



Musterlösung

Aufgaben sind in Times-, *Lösungen in blauer Monotype-Corsiva-Schrift gehalten (in der PDF-Variante in einer Ersatzschrift).*

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungstermin und Ort: Mittwoch, 6. Juli 2005, 10³⁰ bis 12³⁰ Uhr, Aula

Prüfungsdauer: 120 Minuten

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

Anzahl der Aufgabenseiten: 10

Aufgabe	Punkte		Aufgabe	Punkte	
	möglich	erreicht		möglich	erreicht
1	18		5	20	
2	26		6	25	
3	44		7	15	
4	16				
Summe möglich:		164	Summe erreicht:		
			Note:		

- Zum Bestehen der Prüfung sind **60**, für die Note „Eins“ **120** Punkte erforderlich.
- 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.
- Geben Sie alle erhaltenen Blätter zusammengeheftet wieder ab!



Viel Erfolg!

Aufgabe 1 Programmablaufverfolgung bei Rekursion

(Punkte: 18 |.....) Verfolgen Sie die Ausführung der folgenden rekursiven Funktion bei den unten gegebenen Werten von row; tragen Sie die Werte der aktuellen Parameter, der lokalen Variablen und der Funktionsaufrufergebnisse in die Tabelle ein!

```

VAR row : ARRAY 17 OF INTEGER;
PROCEDURE IndexOf (x, le, ri : INTEGER) : INTEGER;
    VAR m : INTEGER;
BEGIN
    ASSERT ((0 <= le) & (ri < LEN (row)), BEC.precondition);
    ASSERT (Sorted (row), BEC.precondition);
    IF le > ri THEN
        RETURN -1
    ELSE
        m := (le + ri) DIV 2;
        IF x < row [m] THEN
            RETURN IndexOf (x, le, m - 1)
        ELSIF x > row [m] THEN
            RETURN IndexOf (x, m + 1, ri)
        ELSE
            RETURN m
        END
    END
END
END IndexOf;
    
```

row																
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]
3	6	12	15	24	27	32	34	41	43	54	57	65	68	76	80	99

Aufruffolge	Werte der aktuellen Parameter			Wert von	Ergebniswert
IndexOf	x	le	ri	m	result
1. Aufruf	55	0	16	8	-1
2. Aufruf	55	9	16	12	-1
3. Aufruf	55	9	11	10	-1
4. Aufruf	55	11	11	11	-1
5. Aufruf	55	11	10		-1

Verbalisieren Sie, was die Funktion liefert!

Falls x in row [le .. ri] enthalten: ein Index i mit row [i] = x; sonst: -1.

Formalisieren Sie mit einer Nachbedingung, was die Funktion liefert!

(result = -1) OR ((le <= result) AND (result <= ri) AND (row [result] = x))

Aufgabe 2 Warteschlange mit zwei Kellern realisieren

(Punkte: 26 |.....) Eine **Warteschlangenklasse** mit der bekannten Schnittstelle

```
QueueDesc = RECORD
  (VAR q : QueueDesc) Init, NEW;
  (IN q : QueueDesc) IsEmpty () : BOOLEAN, NEW;
  (IN q : QueueDesc) Item () : Element, NEW;
  (VAR q : QueueDesc) Put (x : Element), NEW;
  (VAR q : QueueDesc) Remove, NEW;
END;
```

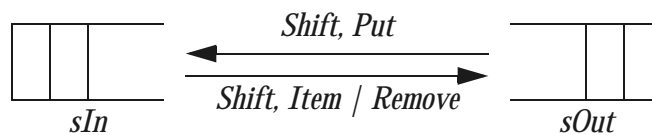
und **First-In-First-Out**-Semantik ist zu realisieren. Als Objektdaten sind nur zwei Objekte einer **Kellerklasse** mit der bekannten Schnittstelle

```
StackDesc = RECORD
  (VAR s : StackDesc) Init, NEW;
  (IN s : StackDesc) IsEmpty () : BOOLEAN, NEW;
  (IN s : StackDesc) Item () : Element, NEW;
  (VAR s : StackDesc) Put (x : Element), NEW;
  (VAR s : StackDesc) Remove, NEW;
END;
```

und **Last-In-First-Out**-Semantik zugelassen. Element ist ein beliebiger Elementtyp.

Visualisieren und verbalisieren Sie die **Idee** zur Implementation der Warteschlangenklasse und skizzieren Sie die **Algorithmen** ihrer Schnittstellenoperationen!

Idee



*Für Put:
alle Elemente nach sIn,
neues Element auf sIn
legen.*

*Für Item, Remove:
alle Elemente nach sOut, oberstes Element von sOut abfragen bzw. löschen.*

Algorithmen

VAR sIn, sOut : StackDesc

Init:

*sIn.Init
sOut.Init*

IsEmpty:

RETURN sIn.IsEmpty AND sOut.IsEmpty

Item:

*Shift (sIn, sOut)
RETURN sOut.Item*

Put (x):

*Shift (sOut, sIn)
sIn.Put (x)*

Remove:

*Shift (sIn, sOut)
sOut.Remove*

Shift (INOUT from, to : StackDesc):

*WHILE NOT from.IsEmpty DO
to.Put (from.Item)
from.Remove*

END

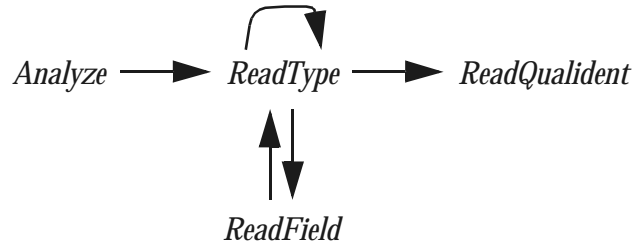
Aufgabe 3 Parser mit rekursivem Abstieg

(Punkte: 44 |.....) Zur Sprache, deren Syntax durch die EBNF-Regeln

```
Type      = Qualident
            | RECORD [ "(" Qualident ")" ] Field { ";" Field } END
            | POINTER TO Type.
Field      = [ ident ":" Type ].
Qualident  = ident [ "." ident ].
```

gegeben ist, ist ein Parser mit rekursivem Abstieg zu entwerfen. Stellen Sie die **Aufrufbeziehungen** der benötigten Prozeduren in einem Diagramm dar und skizzieren Sie die **Algorithmen** der Prozeduren mit den unten angegebenen Vereinbarungen!

Aufrufbeziehungen



Algorithmen

symbol zuletzt gelesenes Symbol mit möglichen Werten record, leftParenthesis, rightParenthesis, semicolon, end, pointerTo, ident, colon (:), period, illegal
 next Prozedur, die nächstes Symbol liest und symbol den passenden Wert zuweist
 first (E) Menge der Startsymbole des EBNF-Ausdrucks E
 error Prozedur, die Syntaxfehler meldet

Analyze:

```
next
IF symbol IN {ident, record, pointerTo} THEN ReadType ELSE error END
```

ReadType:

```
IF symbol = ident THEN
  ReadQualident
ELSIF symbol = record THEN
  next
  IF symbol = leftParenthesis THEN
    next
    ReadQualident
    IF symbol = rightParenthesis THEN next ELSE error END
  END
  ReadField
  WHILE symbol = semicolon DO
    next
    ReadField
  END
  IF symbol = end THEN next ELSE error END
ELSIF symbol = pointerTo THEN
  next
  ReadType
ELSE
  error
END
```

ReadField:

```
IF symbol = ident THEN  
  next  
  IF symbol = colon THEN next ELSE error END  
  ReadType  
END
```

ReadQualident:

```
IF symbol = ident THEN next ELSE error END  
IF symbol = period THEN  
  next  
  IF symbol = ident THEN next ELSE error END  
END
```

Aufgabe 4 Zeiger

(Punkte: 16 |.....) Geben Sie unter der Voraussetzung der Vereinbarungen

TYPE A = POINTER TO ADesc; ADesc = RECORD i : INTEGER END;
 VAR s, t : A; sD, tD : ADesc;

zu jeder der folgenden Zuweisungen an, unter welcher Bedingung sie korrekt oder warum sie falsch ist!

Zuweisung	korrekt, falls...	falsch, weil...
sD := t^	<i>t # NIL</i>	
sD^ := t		<i>sD als Verbund nicht dereferenzierbar ist</i>
s := 3		<i>die Ganzzahl 3 nicht mit dem Zeiger s zuweisungskompatibel ist</i>
s^ := t		<i>der Zeiger t nicht mit dem Verbund s^ zuweisungskompatibel ist</i>
sD.i := 5	<i>immer</i>	
sD^i := tD		<i>sD als Verbund nicht dereferenzierbar ist</i>
s.i := t^		<i>der Verbund t^ nicht mit der Ganzzahlvariable s.i zuweisungskompatibel ist</i>
s^i := t.i	<i>s # NIL & t # NIL</i>	

Geben Sie zu jedem der folgenden Diagramme den Typ der Variablen le, ri (ADesc oder A) an und Anweisungen, die den Vorzustand in den Nachzustand überführen!

Typ von		Vorzustand	Anweisungen	Nachzustand
le	ri			
ADesc	ADesc	ri: le:	<i>le := ri</i>	ri: le:
A	ADesc	ri: le:	<i>le^ := ri</i>	ri: le:
A	A	ri: le:	<i>le^ := ri^</i>	ri: le:
A	A	ri: le:	<i>le := ri</i> oder <i>le := NIL</i>	ri: le:

Aufgabe 5 Vererbung, Polymorphie

(Punkte: 20 |.....)

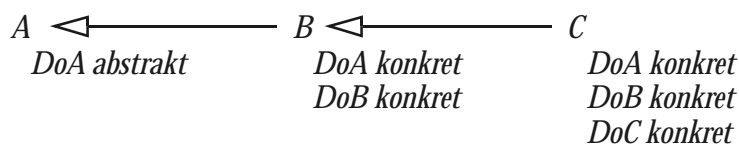
```

TYPE  A = POINTER TO ABSTRACT RECORD END;
      B = POINTER TO EXTENSIBLE RECORD (A) END;
      C = POINTER TO RECORD (B) END;

PROCEDURE (a : A) DoA, NEW, ABSTRACT;
PROCEDURE (b : B) DoA, EXTENSIBLE; BEGIN Out.String ('DoA of B') END DoA;
PROCEDURE (b : B) DoB, NEW, EXTENSIBLE; BEGIN Out.String ('DoB of B') END DoB;
PROCEDURE (c : C) DoA; BEGIN Out.String ('DoA of C') END DoA;
PROCEDURE (c : C) DoB; BEGIN Out.String ('DoB of C') END DoB;
PROCEDURE (c : C) DoC, NEW; BEGIN Out.String ('DoC of C') END DoC;
    
```

Stellen Sie die Information dieser Vereinbarungen in einem **Klassendiagramm** dar, wobei Sie bei jeder Klasse die daran gebundenen Prozeduren, und ob die Prozeduren abstrakt oder konkret sind, angeben!

Klassendiagramm

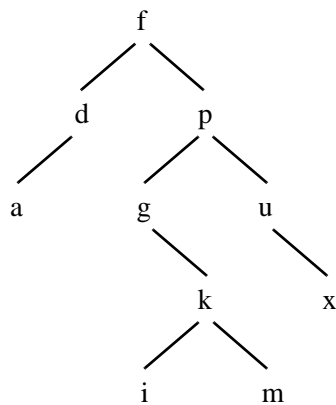


Die Variablenvereinbarungen a : A; b : B; c : C; vorausgesetzt, geben Sie zu jeder der folgenden Anweisungsfolgen an: Solange die Anweisungen korrekt sind, ihren **Effekt** (Diagramm erzeugter Objekte, Ausgabertext); bei fehlerhaften Anweisungen die Stelle und den Grund des Fehlers.

Anweisungsfolge	Objekte im Speicher	Ausgabe im Log	Fehlerursache
NEW (b); b.DoA; b.DoB		DoA of B DoB of B	
NEW (c); a := c; a.DoA; a.DoB; <input checked="" type="checkbox"/> a.DoC <input checked="" type="checkbox"/>		DoA of C	Typfehler zur Übersetzungszeit: DoB in A unbekannt DoC in A unbekannt
NEW (c); b := c; b.DoA; b.DoB		DoA of C DoB of C	
NEW (b); c := b; <input checked="" type="checkbox"/> c.DoC			Typfehler zur Übersetzungszeit: Da B nicht von C erbt, ist b nicht mit c zuweisungskompatibel

Aufgabe 6 Dynamische Datenstruktur Binärbaum

(Punkte: 25 |.....)



Geben Sie zu dem **Binärbaum** links die Reihenfolgen an, in der die Traversierungen die Elemente besuchen!

Preorder:...fdapgkimux.....

Inorder:...adfgikmpux.....

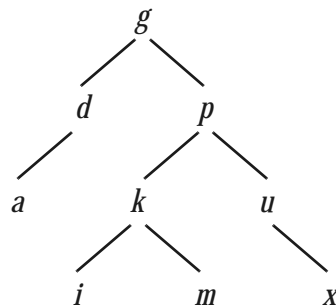
Postorder:...adimkgxupf.....

Aus dem geordneten Binärbaum links wird das Element **f entfernt** mit diesem Algorithmus:

- Falls der Baum nicht leer ist:
 - falls item kleiner als das Element der Wurzel ist entferne item aus dem linken Teilbaum,
 - sonst falls item größer als das Element der Wurzel ist entferne item aus dem rechten Teilbaum,
 - sonst ist item an der Wurzel, also falls der linke Teilbaum leer ist ersetze die Wurzel durch den rechten Teilbaum,
 - sonst falls der rechte Teilbaum leer ist ersetze die Wurzel durch den linken Teilbaum,
 - sonst sind beide Teilbäume nicht leer, also lasse den linken Teilbaum an seiner Stelle, entferne das kleinste Element aus dem rechten Teilbaum und setze es an die Stelle des zu entfernenden item an der Wurzel.

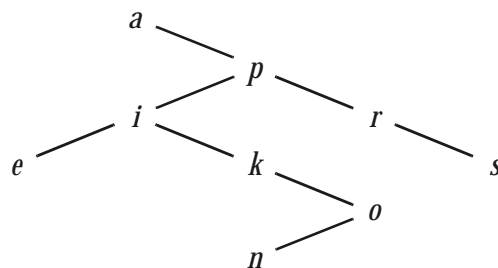
Geben Sie den daraus entstehenden Baum an!

Reduzierter Baum



Aus einem leeren Baum entsteht durch **Einfügen** der Elemente
a p r i k o s e n
dieser **geordnete Binärbaum:**

Neuer Baum



HeightIm Folgenden gelten die Vereinbarungen:

```

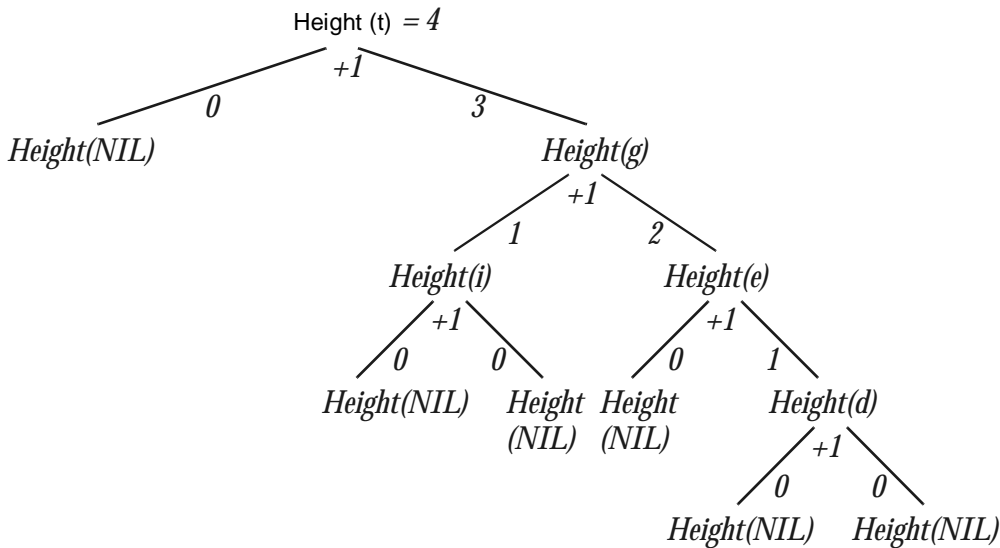
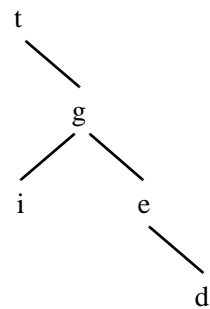
TYPE Element = CHAR;
Node = POINTER TO RECORD
    item : Element;
    left, right : Node
END;
    
```

Die Funktion

```

PROCEDURE Height (tree : Node) : INTEGER;
BEGIN
    IF tree = NIL THEN
        RETURN 0
    ELSE
        RETURN 1 + MAX (Height (tree.left), Height (tree.right))
    END
END Height;
    
```

wird mit dem Baum rechts als aktuellem Parameter aufgerufen. Geben Sie den **Aufrufbaum** an, wobei Sie als Parameter tree.item statt tree hinschreiben, falls tree # NIL ist.



Implementieren Sie die folgende Funktion, die die Anzahl der Blätter (d.h. kinderlosen Knoten) des übergebenen Baums angibt, mit einem rekursiven Algorithmus!

```

PROCEDURE NumberOfLeaves (tree : Node) : INTEGER;
BEGIN
    IF tree = NIL THEN
        RETURN 0
    ELSIF (tree.left = NIL) & (tree.right = NIL) THEN
        RETURN 1
    ELSE
        RETURN NumberOfLeaves (tree.left) + NumberOfLeaves (tree.right)
    END
END NumberOfLeaves;
    
```

```

PROCEDURE NumberOfLeaves (tree : Node) : INTEGER;
BEGIN
  IF tree = NIL THEN
    RETURN 0
  ELSE
    RETURN MAX (1, NumberOfLeaves (tree.left) + NumberOfLeaves (tree.right))
  END
END NumberOfLeaves;

```

Aufgabe 7 Dynamische Objektstruktur lineare Liste

(Punkte: 15 |.....) Die Mengenklaſſe ContainersSetsOfAny.SetDesc, die mit den Vereinbarungen

```

TYPE Element* = BG.Any;
Node = POINTER TO RECORD
  item : Element;
  next : Node
END;
SetDesc* = RECORD
  first : Node
END;

```

als **einfach verkettete Liste** implementiert ist, ist um **bedingte Iteratoren** zu erweitern. Die Bedingung und die Aktion bestimmt der Kunde, indem er bekannte Schablonenmethoden implementiert:

```

TYPE ActionDesc* = ABSTRACT RECORD END;
PROCEDURE (IN a : ActionDesc) Valid* (item : Element) : BOOLEAN, NEW, ABSTRACT;
PROCEDURE (VAR a : ActionDesc) Do* (item : Element), NEW, ABSTRACT;

```

Implementieren Sie folgende **Iteratorprozeduren** mit iterativen Algorithmen!

```

PROCEDURE (IN set : SetDesc) WhileValidDo* (VAR action : ActionDesc), NEW;
  (*
  Iterate through the items in set applying action.Do (item) while action.Valid (item) holds.
  !*)
  VAR node : Node;
BEGIN
  node := set.first;
  WHILE (node # NIL) & action.Valid (node.item) DO
    action.Do (node.item);
    node := node.next
  END
END WhileValidDo;

PROCEDURE (IN set : SetDesc) IfValidDo* (VAR action : ActionDesc), NEW;
  (*
  Iterate through the items in set applying action.Do (item) if action.Valid (item) holds.
  !*)
  VAR node : Node;
BEGIN
  node := set.first;
  WHILE node # NIL DO
    IF action.Valid (node.item) THEN
      action.Do (node.item)
    END;
    node := node.next
  END
END IfValidDo;

```