Next Phase of the Semester Project 2025 : The NAME-SCOPE-Rules for"SPL"

In general:

- **SPL is statically scoped**, such that a **Hash Table (or Relational Database) can be used as Symbol Table**.
- In this phase of the project, we want to implement a tree-crawling static semantic checker to find out whether all user-defined-names are compliant with the Name-Scope-Rules which are defined below in this work sheet.

And now: **HAPPY WORKING :)**

Syntactic Rule

```
SPL_PROG  ::=   glob  { VARIABLES }
                proc  { PROCDEFS }
                func  { FUNCDEFS }
                main { MAINPROG }
```

Corresponding Semantic Rules

- SPL_PROG, together with the entire syntax tree underneath, forms the "**Everywhere**" scope.
- Inside the "Everywhere" scope, the VARIABLES node together with its entire sub-tree forms the "**Global**" scope.
- Inside the "Everywhere" scope, the PROCDEFS node together with its entire sub-tree forms the "**Procedure**" scope.
- Inside the "Everywhere" scope, the FUNCDEFS node together with its entire sub-tree forms the "**Function**" scope.
- Inside the "Everywhere" scope, the MAINPROG node together with its entire sub-tree forms the "**Main**" scope.
- Inside the "Everywhere" scope, NO **variable name may be identical with any function name**
- Inside the "Everywhere" scope, NO **variable name may be identical with any procedure name**
- Inside the "Everywhere" scope, NO **function name may be identical with any procedure name**
- **Otherwise, the semantics checker must emit a name-rule-violation notification !**
- For each syntax tree node N anywhere in the "Everywhere" scope, appropriate Scope Information must be written into the corresponding field of the Symbol Table which is linked to the node N, whereby the ID of node N serves as "foreign key" to the Symbol Table. For this purpose, every node N in the Syntax Tree must have its own unique Node identity number.

Syntactic Rule

```
VARIABLES ::=   VAR VARIABLES
VAR       ::=   user-defined-name
```

Corresponding Semantic Rules

- Inside the Sub-Tree underneath the left-hand-side node VARIABLES, no two VAR names are allowed to be identical. (**No double-declaration** in *same* scope)
- Otherwise, the semantics checker must emit a name-rule-violation notification **!**

Syntactic Rule

PROCDEFS ::= PDEF PROCDEFS

Corresponding Semantic Rules

- Inside the Sub-Tree underneath the left-hand-side node PROCDEFS, no two PDEF-names are allowed to be identical. (**No double-declaration** in *same* scope)
- Otherwise, the semantics checker must emit a name-rule-violation notification !

Syntactic Rule

FUNCDEFS ::= FDEF FUNCDEFS

Corresponding Semantic Rules

- Inside the Sub-Tree underneath the left-hand-side node FUNCDEFS, no two FDEF-names are allowed to be identical. (**No double-declaration** in *same* scope)
- Otherwise, the semantics checker must emit a name-rule-violation notification !

Syntactic Rule

PDEF ::= NAME (PARAM) { BODY }
BODY ::= **local** { MAXTHREE } ALGO

Corresponding Semantic Rules

- PDEF, together with its entire Sub-Tree, constitutes its own "**Local**" scope.
- For each syntax tree node N anywhere in the "**Local**" scope, appropriate Scope Information must be written into the corresponding field of the Symbol Table which is linked to the node N, whereby the ID of node N serves as "foreign key" to the Symbol Table.
- Inside the Sub-Tree underneath the left-hand-side node PDEF, no VAR-name in the Sub-tree of MAXTHREE may be identical with any VAR-name in the Sub-Tree of PARAM, (i.e.: no "shadowing" of parameter names by local variable declarations).
- Otherwise, the semantics checker must emit a name-rule-violation notification !

Syntactic Rule

FDEF ::= NAME (PARAM) { BODY ; **return** ATOM }
BODY ::= **local** { MAXTHREE } ALGO

Corresponding Semantic Rules

- FDEF, together with its entire Sub-Tree, constitutes its own "**Local**" scope.
- For each syntax tree node N anywhere in the "**Local**" scope, appropriate Scope Information must be written into the corresponding field of the Symbol Table which is linked to the node N, whereby the ID of node N serves as "foreign key" to the Symbol Table.
- Inside the Sub-Tree underneath the left-hand-side node FDEF, no VAR-name in the Sub-tree of MAXTHREE may be identical with any VAR-name in the Sub-Tree of PARAM, (i.e.: no "shadowing" of parameter names by local variable declarations).
- Otherwise, the semantics checker must emit a name-rule-violation notification !

Syntactic Element

MAXTHREE

Corresponding Semantic Rules

- **Inside the Sub-Tree underneath** the node MAXTHREE, no two VAR-names are allowed to be identical. (**No double-declarations**)
- Otherwise, the semantics checker must emit a name-rule-violation notification **!**

Syntactic Rule

MAINPROG ::= **var** { VARIABLES } ALGO

Corresponding Semantic Rules

- Inside the Sub-Tree underneath the node VARIABLES, no two VAR-names are allowed to be identical. (**No double-declarations**)
- Otherwise, the semantics checker must emit a name-rule-violation notification !

Syntactic Element

VAR

Corresponding Semantic Rules :

If VAR **is "in" the Sub-Tree** of some ALGO, **then**:

IF this VAR's ALGO belongs to the **Local scope of a Procedure**, then:

- If this VAR's name occurs in **this** Procedure's PARAM then this VAR is ***semantically identical*** with the parameter of that name: Update the Symbol Table accordingly !
- If this VAR's name occurs in **this** Procedure's locally declared variables, then this VAR is ***semantically identical*** with the locally declared variable of that name. Update the Symbol Table accordingly !
- If this VAR's name occurs *neither* in its Procedure's locally declared variables, *nor* in its Procedure's PARAM, then this VAR *must* be semantically identical with a *globally* declared variable of that name. Update the Symbol Table accordingly !
- Otherwise, the semantics checker must emit an **undeclared variable** notification **!**

IF this VAR's ALGO belongs to the **Local scope of a Function**, then:

- If this VAR's name occurs in **this** Function's PARAM then this VAR is ***semantically identical*** with the parameter of that name: Update the Symbol Table accordingly !
- If this VAR's name occurs in **this** Function's locally declared variables, then this VAR is ***semantically identical*** with the locally declared variable of that name. Update the Symbol Table accordingly !
- If this VAR's name occurs *neither* in its Function's locally declared variables, *nor* in its Function's PARAM, then this VAR *must* be semantically identical with a *globally* declared variable of that name. Update the Symbol Table accordingly !
- Otherwise, the semantics checker must emit an **undeclared variable** notification **!**

IF this VAR's ALGO belongs to the **Main scope**, then:

- If this VAR's name occurs in Main's own declared variables, then this VAR is ***semantically identical*** with the main-declared variable of that name. Update the Symbol Table accordingly !
- If this VAR's name does not occur in its Main's declared variables, then this VAR *must* be semantically identical with a *globally* declared variable of that name. Update the Symbol Table accordingly !
- Otherwise, the semantics checker must emit an **undeclared variable** notification **!**

You will have noticed that these Variable-Declaration-Rules are very similar to the Rules which you know from your usual programming languages such as C++ or Java.

On the basis of these specifications, you and your Project Group members can now:

- implement a suitable **Symbol Table** that is "connected" to the Syntax Tree by means of "foreign key" Node ID numbers,
- implement a **Tree-Crawling Algorithm** that can **search** "inside" a given Syntax Tree (which comes from the Parser) for the necessary **Semantic Information**, and which **updates** the Symbol Table accordingly,
- implement the feature that **emits Name-Rule-Violation messages** wherever such violations are detected
- implement the feature that **emits Undeclared-Variable messages** wherever such errors are detected.
- **By the way**: It is of course *allowed* that the *same* variable *name* occurs in *different scopes:* such cases the two variables are *semantically different* entities, *in spite of* carrying the same name.

IF anything is still un-clear, then visit the Tutors in the Orange Lab on Tuesdays (as usual)