**Theory of Compilation (61304)**

**Assignment 3 – Semantic Analysis**

Submission date: by 4.7.2019

The main goal here is to extend the software that already implements lexical and syntax analysis (developed in Assignment 2), and to add a new capability – semantic analysis.

Main issues to be covered here are:

- Type checking

Here it is checked that each object in the compiled program is used in accordance to its definition.

- Scope analysis

The grammar rule that introduces the possibility of multiple nested scopes and definition of local variables is:

BLOCK 🡪 **{** VAR\_DEFINITIONS; STATEMENTS **}**

Also, functions are considered to be defined in a virtual global (top level) block/scope. The program block that is defined by the rule

PROGRAM 🡪 **program** DEFINITIONS; STATEMENTS **end**

as well as function blocks are considered to be immediate sons of this global block.

Semantic actions to be added to the grammar to achieve these goals will need to work with semantic attributes. Implementation should properly address this issue.

Your code should be properly documented (provide comments that explain what the code is supposed to do).

While working on this part of the project, you can fix the problems remained from the previous stages.

**Semantic rules for the project language**

Whenever an ID is defined in the compiled program, all attributes related to its definition are collected and stored in the symbol table.

When an ID is used in commands and expressions, a check should be performed to ensure that the way it is used fits its definition.

The following semantic rules should be fulfilled:

**Definitions and uses of objects**

* All used IDs must be declared
* Every declared variable must be used at least once bonus (5 points)
* Duplicated declaration of the same name within same scope is forbidden
* It is allowed that in different scopes variables with same name are declared (local declarations).

**Restriction in assignments**

In the following cases, assignment to an ID is forbidden:

* the ID is declared as array (but assignments to array elements are allowed)
* the ID is declared as function

**Restrictions related to use of arrays in expressions**

* expressions can’t refer to entire array
* use of array elements in expressions is allowed; in variable of the form id[index] , the id must be declared as array
* in references to array elements of the form id[index], it is required that index<=size-of-array.

**Type of expression values**

* the type of int\_number is integer and of real\_number is real
* the type of id that is a variable name is determined as either integer or real once it is declared
* the type of array element is determined in accordance with the array’s declaration
* if at least one of the elements in the expression is undefined, then the type of the expression is undefined (error\_type)
* if all elements of the expression are defined as integer, then the expression’s type is integer
* if all elements of the expression are defined, and at least one is real, then the expression’s type is also real
* for types of function call (returned values), see below.

**Type consistency in assignments**

* either left and right sides are both of the same type
* or the left side is real and the right side is integer (the opposite is forbidden)

**Rules related to functions**

* function overloading (multiple functions with the same name) is forbidden
* a function can call itself and any other function that is defined **before** it in the text (a call to a function can appear only after its declaration)
* the number of parameters in a function call should match the function’s definition
* the types of parameters in a function call should match the function’s definition bonus (12 points)
* returned values:
* a return statement residing in a void function or in the main/program block may not return a value
* a function whose returned type is not void, must contain at least one return statement with a returned value
* the type of the expression in a return statement must match the type of function’s returned value bonus (8 points)
* Function can’t have a local variable with the same name as its parameter.

**TASKS**

1. Implementation of symbol table:

* For each scope there is a separate symbol table; these tables are connected to reflect the hierarchy of scopes (see file Scope checking on the course site)
* For each element (each ID) in a table, the following information should be stored:

עבור משתנים:

* שם
* טיפוס (integer, real)
* גודל (במידה ומדובר על משתנה שהוגדר כמערך)

עבור פונקציות:

* שם
* טיפוס של ערכים מוחזרים
* מספר הפרמטרים
* רשימת הפרמטרים (כאשר כל פרמטר הוא למעשה סוג של משתנה שמוגדר בפונקציה; כך שגם עבור הפרמטרים נשמר מידע דומה למידע על משתנים)
  + Symbol table is not necessarily implemented as an array (especially because amount of objects declared in the compiled program is not known in advance).
  + The simplest structure for a symbol table is a list of elements stored in it.

However, it doesn’t allow for efficient search by ID’s name.

* + Better structures are:
  + binary search tree (each node holds an element)
  + hash table – this is the best for efficiency; there are ready-to-use packages on Internet (or you can develop it yourself)
* Implement all functions needed for the work with symbol table. The list of functions is presented in section “Symbol Table Interface” in file Scope checking.

1. Define and implement a syntax-directed scheme (combining the grammar rules and semantic actions) that allows to perform type checking and scope checking. This requires:

* definition and classification of the needed attributes (synthesized, inherited)
* definition of semantic actions that use calls to symbol table functions for implementation of type and scope checking
* integration of the semantic actions in functions of the parser; see the example on the course site:

[Analysis with attributes of mixed types](http://md.hit.ac.il/mod/resource/view.php?id=250547)

1. Error handling:

- Each time a semantic error is discovered, the program should send an

appropriate error message that clearly explains

* + what is wrong
  + where the error occurred (line number)

- No error recovery is made when a semantic error occurs, and compiler should just continue its regular action.