Documentation Report
Assignment 5: Code Generation

COMP 442-NN

WINTER 2023

Jeremy Piperni
40177789

1. Analysis

1.1 |X| Allocate memory for basic types.
1.2 |X| Allocate memory for arrays of basic types.
1.3 || Allocate memory for objects.
1.4 || Allocate memory for arrays of objects.
2.1 |X| 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 |X| Assignment statement
3.2 |X| Conditional statement
3.3 |X| Loop statement
3.4 |X| Input/Output statement
4.1 |X| 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 |X| 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.

11/20

2. Design

I implemented a tag-based approach to my code generation. I chose this approach because I knew I would be lacking in time to do every code generation section and a tag-based approach is simpler for the easier parts of the assignment. This did come with some drawbacks. For example, while I do have basic function calls working, a tag-based approach limits the use of nested function calls. Arrays of basic types work but only one-dimensional arrays. I implemented the following two visitors to get the code gen working.

MemAllocVisitor

This visitor was responsible for allocating memory to basic types, and arrays of basic types. The allocation of this memory was done by adding entries to the symbol tables that were previously created in the last assignment. Once memory allocation is done, functions add up the size of all their memory members to find out how much memory that function would take up.

CodeGenerationVisitor

This visitor was responsible for the code generation of all the sections listed in 1. Analysis. As previously stated a tags-based approach was used

3. Use of Tools

Tools used:

1. Lecture slides provided by Joey Paquet for code generation instructions.
2. Moon Processor documentation by Peter Grogono.