

Informatik 1

Prüfung im Sommersemester 2001

Name, Vorname:

Matrikelnummer:

Fachhochschule Reutlingen, Hochschule für Technik und Wirtschaft
Fachbereich Elektrotechnik und Maschinenbau

Prüfungsfach/Studiengang/Semester: Informatik 1 in Elektronik 1

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungstermin (Datum, Uhrzeit) und Ort: 12. 7. 2001, 8⁰⁰-10⁰⁰ Uhr, 5-013, 10-012

Prüfungsdauer: 120 Minuten

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

Anzahl der Aufgabenblätter: 9

Aufgabe	Punkte		Aufgabe	Punkte	
	möglich	erreicht		möglich	erreicht
1	10		6	19	
2	18		7	20	
3	19		8	15	
4	18		9	9	
5	12				
Summe möglich:		140	Summe erreicht:		

- Zum Bestehen der Prüfung sind 60, für die Note „Eins“ 120 Punkte erforderlich.
- Bearbeiten Sie die Aufgaben auf ggf. dazu freigelassenen Stellen! 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: Spezifikation durch Vertrag: Menge (Punkte: 10 |.....)

Ergänzen Sie die Spezifikation des Moduls, das eine Menge im Sinne der Mengenlehre modelliert, mittels Nachbedingungen und Invarianten! `Element` ist ein beliebiger Elementtyp. Die Aktionen sind idempotent, d.h. der Effekt von `Put(x)`; `Put(x)` ist gleich dem von `Put(x)`; der von `Remove(x)`; `Remove(x)` ist gleich dem von `Remove(x)`; der von `WipeOut`; `WipeOut` ist gleich dem von `WipeOut`.

MODULE Set

QUERIES

`IsEmpty` : BOOLEAN (** Enthält die Menge kein Element? **)

`Count` : INTEGER (** Wieviele Elemente enthält die Menge? **)

`Has (IN x : Element)` : BOOLEAN (** Enthält die Menge x? **)

POST

result ...

ACTIONS

`Put (IN x : Element)` (** Füge x zur Menge hinzu. **)

POST

...

NOT OLD (`Has (x)`) = (`Count` = **OLD** (`Count`) + 1)

...

`Remove (IN x : Element)` (** Entferne x aus der Menge. **)

POST

...

...

...

`WipeOut` (** Entferne alle Elemente aus der Menge. **)

POST

...

INVARIANTS

...

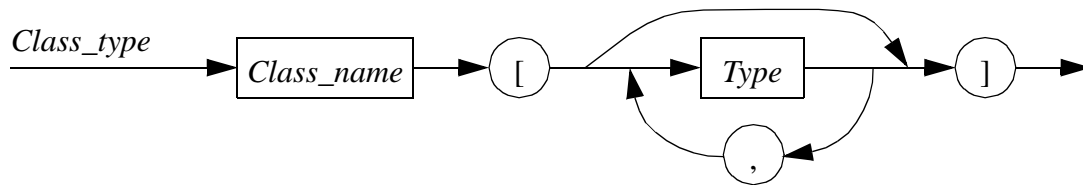
...

END Set

Aufgabe 2: Syntaxbeschreibung

(Punkte: 18 |.....)

Stellen Sie das Syntaxdiagramm



mit einer EBNF-Regel dar!

Class_type =

Gegeben ist die Syntax einer einfachen Kommandosprache in EBNF-Notation:

Command = *Pathname* [*Option*] { *Operand* }.

Pathname = *ident* { „,\" *ident* }.

Option = „/“ *letter* [„=“ *digit*].

Operand = *Pathname* | „*“.

Zeichnen Sie zu diesen vier Regeln äquivalente Syntaxdiagramme (links, Rückseite von Blatt 2)!

Ein Wort dieser Sprache ist z.B.: COPY /C=2 Stud\Mod *

Geben Sie eine Ableitung für dieses Wort an! (Es genügen fünf Hauptschritte, die jeweils mehrere einzelne Ersetzungen umfassen können.)

Aufgabe 3: Sprachkonstrukte, Typregeln

(Punkte: 19 |.....)

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

Aussage	Begründung / Bedingung
<i>Der Typ einer Variable ist zur Laufzeit veränderbar.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Typsicherheit bedeutet, dass jeder Typ sicher gegen Angriffe von Viren geschützt ist.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Ein Aufruf einer Prozedur kann in einem Ausdruck vorkommen,...</i>	..., wenn

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

Expressions are constructs denoting rules of computation whereby constants and current values of variables are combined to compute other values by the application of operators and function procedures. Expressions consist of operands and operators. Parentheses may be used to express specific associations of operators and operands.

Mit den Vereinbarungen

.....
besteht die Anweisung

```
IF b < 'c' THEN d := LEN (e) * 1.3 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, bzw. wo ein Typverträglichkeitsfehler vorliegt!

```
( x + 2 > i ) OR ( b = c )
```

Aufgabe 4: Optimierung

(Punkte: 18 |.....)

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

Vereinfachen Sie den folgenden booleschen Ausdruck so weit wie möglich!

$$((x > 6) \text{ OR } \sim(x < 7)) \& \sim(x \leq 5) \& (x \leq 8)$$

.....

Transformieren Sie die folgende Anweisung in eine äquivalente Anweisungsfolge, die um sechs Größenordnungen schneller läuft!

```
FOR i := 1 TO 1000000 DO
  IF k = i THEN
    x := 5 * k
  END;
y := x + 8
END
```

.....

Transformieren Sie die folgende Anweisung in drei bis vier Schritten und/oder mittels einer Wertetabelle in eine äquivalente Zuweisung!

```
IF ~(a = TRUE) THEN
  c := ~FALSE OR a
ELSE
  IF ~b # FALSE THEN
    c := ~TRUE & b
  ELSE
    c := ~b & TRUE
  END
END
```

END

Aufgabe 5: Schleifenablauf und -terminierung

(Punkte: 12 |.....)

Verfolgen Sie jeweils drei Durchläufe der beiden Schleifen und begründen Sie zu jeder Schleife, warum sie abbricht bzw. warum nicht!

Schleife 1		Schleife 2	
<pre>i := 100; k := 2; WHILE (i > 0) & (k > 1) DO i := i - k; k := k + 1 END</pre>		<pre>i := 2; k := 0; WHILE k <= 1 DO i := i DIV 2; k := k + i END</pre>	
i	k	i	k
100	2	2	0
<input type="checkbox"/> Terminiert <input type="checkbox"/> Läuft endlos, weil		<input type="checkbox"/> Terminiert <input type="checkbox"/> Läuft endlos, weil	

Aufgabe 7: Implementation von Funktionen zu Reihungen

(Punkte: 20 |.....)

Implementieren Sie die folgenden Funktionen!

Dabei ist Any ein beliebiger Typ und **CONST** notFound = -1.**PROCEDURE**Count (**IN** x : **ARRAY OF** Any; item : Any) : INTEGER;

(* Anzahl der Vorkommen von item in x.

Postcondition:

result is the number of indexes i with x [i] = item. *)

END Count ;**PROCEDURE**MaxIndexOf (**IN** x : **ARRAY OF** Any; item : Any) : INTEGER;

(* Index des letzten Vorkommens von item in x, falls existent;

sonst notFound.

Postcondition: (result = notFound) **OR**

(result is the largest index such that x [result] = item). *)

END MaxIndexOf ;

Aufgabe 8: Programmierfehlerquellen

(Punkte: 15 |.....)

Die Funktion

```
PROCEDURE Mean (IN a: ARRAY OF REAL; len: INTEGER): REAL;
  VAR   sum : REAL;   i : INTEGER;
BEGIN
  FOR i := 1 TO len DO
    sum := sum + a [i]
  END;
  RETURN sum / len
END Mean;
```

soll den arithmetischen Mittelwert der Reihung *a* berechnen; *len* ist die Elementanzahl von *a*. Die Funktion ist syntaktisch korrekt und übersetzbar, enthält aber *fünf Mängel*, die sich als Ursachen für Laufzeitfehler erweisen können.

Spüren Sie die Schwachstellen auf, nennen Sie mögliche *Folgen* und geeignete *Änderungen*, um die Funktion zuverlässig zu gestalten (links, Rückseite von Blatt 8)!

Aufgabe 9: Programmierrichtlinien

(Punkte: 9 |.....)

Viele Bücher über Programmiertechnik enthalten Empfehlungen und Regeln für guten Programmierstil. Durch Ihre Programmiererfahrung haben Sie sich Leitlinien angeeignet, an die Sie sich selbst beim Programmieren halten.

Formulieren Sie drei Ihrer Programmierrichtlinien mit eigenen Worten und begründen Sie jeweils, welchem gutem Zweck die Richtlinie dient!