Solutions modèles en Pascal

Problème I - 3N + 1 — 10 points

Suite intéressante, mais programme simple, pas de commentaires.

```
program Suite;

var n : integer;

begin
  readln(n);
  write(n,' ');
  while n > 1 do
     begin
          if n mod 2 = 0
          then n := n div 2
          else n := 3 * n + 1;
          write(n,' ');
     end;
  readln
end.
```

Problème II - La tirelire d'Alexandre — 20 points

Ce problème est du domaine des arrangements, combinaisons et permutations. Ici il s'agit des arrangements sans répétition de 3 éléments choisis parmi 9. Le nombre d'arrangements sans répétition de p éléments choisis parmi n est le nombre défini par:

$$A_n^p = \frac{n!}{(n-p)!}$$

Dans notre exemple $n = 9$ et $p = 3$.Nous obtenons donc:

$$A_9^3 = \frac{9!}{(9-3)!} = \frac{9!}{6!} = \frac{9 \cdot 8 \cdot 7 \cdot 6!}{6!} = 9 \cdot 8 \cdot 7 = 504$$

Notre programme doit par conséquent afficher 504 combinaisons différentes.

Voici le programme:

```
program Tirelire;

var i,j,k,num : integer;
    f : text;

begin
   assign(f,'TIRELIRE.TXT');
   rewrite(f);
   num := 0;
   for i := 1 to 9 do
      for j := 1 to 9 do
         for k := 1 to 9 do
            if (i <> j) and (j <> k) and (i <> k)
            then begin
                    writeln(f,i,j,k);
                    num := num + 1
                 end;
   writeln(f,num);
   close(f)
end.
```

On essaye toutes les combinaisons possibles.

Mais on n'affiche que celles qui ne contiennent pas de répétitions.

Ce programme affiche effectivement 504 combinaisons.

Problème III - Money 30 points

Avant de programmer calculons la place mémoire nécessaire pour enregistrer les différents montants.

L'énoncé stipule un maximum de 10 billets de chaque sorte. Le montant le plus grand vaut donc:

$$10 \cdot 5 + 10 \cdot 10 + 10 \cdot 20 + 10 \cdot 50 + 10 \cdot 100 + 10 \cdot 200 + 10 \cdot 500 =$$
$$50 + 100 + 200 + 500 + 1000 + 2000 + 5000 = 8850$$

Ce montant en pas de cinq euros:

$$\frac{8850}{5} = 1770$$

Nous avons donc besoin d'un tableau d'entiers de dimension maximale de 1770.

```
program Money;

type ttab = array [1..1770] of integer;

var f : text;
    A5,A10,A20,A50,A100,A200,A500 : integer;
    I,J,K,L,M,N,O,sum,t : integer;
    tab : ttab;

procedure exchange (var a,b : integer);
var t : integer;
begin
   t := a;
   a := b;
   b := t
end;
```

La fonction *intab* vérifie si *sum* est contenu dans le tableau *ta*.

```
function intab (sum : integer; ta :ttab; t : integer) : boolean;
var temp : boolean;
    i : integer;
begin
   temp := false;
   for i := 1 to t do
      if ta[i] = sum
      then temp := true;
   intab := temp
end;
```

Cette procédure trie le tableau *ta* par la méthode de sélection.

```
procedure sort (var ta : ttab; t : integer);
var i,j,min,posmin : integer;
begin
   for i := 1 to t-1 do
      begin
         min := ta[i];
         posmin := i;
         for j := i+1 to t do
            if ta[j] < min
            then begin
                    min := ta[j];
                    posmin := j
                 end;
         exchange(ta[i],ta[posmin])
      end
end;

procedure output (ta : ttab; t : integer);
var i : integer;
    ff : text;
begin
   assign(ff,'MONEY_OUT.TXT');
   rewrite(ff);
   for i := 1 to t do
      writeln(ff,ta[i]);
   close(ff)
end;
```

```
begin
  assign(f,'MONEY_IN.TXT');
  reset(f);
  read(f,A5);
  read(f,A10);
  read(f,A20);
  read(f,A50);
  read(f,A100);
  read(f,A200);
  read(f,A500);
  close(f);
  t := 0;
  for I := 0 to A5 do
     for J := 0 to A10 do
        for K := 0 to A20 do
           for L := 0 to A50 do
              for M := 0 to A100 do
                 for N := 0 to A200 do
                    for O := 0 to A500 do
                       begin
                          sum := I*5 + J*10 + K*20 + L*50 +
                                 M*100 + N*200 + O*500;
                          if sum > 0
                          then if not intab(sum,tab,t)
                               then begin
                                       t := t + 1;
                                       tab[t] := sum
                                    end
                       end;
  sort(tab,t);
  output(tab,t);
  writeln(t);
  readln
end.
```

Dans ces 7 répétitives imbriqués on construit tous les montants possibles.

Classement du tableau !

On ajoute le montant *sum* au tableau seulement s'il est différent de 0 et s'il n'est pas déjà contenu dans le tableau.

Problème IV - Sudoku 40 points

```
program Sudoku;

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

var Sudo : tSudo;
    Liste : tListe;
    SNu : boolean;
```

```
procedure lireSudo (var Sudo : tSudo);
var i,j : integer;
    f : text;
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;
```

Lecture du Sudoku dans le fichier externe 'SUDOIN.txt'. Pour des explications supplémentaires, consulter les solutions modèles de l'année passée !

```
function ok(k,i,j : integer) : boolean;
var Temp : boolean;
    lig,col,xBox,yBox : integer;
begin
   Temp := true;
   { lignes }
   for col := 1 to 9 do
      if Sudo[i,col] = k
      then Temp := false;
   { colonnes }
   for lig := 1 to 9 do
      if Sudo[lig,j] = k
      then Temp := false;
   { sous-carrés 3x3 }
   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;
```

La fonction *ok* vérifie si le candidat k peut être placé dans la cellule [i,j].
La vérification se fait dans les lignes, les colonnes et les sous-carrés 3x3.

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

Pour construire la liste des candidats, on vérifie tous les chiffres de 1 à 9 dans toutes les cases du Sudoku à l'aide de la fonction *ok*. Pour les chiffres déjà placés on met *xxx* dans la liste des candidats.

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

Un solitaire nu est le seul candidat de la cellule. La liste des candidats ne contient donc qu'un seul candidat (length = 1).

```
function Done : boolean;
begin
   Done := not Snu
end;
```

```
procedure ecrireSudo (Sudo : tSudo);
var i,j : integer;
    f : text;
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;
```

Ecriture du Sudoku dans le fichier externe 'SUDOOUT.txt'. Pour des explications supplémentaires, consulter les solutions modèles de l'année passée!

```
begin
   lireSudo(Sudo);
   construireListe(Liste);
   repeat
       placerSolitaireNu(Sudo);
       construireListe(Liste)
   until Done;
   ecrireSudo(Sudo);
end.
```

Test des programmes

Les programmes des élèves ont été testés avec les entrées suivantes:

I. Suite (10 pts, 2 pts par test)

1.	11	(exemple donné)
2.	0	(hors limites)
3.	1	(nombre limite)
4.	12	(un nombre pair)
5.	1000000	(nombre maximal)

II. Tirelire (20 pts)

résultat correct = 20 pts

III. Money (30 pts, 6 pts par test)

1.	1 0 1 0 0 0 0	(exemple donné)
2.	10 10 10 10 10 10 10	(nombres maximaux)
3.	0 0 0 0 0 0 0	(cas limite)
4.	5 5 5 5 5 5 5	(cas quelconque)
5.	0 0 0 1 0 0 0	(test simple)

IV. Sudoku (40 pts, 8 pts par test)

1. (exemple donné dans l'énoncé)

```
0 7 6 2 4 8 0 5 0
0 0 0 7 5 0 0 0 0
0 8 0 0 0 0 6 0 0
0 0 0 0 0 0 0 7 4
4 0 8 0 1 0 3 0 2
2 3 0 0 0 0 0 0 0
0 0 3 0 0 0 0 6 0
0 0 0 0 2 9 0 0 0
0 9 0 8 6 5 1 4 0
```

2. (exemple impossible)

```
0 7 6 2 4 8 0 5 0
0 0 0 7 5 0 0 0 0
0 8 0 0 0 0 6 0 0
0 0 0 0 0 0 0 7 4
4 0 8 0 1 0 3 0 2
2 3 0 0 0 0 0 0 0
0 0 3 0 0 0 0 6 0
0 0 0 0 2 9 0 0 0
0 9 0 8 6 5 1 4 1
```

3. (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
```

4.
```
9 7 0 3 0 4 0 6 5
0 2 0 5 0 6 0 8 0
0 0 0 0 0 0 0 0 0
0 0 5 8 0 2 9 0 0
0 0 2 0 4 0 3 0 0
0 0 8 7 0 5 1 0 0
0 0 0 0 0 0 0 0 0
0 6 0 2 0 8 0 3 0
8 4 0 1 0 9 0 2 7
```
(exemple permettant de résoudre complètement le Sudoku)

5.
```
9 7 1 3 8 4 2 6 5
3 2 4 5 1 6 7 8 9
5 8 6 9 2 7 4 1 3
4 1 5 8 3 2 9 7 6
7 9 2 6 4 1 3 5 8
6 3 8 7 9 5 1 4 2
2 5 7 4 6 3 8 9 1
1 6 9 2 7 8 5 3 4
8 4 3 1 5 9 6 2 7
```
(Soduko complété, solution du Sudoku précédant)