

Informatik 1

Prüfung im Wintersemester 1997/98 - Musterlösung

- Aufgaben sind in Times- *Times-Kursiv*-, Courier- und **Courier-Fett**-, Lösungen in Arial-Kursiv- oder **Arial-Kursiv-Fett**-Schrift gehalten.

Fachhochschule für Technik und Wirtschaft Reutlingen, Fachbereich Elektronik

Prüfungsfach/Studiengang/Semester: Informatik 1 in Elektronik 1

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungsdauer: 120 Minuten

Zugelassene Hilfsmittel: Alle (Vorlesungsmitschrift, Literatur,...)

Anzahl der Aufgaben- und Lösungsblätter: 12

Aufgabe	Punkte		Aufgabe	Punkte	
	möglich	erreicht		möglich	erreicht
1	12		7	11	
2	21		8	11	
3	13		9	19	
4	10		10	11	
5	11		11	9	
6	10		12	13	
Summe möglich:		151	Summe erreicht:		

- Zum Bestehen der Prüfung sind 60, für die Note „Eins“ 120 Punkte erforderlich.
- Bearbeiten Sie die Aufgaben auf den dazu freigelassenen Stellen! Kreuzen Sie richtige Kästchen an, tragen Sie in Tabellen und an punktierten Stellen ... im Text passende Angaben ein! (Ausgenommen die Punktzahl jeweils rechts oben!) Falls der Platz nicht reicht, verwenden Sie die vorangehenden Blattrückseiten!

👉 **Viel Erfolg!**

Aufgabe 1: Gleitpunktzahlenarithmetik

(Punkte: 12 |.....)

Ein Rechner arbeite im Dezimalsystem (Basis $B = 10$). Zur Darstellung von Gleitpunktzahlen stehen für die Mantisse $m = 4$ Stellen und für den Exponenten 1 Stelle zur Verfügung, sowie jeweils eine Stelle für das Vorzeichen. Die Gleitpunktarithmetik runde unsymmetrisch (Abschneiden).

Wieviele Gleitpunktzahlenwerte sind damit darstellbar?

Ausdruck (nicht ausrechnen!): $2 * 9 * 10^3 * (2 * 9 + 1) + 1$

Wert der relativen Rechnergenauigkeit: $\delta_R = B^{1-m} = 10^{-3} = 0.001$

Näherungswert für absoluten Darstellungsfehler der Gleitpunktdarstellung von $1/3$: $0.0000333... \sim 0.3 * 10^{-4}$

x , y und z seien Gleitpunktzahlen mit den Werten $x = 0.3000E4$, $y = 0.7000E0$, $z = 0.2000E3$ und es sei der Ausdruck

$$(x + y) * z \quad \text{bzw.} \quad x * z + y * z$$

zu berechnen. Berechnen Sie in Gleitpunktarithmetik die Ausdrücke

$$gl(gl(x + y) * z) \quad \text{und} \quad gl(gl(x * z) + gl(y * z))!$$

$$gl(x + y) =$$

$$gl(0.3E4 + 0.7E0) = gl(0.3E4 + 0.00007E4) = gl(0.30007E4) = 0.3E4$$

$$gl(gl(x + y) * z) =$$

$$gl(0.3E4 * 0.2E3) = gl(0.06E7) = gl(0.6E6) = 0.6E6$$

$$gl(x * z) =$$

$$gl(0.3E4 * 0.2E3) = 0.6E6$$

$$gl(y * z) =$$

$$gl(0.7E0 * 0.2E3) = gl(0.14E3) = 0.14E3$$

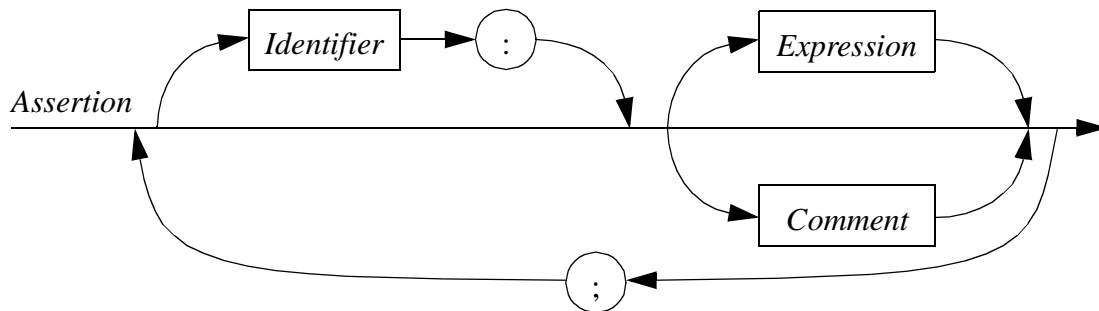
$$gl(gl(x * z) + gl(y * z)) =$$

$$gl(0.6E6 + 0.14E3) = gl(0.6E6 + 0.00014E6) = gl(0.60014E6) = 0.6001E6$$

Aufgabe 2: Syntaxbeschreibung

(Punkte: 21 |.....)

Stellen Sie das Syntaxdiagramm



mit zwei EBNF-Regeln dar, von denen die erste eine Iteration enthält!

Assertion = *AssertionClause* { „;“ *AssertionClause* }.

AssertionClause = [*Identifier* „:“] [*Expression* | *Comment*].

Zeichnen Sie zu folgenden nichtterminalen Einheiten der Syntax einer einfachen Assemblersprache in EBNF-Notation äquivalente Syntaxdiagramme!

Program = { *Instruction* }.

Instruction = *Opcode* [„:“ *Oplength*] [*Operand* [„:“ *Operand*]].

Opcode = *letter* { *letter* }.

Oplength = „B“ | „W“ | „L“.

Operand = *ident* | *integer* | „(“ *ident* „)“.

(Die lexikalischen Einheiten *letter*, *ident* und *integer* entsprechen denen von Oberon.)

Die Musterlösung verzichtet leider auf die Darstellung der Syntaxdiagramme.

Geben Sie zu folgenden Assembler-Instruktionen die nichtterminalen Bestandteile bzw. eine Syntaxfehlermeldung an!

MOVE.W source, target

Opcode . *Oplength* *Operand* , *Operand*

CMP.B (A1) , 9

Opcode . *Oplength* *Operand* , *Operand*

BEQ cond

Opcode *Operand*

ADD 1 TO Reg1

Opcode *Operand* FEHLER: „,“ erwartet!

NOP

Opcode

Aufgabe 4: Optimierung

(Punkte: 10 |.....)

Die gegebenen Programmstücke sind durch äquivalente Programmstücke zu ersetzen, d.h. ihre Semantik muß erhalten bleiben.

Vereinfachen Sie den folgenden Boole'schen Ausdruck so weit wie möglich!

$$\begin{aligned} & ((i \geq 4) \& \sim(k \leq 6)) \text{ OR } (\sim(i < 8) \& (k > 6)) \\ & ((i \geq 4) \& (k > 6)) \text{ OR } ((i \geq 8) \& (k > 6)) \\ & ((i \geq 4) \text{ OR } (i \geq 8)) \& (k > 6) \\ & (i \geq 4) \& (k > 6) \end{aligned}$$

Transformieren Sie die folgende Anweisung in drei Schritten in eine äquivalente Zuweisung!

```

IF ~a = TRUE THEN
  IF ~b = FALSE THEN
    c := ~b OR TRUE
  ELSE
    c := ~TRUE
  END
ELSE
  c := FALSE
END

```

Schritt 1:

```

IF ~a THEN
  IF b THEN
    c := TRUE
  ELSE
    c := FALSE
  END
ELSE
  c := FALSE
END

```

Schritt 2:

```

IF ~a THEN
  c := b
ELSE
  c := FALSE
END

```

Schritt 3:

```

c := ~a & b

```

Aufgabe 5: Schleifenterminierung und -transformation (Punkte: 11 |.....)

Begründen Sie zu jeder Schleife, warum sie abbricht bzw. warum nicht!

Schleife	terminiert / läuft endlos, weil...
<pre>i := 0; x := 1.0E10; WHILE (i < 9) & (x > 0) DO x := x - F (x); INC (i); END;</pre>	<p>Terminiert, weil <i>i</i> bei jeder Iteration erhöht wird und <i>i</i> >= 9 eine Abbruchbedingung ist, die nach endlich vielen Iterationen erfüllt ist. Außerdem gibt es eine weitere Abbruchbedingung.</p>
<pre>k := 0; FOR i := 0 TO 99 DO INC (k, i); END;</pre>	<p>Terminiert, weil es eine normale Zählschleife mit korrekt benutzter Zählvariable ist.</p>
<pre>i := 1; k := 2; REPEAT k := (k + i) DIV 2; UNTIL i < k;</pre>	<p>Läuft endlos, da zuerst $(k + i) \text{ DIV } 2 = (1 + 2) \text{ DIV } 2 = 1$, also $k = 1$ wird, und dann immer $(k + i) \text{ DIV } 2 = (1 + 1) \text{ DIV } 2 = 1$, also $k = 1$ invariant bleibt, also stets $i = k$, d.h. die Abbruchbedingung stets falsch ist.</p>

Die Programmiersprache Eiffel bietet nur ein Konstrukt für Wiederholungsanweisungen:

```
from initialeAnweisungen
until Abbruchbedingung
loop
  iterierteAnweisungen
end
```

Es ist semantisch äquivalent mit dem Oberon-Konstrukt

```
initialeAnweisungen;
WHILE ~Abbruchbedingung DO
  iterierteAnweisungen;
END;
```

Formulieren Sie die drei Oberon-Schleifen in obiger Tabelle mit dem Eiffel-Konstrukt!

```
from i := 0
  x := 1.0E10
until (i >= 9) or (x <= 0)
loop
```

```

    x := x - F(x)
    INC(i)
end
from k := 0
    i := 0
until i > 99
loop
    INC(k, i)
    INC(i)
end
from i := 1
    k := 2
    k := (k + i) div 2
until i < k
loop
    k := (k + i) div 2
end

```

Aufgabe 6: Zusicherungen

(Punkte: 10 |.....)

Tragen Sie in die Leerzeilen des folgenden Programmfragments möglichst starke Zusicherungen ein!

```

WHILE a > b DO
    Anweisungen1;
END;
ASSERT (a <= b);
IF x < 0 THEN
    ASSERT (x < 0);
    Anweisungen2;
ELSIF y = 0 THEN
    ASSERT ((x >= 0) & (y = 0));
    Anweisungen3;
ELSIF z >= 1 THEN
    ASSERT ((x >= 0) & (y # 0) & (z >= 1));
    Anweisungen4;
ELSE
    ASSERT ((x >= 0) & (y # 0) & (z < 1));
    Anweisungen5;
END;

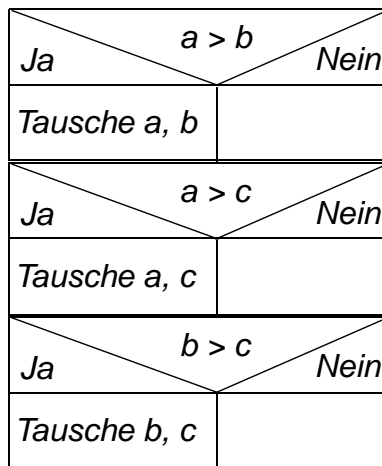
```

Aufgabe 7: Entwurf mit Struktogramm

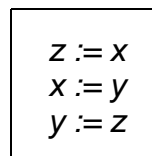
(Punkte: 11 |.....)

Entwerfen Sie mit einem Struktogramm einen Algorithmus, der die Werte von drei Variablen $a, b, c : \text{REAL}$ so vertauscht, daß anschließend $a \leq b$ und $b \leq c$ gilt! Schreiben Sie auch Vereinbarungen eventuell erforderlicher Hilfsvariablen hin!

Sortiere a, b, c :



Tausche x, y :



Vereinbarung der Hilfsvariablen:

VAR $x, y, z : \text{REAL}$

Aufgabe 8: Programmablaufverfolgung

(Punkte: 11 |.....)

Untersuchen Sie das Programmstück wie weiter unten beschrieben!

```

CONST nbr = 6;
VAR row : ARRAY nbr OF INTEGER; i1, i2, s, a : INTEGER;
... BEGIN
... (* Initialisierung der Variable row. *) ...
FOR i1 := 0 TO nbr - 2 DO
  s := i1;
  FOR i2 := i1 + 1 TO nbr - 1 DO
    IF row [s] > row [i2] THEN s := i2 END;
  END;
  a := row [i1];
  row [i1] := row [s];
  row [s] := a;
  (* 1 *)
END;
  (* 2 *)
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 (* 1 *) erreicht hat, aufgezeichnet. (Zwischenwerte der Variablen an anderen Programmstellen sind aus der Tabelle nicht ersichtlich.)

Die Programmausführung befindet sich an der Stelle (* 1 *). Verfolgen Sie den Programmablauf weiter und vervollständigen Sie die Tabelle, indem Sie jeweils den Zustand der Variablen beim Erreichen der Stelle (* 1 *) in eine neue Zeile eintragen, bis die Stelle (* 2 *) erreicht ist!

row						i1	i2	s	a
[0]	[1]	[2]	[3]	[4]	[5]				
8	34	79	12	17	45	0	1 2 3 4 5 6	0 5	45
8	12	79	34	17	45	1	2 3 4 5 6	1 3	34
8	12	17	34	79	45	2	3 4 5 6	2 3 4	79
8	12	17	34	79	45	3	4 5 6	3	34
8	12	17	34	45	79	4	5 6	4 5	79
						5			

Aufgabe 9: Funktionen

(Punkte: 19 |.....)

```

PROCEDURE PreMul2 (VAR i : INTEGER) : INTEGER;
BEGIN
    i := i * 2;
    RETURN i;
END PreMul2;

PROCEDURE PostMul2 (VAR i : INTEGER) : INTEGER;
    VAR k : INTEGER;
BEGIN
    k := i;
    i := i * 2;
    RETURN k;
END PostMul2;
    
```

Werten Sie folgende Ausdrücke aus, und zwar jeweils unter der Voraussetzung

VAR i : INTEGER; ... i := 1;

Ausdruck	mögliche Werte	
	des Ausdrucks	von i nach Auswertung des Ausdrucks
PreMul2 (i) + i	4, 3	2
i + PostMul2 (i)	2, 3	2
PreMul2 (i) - PostMul2 (i)	0, 3	4
PreMul2 (i) - PreMul2 (i)	-2, 2	4

Ersetzen Sie die Anweisung jeweils durch Anweisungsfolgen ohne Funktionsaufrufe!

Anweisung	äquivalente Anweisungsfolge
m := PreMul2 (n);	n := n * 2; m := n;
m := PostMul2 (n);	m := n; n := n * 2;

Welche Eigenschaft weisen die beiden Funktionen PreMul2 und PostMul2 auf, die sie von Funktionen im mathematischen Sinn unterscheidet?

Sie bewirken Seiteneffekte (über Referenzparameter).

Begründen Sie, warum die folgende Aussage richtig oder falsch ist!

*Der Übersetzer kann den Ausdruck PreMul2 (i) + PreMul2 (i) problemlos durch 2 * PreMul2 (i) ersetzen.*

Die Ausdrücke sind nicht äquivalent, da der Seiteneffekt zweimal bzw. einmal bewirkt wird.

Aufgabe 10: Prozedur Dreiecksart bestimmen

(Punkte: 11 |.....)

Programmieren Sie den Rumpf der Prozedur

```

PROCEDURE DreiecksartBestimmen
  (a, b, c           : REAL;
   OUT gleichseitig,
        gleichschenkelig,
        rechtwinklig : BOOLEAN);
BEGIN
  gleichseitig := (a = b) & (a = c);
  gleichschenkelig := (a = b) OR (a = c) OR (b = c);
  rechtwinklig := (a*a + b*b = c*c) OR (a*a + c*c = b*b) OR (b*b + c*c = a*a);
  ASSERT ( ~gleichseitig OR (gleichschenkelig & ~rechtwinklig) );
END DreiecksartBestimmen;

```

bei der die Parameter a, b, c die Seitenlängen eines Dreiecks bezeichnen, dessen Art bestimmt werden soll! Sie können hier exakte Arithmetik verwenden. Geben Sie eine möglichst starke Nachbedingung an die Ausgabeparameter an!

Aufgabe 11: Programmierfehlerquellen

(Punkte: 9 |.....)

Die Funktion

```

PROCEDURE F ( ) : REAL;
  VAR x : REAL;
BEGIN
  WHILE Math.Sin (x) / x # 1 DO
    x := x / 10;
  END;
  RETURN x;
END F;

```

enthält drei Schwachstellen, die sich als Fehlerursachen erweisen könnten. Spüren Sie diese auf, nennen Sie mögliche Folgen und geeignete Änderungen, um die Probleme zu beseitigen!

	Schwachstelle	mögliche Folge	Abhilfe
1.	Initialisierung von x fehlt	fehlerhafte Berechnung	$x := 1$
2.	Fortsetzungsbedingung zu schwach, exakter Vergleich von Gleitpunktzahlen	Endlosschleife	Test auf „fast gleich“
3.	Bei $\text{Sin}(x) / x$ u.U. Division durch 0	Programmabbruch	Test auf $x \neq 0$ einfügen

BEGIN

$x := 1;$

WHILE $(x > 0) \ \& \ \sim \text{MathArithR.Equal}(\text{Math.Sin}(x) / x, 1)$ **DO**

$x := x / 10;$

END;

RETURN $x;$

END F;

Aufgabe 12: Funktion Grenzwert

(Punkte: 13 |.....)

Programmieren Sie eine Funktion, die den Grenzwert des Ausdrucks

$$\frac{\log \sin \frac{2}{n}}{\log \sin \frac{1}{n}}$$

für $n \rightarrow \infty$ näherungsweise berechnet und als Ergebnis liefert! Verwenden Sie dabei die Funktionen Ln (für log) und Sin aus dem Bibliotheksmodul Math!

PROCEDURE $L () : \text{REAL};$

VAR $n : \text{LONGINT};$

$x, y : \text{REAL};$

BEGIN

$n := 0;$

$x := 0;$

$y := 1;$

WHILE $\sim \text{MathArithmeticsR.Equal}(x, y) \ \& \ (n < \text{MAX}(\text{LONGINT}))$ **DO**

$\text{INC}(n);$

$x := y;$

$y := \text{Math.Ln}(\text{Math.Sin}(2 / n)) / \text{Math.Ln}(\text{Math.Sin}(1 / n));$

END;

RETURN $y;$

END L;