

# Informatik 1

## mki-Bachelor 1 Alte Prüfungsaufgaben

Prüfer: Prof. Dr. Karlheinz Hug  
Prüfungsdauer: 120 Minuten  
Zugelassene Hilfsmittel: Alle (Literatur, Skripten, Übungsmaterial,...)  
Umfang: 24 Aufgaben auf 21 Seiten

- Zum Bestehen der Prüfung sind **60**, für die Note „1.0“ **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.



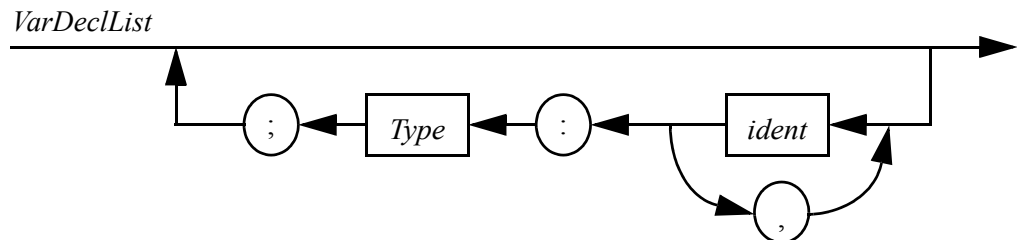
**Viel Erfolg!**

### Aufgabe 1

(Punkte: 4 |.....)  
WS 05/06

### Syntax beschreiben

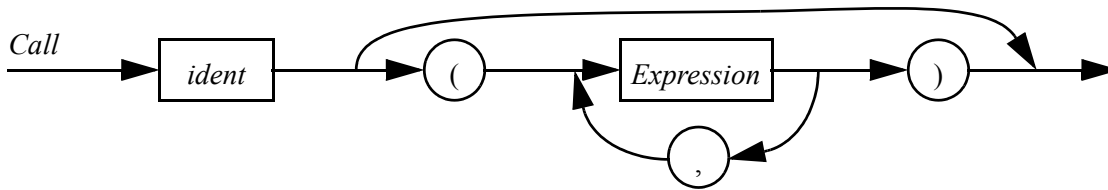
Stellen Sie das folgende Syntaxdiagramm mit einer EBNF-Regel dar!



*VarDeclList* =

## Aufgabe 2 Syntax beschreiben

(Punkte: 18 |.....) (1) Stellen Sie das folgende Syntaxdiagramm mit einer EBNF-Regel dar!  
 WS 03/04



Call =

(2) Gegeben ist die Syntax einer Internetadresse in EBNF-Notation:

*URL* = *Protocol* „!“ *Hostname* [ „!“ *Pathname* ].

*Protocol* = „file“ | „ftp“ | „http“.

*Hostname* = *ident* „.“ ( „de“ | „com“ ).

*Pathname* = *ident* { „!“ *ident* }.

(lexikalische Einheit: *ident* = *letter* { *letter* | *digit* | „-“ | „.“ }.)

Zeichnen Sie zu den ersten vier dieser Regeln äquivalente Syntaxdiagramme!

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

<http://www.mki.reutlingen-university.de/i1.htm>

### Aufgabe 3 Invarianten eines Vierecks bestimmen

(Punkte: 13 |.....)  
 WS 04/05

Bei einem Viereck kann man die vier Seiten, die vier Winkel (im Bogenmaß), den Umfang, die Fläche und ob es ein Drachen (*kite*, zwei Paar Nachbarseiten gleich lang), ein Parallelogramm (*parallelogram*, gegenüberliegende Seiten gleich lang), eine Raute (*rhombus*, vier Seiten gleich lang), ein Rechteck (*rectangle*) oder ein Quadrat (*square*) ist, abfragen. Ergänzen Sie in der folgenden Schnittstelle 13 fehlende Invarianten und korrigieren Sie 13 fehlerhafte! (pi ist die Kreiszahl.)

**Programm 3.1**  
 Cleo: Viereck

INTERFACE Quadrangle

QUERIES

(\* Seiten: \*)

a : REAL  
 b : REAL  
 c : REAL  
 d : REAL

(\* Winkel: \*)

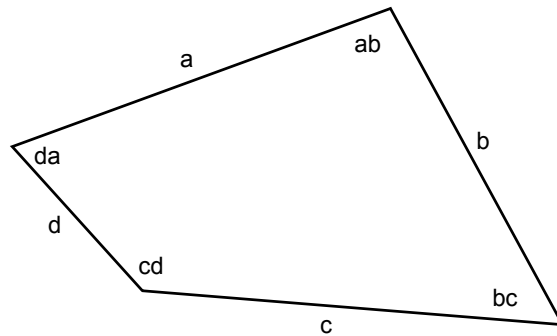
ab : REAL  
 bc : REAL  
 cd : REAL  
 da : REAL

(\* Maße: \*)

perimeter : REAL  
 area : REAL

(\* Eigenschaften: \*)

isKite : BOOLEAN  
 isParallelogram : BOOLEAN  
 isRhombus : BOOLEAN  
 isRectangle : BOOLEAN  
 isSquare : BOOLEAN



INVARIANTS

(\* Bedingungen an Seiten: \*)

a > 0;  
 b > 0;  
 c <= 0;

a + b + c > d;  
 b + c + d > a;

(\* Bedingungen an Winkel: \*)

ab > 0;

da = 0;

bc < pi;

ab + bc + da = 999 \* pi;

(\* Bedingungen an Umfang und Fläche: \*)

perimeter = a + d;

(\* Bedingungen an Eigenschaften und Seiten: \*)

isKite = ((a = b) AND (c = d)) OR ((a = d) AND (b # c));

isRhombus = (a = b) OR (b = c) AND (c = d);

isRectangle IMPLIES ((a = c) AND (b = d));

(\* Bedingungen an Eigenschaften und Winkel: \*)

isKite IMPLIES ((ab = cd) OR (bc = da));

isParallelogram = (ab = cd) AND (bc = da);

isRhombus IMPLIES ((ab = cd) AND (bc = 3 \* pi));

isRectangle = (ab = bc) AND (bc = cd);

isRectangle = (ab = pi/2) AND (bc = pi/2) OR (cd = pi/2);

isSquare IMPLIES ((ab = bc) OR (bc = cd) OR (cd = da));

(\* Zusammenhänge zwischen Eigenschaften: \*)

isRhombus = isKite AND isSquare;

isRectangle IMPLIES isRhombus;

isSquare = isRhombus OR isRectangle

END Quadrangle

### Aufgabe 4      Zusicherungen ergänzen

(Punkte: 7 |.....)  
SS 98

Tragen Sie in das folgende Programmfragment möglichst starke Zusicherungen ein!

```

WHILE ~Stack.IsEmpty () DO
    Anweisungen1
END;

ASSERT (.....);
IF x <= 1 THEN

    ASSERT (.....);
    Anweisungen2
ELSIF x > 9 THEN

    ASSERT (.....);
    Anweisungen3
ELSIF x = 8 THEN

    ASSERT (.....);
    Anweisungen4
ELSE

    ASSERT (.....);
    Anweisungen5
END
    
```

### Aufgabe 5      Fachenglisch übersetzen

(Punkte: 5 |.....)  
WS 05/06

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

*An **array** is a structure consisting of a number of elements which are all of the same type, called the **element type**. The number of elements of an array is called its **length**. The elements of the array are designated by indices, which are integers between 0 and the length minus 1.*

### Aufgabe 6 Sprachkonstrukte beschreiben

(Punkte: 16 |.....)  
 SS 98  
 WS 98/99  
 WS 03/04  
 WS 04/05  
 WS 05/06

Begründen Sie, warum die folgenden Aussagen für *Component Pascal* richtig oder falsch sind!

Aussage	Begründung
<i>Ein aktueller Parameter ist immer eine Variable.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Eine Zusicherung kann Vereinbarungen enthalten.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Ein Aufruf einer Prozedur kann in einem Ausdruck vorkommen.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Vereinbarungen sind Anweisungen.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Anweisungen sind Ausdrücke.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Die Typen aktueller Parameter werden zur Laufzeit ermittelt.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Die Typen von Ausdrücken werden zur Laufzeit ermittelt.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil
<i>Die Typen von Formalparametern werden zur Laufzeit ermittelt.</i>	<input type="checkbox"/> Richtig <input type="checkbox"/> Falsch, weil

**Aufgabe 7**      **Typregeln anwenden**

(Punkte: 12 |.....)  
SS 98  
WS 98/99

Mit den Vereinbarungen

besteht die Anweisung

```
IF ORD (a) < b MOD 3 THEN c := b * 1.2 END
```

alle Prüfungen der Typverträglichkeit.

Mit den Vereinbarungen

besteht die Anweisung

```
IF x < 'y' THEN z := ORD (x) * 3.4 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!

```
(( i < x ) OR ( c = CHR (i) )) & ( b + 1 )
```

## Aufgabe 8 Programmstücke verbessern

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

(Punkte: 2 |.....)  
WS 01/02

Ersetzen Sie die folgende Auswahlanweisung durch *eine* äquivalente Zuweisung! (Es gilt  $i, k, n : \text{INTEGER}$ .)

```
ASSERT ((-1 <= i) & (i <= n) & (0 < n));
IF i = -1 THEN
  k := n - 1
ELSIF i = n THEN
  k := 0
ELSE
  k := i
END
```

(Punkte: 14 |.....)  
WS 05/06

Ersetzen Sie in der folgenden Funktion zum Prüfen der Primeigenschaft einer Ganzzahl die ineffiziente Zählschleife durch eine Anweisungsfolge, die berücksichtigt, dass

- es sich um ein Suchproblem handelt;
- jede gerade Zahl außer 2 keine Primzahl ist;
- wenn  $n$  durch  $i$  teilbar ist, es ein  $k$  gibt mit  $n = i * k$  und ( $i \leq k$  oder  $k \leq i$ );
- die Wurzelfunktion nicht benutzt werden darf.

**Programm 8.1**  
CP: Primeigenschaft  
prüfen

```
PROCEDURE IsPrime (n : INTEGER) : BOOLEAN;
  VAR
    result : BOOLEAN;
    i      : INTEGER;
  BEGIN
    IF n <= 1 THEN
      result := FALSE
    ELSE
      result := TRUE;
      FOR i := 2 TO n - 1 DO
        IF n MOD i = 0 THEN
          result := FALSE
        END
      END
    END;
    RETURN result
  END IsPrime;
```

(\* Zu ersetzen! \*)

Den obigen Kasten ersetzende Anweisungsfolge:

(Punkte: 11 |.....)  
WS 06/07

Verbessern Sie die Implementation der Funktion F schrittweise so, dass sie halb so lang ist und doppelt so schnell läuft!

**Programm 8.2**  
CP: Schwache  
Implementation

```
PROCEDURE F (n : INTEGER) : REAL;  
  VAR  
    i, m : INTEGER;  
    x : REAL;  
  BEGIN  
    i := 123 * n;  
    m := -n;  
    x := 0;  
    FOR i := m TO n DO  
      IF i < 0 THEN  
        x := x - 3 / i  
      ELSIF i > 0 THEN  
        x := x + 3 / i  
      END;  
      m := 2 * i + 1  
    END;  
    RETURN x / 6  
  END F;
```

### Aufgabe 9 Schleifenterminierung prüfen und Schleifen transformieren

(Punkte: 16 |.....) Begründen Sie zu jeder Schleife, warum sie abbricht oder nicht abbricht!  
 WS 98/99

Schleife	terminiert / läuft endlos, weil...
<pre>i := 100; x := 2; WHILE (i &gt; 0) &amp; (x &gt; 1) DO   i := i - ENTIER (x);   x := F (x) END</pre>	
<pre>k := 123; FOR i := k TO 2 * k DO   k := k + i END</pre>	
<pre>x := 1; y := 0; REPEAT   x := x / 2;   y := y + x UNTIL y &gt; 1</pre>	

Die Programmiersprachen *C*, *C++*, *Java* bieten ein Konstrukt für Wiederholungsanweisungen, das vereinfacht so aussieht:

```
for (initAnweisung; Fortsetzungsbedingung; termAnweisung)
{
  iterierteAnweisungen
}
```

Es ist semantisch äquivalent mit dem *Component-Pascal*-Konstrukt

```
initAnweisung;
WHILE Fortsetzungsbedingung DO
  iterierteAnweisungen;
  termAnweisung
END
```

Eine Anweisung kann leer sein.

Formulieren Sie die zweite (FOR) und die dritte (REPEAT) *Component-Pascal*-Schleife in obiger Tabelle mit der *C*-for-Schleife!

### Aufgabe 10 Schleifenablauf und -terminierung prüfen

(Punkte: 5 |.....)  
WS 03/04

Verfolgen Sie zur Schleife

```
p := 16;
q := 59;
r := 0;
WHILE q > 0 DO
  q := q - 1;
  r := r + p
END
```

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

Terminiert  Läuft endlos, weil

q	r
59	0

### Aufgabe 11 Schleife umformen

(Punkte: 12 |.....)  
WS 06/07

Transformieren Sie die rumpfgesteuerte Bedingungschleife

```
LOOP
  A;
  IF B THEN
    EXIT
  END;
  C;
  IF D THEN
    EXIT
  END;
  E
END;
ASSERT (B OR D)
```

in eine äquivalente kopfgesteuerte Bedingungschleife mit der zusätzlichen Variablen

```
VAR done : BOOLEAN;
```

```
ASSERT (B OR (done & D))
```

und schreiben Sie neben jede Variante ihre Stärken (+) und Schwächen (-) hin!

### Aufgabe 12 Schleifenablauf und -terminierung prüfen

(Punkte: 14 |.....)  
WS 05/06

Zu jeder der beiden Schleifen sollen Sie sechs Durchläufe verfolgen, möglichst starke Zusicherungen ergänzen, und begründen, warum sie abbricht oder nicht abbricht!

Schleife 1		Schleife 2		
<pre>p := 18; q := 99; r := 9; WHILE (q # 0) OR (r # 0) DO   q := q - 1;   r := (r + p) MOD 10;    ASSERT(.....) END</pre>		<pre>p := 12; q := 3; r := 4; WHILE p &gt; 0 DO   IF q &lt;= r THEN     p := p - q;     q := q + r   ELSE     r := 2 * q;     p := p + r   END;   ASSERT(.....)   .....   ..... END</pre>		
q	r	p	q	r
99	9	12	3	4
<input type="checkbox"/> Terminiert <input type="checkbox"/> Läuft endlos, weil		<input type="checkbox"/> Terminiert <input type="checkbox"/> Läuft endlos, weil		

### Aufgabe 13 Programmablauf verfolgen

(Punkte: 7 |.....) Untersuchen Sie Programm 13.1 wie unten beschrieben!

**Programm 13.1**  
 CP: Unbekannter  
 Algorithmus

```

CONST
    number = 8;
VAR
    row                : ARRAY number OF INTEGER;
    thisLow, thisHigh, thisSum,
    bestLow, bestHigh, bestSum : INTEGER;
... BEGIN
... (* Initialisierung der Variablen. *) ...
FOR thisHigh := 1 TO number - 1 DO
    IF thisSum <= 0 THEN
        thisLow := thisHigh;
        thisSum := 0;
    END;
    INC (thisSum, row [thisHigh]);
    IF thisSum > bestSum THEN
        bestLow := thisLow;
        bestHigh := thisHigh;
        bestSum := thisSum;
    END;
END
(* 1 *)
END
(* 2 *)
END ...
    
```

Die folgenden Tabellen ordnen jeder Variablen eine Spalte zu. In den Zeilen (außer den Kopfzeilen) 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							
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
7	-9	5	1	-7	3	9	-1

thisLow	thisHigh	thisSum	bestLow	bestHigh	bestSum
2	3	6	0	0	7

### Aufgabe 14

(Punkte: 14 |.....)  
WS 98/99

**Programm 14.1**  
CP: Zurückgeben und Setzen

### Seltene Funktionen untersuchen

```
PROCEDURE PreSet (OUT i : INTEGER; k : INTEGER) : INTEGER;
BEGIN
    i := k;
    RETURN i
END PreSet;
```

```
PROCEDURE PostSet (VAR i : INTEGER; k : INTEGER) : INTEGER;
    VAR n : INTEGER;
BEGIN
    n := i;
    i := k;
    RETURN n
END PostSet;
```

Werten Sie folgende Ausdrücke aus, und zwar *jeweils einzeln* unter der Voraussetzung  
VAR i : INTEGER; ... i := 1

Ausdruck	mögliche Werte	
	des Ausdrucks	von i nach Auswertung des Ausdrucks
PreSet (i, 2) + i		
i + PostSet (i, 2)		
PostSet (i, 2) + PreSet (i, 3)		

Ersetzen Sie die Anweisung jeweils durch Anweisungsfolgen ohne Funktionsaufrufe!

Anweisung	äquivalente Anweisungsfolge
m := PreSet (n, 2)	
m := PostSet (n, 2)	

Warum dürfen die Funktionen PreSet, PostSet nicht wie „normale“ Abfragen in Zusicherungen aufgerufen werden?

### Aufgabe 15 Programmiermängel suchen und beseitigen

(Punkte: 15 |.....)  
WS 01/02

Die folgende Funktion soll feststellen, ob die Reihungen x und y elementweise gleich sind. Dabei ist Any ein beliebiger einfacher Typ.

**Programm 15.1**  
CP: Defekter  
Vektorvergleich

```
PROCEDURE Equal (IN x, y : ARRAY OF Any) : BOOLEAN;
  VAR
    i : INTEGER;
  BEGIN
    i := 1;
    WHILE (x [i] = y [i]) & (i <= LEN (x)) DO
      i := i + 1
    END;
    RETURN i > LEN (y)
  END Equal;
```

Programm 15.1 ist syntaktisch korrekt und übersetzbar, enthält aber *fünf Mängel*, die es unbrauchbar machen. Markieren Sie in Programm 15.1 die Mängel mit (1) .. (5)! Beschreiben Sie unten die Mängel und nennen Sie mögliche Folgen und Änderungen, die die Mängel beseitigen!

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

**Aufgabe 16****Prozedur implementieren**(Punkte: 12 |.....)  
SS 98

Programmieren Sie den Rumpf der folgenden Prozedur, bei der die Größen *ab*, *bc*, *ca* die Winkel eines Dreiecks bezeichnen, dessen Art sie bestimmen soll! *pi* ist die Kreiszahl. Sie dürfen hier exakte Arithmetik verwenden. Geben Sie eine möglichst schwache Vorbedingung an die Eingabeparameter an, die alle fehlerhaften Aufrufe abfängt!

**Programm 16.1**

CP: Dreiecksart bestimmen

```
PROCEDURE DreiecksartBestimmen
  (ab, bc : REAL; OUT gleichseitig, gleichschenkelig, rechtwinklig : BOOLEAN);
  VAR ca : REAL;
  BEGIN
    ASSERT (
```

```
      ASSERT (~gleichseitig OR (gleichschenkelig & ~rechtwinklig))
    END DreiecksartBestimmen;
```

**Aufgabe 17****Prozedur implementieren**

(Punkte: 7 |.....)

Die *Cleo*-Schnittstelle**Programm 17.1**

Cleo: Uhr

```
INTERFACE Clock
  QUERIES
    hour, minute, second : NATURAL
  INVARIANTS
    hour < 24; minute < 60; second < 60
  ACTIONS
    Tick
    POST
      second = (OLD (second) + 1) MOD 60;
      (second = 0) IMPLIES (minute = (OLD (minute) + 1) MOD 60);
      ((second = 0) AND (minute = 0)) IMPLIES (hour = (OLD (hour) + 1) MOD 24)
    END
END Clock
```

END Clock

spezifiziert eine Uhr. Implementieren Sie Tick als Prozedur in *Component Pascal*!**Programm 17.2**

CP: Ticken

```
PROCEDURE Tick;
  BEGIN
```

```
  END Tick;
```

**Aufgabe 18****Algorithmus entwerfen**

(Punkte: 14 |.....)  
WS 98/99

Entwerfen Sie einen Algorithmus (mit einem Struktogramm oder in Pseudocode), der in einer mit positiven Ganzzahlen initialisierten Reihung

a : ARRAY n OF INTEGER

die Elemente mit geradem Wert halbiert und die anderen verdreifacht und um eins erhöht, und dies so lange wiederholt, bis eines der Elemente den Wert eins hat! Schreiben Sie ggf. Vereinbarungen benötigter Hilfsvariablen hin!

**Aufgabe 19****Datentyp und Algorithmus entwerfen**

(Punkte: 11 |.....)  
SS 00

- (1) Modellieren Sie die Punktetabelle auf dem Deckblatt einer veröffentlichten Prüfung als einen Verbundtyp! Schreiben Sie geeignete Vereinbarungen hin, die folgende Regeln erfüllen:
  - ◆ Alle vorkommenden Konstanten außer 0 und 1 haben einen Namen.
  - ◆ Es kommen keine anonymen Typen vor.
  - ◆ Die Vereinbarungen sind kommentarlos verständlich.
- (2) Entwerfen Sie einen Algorithmus (mit einem Struktogramm oder in Pseudocode), der den Wert eines Summenfelds berechnet und setzt!
- (3) Programmieren Sie zu dem Verbundtyp von (1) eine Prozedur, die den Algorithmus von (2) zur Berechnung der Summe der möglichen Punkte benutzt!

**Aufgabe 20 Funktion für Grenzwert programmieren**

(Punkte: 13 |.....) Programmieren Sie eine Funktion, die den Grenzwert des Ausdrucks

$$n + 1 - 1 / (\log(n + 1) - \log n)$$

für  $n \rightarrow \infty$  näherungsweise berechnet und als Ergebnis liefert! Verwenden Sie dabei die Funktion Log aus dem Bibliotheksmodul Math!

**Programm 20.1**

CP: Grenzwert  
berechnen

**Aufgabe 21 Funktion für Reihe programmieren**

(Punkte: 15 |.....)  
WS 98/99 Programmieren Sie eine Funktion, die den Wert der unendlichen Reihe

$$\sum_{i=0}^{\infty} \frac{1}{(2i+1)^2}$$

näherungsweise berechnet und als Ergebnis liefert! Der exakte Wert der Reihe ist  $\pi^2/8$ . Formulieren Sie eine Zusicherung, die die semantische Korrektheit der Funktion prüft!

**Programm 21.1**

CP: Reihe berechnen

**Aufgabe 22****Zeichenmengenmodul implementieren**

(Punkte: 23 |.....)  
SS 00

Ein Modul für eine Menge von Zeichen ist mit einer booleschen Reihung implementiert:

**Programm 22.1**  
CP: Zeichenmenge

```

MODULE ContainersSetOfChar;
  IMPORT ...
  VAR has : ARRAY ORD (MAX (CHAR)) - ORD (MIN (CHAR)) + 1 OF BOOLEAN;
  (* Queries *)
  PROCEDURE IsEmpty* () : BOOLEAN;
    VAR
      result : BOOLEAN;
      i      : INTEGER;
    BEGIN
      result := ~has [0];
      i      := 1;
      WHILE result & (i < LEN (has)) DO
        result := ~has [i];
        INC (i)
      END;
      RETURN result
    END IsEmpty;
  PROCEDURE Has* (x : CHAR) : BOOLEAN; ...
  (* Actions *) ...
END ContainersSetOfChar.

```

- (1) Verbessern Sie die Implementation der Abfrage **IsEmpty**, sodass sich ihre mittlere Laufzeit auf etwa die Hälfte verkürzt!
- (2) Erweitern Sie das Modul um eine Abfrage **Count**\* () : INTEGER, die die Anzahl der Elemente in der Menge durch Berechnung liefert!
- (3) Formulieren Sie zwei Invarianten des Mengenmoduls als boolesche Ausdrücke, in denen die neue Abfrage **Count** vorkommt!

**Aufgabe 23**(Punkte: 8 |.....)  
WS 03/04**Programm 23.1**  
CP: Zwillings-  
eigenschaft prüfen**Funktion strukturiert implementieren**

Implementieren Sie einen strukturierten Algorithmus zur folgenden Funktion, die untersucht, ob ihr Parameter „Zwillingsziffern“ hat!

PROCEDURE HasSiblings (n : INTEGER) : BOOLEAN;

(\*)

*Gibt es zwei gleiche, direkt benachbarte Ziffern in der Dezimaldarstellung von n?**Beispiele:*

HasSiblings (12341) = FALSE;

HasSiblings (12234) = TRUE;

HasSiblings (12223444) = TRUE.

\*)

END HasSiblings;

**Aufgabe 24**

(Punkte: 12 |.....)

**Programm 24.1**  
CP: Fehlstellung  
suchen**Funktion strukturiert implementieren**

Implementieren Sie einen strukturierten Algorithmus zur folgenden Funktion! Dabei ist Comparable ein beliebiger Typ, für den die Ordnungsrelationen &lt;, &gt;, &lt;=, &gt;= definiert sind, und CONST notFound = -1.

PROCEDURE MinIndexOfUnsorted (IN x : ARRAY OF Comparable) : INTEGER;

(\*)

*Index des ersten Vorkommens eines Elements in x,**das kleiner als das vorhergehende Element ist, falls existent; sonst notFound.***Postcondition:**

(result = notFound) OR (result is the smallest index such that x [result] &lt; x [result - 1]).

\*)

END MinIndexOfUnsorted;