Prüfung aus Übersetzerbau 19.6.1998 Musterlösung

1. 25 % Quadrupel-Code

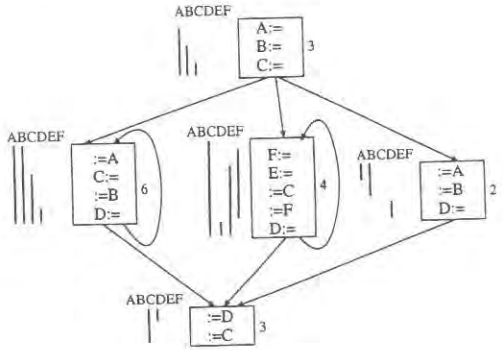
Erzeugen Sie für das rechte Programmstück Quadrupel-Code nach der Kontrollflußmethode. Ein INTEGER ist 4 Byte groß. Die Untergrenze aller Indexbereiche ist 0.

```
VAR
  a: ARRAY[30,10] OF INTEGER;
  b: ARRAY[20] OF INTEGER;
  i,j,s: INTEGER;
:
WHILE J<20 DO
  IF (i>=15) AND (j<10) THEN
    s := s + a[i,j] * b[j];
  ELSE
    s := s + b[j+10];
  END
j := j + 1;
END
```

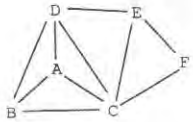
```
Wbegin:
    if j<20 goto Wtrue
    goto Wfalse
Wtrue:
    if i>=15 goto ANDtrue
    goto IFfalse
ANDtrue:
    if j<10 goto IFtrue
    goto IFfalse
IFtrue:
    t1 := i * 10
    t2 := t1 + j
    t3 := t2 * 4
    t4 := t3 + adr(a)
    t5 := @t4
    t6 := j * 4
    t7 := t6 + adr(b)
    t8 := @t7
    t9 := t5 * t8
    t10 := t9 + s
    s := t10
    goto IFend
IFfalse:
    t11 := j + 10
    t12 := t11 * 4
    t13 := t12 + adr(b)
    t14 := @t13
    t15 := t14 + s
    s := t15
IFend:
    t16 := j + 1
    j := t16
    goto Wbegin
Wfalse:
```


2. 25 %

Gegeben sei der folgende Kontrollflußgraph. Links von den Blöcken sind die Aktivitätsbereiche der Pseudoregister angegeben, rechts davon die erwarteten Ausführungshäufigkeiten.



a) (15 %) Geben Sie den Konfliktgraphen und die **Auslagerungskosten** für alle Pseudoregister an - dabei soll angenommen werden, daß ein Speicherbefehl einen Zyklus und ein Ladebefehl zwei Zyklen kosten soll.



	R:=	:=R	Kosten
A	3	8	19
B	3	8	19
C	9	7	23
D	12	3	18
E	4	0	4
F	4	4	12

$Kosten = (\mathtt{R}:=) + (:=\mathtt{R}) * 2$

b) (10 %) Bestimmen Sie eine Reihenfolge der **Registerbelegung**, belegen Sie die Pseudoregister mit realen Registern und kennzeichnen Sie die auszulagernden Pseudoregister. Nehmen Sie an, daß 3 reale Register zur Verfügung stehen.

Registerbelegung
Reihenfolge von links nach rechts

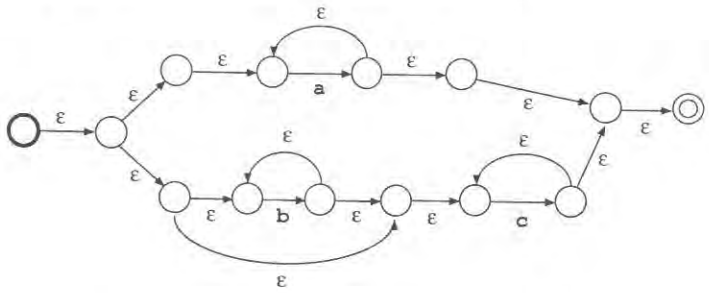
A	B	C	D	E	F
1	2	3	*	2	1

Das Register D muß ausgelagert werden weil seine Priorität mit $\frac{18}{3}$ am niedrigsten ist (A: $\frac{19}{3}$, B: $\frac{19}{3}$, C: $\frac{23}{3}$).
Anmerkung: Die obige Reihenfolge der Registerbelegung ist nur eine von mehreren möglichen Lösungen.

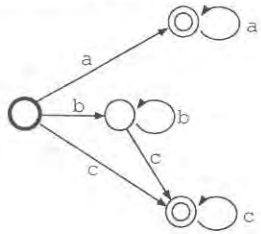
3. 25 % NFA und DFA

Gegeben sei der reguläre Ausdruck `a+|(b*c+)`

a) (10 %) Formen Sie den regulären Ausdruck in einen äquivalenten **NFA** mit ε - Kanten um. Verwenden Sie dazu die Umformungsregeln aus Kapitel 3 des Skriptums. (Eine Lösung ohne Verwendung des Algorithmus ist ungültig.)



b) (15 %) Erzeugen Sie aus dem NFA einen entsprechenden **DFA** und **Minimieren** Sie den entstandenen **DFA**, falls dies möglich ist.



4. 25 % Attributierte Grammatik

Ein Filesystem `FS` besteht aus einer Liste von Harddisks. Es gibt mindestens eine Harddisk. Nach dem Namen `name` einer Harddisk `HD` steht die Clustergröße `num` in Bytes, ein `":"` und danach die Directories. Jede Harddisk hat mindestens ein Directory. Ein Directory `D` hat einen Namen `name` und danach steht die Anzahl der Cluster `num` die es belegt. Die Directories sind durch `","` und die Harddisks durch `";"` getrennt, z.B.:

```
alpha 2048: bin 10, etc 7; beta 4096: ftp 20, pub 12; gamma 8192: www 35
```

a) (7 %) Erstellen Sie eine Grammatik die ein ganzes Filesystems einlesen kann. (Verwenden Sie dazu die Terminalsymbole `"name"`, `"num"`, `":"`, `","` und `";"` und ignorieren Sie etwaige Leerzeichen.)

FS	→	HDL
HDL	→	HDL `";"` `name num` `":"` DL
HDL	→	`name num` `":"` DL
DL	→	DL `","` `name num`
DL	→	`name num`

b) (18 %) Erweitern Sie die Grammatik aus a) um Attribute zur Berechnung des verbrauchten Platzes. Die Ausgabe soll so wie das Eingabeformat sein und im synthetisierten Attribut `"FS.x"` erfolgen. Alle Größen sind in Byte anzugeben. Im Attribut `"name.x"` steht der Name und in `"num.val"` der Zahlenwert des entsprechenden Terminalsymbols. Der Operator `"|"` hängt Zeichen- und Zahlen intelligent aneinander (`"gnu"|123` ergibt `"gnu 123"`). (Ignorieren Sie auch hier etwaige Leerzeichen.) Das obige Beispiel ergibt:

```
alpha 34816: bin 20480, etc 14336; beta 131072: ftp 81920, pub 49152; gamma
286720: www 286720
```

FS	→	HDL	FS.x := HDL.x
HDL	→	HDL `";"` `name num` `":"` DL	HDL0.x := HDL1.x | `";"` | `name`.x | DL.size | `":"` | DL.x DL.cg := `num`.val
HDL	→	`name num` `":"` DL	HDL.x := `name`.x | DL.size | `":"` | DL.x DL.cg := `num`.val
DL	→	DL `","` `name num`	DL0.x := DL1.x | `","` | `name`.x | `num`.val * DL0.cg DL0.size := DL1.size + `num`.val * DL0.cg
DL	→	`name num`	DL.x := `name`.x | `num`.val * DL.cg DL.size := `num`.val * DL.cg

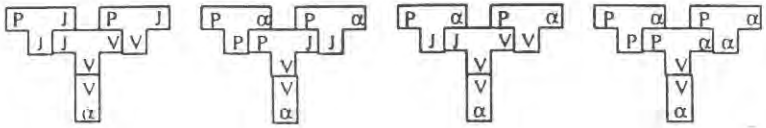
Musterlösung der
Prüfung aus Übersetzerbau vom 14.03.1997

1. 20 %

Sie verfügen über einen in der JavaVM vorliegenden Compiler, der Java auf die JavaVM übersetzt, einen JavaVM Interpreter für den Alpha-Prozessor, einen in Java geschriebenen Compiler, der Pizza auf Java übersetzt, und einen in Pizza geschriebenen Compiler, der Pizza auf Alpha-Maschinencode übersetzen kann.

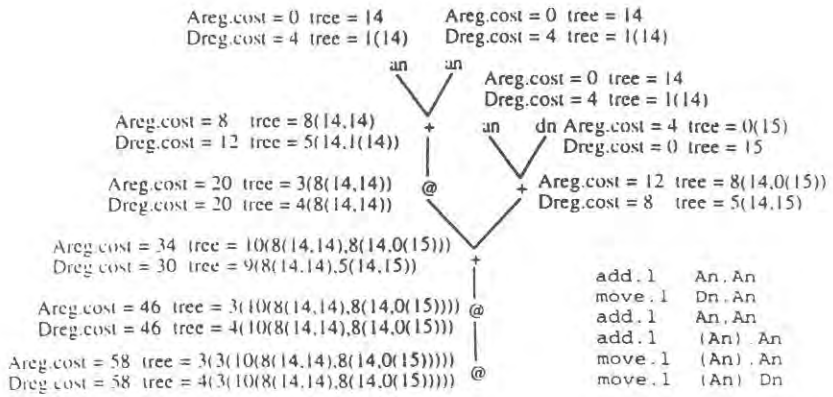
a) (10 %) Stellen Sie die beschriebenen Compiler und Interpreter mit T- und I-Diagrammen dar.

b) (10 %) Erzeugen Sie mit den vorhandenen Programmen einen Pizza-Compiler, der auf dem Alpha-Prozessor direkt lauffähig ist und ausführbare Programme dafür erzeugt.



4. 25 %

Erzeugen Sie für folgenden Zwischencode-Baum optimalen Maschinencode für den 68000, wobei das Ergebnis in einem Datenregister (Dreg) landen soll. Verwenden Sie dazu die im Skriptum angegebene Befehlsauswahl-Baumgrammatik. (Optimaler Code hat in erster Linie die geringsten Kosten, in zweiter Linie die geringste Anzahl Befehle).



2. 35%

Ein Punkt wird durch die in runde Klammern eingeschlossene und durch einen Beistrich getrennte x- und y-Koordinate dargestellt. Ein Linienzug wird durch den Buchstaben L und eine Folge von mindestens zwei Punkten dargestellt. Ein Graph wird durch den Buchstaben G und eine Folge von mindestens einem Linienzug dargestellt. Es können beliebig viele Graphen beschrieben werden.

Beispiel: G L (0,0) (3,0) (3,3) (0,3) L (0,3) (0,0) G L (1,1) (1,3) (2,3) (1,1)

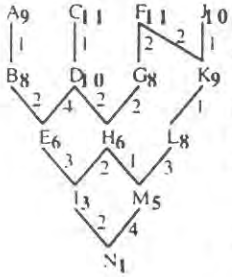
a) (10 %) Entwerfen Sie eine Grammatik, die beliebig viele Graphen verarbeiten kann.

Gs →	"G" Ls Gs	| "G" Ls
Ls →	"L" P Ps Ls	| "L" P Ps
Ps →	P Ps	| P
P →	"(" num "," num ")"	

b) (25 %) Erweitern Sie die Grammatik aus a) zu einer attributierten Grammatik, die die durchschnittliche Linienlänge eines Graphen berechnet und in einem Attribut des Startsymbols zurückliefert. (Die Länge einer Linie ergibt sich aus der Wurzel der Summe der Quadrate der Differenzen der x- bzw. y-Koordinaten des Start- und Endpunktes. Die Linienlänge eines Graphen ist die Summe der Längen aller Linien eines Graphen.)

S →	Gs	{S.r = Gs.s / Gs.n;}
Gs →	"G" Ls Gs_1	{Gs.n = 1 + Gs_1.n; Gs.s = Ls.s + Gs_1.s;} |
	"G" Ls	{Gs.n = 1; Gs.s = Ls.s;}
Ls →	"L" P Ps Ls_1	{Ps.x = P.x; Ps.y = P.y; Ls.s = Ps.s + Ls_1.s;} |
	"L" P Ps	{Ps.x = P.x; Ps.y = P.y; Ls.s = Ps.s;}
Ps →	P Ps_1	{Ps_1.x = P.x; Ps_1.y = P.y; Ps.s = Ps_1.s + √((P.x - Ps.x)² + (P.y - Ps.y)²);} |
	P	{Ps.s = √((P.x - Ps.x)² + (P.y - Ps.y)²);}
P →	"(" num_1 "," num_2 ")"	{P.x = num_1.val; P.y = num_2.val;}

3. 20%



Gegeben sei der folgende Datenabhängigkeitsgraph für die Befehle A bis N. Im Datenabhängigkeits-graphen sind die Kantenlängen bereits eingetragen. Bestimmen sie die Pfadlängen im Datenabhängigkeitsgraphen und geben Sie die optimierte Befehlsanordnung für die Befehle A bis N an, die Sie mittels *list scheduling* erhalten. Keiner der Befehle ist ein Sprungbefehl.

Lösung: F C D J A K G B L E H M I N

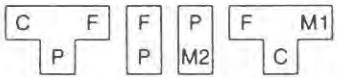
PRÜFUNGSORDNER - ein Service Deiner Fachschaft Informatik!
LVA: Compilerbau (Übersetzerbau) - VO - TU Preis:

Prüfung aus Übersetzerbau 08.11.1996
Musterlösung

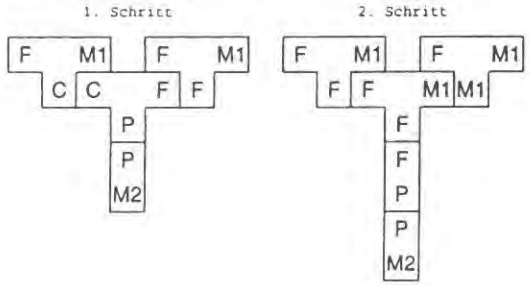
1. 20 %

Sie verfügen über einen in Prolog geschriebenen Compiler, der C in Fortran übersetzen kann, einen in Prolog geschriebenen Fortran-Interpreter, einen auf einer Maschine M2 lauffähigen Prolog-Interpreter, sowie einen in C geschriebenen Compiler, der Fortran in auf einer Maschine M1 ausführbare Programme übersetzen kann.

a) (5 %) Stellen Sie die beschriebenen Compiler und Interpreter mit T- und I-Diagrammen dar



b) (15 %) Erzeugen Sie mit den vorhandenen Programmen einen Fortran-Compiler der auf der Maschine M1 direkt lauffähig ist und ausführbare Programme dafür erzeugt.



2. 25 % Quadrupel-Code

```
VAR
  a: ARRAY[8,7] OF INTEGER;
  b: ARRAY[10] OF INTEGER;
  i,j: INTEGER;
:
WHILE i<j OR b[i]>0 DO
  a[i,j] := b[i] + j;
  IF i = 10 THEN
    i := 0;
  ELSE
    i := i + 1;
  END
END
```

Erzeugen Sie für das obige Programmstück Quadrupel-Code nach der Kontrollflußmethode. Ein INTEGER ist 4 Byte groß. Die Untergrenze aller Indexbereiche ist 0.

```
Wbegin:
  if i < j goto Wtrue
  goto Ofalse
Ofalse:
  t1 := i * 4
  t2 := t1 + adr(b)
  t3 := @t2
  if t3 > 0 goto Wtrue
  goto Wfalse
Wtrue:
  t4 := i * 4
  t5 := t4 + adr(b)
  t6 := @t5
  t7 := t6 + j
  t8 := i * 7
  t9 := t8 + j
  t10 := t9 * 4
  t11 := t10 + adr(a)
  @t11 := t7
  if i = 10 goto Itrue
  goto Ifalse
Itrue:
  i := 0
  goto Iend
Ifalse:
  t12 := i + 1
  i := t12
Iend:
  goto Wbegin
Wfalse:
  /* THE END */
```

3. 25 %

Gegeben sei folgende Grammatik:

$E \rightarrow$ a A B
$A \rightarrow$ c B | d
$B \rightarrow A \mid$ (E) | ε

Bestimmen Sie die First- und Follow-Mengen aller Nonterminale.

	First	Follow
E	{a}	{$,)}
A	{c,d}	{(,),c,d,$}
B	{(,c,d,ε}	{(,),c,d,$}

4. 30 %

a) (10 %) Entwerfen Sie eine Grammatik zur Beschreibung von arithmetischen Ausdrücken, bei denen Teilausdrücke immer explizit geklammert sind, und die sowohl Zahlen (Terminalsymbol num) als auch Bezeichner enthalten können (Terminalsymbol id). Die Ausdrücke enthalten mindestens einen Operator. Doppelte Klammerung, Klammerung einzelner Bezeichner oder Zahlen, sowie der obersten Ebene sind nicht erlaubt. Erlaubte Operatoren sind + und *, das Startsymbol soll S heißen.

Beispiele:

- **erlaubt:** $a + b, 2 + a, (3 + 6) * b, (1 * a) + (3 + (b * 4))$
- **nicht erlaubt:** $(a + b), a * (b), 2 + ((d + c)), 3 + 4 + 5, a$

Hinweis: Sie benötigen Produktionen der Form $E \rightarrow F$ *op* F

$S \rightarrow E$
$E \rightarrow F + F \mid F * F$
$F \rightarrow$ id | num | (E)

b) (20 %) Erweitern Sie die Grammatik zu einer attributierten Grammatik, die im Startsymbol folgende 3 Attribute zurückliefert:

- S.const ist ein boolscher Wert, der true ist, wenn der Ausdruck konstant war.
- S.val ist der Wert des Ausdrucks, falls S.const true ist, sonst undefiniert.
- S.str ist der Ausdruck mit ausgewerteten konstanten Teilausdrücken.

Das Attribut num.val enthält den numerischen Wert von num, id.const ist true, wenn der Bezeichner id einen Wert besitzt und id.val enthält dann diesen Wert. Benutzen Sie den bekannten Stringverkettungsoperator ||.

Achtung: Werten Sie alle Teilausdrücke aus, die berechnet werden können.

Beispiel Eingabe: (a+3)*c

Der Wert von a ist 2, c besitzt keinen Wert (d.h. ist nicht konstant).

Ergebnis: S.const = false, S.str = "5*c"

S	$\rightarrow$	E	S.const := E.const; S.val := E.val; S.str := E.str

E	$\rightarrow$	F_1 + F_2	E.const := F_1.const and F_2.const;
			if E.const then E.val := F_1.val + F_2.val;
			else E.str := F_1.str || "+" || F_2.str;

E	$\rightarrow$	F_1 * F_2	E.const := F_1.const and F_2.const;
			if E.const then E.val := F_1.val * F_2.val;
			else E.str := F_1.str || "*" || F_2.str;

F	$\rightarrow$	id	F.const := id.const;
			if F.const then begin F.val := id.val; F.str := id.val; end;
			else F.str := id.name;

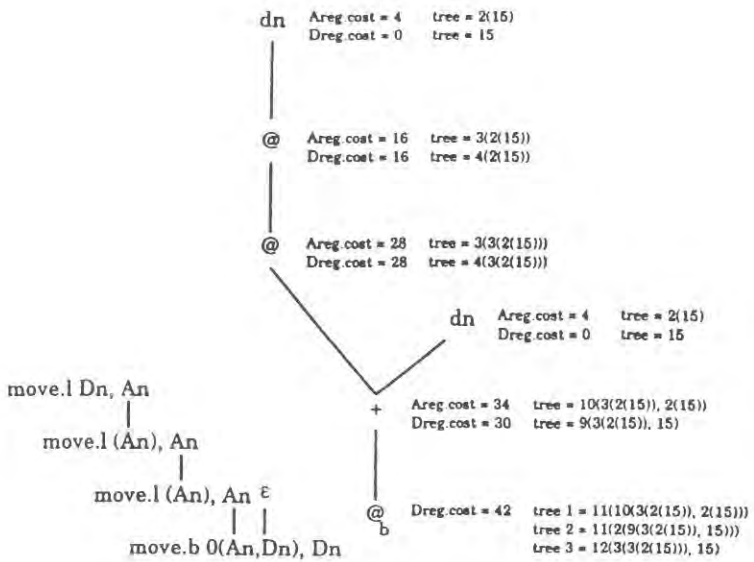
F	$\rightarrow$	num	F.const := true; F.val := num.val; F.str := num.val;
F	$\rightarrow$	(E)	F.const := E.const; if F.const then begin F.val := E.val; F.str := F.val; end; else F.str := "(" || E.str || ")";

(Anm.: Typ-Konversionen von Zahlen in Strings sind hier implizit)

Prüfung aus Übersetzerbau 21.6.1996
Musterlösung

1. 25 % Codeerzeugung

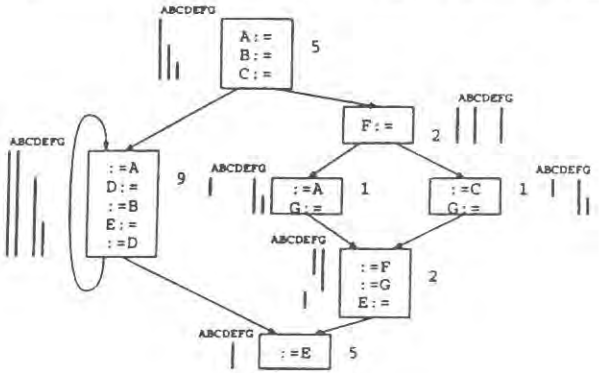
Erzeugen Sie für den folgenden Zwischencode-Baum optimalen Maschinencode für den 68000, wobei das Ergebnis in einem Datenregister landen soll. Verwenden Sie dazu die im Skriptum angegebene Befehlsauswahl-Baumgrammatik. (Optimaler Code hat in erster Linie die geringsten Kosten, in zweiter Linie die geringste Anzahl Befehle.)



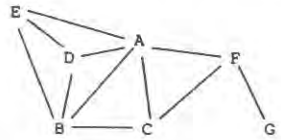
Die Bäume 1, 2 und 3 haben alle die gleichen Kosten, nämlich 42. Baum 3 beinhaltet jedoch nur vier Befehle im Gegensatz zu den anderen Bäumen mit je fünf Befehlen. Deshalb wird als optimale Codefolge Baum 3 ausgewählt.

2. 25 % Konfliktgraph und Auslagerungskosten

Gegeben sei der folgende Kontrollflußgraph. Neben den Blöcken sind die Aktivitätsbereiche der Pseudoregister und die erwarteten Ausführungshäufigkeiten angegeben.



a) (15 %) Geben Sie den Konfliktgraphen und die Auslagerungskosten für alle Pseudoregister an. Dabei soll angenommen werden, daß eine Zuweisung (R:=) einen Zyklus und eine Verwendung (:=R) zwei Zyklen kostet.



	R:=	:=R	Kosten
A	5	10	25
B	5	9	23
C	5	1	7
D	9	9	27
E	11	5	21
F	2	2	6
G	2	2	6

$Kosten = (\mathtt{R}:=) + (:=\mathtt{R}) * 2$

b) (10 %) Bestimmen Sie eine Reihenfolge der Registerbelegung. Belegen Sie die Pseudoregister mit realen Registern und kennzeichnen Sie die auszulagernden Pseudoregister. Nehmen Sie an, daß drei reale Register zur Verfügung stehen.

Registerbelegung
Reihenfolge von links nach rechts

A	B	D	E	C	F	G
1	2	3	*	3	2	1

Das Register E muß ausgelagert werden weil seine Priorität mit $\frac{21}{3}$ am niedrigsten ist (A: $\frac{25}{3}$, B: $\frac{23}{3}$, D: $\frac{27}{3}$).

Anmerkung: Die obige Reihenfolge der Registerbelegung ist nur eine von mehreren möglichen Lösungen.

3. 25 % Quadrupel-Code

```
VAR
  a: ARRAY[5,10,20] OF LONG;
  r: LONG;
  i,j,k: INTEGER;

WHILE (j<10) AND (NOT ((j>i) OR (j>k)) ) DO
  r := a[i,j,20-j];
  j := j + 1;
END
```

Erzeugen Sie für das obige Programmstück Quadrupel-Code nach der Kontrollflußmethode. Ein LONG ist 8 Byte und ein INTEGER 4 Byte groß. Die Untergrenze aller Indexbereiche ist 0.

```
Wbegin:
    if j<10 goto ANDtrue
    goto Wfalse
ANDtrue:
    if j>i goto Wfalse
    goto ORfalse
ORfalse:
    if j>k goto Wfalse
    goto Wtrue
Wtrue:
    t1 := i * 10
    t2 := t1 + j
    t3 := t2 * 20
    t4 := 20 - j
    t5 := t3 + t4
    t6 := t5 * 8
    t7 := t6 + adr(a)
    t8 := @t7
    r := t8
    t9 := j + 1
    j := t9
    goto Wbegin
Wfalse:
```


4. 25 % Attributierte Grammatik

In einem Supermarkt gibt es eine Maschine zur Flaschenrückgabe. Erweitern Sie die nachfolgende Grammatik um Attribute, um den Flaschen- bzw. Kisteneinsatz zu berechnen.

	→	
S	→	RL
RL	→	R RL | R
R	→	K FL | F
K	→	`milch` | `bier`
FL	→	F FL | F | ε
F	→	`milch` | `bier`

	Einsatz	
	Flasche	Kiste
Milch	4 S	30 S
Bier	5 S	50 S

Ein Element R der Rückgabeliste RL besteht aus einer Flasche F oder einer Kiste K mit Flaschen FL. In einer Kiste müssen *alle* Flaschen vom gleichen Typ sein (z.B. passen in Milchkisten nur Milchflaschen) sonst wird für die *ganze* Kiste kein Einsatz berechnet.

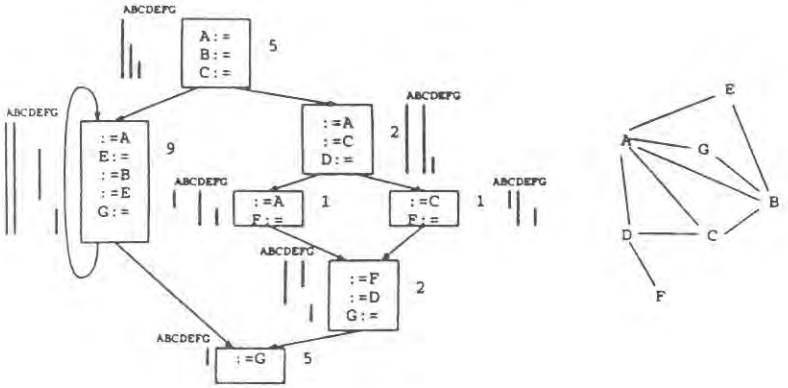
Die Anzahl der Flaschen, Kisten und fehlerhaften Flaschen (z.B. eine Bierflasche in einer Milchkiste) sowie der errechnete Einsatz der angenommenen Flaschen und Kisten sollen in den synthetisierten Attributen S.f, S.k, S.ff und S.e geliefert werden.

S	→	RL	S.f = RL.f; S.k = RL.k; S.ff = RL.ff; S.e = RL.e
RL	→	R RL_1	RL.f = R.f + RL_1.f; RL.k = R.k + RL_1.k RL.ff = R.ff + RL_1.ff; RL.e = R.e + RL_1.e
RL	→	R	RL.f = R.f; RL.k = R.k; RL.ff = R.ff; RL.e = R.e
R	→	K FL	FL.kt = K.kt if(FL.fehler) R.f = 0; R.k = 0; R.ff = FL.n; R.e = 0 else R.f = FL.n; R.k = 0; R.ff = 0; R.e = K.e + FL.e
R	→	F	R.f = 1; R.k = 0; R.ff = 0; R.e = F.e
K	→	`milch`	K.kt = milch; K.e = 30
K	→	`bier`	K.kt = bier; K.e = 50
FL	→	F FL_1	FL_1.kt = FL.kt FL.n = 1 + FL_1.n if(FL.kt == F.ft) FL.e = F.e + FL_1.e; FL.fehler = FL_1.fehler else Fl.e = 0; FL.fehler = true
FL	→	F	FL.n = 1 if(FL.kt == F.ft) FL.e = F.e; FL.fehler = false else FL.e = 0; FL.fehler = true
FL	→	ε	FL.n = 0 FL.e = 0 FL.fehler = false
F	→	`milch`	F.ft = milch; F.e = 4
F	→	`bier`	F.ft = bier; F.e = 5

Prüfung aus Übersetzerbau 15.3.1996 Musterlösung

1. 25 % Konfliktgraph und Auslagerungskosten

Gegeben sei der folgende Kontrollflußgraph. Die Ausführungshäufigkeiten stehen rechts neben den Blöcken. Geben Sie den *Konfliktgraphen* und die *Auslagerungskosten* für alle Pseudoregister an. Dabei wird angenommen, daß ein Speicherbefehl *zwei* Zyklen und ein Ladebefehl *drei* Zyklen kostet.



R	R:=	:=R	Summe
A	5	12	46
B	5	9	37
C	5	3	19
D	2	2	10
E	9	9	45
F	2	2	10
G	11	5	37

2. 20 % Reguläre Definition

Ein URL (Uniform Resource Locator) gibt die Adresse eines Objekts im World Wide Web an. Er besteht aus der Zugriffsart (`http`, `ftp`, `gopher`, `mailto`, `news`) gefolgt von einem Doppelpunkt ":" und einer genaueren Beschreibung des Objekts, abhängig von der Zugriffsart. Bei `http`, `ftp` und `gopher` folgen zwei Schrägstriche "`//`", ein *Hostname* und ein Zugriffspfad. Ein *Hostname* besteht entweder aus vier Zahlen mit maximal drei Ziffern, *getrennt* durch einen Punkt (z.B. 128.130.173.8) oder aus einer beliebigen Anzahl von Buchstabengruppen (Groß- und Kleinbuchstaben und "`-`") *getrennt* durch einen Punkt (der *Hostname* muß in diesem Fall mit einem Buchstaben beginnen, z.B. ftp.uni-paderborn.de). Ein Zugriffspfad ist vom *Hostname* durch einen Schrägstrich getrennt. Er besteht aus beliebig vielen (nicht leeren) Filenamen, die durch Schrägstriche getrennt sind und darf auch mit einem Schrägstrich enden. Ein Filename besteht aus beliebig vielen Groß- und Kleinbuchstaben, "`-`" und "`.`". Bei der Zugriffsart `news` folgt nach dem Doppelpunkt nur ein *Hostname*. Bei `mailto` folgt nach dem Doppelpunkt ein Name, ein "`@`" und ein *Hostname*. Ein Name besteht aus beliebig vielen Groß- und Kleinbuchstaben.

gültige URLs	ungültige URLs
`http://www.Info.ch/info.html`	`http://name@host.name`
`ftp://1.2.3.4/..A--B../`	`ftp:/123.4567.2.9873//file.txt`
`news:news.tuwien.ac.at`	`news://info.at/news`
`mailto:alex@complang.tuwien.ac.at`	`mailto:complang..tuwien.ac.at`

Erstellen Sie eine *reguläre Definition* für einen URL.

```
ziff  =  [0-9]
zahl  =  [0-9] ziff? ziff?
 bst  =  [a-z]|[A-Z]
host  =  ip|hn
  ip  =  zahl "." zahl "." zahl "." zahl
  hn  =  bst (bst|"-")* ("." (bst|"-")+)*
file  =  (bst|"-"|".")+
pfad  =  file ("/" file)* "/"?
 art  =  "http"|"ftp"|"gopher"
 URL  =  (art "://" host ("/" pfad)|("/"?) ) |
         ("news:" host) |
         ("mailto:" bst+ "@" host)
```

3. 25 % Quadrupel-Code

```
VAR
  a: ARRAY[4,20,40] OF LONG;
  b: ARRAY[60] OF INTEGER;
  i,j,k,f: INTEGER;

IF NOT((a[i,j,k] > 5) OR (i > j)) THEN
  f := b[j + k];
ELSE
  f := k + i + j;
END
k := k + 1;
```

Erzeugen Sie für das obige Programmstück Quadrupel-Code nach der Kontrollflußmethode. Ein `LONG` ist 8 Byte und ein `INTEGER` 4 Byte groß. Die Untergrenze aller Indexbereiche ist 0.

```
          t1 := i * 20
          t2 := t1 + j
          t3 := t2 * 40
          t4 := t3 + k
          t5 := t4 * 8
          t6 := t5 + adr(a)
          t7 := @t6
          if t7 > 5 goto IFfalse
          goto ORfalse
ORfalse:  if i>j goto IFfalse
          goto IFtrue
 IFtrue:  t8 := j + k
          t9 := t8 * 4
          t10 := t9 + adr(b)
          t11 := @t10
          f := t11
          goto end
IFfalse:  t12 := k + i
          t13 := t12 + j
          f := t13
    end:  t14 := k + 1
          k := t14
```


4. 30 % Attributierte Grammatik

In HTML (Hypertext Markup Language) gibt es Konstrukte, um verschachtelte Listen zu definieren. Das Kommando `<OL>` (ordered list) leitet eine numerierte Liste ein. `<UL>` (unordered list) eine Liste mit *. Ein Listeneintrag beginnt mit `<LI>` (list item) und eine ganze Liste wird mit `</OL>` bzw. `</UL>` beendet. Auf dem Bildschirm wird der Text entsprechend seiner logischen Struktur eingerückt dargestellt.

HTML	Bildschirm

`Das sind Listen:`
`<OL><LI> Text eins`
`<LI> Text zwei <UL>`
`<LI> erste Zeile`
`<LI> zweite Zeile`
`</UL>`
`<LI> Text drei`
`</OL>`
`Hier geht's weiter.`
	`Das sind Listen:`
	`1. Text eins`
	`2. Text zwei`
	`* erste Zeile`
	`* zweite Zeile`
	`3. Text drei`
	`Hier geht's weiter.`

Erweitern Sie die rechts gezeigte Grammatik um Attribute für die Darstellung von Listen in einem HTML-Text. Unterscheiden Sie numerierte von nicht numerierten *Items* (`Is`). Der Text soll im synthetisierten Attribut `S.x` geliefert werden. Die Funktion `tabs(n)` liefert `n` Tabulatoren. Ein *newline* wird durch `"\n"` dargestellt. Das Attribut `chars.x` enthält die Textstücke zwischen den Kommandos. Verwenden Sie den Operator ||, um Zeichenketten zusammenzuhängen.

S	→	Html	
Html	→	Text Html | Text	
Text	→	`chars` | List	
List	→	`<UL>` Is `</UL>` |	
		`<OL>` Is `</OL>`	
Is	→	Item Is | Item	
Item	→	`<LI>` Html	

S	→	Html	S.x := Html.x Html.i := 0 /* noch keine Einrückung */
Html	→	Text $Html_1$	Html.x := Text.x || "\n" || $Html_1$.x Text.i := Html.i; $Html_1$.i := Html.i
Html	→	Text	Html.x := Text.x; Text.i := Html.i
Text	→	`chars`	Text.x := `chars`.x
Text	→	List	Text.x := List.x; List.i := Text.i + 1 /* einrücken */
List	→	`<UL>` Is `</UL>`	Is.i := List.i; List.x := Is.x Is.n := 1; Is.o := false /* keine Numerierung */
List	→	`<OL>` Is `</OL>`	Is.i := List.i; List.x := Is.x Is.n := 1; Is.o := true /* mit Numerierung */
Is	→	Item Is_1	Item.i := Is.i; Is_1.i := Is.i Is_1.n := Is.n + 1 /* nächste Nummer */ Item.n := Is.n; Item.o := Is.o; Is_1.o := Is.o Is.x := Item.x || "\n" || Is_1.x
Is	→	Item	Item.i := Is.i; Item.n := Is.n Item.o := Is.o; Is.x := Item.x
Item	→	`<LI>` Html	Html.i := Item.i IF Item.o THEN Item.x := `tabs`(Item.i) || Item.n || ". " || Html.x ELSE Item.x := `tabs`(Item.i) || "* " || Html.x

Prüfung aus Übersetzerbau 10.3.1995 Musterlösung

1. 20 % Gegeben sei folgende LL(1)-Grammatik:

```
S     →  Stm R
R     →  ; Stm R | ε
Stm   →  if id then Stm Else endif | id
Else  →  else Stm | ε
```

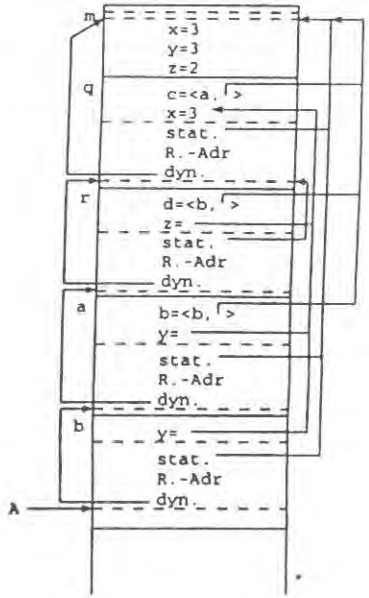
Erstellen Sie die Analysetabelle mit Fehleraktionen (`skip`, `stop`) für die tabellengesteuerte Top-Down-Analyse.

```
First(S) = { if, id }        Follow(S) = { $ }
First(R) = { ";", ε }        Follow(R) = { $ }
First(Stm) = { if, id }      Follow(Stm) = { ";", $, else, endif }
First(Else) = { else, ε }    Follow(Else) = { endif }
```

	";"	if	id	then	endif	else	$
S	skip	Stm R	Stm R	skip	skip	skip	stop
R	";"Stm R	ε(*)	ε(*)	ε(*)	ε(*)	ε(*)	ε
Stm	stop	if ...endif	id	skip	stop	stop	stop
Else	ε(*)	ε(*)	ε(*)	ε(*)	ε	else Stm	ε(*)

2. 25 % Gegeben sei ein Modula-2-Programm. Zeichnen Sie den Stack der Activation Records (mit lokalen Größen, Wert- und Variablenparametern, statischer und dynamischer Kette) zum Zeitpunkt (*here*) ohne Verwendung eines Displays.

```
MODULE m;
TYPE IP=PROCEDURE(VAR INTEGER);
TYPE IPP=PROCEDURE(IP, VAR INTEGER);
VAR x,y,z:INTEGER;

  PROCEDURE a(b:IP; VAR y:INTEGER);
  BEGIN
    b(y);
  END a;

  PROCEDURE b(VAR y:INTEGER);
  BEGIN
    z:=x+1; x:=y; y:=z+1;      (*here*)
  END p;

  PROCEDURE q(c:IPP; x:INTEGER);

    PROCEDURE r(d:IP; VAR z:INTEGER);
      BEGIN
        c(d,z);
      END r;

  BEGIN
    r(b,x);
  END q;
BEGIN
  x:=1;y:=3;z:=1;
  q(a,y);
END m.
```



3. 30 % In einer einfachen Sprache zur Beschreibung von Bildschirmmasken stehen die Ausgabeelemente `vbox`, `hbox` und `feld` zur Verfügung: `vbox` hat als Parameter eine Liste von Ausgabeelementen, die vertikal untereinander dargestellt werden sollen, `hbox` hat als Parameter eine Liste von Ausgabeelementen, die horizontal nebeneinander dargestellt werden sollen; `feld` ist ein nicht weiter zerlegbares Ausgabefeld.

Beispiel:

```
hbox(
   vbox(
      hbox(feld1,feld2),
      feld3,
      hbox(feld4,feld5) ),
   vbox(feld6,feld7) )
```

erzeugt folgende Ausgabe:

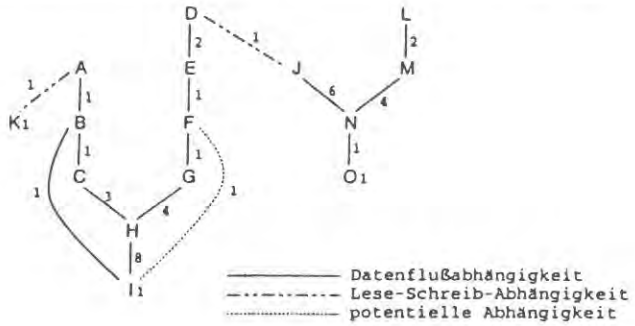
```
feld1 feld2   feld6
feld3         feld7
feld4 feld5
```

a) (10 %) Geben Sie eine Grammatik zur Beschreibung dieser Sprache an. Token sind die Symbole `vbox`, `hbox`, `feld`, `"("`, `")"` und `","`.

b) (20 %) Erweitern Sie die Grammatik zu einer Attributierten Grammatik, die für jedes Ausgabeelement die Anfangskoordinaten am Bildschirm errechnet und diese den Attributen `px` und `py` zuweist. Die Anfangsposition der gesamten Darstellung ist `(0,0)` (gibt die Koordinaten `x` und `y` an). Die Feldgröße von `feld` steht in den synthetisierten Attributen `sx` (horizontal) und `sy` (vertikal) zur Verfügung. Die vertikale Größe eines Elements `hbox` ergibt sich aus dem Maximum der vertikalen Größen seiner Parameter. Die horizontale Größe eines Elements `hbox` ergibt sich aus der Summe der horizontalen Größen seiner Parameter. Die vertikale Größe eines Elements `vbox` ergibt sich aus der Summe der vertikalen Größen seiner Parameter. Die horizontale Größe eines Elements `vbox` ergibt sich aus dem Maximum der horizontalen Größen seiner Parameter. Nehmen wir zum Beispiel an, daß jedes `feld` in obigem Beispiel eine vertikale Größe von 1 und eine horizontale von 3 besitzt, dann beginnt $\texttt{feld}_1$ an der Stelle `(0,0)`, $\texttt{feld}_2$ auf `(3,0)`, $\texttt{feld}_3$ auf `(0,1)`, $\texttt{feld}_4$ auf `(0,2)`, $\texttt{feld}_5$ auf `(3,2)`, $\texttt{feld}_6$ auf `(6,0)` und $\texttt{feld}_7$ auf `(6,1)`.

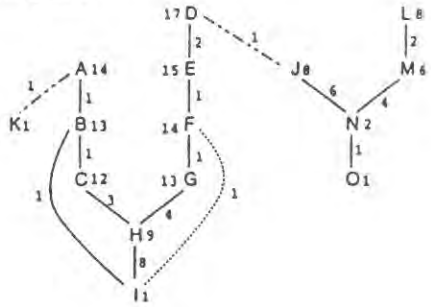
a)	b)
S → L	L.px:=0; L.py:=0; L.hflag:=true
L → hbox (L_1)	L_1.hflag:=true; L_1.px:=L.px; L_1.py:=L.py; L.sx:=L_1.sx; L.sy:=L_1.sy;
L → vbox (L_1)	L_1.hflag:=false; L_1.px:=L.px; L_1.py:=L.py; L.sx:=L_1.sx; L.sy:=L_1.sy;
L → L_1 "," L_2	L_1.px:=L.px; L_1.py:=L.py; IF L.hflag THEN {L.sy:=max(L_1.sy,L_2.sy); L.sx:=L_1.sx+L_2.sx; L_2.px:=L.px+L_1.sx; L_2.py:=L.py;} ELSE {L.sx:=max(L_1.sx,L_2.sx); L.sy:=L_1.sy+L_2.sy; L_2.py:=L.py+L_1.sy; L_2.px:=L.px;}
L → feld	feld.px:=L.px; feld.py:=L.py; L.sx:=feld.sx; L.sy:=feld.sy;

4. 25 % Gegeben sei der folgende Datenabhängigkeitsgraph für die Befehle `A` bis `O`. Im Datenabhängigkeitsgraphen sind die Kantenlängen bereits eingetragen.



Bestimmen sie die Pfadlängen im Datenabhängigkeitsgraphen und geben Sie die optimierte Befehlsanordnung für die Befehle `A` bis `O` an, die Sie mittels *list scheduling* erhalten. Keiner der Befehle sei ein Sprungbefehl.

Graph mit Pfadlängenangaben:



Optimierte Befehlsanordnung:

`D A E F B G C L J H M K N O I` (→ 18 statt 30 Zyklen)

Name	Kennzahl	Matrikelnummer

	Bitte tragen Sie sofort mit Kugelschreiber Name, Kennzahl und Matrikelnummer ein ! Halten Sie bitte den Studentenausweis bereit !
Summe	Lesen Sie die Angaben genau, bevor Sie beginnen ! Verwenden Sie bitte nur diesen Bogen für Ihre Antworten ! Die Prüfungsdauer beträgt 90 Minuten.
1.Bsp. 2.Bsp. 3.Bsp. 4.Bsp.	**Institut für Computersprachen (Prof. Brockhaus)**

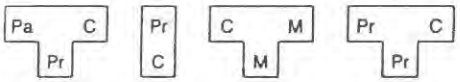
Ich trete zum ... mal zur Prüfung an.

Prüfung aus Übersetzerbau 23.6.1995

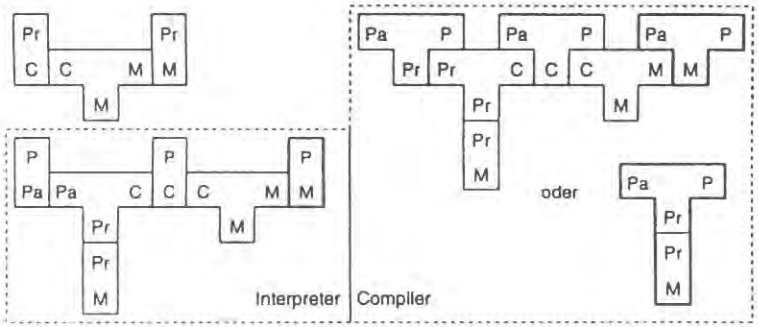
1. 20 %

Sie verfügen über einen in Prolog geschriebenen Pascal-Compiler, der C erzeugt, einen in C geschriebenen Prolog-Interpreter, einen C-Compiler für die Maschine M und einen in Prolog geschriebenen Prolog-Compiler, der C erzeugt.

a) (5 %) Stellen Sie die beschriebenen Compiler und Interpreter mit T- und I-Diagrammen dar.



b) (15 %) Sie haben einen in Prolog geschriebenen Pascal-Compiler, der P-Code erzeugt, und einen in Pascal geschriebenen P-Code Interpreter. Wie können diese mit Hilfe der vorhandenen Compiler und Interpreter übersetzt werden, so daß sie auf der Maschine M lauffähig sind? Stellen Sie die Vorgangsweise mittels T- und I-Diagrammen dar.



2. 30 % Gegeben sei folgende Grammatik, die aus verschachtelten indizierten Variablen aufgebaute Ausdrücke beschreibt:

$S \rightarrow E$
$E \rightarrow$ `id [` L `]` | `num`
$L \rightarrow L$ `","` $E \mid E$

Erweitern Sie diese oder eine äquivalente Grammatik zu einer attributierten Grammatik, die die Übersetzung der Ausdrücke in Quadrupelcode beschreibt, der den Wert des Ausdrucks berechnet. Der Quadrupelcode soll im synthetisierten Attribut *S*.code generiert werden. Das Attribut `num.val` enthält den numerischen Wert der Zahl, `id.sym` den Namen der Arrayvariable. Zur Bestimmung der Anzahl der Elemente der i-ten Dimension eines Arrays *sym* steht die Funktion `dim(`*sym*`,`*i*`)` zur Verfügung. Die Funktion `newtemp` liefert eine neu erzeugte Hilfsvariable zurück.

$S \rightarrow E$
```
S.code := E.c
S.v := E.v
```

$E \rightarrow$ `id [` L `]`
```
L.sym := id.sym
t0 := newtemp
t1 := newtemp
E.v := newtemp
E.c := L.c||gen(t0 ':=' L.v '*' 4)||
    gen(t1 ':=' t0 '+adr(' id.sym ')')||gen(E.v ':=' '@' t1)
```

$E \rightarrow$ `num`
```
E.v := num.val
E.c := ''
```

$L \rightarrow L_1$ `","` E
```
L1.sym := L.sym
L.i := L1.i + 1
t1 := newtemp
L.v := newtemp
L.c := L1.c||E.c||gen(t1 ':=' L1.v '*' dim(L.sym,L.i))||
    gen(L.v ':=' t1 '+' E.v)
```

$L \rightarrow E$
```
L.i := 1
L.v := E.v
L.c := E.c
```

Durch Umformung der Grammatik könnten die ererbten Attribute eliminiert werden.

23.6.95

3. 25 % Gegeben sei folgendes Modula-Programmstück:

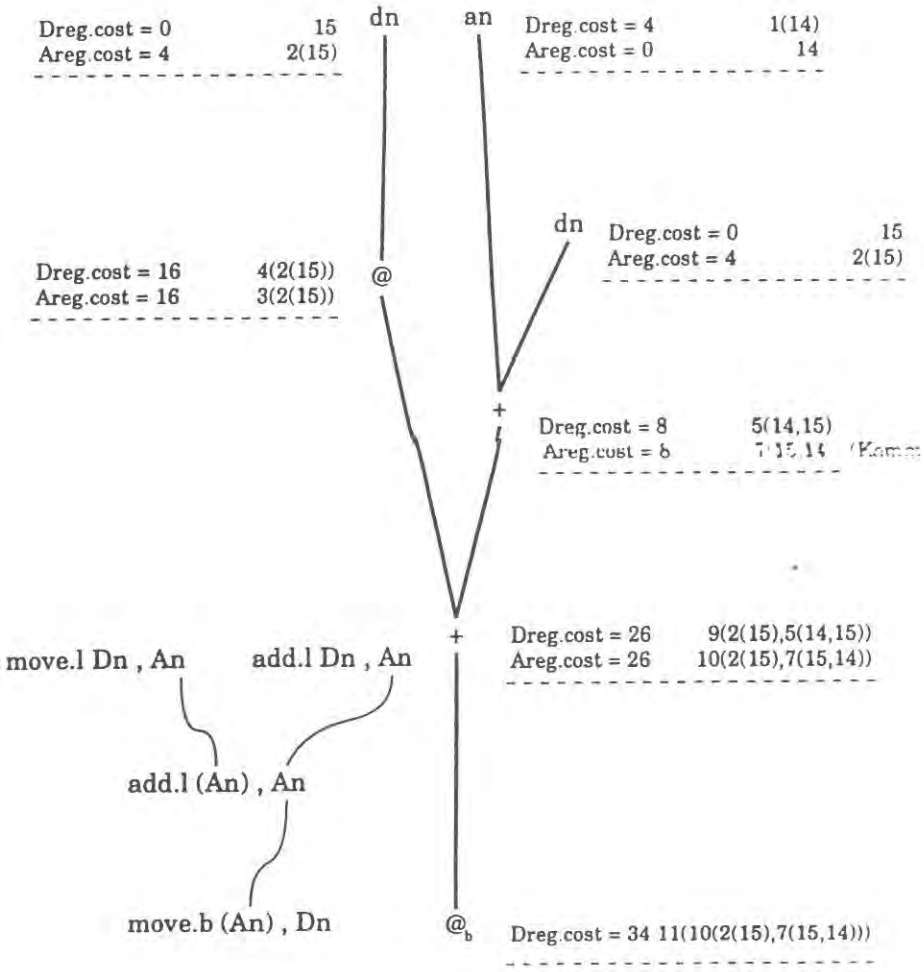
```
VAR
   a: ARRAY[0..N] OF INTEGER; b: ARRAY[1..M] OF INTEGER;
   n,j,k: INTEGER;

WHILE a[n+j] > k DO
   IF n < 8-j THEN
      n := n + 1;
   ELSE
      k := b[n];
   END
END;
```

Erzeugen Sie für das obige Programmstück Quadrupel-Code nach der Kontrollfluß-methode.

```
Start:
      t1 := n+j
      t2 := 4*t1
      t3 := t2+adr(a)
      t4 := @t3
      if t4>k goto L1
      goto Ende
L1:   t5 := 8-j (* In der WHILE-Schleife *)
      if n<t5 goto L2
      goto L3
L2:   t6 := n+1 (* THEN-Zweig *)
      n := t6
      goto L4
L3:   t7 := n-1 (* ELSE-Zweig *)
      t8 := t7*4
      t9 := t8+adr(b)
      t10 := @t9
      k := t10
L4:   goto Start
Ende:
```

4. 25 %

Erzeugen Sie für den folgenden Zwischencode-Baum optimalen Maschinencode für den 68000, so daß das Ergebnis in einem Datenregister landet. Verwenden Sie dazu die im Skriptum angegebene Befehlsauswahl-Baumgrammatik.



Gedankenprotokoll der Mündliche Prüfung am 30.April 2003
Prüfer: Andreas Krall

Struktur der Fragen: der Prüfer wollte die prinzipielle Funktionsweise, nicht aber den genauen Algorithmus wissen, dh. zb. Tabellengesteuerte Top Down Analyse: was steht in der Tabelle, wie wird sie verwendet, nicht aber der genaue Algorithmus zum befüllen der Tabelle
Die ganze Prüfung kam übrigens ohne Papier aus, dh. bei keiner Frage musste ein Algorithmus anhand eines praktischen Beispiels erklärt werden.

Was wissen sie über:

- Top Down Analyse?
  - Tabellengesteuertes Verfahren
  - Funktionsgesteuertes Verfahren
  - Was sind First / Follow Mengen?
  - Ist eine LL(k) Grammatik mächtiger als eine LL(1) Grammatik?
- Bottom Up Analyse?
  - Mächtigkeit der Grammatiken...
- Attributierte Grammatik
  - Wie funktioniert bzw. was ist ein Übersetzungsschema

Wie funktioniert

- Codeauswahl
  Kann eine optimale Lösung gefunden werden?
- Registerauswahl
- Befehlsanordnung / Optimierung
  Kann eine optimale Lösung gefunden werden?

Was ist ein bzw. Wie funktioniert ein

- Virtueller Methodenaufruf
- Activation Record
  In welchem Fall wird der statische Vorgänger nicht benötigt?
  Wann wird der dynamische Vorgänger nicht benötigt?