Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Wintersemester 2006/2007
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 18),
   - 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.

   Nur wenn Sie beide 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.
4. Schreiben Sie auf jedem von Ihnen beschriebenen 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) sind **keine** weiteren Hilfsmittel zugelassen. Die Muss-Regeln des Programmierstils finden Sie im Anschluss an die Aufgabenstellung.
6. Es sind maximal 27 Punkte erreichbar. Sie haben die Klausur sicher dann bestanden, wenn Sie mindestens 14 Punkte erreicht haben.

Wir wünschen Ihnen bei der Bearbeitung der Klausur viel Erfolg!

Kurs 1613 "Einführung in die imperative Programmierung"

Nachklausur am 31.03.2007

Aufgabe 1 (4 Punkte)

Gegeben seien folgende Typvereinbarungen für Arrays von Integer-Zahlen einer Länge `MAX` ($\in$ IN) bzw. doppelter Länge (2·`MAX`).

```
const MAX = ?;
      MAXDOPPELT = 2*MAX;

type
  tIndex = 1..MAX;
  tIndexDoppelt = 1..MAXDOPPELT;
  tIntFeld = array[tIndex] of integer;
  tIntFeldDoppelt = array[tIndexDoppelt] of integer;
```

Schreiben Sie eine Funktion, welche zwei Arrays vom Typ `tIntFeld` zu einem doppelt so langen Feld vom Typ `tIntFeldDoppelt` zusammenführt, indem abwechselnd je ein Element aus dem ersten eingegebenen Feld und eines aus dem zweiten ins Zielfeld kopiert werden.

Ein Beispiel:
Gegeben seien zwei Felder `Feld1` und `Feld2` vom Typ `tIntFeld` wie in folgender Tabelle (bei `MAX`=5).

Index `i`	1	2	3	4	5
Wert `Feld1[i]`	0	1	2	3	4
Wert `Feld2[i]`	5	6	7	8	9

Diese beiden Felder sollen zu folgendem Feld `Feld` verschmolzen werden:

Index `j`	1	2	3	4	5	6	7	8	9	10
Wert `Feld[j]`	0	5	1	6	2	7	3	8	4	9

Benutzen Sie folgenden Prozedurkopf und ergänzen Sie dabei an den mit **??** gekennzeichneten Stellen die Übergabeart der Parameter:

```
procedure merge(??Feld1, ??Feld2: tIntFeld; ??Feld: tIntFeldDoppelt);
```

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

Nachklausur am 31.03.2007

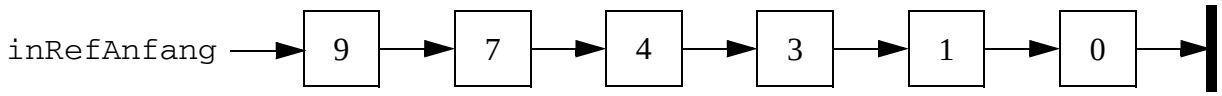
Aufgabe 2 (4 Punkte)

Gegeben sei folgende Typdeklaration für Elemente einer linearen Liste:

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

Schreiben Sie eine Funktion `AltSumme`, welche die Werte der Elemente einer Liste abwechselnd addiert und subtrahiert und das Ergebnis zurückgibt. (D.h. das zweite Listenelement wird vom ersten subtrahiert, das dritte wieder addiert u.s.w.)

Ein Beispiel:




```
AltSumme := 9 - 7 + 4 - 3 + 1 - 0 = 4
```

Benutzen Sie den Funktionskopf:

```
function AltSumme(inRefAnfang: tRefElement): integer;
```

Die Funktion darf die Liste nur genau einmal durchlaufen.

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

Nachklausur am 31.03.2007

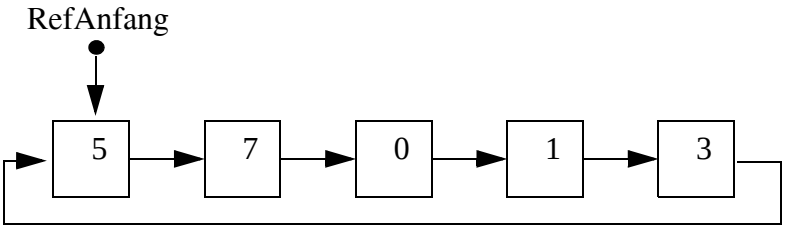
Aufgabe 3 (3+4 Punkte)

Gegeben sei folgende Typdeklaration für Elemente einer linearen Liste:

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

Damit sei nun eine *zyklische* lineare Liste realisiert, indem der `next`-Zeiger des letzten Elementes nicht auf **`nil`**, sondern auf das erste Element zeigt. Es sei nach wie vor ein ganz bestimmtes Element des so entstehenden Ringes das erste Element, welches durch einen Anfangszeiger referenziert wird.

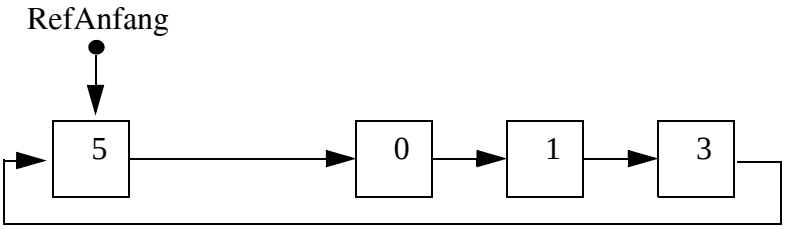
Die folgende Skizze zeigt eine solche Liste mit 5 Elementen:



Sie dürfen bei den folgenden Teilaufgaben jeweils davon ausgehen, dass die übergebene Liste mindestens zwei Elemente besitzt.

a) Schreiben Sie eine Funktion, die das Maximum einer solchen Liste bestimmt. Genauer soll ein Zeiger auf das *Vorgängerelement* desjenigen Elements mit dem größten `info`-Wert zurückgegeben werden. In obger Abbildung wäre beispielsweise die Zahl 7 das Maximum, es müsste also ein Zeiger auf das erste Element zurückgegeben werden. Benutzen Sie folgenden Funktionskopf:
`function` `vorMaximum(inRefAnfang: tRefElement): tRefElement;`

b) Schreiben Sie eine Prozedur, die das größte Element einer solchen Liste löscht, indem sie dessen Vorgänger mit Hilfe der Funktion `vorMaximum` bestimmt und dessen Nachfolger aus der Liste entfernt. Für obiges Beispiel würde die Liste nach der Löschoperation wie folgt aussehen:



Benutzen Sie folgenden Prozedurkopf:
`procedure` `loescheMaximum(`**`var`** `ioRefAnfang: tRefElement);`

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Aufgabe 4 (4+2 Punkte)

Gegeben seien folgende Typdeklarationen:

```
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.
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 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

Nachklausur am 31.03.2007

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

Gegeben seien folgende Vereinbarungen:

```
const
  FELDMAX = 4;

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

function enthaeltNull(inFeld: tFeld): boolean;
{ Gibt genau dann true zurueck, wenn inFeld eine 0 enthält. }
var gefunden: boolean;
begin
  gefunden := false;
  i := 1;
  repeat
    i := i + 1;
    if (inFeld[i] = 0) then
      gefunden := true;
  until gefunden or (i = FELDMAX);
  enthaeltNull := gefunden;
end;
```

Weiterhin seien folgende Testfälle gegeben:

T1 = { ([2, 0, 4, 5], true) }
T2 = { ([1, 2, 0, -1], true) }
T3 = { ([2, 0, 4, 5], true), ([0, 2, 3, 4], true)}

Ein Quadrupel in eckigen Klammern stelle dabei jeweils eine Ausprägung eines Arrays vom Typ `tFeld` dar. [2, 0, 4, 5] stehe beispielsweise für ein Array a mit a[1] = 2, a[2] = 0, a[3] = 4 und a[4] = 5.

a) Anweisungsüberdeckungstest (C_0-Test):
   Jeder der drei genannten Testfälle führt zu einer vollständigen Anweisungsüberdeckung. Decken die Tests zu diesen Testfällen also keine Fehler auf? Begründen Sie Ihre Antwort.

b) Zweigüberdeckungstest (C_1-Test):
   Welche der drei Testfälle führen *nicht* zu einer vollständigen Zweigüberdeckung? Begründen Sie Ihre Auswahl, indem Sie die nicht überdeckten Zweige angeben.

c) Einfacher Bedingungsüberdeckungstest (C_2-Test):
   Welche der drei Testfälle führen *nicht* zu einer vollständigen einfachen Bedingungsüberdeckung? Begründen Sie Ihre Auswahl, indem Sie die nicht überdeckten einfachen (atomaren) Bedingungen angeben.

d) Umfasst der C_2-Test allgemein immer den C_1-Test?

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Kurs 1613 “Einführung in die imperative Programmierung”

Nachklausur am 31.03.2007

Name: _____ Matrikelnummer: _____

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. Einzige Ausnahme sind Modul-lokale Variablen, die in den Parameterlisten der exportierten Prozeduren und Funktionen des Moduls nicht auftauchen, selbst wenn sie von diesen geändert werden.
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 Schlüsselworte **`unit`**, **`interface`** und **`implementation`** werden ebenso wie **`begin`** und **`end`** des Initialisierungsteils linksbündig positioniert. Nach dem Schlüsselwort **`unit`** folgt ein Kommentar, der die Aufgabe beschreibt, welche die Unit löst.
16. Für die Schnittstelle gelten dieselben Programmierstilregeln wie für ein Programm. Dies betrifft Layout und Kommentare. Nach dem Schlüsselwort **`interface`** folgt im Normalfall kein Kommentar.
17. Für den Implementationsteil gelten dieselben Programmierstilregeln wie für ein Programm. Nach dem Schlüsselwort **`implementation`** folgt nur dann ein Kommentar, wenn die Realisierung einer Erläuterung bedarf (z.B. wegen komplizierter Datenstrukturen und/oder Algorithmen).
18. In Programmen oder Moduln, die andere Moduln importieren („benutzen"), wird das Schlüsselwort **`uses`** auf dieselbe Position eingerückt wie die Schlüsselworte **`const`**, **`type`**, **`var`** usw.
19. Die Laufvariable wird innerhalb einer **`for`**-Anweisung nicht manipuliert.
20. Die Grundsätze der strukturierten Programmierung sind strikt zu befolgen.