Concordia University
Department of Computer Science and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2017

Assignment 4, Semantic Verification and Code Generation

Deadline:	Wednesday April 12th, 2017
Evaluation:	10% of final grade
Late submission:	not accepted

This assignment consists of three parts. You have (1) to define (informally) the semantics of the language used in the project, (2) implement a semantic verification phase and (3) a code generation phase.

The syntax of our language states that a program consists of a set of free functions (including the program function), as well as a set of classes (i.e. object-oriented user-defined types). In the function bodies, we can use assignment statements, conditional statements, loops, function calls, etc. There are many details that are not suggested by the syntax, and that we already informally stated, e.g.:

- The execution of the program starts with the program function;
- Each identifier declared in a given scope can only be used inside this scope. In cases where a sub-scope contains a re-declaration of an identifier, the local definition overrides the higher-level one;
- Each identifier used (function, variable or parameter) should be declared exactly once in a given scope;
- Function calls must match the number and types of parameters declared in the function definition;
- Variables declared using a user-defined type (i.e. class) are allowed anywhere a variable declaration is allowed, including as a parameter to a function.

However, there are many semantic details that have not been specified yet, e.g.:

- Do we allow operations involving entire arrays?
- Do we allow in an expression or an assignment statement a mixture of integer and float variables?
- How are parameters passed to procedures (by value, by reference, etc.)?
- Is there a limit on the number of function parameters?
- Is function overloading allowed, i.e. functions with the same name but with varied number/type of parameters in the same scope?
- Are recursive function calls allowed, or can two instances of the same function be active at the same time?
- etc

Work to be done

- Define the semantics of the language. You can use any formalism you find suitable. (The English language being the minimal suitable formalism.) You are responsible for the specification of all unspecified semantic details.
- Implement semantic verification according to these semantic rules. This includes (and is actually preceded by) the programming of an attribute-migration scheme.
- Implement a code generation scheme that generates code for the Moon machine. See the handout for the definition of the Moon machine and its instruction set. Both this handout and the C code for the Moon machine simulator is available on the course web site.

Suggestions for the code generation

- Since no code optimization is required, you can generate the code for the Moon machine without any use of intermediate code or intermediate representation.
- Think about how the registers of the Moon machine will be used, e.g. some might be used as temporary storage during expression evaluation, some for parameter passing, some for return address of procedure calls, as array index registers, etc.
- Proceed incrementally. Start with the code generation for variable declarations, expressions, then assignment statements, loop statements, conditional statements, and then the function calls. Start with simple data types (float and integer) and then generalize to arrays and classes/objects;

Floating point number representation on the Moon machine

The instructions for the Moon machine don't have operations on floating point numbers. We can implement them using a fixed-point representation. In a fixed-point representation of floating point numbers, we consider that the 31 bits that represent a number are divided into two segments. The first segment of say, 22 bits represents the integer part of a number and the remaining 8 bits represent the fractional part of a number. This means that the precision of a floating point number is limited to 8 bits, but on the other hand, we can use the arithmetic operation for integers to get arithmetic operations on floating point numbers. It is suggested that you use this type of representation in your project.

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 (symbol table output and error reporting, as described above).
- a simple document containing the information requested above.

You are also responsible to give proper compilation and execution instructions to the marker in a README file. If the marker cannot compile and execute your programs, you might have to have a meeting for a demonstration.

Evaluation criteria and grading scheme

Analysis:		
Description of the semantic rules used in the implementation	ind 2.1	2 pts
Grammar augmented with the placing of the semantic actions	ind 2.2	3 pts
Design/implementation:		
Description/rationale of the overall structure of the solution and the roles of the individual components used in the applied solution.	ind 4.3	4 pts
Correct implementation of semantic verification according to original assignment statement.	ind 4.4	8 pts
Correct implementation of code generation according to original assignment statement.	ind 4.4	7 pts
Output of clear error messages (error description and location).	ind 4.4	3 pts
Output of executable code in a separate file that can be run by the Moon machine.	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**