Andre Ibrahim
40132881

SECTION 1: Analysis (itemized list)

Legend

- Completed
- Partially completed
- Not done

1.1 Allocate memory for basic types (integer, float).
1.2 Allocate memory for arrays of basic types.
1.3 Allocate memory for objects.
1.4 Allocate memory for arrays of objects.
2.1 Branch to a function's code block, execute the code block, branch back to the calling function.
2.2 Pass parameters as local values to the function's code block.
2.3 Upon execution of a return statement, pass the return value back to the calling function.
2.4 Call to member functions that can use their object's data members.
3.1 Assignment statement: assignment of the resulting value of an expression to a variable, independently of what is the expression to the right of the assignment operator.
3.2 Conditional statement: implementation of a branching mechanism.
3.3 Loop statement: implementation of a branching mechanism.
3.4 Input/output statement: Moon machine keyboard input/console output
4.1. For arrays of basic types (integer and float), access to an array's elements.
4.2. For arrays of objects, access to an array's element's data members.
4.3. For objects, access to members of basic types.
4.4. For objects, access to members of array or object types.
5.1. Computing the value of an entire complex expression.
5.2. Expression involving an array factor whose indexes are themselves expressions.
5.3. Expression involving an object factor referring to object members.

SECTION 2: Design

There are three components to the design, a IntermiediateVarVisitor, a MemSizeSetter and finally a codeGenVisitor. The IntermediateVarVisitor parses the AST using DFS and adds lit variables and temp variables where it is needed; this needs to happen prior to the code gen because it is used to calculate the memory. The memSizeSetter parses the symboltable and sets the memory size for all types including arrays and object types. Finally the CodeGenVisitor parses the AST to generate the moon machine code to be executed.

SECTION 3: Use of Tools

The attribute grammar needed to be updated to facilitated the code generation so the same Tools used in grammar transformation from the previous assignment are also listed bellow:

1. To get the parsing table used the university of calgary tool

https://smlweb.cpsc.ucalgary.ca/start.html For some context I needed to do that since inorder to make it easier to inject the semantic actions I added them to my grammar as nullable nonTerminals.

2. The ucal tool generates the parsing table as html and we can get that table and put it through this tool https://www.convertcsv.com/html-table-to-csv.htm to convert the tables into csv format.

3. At some point I had an issue with the ucal tool telling me that the url request was too short to parse my grammar so I used this tool to condense it before feeding it to ucal https://www.removelinebreaks.net/

Tools used in code:

1. The only tool used in the code is the Lexer that I had built in assignment 1
Everything else is vanilla Typescript.