Kurs 1613 "Einführung in die imperative Programmierung"

Musterlösung zur Klausur am 05.03.2005

Aufgabe 1

In einer ersten **for**-Schleife wird das Kandidatenfeld initialisiert, so dass alle Kandidaten „hinten" stehen.
Eine zweite **for**-Schleife simuliert die 50 Durchgänge des Auswahlverfahrens. Dabei werden im i-ten Durchlauf (i = 1, ..., MAXKANDIDAT) jeweils mit einer inneren **while**-Schleife die Vielfachen von i abgezählt, also die Kandidaten, die ihre Position ändern müssen, indem sie vortreten, falls sie „hinten" stehen, bzw. zurücktreten, falls sie „vorne" stehen.
Am Ende des Verfahrens stehen alle Kandidaten „vorne", deren Nummern Quadratzahlen sind.

```
procedure BerechneZustand (var outFeld : tKandidaten);
{ berechnet die zugelassenen und abgelehnten Kandidaten }

    var
    i: tKandidatenBereich; {Durchlaeufe}
    j: tNatZahl; {Im i-ten Durchlauf werden hiermit die
                  Kandidaten (Vielfache von i) abgezaehlt}

begin
  { Initialisierung: alle Kandidaten stehen „hinten" }
  for i:= 1 to MAXKANDIDAT do
    outFeld[i] := hinten;
  { Durchlaeufe starten... }
  for i:= 1 to MAXKANDIDAT do
  begin
    j:=i; {erstes Vielfaches von i}
    while j <= MAXKANDIDAT do
    begin
      if outFeld[j] = vorn then
        outFeld[j] := hinten
      else
        outFeld[j] := vorn;
      j := j + i; {naechstes Vielfaches von i}
    end; { while }
  end { for }
end; { BerechneZustand }
```

Aufgabe 2

Zur Lösung unseres Problems durchlaufen wir `inListe1` mit den Zeigern `Zeiger1` und `ZeigHilf1` und `inListe2` mit den Zeigern `Zeiger2` und `ZeigHilf2`, wobei stets `Zeiger1^.next=ZeigHilf1` und `Zeiger2^.next=ZeigHilf2` gilt. Dabei ändern wir die Folgezeiger der Listenelemente, bis das Ende von `inListe1` und damit von

---

`inListe2` erreicht ist. Die angegebene Funktion arbeitet auch für den Fall ungleich langer Listen. Wir geben zusätzlich ein Rahmenprogramm an, welches den Test der Funktion `ListenMerge` ermöglicht:

```
program Aufgabe (input, output);
{ Testprogramm fuer die Funktion ListenMerge }

  type
  tRefElement = ^tElement;
  tElement    = record
                  info : integer;
                  next : tRefElement
                end;
  tNatZahlPlus = 1..maxint;

  var
  Liste,
  Liste1,
  Liste2 : tRefElement;
  Laenge : tNatZahlPlus;

  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
    zurück }

    var
    Zeiger1,
    Zeiger2,
    ZeigHilf1,
    ZeigHilf2 : tRefElement;

  begin
    Zeiger1 := inListe1;
    Zeiger2 := inListe2;
    while (Zeiger1 <> nil) and (Zeiger2 <> nil) do
    begin
      ZeigHilf1 := Zeiger1^.next; { 1) Nachfolger sichern }
      ZeigHilf2 := Zeiger2^.next;
      Zeiger1^.next := Zeiger2;   { 2) Verkettung aendern }
      Zeiger2^.next := ZeigHilf1;
```

```
    Zeiger1 := ZeigHilf1;       { 3) Zeiger weiterruecken }
    Zeiger2 := ZeigHilf2
  end;
  ListenMerge := inListe1;  { Neuen Anfang ausgeben }
end; { ListenMerge }

procedure ListeAufbauen (
                inAnz : tNatZahlPlus;
            var ioListe : tRefElement );
{ liest inAnz Zahlen von der Tastatur ein und speichert sie in
  der Reihenfolge der Eingabe in der linearen Liste ioListe }

  var
  Zahl : integer;
  AnfListe,
  EndListe,
  hilf : tRefElement;
  k : tNatZahlPlus;

begin
  AnfListe := nil;
  writeln ('Bitte ', inAnz, 'natuerliche Zahlen eingeben:');
  for k := 1 to inAnz do
  begin
    read (Zahl);
    new (hilf);
    hilf^.info := Zahl;
    hilf^.next := nil;
    if AnfListe = nil then
    begin
    { erste Zahl }
      AnfListe := hilf;
      EndListe := hilf
    end
    else
    begin
    { zweite und weitere Zahlen }
      EndListe^.next := hilf;
      EndListe := hilf
    end
  end;
  ioListe := AnfListe
end; { ListeAufbauen }
```

```
begin
  writeln ('Bitte die Listenlaenge eingeben (>0):');
  readln (Laenge);
  ListeAufbauen (Laenge, Liste1);
  ListeAufbauen (Laenge, Liste2);
  Liste := ListenMerge (Liste1, Liste2);
  writeln ('Die "verkettete" Liste lautet:');
  repeat
    write (Liste^.info);
    Liste := Liste^.next
  until Liste = nil
end. { Aufgabe}
```

Aufgabe 3

```
  procedure NachVorn( inWert: integer;
                      var ioRefAnfang: tRefElement);
  {Sucht das erste vorkommende Element mit inWert in der
  info-Komponente und verschiebt dieses Element an den Anfang
  der Liste.
  ACHTUNG: Ein solches Element muss in der Liste vorkommen!}

    var lauf,
        elem: tRefElement;

  begin
    {Sonderfallprüfung: Falls das erste Element den Wert
    inWert hat, so ist nichts zu tun.}
    if (ioRefAnfang^.info <> inWert) then
    begin
      {Den Vorgaenger des zu verschiebenden Elementes suchen:}
      lauf := ioRefAnfang;
      while lauf^.next^.info <> inWert do
        lauf := lauf^.next;
      elem := lauf^.next;
      {elem zeigt nun auf das zu verschiebende Element,
       lauf auf dessen Vorgaenger. Jetzt verschieben:}
      lauf^.next := elem^.next;
      elem^.next := ioRefAnfang;
      ioRefAnfang := elem;
    end; {if}
  end; {NachVorn}
```

Laut Aufgabenstellung durfte vorausgesetzt werden, dass der Suchwert in der Liste vorkommt, was insbesondere bedeutet, dass die Liste nicht leer sein kann, und dass, falls das Element nicht am Anfang steht, die Liste mindestens zwei Elemente hat. Die While-Bedingung oben ist also unter dieser Voraussetzung problemlos, es kann nicht zu einer Dereferenzierung von NIL kommen.

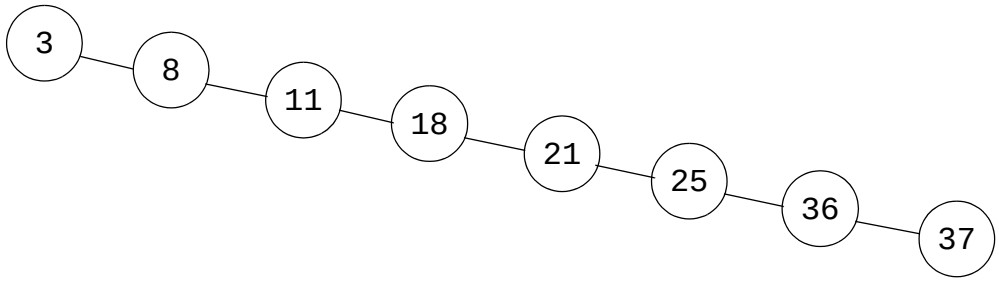
Aufgabe 4

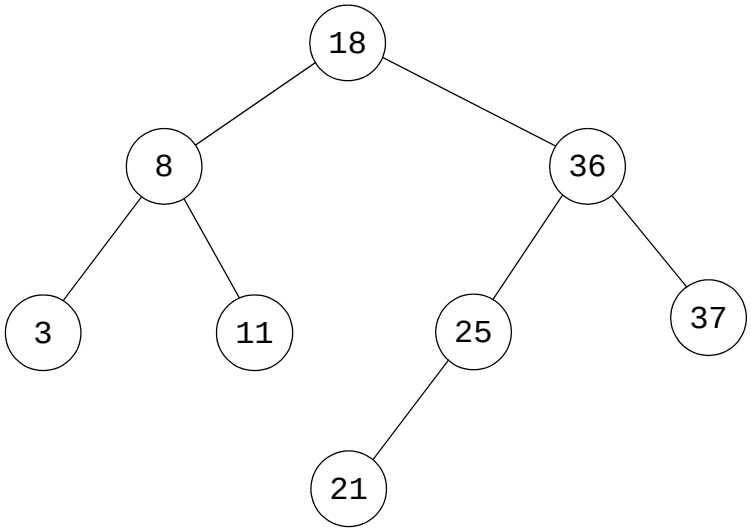
a)

```
function countLeaf(inRefWurzel : tRefBinBaum ): tNatZahl;
{die Funktion liefert die Anzahl der Blätter des inRefWurzel}

begin
  if inRefWurzel = nil then
    countLeaf := 0
  else
    if (inRefWurzel^.links = nil) and
                          (inRefWurzel^.rechts = nil) then
      countLeaf := 1
    else
      countLeaf := countLeaf(inRefWurzel^.links) +
                          countLeaf(inRefWurzel^.rechts);
end; { countLeaf }
```

b)





Aufgabe 5

Ein Testfall ist eine Menge von Testdaten, wobei ein Testdatum wiederum ein Paar ist, welches zum Ersten die Eingabedaten (hier: Eingabe für Parameter inZahl) und zum Zweiten die bei dieser Eingabe laut Spezifikation *erwartete* Ausgabe enthält:

Abweisender Test:	$T_0 = \{ (0, 0) \}$
Boundary-Test:	$T_1 = \{ (e, e) \mid e \in \{1, ..., 9\} \}$
Interior-Test (2 Durchläufe):	$T_2 = \{ (e, a) \mid e \in \{10, ..., 99\} \wedge$ a ist die größere der zwei Ziffern von e$\}$