Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Wintersemester 2008/2009
Hinweise zur Bearbeitung der Klausur
zum Kurs 1613 „Einführung in die imperative Programmierung“

Wir begrüßen Sie zur Klausur „Einführung in die imperative Programmierung“. Lesen Sie sich diese Hinweise vollständig und aufmerksam durch, bevor Sie mit der Bearbeitung der Aufgaben beginnen:

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   - 2 Deckblätter,
   - 1 Formblatt für eine Bescheinigung für das Finanzamt,
   - diese Hinweise zur Bearbeitung,
   - 6 Aufgaben (Seite 2 - Seite 21),
   - die Muss-Regeln des Programmierstils.
2. Füllen Sie, **bevor** Sie mit der Bearbeitung der Aufgaben beginnen, folgende Seiten des Klausurexemplares aus:
   a) **Beide Deckblätter** mit Namen, Anschrift sowie Matrikelnummer. **Markieren Sie vor der Abgabe auf beiden Deckblättern die von Ihnen bearbeiteten Aufgaben.**
   b) Falls Sie eine Teilnahmebescheinigung für das Finanzamt wünschen, füllen Sie bitte das entsprechende Formblatt aus und belassen Sie es in der Klausur. Sie erhalten es dann zusammen mit der Korrektur abgestempelt zurück.

   Nur wenn Sie die Deckblätter vollständig ausgefüllt haben, können wir Ihre Klausur korrigieren!
3. Schreiben Sie Ihre Lösungen auf den freien Teil der Seite unterhalb der Aufgabe bzw. auf die leeren Folgeseiten. Sollte dies nicht möglich sein, so vermerken Sie, auf welcher Seite die Lösung zu finden ist.
   Streichen Sie ungültige Lösungen deutlich durch! (Sollten Sie mehr als eine Lösung zu einer Aufgabe abgeben, so wird nur eine davon korrigiert – und nicht notwendig die bessere.)
4. Schreiben Sie auf jedes von Ihnen beschriebene Blatt oben links Ihren Namen und oben rechts Ihre Matrikelnummer. Wenn Sie weitere eigene Blätter benutzt haben, heften Sie auch diese, mit Namen und Matrikelnummer versehen, an Ihr Klausurexemplar. Nur dann werden auch Lösungen außerhalb Ihres Klausurexemplares gewertet!
5. Neben unbeschriebenem Konzeptpapier und Schreibzeug (Füller oder Kugelschreiber, benutzen Sie **keinen Bleistift**!) sind **keine weiteren Hilfsmittel** zugelassen. Die Muss-Regeln des Programmierstils finden Sie im Anschluss an die Aufgabenstellung.
6. Es sind maximal 32 Punkte erreichbar. Sie haben die Klausur sicher dann bestanden, wenn Sie mindestens 16 Punkte erreicht haben.

Wir wünschen Ihnen bei der Bearbeitung der Klausur viel Erfolg!

Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Aufgabe 1 (3+1 Punkte)

Gegeben seien folgende Typdefinitionen für ein Feld mit Integer-Zahlen:

```
const GRENZE = 10;
type  tIndex = 1 .. GRENZE;
      tFeld  = array[tIndex] of integer;
```

a) Schreiben Sie eine Funktion, die überprüft, ob ein ihr übergebenes Feld vom Typ `tFeld` aufsteigend sortiert ist, d.h. ob keines der Elemente kleiner als sein Vorgängerelement ist.
Verwenden Sie dabei eine For-Schleife zum Durchlauf des Feldes!
Verwenden Sie weiterhin folgenden Funktionskopf:

```
function istSortiert(var inFeld: tFeld): Boolean;
{Ergebnis ist true, genau dann wenn inFeld aufsteigend sortiert
 ist, also keines der Elemente kleiner als sein Vorgänger ist.}
```

b) Das Verwenden einer For-Schleife, wie in Teil a) gefordert, ermöglicht zwar eine kurze Lösung, ist jedoch für dieses Problem nicht optimal im Hinblick auf die Laufzeit. Warum nicht?

Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

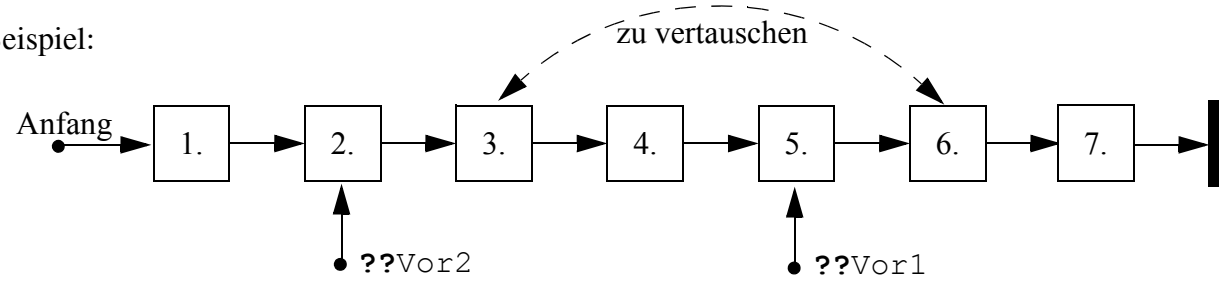
Aufgabe 2 (3+5 Punkte)

Gegeben seien folgende Typvereinbarungen für Elemente einer linearen Liste von Integer-Zahlen:

```
type
tRefElement = ^tElement;
tElement = record
             info : integer;
             next : tRefElement
           end;
```

Gesucht ist eine Prozedur, die zwei Elemente einer nicht-leeren Liste durch Änderung der Verkettung vertauscht. Die Prozedur bekommt zu diesem Zweck zwei Zeiger **??**`Vor1` und **??**`Vor2` auf Listenelemente übergeben. Es sind die Nachfolger der durch **??**`Vor1` und **??**`Vor2` referenzierten Elemente zu vertauschen.

Beispiel:



a) Dass sowohl **??**`Vor1` als auch **??**`Vor2` tatsächlich auf ein Element der Liste zeigen, dürfen Sie als Vorbedingung voraussetzen. Beschreiben Sie kurz, welche Sonderfälle (also Fälle, in denen die Prozedur anders agieren muss als im oben beispielhaft skizzierten Normalfall) unter dieser Vorbedingung noch eintreten können.

b) Schreiben Sie nun die Prozedur. Dabei muss Ihre Prozedur die Sonderfälle nicht behandeln, d.h. es genügt, wenn sie für den Normalfall funktioniert.
Benutzen Sie folgenden Prozedurkopf und ergänzen Sie dabei an den mit **??** gekennzeichneten Stellen die Übergabeart der Parameter:

```
procedure tauschen(?? ??Vor1, ?? ??Vor2:tRefElement);
{Vertauscht zwei Elemente einer Liste durch Änderung der
 Verkettung.
 ??Vor1 und ??Vor2 sind dabei Zeiger auf die Vorgängerelemente der
 zu vertauschenden Elemente.
 Vorbedingung: ??Vor1 und ??Vor2 zeigen jeweils auf ein existie-
 rendes Element derselben Liste.[1] }
```

1. Normalerweise sollte der Kommentar zur Vorbedingung auch die ausgeschlossenen Sonderfälle genau beschreiben, was hier jedoch unterbleibt, um die Lösung zu Teil a) nicht vorwegzunehmen.

Aufgabe 3 (3+1+1+1 Punkte)

Gegeben seien die folgenden Typdefinitionen für einen Binärbaum sowie die (unkommentierte) Prozedur `whatDoIDo`.

```
type
  tRefKnoten = ^tKnoten;
  tKnoten = record
              t: integer;
              links,
              rechts: tRefKnoten
            end;

procedure whatDoIDo(inWurzel: tRefKnoten; inT: integer);
begin
  if inWurzel <> nil then
  begin
    whatDoIDo(inWurzel^.links, inT+1);
    inWurzel^.t := inT;
    whatDoIDo(inWurzel^.rechts, inT+1)
  end
end;
```

a) Überlegen Sie, was die Prozedur `whatDoIDo` leistet.

- Was passiert beispielsweise, wenn der Prozedur im Parameter `inWurzel` der auf der folgenden Seite abgebildete Baum und im Parameter `inT` der Wert 1 übergeben wird?
  Zeichnen Sie den resultierenden Baum mit den neuen Knotenwerten.
- Beschreiben Sie dann knapp (in maximal drei Sätzen), was `whatDoIDo` allgemein bei Eingabe eines beliebigen Baumes sowie `inT` = 1 leistet.

b) Die Prozedur `whatDoIDo` durchläuft den Baum in einer bestimmten Reihenfolge. Wie wird diese Durchlauf-Reihenfolge genannt?

c) Ist die Durchlauf-Reihenfolge für `whatDoIDo` relevant oder würde die Prozedur auch mit einer anderen Durchlauf-Reihenfolge unverändert funktionieren? Begründen Sie kurz Ihre Antwort!

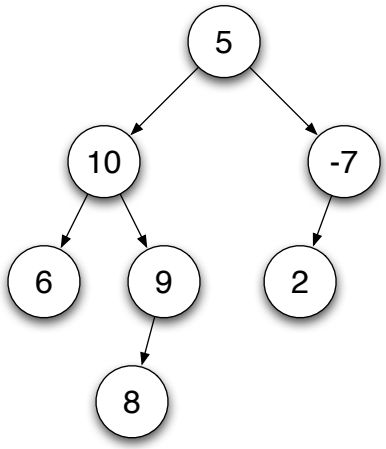
d) Handelt es sich bei `whatDoIDo` um eine sinnvolle Anwendung der Rekursion? Begründen Sie Ihre Antwort!

Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Name: _____ Matrikelnummer: _____

Abbildung zu Teil a):



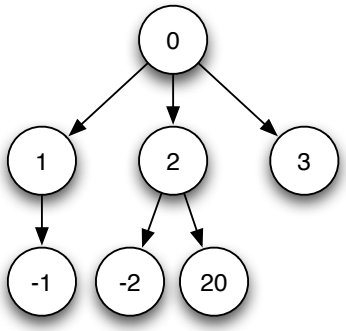
Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Aufgabe 4 (6 Punkte)

Die im Kurs eingeführten Binärbäume zeichnen sich dadurch aus, dass jeder Knoten genau zwei (ggf. leere) binäre Teilbäume als Nachfolger hat. Bäume müssen jedoch nicht unbedingt binär sein.

Im Folgenden werden Bäume betrachtet, bei denen jeder einzelne Knoten eine beliebige Anzahl von Nachfolgern haben kann, wie in rechts stehender Abbildung beispielhaft dargestellt.

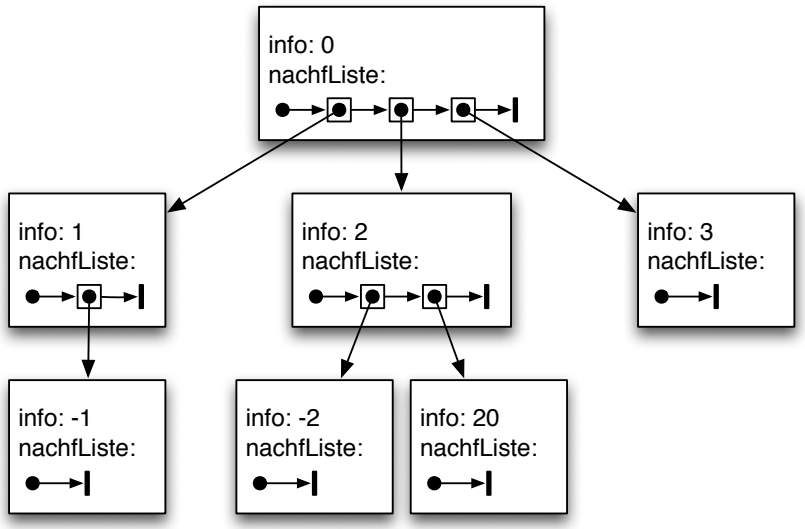


Wir realisieren einen solchen Baum, indem jeder Knoten neben seinem Info-Wert eine lineare Liste von Zeigern auf nicht-leere Teilbäume enthält (jeder Zeiger aus dieser Liste ist also stets ungleich **`nil`**). Für Blätter ist diese Nachfolgerliste leer.

Dazu verwenden wir folgende Typdefinitionen:

```
type
  tRefMultibaumKn = ^tMultibaumKn;
  tRefNachfListenElem = ^tNachfListenElem;
  tNachfListenElem =    record
                          teilbaumWurzel : tRefMultibaumKn;
                          next : tRefNachfListenElem;
                        end;
  tMultibaumKn =  record
                    info : integer;
                    nachfListe : tRefNachfListenElem;
                  end;
```

Die nachfolgende Abbildung zeigt den selben Baum wie die obenstehende Abbildung, stellt aber für jeden Knoten die Details des Aufbaus einschließlich der Nachfolgerliste mit dar:



Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Name: _____ Matrikelnummer: _____

Schreiben Sie eine rekursive Prozedur, die einen solchen Baum durchläuft und dabei den Info-Wert jedes Knotens durch seinen negativen Wert ersetzt (also mit –1 multipliziert). Verwenden Sie folgenden Prozedurkopf:

```
procedure baumNegation(inRefWurzel: tRefMultibaumKn);
{ Durchläuft den Baum und ersetzt den Info-Wert jedes Knotens durch
seinen negativen Wert.}
```

Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Aufgabe 5 (1+2 Punkte)

Eine Funktion `abs` zur Berechnung des absoluten Betrags einer ganzen Zahl soll einem funktionsorientierten Test mit Hilfe von Eingabeäquivalenzklassen unterzogen werden.

Im Folgenden gelte $0 \in \mathbb{N}$ und sei $\mathbb{N}^+ := \mathbb{N} \setminus \{0\}$ die Menge der positiven ganzen Zahlen ohne Null.

Die formale Spezifikation für die Funktion `abs` lautet:

$\texttt{abs}: \mathbb{Z} \to \mathbb{N}, \texttt{abs}(n) := |n|$

Wir geben anbei zwei Eingabeäquivalenzklassen vor:

$E_1 := \mathbb{N}^+$ (Klasse der positiven Eingaben)

$E_2 := \{ -n \mid n \in \mathbb{N}^+ \}$ (Klasse der negativen Eingaben)

a) Um die Menge der zulässigen Eingaben in Eingabeäquivalenzklassen zu zerlegen, fehlt noch eine dritte Äquivalenzklasse E_3. Wie lautet sie?

b) Geben Sie den Testfall zu Eingabeäquivalenzklasse E_2 an.

Aufgabe 6 (2+3 Punkte)

a) Die folgende Abbildung nennt vier ausgewählte Überdeckungsmaße für strukturorientierte Tests:

Zeichnen Sie in diese Abbildung die Beziehungen zwischen den dargestellten Maßen wie folgt ein: Zeichnen Sie einen Pfeil von einem Maß A zu einem Maß B, wenn eine vollständige Überdeckung gemäß Maß A auch eine vollständige Überdeckung gemäß Maß B impliziert, wenn Maß A also Maß B umfasst.

Beispiel: Falls eine vollständige Zweigüberdeckung auch eine vollständige Anweisungsüberdeckung sicherstellt, zeichnen Sie einen Pfeil von „Zweigüberdeckung“ nach „Anweisungsüberdeckung“ ein.

Kurs 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 21.02.2009

Name: _____ Matrikelnummer: _____

---

b) Die Funktion `MinZiffer` soll die kleinste Ziffer der Dezimaldarstellung der nicht-negativen ganzen Zahl `inZahl` bestimmen.

```
type
tZiffer = 0..9;
tNatZahl = 0..maxint;

function MinZiffer (inZahl:tNatZahl): tZiffer;
{ bestimmt die kleinste Ziffer von inZahl}

  var
  rest: tNatZahl;
  min,
  ziffer: tZiffer;

1   begin
2     rest := inZahl;
3     min := 9;
4     while rest > 0 do
5     begin
6       ziffer := rest mod 10;
7       rest := rest div 10;
8       if ziffer < min then
9         min := ziffer;
10    end;
11    MinZiffer := min
12  end; { MinZiffer }
```

b1) Wird mit dem Testdatum (15, 1) eine *vollständige Anweisungsüberdeckung* erreicht?
b2) Wird mit dem Testdatum (15, 1) eine *vollständige Zweigüberdeckung* erreicht?
Begründen Sie jeweils Ihre Antwort.

Zusammenfassung der Muss-Regeln

---

1. Selbstdefinierte Konstantenbezeichner bestehen nur aus Großbuchstaben. Bezeichner von Standardkonstanten wie z.B. `maxint` sind also ausgenommen
2. Typbezeichnern wird ein `t` vorangestellt.
   Bezeichner von Zeigertypen beginnen mit `tRef`.
   Bezeichner formaler Parameter beginnen mit `in`, `io` oder `out`.
3. Jede Anweisung beginnt in einer neuen Zeile;
   `begin` und **`end`** stehen jeweils in einer eigenen Zeile
4. Anweisungsfolgen werden zwischen **`begin`** und **`end`** um eine konstante Anzahl von 2 - 4 Stellen eingerückt. **`begin`** und **`end`** stehen linksbündig unter der zugehörigen Kontrollanweisung, sie werden nicht weiter eingerückt.
5. Anweisungsteile von Kontrollanweisungen werden genauso eingerückt.
6. Im Programmkopf wird die Aufgabe beschrieben, die das Programm löst.
7. Jeder Funktions- und Prozedurkopf enthält eine knappe Aufgabenbeschreibung als Kommentar. Ggf. werden zusätzlich die Parameter kommentiert.
8. Die Parameter werden sortiert nach der Übergabeart: Eingangs-, Änderungs- und Ausgangsparameter.
9. Die Übergabeart jedes Parameters wird durch Voranstellen von `in`, `io` oder `out` vor den Parameternamen gekennzeichnet.
10. Das Layout von Funktionen und Prozeduren entspricht dem von Programmen.
11. Jede von einer Funktion oder Prozedur benutzte bzw. manipulierte Variable wird als Parameter übergeben. Es werden keine globalen Variablen manipuliert.
12. Jeder nicht von der Prozedur veränderte Parameter wird als Wertparameter übergeben. Lediglich Felder können auch anstatt als Wertparameter als Referenzparameter übergeben werden, um den Speicherplatz für die Kopie und den Kopiervorgang zu sparen. Der Feldbezeichner beginnt aber stets mit dem Präfix `in`, wenn das Feld nicht verändert wird.
13. Funktionsprozeduren werden wie Funktionen im mathematischen Sinne benutzt, d.h. sie besitzen nur Wertparameter. Wie bei Prozeduren ist eine Ausnahme nur bei Feldern erlaubt, um zusätzlichen Speicherplatz und Kopieraufwand zu vermeiden.
14. Wertparameter werden nicht als lokale Variable missbraucht.
15. Die Laufvariable wird innerhalb einer **`for`**-Anweisung nicht manipuliert.
16. Die Grundsätze der strukturierten Programmierung sind strikt zu befolgen.