Solutions modèles écrites en Delphi avec l'option "Console application"

Problème I – Paires amicales — 20 points

```
program PAIRES_AMICALES;
{$APPTYPE CONSOLE}
uses SysUtils;

var I,A,B : integer;

function SD (X : integer) : integer;
var SO,I : integer;
begin
   SO := 1;
   for I := 2 to round(sqrt(X)) do
      if X mod I = 0
      then SO := SO + I + X div I;
   SD := SO
end;

begin
  for I := 1 to 32767 do
     begin
        A := SD(I);
        B := SD(A);
        if (B = I) and (A < I)
        then writeln (A,'  ',I)
     end;
  readln
end.
```

La fonction SD (Somme_Diviseurs) retourne la somme des diviseurs demandée du nombre X.

Optimisation: on ne calcule qu'à la racine carrée de X, les diviseurs supérieurs à la racine carrée sont trouvés par X div I

La condition (A < I) évite l'affichage double:

220 284
et
284 220

et
des affichages comme

6 6
28 28
etc.

qui sont des nombres parfaits.

Remarque: une boucle suffit!

(* thx Claudi *)

Problème II - Occurrences 20 points

```
program occurrences;
{$APPTYPE CONSOLE}
uses SysUtils;

var f : text;
    occ : array ['A' .. 'Z'] of integer;
    i,ch : char;

begin

   (* ouverture du ficher d'entrée en lecture *)
   assign(f,'OCC_IN.TXT');
   reset(f);

   (* initialisation du tableau *)
   for i := 'A' to 'Z' do
      occ[i] := 0;

   (* traitement du fichier d'entrée *)
   while not eof(f) do
      begin
         read(f,ch);
         if ord(ch) > 91
         then ch := chr(ord(ch)-32);
         occ[ch] := occ[ch] + 1
      end;

   (* fermeture du ficher d'entrée *)
   close(f);

   (* affichage du résultat *)
   for i := 'A' to 'Z' do
      writeln(i,'  ',occ[i]);
   readln

end.
```

En définissant un tableau dont l'index est du type "caractère" (*char*) on obtient un programme court et facile. Le type *char* étant énumérable on peut l'utiliser à cette fin.

conversion en majuscules (voir code ASCII)

Un mot à propos des fichiers externes:

Pour utiliser un fichier externe de type texte (utilisé le plus souvent) on doit déclarer une variable du type *text*:

```
var f : text;
```

Pour établir la relation entre la variable `f` et un fichier externe on utilise l'instruction:

```
assign (f,'path:<nom du fichier>');
```

Pour ouvrir un fichier en lecture: `reset(f);`

Pour lire dans un fichier: `read(f,<nom de variable>)`

La fonction booléenne `eof(f)` (*end of file*) indique la fin du fichier.

Pour ouvrir un fichier en écriture: `rewrite(f);`

Pour écrire dans un fichier: `write(f,<nom de variable>)`

Pour fermer un fichier: `close(f)`

Problème III - Sudoku 30 points

```
program SUDOKU;
{$APPTYPE CONSOLE}
uses SysUtils;

var F : text;
    I,J,ROW,COL,ZERO,SOMME,X,Y : integer;
    SU : array [1..9,1..9] of integer;
    MODIFIE : boolean;

begin

  (* lecture du fichier d'entrée *)
  assign(F,'SUDO_IN.TXT');
  reset(F);
  for I := 1 to 9 do
     for J := 1 to 9 do
        read(F,SU[I,J]);
  close(F);

  repeat
     MODIFIE := false;

     (* on examine chaque ligne *)
     for I := 1 to 9 do
        begin
           ZERO := 0;
           SOMME := 0;
           COL := 0;
           for J := 1 to 9 do
              begin
                 SOMME := SOMME + SU[I,J];
                 if SU[I,J] = 0
                 then begin
                         ZERO := ZERO + 1;
                         COL := J
                      end
              end;
           if ZERO = 1
           then begin
                   SU[I,COL] := 45 - SOMME;
                   MODIFIE := true
                end
        end;

     (* on examine chaque colonne *)
     for J := 1 to 9 do
        begin
           ZERO := 0;
           SOMME := 0;
           ROW := 0;
           for I := 1 to 9 do
              begin
                 SOMME := SOMME + SU[I,J];
                 if SU[I,J] = 0
                 then begin
                         ZERO := ZERO + 1;
                         ROW := I
                      end
              end;
           if ZERO = 1
           then begin
                   SU[ROW,J] := 45 - SOMME;
                   MODIFIE := true
                end
        end;
```

Cette variable booléenne est mise à *true* si le programme modifie le Sudoku. Dans ce cas le traitement est relancé par la répétitive *repeat*.

On calcule la somme de tous les nombres de la i-ième ligne. La somme des nombres de 1 à 9 étant 45, le neuvième chiffre manquant est égal à: 45 - SOMME

On compte le nombre de zéros de la i-ième ligne.

Si on a trouvé un seul zéro dans la i-ième ligne on ajoute le chiffre manquant au Sudoku. Le Sudoku étant maintenant modifié, on met la valeur de la variable MODIFIE à vrai.

```
      (* on examine chaque sous-carré 3x3 *)
      X := 1;
      while X <= 7 do
         begin
            Y := 1;
            while Y <= 7 do
               begin
                  ZERO := 0;
                  SOMME := 0;
                  COL := 0;
                  ROW := 0;
                  for I := X to X + 2 do
                     for J := Y to Y + 2 do
                        begin
                           SOMME := SOMME + SU[I,J];
                           if SU[I,J] = 0
                           then begin
                                   ZERO := ZERO + 1;
                                   ROW := I;
                                   COL := J
                                end
                        end;
                        if ZERO = 1
                        then begin
                                SU[ROW,COL] := 45 - SOMME;
                                MODIFIE := true
                             end;
                  Y := Y + 3
               end;
            X := X + 3
         end

   until not MODIFIE;


   (* écriture du fichier de sortie *)
   assign(F,'SUDO_OUT.TXT');
   rewrite(F);
   for I := 1 to 9 do
      begin
         for J := 1 to 9 do
            write(F,SU[I,J],' ');
         writeln(F)
      end;
   close(F)

end.
```

Les premiers chiffres en haut et à gauche des sous-carrés ont les coordonnées:

1,1	1,4	1,7
4,1	4,4	4,7
7,1	7,4	7,7

d'où les instructions:

```
X := 1;
while X <= 7 do
for I := X to X + 2 do
```

et

```
X := X + 3
```

de même pour la variable Y

Problème IV – Maison de St. Nicolas 30 points

Bei dieser Aufgabe handelt es sich um ein Problem aus dem Bereich der Graphen. Hier wird ein so genannter Euler-Weg gesucht (siehe auch das Königsbergerbrückenproblem z.B. im "Duden für Informatik").

Als Literatur sei das Buch von Andreas Brandstädt empfohlen, Graphen und Algorithmen, B.G. Teubner Stuttgart. Es bleibt allerdings relativ theoretisch.

In unserer Aufgabe ist eine Lösung nur dann möglich, wenn als Startpunkt ein Knoten mit einer ungeraden Kantenzahl gewählt wird. Nur bei A und bei B ist die Kantenzahl ungerade nämlich 3. Bei C und E gibt es 4 Kanten und bei D sind es deren zwei. Wird als Startpunkt A gewählt ist der Endpunkt notgedrungen B und umgekehrt.

Im unteren Beispielprogramm wird daher nur Punkt A als Startpunkt benutzt. Mit einer Schleife ist es aber einfach alle Punkte zu testen (siehe unten). Das Programm gibt dann 44 verschiedene Lösungen aus. Mit B als Startpunkt gibt es also ebenfalls 44 Lösungen die zu den 44 "A"-Lösungen symmetrisch sind. Insgesamt ergeben sich demzufolge 88 verschiedene Lösungen.

Die hier benutzte Programmiermethode ist das so genannte "*Backtracking*". Hierbei werden alle möglichen Wege durchlaufen. Beim Erreichen einer Sackgasse wird der letzte Schritt rückgängig gemacht und ein anderer Weg ausprobiert. Dieses Rücknehmen (*backtracking*) und Neuprobieren erfolgt mit Hilfe rekursiver Aufrufe der Prozedur *TRYOUT*. Unter Rekursion versteht man dass Prozeduren sich selbst aufrufen.

Hierzu ist ein Studium des Buches von Niklaus Wirth, Datenstrukturen und Algorithmen, B.G. Teubner Stuttgart, angeraten. Unter dem Kapitel "*Backtracking*" findet man unter anderem die bekannten Aufgaben "Acht Damen Problem" und "Springerweg".

Zuerst muss die so genannte Adjazenzmatrix definiert werden. Wir wählen dazu ein zweidimensionales Feld mit Namen **P** (für *possible*). Als Index dienen die Buchstaben von A bis E und die Werte sind *true* oder *false*. Ein Wert *true* bedeutet dann, dass zwischen den beiden Knoten eine Verbindung besteht, *false* bedeutet, dass keine Verbindung besteht.

Beispiel:	P['A','B'] := true	zwischen A und B besteht eine Verbindung
	P['A','D'] := false	zwischen A und D besteht keine Verbindung

Dabei reicht es aus die Werte *true* zu definieren. Delphi setzt die Werte *false* als *default*.

Nun wird die Resultatsvariable *RES* mit dem Startpunkt (A) initialisiert und die Prozedur *TRYOUT* mit diesem Startpunkt als Parameter aufgerufen.

Die Prozedur *TRYOUT* versucht nun einen ersten Verbindungsweg mit Hilfe der Adjazenzmatrix zu finden. Wenn dies gelingt wird in der Adjazenzmatrix den eingeschlagene Weg als *false*, also nicht zur Verfügung stehend, markiert: `P[X,SUIV] := false;`. Auch der Rückweg wird versperrt: `P[SUIV,X] := false;`. Dann erfolgt der rekursive Aufruf der Prozedur *TRYOUT* mit dem jetzt erreichten Punkt als Parameter.

Falls kein Weg mehr zur Verfügung steht, wenn also vom zuletzt erreichten Punkt kein Weg mehr in der Adjazenzmatrix frei ist, dann kommt die Prozedur *TRYOUT* zum Ende und der rekursive Aufruf reicht eine Stufe zurück. Der zuletzt eingeschlagene Weg wird wieder frei:

```
P[X,SUIV] := true; P[SUIV,X] := true;
```

und in der Resultatsvariablen wird die letzte Stelle entfernt (rücknehmen: *backtracking*).

Mit einer einfachen Veränderung des Programms kann man sich alle Zwischenschritte anzeigen lassen (siehe veränderte Prozedur *TRYOUT*).

```
program NICOLAS;

{$APPTYPE CONSOLE}
uses SysUtils;

var P : array ['A'..'E','A'..'E'] of boolean;
    RES : string;

procedure TRYOUT (X : char);
var SUIV : char;
begin
   for SUIV := 'A' to 'E' do
      if P[X,SUIV]
      then begin
              P[X,SUIV] := false;
              P[SUIV,X] := false;
              RES := RES + SUIV;
              TRYOUT(SUIV);
              P[X,SUIV] := true;
              P[SUIV,X] := true;
              RES := copy(RES,1,length(RES)-1)
           end;
   if length(RES) = 9 then writeln(RES)
end;

begin
   P['A','B'] := true; P['A','C'] := true; P['A','E'] := true;
   P['B','A'] := true; P['B','C'] := true; P['B','E'] := true;
   P['C','A'] := true; P['C','B'] := true; P['C','D'] := true; P['C','E'] := true;
   P['D','C'] := true; P['D','E'] := true;
   P['E','A'] := true; P['E','B'] := true; P['E','C'] := true; P['E','D'] := true;
   RES := 'A';
   TRYOUT ('A');
   readln
end.
```

Ausprobieren eines möglichen Weges

Rekursiver Aufruf

Zurücknehmen

Ein Weg gefunden ! Ausgabe des Resultats.

Adjazenzmatrix

Um alle Punkte als Startpunkt zu testen:

```
for I := 'A' to 'E' do
  begin
    RES := I;
    TRYOUT (I)
  end;
```

Veränderte Prozedur TRYOUT, zum Anzeigen aller Zwischenergebnisse:

```
procedure TRYOUT (X : char);
var SUIV : char;
begin
   for SUIV := 'A' to 'E' do
      if P[X,SUIV]
      then begin
              RES := RES + SUIV;
              P[X,SUIV] := false;
              P[SUIV,X] := false;
              writeln(RES);
              readln;
              TRYOUT(SUIV);
              P[X,SUIV] := true;
              P[SUIV,X] := true;
              RES := copy(RES,1,length(RES)-1);
              writeln(RES);
              readln;
           end;
   if length(RES) = 9 then writeln('Solution found: ',RES)
end;
```

Anzeigen der Zwischenresultate (vorwärts)

Anzeigen der Zwischenresultate (rückwärts)