Kurs 1613 "Einführung in die imperative Programmierung"

Klausur am 01.03.2008

Aufgabe 1 (2 + 4 Punkte)

Gegeben seien folgende Typvereinbarungen sowie die unkommentierte Funktion `indexVon`.

```
const
  MAXINDEX = 100;
type
  tIndex1 = 1..MAXINDEX;
  tIndex0 = 0..MAXINDEX;
  tFeld = array[tIndex1] of integer;

function indexVon(var inFeld:tFeld; inWert:integer):tIndex0;
{Funktionsbeschreibung: Siehe Teilaufgabe a)}
var
  ergebnis:tIndex0;
  i:tIndex1;
begin
  ergebnis := 0;
  for i:=1 to MAXINDEX do
    if (ergebnis = 0) and (inFeld[i] = inWert) then
      ergebnis := i;
  indexVon := ergebnis;
end;
```

a) Beschreiben Sie kurz das Ergebnis der Funktion `indexVon` in Abhängigkeit von den Eingaben `inFeld` und `inWert`. Beachten Sie dabei auch ggf. vorhandene Sonderfälle.

b) Die For-Schleife in obiger Implementierung absolviert immer genau `MAXINDEX` Durchläufe, obwohl das Ergebnis oft schon nach deutlich weniger Durchläufen, im Extremfall sogar schon nach dem ersten Durchlauf feststeht.
Schreiben Sie eine effizientere Variante der Funktion mit Hilfe einer While-Schleife, die abbricht, sobald das Ergebnis feststeht.
Verwenden Sie den folgenden Rahmen:

```
function indexVon_Neu(var inFeld:tFeld; inWert:integer):tIndex0;
{Funktionsbeschreibung: Siehe Teilaufgabe a)}
var
  ergebnis:tIndex0;
  i:tIndex0;
begin
  {Anweisungsteil: Siehe Teilaufgabe b)}
end;
```

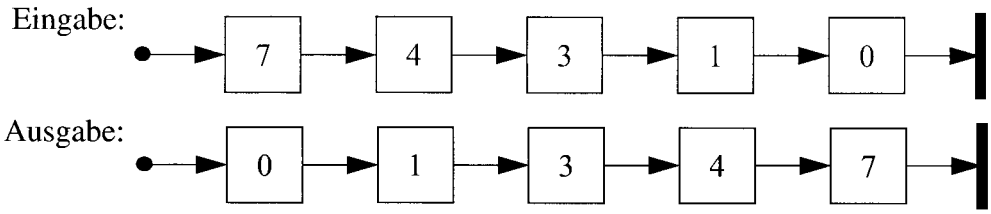
Aufgabe 2 (5 Punkte)

Gegeben seien folgende Typvereinbarungen zur Beschreibung einfach verketteter linearer Listen:

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

Schreiben Sie eine *iterative* Funktion `gespiegelteKopieVon`, die eine solche Liste übergeben bekommt und als Ergebnis eine neu erzeugte Liste ausgibt, welche die gleichen Elemente in jedoch umgekehrter Reihenfolge enthält. Die eingegebene Liste soll unverändert weiter existieren, darf also nicht verändert werden!

Die folgende Abbildung soll das Problem beispielhaft verdeutlichen:



Die eingegebene Quell-Liste darf dazu nur einmal durchlaufen werden!

Hinweis: Überlegen Sie sich zunächst, wo die Kopien der Elemente in die neu aufzubauende Liste einzufügen sind, um den Spiegelungseffekt zu erzielen.

Verwenden Sie folgenden Funktionskopf:

```
function gespiegelteKopieVon(inRefOriginal: tRefElement):tRefElement;
{Erzeugt eine gespiegelte Kopie der durch inRefOriginal referenzierten
Liste. Die Original-Liste bleibt unverändert.}
```

Kurs 1613 "Einführung in die imperative Programmierung"

Klausur am 01.03.2008

Aufgabe 3 (7 Punkte)

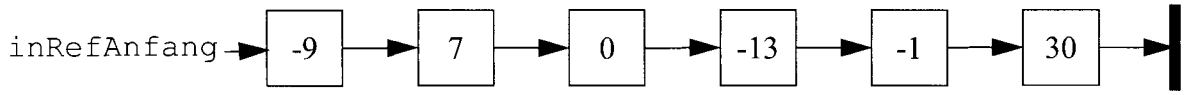
Gegeben ist folgende Typdeklaration für eine lineare Liste:

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

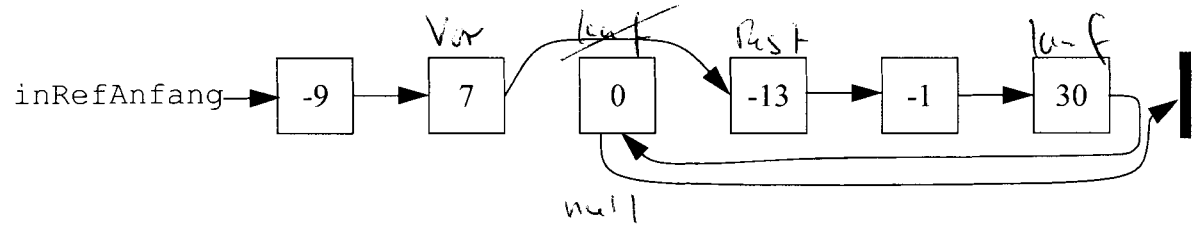
Implementieren Sie eine Prozedur `Tausche`, die in einer übergebenen Liste das Element mit `info`-Komponente 0 sucht und dieses *nur durch Ändern der Verkettung* an das Listenende setzt. Sie können davon ausgehen, dass die 0 in der Liste genau einmal, aber nicht als erstes Element vorkommt. Verwenden Sie folgenden Prozedurkopf:

```
procedure Tausche(inRefAnfang : tRefElement);
```

Ein Beispiel:
Vor dem Aufruf:



Danach:



Aufgabe 4 (2+3+1 Punkte)

Gegeben seien folgende Typvereinbarungen:

```
type
  tRefBinBaum = ^tBinBaum;
  tBinBaum =  record
                info: integer;
                links: tRefBinBaum;
                rechts: tRefBinBaum
              end;
```

a) Stellen obige Typvereinbarungen sicher, dass ein Element des Typs `tBinBaum` einen Knoten eines binären Baums bildet (also die Wurzel eines (Teil-)Baums darstellt)? Begründen Sie kurz Ihre Antwort.

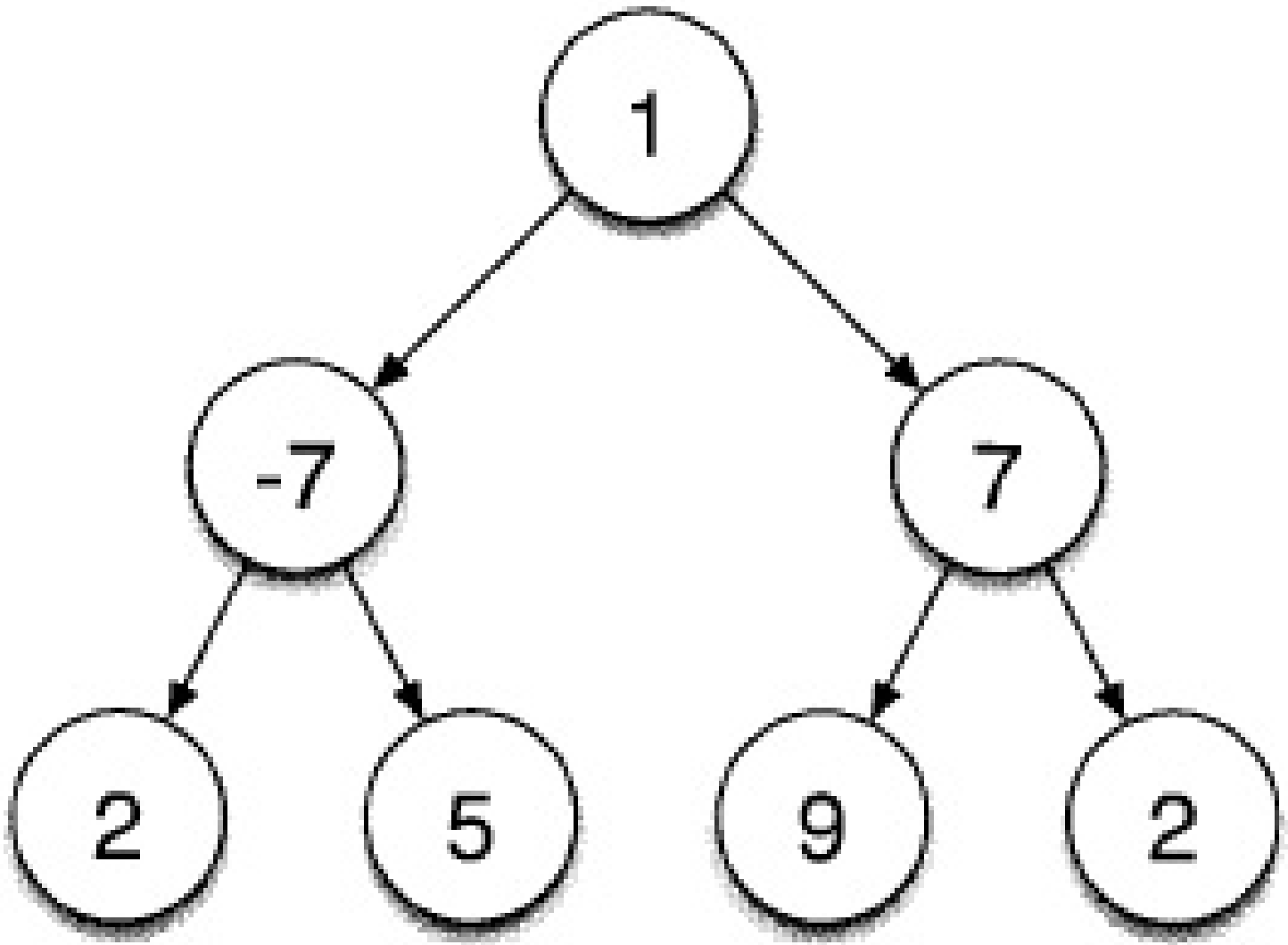
Weiterhin sei folgende Prozedur gegeben:

```
procedure whatDoIDo(inRefBaum: tRefBinBaum);
var hilf:tRefBinBaum;
begin
  if inRefBaum <> nil then
  begin
    whatDoIDo(inRefBaum^.links);

    hilf := inRefBaum^.links;
    inRefBaum^.links := inRefBaum^.rechts;
    inRefBaum^.rechts := hilf;

    whatDoIDo(inRefBaum^.links);
  end;
end;
```

b) Was leistet die Prozedur `whatDoIDo`?
Welcher Baum entsteht beispielsweise, wenn der auf der folgenden Seite abgebildete Baum an `whatDoIDo` übergeben wird? Zeichnen Sie den resultierenden Baum auf.
Beschreiben Sie kurz (in einem Satz), ob und ggf. inwiefern die Prozedur einen beliebigen ihr übergebenen binären Baum verändert.

c) Handelt es sich bei `whatDoIDo` um eine sinnvolle Anwendung der Rekursion? Begründen Sie Ihre Antwort.

Abbildung zu Teil b):

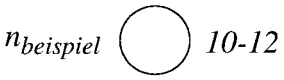


Aufgabe 5 (3+2+2 Punkte)

Gegeben seien folgende Vereinbarungen:

```
const
  FELDMAX = 4;

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

1   function enthaeltNull(inFeld: tFeld): boolean;
    { Gibt genau dann true zurueck, wenn inFeld eine 0 enthält. }
2   var gefunden: boolean;
3       i: integer;
4   begin
5     gefunden := false;
6     i := 0;
7     repeat
8       i := i + 1;
9       if (inFeld[i] = 0) then
10        gefunden := true;
11    until gefunden or (i = FELDMAX);
12    enthaeltNull := not gefunden;
13  end;
```

a) Zeichnen Sie den kompakten Kontrollflussgraphen für die Funktion `enthaeltNull`. Geben Sie dabei zu jedem Knoten an, welche Programmzeilen von diesem Knoten repräsentiert werden, indem Sie die Zeilennummern rechts neben den Knoten schreiben. Notieren Sie links den Namen des Knotens. Beispiel:



b) Angestrebt wird ein Boundary-Interior-Pfadtest. Nennen Sie — sofern möglich — für den keinmaligen, einmaligen und zweimaligen Schleifendurchlauf jeweils ein Testdatum.
Beim Formulieren der Testdaten stellen Sie eine Ausprägung eines Arrays vom Typ `tFeld` bitte als ein Quadrupel in eckigen Klammern 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.

c) Umfasst der Boundary-Interior-Pfadtest (für die Funktion `enthaeltNull`) auch zwangsläufig die vollständige Zweigüberdeckung? Begründen Sie Ihre Antwort.