Kurs 01578 „PASCAL-Programmierkurs für Informatiker“

Klausur am 13.02.1999
Lösungshinweise

A u f g a b e 1

```
program Aufruecken (input, outout);
{ Liest von der Standardeingabe Zahlen im Wertebereich von integer ein,
  eine pro Zeile, und gibt sie ueber die Standardausgabe in Zeilen mit
  maximal 72 Zeichen in unveraenderter Reihenfolge wieder aus, zwei Zahlen
  in der Zeile durch ein Leerzeichen getrennt, keine Zahl ueber mehrere
  Zeilen gebrochen. }
{ Version 1.0 vom 17.12.98 UBec,Lg }

  const
  MAXLAENGE = 72; { 10 hoch (MAXLAENGE - 1) > maxint }

  type
  tLaenge    = 0..MAXLAENGE;
  tRefZahliL = ^tZahliL;
  tZahliL    = record                          { Zahl in Liste }
                 Zahl        : integer;
                 RefNaechste: tRefZahliL
               end;

  var
  Zeilenlaenge,
  Zeichenzahl      : tLaenge;
  Zahl             : integer;
  RefAnfang,
  RefEnde,
  Zeiger,
  RefZuLoeschende: tRefZahliL;

  grocedure haengeAn
  (     inZahl      : integer;
    var ioRefAnfang,
        ioRefEnde   : tRefZahliL );
  { Haengt inZahl an die Liste an, auf deren Anfang und Ende ioRefAnfang
    bzw. ioRefEnde zeigen. }
  { Version 1.0 vom 16.10.98 Ubec,Lg }

    var
    RefNeue : tRefZahliL;

  begin
    new (RefNeue);
    RefNeue^.Zahl := inZahl;
    RefNeue^.RefNaechste := nil;
    if ioRefAnfang = nil then
      ioRefAnfang := RefNeue
    else
      ioRefEnde^.RefNaechste := RefNeue;
    ioRefEnde := RefNeue
```

```
  function Ziffernzahl ( inZahl: integer ): tLaenge;
  { Ermittelt die Anzahl der Dezimalstellen von inZahl. }
  { Version 1.0 vom 17.12.98 UBec,Lg }

    var
    Anzahl : tLaenge;
    Zahl   : integer;

  begin
    Zahl  := inZahl;
    Anzahl := 0;
    repeat
      Anzahl  := Anzahl + 1;
      Zahl  := Zahl div 10'
    until Zahl = 0;
    Ziffernzahl := Anzahl
  end; { Ziffernzahl }

begin { Aufruecken }

  { Eingabeaufforderung siehe Aufgabenstellung. }

  { Lege eine Liste fuer die Zwischenspeicherung an. }
  RefAnfang := nil;
  RefEnde := nil;

  { Lese die Zahlen ein. }
  while not eof do
  begin
    readln (Zahl);
    haengeAn (Zahl, RefAnfang, RefEnde)
  end; { while not eof }

  { Gib die Zahlen aus. }
  writeln;
  Zeilenlaenge := 0;
  Zeiger  := RefAnfang;
  while Zeiger <> nil do
  begin
    Zahl  : =   Zeiger^.Zahl;
    Zeichenzahl  := Ziffernzahl (Zahl);
    { Beruecksichtige ggf.  ein negatives Vorzeichen. }
    if Zahl < 0 then
      Zeichenzahl  := Zeichenzahl + 1;
    { Steht bereits etwas in der Zeile, so trenne die Zahl davon. }
    if Zeilenlaenge > 0 then
    begin
      { Wenn der Platz in der Zeile fuer die Zahl und ein trennendes
        Leerzeichen nicht mehr ausreicht... }
      if Zeichenzahl + 1 > MAXLAENGE - Zeilenlaenge then
      { ...so beginne eine neue Zeile. }
      begin
        writeln;
        Zeilenlaenge  := 0
      end { if Zeichenzahl... then }
      else
      { Andernfalls schreibe ein trennendes Leerzeichen. }
      begin
        write  (' ');
        Zeilenlaenge  := Zeilenlaenge + 1
      end { if Zeichenzahl... else }
    end; { if Zeilenlaenge > 0 }
```

```
    write (Zahl:Zeichenzahl);
    Zeilenlaenge  := Zeilenlaenge + Zeichenzahl;
    Zeiger  := Zeiger^.RefNaechste
  end; { while Zeiger <> nil }
  writeln:

  { Loesche die Liste. }
  Zeiger  := RefAnfang;
  while Zeiger <> ail do
  begin
    RefZuLoeschende := Zeiger;
    Zeiger := Zeiger^.RefNaechste;
    dispose (RefZuLoeschende)
  end { while Zeiger <> nil }

end. { Aufruecken }
```

Aufgabe 2

```
function sindZwillinge ( inErste, inZweite: integer ): boolean;
{ Ermittelt, ob inErste und inZweite Zwillinge sind. }
{ Version 1.0 vom 22.12.98 UBec,Lg }

begin
  if inErste = inZweite then
    sindZwillinge := false
  else
    if inErste < inZweite then
      sindZwillinge := (inErste + 1 = inZweite)
    else
      { inErste > inZweite }
      sindZwillinge := (inErste - 1 = inZweite)
end; { sindZwillinge }
```

Aufgabe 3

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

function mitZwillingen( inRefWurzel: tRefKnoten): boolean;
{ Ermittelt, ob in dem binaeren Baum, auf dessen Wurzel inRefWurzel zeigt,
  ein Knoten vorkommt, der Zwillinge zu.Soehnen hat. }
{ Version 1.0 vom 22.12.98 Ubec,Lg }

  var
  gefunden : boolean;

begin
  gefunden := false;
  if inRefWurzel <> nil then
    with inRefWurzel^ do
    begin
      if (RefLinker <> nil) and (RefRechter <> nil) then
        gefunden := sindZwillinge (RefRechter^.Zahl, RefLinker^.Zahl);
      if not gefunden then
        gefunden := mitZwillingen (RefLinker);
      if not gefunden then
        gefunden := mitZwillingen (RefRechter)
    end; { with }
  mitZwillingen := gefunden
end; { mitZwillingen }
```

Aufgabe 4

```
type
tRefZahliL = ^tZahliL;
tZahliL    = record                                { Zahl in Liste )
               Zahl        : integer;
               RefNaechste : tRefZahliL
             end;

proceduro ergaenzeZwillinge
( var ioRefAnfang: tRefZahliL );
{ Ergaenzt in der Liste, auf deren Anfang ioRefAnfang zeigt, vor jeder Zahl
  einen Zwilling von ihr, sofern dort nicht schon einer steht und es einen
  gibt, der dort stehen darf.
  Vor- und Nachbedingung: Die Zahlen in der Liste sind aufsteigend geordnet
  und paarweise verschieden. )

{ Version 1.0 vom 17.12.98 UBec,Lg )

  var
  Zeiger,
  RefNeue : tRefZahliL;

begin

  if ioRefAnfang <> nil then
  begin

    { Fuege, wenn moeglich, vor der ersten Zahl einen Zwilling von ihr ein.
      Wegen der Anordnung der Zahlen in der Liste kommt nur ein kleinerer
      Zwilling in Betracht. )
    Zeiger := ioRefAnfang;
    if Zeiger^.Zahl > -maxint then
    begin
      new (ioRefAnfang);
      ioRefAnfang^.Zahl := Zeiger^.Zahl - 1;
      ioRefAnfang^.RefNaechste := Zeiger;
    end; { if Zeiger^.Zahl > -maxint }

    { Weitere Zahlen in der Liste sind groesser als -maxint.
      Gehe sie durch.' }
    while Zeiger^.RefNaechste <> nil do
      { Steht bereits ein Zwilling vor ihr... )
      if Zeiger^.RefNaechste". Zahl - 1 = Zeiger^.Zahl then
        { ...so gehe weiter. }
        Zeiger := Zeiger^.RefNaechste
      else
      { Andernfalls fuege einen ein... )
      begin
        new (RefNeue);
        RefNeue^.Zahl := Zeiger^.RefNaechste^.Zahl - 1;
        RefNeue^ .RefNaechste := Zeiger^.RefNaechste;
        Zeiger^.RefNaechste := RefNeue;
      . { . . .und gehe weiter. )
        Zeiger := RefNeue" .RefNaechste
      end { if Zeiger^.RefNaechste^.Zahl - 1 = Zeiger^.Zahl else )

  end { if ioRefAnfang <> nil )

end; { ergaenzeZwillinge }
```

Aufgabe 5

```
function zehnprim ( inZahl: integer ): boolean;
{ Bestimmt, ob inZahl 10-prim ist. }
{ Version 1.0 vom 09.10.98 UBec,Lg }

  var
  vielleicht : boolean;
  Zahl       : integer;

begin
  vielleicht := (inZahl >= 10);
  Zahl  := 10;
  while vielleicht and (Zahl < inZahl) do
  begin
    if zehnprim (Zahl) then
      vielleicht := (inZahl mod Zahl <> 0);
    Zahl  := succ (Zahl)
  end; { while }
  zehnprim  := vielleicht
end; { zehnprim }
```

Aufgabe 6

```
const
MAXINDEX = 20;

type
tFarbe  = (blau, gelb, gruen, rot, schwarz, weiss);
tIndex  = 1..MAXINDEX;
tMosaik = array [tIndex, tIndex] of tFarbe;

function kommtVor( inMosaik: tMosaik; inMaxZeile, inMaxSpalte: tIndex )
  : boolean;

{ Prueft, ob in inMosaik eine Farbe in jeder Spalte und in jeder Zeile
  vorkommt. }
{ Version 1.0 vom 11.12.98 UBec,Lg }

  type
  tFarbenmenge = set of tFarbe;

  var
  MdMoeglichen,                    { Menge der Moeglichen }
  MdVorkommenden : tFarbenmenge; { Menge der Vorkommenden }
  Zeile,
  Spalte          : tIndex;

begin

  { Jede Farbe koennte in jeder Reihe vorkommen, bis zum Beweis des
    Gegenteils  }
  MdMoeglichen := [ blau..weiss ];

  { Pruefe die Zeilen. }
  for Zeile := 1 to inMaxZeile do
  begin
    { Ermittle die in der Zeile vorkommenden Farben. }
    MdVorkommenden := [];
    for Spalte := 1 to inMaxSpalte do
      MdVorkommenden := MdVorkommenden + [ inMosaik [Zeile, Spalte] ];
    { Nur sie kommen weiterhin in Betracht. }
    MdMoeglichen := MdMoeglichen * MdVorkommenden
  end; { for Zeile }

  { Pruefe die Spalten, }
  for Spalte := 1 to inMaxSpalte do
  begin
    { Ermittle die in der Spalte vorkommenden Farben. }
    MdVorkommenden := [];
    for Zeile := 1 to inMaxZeile do
      MdVorkommenden := MdVorkommenden + [ inMosaik [Zeile, Spalte] ];
    { Nur sie kommen weiterhin in Betracht. }
    MdMoeglichen := MdMoeglichen * MdVorkommenden
  end; { for Spalte }

  kommtVor := (MdMoeglichen <>[])

end; { kommtVor }
```

Aufgabe 7

```
procedure beginneZeileGross
( var ioText: text );
{ Ersetzt in ioText alle Buchstaben am Zeilenanfang durch entsprechende
  Grossbuchstaben. }
{ Version 1.0 vom 16.10.98 UBec,Lg }

  var
  Zeichen  : char;
  Hilfstext : text;

begin

  { Schreibe den Text in modifizierter Form in die Hilfsdatei. }
  reset (ioText);
  rewrite (Hilfstext);
  while not eof (ioText) do
  begin
    if not eoln (ioText) then
    begin
      read (ioText, Zeichen);
      { Uebernehme etwaige Leerzeichen am Zeilenanfang. }
      while (Zeichen = ' ') and not eoln (ioText) do
      begin
        write (Hilfstext, Zeichen);
        read (ioText, Zeichen)
      end; { while (Zeichen = '')...}
      { Wandle einen etwaigen ersten Kleinbuchstaben in den entsprechenden
        grossen. }
      if (Zeichen >= 'a') and (Zeichen <= 'z') then
        Zeichen := chr (ord (Zeichen) - ord ('a') + ord ('A'));
      { Schreibe den restlichen Text in die Hilfsdatei. }
      write (Hilfstext, Zeichen);
      while not eoln (ioText) do
      begin
        read (ioText, Zeichen);
        write (Hilfstext, Zeichen)
      end { while not eoln (ioText) }
    snd; { if not eoln (ioText) }
    readln (ioText);
    writeln (Hilfstext)
  end; { while not eof (ioText) }

  { Kopiere den geaenderten Text nach ioText zurueck. }
  reset (Hilfstext);
  rewrite (ioText);
  while not eof (Hilfstext) do
  begin
    while not eoln (Hilfstext) do
    begin
      read (Hilfstext, Zeichen);
      write (ioText, Zeichen)
    end; { while not eoln (Hilfstext) }
    readln (Hilfstext);
    writeln (ioText)
  end { while not eof (Hilfstext) }

snd; { beginneZeileGross }
```

Die Prozedur kann nicht auf input angewendet werden, da input nicht beschrieben werden kann.