Prüfung

Informatik 1

Hochschule Reutlingen - Reutlingen University
Fakultät Informatik
Studiengang Medien- und Kommunikationsinformatik
akkreditiert von der ASIIN

mki-Bachelor 1
Wintersemester 2009/10

☞ Name, Vorname .....

☞ Matrikelnummer .....

☞ Platznummer .....

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungstermin und Ort: Dienstag, 02. Februar 2010, 10^{30} - 12^{30} Uhr, Aula

Prüfungsdauer: 120 Minuten

Zugelassene Hilfsmittel: Alle (Skripten, Übungsmaterial, Literatur,...)

Anzahl der Aufgabenseiten: 12

Aufgabe	Punkte		Aufgabe	Punkte	
	möglich	erreicht		möglich	erreicht
1	22		6	14	
2	14		7	17	
3	13		8	24	
4	16		9	12	
5	22		**Summe:**	**154**	
			Note:		

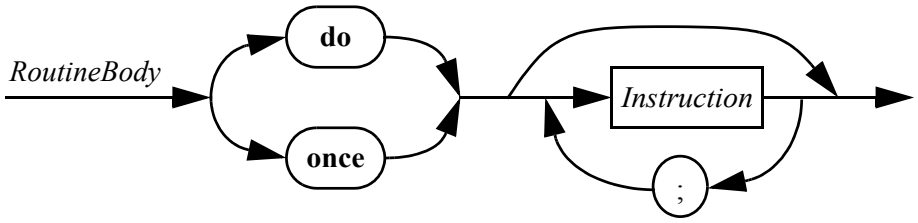
- Zum Bestehen der Prüfung sind **60**, für die Note „Eins“ **120** Punkte erforderlich.
- Lassen Sie die erhaltenen Blätter zusammengeheftet und geben Sie alle Blätter geheftet wieder ab, sonst droht Abzug von Punkten!
- Bearbeiten Sie die Aufgaben auf dazu freigelassenen Stellen! Tragen Sie in Tabellen und an punktierten Stellen ... im Text passende Angaben ein!
- Der Platz reicht normalerweise; vermeiden Sie Extrablätter!
- Teilaufgaben sind meist unabhängig voneinander lösbar.

✌ **Viel Erfolg!**

Aufgabe 1 Syntax beschreiben

(Punkte: 22 |.....)

(1) Stellen Sie das folgende Syntaxdiagramm mit einer EBNF-Regel dar!



RoutineBody =

(2) Gegeben ist die Syntax eines Namens in EBNF-Notation:

Vollname = *Titel Nachname* „,“ *Vornamen*.
Titel = [„Prof.“] { „Dr.“ [„h.c.“ | „rer.nat.“] }.
Nachname = *Name*.
Vornamen = *Name* { *Name* }.
Name = *ident* [„-“ *ident*].
(lexikalische Einheit: *ident* = *letter* { *letter* }.)

Zeichnen Sie zu den ersten fünf dieser Regeln äquivalente Syntaxdiagramme!

Zeichnen Sie zum folgenden Wort den konkreten Syntax-Zerteilungsbaum (ohne *ident* in einzelne Zeichen zu zerteilen)!

Prof. Dr. rer.nat. Dr. Adler-Burger, Cora Dori-Evi

Aufgabe 2 Tabelle vertraglich spezifizieren

(Punkte: 14 |.....)

Ergänzen Sie die Spezifikation der Schnittstelle, die eine Tabelle modelliert, mit Invarianten, Vor- und Nachbedingungen und Kommentaren zu Bedingungen! Key und Element sind beliebige Typen. Die Tabelle enthält Einträge von Schlüssel-Element-Paaren; sie realisiert eine Abbildung von Schlüsseln auf Elemente.

Programm 2.1
Cleo: Tabelle

```
INTERFACE Table
  QUERIES
    Capacity              : NATURAL     (* Konstante Maximalzahl von Einträgen *)
    Count                 : NATURAL     (* Wieviele Einträge sind enthalten? *)
    notFound              : Element     (* Konstante, die nicht als echtes Element vorkommt *)
    Has (IN x : Element)  : BOOLEAN     (* Ist das Element x enthalten? *)
      POST
        result IMPLIES (Count > 0)
      END
    Item (IN key : Key)   : Element
                          (* Falls Schlüssel key enthalten, zugehöriges Element, sonst notFound *)
      POST
        (result = notFound) OR Has (result)
      END
  INVARIANTS
    ..........................................................   (* Beziehung Count - Capacity *)
    ..........................................................   (* notFound kein echtes Element *)
  ACTIONS
    Put (IN key : Key; IN x : Element)                          (* Füge (key, x) hinzu. *)
      PRE
        Count < Capacity;                   (* ..... *)
        x # notFound;                       (* ..... *)
        Item (key) = notFound               (* ..... *)
      POST
        Count = OLD (Count) + 1;            (* ..... *)
        Has (x);                            (* ..... *)
        Item (key) = x                      (* ..... *)
      END
    Remove (IN key : Key)                   (* Entferne key und zugehöriges Element. *)
      PRE
        ..........................................................   (* nicht leer *)
        ..........................................................   (* Schlüssel key enthalten *)
      POST
        ..........................................................   (* ein Eintrag weniger *)
        ..........................................................   (* Schlüssel key nicht enthalten *)
      END
    WipeOut                                 (* Entferne alle Einträge. *)
      POST
        ..........................................................   (* ..... *)
      END
END Table
```

Aufgabe 3 Korrektheit eines Algorithmus beweisen

(Punkte: 13 |.....)

Die folgende Prozedur liefert das Minimum der Werte ihrer drei Eingabeparameter als Wert des Ausgabeparameters. Beweisen Sie die Korrektheit des Algorithmus, indem Sie möglichst starke Zusicherungen ergänzen!

Programm 3.1
CP: Minimum dreier Zahlen

```
PROCEDURE GetMin (a, b, c : REAL; OUT min : REAL);
BEGIN
   IF a < b THEN

      ASSERT (.....);
      IF a < c THEN

         ASSERT (.....);
         min := a;

         ASSERT (.....)
      ELSE

         ASSERT (.....);
         min := c;

         ASSERT (.....)
      END;

      ASSERT (.....

         .....)
   ELSIF c < b THEN

      ASSERT (.....);
      min := c;

      ASSERT (.....)
   ELSE

      ASSERT (.....);
      min := b;

      ASSERT (.....)
   END;

   ASSERT (.....

      .....

      .....

      .....);
   ASSERT ((min <= a) & (min <= b) & (min <= c) &
            ((min = a) OR (min = b) OR (min = c)), postcondition)
END GetMin;
```

Aufgabe 4 Übersetzen, Typregeln anwenden

(Punkte: 16 |.....) Übersetzen Sie den folgenden Abschnitt aus dem *Component Pascal Language Report*!

> *A **module** is a collection of declarations of constants, types, variables, and procedures, together with a sequence of statements for the purpose of assigning initial values to the variables. A module constitutes a text that is compilable as a unit.*

Mit den Vereinbarungen

besteht die Anweisung

```
IF (k [m] < n) = p THEN n := 2 / (m + 3) ELSE n := m MOD 4 END
```

alle Prüfungen der Typverträglichkeit.

Geben Sie mit den Vereinbarungen

```
VAR b : BOOLEAN; c : CHAR; i : INTEGER; x : REAL;
```

zu jedem Teilausdruck des folgenden Ausdrucks an, von welchem Typ er ist, und wo ein Wert implizit an einen anderen Typ angepasst wird, und wo ein Typverträglichkeitsfehler vorliegt!

```
( b = ( i > 5.6E-7 ) ) # ( ORD (c) + x <= 'x' )
```

Aufgabe 5 Programmstücke verbessern

(Punkte: 22 |.....)

Die gegebenen Programmstücke sind in äquivalente Programmstücke umzuformen, d.h. ihre Semantik muss erhalten bleiben.

Geben Sie zur nebenstehenden Zuweisungsfolge die Eingabegrößen ..... und

Ausgabegrößen ..... an und verkürzen Sie die Folge in sechs Schritten zu einer einzigen Zuweisung, die ohne die Variablen x, y auskommt!

```
z := a;
y := b;
x := c;
y := y + x;
z := z * y;
y := d;
z := z / y
```

Transformieren Sie die nebenstehende Auswahlanweisung schrittweise mit Erklärungen in bessere Varianten, mindestens die letzte Variante schön einrückend!

```
IF q > 0 THEN C; D
ELSE IF q < 0 THEN B;
A; D ELSE IF q = 0
THEN D ELSE R; Z; D
END END END
```

Vereinfachen Sie die Implementation der Funktion F schrittweise mit Erklärungen so, dass sie mit zwei Variablen, zwei Zuweisungen, einer Zählschleife und einer **RETURN**-Anweisung auskommt!

Programm 5.1
CP: Aufgeblähte Implementation

```
PROCEDURE F (n : INTEGER) : REAL;
   VAR
      i      : INTEGER;
      x, y   : REAL;
BEGIN
   x := 0;
   y := x + 4;
   i := 1;
   WHILE i <= n DO
      x := x + 5 / i;
      y := x + 4;
      i := i + 1
   END;
   IF n MOD 2 = 1 THEN
      x := x - 1
   END;
   RETURN x * y
END F;
```

Aufgabe 6 Schleifen: Terminierung prüfen, umformen, Aufwand berechnen

(Punkte: 14 |.....)

Verfolgen Sie zur Schleife

```
p := 3; q := 2; r := 1;
WHILE p > 0 DO
   IF q <= r THEN
      p := p - r;
      q := q + r
   ELSE
      p := p + r;
      r := r + 1
   END
END
```

sechs Durchläufe und begründen Sie, warum sie abbricht oder nicht!

p	q	r
3	2	1

❒ Terminiert ❒ Läuft endlos, weil

Transformieren Sie die Zählschleife

```
FOR k := 4 * k TO 5 * k DO
   Do (k)
END
```

in eine äquivalente kopfgesteuerte Bedingungsschleife!

Wenn die Anweisung X stets *x*, die Anweisung Y stets *y* Zeiteinheiten benötigt und die Zeit für die restliche Arbeit vernachlässigt wird, welche Ausführungszeit benötigt dann die folgende (äußere) Schleife?

```
FOR i := 1 TO m DO
   FOR k := 1 TO n DO
      X (i, k)
   END;
   Y (i)
END
```

Ausdruck für Aufwand

Die Anweisung X sollte zuerst beschleunigt werden, wenn gilt:

Ungleichung

Aufgabe 7 Programmablauf verfolgen

(Punkte: 17 |.....) Untersuchen Sie Programm 7.1 wie unten beschrieben!

Programm 7.1
CP: Unbekannter Algorithmus

```
VAR row : ARRAY 6 OF INTEGER;
    i1, i2, s, a : INTEGER;
(* Initialisierung der Variablen row. *)                    (* 1 *)
FOR i1 := 0 TO LEN (row) - 2 DO
   s := i1;
   FOR i2 := i1 + 1 TO LEN (row) - 1 DO
      IF row [s] > row [i2] THEN
         s := i2
      END
   END;
   a          := row [i1];
   row [i1]  := row [s];
   row [s]   := a                                           (* 2 *)
END                                                         (* 3 *)
```

Die folgende Tabelle ordnet jeder Variablen eine Spalte zu. Der Programmablauf hat die Stelle *(* 1 *)* erreicht. Verfolgen Sie den Ablauf weiter und vervollständigen Sie die Tabelle, indem Sie jeweils die Zustände der Variablen bis zum Erreichen der Stelle *(* 2 *)* in eine neue Zeile eintragen, bis die Stelle *(* 3 *)* erreicht ist! (Streichen Sie ggf. Zwischenwerte der Variablen so, dass die Änderungen nachvollziehbar sind!)

row						i1	s	i2	a
[0]	[1]	[2]	[3]	[4]	[5]				
18	76	2	3	94	5				

Aufgabe 8 Programmiermängel suchen und beseitigen

(Punkte: 24 |.....)

Mathematisch ist das Skalarprodukt zweier Vektoren $x = (x_1,.., x_n)$, $y = (y_1,.., y_n) \in \mathbb{R}^n$ durch $x * y := \sum_{i=1}^{n} x_i \cdot y_i$ definiert. Die folgende Funktion soll es berechnen.

Programm 8.1
CP: Defektes Skalarprodukt

```
PROCEDURE ScalarProduct (x, y : ARRAY 123 OF REAL) : REAL;
   VAR
      result : REAL; i : INTEGER;
BEGIN
   FOR i := 1 TO 123 DO
      result := result + x [i] * y [i]
   END;
   RETURN result
END ScalarProduct;
```

Sechs Mängel machen Programm 8.1 unbrauchbar. Markieren Sie in Programm 8.1 die Mängel mit (1) .. (6)! Beschreiben Sie unten die Mängel und nennen Sie mögliche Folgen und Änderungen, die die Mängel beseitigen!

Mangel	Folge	Änderung
(1)		
(2)		
(3)		
(4)		
(5)		
(6)		

Geben Sie eine korrekte, für beliebig große Vektoren verwendbare, effiziente Implementation der Funktion ScalarProduct an!

Programm 8.2
CP: Korrektes wiederverwendbares effizientes Skalarprodukt

Aufgabe 9 Rekursionsformel und Schleife konstruieren

(Punkte: 12 |.....) Geben Sie eine Rekursionsformel für die Glieder der Reihe

$$\sum_{i=0}^{n} \frac{x^i}{i!}, x \in \mathbb{R}$$

an und skizzieren Sie einen iterativen Algorithmus zur Berechnung der Reihe!
Hinweis: 0! = 1, $i!$ = $(i-1)!$ * i für $i > 0$.

Rekursionsformel

Algorithmus