Solutions modèles en Delphi-Pascal

Remarque générale

Nos programmes ne prétendent pas être les solutions optimales et ne tournent que sous Delphi. Nos programmes n'ont pas été conçus pour participer aux concours CIL et IOI. Ils ont été écrits pour montrer une approche possible à la résolution des problèmes posés aux élèves participants et aux élèves intéressés à une participation ultérieure. En matière d'entrée/sortie, nos programmes ne correspondent pas nécessairement à l'énoncé, mais affichent souvent des résultats intermédiaires à l'écran pour montrer leur déroulement.

Problème I - Simplification pittienne — 20 points

Dans l'intervalle donné (10..99) se trouvent 4 fractions simplifiables par la méthode "pittienne", ce sont:

```
16/64
19/95
26/65
49/98
```

```
program Simplification;
{$APPTYPE CONSOLE}

uses SysUtils;

var i,j : integer;

function simplif(a,b : integer) : boolean;
var a1,a2,b1,b2,ares,bres : integer;
begin
    ares := 100;
    bres := 100;
    a1 := a div 10;
    a2 := a mod 10;
    b1 := b div 10;
    b2 := b mod 10;
    if (a1 <> 0) and (b1 <> 0) and (a2 <> 0) and (b2 <> 0)
    then begin
            if a1 = b1
            then begin
                    ares := a2;
                    bres := b2
                end;
            if a1 = b2
            then begin
                    ares := a2;
                    bres := b1
```

Calcul des valeurs pour les 4 chiffres intervenants.

Exclusion de la valeur zéro.

```
                end;
            if a2 = b1
            then begin
                    ares := a1;
                    bres := b2
                end;
            if a2 = b2
            then begin
                    ares := a1;
                    bres := b1
                end
         end;
    simplif := a/b = ares/bres
end;

begin
  for i := 10 to 99 do
     for j := i + 1 to 99 do
        if simplif(i,j)
        then writeln(i,'/',j);
  readln
end.
```

Les 2 variables *ares* et *bres* contiennent les chiffres restants après la simplification.

Dans les 2 boucles que voici toutes les fractions entre 10 et 99, dont le numérateur est strictement inférieur au dénominateur, sont testées et les fractions simplifiables par la méthode "pittienne" sont affichées à l'écran.

Test des programmes:

Points accordés: 5 points par fraction exacte (4 x 5 = 20 pts)

Problème II – Sudoku - solitaires nus, solitaires cachés 40 points

```
program sudoku;
{$APPTYPE CONSOLE}

uses SysUtils;

type tSudo = array [1..9,1..9] of integer;
     tListe = array [1..9,1..9] of string;

var Sudo : tSudo;
    Liste : tListe;
    SNu,SCache : boolean;

procedure input(var Sudo : tSudo);
var f : text;
    i,j : integer;
begin
   assign(f,'SUDOIN.TXT');
   reset(f);
   for i := 1 to 9 do
      begin
         for j := 1 to 9 do
            read(f,Sudo[i,j]);
         readln(f)
      end;
   close(f)
end;

procedure output(Sudo : tSudo);
```

Pour implémenter la liste des candidats, on a plusieurs possibilités. Ici, on utilise un tableau à 2 dimensions contenant des chaînes de caractères; soit des chiffres entre 1 et 9, pour les cellules vides, soit '*xxx*' pour les cellules occupées.

Lecture du fichier d'entrée.

```
var f : text;
    i,j : integer;
begin
   assign(f,'SUDOOUT.TXT');
   rewrite(f);
   for i := 1 to 9 do
      begin
         for j := 1 to 9 do
            write(f,Sudo[i,j],' ');
         writeln(f)
      end;
   close(f)
end;

procedure initListe(var Liste : tListe);
var i,j : integer;
begin
   for i := 1 to 9 do
      for j := 1 to 9 do
         Liste[i,j] := ''
end;

function ok(k,i,j : integer) : boolean;
var Temp : boolean;
    lig,col,xBox,yBox : integer;
begin
   Temp := true;
   for col := 1 to 9 do
      if Sudo[i,col] = k
      then Temp := false;
   for lig := 1 to 9 do
      if Sudo[lig,j] = k
      then Temp := false;
   if i <= 3
   then xBox := 1
   else if i <= 6
        then xBox := 4
        else xBox := 7;
   if j <= 3
   then yBox := 1
   else if j <= 6
        then yBox := 4
        else yBox := 7;
   for lig := xBox to xBox + 2 do
      for col := yBox to yBox + 2 do
         if Sudo[lig,col] = k
         then Temp := false;
   ok := temp
end;

procedure doListe(var Liste : tListe);
var i,j,k : integer;
begin
   initListe(Liste);
   for i := 1 to 9 do
      for j := 1 to 9 do
         begin
            for k := 1 to 9 do
               if ok(k,i,j)
               then Liste[i,j] := Liste[i,j] + inttostr(k);
            if Sudo[i,j] <> 0
            then Liste[i,j] := 'xxx'
         end
end;
```

Ecriture du fichier de sortie.

Initialisation de la liste des candidats avec des chaînes vides.

Cette fonction vérifie si on peut mettre le chiffre *k* dans la cellule *(i,j)*.

Cette procédure construit la liste des candidats en utilisant la fonction *ok*.

Initialisation avec des chaînes vides.

Chaîne contenant tous les candidats de la cellule *(i,j)*. Exemple: *'2578'*.

Pour les cellules occupées, la liste contient la chaîne *'xxx'*.

```
procedure showListe(Liste : tListe);
var i,j : integer;
begin
   for i := 1 to 9 do
      begin
         for j := 1 to 9 do
            write(Liste[i,j]:7,' ');
         writeln
      end;
   writeln
end;

procedure showSudo(Sudo : tSudo);
var i,j : integer;
begin
   for i := 1 to 9 do
      begin
         for j := 1 to 9 do
            write(Sudo[i,j],' ');
         writeln
      end;
   writeln
end;

procedure SolitNu(var Sudo : tSudo);
var i,j : integer;
begin
   SNu := false;
   for i := 1 to 9 do
      for j := 1 to 9 do
         begin
            if length(Liste[i,j]) = 1
            then begin
                    Sudo[i,j] := strtoint(Liste[i,j]);
                    SNu := true
                 end
         end
end;

function Cont(k : integer; str : string) : boolean;
var Temp : boolean;
    i : integer;
begin
   Temp := false;
   for i := 1 to length(str) do
      if str[i] = inttostr(k)
      then Temp := true;
   Cont := Temp
end;

procedure SolitCache(var Sudo : tSudo);
var k,i,j,NumK,xK,yK,lig,col,carre,cX,cY : integer;
begin
   SCache := false;
   { lignes }
   for k := 1 to 9 do
      for lig := 1 to 9 do
         begin
            NumK := 0;
            for i := 1 to 9 do
               if Cont(k,Liste[lig,i])
               then begin
                       NumK := NumK + 1;
```

Affichage de la liste des candidats à l'écran.

Affichage du Sudoku à l'écran.

Si la liste des candidats ne contient qu'un seul candidat dans une cellule, on a trouvé un solitaire nu.

Cette fonction vérifie la présence du candidat *k* dans la chaîne *str*.

Pour les solitaires cachés, il faut compter leurs occurrences. On trouve un solitaire caché si le nombre d'occurrences vaut 1.

On compte les occurrences des candidats dans les lignes.

```
                        xK := lig;
                        yK := i
                    end;
            if NumK = 1
            then begin
                    Sudo[xK,Yk] := k;
                    SCache := true
                end
        end;
    { colonnes }
    for k := 1 to 9 do
       for col := 1 to 9 do
          begin
             NumK := 0;
             for i := 1 to 9 do
                if Cont(k,Liste[i,col])
                then begin
                        NumK := NumK + 1;
                        xK := i;
                        yK := col
                    end;
             if NumK = 1
             then begin
                     Sudo[xK,Yk] := k;
                     SCache := true
                  end
          end;
    { sous-carrés }
    for k := 1 to 9 do
       for carre := 1 to 9 do
          begin
             NumK := 0;
             case carre of
               1,4,7 : cX := 1;
               2,5,8 : cX := 4;
               3,6,9 : cX := 7
             end;
             case carre of
               1,2,3 : cY := 1;
               4,5,6 : cY := 4;
               7,8,9 : cY := 7
             end;
             for i := cX to cX + 2 do
                for j := cY to cY + 2 do
                   if Cont(k,Liste[i,j])
                   then begin
                           NumK := NumK + 1;
                           xK := i;
                           yK := j
                        end;
                if NumK = 1
                then begin
                        Sudo[xK,Yk] := k;
                        SCache := true
                     end
          end
end;

begin
    input(Sudo);
    showSudo(Sudo);
    readln;
    initListe(Liste);
    doListe(Liste);
```

Dans les colonnes.

Et dans les sous-carrés 3x3.

Programme principal.

```
    showListe(Liste);
    readln;
    repeat
       SolitNu(Sudo);
       if SNu
       then begin
               writeln('Solitaire nu !');
               readln;
               showSudo(Sudo);
               doListe(Liste);
               showListe(Liste);
               readln
            end;
       SolitCache(Sudo);
       if SCache
       then begin
               writeln('Solitaire cache !');
               readln;
               showSudo(Sudo);
               doListe(Liste);
               showListe(Liste);
               readln
            end
    until (not SNu) and (not SCache);
    output(Sudo)
end.
```

Recherche de solitaires nus.

Recherche de solitaires cachés.

Jusqu'à ce qu'on ne trouve plus de solitaires nus ou cachés.

Test des programmes:

Points accordés: 10 points par test (4 x 10 = 40 pts)

SUDOIN.TXT	SUDOOUT.TXT
1) Exemple donné: 8 0 0 0 0 0 1 0 0 0 9 0 0 0 3 0 8 2 0 2 0 4 9 0 0 0 0 0 3 0 0 0 2 0 7 0 7 0 0 5 0 6 0 0 1 0 6 0 9 0 0 0 2 0 0 0 0 0 2 4 0 5 0 2 5 0 8 0 0 0 9 0 0 0 3 0 0 0 0 0 8	 8 0 0 2 0 5 1 3 9 0 9 0 0 1 3 4 8 2 3 2 1 4 9 8 0 6 0 0 3 9 1 0 2 0 7 0 7 8 2 5 3 6 9 4 1 1 6 0 9 0 7 0 2 3 9 1 8 3 2 4 0 5 0 2 5 0 8 0 1 3 9 4 0 0 3 0 5 9 2 1 8
2) Sudoku vide: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

3) Exemple permettant une résolution complète: 0 0 0 5 0 0 2 0 9 0 0 0 7 9 8 0 4 0 0 0 3 0 0 0 0 0 0 0 0 1 0 0 7 5 9 8 9 0 6 0 0 0 3 0 7 7 8 5 3 0 0 4 0 0 0 0 0 0 0 0 7 0 0 0 3 0 9 7 4 0 0 0 1 0 4 0 0 2 0 0 0	8 6 7 5 4 3 2 1 9 5 1 2 7 9 8 6 4 3 4 9 3 2 1 6 8 7 5 3 2 1 4 6 7 5 9 8 9 4 6 1 8 5 3 2 7 7 8 5 3 2 9 4 6 1 2 5 9 6 3 1 7 8 4 6 3 8 9 7 4 1 5 2 1 7 4 8 5 2 9 3 6
4) Sudoku terminé: 8 6 7 5 4 3 2 1 9 5 1 2 7 9 8 6 4 3 4 9 3 2 1 6 8 7 5 3 2 1 4 6 7 5 9 8 9 4 6 1 8 5 3 2 7 7 8 5 3 2 9 4 6 1 2 5 9 6 3 1 7 8 4 6 3 8 9 7 4 1 5 2 1 7 4 8 5 2 9 3 6	8 6 7 5 4 3 2 1 9 5 1 2 7 9 8 6 4 3 4 9 3 2 1 6 8 7 5 3 2 1 4 6 7 5 9 8 9 4 6 1 8 5 3 2 7 7 8 5 3 2 9 4 6 1 2 5 9 6 3 1 7 8 4 6 3 8 9 7 4 1 5 2 1 7 4 8 5 2 9 3 6

Problème III – Anagrammes 40 points

Avant de programmer, voici quelques réflexions préliminaires. Nous nous demandons combien d'anagrammes différentes existent.

a) Toutes les lettres sont différentes (par exemple: *JEAN*)

Soit N le nombre de lettres (dans l'exemple N = 4)
alors on a N! (factorielle) possibilités (dans l'exemple 4! = 1 · 2 · 3 · 4 = 24)

pour *JEAN* on obtient:

JEAN	*EJAN*	*AJEN*	*NJEA*
JENA	*EJNA*	*AJNE*	*NJAE*
JAEN	*EAJN*	*AEJN*	*NEJA*
JANE	*EANJ*	*AENJ*	*NEAJ*
JNEA	*ENJA*	*ANJE*	*NAJE*
JNAE	*ENAJ*	*ANEJ*	*NAEJ*

l'énoncé restreint le nombre de lettres au total à 7
le nombre maximal d'anagrammes est donc 7! = 5040

b) Le mot contient des répétitions de lettres (par exemple: *JANA*)

On obtient alors moins de 24 possibilités.

Soit N le nombre total de lettres (dans l'exemple N = 4)
soit X le nombre de lettres différentes (dans l'exemple X = 3; *J*, *A* et *N*)

soit $M_1, M_2, \ldots, M_x$ l'occurrence des lettres (dans l'exemple $M_1 = 1$; pour la lettre *J*, $M_2 = 2$; pour la lettre *A* et $M_3 = 1$; pour la lettre *N*)

alors le nombre de possibilités est

$$\frac{N!}{M_1! \cdot M_2! \cdot \ldots \cdot M_x!}$$

dans l'exemple:

$$\frac{4!}{1! \cdot 2! \cdot 1!} = \frac{4 \cdot 3 \cdot 2!}{2!} = 4 \cdot 3 = 12$$

Les 12 anagrammes sont:

JANA	*ANJA*
JAAN	*ANAJ*
JNAA	*AAJN*
AJNA	*NJAA*
ANJA	*NAJA*
AJNA	*NAAJ*

Un dernier exemple:

Soit le mot:	*ABBA*
nombre de lettres au total:	N = 4
nombre de lettres différentes:	X = 2
occurences:	$M_1 = 2$ et $M_2 = 2$
possibilités:	$\frac{4!}{2! \cdot 2!} = \frac{1 \cdot 2 \cdot 3 \cdot 4}{1 \cdot 2 \cdot 1 \cdot 2} = 2 \cdot 3 = 6$
les 6 anagrammes:	*ABBA* *ABAB* *AABB* *BABA* *BAAB* *BBAA*

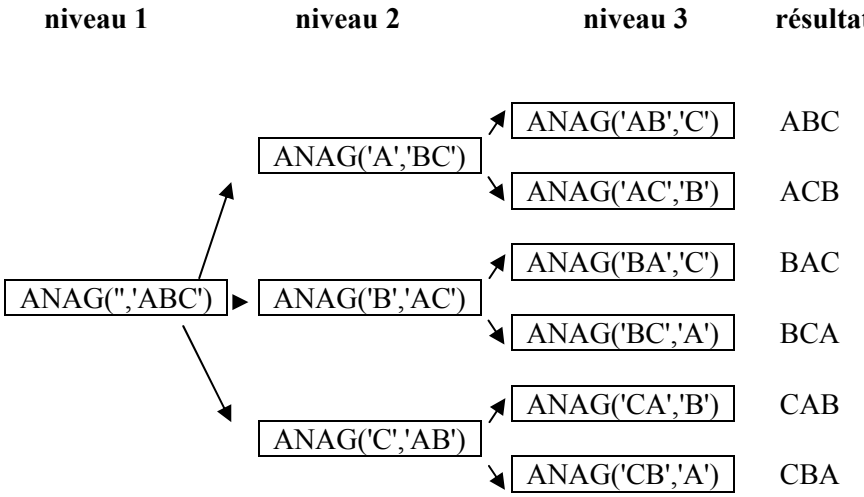
Programme a) (toutes les lettres sont différentes): incomplet pour les exigences de l'énoncé

```
program Anagrammes;

var str : string;
    Howmany : integer;

procedure ANAG (deb, ch : string);
var i,l : integer;
    ch2 : string;
begin
   l := length(ch);
   if l = 1
   then begin
```

```
            Howmany := howmany + 1;
            writeln(deb + ch)
         end
    else for i := 1 to l do
            begin
               ch2 := copy(ch,1,i-1) + copy(ch,i+1,l-i);
               ANAG(deb + ch[i],ch2)
            end
end;

begin
  Howmany := 0;
  readln(str);
  ANAG('',str);
  writeln(Howmany);
  readln
end.
```

Appel récursif:
Etudier l'exemple suivant !

Exemple d'exécution: str = 'ABC'



Programme b) (le mot contient des répétitions de lettres): complet pour les exigences de l'énoncé

```
program Anagrammes;

var str : string;
    liste : array [1..5040] of string;    { 7! }
    Howmany : integer;

function inList(str : string) : boolean;
var temp : boolean;
    i : integer;
begin
   temp := false;
   for i := 1 to howmany do
      if liste[i] = str
      then temp := true;
   inList := temp
end;

procedure AddtoList (str : string);
begin
   if not inList(str)
   then begin
           Howmany := Howmany + 1;
```

Pour éviter les répétitions, on met l'anagramme trouvée dans une liste, mais seulement si elle ne s'y trouve pas encore.

```
            liste[Howmany] := str
         end
end;

procedure outList;
var i : integer;
begin
   for i := 1 to Howmany do
      writeln(liste[I])
end;

procedure ANAG (deb, ch : string);
var i,l : integer;
    ch2 : string;
begin
   l := length(ch);
   if l = 1
   then addtolist(deb + ch)
   else for i := 1 to l do
           begin
              ch2 := copy(ch,1,i-1) + copy(ch,i+1,l-i);
              ANAG(deb + ch[i],ch2)
           end
end;

begin
  Howmany := 0;
  readln(str);
  ANAG('',str);
  outList;
  writeln(Howmany);
  readln
end.
```

On met l'anagramme trouvée dans une liste.

On affiche également le nombre d'anagrammes trouvées pour vérifier nos réflexions du début.

Test des programmes:

Points accordés: 10 points par test (4 x 10 = 40 pts)

1.	JEAN	(exemple donné)
2.	ABBA	(exemple donné)
3.	A	(cas limite)
4.	ABCDEFG	(cas limite)