Corrigé du test algorithmique n°2

Exercice 1 : transformations de chaînes de caractères

ÉCHANGE DE LETTRES

1) Une simple boucle explore la chaîne `ch`, et change le caractère courant si celui-ci est un `'A'` :

```
procedure AversE(var ch : string);

var i : integer;

begin
   for i := 1 to length(ch) do
      if ch[i] = 'A' then ch[i] := 'E'
end; { AversE }
```

2) Il suffit de transformer un peu le test interne :

```
procedure AaversEe(var ch : string);

var i : integer;

begin
   for i := 1 to length(ch) do
      if ch[i] = 'A'
         then ch[i] := 'E'
      else if ch[i] = 'a'
         then ch[i] := 'e'
end; { AversE }
```

JAVANAISE

Ici, la chaîne retournée est construite caractère par caractère, par concaténation avec l'opérateur `+`. Le premier exemple montre l'utilisation astucieuse d'une *factorisation* de `if` imbriqués, le second l'utilisation d'un `case`.

```
procedure javanaise(ch : string; var chr : string);

var
   i : integer;
   a : char;

begin
   chr := '';
   for i := 1 to length(ch) do begin
      a := ch[i]
      if (a = 'A') or (a = 'E') or (a = 'I')
         or (a = 'O') or (a = 'U') or (a = 'Y')
      then chr := chr + a + 'V' + a
      else chr := chr + ch[i]
   end;
end; { javanaise }
```

```
procedure javanaisecase(ch : string; var chr : string);

var i : integer;

begin
   chr := '';
   for i := 1 to length(ch) do
      case ch[i] of
        'A' : chr := chr + 'AVA';
        'E' : chr := chr + 'EVE';
        'I' : chr := chr + 'IVI';
        'O' : chr := chr + 'OVO';
        'U' : chr := chr + 'UVU';
        'Y' : chr := chr + 'YVY';
      else chr := chr + ch[i]
      end;
end; { javanaisecase }
```

Exercice 2 : remplissage de tableaux

REMPLISSAGE DE TABLEAUX

Pour remplir un tableau, on utilise une boucle simple.

1) La procédure la plus simple : on met toutes les cases à 0.

```
procedure init_tableau(var t : tableau);

var i : integer;

begin
   for i := 1 to N do t[i] := 0
end; { init_tableau }
```

2) On peut procéder de deux façons différentes :

- initialiser toutes les cases à 0 puis changer le 0 de la case d'indice i en 1 :

```
procedure unun_tableau(var t : tableau; i : integer);

var k : integer;

begin
   for k := 1 to N do t[k] := 0;
   t[i] := 1
end; { unun_tableau }
```

- mettre directement la bonne valeur dans chaque case à l'aide d'un test :

```
procedure unun_tableau_bis(var t : tableau; i : integer);

var k : integer;

begin
   for k := 1 to N do
      if k=i then t[k] := 1 else t[k] := 0
end; { unun_tableau_bis }
```

La première version semble préférable, puisqu'elle évite la répétition du test à l'intérieur de la boucle. La seconde méthode n'est meilleure que si l'on ne peut pas revenir en arrière (cas de l'écriture d'un fichier séquentiel), ou si le coût de l'écriture est largement supérieur à celui d'un test (cas de l'écriture sur un support très lent, et non bufferisé).

3) Voici deux versions de cette procédure de remplissage alterné :

- une première version, astucieuse, profite du fait que si `x` prend l'une des deux valeurs `0` ou `1`, `1-x` prend l'autre :

```
procedure unzeroalt_tableau(var t : tableau);

var k : integer;

begin
   t[1] := 1;
   for k := 2 to N do
      t[k] := 1-t[k-1]
end; { unzeroalt_tableau }
```

- une deuxième version qui limite au maximum les accès au tableau `t` :

```
procedure unzeroalt_tableau_bis(var t : tableau);

var
   k,d : integer;

begin
   d := 1;
   for k := 1 to N do begin
      t[k] := d;
      if d = 1     (* On change la valeur de d *)
      then d := 0
      else d := 1
   end
end; { unzeroalt_tableau_bis }
```

REMPLISSAGE DE MATRICES

Le remplissage d'une matrice nécessite l'usage de deux boucles imbriquées, une qui explore les lignes, l'autre les colonnes.

1) L'initialisation à `0` est la procédure la plus simple :

```
procedure init_matrice(var a : matrice);

var i,j : integer;

begin
   for i := 1 to P do
      for j := 1 to Q do
         a[i,j] := 0
end; { init_matrice }
```

2) On reprend la première version de la procédure similaire pour les tableaux :

```
procedure unun_matrice(var a : matrice; i,j : integer);

var k,l : integer;

begin
   for k := 1 to P do
      for l := 1 to Q do
         a[k,l] := 0;
   a[i,j] := 1
end; { unun_matrice }
```

3) Pour les mêmes raisons que précédemment, la version directe ne doit être préférée que si les écritures sont bien plus coûteuses que les tests.

- Première version :

```
procedure lico_mat_deux_temps(var a : matrice; i,j : integer);

var k,l : integer;

begin
   for k := 1 to P do
      for l := 1 to Q do
         a[k,l] := 0;
   for k := 1 to P do a[k,j] := 1;
   for l := 1 to Q do a[i,l] := 1
end; { lico_mat_deux_temps }
```

- deuxième version :

```
procedure lico_mat_direct(var a : matrice; i,j : integer);

var k,l : integer;

begin
   for k := 1 to P do
      for l := 1 to Q do
         if (k=i) or (l=j)
         then a[k,l] := 1
         else a[k,l] := 0
end; { lico_mat_direct }
```

4) On reprend l'astuce de la partie sur les tableaux.

```
procedure unzero_alt_matrice(var a : matrice);

var i,j,ds : integer;

begin
   ds := 1; (* ds contient la valeur de la première *)
            (* case de la ligne courante.          *)
   for i := 1 to P do begin
      a[i,1] := ds;       (* Première case de la ligne j *)
      for j := 2 to Q do (* On remplit la ligne j       *)
         a[i,j] := 1-a[i,j-1];
      ds := 1-ds          (* On change la valeur de ds   *)
   end
end; { unzero_alt_matrice }
```