Concordia University
Department of Computer Science and Software Engineering

Compiler Design (COMP 442/6421)
Winter 2018

Assignment 3, Semantic Analysis

Deadline:	Monday March 19th, 2018
Evaluation:	10% of final grade
Late submission:	penalty of 50% for each late working day

In this assignment, you are to implement a semantic analysis phase for the language described in assignment 2. The implementation of the semantic analysis phase implies two inter-related sub-phases: (1) implementation of semantic actions for the generation of symbol tables and (2) implementation of semantic checking and type-checking semantic actions. The following is a discussion of various aspects of the semantics of the language, and the tasks required for their implementation:

- The purpose of the symbol table is to help resolve the typing and cross-referencing of the identifiers used in the program, taking into consideration the typing and scoping of the identifiers as expressed in the analyzed program.
- A program includes a set of free functions, minimally including one and only one main function (the **`program`** function). Information about the free functions must be available in the symbol table before function calls can be semantically checked. In order 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 its member functions. They are all defined before the free functions. 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. In any case, circular class dependencies should be reported as a semantic error.
- 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.
- It is allowed to have two members 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:
  - A symbol table contains an entry for all identifiers (*variables, functions, classes)* defined in its own scope. There are scopes for each class definition, free or member function definition, and a global scope for the whole program.
  - The global symbol table, representing all the symbols defined in the global scope, exists until the end of the compilation process.
  - All local symbol tables are representing sub-scopes and should be bound to their respective elements in the current symbol table.
  - A local symbol table is created at the beginning of the compilation 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. You have to keep in mind that you might have to change the structure of these records later in the design of code generation phase of the compiler. Make sure you can change the record structure with minimal changes to your symbol table manipulating functions.

- When using operators in expressions, type inference and 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.

Implementation Requirements

1. Symbol table creation phase

1.1 Implement the data structures and functions for the symbol tables so that they can support the type checking semantic actions.
1.2 Using syntax-directed translation, implement AST tree traversal that triggers semantic actions so that:
1.2.1 A new table is created at the beginning of the program for the global scope.
1.2.2 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.
1.2.3 An entry in the appropriate table is created for each variable defined in the program, i.e. class data members or function's local variables.
1.2.4 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.
1.2.5 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 for shadowed inherited members.
1.2.6 The content of the symbol tables should be output into a file in order to demonstrate their correctness/completeness.
1.2.7 In functions, (including member functions in classes) variables should be defined before being used in the statements. If not, an "undeclared variable" error message is issued.
1.2.8 An identifier cannot be declared twice in the same scope. In such a case, a "multiply declared identifier" message is issued.

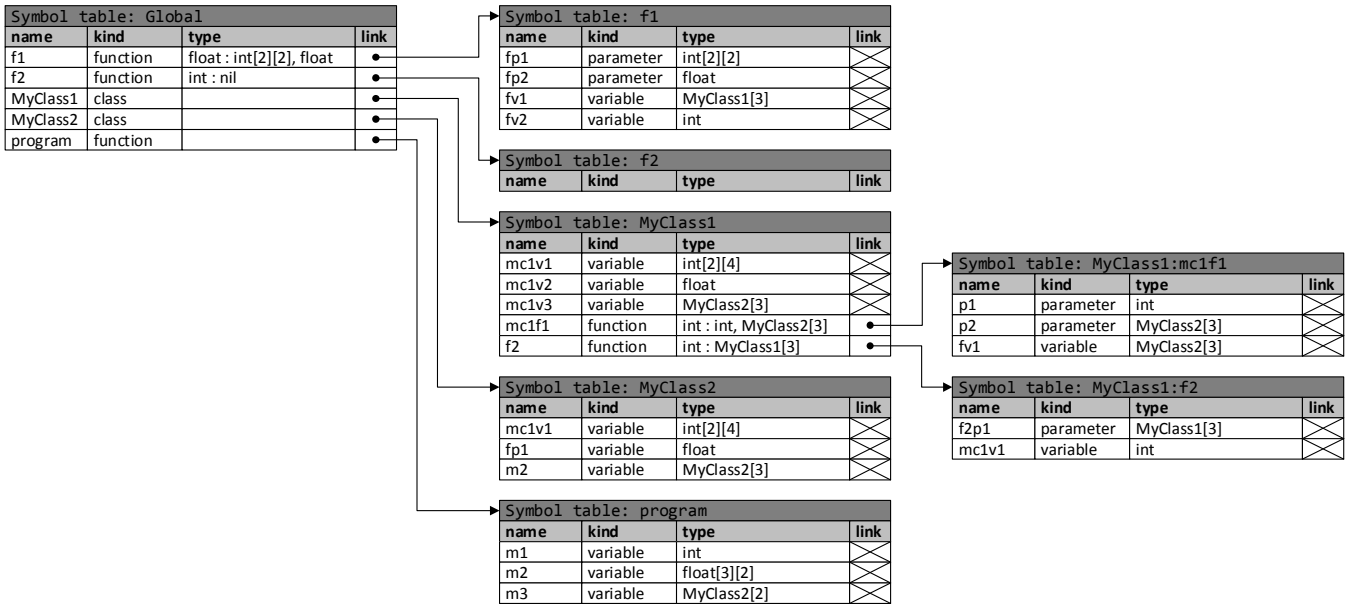
2. Semantic/type checking phase

2.1 Using syntax-directed translation, implement AST tree traversal that triggers semantic actions so that the following semantic checks are done, which should result in error reporting if the check fails**:**
2.1.1 Type checking is applied on expressions, as well as on assignment and return statements.
2.1.2 Any id referred to is defined in the scope where it is used (failure results in the following error messages: undefined local variable, undefined free function, undefined member, undefined class)
2.1.3 Function calls are made with the right number and type of parameters. The types of expressions passed as parameters in a function call must be of the same type as declared in the function declaration.
2.1.4 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.
2.1.5 Circular class dependencies (through data members or inheritance) should be reported as a semantic error.
2.1.6 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 "undefined member" should be issued.
2.1.7 The type of the operand of a return statement must be the same as declared in its function's return type, as declared in the function's declaration.

3. Error reporting

3.1 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.
3.2 Only existing errors should be reported.
3.3 Errors should be reported in synchronized order, even if different phases are implemented and errors are found in different phases.

Symbol table example



Symbol table: Global

name	kind	type	link
f1	function	float : int[2][2], float	•
f2	function	int : nil	•
MyClass1	class		•
MyClass2	class		•
program	function		•

Symbol table: f1

name	kind	type	link
fp1	parameter	int[2][2]	
fp2	parameter	float	
fv1	variable	MyClass1[3]	
fv2	variable	int	

Symbol table: f2

name	kind	type	link

Symbol table: MyClass1

name	kind	type	link
mc1v1	variable	int[2][4]	
mc1v2	variable	float	
mc1v3	variable	MyClass2[3]	
mc1f1	function	int : int, MyClass2[3]	•
f2	function	int : MyClass1[3]	•

Symbol table: MyClass1:mc1f1

name	kind	type	link
p1	parameter	int	
p2	parameter	MyClass2[3]	
fv1	variable	MyClass2[3]	

Symbol table: MyClass1:f2

name	kind	type	link
f2p1	parameter	MyClass1[3]	
mc1v1	variable	int	

Symbol table: MyClass2

name	kind	type	link
mc1v1	variable	int[2][4]	
fp1	variable	float	
m2	variable	MyClass2[3]	

Symbol table: program

name	kind	type	link
m1	variable	int	
m2	variable	float[3][2]	
m3	variable	MyClass2[2]	

```
class MyClass1 {
  int mc1v1[2][4];
  float mc1v2;
  MyClass2 mc1v3[3];
  int mc1f1(int p1, MyClass2 p2[3]) {
    MyClass2 fv1[3];
    …
  }
  int f2(MyClass1 f2p1[3]) {
    int mc1v1;
    …
  }
}
class MyClass2 {
  int mc1v1[2][4];
  float fp1;
  MyClass2 m2[3];
  …
}
program {
  int m1;
  float[3][2] m2;
  MyClass2[2] m3;
  ...
}
float f1(int fp1[2][2], float fp2) {
  MyClass1[3] fv1;
  int fv2;
  …
}
int f2() {
...
}
```

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 3*". 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 with the marker for a demonstration.

Evaluation criteria and grading scheme

Analysis:		
Description of the semantic rules used in the implementation	ind 2.1	1 pts
Description of the purpose of each phase involved in the implemented semantic analysis. For each phase, mapping of semantic actions to AST nodes, along with a description of the effect/role of each semantic action.	ind 2.2	4 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	3 pts
Correct implementation according to the above-stated requirements.	ind 4.4	15 pts
Output of clear error messages (error description and location).	ind 4.4	3 pts
Output of symbol tables in separate 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**