Aufgabe 1

(a)

Die Lex-Spezifikation ist sehr einfach. Für jedes Zeichen einer römischen Zahl wird sein numerischer Wert in der Kommunikationsvariablen *yylval* und das zugehörige Token zurückgegeben:

```
%%
[iI]  {yylval = 1; return(I);}
[vV]  {yylval = 5; return(V);}
[xX]  {yylval = 10; return(X);}
[lL]  {yylval = 50; return(L);}
[cC]  {yylval = 100; return(C);}
[dD]  {yylval = 500; return(D);}
[mM]  {yylval = 1000; return(M);}
\n    {return(yytext[0]);}
.     ;
```

Anzumerken bleibt, daß der Deklarationsteil und der Teil für Hilfsprozeduren weggelassen wurden, da sie nicht benötigt werden.

(b)

Die Yacc-Spezifikation kann wie folgt aussehen:

```
%{
int fuenf = 0, fuenfzig = 0, fuenfhundert = 0;
%}

%token I V X L C D M

%%
zahlen          : rom_zahl '\n'
                | zahlen {init();} rom_zahl '\n'
                ;
rom_zahl        : tausend {if (pruef()) printf(" ==> %d\n", $1);}
                | rest {if (pruef()) printf(" ==> %d\n",$1);}
                | tausend rest {if (pruef()) printf(" ==> %d\n", $1+$2);}
                ;
tausend         : M
                | tausend M {$$=$1+$2;}
                ;
rest            : hundert
                | zehner_einer
                | hundert zehner_einer {$$=$1+$2;}
                ;
zehner_einer    : einer
                | zehner
                | zehner_einer {$$=$1+$2;}
                ;
hundert         : hund_200_300
                | ein_5_10_50_100 D {fuenfhundert++;$$=$2-$1;}
                | D {fuenfhundert++;}
                | D hund_200_300 {fuenfhundert++;$$=$1+$2;}
                | ein_5_10_50_100 M {$$=$2-$1;}
```

```
                | D M {fuenfhundert++;$$=$2-$1;}
                ;
zehner          : zehn_20_30
                | ein_5_10 L{fuenfzig++;$$=$2-$1;}
                | L {fuenfzig++;}
                | L zehn_20_30 {fuenfzig++;$$=$1+$2;}
                | ein_5_10_50 C {$$=$2-$1;}
                ;
einer           : ein_2_3
                | I V {fuenf++;$$=$2-$1;}
                | V {fuenf++;}
                | V eins_2_3 {fuenf++;$$=$1+$2;}
                | I X {$$=$2-$1;}
                ;
eins_2_3        : I
                | I I {$$ ($$=$1+$2;}
                | I I I {$$=$1+$2+$3;}
                ;
zehn_20_30      : X
                | X X {$$=$1+$2;}
                | X X X {$$=$1+$2+$3;}
                ;
hund_200_300    : C
                | C C {$$=$1+$2;}
                | C C C {$$=$1+$2+$3;}
                ;
ein_5_10        : I
                | V {fuenf++;}
                | X
                ;
ein_5_10_50     : ein_5_10
                | L {fuenfzig++;}
                ;
ein_5_10_50_100 : ein_5_10_50
                | C
                ;


%%
#include "lex.yy.c"

void init()
{
  fuenf = fuenfzig = fuenfhundert = 0;
}

int pruef()
{
  if (fuenf > 1)
    {printf("Fehler: V nur einmal erlaubt!\n"); return 0;}
  else if (fuenfzig > 1)
    {printf("Fehler: L nur einmal erlaubt!\n"); return 0;}
  else if (fuenfhundert > 1)
    {printf("Fehler: D nur einmal erlaubt!\n"); return 0;}
  else
    return 1;
}
```

Einige Erläuterungen zur Lösung: Die Regel für *zahlen* beschreibt eine durch Zeilenendezeichen getrennte Folge von römischen Zahlen. Zu beachten ist, daß zwischendurch die drei Variablen *fuenf*, *fuenfzig* und *fuenfhundert*, die die Häufigkeit des Auftretens ihrer entsprechenden Zeichen festhalten, neu initialisiert werden müssen (Funktion *init*()). Die Regeln für *rom_zahl*, *tausend*, *rest* und *zehner_einer* sollten leicht verständlich sein. Bezüglich der Regel *hundert*, die die Hunderter innerhalb der römischen Zahl behandelt, können verschiedene Fälle auftreten. Entweder gibt es (a) ein, zwei oder drei C nebeneinander, (b) ein Zeichen aus {I, V, X, L, C} vor einem D, (c) nur ein D, (d) ein D gefolgt von einem, zwei oder drei C oder (e) ein Zeichen aus {I, V, X, L, C, D} vor einem M. Die Regeln für *zehner* und *einer* sind analog aufgebaut. Bei den gerade bechriebenen letzten drei Regeln ist insbesondere R6 zu beachten. Die restlichen Regeln sollten leicht verständlich sein. Die Funktion *prueef*() prüft die Einhaltung von R3 und gibt bei einer Verletzung eine entsprechende Meldung aus.

Aufgabe 2

(a)

Wir vereinfachen G zu $G' = (N', \Sigma', P', E)$ mit

$$P' = \{\ E \rightarrow (E) \mid E+E \mid E*E \mid E \uparrow E \mid \mathbf{id}\ \}.$$

Die Tabelle zur Operator-Vorrang-Analyse hat dann folgendes Aussehen:

		Eingabe						
		+	*	↑	(	)	**id**	$
Stack	+	·>	<·	<·	<·	·>	<·	·>
	*	·>	·>	<·	<·	·>	<·	·>
	↑	·>	·>	<·	<·	·>	<·	·>
	(	<·	<·	<·	<·	≐	<·	
	)	·>	·>	·>		·>		·>
	id	·>	·>	·>		·>		·>
	$	<·	<·	<·	<·		<·	

(b)

Stack		*Eingabe*	*Aktion*
\$	$\lessdot$	**id+id↑(id*id)**\$	*shift*
\$$\lessdot$**id**	$\gtrdot$	+**id↑(id*id)**\$	*reduce* mit $E \to$ **id**
\$$E$	$\lessdot$	+**id↑(id*id)**\$	*shift*
\$$E\lessdot$+	$\lessdot$	**id↑(id*id)**\$	*shift*
\$$E\lessdot$+$\lessdot$**id**	$\gtrdot$	**↑(id*id)**\$	*reduce* mit $E \to$ **id**
\$$E\lessdot$+$E$	$\lessdot$	**↑(id*id)**\$	*shift*
\$$E\lessdot$+$E\lessdot$↑	$\lessdot$	**(id*id)**\$	*shift*
\$$E\lessdot$+$E\lessdot$↑$\lessdot$(	$\lessdot$	**id*id)**\$	*shift*
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($\lessdot$**id**	$\gtrdot$	***id)**\$	*reduce* mit $E \to$ **id**
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($E$	$\lessdot$	***id)**\$	*shift*
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($E\lessdot$*	$\lessdot$	**id)**\$	*shift*
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($E\lessdot$*$\lessdot$**id**	$\gtrdot$	**)**\$	*reduce* mit $E \to$ **id**
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($E\lessdot$*$E$	$\gtrdot$	**)**\$	*reduce* mit $E \to E$*E
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($E$	$\doteq$	**)**\$	*shift*
\$$E\lessdot$+$E\lessdot$↑$\lessdot$($E\doteq$)	$\gtrdot$	\$	*reduce* mit $E \to (E)$
\$$E\lessdot$+$E\lessdot$↑$E$	$\gtrdot$	\$	*reduce* mit $E \to E$↑E
\$$E\lessdot$+$E$	$\gtrdot$	\$	*reduce* mit $E \to E$+E
\$$E$	$\gtrdot$	\$	*accept*

(c)

Eine mögliche Definition der Funktionen f und g ist in folgender Tabelle dargestellt:

	+	*	↑	(	)	**id**	\$
f	2	4	4	0	8	8	0
g	1	3	5	7	0	7	0

Aufgabe 3

(a)

Zunächst zeigt sich, daß G aufgrund der beiden Produktionen $S \rightarrow S\ ;\ S$ und $E \rightarrow E + E$ mehrdeutig ist. Wir lösen diese Mehrdeutigkeit durch Einführung zweier weiterer Nichtterminalsymbole T und F wie folgt auf:

Wir ersetzen die Produktion $S \rightarrow S\ ;\ S$ durch $S \rightarrow S\ ;\ T$ und fügen die Produktion $T \rightarrow \mathbf{id} := E$ und $T \rightarrow \mathbf{print}\ (L)$ hinzu. Ferner ersetzen wir die Produktion $E \rightarrow E + E$ durch $E \rightarrow E + F$ und fügen die Produktionen $F \rightarrow \mathbf{id}$ und $F \rightarrow \mathbf{num}$ hinzu. Insgesamt erhalten wir die Grammatik $G_1 = (N_1, \Sigma_1, P_1, S)$ mit

$$
\begin{aligned}
N_1 &= \{ S, E, L, T, F \}, \\
\Sigma_1 &= \{ \mathbf{id}, \mathbf{print}, \mathbf{num}, ;, ,, (,), :=, + \} \text{ und} \\
P_1 &= \{ S \rightarrow S\ ;\ T \mid \mathbf{id} := E \mid \mathbf{print}(L), \\
&\quad T \rightarrow \mathbf{id} := E \mid \mathbf{print}(L), \\
&\quad E \rightarrow \mathbf{id} \mid \mathbf{num} \mid E + F, \\
&\quad F \rightarrow \mathbf{id} \mid \mathbf{num}, \\
&\quad L \rightarrow E \mid L, E \}.
\end{aligned}
$$

Die Grammatik G_1 ist nicht LL(1), da linksrekursive Produktionen auftreten, die beseitigt werden müssen. Wir ersetzen dazu die Produktionen

$$S \rightarrow S\ ;\ T \mid \mathbf{id} := E \mid \mathbf{print}(L)$$

durch

$$
\begin{aligned}
S &\rightarrow \mathbf{id} := E\,S' \mid \mathbf{print}(L)\ S' \\
S' &\rightarrow ;\ T\ S' \mid \varepsilon,
\end{aligned}
$$

die Produktionen

$$E \rightarrow \mathbf{id} \mid \mathbf{num} \mid E + F$$

durch

$$
\begin{aligned}
E &\rightarrow \mathbf{id}\ E' \mid \mathbf{num}\ E' \\
E' &\rightarrow +\ F\ E' \mid \varepsilon
\end{aligned}
$$

und die Produktionen

$$L \rightarrow E \mid L, E$$

durch

$$L \rightarrow E\ L'$$
$$L' \rightarrow ,\ E\ L' \mid \varepsilon$$

Insgesamt erhalten wir die Grammatik $G_2 = (N_2, \Sigma_2, P_2, S)$ mit

$$N_2 = \{ S, S', E, E', L, L', T, F \},$$
$$\Sigma_2 = \{ \textbf{id}, \textbf{print}, \textbf{num}, ;, ,, (,), :=, + \} \text{ und}$$
$$P_2 = \{ S \rightarrow \textbf{id} := E\ S' \mid \textbf{print}(L)\ S',$$
$$S' \rightarrow ;\ T\ S' \mid \varepsilon,$$
$$T \rightarrow \textbf{id} := E \mid \textbf{print}(L),$$
$$E \rightarrow \textbf{id}\ E' \mid \textbf{num}\ E',$$
$$E' \rightarrow +\ F\ E' \mid \varepsilon,$$
$$F \rightarrow \textbf{id} \mid \textbf{num},$$
$$L \rightarrow E\ L',$$
$$L' \rightarrow ,\ E\ L' \mid \varepsilon \}.$$

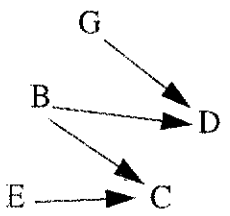
Die Linksfaktorisierung muß nicht durchgeführt werden, da es für kein Nichtterminal $X \in N_2$ verschiedene X-Produktionen mit gleichem Präfix gibt. Wir erhalten die LL(1)-Grammatik $G' = G_2$.

Aufgabe 4

(a)

Wir zeichnen den Graphen, der die Berechnungsreihenfolge festlegt. Es gilt

$$N_E = \{A, F, G, E\}$$

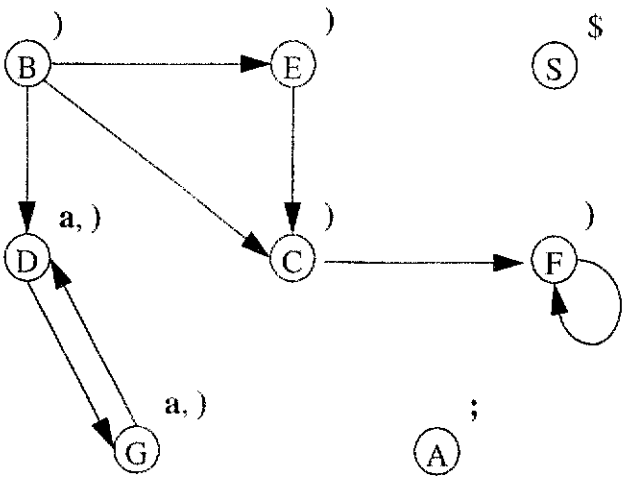


Nun berechnen wir mit Algorithmus 3.14 die FIRST- und initialen Steuermengen.

$FIRST_1$	Produktion	Steuermenge
{**c**}	$S \rightarrow$ **c a** A **;**	{**c**}
{**(**, ε}	$A \rightarrow$ **(** B **)**	{**(**}
	$\rightarrow \varepsilon$	{ε}
{**d**, **a**}	$B \rightarrow D\,E$	{**d**}
	$\rightarrow C$	{**a**}
{**a**}	$C \rightarrow$ **a : b** F	{**a**}
{**;**, ε}	$F \rightarrow$ **;** **a : b** F	{**;**}
	$\rightarrow \varepsilon$	{ε}
{**d**}	$D \rightarrow$ **d a : b ;** G	{**d**}
{**d**, ε}	$G \rightarrow D$	{**d**}
	$\rightarrow \varepsilon$	{ε}
{**a**, ε}	$E \rightarrow C$	{**a**}
	$\rightarrow \varepsilon$	{ε}

(b)

Für die Produktionen (3), (8), (11) und (13) enthält die initiale Steuermenge ε. Für die Nichtterminale der linken Seiten dieser Produktionen werden die FOLLOW-Mengen benötigt. Anwendung des Algorithmus 3.15 (ohne Schritt 3) liefert den folgenden Graphen:



Nun lassen sich die benötigten FOLLOW-Mengen ablesen.

$\text{FOLLOW}_1(A)=\{\mathbf{;}\}$

$\text{FOLLOW}_1(F)=\{\mathbf{)}\}$

$\text{FOLLOW}_1(G)=\{\mathbf{a}, \mathbf{)}\}$

$\text{FOLLOW}_1(E)=\{\mathbf{)}\}$

		Produktion	Steuermenge
(1)	S	$\rightarrow \mathbf{c\ a}\ A\ \mathbf{;}$	{**c**}
(2)	A	$\rightarrow \mathbf{(}\ B\ \mathbf{)}$	{**(**}
(3)		$\rightarrow \varepsilon$	{**;**}
(4)	B	$\rightarrow D\ E$	{**d**}
(5)		$\rightarrow C$	{**a**}
(6)	C	$\rightarrow \mathbf{a : b}\ F$	{**a**}
(7)	F	$\rightarrow \mathbf{;\ a : b}\ F$	{**;**}
(8)		$\rightarrow \varepsilon$	{**)**}
(9)	D	$\rightarrow \mathbf{d\ a : b ;}\ G$	{**d**}
(10)	G	$\rightarrow D$	{**d**}
(11)		$\rightarrow \varepsilon$	{**a**, **)**}
(12)	E	$\rightarrow C$	{**a**}
(13)		$\rightarrow \varepsilon$	{**)**}

(c) Die Analysetafel hat folgende Gestalt:

$n \in N_1 \setminus t \in \Sigma$	**c**	**d**	**a**	**b**	**(**	**)**	**:**	**;**
S	1							
A					2			3
B		4	5					
C			6					
F						8		7
D		9						
G		10	11			11		
E			12			13		

Aufgabe 5

(a)

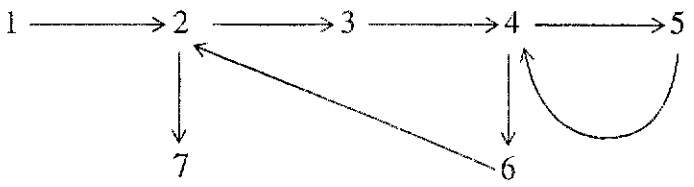
Der Term $n * i$ kann aus der inneren Schleife ausgelagert werden, der Term $\log(n)$ sogar aus der äußeren. Dazu führen wir die Hilfsvariablen q_1 und q_2 ein:

```
q1 := log(n);
FOR i := 1 TO k1 DO
  q2 := n * i;
  FOR j := 1 TO k2 DO
    a := q2;
    b := j * q1;
    DoSomething(a, b);
  END
END
```

(b) Wir geben das 3AC-Programm direkt mit seiner Unterteilung in Basisblöcke an:

Block	Anweisung
1	(1) q_1 := log(n) (2) i := 1
2	(3) if i > k_1 then goto (14)
3	(4) q_2 := n * i (5) j := 1
4	(6) if j > k_2 then goto (12)
5	(7) a := q_2 (8) b := j * q_1 (9) DoSomething(a, b) (10) j := j + 1 (11) goto (6)
6	(12) i := i + 1 (13) goto (3)
7	(14) ...

(c) Der Flußgraph hat folgenden Aufbau:



(d) Die Multiplikationen in den Zeilen (4) und (8) lassen sich in Additionen überführen. Wir zeigen dies für die Addition in der äußeren Schleife in Zeile (4) durch Einführung der Hilfsvariablen q_3.

Block	Anweisung
1	(1) q_1 := log(n) (2) i := 1 (3) q_3 := n * i
2	(4) if i > k_1 then goto (15)
3	(5) q_2 := q_3 (6) j := 1
4	(7) if j > k_2 then goto (13)
5	(8) a := q_2 (9) b := j * q_1 (10) DoSomething(a, b) (11) j := j + 1 (12) goto (7)
6	(13) q_3 := q_3 + n (14) i := i + 1 (15) goto (4)
7	(16) ...