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

Informatik 2

mki-Bachelor 2
Alte und ausgeschiedene Prüfungsaufgaben

Prüfer: Prof. Dr. Karlheinz Hug

Prüfungsdauer: 120 Minuten

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

Anzahl der Aufgabenseiten: 11

- 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 Typregeln bei Verbunden und Zeigern anwenden

Geben Sie unter der Voraussetzung der Vereinbarungen

```
TYPE  A     =  POINTER TO ADesc;
      ADesc =  RECORD
                  i : INTEGER
               END;

VAR   sD, tD  :  ADesc;
      s, t    :  A;
```

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

Zuweisung	korrekt, falls...	falsch, weil...
sD := tD		
sD := tD^		
s := t		
s^ := t^		
sD := t		

Aufgabe 2 Programmablauf bei Rekursion verfolgen

(Punkte: 11 |.....)

Verfolgen Sie die Ausführung der folgenden rekursiven Funktion; tragen Sie die Werte der aktuellen Parameter und die von den Funktionsaufrufen gelieferten Ergebniswerte in die Tabelle ein! Die Felder sind breit genug, um kleine Rechnungen aufzunehmen.

```
PROCEDURE L (n : INTEGER) : INTEGER;
BEGIN
   ASSERT (n > 0, BEC.precondPar1InRange);
   IF n = 1 THEN
      RETURN 0
   ELSIF n MOD 2 = 0 THEN
      RETURN 1 + L (n DIV 2)
   ELSE
      RETURN 2 + L (n + (n + 1) DIV 2)
   END
END L;
```

Aufruffolge	**Werte der aktuellen Parameter** n		**Ergebniswerte** result	
L	als Ausdruck	= als Zahl	als Ausdruck	= als Zahl
1. Aufruf		3		=
2. Aufruf		=		=
3. Aufruf		=		=
4. Aufruf		=		=
5. Aufruf		=		=
6. Aufruf		=		=

Aufgabe 3 — Programmablauf bei Rekursion verfolgen

(Punkte: 14 |.....)
SS 2006

Verfolgen Sie die Ausführung der folgenden rekursiven Prozedur bei den unten gegebenen Anfangswerten der Parameter; tragen Sie die Werte der aktuellen Parameter und der lokalen Variablen in die Tabelle ein!

```
PROCEDURE Do (VAR string : ARRAY OF CHAR; first, last : INTEGER);
   VAR temp : CHAR;
BEGIN
   ASSERT ((0 <= first) & (last < LEN (string$)), BEC.precondParsConsistent);
   IF first < last THEN
      temp         := string [first];
      string [first] := string [last];
      string [last] := temp;
      Do (string, first + 1, last - 1)
   END
END Do;
```

Aufruffolge	**Werte der aktuellen Parameter und lokalen Variablen**													
Do	string											first	last	temp
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]			
1. Aufruf	"B"	"ä"	"r"	"e"	"n"	"t"	"a"	"t"	"z"	"e"	0X	0	9	
2. Aufruf														
3. Aufruf														
4. Aufruf														
5. Aufruf														
6. Aufruf														

Verbalisieren Sie, was die Prozedur bewirkt!

© Karlheinz Hug, Hochschule Reutlingen

Aufgabe 4 Warteschlange mit einfach verketteter Liste realisieren

Eine **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 ist mit einer einfach verketteten ringförmig geschlossenen Liste zu realisieren. Als Listenanker genügt ein Zeiger last auf den letzten Knoten der Liste. Element ist ein beliebiger Elementtyp.

Visualisieren und verbalisieren Sie die **Entwurfsideen**, schreiben Sie die **Datendefinitionen** hin und skizzieren Sie die **Algorithmen** der Schnittstellenoperationen der Klasse!

Aufgabe 5 Prioritätswarteschlange mit einfach verketteter Liste realisieren

Eine Klasse für Prioritätswarteschlangen mit der Schnittstelle

```
Priority = INTEGER;

PriorityQueueDesc = RECORD
   (VAR q : PriorityQueueDesc) Init, NEW;
   (IN q : PriorityQueueDesc) IsEmpty () : BOOLEAN, NEW;
   (IN q : PriorityQueueDesc) Item () : Element, NEW;
   (VAR q : PriorityQueueDesc) PutPriority (x : Element; priority : Priority), NEW;
   (VAR q : PriorityQueueDesc) Remove, NEW;
END;
```

und **Highest-Priority-First**-Semantik ist mit einer einfach verketteten Liste zu realisieren. Element ist ein beliebiger Elementtyp.

Visualisieren und verbalisieren Sie die **Entwurfsideen**, schreiben Sie die **Datendefinitionen** hin und skizzieren Sie die **Algorithmen** der Schnittstellenoperationen der Klasse!

Aufgabe 6 Doppelt verkettete Listen sortieren

Gegeben mit den Definitionen

```
TYPE  Element* = INTEGER;
      Node     = POINTER TO RECORD
                    item : Element;   previous, next : Node
                 END;
      ListDesc* = RECORD
                    first : Node
                 END;
```

sind doppelt verkettete ringförmig geschlossene Listen vergleichbarer Elemente.

Welche Sortieralgorithmen lassen sich leicht zum Sortieren solcher Listen implementieren?

Welche Vor- und Nachteile haben diese Sortieralgorithmen?

Erläutern Sie die Schwierigkeit beim Versuch, Quicksort anzuwenden, und geben Sie die kritische Programmstelle an!

Skizzieren Sie den Sortieralgorithmus, den Sie für bestgeeignet halten!

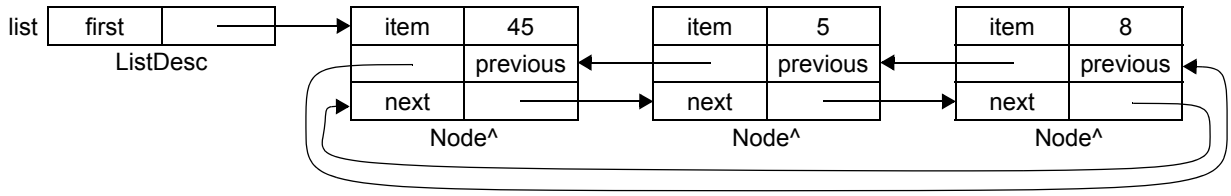
Aufgabe 7 Doppelt verkettete Liste sortieren

(Punkte: 15 |.....)
SS 2008

Gegeben mit den Definitionen

```
TYPE   Element* = INTEGER;
       Node     = POINTER TO RECORD
                    item : Element;   previous, next : Node
                  END;
       ListDesc* = RECORD
                    first : Node
                  END;
```

sind doppelt verkettete ringförmig geschlossene Listen vergleichbarer Elemente, z.B.



Implementieren Sie zum Sortieren solcher Listen das **Sortieren durch Vertauschen**, indem Sie die erste Exchange-Prozedur vervollständigen!

```
PROCEDURE Exchangesort (first : Node);
   VAR     last, node : Node;   item : Element;
BEGIN
```

Elf Anweisungen

```
END Exchangesort;

PROCEDURE (VAR list : ListDesc) Exchangesort*, NEW;
BEGIN
   Exchangesort (list.first)
END Exchangesort;
```

Aufgabe 8 Entwurfsmuster Zustand einsetzen

(Punkte: 8 |.....)
SS 2006

Die Mengenklasse mit den Definitionen

```
TYPE  Element*          = BasisGenerals.Comparable;
      Node              = POINTER TO NodeDesc;
      NodeDesc          = ABSTRACT RECORD END;
      EmptyNodeDesc     = RECORD (NodeDesc) END;
      NonEmptyNodeDesc  = RECORD (NodeDesc)
                            item        : Element;
                            left, right : SetDesc
                          END;
      SetDesc*          = RECORD
                            root        : Node
                          END;
```

ist mit einem geordneten Binärbaum nach dem Entwurfsmuster **Zustand** implementiert. Die Mengenklasse repräsentiert den **Kontext**, die Knotenklassen den **abstrakten** und die **konkreten Zustände**.

Implementieren Sie die folgende Schnittstellenfunktion, die prüft, ob das Element x in der Menge set enthalten ist!

```
PROCEDURE (IN set : SetDesc) Has* (x : Element) : BOOLEAN, NEW;
```

Vier Prozeduren - sechs Anweisungen

Aufgabe 9 Entwurfsmuster Zustand und Schablonenmethode einsetzen

(Punkte: 23 |.....)
SS 2007

Die Mengenklasse mit den Definitionen

```
TYPE  Element*          = BasisGenerals.Comparable;
      Node              = POINTER TO NodeDesc;
      NodeDesc          = ABSTRACT RECORD END;
      EmptyNodeDesc     = RECORD (NodeDesc) END;
      NonEmptyNodeDesc  = RECORD (NodeDesc)
                            item        : Element;
                            left, right : SetDesc
                          END;
      SetDesc*          = RECORD
                            root        : Node
                          END;
```

ist mit einem geordneten Binärbaum nach dem Entwurfsmuster **Zustand** implementiert. Die Mengenklasse repräsentiert den **Kontext**, die Knotenklassen den **abstrakten** und die **konkreten Zustände**.

Zeichnen Sie das **Objektdiagramm** eines Mengenobjekts set : SetDesc, das nacheinander die Werte "ich", "bin", "klug", verpackt in Objekte der bekannten von BasisGenerals.Comparable erweiterten Klasse ContainersStrings.String, aufgenommen hat!

Objektdiagramm

Gegeben ist weiter die abstrakte Klasse

```
TYPE ActionDesc* = ABSTRACT RECORD END;

PROCEDURE (IN a : ActionDesc) Valid* (item : Element) : BOOLEAN, NEW, ABSTRACT;
PROCEDURE (VAR a : ActionDesc) Do* (item : Element), NEW, ABSTRACT;
```

für Schablonenmethoden zum Traversieren der Menge.

Implementieren Sie die folgende **Iteratorprozedur**, die die Menge traversiert und auf jedes Element, für das action.Valid (item) gilt, action.Do (item) anwendet, sowie die dazu erforderlichen drei Knotenprozeduren, davon eine als **Schablonenmethode**!

```
PROCEDURE (IN set : SetDesc) IfValidDo* (VAR action : ActionDesc), NEW;
```

Vier Prozeduren - fünf Anweisungen

Aufgabe 10 Entwurfsmuster Zustand und Schablonenmethode einsetzen

Für drei Varianten von Aufgabe 9 ist die Iteratorprozedur IfValidDo *durch* WhileValidDo *und die Iteratorfunktionen* AreAllValid *und* ExistsSomeValid *zu ersetzen:*

Variante 1

Implementieren Sie die folgende **Iteratorprozedur**, die die Menge traversiert und auf jedes Element, so lang action.Valid (item) gilt, action.Do (item) anwendet, sowie die dazu erforderlichen drei Knotenprozeduren, davon eine als **Schablonenmethode**!

```
PROCEDURE (IN set : SetDesc) WhileValidDo* (VAR action : ActionDesc), NEW;
```

Variante 2

Implementieren Sie die folgende **Iteratorfunktion**, die einen **Allquantor** darstellt und die die Menge traversiert und prüft, ob jedes Element action.Valid (item) erfüllt, sowie die dazu erforderlichen drei Knotenfunktionen, davon eine als **Schablonenmethode**!

```
PROCEDURE (IN set : SetDesc) AreAllValid* (VAR action : ActionDesc) : BOOLEAN, NEW;
```

Variante 3

Implementieren Sie die folgende **Iteratorfunktion**, die einen **Existenzquantor** darstellt und die die Menge traversiert und prüft, ob wenigstens ein Element action.Valid (item) erfüllt, sowie die dazu erforderlichen drei Knotenfunktionen, davon eine als **Schablonenmethode**!

```
PROCEDURE (IN set : SetDesc)
   ExistsSomeValid* (VAR action : ActionDesc) : BOOLEAN, NEW;
```

Aufgabe 11 Schwachstellen in Mengenklasse beseitigen

Die bekannte Mengenklasse ContainersSetsOfComparable.SetDesc mit den Definitionen

```
TYPE  Element* = BG.Comparable; (* = POINTER TO BG.ComparableDesc *)
      Node     = POINTER TO RECORD
                    item        : Element;
                    left, right : Node
                 END;
      SetDesc* = RECORD
                    root        : Node
                 END;
```

hat zwei Schwachstellen, die bei dilettantischer Benutzung zu Fehlern führen können. Was sind diese Schwächen und durch welche Anpassungen an welchen Operationen verschwinden sie?

Aufgabe 12 Wissen über objektorientiertes Programmieren erläutern

(Punkte: 36 |.....)
SS 2006

Visualisieren Sie den Unterschied zwischen Wert- und Referenzsemantik!

Wertsemantik

Referenzsemantik

Component Pascal kennt Referenzsemantik in den beiden Formen

.....

Warum entfaltet objektorientiertes Programmieren nur mit Referenzsemantik seine Problemlösungsmächtigkeit?

Erläutern Sie anhand exemplarischer Programmfragmente die Unterschiede zwischen statischer und dynamischer Typbindung und -prüfung!

Statische Typbindung

Statische Typprüfung

Dynamische Typbindung

Dynamische Typprüfung

Warum ist in objektorientierten Sprachen eine Mischung beider Typkonzepte sinnvoll?

Wozu sind abstrakte Klassen nütze, von denen man doch keine Objekte erzeugen kann?

Abstrakte Klasse

Erläutern Sie, was ein polymorpher Aufruf ist, was dynamisches Binden bedeutet und wozu man dieses Konzept sinnvoll einsetzt!

Polymorphie und dynamisches Binden

Erläutern Sie den allgemeinen Zweck objektorientierter Entwurfsmuster und nennen Sie Beispiele mit ihren Anwendungsbereichen!

Entwurfsmuster