# Prüfung

# Informatik 2

**Hochschule Reutlingen - Reutlingen University**
**Fakultät Informatik**
**Studiengang Medien- und Kommunikationsinformatik**
akkreditiert von der ASIIN

## mki-Bachelor 2
## Sommersemester 2009

☞ Name, Vorname .....

☞ Matrikelnummer .....

☞ Platznummer .....

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungstermin und Ort: Freitag, 10. Juli 2009, $10^{30}$ bis $12^{30}$ Uhr, Aula

Prüfungsdauer: 120 Minuten

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

Anzahl der Aufgabenseiten: 12

| **Aufgabe** | **Punkte** | | **Aufgabe** | **Punkte** | |
|---|---|---|---|---|---|
| | **möglich** | **erreicht** | | **möglich** | **erreicht** |
| 1 | 26 | | 5 | 16 | |
| 2 | 38 | | 6 | 34 | |
| 3 | 15 | | 7 | 10 | |
| 4 | 25 | | **Summe erreicht:** | | |
| **Summe möglich:** | | **164** | **Note:** | | |

- 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!
- Der Platz reicht normalerweise; vermeiden Sie Extrablätter!
- Teilaufgaben sind oft unabhängig voneinander lösbar.

✌ **Viel Erfolg!**

## Aufgabe 1 Keller mit Warteschlange realisieren

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

```
StackDesc = RECORD
   (IN s : StackDesc) Count () : INTEGER, NEW;
   (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 ist zu realisieren. Als Objektdaten in StackDesc ist *nur ein* Objekt einer **Warteschlangenklasse** mit der bekannten Schnittstelle

```
QueueDesc = RECORD
   (IN q : QueueDesc) Count () : INTEGER, NEW;
   (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 zugelassen. Element ist ein beliebiger Elementtyp.

Visualisieren und verbalisieren Sie die **Entwurfsideen** und skizzieren Sie die **Algorithmen** der Schnittstellenoperationen der Kellerklasse!

Entwurfsideen

Daten

```
TYPE  StackDesc* =  RECORD
                      q : QueueDesc
                    END;
```

Algorithmen

## Aufgabe 2 Parser mit rekursivem Abstieg entwerfen

(Punkte: 38 |.....)

Zur Sprache, deren Syntax teilweise durch die EBNF-Regeln

```
Stmts     = Stmt { ";" Stmt } .
Stmt      = IfStmt | LoopStmt | EXIT .
IfStmt    = IF Expr THEN Stmts [ ELSE Stmts ] END .
LoopStmt  = LOOP Stmts END .
```

gegeben ist, ist ein Zerteiler mit rekursivem Abstieg zu entwerfen. Stellen Sie die statischen **Aufrufbeziehungen** der für die oben vorkommenden fünf Nichtterminale benötigten Prozeduren in einem Diagramm dar und skizzieren Sie die **Algorithmen** der Prozeduren zu den obigen vier Regeln mit den unten angegebenen Vereinbarungen!

Aufrufbeziehungen

Lieferanten: Scanner und Fehlerbehandlung

| | |
|---|---|
| symbol | Abfrage: zeigt zuletzt gelesenes Symbol an mit möglichen Werten semicolon, exit, if, then, else, end, loop, endOfInput |
| next | Aktion: liest nächstes Symbol und weist symbol den passenden Wert zu |
| error | Aktion: meldet Syntaxfehler |

Algorithmen

## Aufgabe 3 Ablauf einer rekursiven Funktion verfolgen

(Punkte: 15 |.....)

Verfolgen Sie die Ausführung der rekursiven Funktion `F` beim unten gegebenen Aufruf mit aktuellem Parameterwert `5`; zeichnen Sie dazu den **Aufrufbaum** und schreiben Sie die Ergebniswerte an die Aufrufkanten! Füllen Sie dann die **Wertetabelle**!

```
PROCEDURE F (n : INTEGER) : INTEGER;
BEGIN
   ASSERT ((0 <= n) & (n <= 42), BEC.precondPar1InRange);
   IF n < 2 THEN
      RETURN n
   ELSE
      RETURN F (n - 2) + F (n - 1)
   END
END F
```

Aufrufbaum

F (5)

Wertetabelle

| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| F (n) | | | | | | | | | |

Was ist **problematisch** an der rekursiven Implementation von `F`?

Argumente

## Aufgabe 4 Objektorientierte Begriffe erläutern

(Punkte: 25 |.....)

Gegeben sind diese Vereinbarungen:

```
TYPE   X = POINTER TO ABSTRACT RECORD END;
       Y = POINTER TO RECORD (X) END;
       Z = POINTER TO RECORD (X) y : Y END;

PROCEDURE (this : X) Do (n : INTEGER), NEW, ABSTRACT;

PROCEDURE (this : X) Repeat (n : INTEGER), NEW;
BEGIN
   WHILE n > 0 DO this.Do (n); DEC (n) END
END Repeat;

PROCEDURE (this : Y) Do (n : INTEGER);
BEGIN
   Out.Int (n, 0); Out.Ln
END Do;

PROCEDURE (this : Z) Do (n : INTEGER);
BEGIN
   IF this.y = NIL THEN NEW (this.y) END;
   this.y.Do (n);
   Out.Int (n * n, 0); Out.Ln
END Do;
```

Zeichnen Sie ein **Klassendiagramm** mit Vererbungs- und Bestandteilbeziehungen!

Klassendiagramm

Die Vereinbarungen `x : X; y : Y; z : Z;` vorausgesetzt, geben Sie zu den Anweisungsfolgen an, was sie im Speicher und im Logfenster bewirken!

| Anweisungsfolge | Zeiger und Objekte im Speicher | Ausgabe im Log |
|---|---|---|
| NEW (y);<br>y.Do (10);<br>x := y;<br>x.Do (5);<br>x.Repeat (4); | | |
| NEW (z);<br>z.Do (10);<br>x := z;<br>x.Do (5);<br>x.Repeat (4); | | |

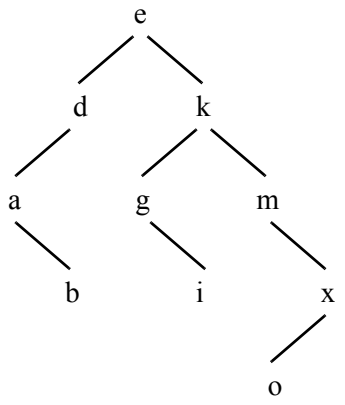
Ordnen Sie den folgenden Begriffen passende, in dieser Aufgabe vorkommende **Programmelemente** zu! (Beispiel: Formalparameter: `n : INTEGER`.)

Programmelemente

| | |
|---|---|
| abstrakte Klasse: | Zeigervariable: |
| abstrakte Prozedur: | dynamische Variable: |
| Empfängerparameter: | Objekterzeugung: |
| Schnittstellenklasse: | |
| Implementationsklasse: | polymorphe Größe: |
| Schablonenmethode: | polymorpher Aufruf: |

## Aufgabe 5 Dynamische Datenstruktur Binärbaum bearbeiten

(Punkte: 16 |.....)




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

**Präorder**:.....

**Inorder**:.....

**Postorder**:.....

Aus dem geordneten Binärbaum links wird das Element „e“ **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.
```

Markieren Sie den zutreffenden Zweig im Algorithmus und geben Sie den daraus entstehenden Baum an!

Reduzierter Baum

Aus einem leeren Baum entsteht durch **Einfügen** der Elemente

k l i m a w a n d e l

dieser **geordnete Binärbaum**:

Neuer Baum

Im Folgenden gelten die Definitionen:

```
TYPE   Element   =  CHAR;

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

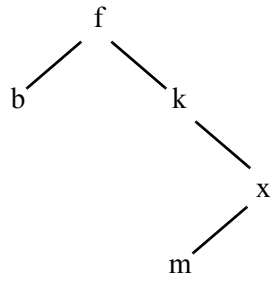
Die Funktion

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

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!

Aufrufbaum

NoL (f)




Implementieren Sie die folgende Funktion, die die Anzahl der **inneren Knoten** (solche mit wenigstens einem Kind) des übergebenen Baums angibt, mit einem rekursiven Algorithmus!

```
PROCEDURE NumberOfInnerNodes (tree : Node) : INTEGER;
BEGIN
```

Drei Anweisungen

```
END NumberOfInnerNodes;
```

## Aufgabe 6 Menge als Binärbaum zeichnen, fabrizieren und transformieren

(Punkte: 34 |.....)

Bekannt sind die Mengenklasse ContainersSetsOfComparable.Set und die von BasisGenerals.Comparable erweiterte Klasse ContainersStrings.String:

```
TYPE  Element*  =  BG.Comparable; (* = POINTER TO ... *)
      Node      =  POINTER TO RECORD
                     item        : Element;
                     left, right : Node
                   END;
      Set*      =  POINTER TO SetDesc;
      SetDesc*  =  RECORD
                     root        : Node
                   END;
      String*   =  POINTER TO EXTENSIBLE RECORD (BG.Comparable)
                     string-     : POINTER TO ARRAY OF CHAR
                   END;
```

Zeichnen Sie das **Objektdiagramm** des Mengenobjekts set : SetDesc, das die in String-Objekte verpackten Werte "bald", "fertig" nacheinander durch Aufrufe von

```
PROCEDURE (VAR set : SetDesc) Put* (x : Element), NEW;
  ASSERT (x # NIL, BEC.precondPar1NotNil);
```

aufgenommen hat!

Objektdiagramm

Implementieren Sie die **Fabrikfunktion** NewSet, die zu einer Reihung von Elementen ein neues Mengenobjekt mit diesen Elementen fabriziert und dazu obiges Put benutzt!

```
PROCEDURE NewSet* (IN a : ARRAY OF Element) : Set;
```

Zwei Vereinbarungen

Fünf Anweisungen

```
END NewSet;
```

Die an die Mengenklasse gebundene **Behältertransformation** AsSortedArray liefert

- bei leerer Empfängermenge NIL, sonst
- den Inhalt der Empfängermenge sortiert in einer dynamisch erzeugten Reihung.

Implementieren Sie die Baumfunktion SortedArray, die die Baumfunktion

```
PROCEDURE Count (tree : Node) : INTEGER;
```

für die Anzahl der Knoten im Baum benutzen kann.

Entwurfsideen

```
PROCEDURE SortedArray (tree : Node) : POINTER TO ARRAY OF Element;
```

Drei Vereinbarungen

Zehn Anweisungen

```
END SortedArray;

PROCEDURE (IN set : SetDesc) AsSortedArray* () : POINTER TO ARRAY OF Element, NEW;
BEGIN
   RETURN SortedArray (set.root)
END AsSortedArray;
```

## Aufgabe 7 Grundwissen über das Testen erläutern

(Punkte: 10 |.....) Ein mki-B6-Student stellt im SOP-Journal vom Januar 2009 die folgenden unbegründeten, widersprüchlichen und falschen Behauptungen auf. Entgegnen Sie ihm mit gut begründeten, konsistenten und korrekten Aussagen!

Er: „*Testen dient der Verifikation eines entwickelten Systems. Komponententests verifizieren das nach außen sichtbare Verhalten einer Komponente.*“

Ich:

Er: „*Die Komponententests mussten komplettiert werden, um die Korrektheit der Komponenten zu zeigen.*“

Ich:

Er: „*Schlägt der Test einer zuvor erfolgreich getesteten Komponente nach Änderungen fehl, so ist die Wahrscheinlichkeit hoch, den Fehler in dieser Komponente zu finden.*“

Ich:

Er: „*Komponententests verbessern die Qualität und Wartbarkeit des Codes.*“

Ich:

Er: „*Der Einsatz eines Testframeworks verhindert inkorrekte Tests. So wird garantiert, dass die Tests selbst fehlerfrei sind.*“

Ich: