Codeoptimierung – Übungsblatt 1
pseudo (1) – Einzelarbeit

Abgabe: 2.5.00 vor 9:30h

pseudo ist ein Compiler, der Pseudocode nach Java Bytecode übersetzt. (Allerdings ist es dann kein "Pseudo"-Code mehr, sondern richtiger Code!) Im ersten Teil sollen lexikalische und Syntaxanalyse für den Pseudocode generiert werden. Gegeben sei folgendende Beschreibung:

Pseudocode-Notation

Zuweisung	Variable := Ausdruck
ArithmOperator	$+ - * /$ DIV MOD
Ausdruck	Variable ganze Zahl aus $\mathbb{Z}$ reelle Zahl aus $\mathbb{R}$ Variable ArithmOperator Variable Ausdruck ArithmOperator Ausdruck
LogOperator	AND OR
VergOperator	= <> > < <= >=
Vergleich	Ausdruck VergOperator Ausdruck Vergleich LogOperator Vergleich NOT Vergleich
Block	BEGIN ... END
Test	IF Vergleich THEN ... ELSE ... ENDIF IF Vergleich THEN ... ENDIF
Schleife	WHILE Vergleich DO ... ENDWHILE REPEAT ... UNTIL Vergleich FOR Variable := Ausdruck TO Ausdruck DO ... ENDFOR
Eingabe	READ(Variable)
Ausgabe	WRITE(Ausdruck)

Man beachte: Es gibt keine Variablendefinitionen. Zeichenfolgen, die mit einem Buchstaben beginnen und auf die Buchstaben oder Ziffern folgen, werden als implizit deklarierte Variablen betrachtet. Ebenso wird vorausgesetzt, daß Punktrechnung vor Strichrechnung in Ausdrücken sowie NOT vor AND und AND vor OR in Vergleichen geht. DIV und MOD werden zur ganzzahligen Division bzw. zur Bestimmung dessen Restes und / zur Division von reellen Zahlen verwendet. Leerzeichen, Tabulatoren und Zeilenumbruch werden ignoriert. Jede Zuweisung, jeder Block, jede Schleife sowie jede Ein- und Ausgabe wird durch ein Semikolon terminiert.

Pseudocode in EBNF

Programm ::= { Anweisung ; } .
Anweisung ::= Zuweisung | Block | Test | Schleife | Eingabe | Ausgabe .
Zuweisung ::= Variable := Ausdruck .
Ausdruck ::= Variable | [Vorzeichen] ganzeZahl | [Vorzeichen] reelleZahl |
  Ausdruck ArithmOperator Ausdruck .
ArithmOperator ::= + | − | ∗ | / | DIV | MOD .
LogOperator ::= AND | OR .
VergOperator ::= = | <> | < | > | <= | >= .
Vergleich ::= Ausdruck VergOperator Ausdruck |
  Vergleich LogOperator Vergleich | NOT Vergleich .
Block ::= BEGIN Anweisung END .
Test ::= IF Vergleich THEN Anweisung [ELSE Anweisung] ENDIF.
Schleife ::= WHILE Vergleich DO Anweisung ENDWHILE |
  REPEAT Anweisung UNTIL Vergleich |
  FOR Variable := Ausdruck TO Ausdruck DO Anweisung ENDFOR .
Eingabe ::= READ (Variable) .
Ausgabe ::= WRITE (Ausdruck) .

Aufgaben:

Loggen Sie sich auf einer Solaris-Maschine ein und geben Sie die folgenden Befehler ein:

```
mkdir -p ~/codeopt/ub1
cd ~/codeopt/ub1
cp ~mueller/classes/codeopt/ub1/Makefile .
make install
```

1. (4 Punkte): Spezifizieren Sie die lexikalische Analyse als pseudo.l, so daß mit Flex ein Lexer generiert werden kann. Dazu gehört die Erkennung von Opertoren, Schlüsselwörtern, Zahlen (int als z.B. 123 und real als z.B. 12.3), Bezeichnern und die Behandlung von Whitespaces. Als Token sollen antsprechende Konstanten generiert werden, die in der Syntaxspezifikation enthalten sind. Es soll eine Angabe der Zeilennummer ermöglicht werden. Bezeichner- und Zahlenwerte werden in yylval as str_ptr / int_val / double_val zur Verfügung gestellt.

2. (6 Punkte): Spezifizieren Sie die syntaktische Analyse als pseudo.y, so daß mit Bison ein Parser generiert werden kann. Dazu gehört die Deklaration aller Token (mit Typisierung für Bezeichner und Zahlenwerte), Assoziativität und Vorrang der Operatoren, Grammatikregeln gemäß EBNF (nur leicht abgewandelt, wo unbedingt nötig) sowie Fehlerausgabe mit Zeilennummer.

Die Abgabe umfaßt die Dateien pseudo.l sowie pseudo.y und erfolgt per Upload durch http://eiche.informatik.hu-berlin.de:8081/codeopt