Kurs 1613 "Einführung in die imperative Programmierung"

Musterlösung zur Klausur am 04.03.2000

Lösung 1 (6+5+3+4= 18 **Punkte)**

a)

```
function countZiff (inZahl : tNatZahl): tNatZahl;
{gibt die Stellenanzahl der inZahl}
var
i, j : tNatZahl ;
begin
i := inZahl ;
J := l;
while i => 10 do begin
J := J + l;
i := i div 10 end;
countZiff := j end;
```

b)

```
function potenziere (inZahl, Potenz: tNatZahl): tNatZahl;
(berechnet die n-te Potenz einer Zahl)
   var
   i, ZwischErg : tNatZahl;
begin
   ZwischErg := l;
   for i := l to Potenz do
      ZwischErg := ZwischErg * inZahl;
   potenziere := ZwischErg
end;
```

c)

In der Funktion isArmstrong ist die Funktion potenziere an der Stelle (2) oder (3) aufzurufen. Eine geeignete Programmzeile ist: ZwischErg := potenziere (Letzteziffer, AnzDerStellen).

d)

```
program ArmstrongZahlen;
var
i, x, y : tNatZahl ;
begin
readln(x) ;
readln(y) ;
for i : = x to y do begin
if isArmstrong(i) then
writeln(i,' ist eine Armstrong Zahl.') ;
end {for} end;
```

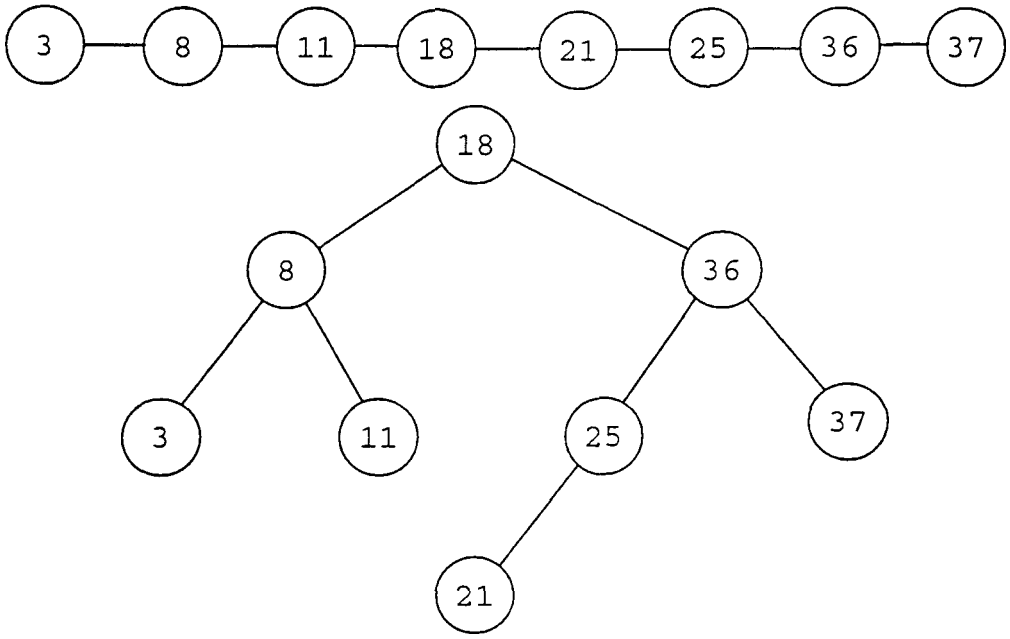
Lösung 2 **(10 + 4 = 14 Punkte)**

a)

```
function countLeaf(inRefWurzel : tRefBinBaum ): tNatZahl;
{die Funktion liefert die Anzahl der Blätter des inRefWurzel}
   var
   Anzahl : tNatZahl ;
begin
   if inRefWurzel = nil then
   Anzahl := 0 eise begin
```

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

b)



Lösung 3 (9 + 9 = 18 Punkte)

Die Schwierigkeit besteht im Aufbau der Liste, da ja am Ende eingefügt werden muß. In der vorgeschlagenen Version wird ein Zeiger auf das Ende der Liste benutzt. (Die Prozedur arbeitet auch, wenn überhaupt kein Element eingegeben wird.) Wir zeigen die Lösung von Teil a) und b) zusammen.

```
program Zahleninversion (input, output) ;

{ erzeugt eine lineare Liste aus einer Folge einzugebender ganzer Zahlen und invertiert diese
Liste mit einein Stapel. }

uses

Stapel;

type

tRefListe = "tListe;

tListe = record

info : integer;

next : tRefListe end;

var

Zahl : integer;

Liste, ZgListe : tRefListe;

{ Loesung von Teil a) der Aufgabe }

procedure Zahlenliste (var outListe : tRefListe ) ;

{ liest integer-Zahlen von der Tastatur ein und speichert
```

```
sie in der Reihenfolge der Eingabe in der linearen Liste
outListe; Null beendet die Eingabe }
var
Zahl : integer;
AnfListe, EndListe, hilf : tRefListe;
begin
AnfListe := nil;
writeln ('Bitte natuerliche Zahlen eingeben, 0=Ende!');
read (Zahl) ;
while Zahl o 0 do begin
new (hilf) ;
hilf ^.info := Zahl;
hilf ^.next := nil;
if AnfListe = nil then begin
{ erste Zahl, Zeiger initialisieren } AnfListe := hilf;
EndListe := hilf end eise begin
{ zweite und weitere Zahlen } EndListe^.next := hilf;
EndListe := hilf
end;
readln (Zahl) end; { while } outListe := AnfListe end; { Zahlenliste }
begin { ab hier wieder Teil b) der Aufgabe } { Zahlen einlesen und Liste aufbauen: } Zahlenliste
(Liste) ;
{ Werte auf den Stapel legen: } ZgListe := Liste;
while ZgListe <> nil do begin
Stapel.push (ZgListe^.info);
ZgListe := ZgListe^.next end;
{ Zahlen in invertierter Reihenfolge ausgeben: } writeln ('Die invertierte Liste der Zahlen ist: ');
while not Stapel.isEmpty do begin
Stapel.top (Zahl);
Stapel.pop;
write (Zahl) end end. { Zahleninversion }
```