Kurs 1613 “Einführung in die imperative Programmierung”

Musterlösung zur Klausur am 01.03.2003

Aufgabe 1

a)

```
procedure InitBoolFeld(inWert: boolean; var ioFeld: tBoolFeld);
{ Initialisiert ioFeld mit inWert }
var i: tBereich;
begin
  for i := 1 to GRAD do
    ioFeld[i] := inWert
end;
```

b)

```
function TesteBoolFeld(inFeld: tBoolFeld):boolean;
{Liefert true, wenn alle Elemente von inFeld true sind }
var i: integer;
    hilf: boolean;
begin
   i:=0;
   hilf := true;
   while hilf and (i<GRAD) do
  {Die Schleife endet, wenn ein false-Wert gefunden oder das
    Ende des Felds erreicht wird }
   begin
      i:=i+1;
      hilf := inFeld[i]
   end;
   TesteBoolFeld := hilf
end;
```

c)

```
function IstHalb(inMatrix: tMatrix):boolean;
{Testet, ob inMatrix halblateinisch ist }
var boolFeldZ, boolFeldS : tBoolFeld;
    i,j: integer;
    hilf : boolean;
begin
  hilf := true;
  i:=1;
  while hilf and (i<=GRAD) do
  {Die Schleife wird beendet, wenn eine Zeile gefunden wird,
    die nicht der halblatein-Bedingung genuegt oder alle Zeilen
  untersucht worden sind }
```

```
  begin
     InitBoolFeld(false, boolFeldZ);
     for j:= 1 to GRAD do
        boolFeldZ[inMatrix[i,j]] := true;
     {Alle in der aktuellen Zeile vorkommenden Werte wurde in
      dem boolschen Feld gespeichert }
     hilf := TesteBoolFeld(boolFeldZ);
      {Es wurde getestet, ob die i.Zeile die halblatein
      Bedingung erfüllt }
     i:=i+1;
  end;
  IstHalb := hilf;
end;
```

Aufgabe 2

```
procedure Tausche(var ioRefListe: tRefListe);
{s. Aufgabenstellung}

var such,
    vor,
    nach : tRefListe;

begin
  vor := NIL;
  such := ioRefListe;
  while such^.info <> 0 do
  begin
    vor := such;
    such := such^.next
  end;
  nach := such^.next^.next;
  {Jetzt sind alle Zeiger positioniert. such zeigt auf das
  Element mit dem Wert 0, vor auf dessen Vorgaenger oder auf nil,
  nach auf den Nachfolger von such^.next}

  if vor=NIL then
  {D.h. es muss am Anfang getauscht werden }
    ioRefListe := such^.next
  else
    vor^.next := such^.next;
```

```
  such^.next^.next := such;
{such^.next ist ungleich nil, da nach Voraussetzung 0 in der
Liste nicht als letztes Element vorkommt }
  such^.next := nach
end;
```

Aufgabe 3

.

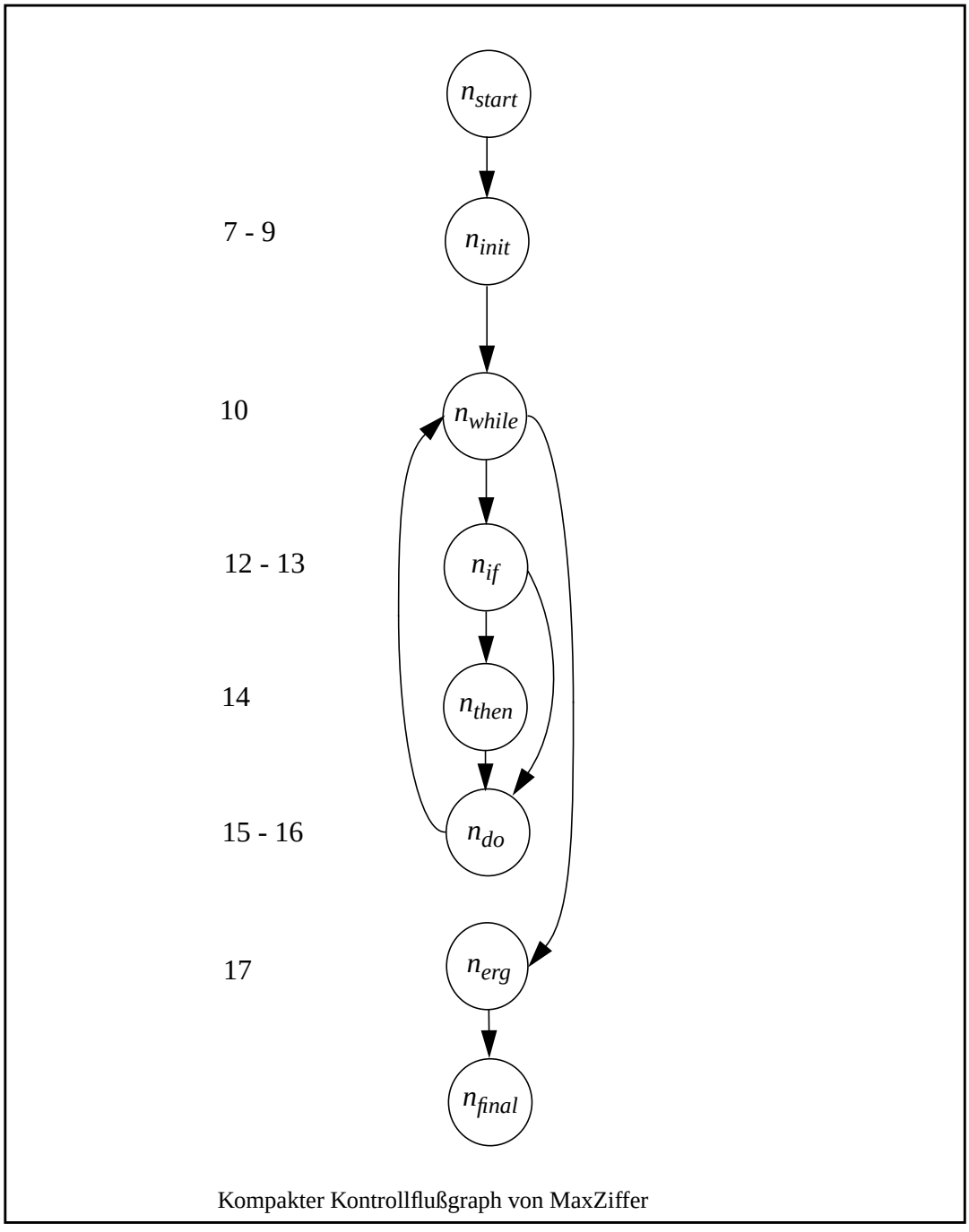
Platzhalter	Anweisung
(1)	ioRefAnfang := elem;
(2)	elem^.next := NIL;
(3)	elem^.next := ioRefAnfang;
(4)	ioRefAnfang := elem;
(5)	ende := true;
(6)	lauf := lauf^.next;
(7)	elem^.next :=lauf^.next
(8)	lauf^.next := elem;

Aufgabe 4

```
procedure HRZwei(inRefWurzel:tRefBinBaum; var ioDrucken:boole-
an);
{Gibt jeden zweiten Knotenwert der Hauptreihenfolge aus }
begin
  if inRefWurzel<>NIL then
  { Nur wenn der Baum nicht leer ist... }
  begin
    if ioDrucken then
    { Muss der Wert ausgegeben werden? }
      writeln(inRefWurzel^.info);
    ioDrucken := not ioDrucken;
    HRZwei(inRefWurzel^.links, ioDrucken);
    HRZwei(inRefWurzel^.rechts, ioDrucken)
  end;
end;
```

Aufgabe 5

a)



Kompakter Kontrollflußgraph von MaxZiffer

b) Folgender Pfad **umgeht** die Schleife: (n_{start} , n_{init} , ,n_{while} , n_{erg} , n_{final}). Das einzig mögliche Testdatum hierfür ist (0,0). Die tatsächliche Ausgabe ist 2. Der Fehler wird gefunden.

Der Pfad (n_{start} , n_{init} , n_{while} , n_{if} , n_{then} ,n_{do} ,n_{while} , n_{erg} , n_{final}) durchläuft die while-Schleife **einmal**. Ein mögliches Testdatum ist (5,5). Der Fehler wird nicht gefunden.

Folgender Pfad gehört zu einem **zweimaligem** Durchlauf:
(n_{start} , n_{init} , n_{while} , n_{if} , n_{then} ,n_{do} ,n_{while} ,n_{if} , n_{then} ,n_{do} ,n_{while} ,n_{erg} , n_{final})
Ein mögliches Testdatum ist:
(43,4)
Der Fehler wird nicht gefunden, denn 4 ist auch das tatsächliche Ergebnis.

c) Der Fehler wird beim boundary interior- Test auf jeden Fall gefunden, da es nur ein Testdatum gibt, mit welchem die Schleife umgangen wird. Dieses Testdatum führt wie oben gezeigt zum Auffinden des Fehlers.

Damit das Programm fehlerfrei funktioniert, muß die Zeile 9 lauten `maxi := 0;`