Prof. Dr. M. Jäger
FB MNI

Compilerbau - Praktikumsaufgabe 4 „Semantische Analyse“

Schritt 1 (SPL Typsystem kennenlernen)

Machen Sie sich mit der Handhabung von Typen vertraut. Studieren Sie die vorgegebenen Definitionen von *PrimitiveType*, *ArrayType* und *ParamTypeList* im Paket *types*.

Schreiben Sie einige Typen in SPL auf und konstruieren Sie den entsprechenden Typgraphen, sowohl auf dem Papier als auch in Form von Konstruktoraufrufen.

Schritt 2 (Symboltabellen kennenlernen)

Machen Sie sich mit Symboltabellen-Einträgen und Symboltabellen vertraut. Studieren Sie die vorgegebenen Definitionen von *VarEntry*, *TypeEntry*, *ProcEntry* und *Table* im Paket *tables*.

Schreiben Sie ein kleines SPL-Programm auf, dass mindestens je eine Typdefinition, eine Prozedurdefinition, eine Parameterdefinition und eine Variablendefinition enthält. Konstruieren Sie die entsprechenden Symboltabellen, sowohl auf dem Papier als auch in Form von Konstruktoraufrufen.

Vorschlag: Schreiben Sie ein kleines Programm, das eine Symboltabelle anlegt, ein paar Einträge vornimmt und am Ende die Einträge ausgeben lässt. Machen Sie dasselbe auch mit einer mehrstufigen Symboltabelle.

Schritt 3 (Visitor-Muster kennenlernen)

Studieren Sie die Einführung zur Verwendung des Visitor-Musters aus den Materialien zur Praktikumsaufgabe 3.

Schritt 4 (Semantische Analyse Durchgang 1: Typen und Symboltabellen konstruieren)

Schreiben Sie nun den ersten Teil der semantischen Analyse, der ausgehend von einem abstrakten Syntaxbaum die zugehörigen Typen und Symboltabellen konstruiert. Benutzen Sie das Visitor-Muster.

Studieren dazu die vorgegebenen Schnittstellen im Paket *semant*.

In der Methode *main* wird die Instanzvariable ’showTables’ auf true gesetzt, falls der Benutzer des SPL-Compilers den Kommandozeilenschalter ’`--tables`’ angegeben hat. Sie können diese Variable zum Steuern Ihrer Ausgabe der Symboltabellen benutzen.

Schritt 5 (Semantische Analyse Durchgang 2: Prüfung der Prozedur-Rümpfe)

Schreiben Sie nun den zweiten Teil der semantischen Analyse, der ausgehend von einem abstrakten Syntaxbaum die Prozedur-Rümpfe auf semantische Korrektheit überprüft. Benutzen Sie das Visitor-Muster.

Als Anhaltspunkt, was es alles zu überprüfen gilt, folgt hier die Liste der semantischen Fehler, die die Referenzimplementierung erkennt:

```
undefined type ... in line ...
... is not a type in line ...
redeclaration of ... as type in line ...
parameter ... must be a reference parameter in line ...
redeclaration of ... as procedure in line ...
redeclaration of ... as parameter in line ...
redeclaration of ... as variable in line ...
assignment has different types in line ...
assignment requires integer variable in line ...
'if' test expression must be of type boolean in line ...
'while' test expression must be of type boolean in line ...
undefined procedure ... in line ...
call of non-procedure ... in line ...
procedure ... argument ... type mismatch in line ...
procedure ... argument ... must be a variable in line ...
procedure ... called with too few arguments in line ...
procedure ... called with too many arguments in line ...
expression combines different types in line ...
comparison requires integer operands in line ...
arithmetic operation requires integer operands in line ...
undefined variable ... in line ...
... is not a variable in line ...
illegal indexing a non-array in line ...
illegal indexing with a non-integer in line ...
procedure 'main' is missing
'main' is not a procedure
procedure 'main' must not have any parameters
```

Schritt 6 (Testen)

Zur Überprüfung des Semantik-Moduls müssen Sie einen ganz neuen Satz von Testprogrammen konstruieren. Jedes dieser Programme muss gezielt einen semantischen Fehler enthalten. Dann können Sie verifizieren, dass Ihr Semantik-Modul diesen Fehler erkennt und eine geeignete Fehlermeldung ausgibt.