Kurs 1613 “Einführung in die imperative Programmierung”

Musterlösung zur Klausur am 06.03.2004

Aufgabe 1

Wir erstellen die Rundreise, indem wir in der Variablen `hier` die aktuelle Stadt, von der aus weitergefahren wird, festhalten. Die nächste Stadt wird anhand der Entfernungen in der durch `hier` indizierten Matrixzeile ausgesucht, wobei mittels der Einträge im Feld `besucht` die noch nicht besuchten Städte ausgewählt werden. Auch hier geben wir wieder ein Rahmenprogramm an, welches den Test der Prozedur `RundReise` ermöglicht:

```
program Aufgabe (input, output);
{ Testprogramm fuer die Prozedur Rundreise }

  const
  MAXORTSZAHL = 10;

  type
  tOrtsIndex = 1..MAXORTSZAHL;
  tEntfernung = 0..maxint;
  tEntfMatrix = array [tOrtsIndex, tOrtsIndex] of tEntfernung;

  var
  EntfMatrix : tEntfMatrix;
  i,
  j : tOrtsIndex;

  procedure Rundreise (var inMat : tEntfMatrix);
  { berechnet eine kurze Rundreise aus der Entfernungsmatrix
    inMat }

    type
    tBoolFeld = array [tOrtsIndex] of boolean;

    var
    besucht : tBoolFeld;
    i,
    hier,
    Spalte,
    naechste : tOrtsIndex;
    Gesamtstrecke,
    Einzelstrecke : tEntfernung;

  begin
    { initialisiere das Feld besuchter Staedte }
    for i := 1 to MAXORTSZAHL do
      besucht [i] := false;
    hier := 1;
    besucht [hier] := true;
```

```
    Gesamtstrecke := 0;
    writeln ('Rundreise: ');
    writeln (hier);
    for i := 1 to MAXORTSZAHL - 1 do
    begin
      Einzelstrecke := maxint;
      { suche die naechstgelegene, noch nicht besuchte Stadt }
      for Spalte := 1 to MAXORTSZAHL do
        if not besucht [Spalte] then
          if inMat [hier, Spalte] < Einzelstrecke then
          begin
            Einzelstrecke := inMat [hier, Spalte];
            naechste := Spalte
          end;
      writeln (naechste);
      Gesamtstrecke := Gesamtstrecke + Einzelstrecke;
      besucht [naechste] := true;
      hier := naechste
    end;
    writeln ('1');
    Gesamtstrecke := Gesamtstrecke + inMat [hier, 1];
    writeln (Gesamtstrecke, ' km')
  end; { Rundreise }

begin
  for i := 1 to MAXORTSZAHL do
    for j := i to MAXORTSZAHL do
    begin
    { Einlesen einer symmetrischen Entfernungsmatrix: }
      write ('Strecke von ', i, 'nach ', j, ': ');
      readln (EntfMatrix [i, j]);
      EntfMatrix [j, i] := EntfMatrix [i, j];
    end;
  Rundreise (EntfMatrix)
end. { Aufgabe }
```

Aufgabe 2

```
procedure Tausche(var ioRefListe: tRefListe);

  var
  vor,ende,
  hilf : tRefListe;
```

```
  begin
    vor := ioRefListe;
    {Vorgänger von gesuchtem Element finden}
    while vor^.next^.info <> 0 do
      vor := vor^.next;
    {zu verschiebendes Element merken und abhängen}
    hilf := vor^.next;
    vor^.next := hilf^.next;
    {letztes Element finden}
    ende := vor;
    while ende^.next <> nil do
      ende := ende^.next;
    {gemerktes Element hinten anhängen}
    ende^.next := hilf;
    hilf^.next := nil
  end; {Tausche}
```

Aufgabe 3

```
procedure DreifachPosLoeschen (inRefAnfang : tRefListe);
  { loescht das dritte, sechste, neunte usw. Element aus der
    Liste, auf deren Anfang inRefAnfang zeigt }

    var
    RefElement,
    RefLoesch : tRefListe;

  begin
    RefElement := inRefAnfang;
    while RefElement <> nil do
    { das Ende der Liste ist noch nicht erreicht }
    begin
      { Vorgaenger eines zu loeschenden Elementes suchen }
      RefElement := RefElement^.next;
      if RefElement <> nil then
      { Vorgaenger eines zu loeschenden Elementes gefunden }
      begin
        RefLoesch := RefElement^.next;
        if RefLoesch <> nil then
        { ein Element muss geloescht werden }
        begin
          RefElement^.next := RefLoesch^.next;
          dispose(RefLoesch)
```

```
      end;
      RefElement := RefElement^.next
    end
  end
end; {DreifachPosLoeschen }
```

Aufgabe 4

a)

```
function Vorkommen (
           inRefWurzel : tRefBinBaum;
           inSuchwert : integer) : tNatZahl;
{ ermittelt die Anzahl aller Knoten des Baumes, auf dessen
  Wurzel inRefWurzel zeigt, bei denen der Wert der
  info-Komponenten mit dem Wert von inSuchwert
  uebereinstimmt }

  var
  Anzahl : tNatZahl;

begin
  if inRefWurzel = nil then
    Anzahl := 0
  else
  begin
    Anzahl := Vorkommen (inRefWurzel^.rechts, inSuchwert)
            + Vorkommen (inRefWurzel^.links, inSuchwert);
    if inRefWurzel^.info >= inSuchwert then
      Anzahl := Anzahl + 1
  end; { if inRefWurzel = nil }
  Vorkommen := Anzahl
end; { Vorkommen }
```

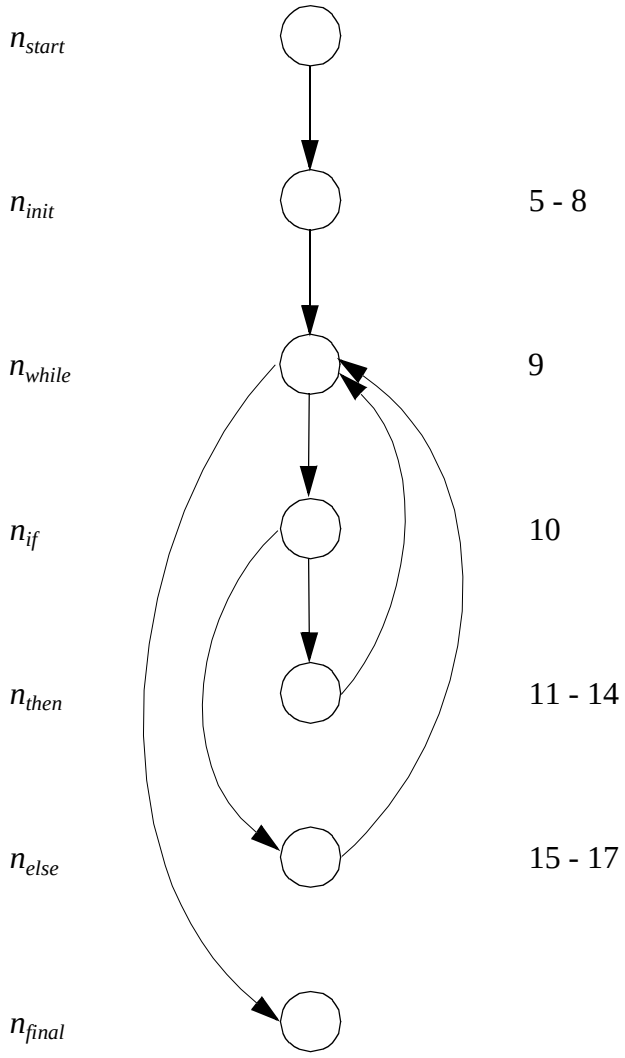
b) Ja. Ein iterativer Algorithmus benötigt eine Hilfsdatenstruktur zur Realisierung des Baumdurchlaufs.

Aufgabe 5

a) Der Aufbau des kompakten Kontrollflußgraphen der Funktion `Primfaktoren` kann der folgenden Graphik entnommen werden.

b) Für einen 'boundary interior'-Test müssen sieben Pfade überprüft werden.
Mit dem Pfad

$(n_{start}, n_{init}, n_{while}, n_{final})$



wird die Schleife umgangen. Es gibt nur ein mögliches Testdatum zu diesem Pfad:

(1, "Ergebnis: ")

Innerhalb des boundary-Tests gibt es zwei Pfade, bei denen die Schleife nur einmal durchlaufen wird:

(n_{start} , n_{init} , n_{while} , n_{if} , n_{then} , n_{while} , n_{final})

mit dem einzig möglichen Testdatum (2, "Ergebnis: 2 ") und

(n_{start} , n_{init} , n_{while} , n_{if} , n_{else} , n_{while} , n_{final})

für den es kein Testdatum gibt, da der **`else`**-Zweig keinen Einfluß auf das Abbruchkriterium hat.

Beim interior-Test müssen vier Pfade berücksichtigt werden. Zu den ersten beiden Pfaden

---

$(n_{start}, n_{init}, n_{while}, n_{if}, n_{then}, n_{while}, n_{if}, n_{else}, n_{while}, n_{final})$
$(n_{start}, n_{init}, n_{while}, n_{if}, n_{else}, n_{while}, n_{if}, n_{else}, n_{while}, n_{final})$

gibt es kein Testdatum, da der **`else`**-Zweig keinen Einfluß auf das Abbruchkriterium hat und der Schleifenrumpf nach durchlaufen des **`else`**-Zweiges mindestens noch einmal durchlaufen werden muß. Die beiden verbliebenen Pfade sind ausführbar, wobei es in beiden Fällen jeweils genau ein mögliches Testdatum gibt:

$(n_{start}, n_{init}, n_{while}, n_{if}, n_{then}, n_{while}, n_{if}, n_{then}, n_{while}, n_{final})$

mit dem Testdatum (4, "Ergebnis: 2 2 ") und

$(n_{start}, n_{init}, n_{while}, n_{if}, n_{else}, n_{while}, n_{if}, n_{then}, n_{while}, n_{final})$

mit dem Testdatum (3, "Ergebnis: 3 ").