Alexandru Bara 40132235 Assignment 5 Comp 442

Part 1

Green means done,

Red means not done,

Yellow means I tried to do it

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: 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.

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.

Part 2

Phase 1:
The general design consists of 2 visitors. I have a class called semanticAnalyzer and it is called the Parser. It then gets in return the head node of the AST created. After that the head node is used to call the visitors. In order to get the visitors setup, I made the class node to contain all information needed at a node. Then each of the semantic concepts inherit the node and change the "toString" representation to indicate what is being held at that node. This way we have a bunch of different nodes and we can enable the visitor pattern. I have added the memory size visitor and the code generation visitor. Since I implemented the visitor pattern properly from assignment 4, it was very straightforward to add the 2 new visitors.

Phase 2:
The memory size visitor is in charge of associating a size to the different objects we have in the tables. It also adds memory size for classes and functions that will be used for the objects. I decided to not implement offset and do everything dynamically in the code generation portion for arrays. I also add literal values and temp values that will be used later in the code generation.

In the code generation visitor I have all my code that generates the moon code. This is where I have the most code because there are a lot of nodes that can generate code. If statements, loops, functions, classes, declarations and statements all need to be implemented. Each one is different and generates the code needed for that small portion to work. I used depth first search in order to make my code generation except for my if statement where I have code being generated between visits.

Part 3

I used AtoCC in order to validate if my grammar is in LL1 format.
I used Ucalgary smlweb in order to create the parsing table and to get the first and follow sets. I tried to use the Ucalgary tool in order to fix my language, but I find that doing it by hand worked better.
I used google sheets to take the table from Ucalgary and make it in a csv file.
I used Joey Paquet's grammar tool to remove some ambiguities and left recursion.
Finally, I used the built in tools from Java like arraylist, hashmap and stacks in order to store my code.