1. Checklist

1. Memory allocation

1.1 Allocate memory for basic types (integer, float). 1.0

Done (see \*any\*Test.moon or allocationTest.moon)

1.2 Allocate memory for arrays of basic types. 0.5

Done (see \*any\*Test.moon or allocationTest.moon)

1.3 Allocate memory for objects. 0.5

Done (see \*any\*Test.moon or allocationTest.moon)

1.4 Allocate memory for arrays of objects. 0.5

Done (see \*any\*Test.moon or allocationTest.moon)

2. Functions

2.1 Branch to a function’s code block, execute the code block, branch back to the calling function. 1.0

Can branch to functions and branch back (see functionTest.moon)

2.2 Pass parameters as local values to the function’s code block. 1.0

Can pass simple params to functions (see functionTest.moon)

2.3 Upon execution of a return statement, pass the return value back to the calling function. 1.0

Can return the final value of functions (see functionTest.moon)

2.4 Call to member functions that can use their object’s data members. 1.0

// Can’t do because member functions have not been declared in the past.

3. Statements

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. 1.5

Can do so for simple and complex expressions (see expressionTest.moon)

3.2 Conditional statement: implementation of a branching mechanism. 1.0

Can do so for simple relational expressions (see ifElseTest.moon)

3.3 Loop statement: implementation of a branching mechanism. 1.0

Can do so for simple relational expressions (see loopTest.moon)

3.4 Input/output statement: execution of a keyboard input statement should result in the user being prompted for a value from the keyboard by the Moon program and assign this value to the parameter passed to the input statement. Execution of a console output statement should print to the Moon console the result of evaluating the expression passed as a parameter to the output statement. 2.0

Got write to work to print integers using putint in util.m (see (any)Test.moon).

Read uses getint from util.m (see readTest.moon)

4. Aggregate data members access

4.1. For arrays of basic types (integer and float), access to an array’s elements. 1.0

Can access arrays elements for printing, assigning values(for ints) (see arrayTest.moon)

4.2. For arrays of objects, access to an array’s element’s data members. 1.0

4.3. For objects, access to members of basic types. 1.0

Can access object basic data members (see classTest.moon)

4.4. For objects, access to members of array or object types. 1.0

Can access object array data members (see allocationTest.moon)

5. Expressions

5.1. Computing the value of an entire complex expression. 2.0

ComputeExpression method recursively handles complex expressions(see expressionTest.moon)

5.2. Expression involving an array factor whose indexes are themselves expressions. 1.0

Done in arrayTest (see arrayTest.moon)

5.3. Expression involving an object factor referring to object members 1.0

Can add variables that are class member, not obj to obj yet (see classTest.moon)

Design

Design using code that has been divided into scopes, as I did not implement the visitor pattern in previous assignments. Scopes are divided as; main, functions, classes.

A tag-based system was created. Attempted offsets as you can see in symbolTable calculations however did not have time to fully implement, therefore scraped it…. Offsets are currently only used with array size declarations.

Order of code generations phases:

Testing:

Tools

The course lecture slides.

Written up many (\*)Test.src files to compile to (\*)Test.moon files

Util.m

And some C# libraries for my code:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;