Kurs 1613 „Einführung in die imperative Programmierung“
Nachklausur 05.03.2011

Name: _____ Matrikelnr.: _____

Wintersemester 2010/2011
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 2-20)
   - die Muss-Regeln des Programmierstils.
2. Füllen Sie, **bevor** Sie mit der Bearbeitung der Aufgaben beginnen, folgende Seiten des Klausurexemplares aus:
   - Beide Deckblätter mit Namen, Anschrift sowie Matrikelnummer. **Markieren Sie vor der Abgabe auf beiden Deckblättern die von Ihnen bearbeiteten Aufgaben.**
   - 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** und **keinen Rotstift**!) sind **keine weiteren Hilfsmittel** zugelassen. Die Muss-Regeln des Programmierstils finden Sie im Anschluss an die Aufgabenstellung.
6. Es sind maximal 36 Punkte erreichbar. Sie haben die Klausur sicher dann bestanden, wenn Sie mindestens 18 Punkte haben.

Wir wünschen Ihnen bei der Bearbeitung der Klausur viel Erfolg!

Aufgabe 1 (2+4 Punkte)

Gegeben sei das folgende Programm:

```
program WasPassiert(input,output);
{                                                                    }
  type
  tNatZahl=1..maxint;
  var
  a:tNatZahl;
  b:tNatZahl;
  c:integer;
begin
  writeln('Geben Sie zwei natürliche Zahlen ein:');
  readln(a);
  readln(b);
  c:=0;
  while (b > 1) do
  begin
    if (b mod 2 = 1) then
    begin
      c:=c+a;
      b:=b-1
    end;
    a:=2*a;
    b:=b div 2
  end;
  writeln('Ergebnis: ',a+c)
end.
```

Überlegen Sie sich was das Programm leistet und wie es dabei vorgeht.

a) Was gibt das Programm für die Eingaben a=3 und b=4 aus? `'Ergebnis:          '`
   Was gibt das Programm für die Eingaben a=6 und b=7 aus? `'Ergebnis:          '`
b) Ergänzen Sie im Programm einen erklärenden Kommentar an der grau eingefärbten Stelle und schreiben Sie eine passende Problemspezifikation:

Eingabe:

Ausgabe:

Nachbedingung:

Name: _____ Matrikelnr.: _____

Aufgabe 2 (5 Punkte)

Schreiben Sie ein Programm, welches eine Real-Zahl `Betrag` und zwei Integer-Zahlen `Zinsen` und `Laufzeit` einliest. Das Programm erzeugt daraufhin für jedes Jahr der Laufzeit eine Zeile, die das angesparte Geld beschreibt, wenn der Geldbetrag `Betrag` mit einer jährlichen Verzinsung in Höhe der eingegebenen Zinsen in Prozent angelegt wird. Zinseszinsen sollen berücksichtigt werden.

Für die Zahlen Betrag=100, Zinsen=3 und Laufzeit= 4 ergibt sich folgende Ausgabe:

```
Nach 1 Jahren 103.00
Nach 2 Jahren 106.09
Nach 3 Jahren 109.27
Nach 4 Jahren 112.55
```

Name: _____ Matrikelnr.: _____

Aufgabe 3 (6 Punkte)

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

```
const
FELDGROESSE=5;

type
tIndex=1..FELDGROESSE;
tFeld=array[tIndex] of integer;
```

Schreiben Sie eine Funktion, die zwei Felder vom Typ tFeld als Eingabe bekommt und einen boolean Wert zurückgibt, der anzeigt, ob alle Werte des ersten Feldes auch im zweiten Feld enthalten sind.

Als Beispiele ergeben
die Eingaben `[3,4,5,5,5]` und `[5,2,3,4,1]` den Wert `true`, währenden
die Eingaben `[1,2,3,3,1]` und `[1,2,4,4,5]` den Wert `false` ergeben.

Name: _____ Matrikelnr.: _____

Name: _____ Matrikelnr.: _____

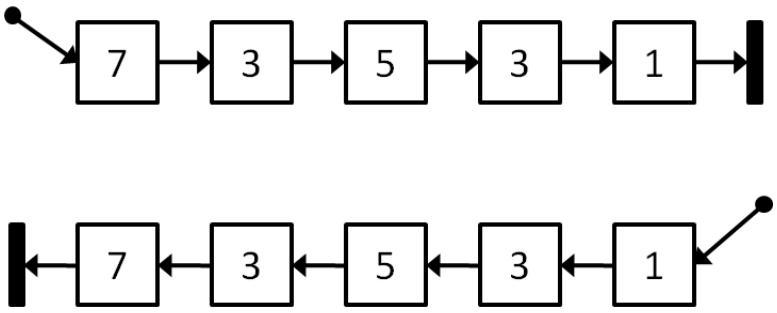
Aufgabe 4 (7 Punkte)

Gegeben sei die folgende Typvereinbarung:

```
type
tRefListe=^tListe;
tListe=record
         Wert:integer;
         next:tRefListe
       end;
```

Die Typen `tRefListe` und `tListe` dienen zur Bildung einer linearen Liste von Integer-Zahlen. Schreiben Sie eine iterative Prozedur ListeUmdrehen, die eine Liste übergeben bekommt und diese umdreht. Dabei soll nur die Verkettung der Elemente verändert werden, es sollen weder neue `tListen`-Elemente erzeugt, noch Werte von bestehenden `tListen`-Elementen verändert werden.

Die folgende Abbildung zeigt ein Beispiel einer linearen Liste oben vor und unten nach dem Aufruf der Prozedur ListeUmdrehen:



Name: _____ Matrikelnr.: _____

Name: _____ Matrikelnr.: _____

Aufgabe 5 (6 Punkte)

Gegeben seien folgende Typdefinitionen für einen Binärbaum von Integer Zahlen:

```
type
tRefBinBaum=^tBinBaum;
tBinBaum=record
           Wert:Integer;
           links,rechts:tRefBinBaum
         end;
```

Schreiben Sie eine rekursive Funktion, die entscheidet, ob ein Integer-Wert in einem Baum vom Typ `tRefBinBaum` enthalten ist. Hierzu soll die Funktion einen Boolean-Wert zurückgeben.

Name: _____ Matrikelnr.: _____

Aufgabe 6 (3+3 Punkte)

Betrachten Sie nochmals das Programm aus Aufgabe 1.

```
1     program WasPassiert(input,output);
2       type
3       tNatZahl=1..maxint;
4       var
5       a:tNatZahl;
6       b:tNatZahl;
7       c:integer;
8     begin
9       writeln('Geben Sie zwei natürliche Zahlen ein:');
10      readln(a);
11      readln(b);
12      c:=0;
13      while (b > 1) do
14      begin
15        if (b mod 2 = 1) then
16        begin
17          c:=c+a;
18          b:=b-1
19        end;
20        a:=2*a;
21        b:=b div 2
22      end;
23      writeln('Ergebnis: ',a+c)
24    end.
```

a) Erstellen Sie den kompakten Kontrollflussgrafen für das Programm WasPassiert. Geben Sie dabei zu jedem Knoten an, welche Programmzeilen von diesem Knoten repräsentiert werden.

b) Angestrebt wir ein Boundary-interior-Pfadtest. Es gibt einen Pfad für 0 Schleifendurchläufe, zwei Pfade für 1 Schleifendurchlauf und vier Pfade für 2 Schleifendurchläufe. Geben Sie für jeden dieser sieben Pfade ein Eingabendatum (a,b) an:

Eingabedatum für 0 Schleifendurchläufe:

Eingabedaten für 1 Schleifendurchlauf:

Eingabedaten für 2 Schleifendurchläufe:

Name: _____ Matrikelnr.: _____

Name: _____ Matrikelnr.: _____

Zusammenfassung der Muss-Regeln

1. Selbstdefinierte Konstantenbezeichner bestehen nur aus Großbuchstaben. Bezeichner von Standardkonstanten wie z.B. `maxint` sind also ausgenommen.
2. Typenbezeichnern wird ein `t` vorangestellt. Bezeichnern von Zeigertypen wird ein `tRef` vorangestellt. 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 Ausgabeparameter.
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.