Concordia University
Department of Computer Science
and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2020

Assignment 4, Code Generation

Deadline:	Wednesday April 14th, 2020
Evaluation:	10% of final grade
Late submission:	not accepted

In this assignment, you are to implement a code generation phase for the language as described in assignment 2 and 3. The following section lists the different specific constructs/concepts for which code generation needs to be implemented for all the aspects of the language to become executable.

Problem statement

1. Memory allocation: As variables are declared, memory space must be allocated that will eventually be used to store their values. Memory cells are either identified using a unique Moon assembly code label, or an offset relative to the stack frame on which the variable will reside during execution of the function scope in which it is declared. The size of the allocated memory block should depend on the declared type/kind of variable (integer, float, array, object, array of objects, etc.)

2. Functions: The code generated for functions should be associated with a Moon assembly code subroutine, i.e. a block of code that can be jumped to using a label, then when the function resolves, jump back to the instruction following the initial jump. Parameters should be passed/received during the function call. The return value should be passed to the calling function when the function resolves. If a call to a member function is made, the member function should have access to the data members of the object from which it was called.

3. Statements: Every kind of statement as defined in the grammar should be implemented: assignment statement, conditional statement, loop statement, input/output statements, and return statement.

4. Aggregate data members access: Aggregate data types such as arrays and objects contain a group of data values. The code must be generated so that contained member values in such an aggregated value can be accessed when referred to as factors in an expression or the left-hand side of an assignment statement. This includes access to array elements and object members.

5. Expressions: Computing of the resulting value of an entire expression, including the simple case when an expression is either a variable name or a single literal value, up to complex expressions involving all kinds of operators (arithmetic, logical, and relational), array indexed with expressions, deeply nested object member access, etc. This involves memory allocation for temporary results, and register allocation/deallocation when necessary.

Work to submit

Document

You must provide a short document that includes the following sections:

Section 1. **Analysis** : Using the (1-5) itemized list provided below (see "Implementation"), provide a checklist that identifies what code generation items are either implemented or not implemented in your assignment.

Section 2. **Design** : I – Overall design – Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution. II – Phases – Description of the purpose of each phase involved in the implementation of this assignment.

Section 3. **Use of tools** : Identify all the tools/libraries/techniques that you have used in your implementation and justify why you have used these particular ones as opposed to others.

Implementation

1.	**Memory allocation**	marks
1.1	Allocate memory for basic types (integer, float).	1.0
1.2	Allocate memory for arrays of basic types.	0.5
1.3	Allocate memory for objects.	0.5
1.4	Allocate memory for arrays of objects.	0.5

2.	**Functions**	marks
2.1	Branch to a function's code block, execute the code block, branch back to the calling function.	1.0
2.2	Pass parameters as local values to the function's code block.	1.0
2.3	Upon execution of a return statement, pass the return value back to the calling function.	1.0
2.4	Call to member functions that can use their object's data members.	1.0

3.	**Statements**	marks
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
3.2	Conditional statement: implementation of a branching mechanism.	1.0
3.3	Loop statement: implementation of a branching mechanism.	1.0
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

4.	**Aggregate data members access**	marks
4.1.	For arrays of basic types (integer and float), access to an array's elements.	1.0
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
4.4.	For objects, access to members of array or object types.	1.0

5.	**Expressions**	marks
5.1.	Computing the value of an entire complex expression.	2.0
5.2.	Expression involving an array factor whose indexes are themselves expressions.	1.0
5.3.	Expression involving an object factor referring to object members.	1.0

Notes : (1) Difficulty rating is provided as green: easy, yellow: medium, red: harder. (2) Marking elements with higher marks indicate constructs that are necessary to have in order to demonstrate that the generated code runs correctly. (3) This is the list of code generation elements due for assignment 4. More code generation elements are going to be added on the grading scheme for the final project requirements.

- **Output of moon code in a file :** The Moon code should be output to a file, including comments in the generated code to highlight the structure of the code generated for each syntactical construct. When compiling a file named, for example, ***`originalfilename.src`***, the code generator should write the Moon code into a file named ***`originalfilename.moon`***.
- **Test cases :** Write a set of source files that enable to test the code generation for all syntactical structures that you can generate code for, including simple cases that test isolated syntactical constructs, and more complex cases that test combinations of syntactical constructs.
- **Driver :** Include a driver that reads your test cases and generates, for each test case file, the corresponding **`.moon`** file.

Assignment submission requirements and procedure

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category "*programming assignment 4*". The file submitted must be a **.zip** file containing:

- all your code
- a set of input files to be used for testing purpose, as well as a printout of the resulting output of the program for each input file (moon code output)
- a simple document containing the information requested above/below

The marking will be done in a short presentation to the marker. A schedule will be provided to you by email in the days before the due date. You will be given a short time for the presentation, so make sure that you are ready to effectively demonstrate all the elements mentioned in the "Work to submit" section above.

Evaluation criteria and grading scheme

Analysis:		
Using the (1-5) itemized list provided above (see "Implementation"), provide a checklist that identifies what code generation items either implemented or not implemented in your assignment. For each item, indicate what file/line number is testing it. (Document Section1)	ind 2.1	2 pts
Design/implementation:		
Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution. (Document Section 2 – I)	ind 4.3	2 pts
Description of the purpose of each phase involved in the implementation of this assignment. (Document Section 2 – II)	ind 4.3	3 pts
Correct implementation according to the above-stated requirements (see "Problem statement" and "Implementation" above).	ind 4.4	20 pts
Output of executable Moon code in a corresponding **`.moon`** file.	ind 4.4	3 pts
Completeness of test cases.	ind 4.4	15 pts
Use of tools:		
Description/justification of tools/libraries/techniques used in the analysis/implementation.	ind 5.2	2 pts
Successful/correct use of tools/libraries/techniques used in the analysis/implementation.	ind 5.1	3 pts
Total		**50 pts**