

# Probepfprüfung

Hochschule Reutlingen - Reutlingen University  
Fakultät Informatik  
Studiengang Medien- und Kommunikationsinformatik

## Informatik 1

mki-Bachelor 1  
Wintersemester 2009/10

☞ Name, Vorname .....

☞ Matrikelnummer .....

☞ Platznummer .....

Prüfer: Prof. Dr. Karlheinz Hug  
Prüfungsdauer: 90 Minuten  
Zugelassene Hilfsmittel: Alle (Skripten, Übungsmaterial, Literatur,...)  
Anzahl der Aufgabenseiten: 10

Aufgabe	Punkte		Aufgabe	Punkte	
	möglich	erreicht		möglich	erreicht
1	18		6	10	
2	6		7	18	
3	6		8	14	
4	10		9	19	
5	7				
<b>Summe möglich:</b>		<b>108</b>	<b>Summe erreicht:</b>		
			<b>Note:</b>		

- Zum „Bestehen“ der Probepfprüfung sind **45**, für die Note „Eins“ **90** Punkte erforderlich.
- Bearbeiten Sie die Aufgaben auf dazu freigelassenen Stellen! Tragen Sie in Tabellen und an punktierten Stellen ... im Text passende Angaben ein! Der Platz reicht normalerweise; vermeiden Sie Extrablätter!
- Aufgabe 3 bezieht sich auf das Modul von Aufgabe 2. Die anderen Aufgaben, meist auch Teilaufgaben, sind unabhängig voneinander lösbar.
- Sie erhalten am Ende der Probepfprüfung eine Musterlösung und können damit Ihre Ergebnisse selbst bewerten.



**Viel Erfolg!**

**Aufgabe 1      Syntax beschreiben**

(Punkte: 18 |.....)    (1)    Eine kleine Sprache ist in EBNF-Notation durch folgende Regeln gegeben.

$$S = Z \mid Z \text{ „+“ } Z.$$

$$Z = \text{ „1“ } \mid \text{ „2“ } \mid \text{ „3“ }.$$

Schreiben Sie alle Wörter dieser Sprache auf!

Vereinfachen Sie die erste Regel mittels einer Option!

(2)    Die erste Regel in (1) wird durch

$$S = Z \{ \text{ „+“ } Z \}.$$

ersetzt. Beschreiben Sie verbal, wie die Wörter dieser Sprache aussehen!

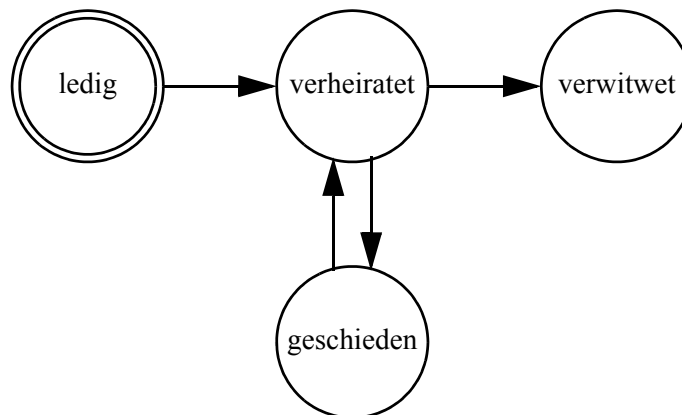
(3)    Geben Sie zu den EBNF-Regeln aus (1) - (2) entsprechende Syntaxdiagramme an!

(4)    Geben Sie zu dem Wort

$$\text{ „1+2+3“ }$$

nach den EBNF-Regeln aus (2) einen Zerteilungsbaum an!

- (5) Geben Sie zum Zustandsdiagramm einen regulären EBNF-Ausdruck an, der alle möglichen Reihenfolgen von Zuständen mit Anfangszustand „ledig“ spezifiziert!



- (6) Beschriften Sie die Pfeile im Zustandsdiagramm von (5) mit passenden Namen für Aktionen und geben Sie einen regulären EBNF-Ausdruck an, der alle möglichen Reihenfolgen von Aktionen spezifiziert!
- (7) Geben Sie zum EBNF-Ausdruck aus (6) ein entsprechendes Syntaxdiagramm an!
- (8) **Hausaufgabe nach Probepfprüfung** (Extrapunkte: 22): Ergänzen Sie das Zustandsdiagramm um die Zustände „ungeboren“ und „gestorben“, passende Pfeile und Beschriftungen der neuen Pfeile und führen Sie die Teilaufgaben (5), (6), (7) für das so erweiterte Zustandsdiagramm aus!

## Aufgabe 2 Reihung vertraglich spezifizieren

(Punkte: 6 |.....) Ergänzen Sie die *Cleo*-Spezifikation des Moduls, das eine Reihung modelliert, mit Invarianten, Vor- und Nachbedingungen und Kommentaren zu Bedingungen! Element ist ein beliebiger Typ. Die Reihung enthält Einträge von Elementen; sie realisiert eine Abbildung von Indizes auf Elemente.

**Programm 2.1**  
Cleo: Reihung

```

MODULE Array
  QUERIES
    Capacity : NATURAL (* Konstante Maximalzahl von Einträgen *)
    Item (IN index : NATURAL) : Element (* Falls Position index gültig, zugehöriges Element *)
    PRE
      index < Capacity (* ..... *)
    POST
      (* Has (result) auskommentiert, da Item elementar, Has abgeleitet *)
    END
    Has (IN x : Element) : BOOLEAN (* Ist das Element x enthalten? *)
    POST
      result IMPLIES
        EXISTS index : NATURAL IT_HOLDS (index < Capacity) AND (Item (index) = x)
    END
  INVARIANTS
    ..... (* Nie leer *)
  ACTIONS
    Put (IN index : NATURAL; IN x : Element) (* Füge x an der Position index ein. *)
    PRE
      ..... (* Position index gültig *)
    POST
      Has (x); (* ..... *)
      ..... (* ..... *)
    END
END Array
    
```

## Aufgabe 3 Reihung mit Zusicherungen testen

(Punkte: 6 |.....) Die *CP*-ähnlichen Anweisungen testen das Reihungsmodul aus Aufgabe 2. Dabei ist Element gleich INTEGER. Ergänzen Sie die Zusicherungen!

**Programm 3.1**  
CP: Reihungstest

```

FOR index := 0 TO Array.Capacity - 1 DO
  Array.Put (index, index * index);
  ASSERT (Array.Has (.....));
  ASSERT (Array.Item (.....) = .....);
END;
FOR index := 0 TO Array.Capacity - 1 DO
  Array.Put (index, Array.Item (index) + 1);
  ASSERT (Array.Has (.....));
  ASSERT (Array.Item (.....) = .....);
END
    
```

### Aufgabe 4      Rekursionsformel und Schleife konstruieren

(Punkte: 10 |.....)      Ergänzen Sie die Nachbedingungen der *Cleo*-Abfrage zu einer Rekursionsformel für die Quersumme der ganzen Zahl *n*!

**Programm 4.1**  
Cleo: Quersumme

```
QUERIES
  SumOfTheDigits (IN n : INTEGER) : INTEGER
  POST
    (n < 0) IMPLIES (result = .....);
    ((0 <= n) AND (n < 10)) IMPLIES .....;
    (10 <= n) IMPLIES
      .....
  END
```

Skizzieren Sie einen Algorithmus, der die Rekursionsformel mit einer Bedingungs-  
schleife realisiert!

### Aufgabe 5 Typregeln anwenden

(Punkte: 7 |.....)

Die CP-Vereinbarungen

VAR i : INTEGER; x, y : REAL;

vorausgesetzt, zeigen Sie in folgender Zuweisung die Typen aller Größen und Ausdrücke sowie Typanpassungen und Typfehler an!

i := (x + 4) & (y > 0)

Welches Werkzeug erkennt Typfehler zu welchem Zeitpunkt?

### Aufgabe 6 Sprachkonstrukte kennen

(Punkte: 10 |.....)

Begründen Sie zu den Aussagen, warum sie richtig bzw. falsch sind!

Aussage	Begründung
In einer Anweisung können Ausdrücke vorkommen.	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
In einem Ausdruck können Anweisungen vorkommen.	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil

Die englische Bezeichnung für Anweisung lautet .....

Die deutsche Bezeichnung für *assertion* lautet .....

Übersetzen Sie den folgenden Abschnitt aus dem *Component Pascal Language Report*!

*A **data type** determines the set of values which variables of that type may assume, and the operators that are applicable. A **type declaration** associates an identifier with a type. In the case of structured types (arrays and records) it also defines the structure of variables of this type. A structured type cannot contain itself.*

## Aufgabe 7 Programmstücke verbessern

(Punkte: 18 |.....) Die gegebenen Programmstücke sind durch einfachere, aber äquivalente Programmstücke zu ersetzen, d.h. ihr Wert bzw. Effekt muss erhalten bleiben.

Vereinfachen Sie den booleschen Ausdruck so weit wie möglich!

$$(\sim(x < 8) \ \& \ (y > 9)) \ \text{OR} \ ((x \geq 8) \ \& \ \sim(y \leq 9))$$

Vereinfachen Sie die Anweisungen so weit wie möglich!

Anweisung	Verbesserte Fassung
<pre> IF k &lt; 3 THEN   x := 4 * a;   y := 5 * x + k;   z := 6 - y ELSE   x := 4 * a;   y := x + 7 * k;   z := 6 - y END                     </pre>	
<pre> IF ~small THEN   IF ~big THEN     medium := TRUE   ELSE     medium := FALSE   END ELSE   medium := FALSE END                     </pre>	

Verbessern Sie die (korrekte) Implementation der Funktionsprozedur zum Testen des Sortiertseins so, dass sie mit *einer* Variable und *zwei* Zuweisungen *weniger* auskommt!

**Programm 7.1**

CP: Sortiertsein prufen

```
PROCEDURE Sorted (IN x : ARRAY OF Comparable) : BOOLEAN;  
  (*! Postcondition: result = the elements of x are ordered. !*)  
  VAR  
    result : BOOLEAN;  
    i      : Index;  
  BEGIN  
    result := TRUE;  
    i := 0;  
    WHILE result & (i < LEN (x) - 1) DO  
      result := (x [i] <= x [i + 1]);  
      INC (i)  
    END;  
    RETURN result  
  END Sorted;
```

### Aufgabe 8 Schleifenablauf und -terminierung prüfen, Schleifen umformen

(Punkte: 14 |.....) Verfolgen Sie jeweils einige Durchläufe der beiden Schleifen und begründen Sie zu jeder Schleife, warum sie abbricht bzw. warum nicht!

Schleife 1	Schleife 2	
<pre>k := 13579; WHILE k # 0 DO   k := k - 2;   IF k &lt; 0 THEN     k := -3 * k   END END END</pre>	<pre>FOR n := 1 TO 2 DO   m := n + 1;   n := m - 2 END</pre>	
k	n	m
<input type="checkbox"/> Terminiert <input type="checkbox"/> Läuft endlos, weil	<input type="checkbox"/> Terminiert <input type="checkbox"/> Läuft endlos, weil	

Formulieren Sie Schleife 2 (FOR ...) in eine äquivalente WHILE-Schleife um!

Zeigen Sie in vier Schritten durch Ersetzen von Programmstücken durch äquivalente Stücke, dass folgende Schleife *nicht* terminiert!

```
mustGoOn := TRUE;
WHILE mustGoOn DO
  mustGoOn := mustGoOn OR FALSE
END
```

## Aufgabe 9 Programmablauf verfolgen

(Punkte: 19 |.....) Untersuchen Sie Programm 9.1 wie weiter unten beschrieben!

### Programm 9.1

CP: Unbekannter  
Algorithmus

```

CONST
  number = 8;

VAR
  row      : ARRAY number OF INTEGER;
  index1, index2 : INTEGER;

BEGIN
  ... (* Initialisierung der Variablen row. *) ...
  FOR index1 := 2 TO LEN (row) - 1 DO
    row [0] := row [index1];
    index2 := index1 - 1;
    WHILE row [0] < row [index2] DO
      row [index2 + 1] := row [index2];
      index2 := index2 - 1
    END;
    row [index2 + 1] := row [0]
  END
END ...
    
```

In der folgenden Tabelle ist jeder Variable eine Spalte zugeordnet. In den Zeilen (außer der Kopfzeile) werden die Werte, die die Variablen besitzen, wenn die Programmausführung die Stelle (\*!\*) erreicht hat, aufgezeichnet.

Der Programmablauf befindet sich an der Stelle (\*!\*). Verfolgen Sie ihn weiter und vervollständigen Sie die Tabelle, indem Sie jeweils die Zustände der Variablen bis zum Erreichen der Stelle (\*!\*) in eine neue Zeile eintragen! (Tragen Sie auch Zwischenwerte der Variable index2 ein und streichen Sie sie ggf. durch.)

row								index1	index2
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]		
56	79	34	12	17	8	43	6		
34	34	79	12	17	8	43	6	2	4 0
12	12	34	79	17	8	43	6	3	<del>2</del> 4 0
17	12	17	34	79	8	43	6	4	<del>3</del> <del>2</del> 1