Aufgabe 1 (Lex-Scanner) *16 Punkte*

Schreiben Sie eine Lex-Spezifikation, die eine Eingabezeichenfolge nach Rechneradressen, E-Mailadressen und Bankverbindungen durchsucht. Der Scanner soll die gefundenen Muster ausgeben. Eine genauere Definition der Muster wird im folgenden beschrieben.

Eine Rechneraddresse soll dabei entweder eine reine IP-Adresse, z.B. 192.176.17.6 oder eine mit Host bzw. Domain-Namen definierte Adresse, z.B. `www.fernuni-hagen.de`, sein. Eine IP-Adresse besteht aus vier durch Punkte abgetrennten Zahlen im Bereich von 0-255. Die Zahlen können durch führende Nullen auf bis zu drei Ziffern aufgefüllt werden. Eine Domain kann im Prinzip beliebig viele Subdomains (mindestens zwei), die jeweils durch Punkte abgetrennt werden, enthalten.

In Domain-Namen sind nur alphanumerische Zeichen und das Minuszeichen (aber nicht am Anfang) erlaubt. Der Namensraum von Toplevel-Domains, z.B. „de“, ist sehr stark eingeschränkt. Um nicht alle aufzuführen, verlangen wir einfach, daß diese nicht mit einer Zahl beginnen dürfen.

E-Mailadressen haben die Form `<name>@<domain>`, wobei die durch spitze Klammern eingeschlossenen Werte noch näher zu spezifizieren sind. Statt @ kann auch „(at)“ stehen und vor bzw. nach den Zeichen @ kann beliebiger Leerraum (Leerzeichen, Tabulatoren, Zeilenumbrüche) stehen. Im Gegensatz zu Rechneradressen kann hier eine Domain auch keine weiteren Subdomains enthalten.

Eine Bankverbindung beginnt mit „Bank“, oder „BLZ“ gefolgt von irgendwelchen Trennsymbolen (jedes beliebige Zeichen außer Ziffern), danach wird eine Zahlenfolge erwartet. Nun können weitere Trennsymbole folgen. Schließlich erwartet man ein ähnliches Muster diesmal jedoch beginnend mit den Schlüsselwörtern „Kto“ bzw. „Konto“. Bei den Schlüsselwörtern soll nicht zwischen Groß- und Kleinbuchstaben unterschieden werden.

Beispiele zu erkennender Muster:

```
a.bc.de, feuerwehr.112.de, 132.0.89.1, 145.176.6.007
support@de.minisoft.com, maus (at) wdr.de, root@localhost
BlZ: 44028774 / Konto: 23987342, BLZ = 440100100 | Kto =
7733444
```

Aufgabe 2 (Top-Down-Parser) *20 Punkte*

Gegeben sei die Grammatik $G = (N, \Sigma, P, S)$ mit

$N = \{S, A, B, C, D\}$

$\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}\}$

$P = \{ S \rightarrow ABC,$
$A \rightarrow \mathbf{a}A \mid \varepsilon,$
$B \rightarrow \mathbf{b} \mid \varepsilon,$
$C \rightarrow D \mid \mathbf{c}A,$
$D \rightarrow \mathbf{d}D \mid B\mathbf{e}$
$\}$

Für diese Grammatik soll ein LL(1) Parser entwickelt werden.

(a) Berechnen Sie die initialen Steuermengen der einzelnen Produktionen. *6 Punkte*

(b) Berechnen Sie die FOLLOW-Mengen aller Nichtterminale. Geben Sie dazu den verwendeten Graphen vor und nach Propagation der Knotenmarkierungen an. *8 Punkte*

(c) Geben Sie die Steuermengen der Produktionen aus G an. Handelt es sich bei G um eine LL(1)-Grammatik? Begründen Sie Ihre Antwort. *6 Punkte*

Aufgabe 3 (Operator-Vorrang-Analyse) *16 Punkte*

Gegeben sei eine Grammatik für Ausdrücke von Mengenoperationen. Es gibt die Operationen $\cup$ (Vereinigung), $\cap$ (Schnitt), $\setminus$ (Differenz) und $\times$ (kartesisches Produkt).

Neben diesen Operationen ist es möglich, Ausdrücke durch Klammern zu verschachteln und Mengen entweder durch Bezeichner, die durch das Token **id** repräsentiert werden, oder durch eine Liste von Elementen, z.B. { a b c d }, anzugeben. Elemente werden durch das Token **elem** dargestellt. Die Grammatik ist folgendermaßen definiert:

$G = \{ \{E, F\},$
$\{\mathbf{id}, \mathbf{elem}, \cap, \cup, \times, \setminus, (,), \{, \}\},$
$\{E \rightarrow (E) \mid E \cap E \mid E \cup E \mid E \times E \mid E \setminus E \mid \mathbf{id} \mid \{F\},$
$F \rightarrow F\ \mathbf{elem} \mid \mathbf{elem}\},$
$E \}$

Für die Grammatik soll ein Shift-Reduce Parser konstruiert werden. Definieren Sie eine Operator-Vorrang-Tabelle und parsen Sie den Ausdruck

(id \ id \ id ∪ id) × { elem elem }

Dabei hat die Differenz die höchste Priorität und soll als einzige Operation rechtsassoziativ sein. Die Operationen Vereinigung und Schnitt binden stärker als das kartesische Produkt.

Aufgabe 4 (Attributierte Grammatiken) *12 Punkte*

Gegeben seien die folgenden Produktionen einer Grammatik G für die Darstellung von Uhrzeiten in deutscher Sprache:

Zeit	→	*S* **Uhr** *M* **und** *K* **Sekunden**
S	→	**null** | **ein** | ... | **dreiundzwanzig**
M	→	ε
		| **eins** | **zwei** | ... | **neunundfünfzig**
K	→	ε
		| **eine** | **zwei** | ... | **neunundfünfzig**

(a) Erweitern Sie die Produktionen von G derart, daß die an einem Tag vergangenen Sekunden als Zahl ausgedrückt werden. Handelt es sich um synthetisierte oder vererbte Attribute?

(b) Geben Sie den Datenflußgraphen für das folgende Beispiel an:

achtzehn Uhr sechsunddreißig und fünf Sekunden

Aufgabe 5 (Zwischensprachen) *16 Punkte*

Gegeben sei ein Programmstück in einer Pascal-ähnlichen Sprache:

```
(1)  x := (a + b) * (a + b);
(2)  y := (a - b) * (a - b);
(3)  z := (a + b) * (a - b);
(4)  m := x / z + y / z;
```

(a) Geben Sie den DAG für diese Zuweisungsfolge an. *5 Punkte*

(b) Übersetzen Sie das Programmstück in Postfix-Notation und geben Sie an, wie der Code mit einer passenden Stackmaschine ausgewertet wird, geben Sie also für jeden Schritt den Befehl und den Stackinhalt an. *7 Punkte*

(c) Übersetzen Sie das Programmstück in 3AC. *4 Punkte*

Aufgabe 6 (Optimierung) *20 Punkte*

Das folgende Programmstück im 3-Adreß-Code soll für eine spätere Optimierung analysiert werden.

```
(5)  max := size
(6)  i := 1
(7)  if i > max goto 9
(8)  T1 := i
(9)  T2 := T1 * 4
(10) x[T2] := 1
(11) i := i + 1
(12) goto 3
(13) T3 := 1
(14) T4 := T3 * 4
(15) x[T4] := 0
(16) k := 2
(17) if k > max goto 29
(18) T5 := k
(19) T6 := 4 * T5
(20) T7 := x[T6]
(21) if t7=0 goto 27
(22) T8 := k
(23) T9 := 2 * T8
(24) j:= T9
(25) if j>max goto 27
(26) T10 := j
(27) T11 := 4 * T11
(28) x[T11] := 0
(29) j := j + k
(30) goto 21
(31) k := k + 1
(32) goto 13
(33) return
```

(a) Markieren Sie die Basisblöcke des Programms und numerieren Sie diese von oben nach unten, beginnend mit 1, durch. Begründen Sie, warum gerade an dieser Stelle ein Basisblock beginnt. *5 Punkte*

(b) Geben Sie den Flußgraphen für das Programm an. *5 Punkte*

(c) Berechnen Sie die *gen*-, *kill*-, *in*- und *out*- Mengen der UD-Verkettung für das Programm, wobei nur die Variablen `i`, `j`, `k` und `max` betrachtet werden sollen. *10 Punkte*