THM | CAMPUS GIESSEN
TECHNISCHE HOCHSCHULE MITTELHESSEN

MNI
Mathematik, Naturwissenschaften und Informatik

THM | Campus Gießen | Wiesenstraße 14 | 35390 Gießen

Prof. Dr. Uwe Meyer
University of Applied Sciences

Praktikumsaufgabe 6
Compilerbau

Zunächst ein Hinweis: Der Code, den Ihr Compiler erzeugt, muss in keiner Hinsicht so aussehen, wie der Code der Referenzimplementierung. Sie dürfen sich zwar gerne diesen Code ansehen; versuchen Sie aber nicht, unter allen Umständen gleichen Code zu erzeugen. Wichtigstes Ziel in dieser Phase ist korrekter Code, d.h. Code, der genau das Verhalten zeigt, das laut Sprachdefinition von ihm gefordert wird.

Schritt 1 (Informieren)

Studieren Sie gründlich das Kapitel zum Assembler-Code-Generator für SPL im Praktikumsskript.

Schritt 2 (Code-Erzeugung anhand des AST)

Programmieren Sie einen rekursiven Durchgang durch die abstrakte Syntax, so dass zu jedem Knotentyp der passende ECO32-Assemblercode ausgegeben wird.

Ergänzen Sie dazu die Datei
C: `src/phases/_06_codegen/codegen.c`
Java:
`src/main/java/de/thm/mni/compilerbau/phases/_06_codegen/CodeGenerator.java`
an der angegebenen Stelle. Wie immer gilt: Erstellen Sie Hilfsklassen und -funktionen!

Entwerfen Sie zunächst jeweils ein ganz kurzes SPL-Programm, das den ins Auge gefassten Knotentyp beim Übersetzen produziert (so etwas sollte eigentlich aus Lösungen zu früher gestellten Aufgaben schon zur Verfügung stehen). Programmieren Sie dann die Codegenerierung für diesen Knotentyp. Prüfen Sie zuletzt, ob der erzeugte Code für Ihr SPL-Programm korrekt funktioniert.

Hier ist ein Vorschlag für die Reihenfolge der Knotentypen:

- `IntLiteral`
- `BinaryExpression` (nur Arithmetik, keine Vergleiche)
- `NamedVariable` (ohne Berücksichtigung von Referenzvariablen)
- `VariableExp`
- `AssignStatement`
- `ArrayAccess` (nicht vergessen: Indexgrenzen-Überprüfung, s.u.)
- `WhileStatement`
- `BinaryExpression` (nur Vergleiche, keine Arithmetik)
- `IfStatement`
- `CallStatement` (ohne Berücksichtigung von Referenzvariablen)
- `ProcedureDeclaration` mit folgenden Teilaufgaben:
  - Framegröße berechnen
  - Prozedur-Prolog ausgeben
  - Code für Prozedurkörper erzeugen

- Prozedur-Epilog ausgeben

Hinweise:

- Falls die Indexgrenzen-Überprüfung fehlschlägt, lassen Sie den Code zum Label `_indexError` verzweigen.
  Das ist eine Bibliotheksfunktion, die das Programm mit einer Fehlermeldung abbricht.
- Denken Sie an die Überprüfung, ob genügend Register für Hilfsvariable zur Verfügung stehen. Beantworten Sie dazu folgende Fragen:
  1. Wie merkt sich der Codegenerator, welche Register belegt sind, und welche noch zur Verfügung stehen (Stichwort „Registerstack“)?
  2. An welchen Stellen wird ein Register „allokiert“? Was muss während der Registerallokation getan werden?
  3. An welchen Stellen wird ein Register freigegeben? Was muss dafür getan werden?
- Auf die Frage nach der Registerverwaltung gibt es für die Java- und die C-Lösung unterschiedliche Ansätze. Fragen Sie im Zweifel die Tutoren um Rat!
- Für die Ausgabe von Eco32-Assemblercode steht Ihnen im Skelett eine Hilfe zur Verfügung. Diese finden sie in der Klasse `CodePrinter`, beziehungsweise in der Datei `codeprint.h`. Nutzen Sie diese.
- (Nur Java) Es gibt die Klasse `Register`. Machen Sie sich Gedanken, warum es diese Klasse gibt? Welchen leicht zu begehenden Fehler bei der Implementierung des Codegenerators versucht sie zu verhindern?

Schritt 3 (Fehler bei Referenzparametern beheben)

Wenn Sie die vorgeschlagene Reihenfolge befolgt haben, verhält sich Ihr Compiler bei Referenzparametern noch nicht richtig. Untersuchen Sie, an welcher Stelle ein Programm, das Referenzparameter benutzt, noch fehlerhaft ist. Ergänzen Sie den Code-Generator so, dass auch SPL-Programme mit Referenzparametern richtig übersetzt werden.

Hinweis: Hier gibt es eine elegante Lösung, die Sie mit nur einer Zeile Java/C-Code implementieren können. Schauen Sie sich Ihren bisher generierten Assemblercode an und finden Sie heraus welche Instruktionen „zu viel“ sind. Vergleichen Sie Ihren Assemblercode mit den Beispielen aus dem Praktikumsskript oder der Vorlesung. Überlegen Sie dann, woher die überflüssigen Instruktionen stammen.

Schritt 4 (Testen)

Überzeugen Sie sich von der Korrektheit des erzeugten Codes für die etwas größeren Testprogramme (bigtest, Queens, Sierpinski, Sortieren eines Arrays).