Concordia University
Department of Computer Science
and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2022

Assignment 4, Semantic Analysis

Deadline:	Sunday March 20th, 2022
Evaluation:	8% of final grade
Late submission:	penalty of 50% for each late working day

This assignment is about the design and implementation of a semantic analyzer for the language described in assignment 2. The implementation of the semantic analysis phase implies that you traverse the AST generated in assignment #3, and trigger semantic actions related to two inter-related sub-phases that should be triggered separately by at least two separate traversals of the AST:

1. generation of symbol tables
2. semantic checking and type-checking semantic actions

The semantic analyzer should use as input the abstract syntax tree generated by the parser in assignment #3. The semantic analyzer should locate, report, and recover from eventual semantic errors. The operation of the semantic analyzer should produce a symbol table data structure that is to be used by the further processing stages, including scoping and binding checks, type checks, and eventually code generation in assignment #5. It should also write to a file a text representation of the symbol table, as well as potential semantic error reporting in another separate file.

The assignment includes three grading source files. These files should be used as-is and not be altered in any way. Completeness of testing is a major grading topic. You are responsible for providing appropriate test cases that test for a wide variety of valid and invalid cases in addition to what is in the grading source files provided.

Semantic concepts to be implemented/verified

The following is a discussion of various aspects of the semantics of the language, and the tasks required for their implementation and documentation of the solution:

- The purpose of the symbol table structure is to help resolve the scoping, binding and typing of the identifiers used in the program.
- A symbol table contains an entry for all identifiers (*variables, functions, classes)* defined in the scope that it represents. There are scopes for each class definition, free or member function definition, and a global scope for the whole program.
- A program includes a set of free functions, minimally including one and only one main function. Information about the free functions must be available in the symbol table before calls to free functions can be bound and semantically checked. In order to allow functions to be called before they are defined, you need to implement at least two passes: a first pass that constructs the symbol tables, and a second pass that uses the information generated in the first pass.
- Classes represent the encapsulation of a user-defined data type and declarations for its member functions. The member functions are all defined in the global scope, but use a scope-resolution operator to identify them as members of a specific class. As with free functions, two passes (as described above) have to be used to allow a class to refer to another class that is declared after it. As member functions are declared in the class declaration and defined later, two passes are necessary to bind the function declaration's symbol table entry with its corresponding local symbol table.
- If a class has an inheritance list, the symbol table of its directly inherited class(es) should be linked in this class, so that inherited members are effectively considered as members of the class, even though they are part of a separate scope. Inherited members with the same name and kind (i.e. variable or function) as a class member should be shadowed by the class member and a warning should be reported in such cases. In any case, circular class dependencies should be reported as a semantic error.
- It is allowed to have members or variables with the same name in two different classes or functions, as they are not defined in the same scope.

- There is no point in checking for indexes out of bounds, as indexes can be expressions, whose value cannot be determined at compile time.
- The variables declared inside the functions (local variables) or classes (data members) are considered local and thus can only be used in the current function or class scope. Data members can be used in all member functions of their respective class. This raises the need for a nested symbol table structure:
  - The global symbol table, representing all the symbols defined in the global scope, exists until the end of the compilation process.
  - Local symbol tables, representing sub-scopes. Each local symbol table should be bound to their respective elements in the higher-level symbol table.
- A local symbol table is created at the beginning of the processing of any function or class. Therefore, you have to associate with each variable and function identifier a symbol table record that contains its properties, and include it in the symbol table representing the scope in which this identifier is declared.
- When using operators in expressions, attribute migration must be done to determine the type of sub-expressions. For simplicity of code generation later, it is suggested that it should be semantically invalid to have operands of arithmetic operators to be of different types. For the same reason, both operands of an assignment must be of the same type.

Work to submit

Document

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

Section 1. **List of semantic rules implemented** : Using the (1-15) itemized list provided below (see "Implementation"), provide a checklist that identifies what semantic checks 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 semantic checking phase involved in the implementation.

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

- **Symbol table creation phase**
  1. A new table is created at the beginning of the AST traversal for the global scope.
  2. A new entry is created in the global table for each class declared in the program. These entries should contain links to local tables for these classes.
  3. An entry in the appropriate table is created for each variable defined in the program, i.e. a class' data members or a function's local variables.
  4. An entry in the appropriate table is created for each function definition (free functions and member functions). These entries should be links to local tables for these functions.
  5. During symbol table creation, there are some semantic errors that are detected and reported, such as multiply declared identifiers in the same scope, as well warnings such as for shadowed inherited members.
  6. All declared member functions should have a corresponding function definition, and inversely. A member function that is declared but not defined constitutes an "no definition for declared member function" semantic error. If a member function is defined but not declared, it constitutes an "definition provided for undeclared member function" semantic error.
  7. The content of the symbol tables should be output into a file in order to demonstrate their correctness/completeness.
  8. Class and variable identifiers cannot be declared twice in the same scope. In such a case, a "multiply declared class", "multiply declared data member", or "multiply declared local variable" semantic error message is issued.
  9. Function overloading (i.e. two functions with the same name but with different parameter lists) should be allowed and reported as a semantic warning. This applies to member functions and free functions.

- **Semantic checking phase – binding and type checking**
  10. Type checking is applied on expressions (i.e. the type of sub-expressions should be inferred). Type checking should also be done for assignment (the type of the left and right hand side of the assignment operator must be the same) and return statements (the type of the returned value must be the same as the return type of the function, as declared in its function header).
  11. Any identifier referred to must be defined in the scope where it is used (failure should result in the following error messages: "use of undeclared variable", "use of undeclared member function", "use of undeclared free function", "use of undeclared class").
  12. Function calls are made with the right number and type of parameters. Expressions passed as parameters in a function call must be of the same type as declared in the function declaration.

13. Referring to an array variable should be made using the same number of dimensions as declared in the variable declaration. Expressions used as an index must be of integer type. When passing an array as a parameter, the passed array must be of compatible dimensionality compared to the parameter declaration.
14. Circular class dependencies (through data members\inheritance) should be reported as semantic errors.
15. The "." operator should be used only on variables of a class type. If so, its right operand must be a member of that class. If not, a "undeclared data member" or "undeclared member function" semantic error should be issued.

- **Error reporting:** All semantic errors/warnings present in the entire program should be reported properly, mentioning the location in the program source file where the error was located. When parsing a file named, for example, ***originalfilename*** and semantic errors or warnings are found, error or warning messages should be printed out in a file named ***originalfilename*****.outsemanticerrors**. Only existing errors/warnings should be reported. Errors should be reported in the file in synchronized order, even if different semantic checking phases are implemented and errors are found in different phases.
- **Output of symbol tables data structure:** After the symbol table structure has been completely created, the program should output it to a file, allowing to easily verify if the symbol table correctly corresponds to the input program. When parsing a file named, for example, ***originalfilename***, the parser should write into a file named ***originalfilename*****.outsymboltables** a text representation of the symbol table data structure that corresponds to the original program.
- **Test cases :** Write a set of source files that enable to test the semantic analysis involved in the language as described above. Include cases testing for a variety of different errors to demonstrate the completeness and accuracy of your error reporting.
- **Driver:** Include a driver that does the semantic analysis for all your test files. For each test file, the corresponding ***outsymboltables*** files should be generated, in addition to all the outputs described in the previous assignments.

Assignment submission requirements and procedure

- Each submitted assignment should contain four components: (1) the source code, (2) a group of test files, (3) a brief report, and (4) an executable named **`semanticanalyzerdriver`**, that does the semantic analysis from all your test files. For each test file, the corresponding **`outsemanticerrors`**, and **`outsymboltables`** files should be generated.
- The assignment statement provides test files (**`.src`**) and their corresponding output files.
- The source code should be separated into modules using a comprehensible coding style.
- The assignment should be submitted through moodle in a file named: "**`A#_student-id`**" (e.g. **`A1_1234567`**, for Assignment #1, student ID 1234567) and the report must in a PDF format.
- You may use any language you want in the project and assignments but the only fully supported language during the lab is Java.
- You have to submit your assignment before midnight on the due date on moodle.
- The file submitted must be a **.zip** file

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 (1-15) itemized list provided above (see "Implementation"), provide a checklist that identifies what semantic checks are either implemented or not implemented in your assignment (document Section 1).	ind 2.1	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 (document Section 2-I).	ind 4.3	2 pts
Description of the purpose of each semantic checking phase involved in the implementation. For each phase, mapping of semantic actions to AST nodes, along with a description of the effect/role of each semantic action (document Section 2-II).	ind 4.3	2 pts
Correct implementation according to the above-stated requirements.	ind 4.4	20 pts
Output of clear error messages (error description and location) in a ***`.outsemanticerrors`*** file.	ind 4.4	4 pts
Output of symbol tables in a ***`.outsymboltables`*** file.	ind 4.4	4 pts
Completeness of test cases.	ind 4.4	10 pts
Use of tools:		
Description of tools/libraries/techniques used in the analysis/implementation. Description of other tools that might have been used. Justification of why the chosen tools were selected – document Section 3.	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**