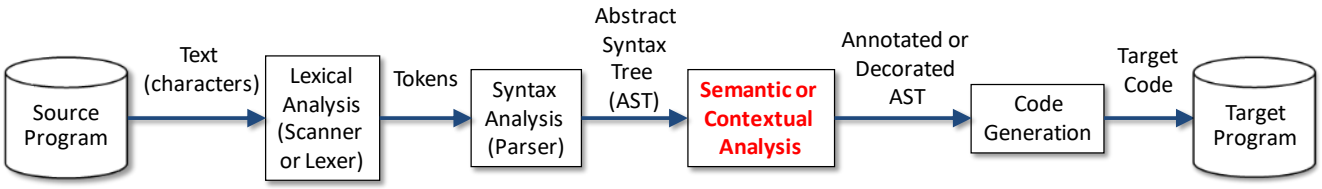
Programming Language Design
Department of Computer Science
Dr. Francisco Ortin
francisco.ortin@mtu.ie

Laboratory 7 – Identification Phase

In this laboratory, we are going to associate variables with their corresponding definition. This is done in the identification phase, the first traversal of semantic analysis.



Motivation

To type check the following program:

```
01.      double a;
02.      int b;
03.
04.      double f(double b) {
05.        return a + b;   // a is global, b is local (parameter)
06.      }
07.
08.      void main() {
09.        int a;
10.        int c;
11.        write a + b + c; // a and c are local, b is global
12.        write f(3.8)   ; // f is global
13.      }
```

The semantic analyzer must know the types of variables `a`, `b`, `c` and `f`. Since we used context-free grammars for the syntactic specification, we have no connection (context) between variables (including functions, e.g. `f`) and their definitions. Thus, an AST traversal must bind variables (including functions) to their definition. That traversal is the identification phase, implemented in the `IdentificationVisitor` class.

The objective of `IdentificationVisitor` is to bind `Variable` nodes to their corresponding `Definition` nodes. To this aim, a new `definition` field must be added to `Variable`. This is the purpose of this lab. In the next lab, that `definition` link will be used to type-check variables. At code generation, the link will be utilized to compute the memory addresses of each variable.

In the program above, the `Variable` node representing the first occurrence of variable `a` (line 5) must be linked to the first global variable definition (`VarDefinition` node). Likewise, `b` in line 5 must be bound to the parameter definition in function `f`. The same happens with `a`, `b`, `c` and `f` variables in the `main` function (see the comments in the code above). Notice that variable `f` (line 12) must be linked to a `FuncDefinition` node. That is why the type of the new `definition`

field in `Variable` must be `Definition`, since `Definition` is a generalization of `VarDefinition` and `FuncDefinition`.

Scopes

Notice that there are two different scopes in C--: global and local. Even though C and Java have potentially infinite scopes, we are only considering these two scopes in C--. Recall that, in C--, no local variables can be defined inside if/else/while bodies.

In C--, a local variable hides a global one with the same name (including functions). This happens with variable a in the `main` function of the previous example.

Go ahead

The first step is to implement a symbol table data structure to provide scoping in C--. Open the `SymbolTable.java` file provided and complete the implementation. To know the expected behavior of each method, see the `SymbolTableTest.java` file. To test the implementation of `insert`, `find`, `set` and `reset` methods, run `SymbolTableTest` with asserts enabled (i.e., pass `-ea` to the Java Virtual Machine upon application execution because, by default, asserts are disabled in Java). The tests are passed when no assert message is displayed.

Second, implement the `IdentificationVisitor` described in this document. Test that all the variables including functions are correctly bound to their definitions, using Instrospector and the `input.txt` file provided.

Finally, make sure your compiler detects all the expected errors in the `input-wrong.txt` file. Use the error handler package already utilized in the previous lab, taken from the mini compiler.