

Informatik 1

Prüfung im Wintersemester 1997/98

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!):.....

Wert der relativen Rechnergenauigkeit:.....

Näherungswert für absoluten Darstellungsfehler der Gleitpunktdarstellung von $1/3$:.....

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

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

$$gl(x + y) =$$

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

$$gl(x * z) =$$

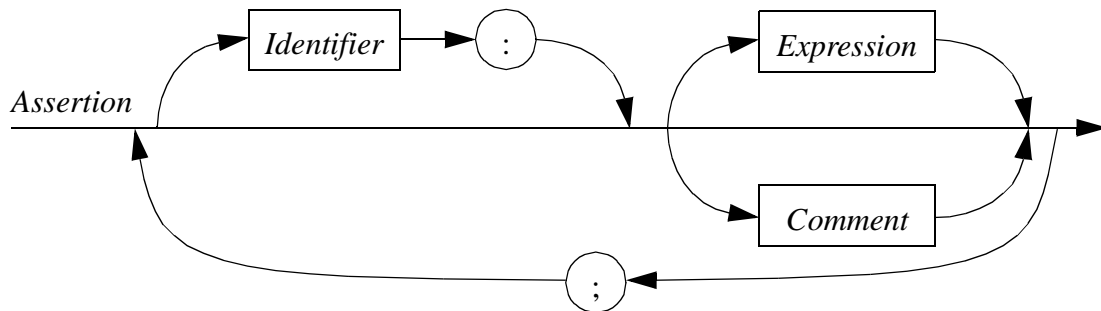
$$gl(y * z) =$$

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

Aufgabe 2: Syntaxbeschreibung

(Punkte: 21 |.....)

Stellen Sie das Syntaxdiagramm



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

Assertion =

AssertionClause =

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.)

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

MOVE.W source, target

CMP.B (A1), 9

BEQ cond

ADD 1 TO Reg1

NOP

Aufgabe 3: Gemischte Fragen

(Punkte: 13 |.....)

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

Aussage	Begründung
<i>Der Typ einer Variable ist zur Übersetzungszeit bestimmt.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>In einer Anweisung können Vereinbarungen vorkommen.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Ein formaler Parameter ist immer ein Wertparameter.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Bei einem Referenzparameter muß der aktuelle Parameter eine Variable sein.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil

Mit den Vereinbarungen

besteht die Anweisung

```
IF CHR (a) < b THEN c := a / 2 END
```

alle Prüfungen der Typkompatibilität.

Geben Sie mit den Vereinbarungen

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

zu jedem Teilausdruck des folgenden Ausdrucks an, von welchem Typ er ist und wo ein Typ implizit angepaßt wird, bzw. wo ein Typkompatibilitätsfehler vorliegt!

```
b & (ORD (c) + x < i)
```

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!

$$((i \geq 4) \ \& \ \sim(k \leq 6)) \ \mathbf{OR} \ (\sim(i < 8) \ \& \ (k > 6))$$

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
```

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>	
<pre> k := 0; FOR i := 0 TO 99 DO INC (k, i); END; </pre>	
<pre> i := 1; k := 2; REPEAT k := (k + i) DIV 2; UNTIL i < k; </pre>	

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!

Aufgabe 6: Zusicherungen

(Punkte: 10 |.....)

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

```
WHILE a > b DO
    Anweisungen1;
END;

IF x < 0 THEN

    Anweisungen2;
ELSIF y = 0 THEN

    Anweisungen3;
ELSIF z >= 1 THEN

    Anweisungen4;
ELSE

    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!

Aufgabe 8: Programmablaufverfolgung

(Punkte: 11 |.....)

Untersuchen Sie das Programmstück wie weiter unten beschrieben!

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		
i + PostMul2 (i)		
PreMul2 (i) - PostMul2 (i)		
PreMul2 (i) - PreMul2 (i)		

Ersetzen Sie die Anweisung jeweils durch Anweisungsfolgen ohne Funktionsaufrufe!

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

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

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.*

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
```

```
  ASSERT (.....);
```

```
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!

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`!