails *most* of the time confirming that functions are provided with the correct parameters. The remaining features are functional by my own testing.

Code Generation is wholly non-functional, op-codes for the necessary actions were setup in preparation however due to the workload of this and other courses Code Generation never began meaningful development.

## Changes and Fixes from A2

Beginning A3, a number of changes had to be made to my Symbol Table, correcting a number of minor issues in helper functions such as the various variants of lookup(); and the addition of a variety of helper attributes to the original Token class from a1, such as being able to get the tokenList index directly from the token rather than having to search the list. They symbol table additionally tags symbol occurrences as being definitions or assignments, to assist in the semantic checking process via the added Token attributes.

The original A2 submission included a bug in the grammar the rules var, varTail and varTailTail failed to accept input in the form of structName.var and arrayName[index].var for structs. This was corrected by adding TDOTT to the first table for Tail and TailTail. This bug snuck under the radar due to varTail and varTailTail being epsilon rules and the test files I used primarily throughout the process failed to trip this rule in a way that it couldn’t recover from. Updated test files and a greater variety of test files in A3 exposed this error and it has been corrected.

## Semantic Analysis

CD24 program names must exist at beginning and end and must have the same lexeme. The current semanticChecker handles this fine.

Id’s must be declared before use. The current semanticChecker handles this fine. The structer of “before use” is that a referenced ID must have a definition of it’s symbol either within the same scope e.g. main or functions. If there is no definition of the symbol within it’s own scope then the symbol must have a definition within global. Additionally that definition must exist before (taking line number and column number) any given occurrence of that symbol. If a symbol is defined in global and redefined within a function or main, then that redefinition will take priority within that scope, effectively overwriting the original until leaving that scope, meaning main and global are different scopes.

ArraySizing must be declared upon array definition. The current semanticChecker handles this fine. Array Sizing can only be done with constants, or int literals and handles expressions allowing [const+intLit-intLit^const] as an example index. Note that indexes do not work to the same degree in array references due to arrayVar[index].intVar being a legal way to declare an index. The current implementation cannot handle this sort of data in either array indexes or function parameters. Notably structs seem to be functional however have not been tested sufficiently to believe that they’re fully functional.

Strong Typing and Assignment Operations are unimplemented. In an ideal scenario the mess that the SymbolTable file has become would be cleaned up, functions retooled for reusability however as it stands many functions are bespoke solutions for single purposes inflating the length of the file and reproducing the same functionality unnecessarily. Working alone I did not have the time to correct this and it severely hampered my ability to complete many of the semantic checks. This issue alongside the need to expand the number of different operations the symbolTable could undertake after A2 were the key reasons for the difficulty in completing all the functionality.

Functions must contain minimum one return statement of the correct type. The current semanticChecker handles this fine. The symbol table was expanded with to track the return type of functions post A2 and that made the process of ensuring these returns work correctly fairly easy.

Function parameters must match the typing of their formal parameter requirements. Works to a degree. Parameters that are one variableName of non array typing tend to be functional however issues sometimes occur when functions require more than one symbol as a parameter. Array references struggle here due to the recursion present in array Indexes, the system cannot handle an index that is itself an array[index].var reference or a function call. I believe there are bugs present in single int, float, bool variables and structs however tend to be more consistently functional, though fail under conditions that are current undetermined.

The number of parameters in a procedure call must be equal to the number of definition parameters. The semantic checker handles this fine, however, previously it began failing for an unknown reason before correcting itself for an unknown reason. It would not be out of the realm of possibility for this to happen again under testing.

## Conclusion

I am disappointed by my own output in this Assignment, however I would request some leniency in marking as I have struggled with mental health, my adhd and working alone on this assignment, during the second half of this course. I should have requested assistance before now however completing work for other courses and requirements outside of university have led to a lot of time where I was unable to progress here.

More practically working on semantic checking many issues I encountered I had planned out before hand such as saving return types for functions and saving the value / parameter tokens in definitions during the symbol table initialisation. These plans were lost in the process of correcting issues, dealing with the need to fix and resubmit A2 issues, and constructing the helper functions I required but did not anticipate.

Additionally, I should have begun the symbol checker process by focusing in on a context agnostic way of consuming a list of tokens, and determining the return type by referencing against the symbol tale which contains already most of the necessary data. Due to this lack or foresight I spent a lot of time re implementing similar functionality over and over especially while fatigued.