# Kurs 01578 „PASCAL-Programmierkurs für Informatiker“

## Klausur am 12.02.2000
## Lösungshinweise

## Aufgabe 1

(a)

```
type
tFarbmenge = set of tFarbe;
```

(b)

```
const
MAXZEILE  = 30;
MAXSPALTE = 20;

type
tZeile  = 1..MAXZEILE;
tSpalte = 1..MAXSPALTE;
tMosaik = array [tZeile, tSpalte] of tFarbmenge;
```

(c)

```
function schimmert ( var inMosaik : tMosaik ) : boolean;
{ Ermittelt, ob es eine Farbe gibt, in der jede Fliese von inMosaik
  schimmert. }
{ Version 1.0 von 19.12.99 UBec,Lg }

  var
  Zeile        : tZeile;
  Spalte       : tSpalte;
  MdMoeglichen : tFarbmenge;          { Menge der moeglichen Farben }

begin

  { Als Farben, in denen das gesamte Mosaik schimmert, kommen nur die
    Farben in Frage, in denen insbesondere die Fliesen am oberen und die am
    rechten Rand schimmern. Zu diesen Fliesen gibt es keine Nachbarfliese
    rechts oben, sie schimmern also nur in ihren eigenen Farben. }
  MdMoeglichen := inMosaik [MAXZEILE, MAXSPALTE];
  for Spalte := 1 to MAXSPALTE - 1 do
    MdMoeglichen := MdMoeglichen * inMosaik [MAXZEILE, Spalte];
  for Zeile := 1 to MAXZEILE - 1 do
    MdMoeglichen := MdMoeglichen * inMosaik [Zeile, MAXSPALTE];
```

```
  { In diesen Farben schimmern zwangslaeufig auch die Fliesen in der
    zweiten Zeile von oben und die in der zweiten Spalte von rechts.
    Bleiben die uebrigen Fliesen zu pruefen. }
  for Zeile := 1 to MAXZEILE - 2 do
    for Spalte := 1 to MAXSPALTE - 2 do
        MdMoeglichen := MdMoeglichen * (inMosaik [Zeile, Spalte] +
                          inMosaik [Zeile + 1, Spalte + 1]);

  schimmert := (MdMoeglichen <> [ ])

end; { schimmert }
```

(d)

```
procedure bestimmeFarben
( var inMosaik : array [conMinZeile..conMaxZeile : integer;
                        conMinSpalte..conMaxSpalte : integer] of tFarbmenge;
  var outFarbmenge : tFarbmenge                                          );
{ Ermittelt die Menge der Farben, die in inMosaik vorkommen. }
{ Version 1.0 von 09.12.99 UBec,Lg }

  var
  Zeile,
  Spalte    : integer;
  Farbmenge : tFarbmenge;

begin
  Farbmenge := [ ];
  for Zeile := conMinZeile to conMaxZeile do
    for Spalte := conMinSpalte to conMaxSpalte do
      Farbmenge := Farbmenge + inMosaik [Zeile, Spalte];
  outFarbmenge := Farbmenge
end; { bestimmeFarbmenge }
```

(e)

```
type
tFarbe = (Rot, Gruen, Gelb, Blau, Violett, Weiss, Grau, Schwarz);
```

Die Menge aller Farben im Wertebereich von `tFarbe` ist `[Rot..Schwarz]`.

## Aufgabe 2

(a)

```
type
tRefKnoten = ^tKnoten;
tKnoten    = record
               Zahl       : integer;
               RefLinker,
               RefRechter : tRefKnoten
             end;
```

(b)

```
procedure loescheBinBaum
( inRefWurzel : tRefKnoten);
{ Loescht in dem binaeren Baum, auf dessen Wurzel inRefWurzel zeigt, alle
  Knoten. }
{ Version 1.0 vom 13.12.99 UBec,Lg }

begin
  if inRefWurzel <> nil then
  begin
    loescheBinBaum (inRefWurzel^.RefLinker);
    loescheBinBaum (inRefWurzel^.RefRechter);
    dispose (inRefWurzel)
  end { if inRefWurzel }
end; { loescheBinBaum }
```

(c)

```
procedure loescheUnverzweigte
( var ioRefWurzel : tRefKnoten );
{ Loescht in dem binaeren Baum, auf dessen Wurzel ioRefWurzel zeigt, alle
  Knoten, die genau einen Sohn haben. }
{ Version 1.0 vom 13.12.99 UBec,Lg }

  var
  RefZuLoeschender : tRefKnoten;

begin
  if ioRefWurzel <> nil then
    with ioRefWurzel^ do
    begin
      loescheUnverzweigte (RefLinker);
      loescheUnverzweigte (RefRechter);
      if (RefLinker = nil) and (RefRechter <> nil) then
      begin
        RefZuLoeschender := ioRefWurzel;
        ioRefWurzel := RefRechter;
        dispose (RefZuLoeschender)
      end; { if (RefLinker = nil)... }
      if (RefLinker <> nil) and (RefRechter = nil) then
      begin
        RefZuLoeschender := ioRefWurzel;
        ioRefWurzel := RefLinker;
        dispose (RefZuLoeschender)
      end { if (RefLinker <> nil)... }
    end { with ioRefWurzel^ }
  end; { loescheUnverzweigte }
```

## Aufgabe 3

```
type
tNatZahlNull = 0..maxint;
tRefZahliL   = ^tZahliL;
tZahliL      = record                              { Zahl in Liste }
                 Zahl        : tNatZahlNull;
                 RefNaechste : tRefZahliL
               end;

procedure ermittleLaengste
(     inRefAnfang          : tRefZahliL;
  var outRefAnfangLaengste : tRefZahliL;
  var outLaenge            : tNatZahlNull );
{ Ermittelt Anfang und Laenge der laengsten Sequenz in der Liste, auf deren
  Anfang inRefAnfang zeigt. }
{ Version 1.0 vom 16.12.99 UBec,Lg }

  var
  RefAnfang,
  RefAnfangNeu,
  Zeiger       : tRefZahliL;
  Laenge,
  LaengeNeu    : tNatZahlNull;
  zuEnde       : boolean;

begin
  { Bisher ist die leere Sequenz die laengste. }
  RefAnfang := nil;
  Laenge := 0;
  { Versuche, am Listenanfang eine neue Sequenz zu beginnen. }
  RefAnfangNeu := inRefAnfang;
  LaengeNeu := 0;
  Zeiger := inRefAnfang;
  while Zeiger <> nil do
  begin
    { Nehme Zeiger^ zur neuen Sequenz. }
    LaengeNeu := LaengeNeu + 1;
    { Pruefe, ob auch die naechste Zahl noch zur Sequenz gehoeren kann. }
    if Zeiger^.RefNaechste = nil then
      zuEnde := true
    else
      zuEnde := (Zeiger^.RefNaechste^.Zahl < Zeiger^.Zahl);
    if zuEnde then
    begin
      { Eine neue, nicht verlaengerbare Sequenz ist gefunden. Wenn sie
        laenger ist als die bisher laengste, uebernehme sie. }
      if LaengeNeu > Laenge then
      begin
        RefAnfang := RefAnfangNeu;
        Laenge := LaengeNeu
      end; { if LaengeNeu > Laenge }
      { Eine weitere nicht verlaengerbare Sequenz kann fruehestens mit der
        naechsten Zahl in der Liste beginnen. }
      RefAnfangNeu := Zeiger^.RefNaechste;
      LaengeNeu := 0;
    end; { if zuEnde }
  Zeiger := Zeiger^.RefNaechste
  end; { if zuEnde }
  outRefAnfangLaengste := RefAnfang;
  outLaenge := Laenge
end; { ermittleLaengste }
```

# Aufgabe 4

```
program Einsendungen (output, Einsendedatei);
{ Ermittelt aus der Einsendedatei zu allen Kursen die erfolgreichen
  Teilnehmer und gibt deren Matrikelnummer zusammen mit der von ihnen
  erreichten Punktsumme aus. }
{ Version 1.0 vom 19.12.99 UBec,Lg }

  const
  MAXMATRIKELNR  = 30000;
  MINKURSNR      = 1500;
  MAXKURSNR      = 1999;
  MAXKURSEINHEIT = 7;
  MAXPUNKTE      = 100;
  MINERFOLG      = 200;    { Mindestpunktzahl fuer Erfolg }
  MAXPUNKTSUMME  = 700;    { MAXPUNKTE * MAXKURSEINHEIT }

  type
  tMatrikelnr     = 1..MAXMATRIKELNR;
  tKursnr         = MINKURSNR..MAXKURSNR;
  tKurseinheit    = 1..MAXKURSEINHEIT;
  tPunkte         = 0..MAXPUNKTE;
  tEinsendung     = record
                      Matrikelnr  : tMatrikelnr;
                      Kursnr      : tKursnr;
                      Kurseinheit : tKurseinheit;
                      Punkte      : tPunkte
                    end;
  tEinsendedatei  = file of tEinsendung;
  tGesamtpunkte   = array [tKurseinheit] of tPunkte;
  tRefErgebnis    = ^tErgebnis;
  tErgebnis       = record
                      Matrikelnr   : tMatrikelnr;
                      Gesamtpunkte : tGesamtpunkte;
                      RefNaechster : tRefErgebnis
                    end;
  tGesamtergebnis = array [tKursnr] of tRefErgebnis;
  tPunktsumme     = 0..MAXPUNKTSUMME;

  var
  Einsendedatei  : tEinsendedatei;
  Gesamtergebnis : tGesamtergebnis;

  procedure verbucheEinsendung
  (     inEinsendung     : tEinsendung;
    var ioGesamtergebnis : tGesamtergebnis );
  { Traegt die Punktzahl von inEinsendung in das entsprechende
    Listenelement von ioGesamtergebnis ein. Eine dort bereits vorhandene
    Angabe wird ggf. ueberschrieben. Ist ein entsprechendes Listenelement
    noch nicht vorhanden, wird es ergaenzt. }
  { Version 1.0 vom 13.12.99 UBec,Lg }

    var
    RefVorgaenger,
    RefNachfolger,
    RefEinsender  : tRefErgebnis;
    fertig,
    vorhanden     : boolean;
    Kurseinheit   : tKurseinheit;
```

```
begin { verbucheEinsendung }

  { Suche das erste Element der zum Kurs von inEinsendung gehoerenden
    Liste, das eine hoehere Matrikelnummer hat als inEinsendung. Dies
    wird - falls vorhanden - der Nachfolger des Elements, in dem die
    Einsendung verbucht wird. }
  RefVorgaenger := nil;
  RefNachfolger := ioGesamtergebnis [inEinsendung.Kursnr];
  fertig := (RefNachfolger = nil);
  while not fertig do
  begin
    fertig := (RefNachfolger^.Matrikelnr > inEinsendung.Matrikelnr);
    if not fertig then
    begin
      RefVorgaenger := RefNachfolger;
      RefNachfolger := RefNachfolger^.RefNaechster;
      fertig := (RefNachfolger = nil)
    end  { if not fertig }
  end; { while not fertig }

  { Wenn es vor dem gefundenen Nachfolger gar kein Element oder nur
    Elemente mit kleinerer Matrikelnummer als in der Einsendung gab, muss
    ein neues Element eingefuegt werden. Andernfalls zeigt RefVorgaenger
    auf das Element mit der Matrikelnummer. }
  if RefVorgaenger = nil then
    vorhanden := false
  else
    vorhanden := (RefVorgaenger^.Matrikelnr = inEinsendung.Matrikelnr);
  if vorhanden then
    RefEinsender := RefVorgaenger
  else
  begin
    new (RefEinsender);
    with RefEinsender^ do
    begin
      Matrikelnr := inEinsendung.Matrikelnr;
      for Kurseinheit := 1 to MAXKURSEINHEIT do
        Gesamtpunkte [Kurseinheit] := 0;
      RefNaechster := RefNachfolger
    end; { with RefEinsender^ }
    if RefVorgaenger = nil then
      ioGesamtergebnis [inEinsendung.Kursnr] := RefEinsender
    else
      RefVorgaenger^.RefNaechster := RefEinsender
  end; { if vorhanden }

  { Trage die Punktzahl ein. }
  RefEinsender^.Gesamtpunkte [inEinsendung.Kurseinheit] :=
    inEinsendung.Punkte

end; { verbucheEinsendung }

procedure verbucheAlleEinsendungen
( var ioEinsendedatei    : tEinsendedatei;
  var outGesamtergebnis : tGesamtergebnis );
{ Legt outGesamtergebnis an und verbucht dort alle Einsendungen aus
  ioEinsendedatei. }
{ Version 1.0 vom 13.12.99 UBec,Lg }

  var
  Kursnr : tKursnr;
```

```
  begin { verbucheAlleEinsendungen }

    { Initialisiere das Gesamtergebnis mit leeren Listen. }
    for Kursnr := MINKURSNR to MAXKURSNR do
      outGesamtergebnis [Kursnr] := nil;

    { Verbuche alle Einsendungen aus der Eingabedatei im Gesamtergebnis. }
    reset (ioEinsendedatei);
    while not eof (ioEinsendedatei) do
    begin
      verbucheEinsendung (ioEinsendedatei^, outGesamtergebnis);
      get (ioEinsendedatei)
    end { while not eof (ioEinsendedatei) }

  end; { verbucheAlleEinsendungen }

  procedure gibErfolgreicheAus
  ( var inGesamtergebnis : tGesamtergebnis );
  { Gibt zu allen Kursen mit Einsendungen die Matrikelnummern der
    erfolgreichen Teilnehmer zusammen mit der von ihnen erreichten
    Punktsumme aus. }
  { Version 1.0 vom 19.12.99 UBec,Lg }

    var
    Kursnr       : tKursnr;
    RefEinsender : tRefErgebnis;
    Punktsumme   : tPunktsumme;
    Kurseinheit  : tKurseinheit;

  begin { gibErfolgreicheAus }

    page;
    writeln ('Erfolgreich waren');
    writeln;
    for Kursnr := MINKURSNR to MAXKURSNR do
      if inGesamtergebnis [Kursnr] <> nil then
      begin
        writeln ('zu Kurs ', Kursnr:4);
        writeln ('Matrikelnr. Punkte');
        RefEinsender := inGesamtergebnis [Kursnr];
        repeat
          with RefEinsender^ do
          begin
            Punktsumme := 0;
            for Kurseinheit := 1 to MAXKURSEINHEIT do
              Punktsumme := Punktsumme + Gesamtpunkte [Kurseinheit];
            if Punktsumme >= MINERFOLG then
              writeln (Matrikelnr:8, Punktsumme:8)
          end; { with RefEinsender^ }
          RefEinsender := RefEinsender^.RefNaechster
        until RefEinsender = nil;
        writeln
      end { if Gesamtergebnis [Kursnr] <> nil }

  end; { gibErfolgreicheAus }

begin { Einsendungen }
  verbucheAlleEinsendungen (Einsendedatei, Gesamtergebnis);
  gibErfolgreicheAus (Gesamtergebnis)
end. { Einsendungen }
```

Bei Benutzung von `read` in der Prozedur `verbucheAlleEinsendungen` ist die Variablendeklariation `Einsendung : tEinsendung;` zu ergänzen und der Rumpf der **`while`**-Schleife zu ersetzen durch

```
read (ioEinsendedatei, Einsendung);
verbucheEinsendung (Einsendung, outGesamtergebnis)
```