Theory of Compilation (61304)

Lecturer: Dr. Mark Trakhtenbrot

Assignment 3 – Semantic Analysis

The work should be submitted by February 20, 2020.

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.

The core 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 in the language is:

BLOCK → **block** DECLARATIONS **begin** STATEMENTS **end**

Note 1: your code should be properly documented (provide comments that explain what the code is supposed to do).

Note 2: while working on this stage of the project, you can fix the problems remained from the previous stages.

Type checking

Whenever definition of a variable or of a type appears in the compiled program, all attributes related to this definition are collected and stored in the symbol table.

Important note: while a variable or a type is defined/localized in a block, field is defined/localized in a structure.

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:

Definition of used objects

- All used names must be declared
- Duplicated declaration of the same name within same scope is forbidden.
  Same for fields: duplicated field names within the same structure are forbidden.
- It is allowed that in different scopes variables/types with same name are declared (local declarations).
  For example in one scope variable by name "salary" can be defined as integer, and in another – as real.

Restrictions on assignments

- VAR_ELEMENT in the left-hand side of assignment can be of a simple type or of an enumeration type, but it can not be of an array type or of a structure type.
- Type of the left-hand side must be identical to the type of expression in the right-hand side.

Restrictions in expressions

- Simple expression must be of a simple type or of an enumeration type.
- Arithmetic operations are only applicable to values of simple types (integer or real). Such operations can not be applied to values of enumeration, array and structure types.
- If at least one of the IDs in the expression is not defined, then type of the expression is also not defined (error_type)
- If all arguments of arithmetic expression are of the same simple then the entire expression is of this simple type; otherwise, the expression is of error_type

Restrictions in access to array element

- in id[expression] , the id must be declared as array
- expression appearing as index in access to array element must be of integer type.

Restriction in SWITCH statement

- KEY must be either integer or enumeration type
- Type of every KEY_VALUE must be identical to the type of KEY.

TASKS

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 finctions of the parser; see the example on the course site:

     Analysis with attributes of mixed types

2. Implementation of symbol table:
   - For each scope there is a separate symbol table; these tables are connected to reflect the hierarchy of scopes (see the file Scope checking on the course site)
   - For each element in a table, the following information should be stored:
     - Name
     - Role (variable, type)
     - For variable - store its type.
       Note: if an ID is defined as array, then information about the type of this ID should include the following data: size and type of elements in the array.
     - For type:
       If it is a structure: store the list of its fields
       If it is enumeration: store the list of its values

   - The simplest data structure for a symbol table is array of elements stored in it. However, it doesn't allow for efficient search by ID's name.
   - Better solutions 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)
       Bonus will be given for implementation that supports efficient search.
   - The list of functions needed for the work with symbol table. is presented in section "Symbol Table Interface" in file Scope checking.
   - You should Implement all these functions in accordance with the data structure selected for symbol table.

3. 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.