THM | CAMPUS GIESSEN
TECHNISCHE HOCHSCHULE MITTELHESSEN

MNI
Mathematik, Naturwissenschaften und Informatik

THM | Campus Gießen | Wiesenstraße 14 | 35390 Gießen

Prof. Dr. Uwe Meyer
(Dozent: Björn Pfarr)
University of Applied Sciences

Praktikumsaufgabe 4
Compilerbau

Schritt 1 (SPL Typsystem kennenlernen)

Studieren Sie im Praktikumsskript das Kapitel über Typen, Symboltabellen und Semantische Analyse für SPL.
Machen Sie sich mit der Handhabung von Typen vertraut. Studieren Sie die vorgegebenen Definitionen von `PrimitiveType`, `ArrayType` im Paket `types` und `ParameterType` im Paket `table`. In C heißen die entsprechenden Strukturen `Type` sowie `ParameterType`.
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 `VariableEntry`, `TypeEntry`, `ProcedureEntry` und `SymbolTable` im Paket `tables`. Die entsprechenden C-Strukturen heißen `Entry` sowie `SymbolTable`.

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 im Praktikumsskript.

Schritt 4a (Aufbauen der Symboltabelle)

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.

Ergänzen Sie dazu die Datei:
C: `src/phases/_04a_tablebuild/tablebuild.c`
Java:
`src/main/java/de/thm/mni/compilerbau/phases/_04a_tablebuild/TableBuilder.java`

Die entsprechenden Stellen sind mit einem `TODO` markiert. Schrecken Sie nicht davor zurück zusätzliche Klassen oder Methoden/Funktionen einzuführen! Insbesondere für das Visitor-Muster werden Sie diese brauchen.

Bei der Instanziierung Ihres `TableBuilders`, beziehungsweise beim Aufruf Ihrer `buildSymbolTable`-Funktion wird ein bool'scher Wert übergeben, der bestimmt, ob eine textuelle Ausgabe der Symboltabellen stattfinden soll. Er ist `true` falls beim Aufruf des Compilers die Option `--tables` angegeben wurde.
Nutzen Sie diese, um an den richtigen Stellen während des Tabellenaufbaus die Tabelle auszugeben! Um dasselbe Ausgabeformat wie die Referenzimplementierung zu erreichen steht Ihnen eine Hilfsmethode/-funktion zur Verfügung (`printSymbolTableAtEndOfProcedure`).

Schritt 4b (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.

Ergänzen Sie dazu die Datei:
C: src/phases/_04b_semant/procedurebodycheck.c
Java: src/main/java/de/thm/mni/compilerbau
/phases/_04b_semant/ProcedureBodyChecker.java

Als Anhaltspunkt, was es alles zu überprüfen gilt, finden Sie auf der nächsten Seite die Liste der semantischen Fehler, die die Referenzimplementierung erkennt, und Ihre Implementierung erkennen muss.

Ihre Compiler muss zwingend den vorgegebenen Exit Code ausgeben, da dieser in den Tests Ihrer Hausübungen abgeprüft wird. Die Fehlermeldungen mit ihren jeweiligen Exit-Codes sind im Skelett bereits vorgegeben. Sie finden sie in der Datei:
C: `src/util/errors.h`
Java: `src/main/java/de/thm/mni/compilerbau/utils/SplError.java`

Hinweis: Möglicherweise müssen Sie unterscheiden, ob ein Ausdruck einen arithmetischen, oder einen Vergleichsoperator verwendet. Dafür hat die `BinaryOperator`-Klasse in Java einige Hilfsmethoden, die allerdings noch nicht implementiert sind. Implementieren Sie diese, wenn Sie möchten. Unvollständige Implementierungen solcher Funktionen finden Sie auch in C.

Schritt 5 (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. Die beiden Teilschritte können Sie mit den Compileroptionen `--tables` sowie `--semant` testen.

Anhang (Liste der Fehlerfälle)

Exit Code	Fehlermeldung
101	undefined type ... in line ...
102	... is not a type in line ...
103	redeclaration of ... as type in line ...
104	parameter ... must be a reference parameter in line ...
105	redeclaration of ... as procedure in line ...
106	redeclaration of ... as parameter in line ...
107	redeclaration of ... as variable in line ...
108	assignment has different types in line ...
109	assignment requires integer variable in line ...
110	'if' test expression must be of type boolean in line ...
111	'while' test expression must be of type boolean in line ...
112	undefined procedure ... in line ...
113	call of non-procedure ... in line ...
114	procedure ... argument ... type mismatch in line ...
115	procedure ... argument ... must be a variable in line ...
116	procedure ... called with too few arguments in line ...
117	procedure ... called with too many arguments in line ...
118	expression combines different types in line ...
119	comparison requires integer operands in line ...
120	arithmetic operation requires integer operands in line ...
121	undefined variable ... in line ...
122	... is not a variable in line ...
123	illegal indexing a non-array in line ...
124	illegal indexing with a non-integer in line ...
125	procedure 'main' is missing
126	'main' is not a procedure
127	procedure 'main' must not have any parameters