FernUniversität in Hagen

Bitte hier unbedingt Matrikelnummer und Adresse eintragen, sonst keine Bearbeitung möglich.

Postanschrift: FernUniversität D - 58084 Hagen

Name, Vorname

Straße, Nr.

PLZ, Wohnort

FERNUNIVERSITÄT
- Gesamthochschule -
EINGANG

INF

FERNUNIVERSITÄT
Gesamthochschule
D-58084 Hagen

Fachbereich Informatik

Kurs: 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 5. März 2005

Hörerstatus:

( ) Vollzeitstudent
( ) Teilzeitstudent
( ) Zweithörer
( ) Gasthörer

Klausurort:

( ) Berlin
( ) Bochum
( ) Frankfurt
( ) Hamburg
( ) Karlsruhe
( ) Köln
( ) München
( ) Bregenz
( ) Wien
( ) . . . . . .

Aufgabe	1	2	3	4	5	Summe
erreichbare Punktzahl	6	7	5	9	5	32
bearbeitet						
erreichte Punktzahl						

Datum: _____ Korrektur: _____

FernUniversität in Hagen

Bitte hier unbedingt Matrikelnummer und Adresse eintragen, sonst keine Bearbeitung möglich.

Postanschrift: FernUniversität D - 58084 Hagen

Name, Vorname

Straße, Nr.

PLZ, Wohnort

FERNUNIVERSITÄT
- Gesamthochschule -
EINGANG

INF

FERNUNIVERSITÄT
Gesamthochschule
D-58084 Hagen

Fachbereich Informatik

Kurs: 1613 „Einführung in die imperative Programmierung“

Hauptklausur am 5. März 2005

Hörerstatus:

( ) Vollzeitstudent
( ) Teilzeitstudent
( ) Zweithörer
( ) Gasthörer

Klausurort:

( ) Berlin
( ) Bochum
( ) Frankfurt
( ) Hamburg
( ) Karlsruhe
( ) Köln
( ) München
( ) Bregenz
( ) Wien
( ) . . . . . .

Aufgabe	1	2	3	4	5	Summe
erreichbare Punktzahl	6	7	5	9	5	32
bearbeitet						
erreichte Punktzahl						

Datum: _____ Korrektur: _____

FernUniversität in Hagen

FACHBEREICH
Informatik
Lehrgebiet
Software Engineering
Prof. Dr. H.-W. Six

FernUniversität in Hagen · 58084 Hagen

Ihr Zeichen
Ihre Nachricht vom
Mein Zeichen
Auskunft erteilt I. Schulz-Gerlach
Telefon 02331 987-2553
Telefax 02331 987-317
E-Mail Kurs1612@FernUni-Hagen.de
Hausanschrift Universitätsstr. 1
58084 Hagen
Datum 05.03.2005

BESCHEINIGUNG

(zur Vorlage beim Finanzamt)

Hiermit wird bescheinigt, dass

Herr / Frau _____

Matrikel-Nr. _____

am 5. März 2005 in der Zeit von 10.00 bis 12.00 Uhr

in _____

an der Klausur zum Kurs 1613
“Einführung in die imperative Programmierung”
teilgenommen hat.

Telefonzentrale 02331 987-01
Zentraler Telefaxeingang 02331 987-316
Internet http://www.FernUni-Hagen.de
Buslinien 515
Haltestelle FernUniversität

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Wintersemester 2004/2005
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,
   - 5Aufgaben (Seite 2 - Seite 16),
   - 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 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"

Klausur am 05.03.2005

Aufgabe 1 (6 Punkte)

Zu einer Fernsehshow haben sich 50 Kandidaten gemeldet. Nach einem „Zufallsprinzip" sollen einige für die Show ausgewählt werden. Dazu stellen sich zunächst alle Kandidaten in einer Reihe auf. Die Auswahl geschieht nun nach folgendem Verfahren:

- In einem ersten Durchlauf tritt jeder Kandidat einen Schritt vor.
- Im zweiten Durchgang tritt jeder zweite Kandidat, also die Kandidaten mit den Nummern 2, 4, 6, 8 usw., wieder einen Schritt zurück.
- Danach tritt jeder dritte Kandidat (3, 6, 9, 12 usw.) einen Schritt vor, falls er hinten steht, und einen Schritt zurück, falls er vorne steht.
- Als nächstes tritt jeder 4. Kandidat (4, 8, 12, 16 usw.) einen Schritt vor, falls er hinten steht, und einen Schritt zurück, falls er vorne steht.
- Dieses Verfahren wird bis zum 50. Durchgang fortgesetzt. Es tritt also beim Durchlauf i jeder i-te Kandidat (i, 2i, 3i, 4i usw.) einen Schritt vor, falls er hinten steht, und einen Schritt zurück, falls er vorne steht.
- Alle Kandidaten, die nach dem 50. Durchlauf vorne stehen, dürfen an der Show teilnehmen.

Da es zu teuer ist, alle 50 Kandidaten nur für das Auswahlverfahren anreisen zu lassen, soll ein entsprechendes Programm zur Simulation implementiert werden. Dazu werden alle Kandidaten von 1 bis 50 durchnumeriert. Unter Beachtung der angegebenen Typdeklarationen sollen Sie untenstehende Prozedur `BerechneZustand` an den mit **`??`** gekennzeichneten Stellen ergänzen.

Hinweis: Sie werden dazu zwei Schleifen verschachteln müssen: Die äußere Schleife steuert die 50 Durchläufe des Verfahrens, und im i-ten Durchlauf (i = 1, 2, ..., 50) ist mit einer inneren Schleife jeweils der i-te, der 2i-te, der 3i-te u.s.w. Kandidat zu bestimmen, welcher seinen Zustand ändern soll. Überlegen Sie sich vorab, welcher Schleifentyp für die äußere und welcher für die innere Schleife jeweils am besten geeignet ist.

```
const
MAXKANDIDAT = 50;

type
tNatZahl = 1..maxint;
tZustand = (vorn, hinten);
tKandidatenBereich = 1..MAXKANDIDAT;
tKandidaten = array [tKandidatenBereich] of tZustand;
```

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Name: _____ Matrikelnummer: _____

```
procedure BerechneZustand (?? ??Feld : tKandidaten);
{ berechnet die zugelassenen und abgelehnten Kandidaten }
    var
    i: tKandidatenBereich; {Durchlauf-Zaehler}
    j: tNatZahl; {Im i-ten Durchlauf koennen hiermit die
                  Kandidaten (Vielfache von i) abgezaehlt werden}

  begin
    { Initialisierung: alle Kandidaten stehen „hinten“ }
    for i:= 1 to MAXKANDIDAT do
      outFeld[i] := hinten;

    { Durchlaeufe starten... }

    ??

  end; { BerechneZustand }
```

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

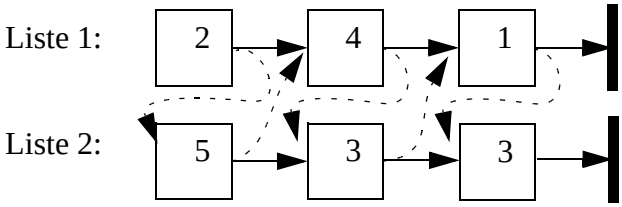
Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

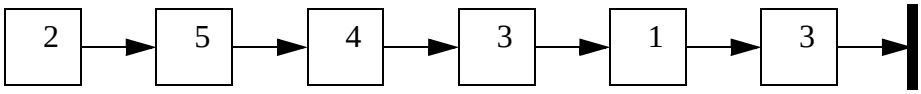
Klausur am 05.03.2005

Aufgabe 2 (7 Punkte)

Schreiben Sie eine PASCAL-Funktion `ListenMerge`, die zwei lineare Listen mit `integer`-Zahlen nach dem aus dem Straßenverkehr bekannten „Reißverschlussverfahren" *nur durch Änderung der Verkettung* zu einer einzigen Liste zusammenfasst. Als Rückgabewert wird der Zeiger auf das erste Element der Ergebnisliste geliefert. Dabei soll das erste Element von Liste 1 an den Listenanfang kommen, wie im folgenden Beispiel gezeigt ist (die gestrichelten Pfeile deuten die Änderung der Verkettung an):



Resultierende Liste:



Sie können davon ausgehen, dass die beiden Listen **nicht-leer** und **gleich lang** sind. Benutzen Sie folgende Typdefinitionen sowie den angegebenen Funktionskopf von `ListenMerge`:

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

function ListenMerge ( inListe1,
                       inListe2 : tRefElement) : tRefElement;
{ fasst die Elemente der nicht-leeren und gleich langen Listen
        inListe1 und inListe2 mit dem Reissverschlussverfahren durch
        Aenderung der Verkettung zu einer einzigen Liste zusammen
        und gibt einen Zeiger auf den Anfang der Ergebnisliste
        zurueck }
```

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

Klausur am 05.03.2005

Aufgabe 3 (5 Punkte)

Gegeben ist folgende Typdeklaration für eine lineare Liste:

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

Es soll eine Prozedur `NachVorn` implementiert werden, die ein Element mit einem als Parameter übergebenen Info-Wert (vom Typ `integer`) sucht und dieses an den Anfang der Liste *nur durch Ändern der Verkettung* einfügt.
Es darf dabei vorausgesetzt werden, dass ein solches Element in der Liste vorkommt!

Benutzen Sie untenstehenden Prozedurkopf und ergänzen Sie auch die Parameterübergabearten.

```
procedure NachVorn( ?? ??Wert: integer;
                    ?? ??ioRefAnfang: tRefElement);

??
```

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

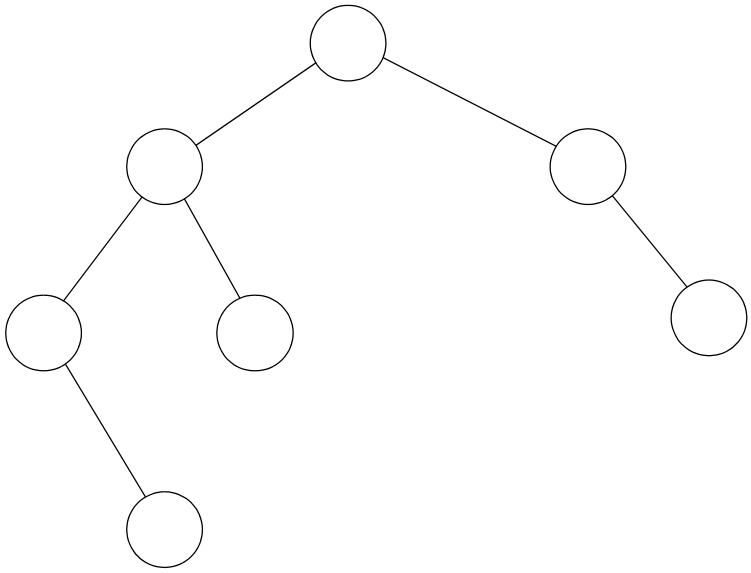
Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

Klausur am 05.03.2005

Aufgabe 4 (7+2 Punkte)

Gegeben sei ein (nicht sortierter) binärer Baum und ein Zeiger (inRefWurzel) auf die Wurzel des Baumes. Die Funktion `countLeaf` soll die Anzahl der Blätter in dem Baum bestimmen. Im Fall eines *leeren Baumes* soll der Wert 0 zurückgegeben werden.
Beispiel: Wird der Funktion `countLeaf` ein Zeiger auf die Wurzel des Baumes der folgenden Abbildung übergeben, so liefert sie als Ergebnis den Wert 3.



a) Ihre Aufgabe ist es, die Funktion `countLeaf` zu implementieren. Gehen Sie dabei von den unten gegebenen Typdefinitionen und der Funktionsdeklaration aus.
Wählen Sie eine *rekursive* Lösung!

b) Zeichnen Sie für die Menge {36, 11, 18, 3, 37, 25, 8, 21} einen degenerierten und einen vollständigen Suchbaum.

```
type
  tNatZahl = 0..maxint;
  tRefBinBaum = ^tBinBaum;
  tBinBaum = record
                info : tNatZahl;
                links,
                rechts : tRefBinBaum
             end;
```

Name: _____ Matrikelnummer: _____

```
function countLeaf(??): ??;
{ ?? }
 ...
```

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Name: _____ Matrikelnummer: _____

Kurs 1613 "Einführung in die imperative Programmierung"

Klausur am 05.03.2005

Aufgabe 5 (5 Punkte)

Die Funktion MaxZiffer soll die größte Ziffer der Dezimaldarstellung der nicht-negativen ganzen Zahl `inZahl` bestimmen.

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

function MaxZiffer (inZahl:tNatZahl): tZiffer;
{ bestimmt die größte Ziffer von inZahl}

  var
  rest : tnatZahl;
  maxi,
  ziffer: tZiffer;

begin
  rest := inZahl;
  maxi := 2;
  while rest > 0 do
  begin
    ziffer := rest mod 10;
    if ziffer > maxi then
      maxi := ziffer;
    rest := rest div 10
  end;
  MaxZiffer := maxi
end; { MaxZiffer }
```

Ermitteln Sie die Testfälle für den *boundary-interior Pfadtest* (interior-Test mit n=2) der Funktion `MaxZiffer`. Sie brauchen nicht die zu den Testfällen gehörenden Pfade im Kontrollflussgraphen anzugeben.

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Name: _____ Matrikelnummer: _____

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

Kurs 1613 “Einführung in die imperative Programmierung”

Klausur am 05.03.2005

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.