PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen, J. Berthold

D-35032 Marburg
Hans-Meerwein-Straße
Lahnberge

Klausur zu „Grundlagen des Compilerbaus“, WS 2005/06

16. Februar 2006

Hinweise:

- **Bearbeitungszeit:** 105 Minuten
  Gesamtpunktzahl: 82 Punkte
  Zum Bestehen der Klausur sind **33 Punkte** erforderlich.
- Hilfsmittel sind nicht erlaubt.
- Jede Aufgabe ist auf einem eigenen Blatt zu bearbeiten.
- Bis auf das Konzeptpapier sind alle ausgehändigten Blätter zurückzugeben.
  Teile des Konzepts, die bewertet werden sollen, sind als solche zu kennzeichnen.

Name: .....

Mat.-Nr.: ..... **Studienfach:** .....

Studiengang: ◯ **Bachelor** ◯ **Diplom** ◯ **Lehramt** ◯ **Magister**

Aufgabe	max. Punktzahl	erreichte Punktzahl
1	8	
2	12	
3	15	
4	14	
5	10	
6		
7		

Wiederholungsblatt mit mündlichen Aufgaben

Aufgaben

Aufgabe 1: T-Diagramme

8 Punkte

Sie haben die Aufgabe, einen Java-Übersetzer zu entwickeln, der auf einer SGI-Workstation läuft und Code für die Sony Playstation 3 erzeugt. Ihr Auftraggeber verlangt, dass neu zu implementierende Teile in Java geschrieben sind.
Ihnen stehen zwei auf der SGI lauffähige C-Übersetzer zur Verfügung, welche Maschinencode für die SGI bzw. für die Playstation erzeugen. Geben Sie in Form von T-Diagrammen an, wie Sie (mit möglichst wenig Aufwand) vorgehen können und an welchen Stellen "Handarbeit" erforderlich ist.

Aufgabe 2: Lexikalische Analyse

12 Punkte

a) Geben Sie (über einem geeigneten Alphabet) reguläre Definitionen für die folgenden Symbolklassen an. Sie können zur Übersicht Hilfsdefinitionen, Aufzählungen und Negation verwenden. / 7

Beispiel: `[1-7]`=Ziffer 1,2,3,4,5,6 oder 7. `[^123]`= bel. Zeichen *außer* 1,2,3.

- **AttValue:** Eine von `"` umschlossene Zeichenkette ohne `<` und `&`
- **CharData:** Eine nichtleere Zeichenkette ohne `<` und `&`
- **Comment:** eine von `<!--` und `-->` umschlossene Zeichenkette , welche *kein* `--` *enthält*
- **ERef:** Das Zeichen `&`, gefolgt von einem Bezeichner, danach das Zeichen `;`

Bezeichner beginnen mit einem Buchstaben und enthalten nur Buchstaben und Ziffern.

b) Für die in a) definierten Symbolklassen (in der angegebenen Reihenfolge) werde mit ALEX ein Scanner erstellt. Welches Resultat liefert der Scanner für die folgende Eingabe? (begründen Sie Ihre Antwort) / 5

'`In <-- diesem Spezial-> XML hat "--->""&bes;ondere" Bedeutung-->`'

Aufgabe 3: Recursive-Descent-Parsing

15 Punkte

a) Überführen Sie die nebenstehende EBNF-Spezifikation in eine kontextfreie Grammatik (ohne spezielle EBNF-Konstrukte), welche die gleiche Sprache erzeugt. / 3

```
S = E ( '+' E ) ?
E = '(' S ')' | D
D = Z+
Z = '0' | '1'| '2' | ... | '9'
```

b) Ändern Sie (ggf.) die Grammatik so ab, dass sie aus $LL(1)$ ist und bestimmen Sie zu Ihrer Grammatik aus a) die la-Mengen. Produktionen für Z können dabei (verständlicherweise) weggelassen werden. / 5

c) Definieren Sie mit Parserkombinatoren und Hilfsfunktionen einen *recursive-descent*-Parser, der den Zahlwert der dargestellten Summe berechnet. Verwenden Sie die nebenstehenden Hilfsfunktionen. / 7

```
lachoice :: [[[tkn]]] -> [Parser tkn b]
            -> Parser tkn b
digit    :: Parser Char Char
tok      :: Eq a => [a] -> Parser a [a]
readInt  :: String -> Int
```

Aufgabe 4: LL(k) und SLR(1)

14 Punkte

$$G_1: S \rightarrow Ba \mid Cc \qquad B \rightarrow bB \mid \varepsilon \qquad C \rightarrow bC \mid \varepsilon$$

$$G_2: S \rightarrow B \qquad B \rightarrow id \mid id\,(E) \qquad E \rightarrow B \mid E, B$$

a) Zeigen Sie *anhand der $LL(k)$-Definition*: für beliebiges $k \in \mathbb{N}$ gilt $G_1 \notin LL(k)$. / 3

b) Begründen Sie möglichst kurz: $G_1 \notin LR(0)$, $G_2 \notin LL(1)$. / 2

c) Geben Sie zu G_2 die $LR(0)$-Mengen und eine $SLR(1)$-Analysetabelle an. / 9

Aufgabe 5: Bottom-Up-Analyseautomat

10 Punkte

a) Beschreiben Sie die Arbeitsweise eines Bottom-Up-Analyseautomaten:
Nennen Sie die Konfigurationsmenge, Start- und Zielkonfiguration für die Analyse sowie die Arten von Konfigurationsübergängen, die der Automat ausführt. / 4

b) Für eine $LR(0)$-Grammatik arbeitet der Automat bekanntlich ohne jeden Lookahead *deterministisch*.
Eine Sprache L heißt "präfixfrei", falls gilt: $\forall w \in L, k < |w| : first_k(w) \notin L$. / 4
Zeigen Sie mit Hilfe eines LR(0)-Analyseautomaten:
Ist eine Sprache L in $\mathcal{L}(LR(0))$, so ist sie präfixfrei.

c) Beweisen oder widerlegen Sie: Reg $\subseteq \mathcal{L}(LR(0))$. / 2

Aufgabe 6: Attributgrammatik für Zuweisungen

10 Punkte

Die folgende Grammatik beschreibt eine Liste von Zuweisungen an Bezeichner *id* von booleschem oder arithmetischem Typ. Dabei bezeichne *bval* einen Wahrheitswert und *numval* eine Zahl.

$$
\begin{array}{lcll}
G_z: Z & \rightarrow & id \texttt{ := } E & (1)\\
E & \rightarrow & bval & (2)\\
E & \rightarrow & numval & (3)
\end{array}
\qquad
\begin{array}{lcll}
E & \rightarrow & E + E & (4)\\
E & \rightarrow & E \,||\, E & (5)\\
E & \rightarrow & E \,\&\&\, E & (6)\\
E & \rightarrow & E = E & (7)
\end{array}
$$

a) Nur Zahlen dürfen mit +, nur Wahrheitswerte mit && und || verknüpft werden. Der Vergleich = ist nur für Argumente *gleichen* Typs definiert. Versehen Sie das Nonterminal E mit Attributen $t \in$ {Bool, Num} für den Typ des Ausdrucks und $err \in$ {true, false} als Typfehlerindikator. / 4

b) Mit den nebenstehenden Produktionen erzeugt eine erweiterte Grammatik G'_z eine ganze Liste von Zuweisungen. Auch Bezeichner können nun auf der rechten Seite verwendet werden.

$$
\begin{array}{lcll}
G'_z: Z & \rightarrow & id \texttt{ := } E \texttt{ ; } Z & (8)\\
E & \rightarrow & id & (9)
\end{array}
$$

Erweitern Sie die Attributierung, so dass die gesamte Liste von Zuweisungen überprüft wird. Bezeichner dürfen nur auf der rechten Seite enthalten sein, falls ihnen bereits vorher ein Wert zugewiesen wurde. Die Verwendung muss typkonform erfolgen (es gilt jeweils die letzte Zuweisung). / 6

Speichern Sie die Typinformationen in einer Hashtabelle mit den nebenstehenden Operationen. Verwenden Sie eine initiale Tabelle als inherites Attribut des Startsymbols Z (Parameter der Attributierung).

```
data Table = .. -- Hashtable
insert :: Table -> (key,value) -> Table
delete :: Table -> key -> Table
lookup :: Table -> key -> Maybe value
```

Aufgabe 7: (MP- und MPP-)Zwischencode

13 Punkte

a) Gegeben sei der folgende **Kellerinhalt der MPP-Maschine** (Kellerspitze links)

Adresse:	12		15		18		21		24
Inhalt:	1 : 3	: 17 : 41 : 17		: 5 : 22 : 3		: 22 : 5 : 12		: 0 : 0 : 0	

```
01: PUSH FP
 ...
12: PUSH <FP-1>
13: LOAD R,<FP-2>
14: POP <R+4>
 ...
18: LOAD SP,FP
19: POP FP
 ...
```

sowie die **Register** $SP = 11, FP = 13, IC = 12, R = 0$.
Geben Sie Stack und Registerbelegung für die Abarbeitung der nächsten drei Befehle im nebenstehenden Codefragment an. Welche PSPP-Anweisung könnte dieser Ausführung zugrunde liegen? / 4

b) Befehle 18 und 19 gehören zum Exit-Code der Prozedur, die auch 12-14 enthält. Geben Sie einen dritten Befehl an, der den Rücksprung zum Aufrufer komplettieren könnte. / 1

Übersetzen Sie das Programm:

```
in/out X,Y;
proc F(); // in a) ohne Klammern
  X:=Y;
F();
```

c) als PSP-Programm in MP-Code / 3

d) als PSPP-Programm in MPP-Code / 5