Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Wintersemester 2009/2010
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,
   - 5 Aufgaben (Seite 2 - Seite 20),
   - 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 30 Punkte erreichbar. Sie haben die Klausur sicher dann bestanden, wenn Sie mindestens 15 Punkte erreicht haben.

Wir wünschen Ihnen bei der Bearbeitung der Klausur viel Erfolg!

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Aufgabe 1 (1+4 Punkte)

Gegeben seien folgende Typdefinitionen für eine lineare Liste von Integer-Zahlen:

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

Gesucht wird eine Prozedur zum „Rotieren“ einer aus Elementen dieses Typs aufgebauten linearen Liste: Die Prozedur bekommt dazu eine solche Liste übergeben und soll sie so modifizieren, dass das ehemals zweite Listenelement zum ersten wird, das dritte zum zweiten, u.s.w., das letzte zum vorletzten sowie das ursprünglich erste zum neuen letzten Element wird.

Wird beispielsweise eine fünfelementige Liste übergeben, die die Werte 1, 2, 3, 4, 5 (in dieser Reihenfolge) enthält, so soll die Liste nach Anwendung der Prozedur dieselben Elemente in der Reihenfolge 2, 3, 4, 5, 1 enthalten.

Tipp: Wir empfehlen, für die Entwicklung des Lösungsansatzes auf Skizzen zurückzugreifen.

a) Vergleichen Sie die folgenden beiden Ansätze zur Realisierung:

   - Umstrukturierung der Liste nur durch Verkettungsänderungen
     (ohne Modifikation von Info-Komponenten)
   - Austauschen der Inhalte (hier: Info-Komponenten) zwischen Listenelementen
     (ohne Verkettungsänderung)

   Welche Variante ist hier zu bevorzugen? Begründen Sie Ihre Antwort!
   (Eine Antwort ohne Begründung wird nicht bewertet.)

b) Implementieren Sie unabhängig von Ihrer Antwort zu Teil a) die erste Variante, d.h. schreiben Sie eine Prozedur, welche die Rotation ausschließlich durch Änderung der Listenverkettung und ohne Modifikation der Info-Komponenten durchführt.

   Verwenden Sie folgenden Prozedurkopf und ergänzen Sie an den mit **??** gekennzeichneten Stellen die passende Parameterübergabeart:

```
procedure RotiereListe(?? ??RefAnfang: tRefElement);
```

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

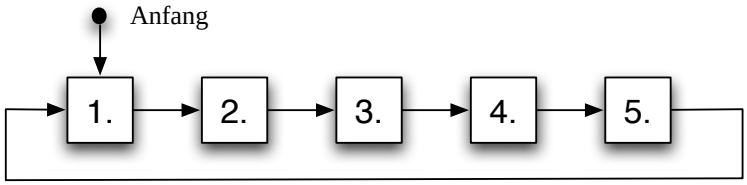
Aufgabe 2 (5 Punkte)

Gegeben seien folgende Typdefinitionen für eine lineare Liste von Integer-Zahlen:

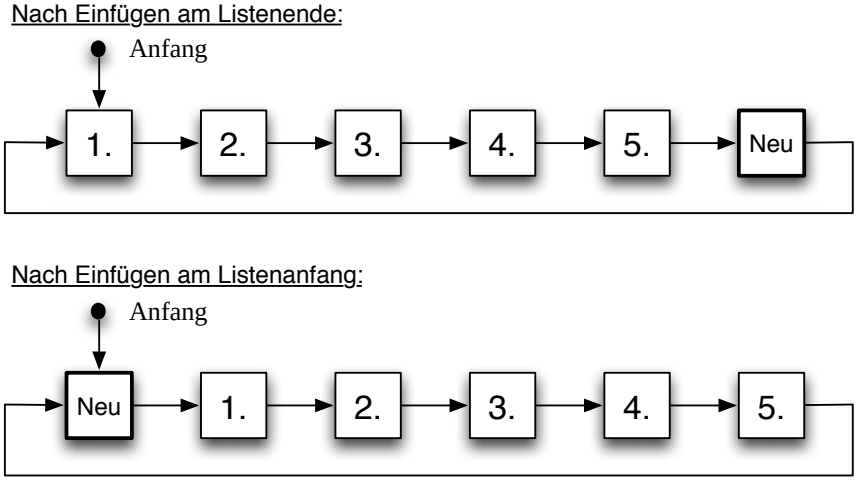
```
type
  tRefElement = ^tElement;
  tElement = record
               info : integer;
               next : tRefElement
             end;
```

Aus Elementen dieses Typs kann man auch *zyklische* lineare Listen bilden, die sich von den aus dem Kurs bekannten linearen Listen nur darin unterscheiden, dass der `next`-Zeiger des letzten Elements nicht den Wert **`nil`** besitzt, sondern wieder auf das erste Listenelement verweist. Als erstes Element eines solchen „Rings“ betrachten wir dabei das Element, auf das der Anfangszeiger verweist. Eine leere Liste wird unverändert durch einen Anfangszeiger mit dem Wert **`nil`** dargestellt.

Die folgende Abbildung zeigt beispielhaft eine solche zyklische Liste mit 5 Elementen:



Gesucht ist eine Prozedur, die einer solchen zyklischen Liste ein Element entweder am Anfang oder am Ende hinzufügt. Der Aufrufer übergibt dazu insbesondere den Anfangszeiger und den im neuen Listenelement zu speichernden Integer-Wert. Ein dritter Parameter bestimmt, ob das neue Element den neuen Listenanfang oder das neue Listenende bilden soll. Die folgende Abbildung stellt für beide Fälle jeweils das Ergebnis des Einfügevorgangs (bezogen auf die oben dargestellten Ausganssituation) grafisch dar:



Tipp: Überlegen Sie vorab, was in beiden Fällen (Einfügen am Anfang bzw. am Ende) jeweils zu tun ist.

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Implementieren Sie diese Prozedur unter Verwendung des folgenden Prozedurkopfs.
Sie dürfen dabei voraussetzen, dass die *Liste nicht leer* ist!

```
procedure einfuegen ( inInfo: integer;
                      inEinfuegenAmAnfang: Boolean;
                  var ioRefAnfang: tRefElement);
{Fügt ein neues Element mit Info-Wert inInfo in eine zyklische Liste
ein, auf deren erstes Element ioRefAnfang zeigt.
Ist inEinfuegenAmAnfang = true, wird am Listenanfang, andernfalls am
Listenende eingefügt.}
```

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Aufgabe 3 (4+1 Punkte)

Gegeben seien folgende Typvereinbarungen für binäre Bäume, deren Knoten ganze Zahlen enthalten:

```
type
  tRefKnoten = ^tKnoten;
  tKnoten =   record
                info : Integer;
                links,
                rechts : tRefKnoten
              end;
```

a) Implementieren Sie eine *rekursive* Funktion, die in einem binären *Suchbaum* aus Knoten des obigen Typs nach einem zu übergebenden `info`-Wert sucht. Kommt ein solcher Wert im Suchbaum vor, soll sie mit `true`, andernfalls mit `false` antworten.
Die Funktion soll nie mehr Knoten besuchen als der längste Pfad enthält. Die Sortierung des Suchbaums ist also auszunutzen.

Benutzen Sie folgenden Funktionskopf:

```
function kommtVor(inRefWurzel: tRefKnoten;
                  inSuchwert: integer): boolean;
{ Rückgabe ist true genau dann wenn der Suchbaum mit Wurzel
  inRefWurzel einen Knoten mit info inSuchwert enthält. }
```

b) Handelt es sich hier um eine sinnvolle Anwendung der Rekursion? Begründen Sie Ihre Antwort. (Eine unbegründete Antwort wie „Ja“ oder „Nein“ wird nicht bewertet!)

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Aufgabe 4 (5+2 Punkte)

Beim beliebten Sudoku-Rätsel ist eine teilweise ausgefüllte 9×9-Matrix zu komplettieren. Die Komponenten der Matrix sind jeweils Ziffern aus {1, ..., 9}. Die folgende Abbildung zeigt ein Beispiel für eine vollständig ausgefüllte Matrix:

3	6	8	5	7	1	4	9	2
7	9	1	3	2	4	5	8	6
2	4	5	8	9	6	1	3	7
4	8	9	1	5	7	6	2	3
5	2	3	6	4	8	7	1	9
6	1	7	2	3	9	8	4	5
1	3	6	9	8	5	2	7	4
9	5	4	7	1	2	3	6	8
8	7	2	4	6	3	9	5	1

Nach den Sudoku-Regeln ist eine solche Matrix genau dann eine Lösung eines Rätsels, wenn sie folgende drei Bedingungen erfüllt:

- Jede Zeile enthält alle Ziffern von 1 bis 9
- Jede Spalte enthält alle Ziffern von 1 bis 9
- Jeder Block enthält alle Ziffern von 1 bis 9

Als „Blöcke“ bezeichnen wir dabei die neun in obiger Abbildung jeweils fett eingerahmten, quadratischen 3×3-Teilfelder. (Die abgebildete Beispiel-Matrix erfüllt übrigens alle drei Bedingungen.)

Gesucht wird nun eine Pascal-Funktion, die für eine einzugebende 9×9-Matrix überprüft, ob es sich um eine Lösung für ein Sudoku-Rätsel handelt, d.h. ob die drei o.g. Bedingungen erfüllt sind.

Auf der nächsten Seite geben wir zunächst die benötigten Typdefinitionen sowie Prozedur-/Funktionsköpfe vor.

Als Hilfsmittel zur Lösung der Aufgabe sind der Typ `tZifferMarkerFeld` sowie die Hilfsprozedur `initZifferMarkerFeld` und die Hilfsfunktion `alleZiffernMarkiert` gegeben, die Sie in Ihrer Lösung benutzen dürfen (ohne sie implementieren zu müssen bzw. deren Implementierun kennen zu müssen). Mit Hilfe eines „Marker-Feldes“ vom Typ `tZifferMarkerFeld` kann z.B. überprüft werden, ob in einer Zeile der Matrix jede Ziffer einmal vorkommt, indem zunächst das Marker-Feld für jede Ziffer mit false initialisiert wird, dann für jede in der Matrixzeile vorkommende Zahl im Marker-Feld eine Markierung (in Form einer Zuordnung von true) angebracht wird, um abschließend zu überprüfen, ob jede Zahl im Marker-Feld markiert wurde.

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

```
type  tZiffer = 1..9;
      tZifferMarkerFeld = array [tZiffer] of boolean;
      tSudokuFeld = array [tZiffer, tZiffer] of tZiffer;

procedure initZifferMarkerFeld(var ioFeld: tZifferMarkerFeld);   gegeben
{Initialisiert ein tZifferMarkerFeld, indem jeder Ziffer der Wert
false zugeordnet wird.}

function alleZiffernMarkiert(inFeld: tZifferMarkerFeld):boolean; gegeben
{Gibt genau dann true zurueck, wenn im Feld alle Ziffern markiert
sind, d.h. wenn das Feld an jedem Index den Wert true enthaelt.}

function pruefeTeilfeld(inSudoku: tSudokuFeld;                   siehe a)
                        inStartZeile, inStartSpalte,
                        inEndZeile, inEndSpalte: tZiffer):boolean;
{Prueft, ob das eingegebene Sudokufeld (inSudoku) innerhalb des durch
die restlichen vier Parameter definierten Ausschnittes jede der Zif-
fern von 1 bis 9 mindestens einmal enthält.}

function pruefeSudoku(inSudoku: tSudokuFeld): boolean;           siehe b)
{Gibt genau dann true zurueck, wenn die Werte im Feld inSudoku
eine gueltige Loesung fuer ein Sudoku-Raetsel darstellen, wenn also
gilt, dass in jeder Zeile, in jeder Spalte und in jedem Block jede
Ziffer von 1 bis 9 genau einmal vorkommt.}
var i, j: tZiffer;
    gueltig: boolean;
begin
  gueltig:=true;
  {Alle 9 Zeilen pruefen:}
  for i:=1 to 9 do
    if not pruefeTeilfeld(inSudoku, i, 1, i, 9) then
      gueltig:=false;
  {Alle 9 Spalten pruefen:}
  for j:=1 to 9 do
    if not pruefeTeilfeld(inSudoku, 1, j, 9, j) then
      gueltig:=false;
  {Alle 3x3 Bloecke pruefen:}
  for i:=1 to 3 do
    for j :=1 to 3 do
      if not pruefeTeilfeld(inSudoku, i*3-2, j*3-2, i*3, j*3) then
        gueltig:=false;
  pruefeSudoku := gueltig
end;
```

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

---

a) Implementieren Sie die Hilfsfunktion `pruefeTeilfeld`, deren Funktionskopf oben gegeben wurde. Sie soll prüfen, ob in einem bestimmten Ausschnitt einer Sudoku-Matrix `inSudoku` jede Ziffer (von 1 bis 9) vorkommt. Der rechteckige zu durchsuchende Ausschnitt wird bestimmt durch eine linke obere Komponente (`inSudoku[inStartZeile,inStartSpalte]`) und eine rechte untere Komponente (`inSudoku[inEndZeile,inEndSpalte]`).

Beispiele:
Bei Eingabe von `inStartZeile`=`inEndZeile`=3, `inStartSpalte`=1, `inEndSpalte`=9 ist der zu überprüfende Ausschnitt genau die dritte Zeile der Matrix.
Bei Eingabe von `inStartZeile`=7, `inStartSpalte`=4, `inEndZeile`=9, `inEndSpalte`=6 ist der zu überprüfende Ausschnitt genau der untere mittlere Block.

b) Betrachten Sie die vorgegebene Implementierung der Funktion `pruefeSudoku`, die durch Anwendung von `pruefeTeilfeld` auf jede der neun Zeilen, jede der neun Spalten und jeden der neun Blöcke einer ihr übergebenen Matrix überprüft, ob die Matrix eine Lösung für ein Sudoku-Rätsel ist.

Diese Implementierung hat im Allgemeinen keine optimale Laufzeit. Erläutern Sie, weshalb und wie das Laufzeitverhalten verbessert werden kann. (Eine neue Implementierung ist nicht verlangt.)

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Aufgabe 5 (3+1+2+2 Punkte)

Gegeben seien folgende Vereinbarungen:

```
    const MAXWERT=256;
          FELDMAX=5;

    type  tIndex = 1..FELDMAX;
          tErg   = -1..FELDMAX;
          tFeld  = array[tIndex] of integer;

    function sucheInFeld(inFeld:tFeld; inZahl:integer): tErg;
    { Sucht im Array tFeld nach einem Vorkommen von inZahl.
      inZahl muss zwischen 1 und MAXWERT liegen.
      Rueckgabe: -1, falls inZahl nicht zwischen 1 und MAXWERT,
                  0, falls inZahl zwar zwischen 1 und MAXWERT,
                      aber nicht im Feld gefunden wurde,
                  Index der Fundstelle im Array (>0) sonst. }
    var i:integer;
1   begin
2     if (inZahl <= 0) or (inZahl > MAXWERT) then
3       sucheInFeld:=-1
4     else
5     begin
6       i:=1;
7       while (i < FELDMAX) and (inFeld[i] <> inZahl) do
8         i:=i+1;
9       sucheInFeld:=i
10    end;
11  end;
```

Die Implementierung ist möglicherweise fehlerhaft und soll getestet werden. Die Spezifikation im Funktionskopf, beschreibt zwei Sonderfälle und einen Normalfall. Dieser Fallunterscheidung entsprechend lassen sich unmittelbar drei Eingabeäquivalenzklassen bilden.

a) Präzisieren Sie diese Idee, indem Sie die drei Eingabeäquivalenzklassen angeben (eine präzise verbale Beschreibung genügt).

Zu jeder der drei Klassen wählen wir ein Testdatum als Repräsentant, und zwar:
D_1 := (A, 0, -1), D_2 := (A, 10, 0), D_3 := (A, 2, 2)

Der Einfachheit halber setzen wir dabei in allen drei Testdaten dasselbe Array A := [6,2,3,4,1] ein (d.h. es gelte A[1]=6, A[2]=2, A[3]=3, A[4]=4 und A[5]=1).

b) Die ersten beiden Komponenten jedes Testdatums stehen für die beiden Eingaben `inFeld` sowie `inZahl`. Was gibt die dritte Komponente der Testdaten an?

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

---

c) Wird mit den drei Testdaten D_1, D_2 und D_3 eine vollständige Zweigüberdeckung erreicht? Begründen Sie Ihre Antwort.

d) Wird mit den drei Testdaten D_1, D_2 und D_3 eine vollständige minimale Mehrfachbedingungsüberdeckung erreicht? Begründen Sie Ihre Antwort!

Bei Bedarf können Sie die folgende Tabelle zu Hilfe nehmen:

Prädikate	***Testdaten (***mit A := [6,2,3,4,1]***)***		
	(A, 0, -1)	(A, 10, 0)	(A, 2, 2)

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Kurs 1613 „Einführung in die imperative Programmierung“

Nachklausur am 06.03.2010

Name: _____ Matrikelnummer: _____

Kurs 1613 „Einführung in die imperative Programmierung“

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. Pascal-Funktionen 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.