Vorname Nachname Matrikel-Nr.

Diese Klausur besteht aus **4 Aufgaben**. Schreiben Sie jede Ihrer Lösungen auf die Vorderseite eines *neuen* Blattes (und lassen Sie die Rückseiten Ihrer Lösungsblätter *leer*).

Für die **Aufgabe 3** ("Prädikate programmieren") gilt: Sie dürfen *zusätzliche Hilfsprädikate* vereinbaren, wenn Sie möchten.

Aufgabe 1 (20 Punkte): Geben Sie eine (kontextfreie, Typ-2-) *Grammatik* an für die Menge aller natürlichen Zahlen (0, 1, 2, ...) im **5-er-System**, die (ohne Rest) **durch 3 teilbar** sind. Startsymbol `RK0` (wie "Restklasse Null").

Aufgabe 2 (20 Punkte): Geben Sie eine *Grammatik für Ausdrücke* an, die aus den folgenden Bestandteilen zusammengesetzt werden können:

Aus Literalen (die aus dem Zwischensymbol `LIT` ableitbar sind),
aus runden Klammern `(...)`,
aus dem einstelligen präfix-Operator `"$D"` mit der Bindungsstärke 4
aus dem einstelligen präfix-Operator `"$C"` mit der Bindungsstärke 3
aus dem zweistelligen, rechtsassoziativen infix-Operator `"$B"` mit der Bindungsstärke 2 und
aus dem zweistelligen, linksassoziativen infix-Operator `"$A"` mit der Bindungsstärke 1

Startsymbol: `AUS1`.
Weitere **Zwischensymbole**: `AUS2`, `AUS3`, ...

Die Regeln für das Zwischensymbol `LIT` sind vorgegeben wie folgt:

```
LIT  : "x"
LIT  : "y"
LIT  : "z"
```

Beispiele (sechs Stück) für Ausdrücke, die aus dem Startsymbol `AUS1` Ihrer Grammatik ableitbar sein sollen (die sechs Ausdrücke sind hier durch mehrere Blanks voneinander getrennt):

```
x      $D y      $C z      x $B y      y $A z      (x $A y) $B z
```

Aufgabe 3 (40 Punkte):

3.1. Schreiben Sie ein Prädikat entsprechend der folgenden Spezifikation:

```
proc listeDerNormalen(L:int[] -> LN:int[])
   // Die Liste LN enthaelt alle normalen Elemente der Liste L.
   // Welche Elemente "normal" sind, entscheidet das Praedikat
   //
   // condition istBesonders(N:int)
   //    // Gelingt, wenn die Zahl N besonders ist
   //    // Gelingt nicht, wenn die Zahl N normal ist
   //
   // Nehmen Sie an, dass das Praedikat istBesonders vorgegeben ist
   // (d.h. Sie brauchen es nicht zu vereinbaren, duerfen es aber aufrufen).
```

3.2. Schreiben Sie ein Prädikat entsprechend der folgenden Spezifikation:

```
condition keineDoppelten(L:int[])
   // Gelingt genau dann wenn in L keine Zahl mehr als einmal vorkommt
   // Beispiele:
   // keineDoppelten(int[1, 2, 3, 2, 5]) gelingt nicht (2 kommt doppelt vor)
   // keineDoppelten(int[1, 2, 3, 4, 5]) gelingt
```

3.3. Schreiben Sie ein Prädikat entsprechend der folgenden Spezifikation:

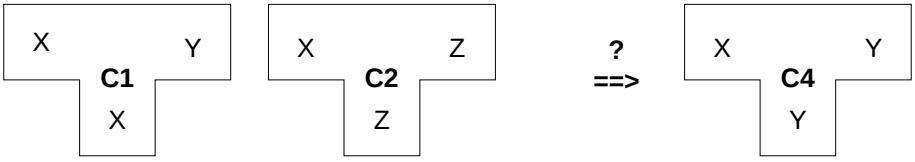
```
proc merge(L1:int[], L2:int[] -> L3:int[])
   // Verlaesst sich darauf, dass L1 und L2 aufsteigend sortiert sind.
   // L3 enthaelt dann alle Elemente von L1 und L2 und ist
   // ebenfalls aufsteigend sortiert.
   // Beispiel: Nach dem Aufruf
   // merge(int[2,4,8], int[5, 6, 8, 9] -> L3)
   // ist L3 gleich int[2, 4, 5, 6, 8, 8, 9]
```

Aufgabe 4 (20 Punkte):

Nehmen Sie an:

Sie haben eine **Z-Maschine**,
und die beiden Compiler **C1** und **C2**.
Wie können Sie damit (in 2 Schritten) den Compiler **C4** erzeugen?

Geben Sie als Lösung für jeden der beiden Schritte ein entsprechendes TI-Diagramme an.



Beurteilung der Klausur:

Punkte↓	
Aufgabe 1:	Note:
Aufgabe 2:	Datum:
Aufgabe 3:	
Aufgabe 4:	
Summe:	

Korrigierte Beurteilung der Klausur:

Punkte↓	
Aufgabe 1:	Note:
Aufgabe 2:	Datum:
Aufgabe 3:	
Aufgabe 4:	
Summe:	

Lösung 1 (20 Punkte): Geben Sie eine (kontextfreie, Typ-2-) Grammatik an für die Menge aller natürlichen Zahlen (0, 1, 2, ...) im 5-**er-System**, die (ohne Rest) **durch 3 teilbar** sind. Startsymbol `RK0` (wie "Restklasse 0").

```
R01: RK0 : "0" | R06: RK0 : RK0 "0" | R11: RK2 : RK1 "0" | R16: RK1 : RK2 "0"
R02: RK1 : "1" | R07: RK1 : RK0 "1" | R12: RK0 : RK1 "1" | R17: RK2 : RK2 "1"
R03: RK2 : "2" | R08: RK2 : RK0 "2" | R13: RK1 : RK1 "2" | R18: RK0 : RK2 "2"
R04: RK0 : "3" | R09: RK0 : RKO "3" | R14: RK2 : RK1 "3" | R19: RK1 : RK2 "3"
R05: RK1 : "4" | R10: RK1 : RK0 "4" | R15: RK0 : RK1 "4" | R20: RK2 : RK2 "4"
```

Lösung 2 (20 Punkte): Geben Sie eine Grammatik für Ausdrücke an, die aus den folgenden Bestandteilen zusammengesetzt werden können:

Aus Literalen (die aus dem Zwischensymbol `LIT` ableitbar sind), runden Klammern `(...)`,
aus dem einstelligen präfix-Operator `"$D"` mit der Bindungsstärke 4
aus dem einstelligen präfix-Operator `"$C"` mit der Bindungsstärke 3
aus dem zweistelligen, rechtsassoziativen infix-Operator `"$B"` mit der Bindungsstärke 2 und
aus dem zweistelligen, linksassoziativen infix-Operator `"$A"` mit der Bindungsstärke 1

Startsymbol: `AUS1`.
Weitere Zwischensymbole: `AUS2`, `AUS3`, ...

Die Regeln für das Zwischensymbol `LIT` sind vorgegeben wie folgt:

```
LIT  : "x"
LIT  : "y"
LIT  : "z"
```

Beispiele (sechs Stück) für Ausdrücke, die aus dem Startsymbol `AUS1` Ihrer Grammatik ableitbar sein sollen (die Ausdrücke sind hier durch mehrere Blanks voneinander getrennt):

```
x      $D y      $C z      x $B y      y $A z      (x $A y) $B z
```

Die Grammatik:

```
AUS1 : AUS1 "$A" AUS2
AUS1 : AUS2

AUS2 : AUS3 "$B" AUS2
AUS2 : AUS3

AUS3 : "$C" AUS4
AUS3 : AUS4

AUS4 : "$D" AUS5
AUS4 : AUS5

AUS5 : LIT
AUS5 : "(" AUS1 ")"
```

Lösung 3 (40 Punkte):

3.1. Schreiben Sie ein Prädikat entsprechend der folgenden Spezifikation:

```
proc listeDerNormalen(L:int[] -> LN:int[])
   // Die Liste LN enthaelt alle normalen Elemente der Liste L.
   // Welche Elemente "normal" sind, entscheidet das Praedikat
   //
   // condition istBesonders(N:int)
   //    // Gelingt, wenn die Zahl N besonders ist
   //    // Gelingt nicht, wenn die Zahl N normal ist
   //
   // Nehmen Sie an, dass das Praedikat istBesonders vorgegeben ist
   // (d.h. Sie brauchen es nicht zu vereinbaren, duerfen es aber aufrufen).

   rule listeDerNormalen(int[] -> int[]):
   rule listeDerNormalen(int[N::R] -> ERG):
      listeDerNormalen(R -> BR)
      {
         istBesonders(N)
         ERG <- BR
      |
         ERG <- int[N::BR]
      }
```

3.2. Schreiben Sie ein Prädikat entsprechend der folgenden Spezifikation:

```
condition keineDoppelten(L:int[])
   // Gelingt genau dann wenn in L keine Zahl mehr als einmal vorkommt
   // Beispiele:
   // keineDoppelten(int[1, 2, 3, 2, 5]) gelingt nicht (2 kommt doppelt vor)
   // keineDoppelten(int[1, 2, 3, 4, 5]) gelingt

   rule keineDoppelten(int[]):
   rule keineDoppelten(int[N::R]):
      kommtNichtVor(N, R)
      keineDoppelten(R)

condition kommtNichtVor(N:int, L:int[])
   rule kommtNichtVor(N, int[]):
   rule kommtNichtVor(N, int[M::R]):
      Unequal(N, M)
      kommtNichtVor(N, R)
```

3.3. Schreiben Sie ein Prädikat entsprechend der folgenden Spezifikation:

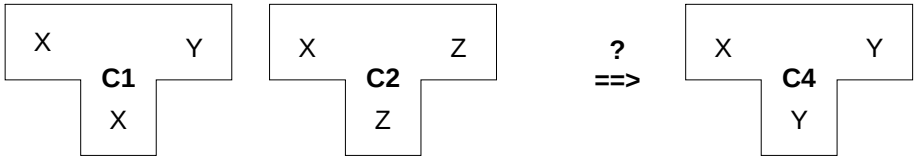
```
proc merge(L1:int[], L2:int[] -> L3:int[])
   // Verlaesst sich darauf, dass L1 und L2 aufsteigend sortiert sind.
   // L3 enthaelt dann alle Elemente von L1 und L2 und ist
   // ebenfalls aufsteigend sortiert.
   // Beispiel: Nach dem Aufruf
   // merge(int[2,4,8], int[5, 6, 8, 9] -> L3)
   // ist L3 gleich int[2, 4, 5, 6, 8, 8, 9]

   rule merge(int[], L2 -> L2):
   rule merge(L1, int[] -> L1):
   rule merge(int[N1::R1], int[N2::R2] -> int[N::L]):
      {
         Less(N1, N2)
         N <- N1
         merge(R1, int[N2::R2] -> L)
      |
         N <- N2
         merge(int[N1::R1], R2 -> L)
      }
```

Lösung 4 (20 Punkte):

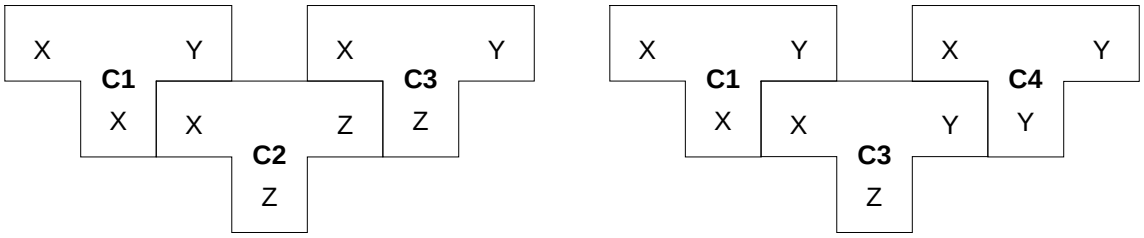
Nehmen Sie an:

Sie haben eine **Z-Maschine**,
und die beiden Compiler **C1** und **C2**.
Wie können Sie damit (in 2 Schritten) den Compiler **C4** erzeugen?

Geben Sie als Lösung für jeden der beiden Schritte ein entsprechendes TI-Diagramme an.



Lösung:



Schritt 1: Schritt 2: