Examen de janvier 2007

durée 2h - documents non autorisés

Exercice 1 : *Les cartes*

Dans cet exercice, on considère une situation initiale des tas de cartes dans laquelle le tas n°1 contient un nombre quelconque de `T` et de `K`, mais en tout cas au moins un `T` qui peut être situé n'importe où.

Q 1 .

Parmi les initialisations du tas n°1 qui suivent une seule correspond à celle souhaitée. Laquelle ? Vous donnerez des réalisations effectives qui montrent que les autres initialisations sont incorrectes.

1. `initTas(1,'T[T]+[K]');`
2. `initTas(1,'[T+K]T[T+K]');`
3. `initTas(1,'(T+K)[T+K]');`

Q 2 . On souhaite que le robot place le dernier `T` (c'est–à–dire celui situé le plus bas) dans le tas n°1 sur le tas n°2, toutes les autres cartes se trouvant sur le tas n°3.

Q 2.1. Commencez par faire en sorte que le premier `T` se retrouve sur le tas n°2, les `K` initialement situés au dessus se retrouvant sur le tas n°3.

Q 2.2. Poursuivez le programme afin d'atteindre la situation souhaitée.

Q 3 . Si on suppose que le tas n°3 est initialement vide comment programmer le robot pour qu'il remette toutes les cartes du tas n°3 sur le tas n°1 ?

Q 4 . On suppose maintenant que le tas n°3 n'est pas nécessairement vide.

Q 4.1. Que faut-il ajouter au robot afin qu'il puisse remettre sur le tas n°1 toutes les cartes du tas n°3 initialement sur le tas n°1 ?

Q 4.2. Reprogrammez le robot en conséquence, et déclarez les variables que vous utilisez.

Q 5 . Réalisez une procédure nommée `deplacerDernierTrefle` à trois paramètres `i,j,k` qui déplace le dernier `T` du tas `i` vers le tas `j` en se servant du tas `k` comme tas intermédiaire, les trois tas étant supposés distincts. Précisez les autres contraintes d'utilisation de cette procédure.

Q 6 . Dans cette dernière question, on suppose que tous les tas sont initialisés comme il a été décrit au départ pour le tas n°1. Utilisez la procédure réalisée précédemment pour arriver à une situation finale où toutes les cartes sont situées dans leur tas initial, sauf les quatre `T` les plus bas de chaque tas qui se trouvent au sommet du tas n°1.

Exercice 2 : *Une transformation*

Voici une fonction écrite en PASCAL qui transforme un nombre entier en un autre.

```
function transforme(n : CARDINAL ) : CARDINAL;
var
   r,m : CARDINAL;
begin
   r := 0;
   m := n;
   while m<>0 do begin
      r := r*10 + (m mod 10);
      m := m div 10;
   end {while};
   transforme := r;
end {transforme};
```

On rappelle que les opérateurs **mod** et **div** calculent respectivement le reste et le quotient de la division euclidienne des entiers.

Q 1 .

Calculez `transforme(5103)` en présentant sous forme d'un tableau la valeur des variables `r` et `m` à la fin de chaque étape de la boucle.

Q 2 . Écrivez le corps d'un programme qui demande à l'utilisateur deux entiers positifs a et b, puis affiche à l'écran les valeurs de la fonction `transforme` pour tous les entiers de a à b, à raison d'une valeur par ligne. L'affichage d'une ligne se fera sous la forme

`5103 : 3015`

Vous n'oublierez pas de déclarer les variables nécessaires.

Exercice 3 : *Sur les chaînes*

Dans cet exercice on réalise une fonction `crochete` paramétrée par une chaîne de caractères `s` et dont le résultat est une chaîne de caractères, obtenue en entourant chacun des caractères de la chaîne `s` par des crochets. Par exemple `crochete('timoleon')` vaut `'[t][i][m][o][l][e][o][n]'`

Q 1 . En supposant déjà écrite la fonction `crochete`, expliquez comment utiliser cette fonction pour initialiser le tas 1 avec un nombre quelconque de trèfles surmontés d'un nombre quelconque de carreaux surmontés d'un nombre quelconque de coeurs, surmontés d'un nombre quelconque de piques ?

Q 2 . Réaliser une fonction nommée `encadre` paramétrée par un caractère `c` dont le résultat est une chaîne de trois caractères, composée d'un crochet ouvrant, du caractère `c` et d'un crochet fermant.

Q 3 . En utilisant la fonction `encadre` écrire la fonction `crochete`

Q 4 . Au vu des instructions qui suivent, déclarez les variables `s` `c`, `res` et `i`.

```
s:='[t][i][m][o]';
res:='';
for i:=1 to length(s) do
begin
  if (i mod 3)=2 then
  begin
    c:=s[i];
    res:=res+c;
  end {if};
end {for};
```

Q 5 . Précisez le contenu de chaque variable après l'exécution des instructions précédentes.

Q 6 . Implantez une fonction `decrochete` telle que pour toute chaîne `s` on ait `decrochete(crochete(s))=s`

Solutions

Exercice 1

Q 1 .
Solution

rappels :
[T] correspond à un nombre quelconque -éventuellement nul- de Trèfle,
T+K correspond à 1 Trèfle OU 1 Carreau,
TKT correspond à 1 Trèfle surmonté d'1 Carreau lui-même surmonté d'1 Trèfle,
[T]+[K] correspond à un nombre quelconque de Trèfle OU un nombre quelconque de Carreau,
[T][K][T] correspond à un nombre quelconque de Trèfle surmonté d'un nombre quelconque de Carreau surmonté d'un nombre quelconque de Trèfle
([T]+[K])T correspond à un nombre quelconque de Trèfle OU un nombre quelconque de Carreau, surmonté d'1 Trèfle

1. `initTas(1,'T[T]+[K]');` est incorrecte, on peut obtenir par exemple : `KKK`;

   on obtient 1 Trèfle surmonté d'un nombre quelconque de Trèfle OU un nombre quelconque de Carreau, le OU peut donc privilégier par exemple 3 Carreau

2. `initTas(1,'[T+K]T[T+K]');` est correcte ; on peut obtenir par exemple : `KKKTKK`;

   la présence du T au coeur de la chaîne assure la présence d'1 Trèfle entre un nombre quelconque de Trèfle OU de Carreau et un autre nombre quelconque de Trèfle OU de Carreau, par exemple 3 Carreau surmontés du T surmonté de 2 Carreau

3. `initTas(1,'(T+K)[T+K]');` est incorrecte, on peut obtenir par exemple : `KKK`.

   on obtient 1 Trèfle OU 1 Carreau, surmonté d'un nombre quelconque de Trèfle OU de Carreau, les deux OU peuvent donc ne privilégier que des Carreau, par exemple 1 Carreau surmonté de 2 Carreau

Q 2 .
Solution

l'instruction `initTas(1,'[T+K]T[T+K]');` permet de remplir le tas 1 avec un nombre quelconque de Trèfle OU de Carreau, puis avec 1 Trèfle, puis avec un nombre quelconque de Trèfle ou de Carreau

considérons par exemple la situation initiale suivante où l'on a un nombre quelconque de Carreau, ici 2 notés K1K2, puis le 1er Trèfle noté T0, puis une alternance de Carreau ou de Trèfle notés avec leur ordre d'empilement sur le tas 1 :
Situation Initiale :
Tas1= K1K2T0K3K4T1T2K5T3K6K7
on se propose d'arriver à la situation finale suivante où la carte T0 aura été enlevée du tas 1 et se trouve sur le tas 2 :
Sit. finale :
Tas 1= K1K2T1K3K4T2T3K5K6K7
Tas 2 = T0
Tas3 =

```
// chercher le 1er T
while not SommetTrefle(1) do begin
  deplacerSommet(1,3);
end {while};

// mettre le 1er T sur le tas 2
deplacerSommet(1,2);

// vider le reste du tas 1 sur le tas 3
// en gérant les éventuels T
while TasNonVide(1) do begin
  if SommetTrefle(1) then begin
    deplacerSommet(2,3);
    deplacerSommet(1,2);
  end else begin
    deplacerSommet(1,3);
  end {if};
end {while};
```

Après le 1er while la situation intermédiaire se traduit par
Tas 1= K1K2T0K3K4T1T2K5
Tas 2 = T3
Tas 3 = K7K6
Avec le 2ème while on poursuit ensuite le transfert des cartes restantes du tas 1 :
si on trouve un Carreau (K5) on le met sur le tas 3
si on trouve un Trèfle (T2), alors celui situé au sommet du tas 2 (T3) n'était pas le dernier, on le déplace du tas 2 vers le tas 3 et le nouveau Trèfle (T2) trouvé sur le tas 1 prend sa place sur le tas 2
On a donc l'enchaînement des situations suivantes -notez que l'ordre initial du tas 1 n'est pas rétabli- :
Tas 1= K1K2T0K3K4T1T2
Tas 2 = T3
Tas 3 = K7K6K5
Tas 1= K1K2T0K3K4T1
Tas 2 = T2
Tas3 = K7K6K5T3
Tas 1= K1K2T0K3K4
Tas 2 = T1
Tas3 = K7K6K5T3T2
Tas 1= K1K2T0K3
Tas 2 = T1
Tas3 = K7K6K5T3T2K4
Tas 1= K1K2T0
Tas 2 = T1
Tas3 = K7K6K5T3T2K4K3
Tas 1= K1K2
Tas 2 = T0
Tas3 = K7K6K5T3T2K4K3T1
Tas 1= K1
Tas 2 = T0
Tas3 = K7K6K5T3T2K4K3T1K2
Tas 1=
Tas 2 = T0
Tas3 = K7K6K5T3T2K4K3T1K2K1

Q 3 .
Solution

si le tas 3 est initialement vide il suffit de retransférer toutes les cartes qu'il contient pour le rendre vide à nouveau

```
while TasNonVide(3) do begin
  deplacerSommet(3,1);
end {while};
```

Q 4 .
Solution

si le tas 3 n'est pas initialement vide il faut retransférer uniquement les cartes qu'on y a transférées donc les compter

1. Un compteur est indispensable pour compter les cartes déplacées du tasn°1 vers le tas n°3.
2.

   la variable `cpt` est de type `cardinal` (entier positif ou nul),
   elle est initialisée à zéro
   et incrémentée à chaque transfert d'une carte du tas 1 vers le tas 3
   dans chacune des 2 instructions while
   puis une boucle **for** permet de reprendre sur le tas 3 le nombre exact des cartes à remettre sur le tas 1
   il faut déclarer l'indice de boucle `i` de type `cardinal`

```
var
   cpt,i : CARDINAL;

...
cpt := 0;
while not SommetTrefle(1) do begin
  deplacerSommet(1,3);
  cpt := cpt + 1;
end {while};
deplacerSommet(1,2);
while TasNonVide(1) do begin
  if SommetTrefle(1) then begin
    deplacerSommet(2,3);
    deplacerSommet(1,2);
  end else begin
    deplacerSommet(1,3);
  end {if};
  cpt := cpt + 1;
end {while};
for i := 1 to cpt do begin
  deplacerSommet(3,1);
end {for};
```

Q 5 .
Solution

la procédure demandée a 3 paramètres `i, j, k`, de type `TasPossibles`, de valeur différente
le compteur et l'indice de boucle sont des variables locales à cette procédure
dans la séquence des instructions précédentes on remplace 1 par `i`, 2 par `j` et 3 par `k`
attention à l'ordre des paramètres dans la déclaration

```
// deplacerDernierTrefle(i,j,k)
// CU : i ≠ j ≠ k, au moins un T sur le tas i
procedure deplacerDernierTrefle(i,j,k : TasPossibles);
var
  cpt,ind : CARDINAL;
begin
  cpt := 0;
  while not SommetTrefle(i) do begin
    deplacerSommet(i,k);
    cpt := cpt + 1;
  end {while};
  deplacerSommet(i,j);
  while TasNonVide(i) do begin
    if SommetTrefle(i) then begin
      deplacerSommet(j,k);
      deplacerSommet(i,j);
    end else begin
      deplacerSommet(i,k);
    end {if};
    cpt := cpt + 1;
  end {while};
  for ind := 1 to cpt do begin
    deplacerSommet(k,i);
  end {for};
end {deplacerDernierTrefle};
```

Q 6 .
Solution

notez l'importance de l'ordre des paramètres dans la déclaration et dans les appels suivants

```
deplacerDernierTrefle(1,2,3);
deplacerSommet(2,1);
deplacerDernierTrefle(2,1,3);
deplacerDernierTrefle(3,1,4);
deplacerDernierTrefle(4,1,3);
```

Exercice 2

Q 1 .
Solution

la fonction `transforme` utilise le reste de la division entière par 10 pour isoler chaque chiffre du nombre `n` passé en paramètre et renverse ainsi l'ordre initial des chiffres

notez l'obligation de déclarer des variables locales `m` et `r` et la dernière instruction qui attribue le résultat `r` au nom de la fonction

r	m	
0	5103	initialisation
3	510	début de la boucle
30	51	
301	5	
3015	0	fin de la boucle

`transforme(5103)`=3015

Q 2 .
Solution

on a besoin de 3 variables, 2 pour les bornes soit `a` et `b`, et un indice de boucle `n` qui varie entre ces bornes

la saisie au clavier des bornes `a` et `b` se fait avec la procédure **readln**, soit 2 appels précédés de l'affichage d'une consigne à l'utilisateur

l'affichage des résultats se fait avec la procédure **writeln**, avec 3 paramètres

l'appel de la fonction `transforme` renvoie une valeur, directement utilisée dans l'appel de **writeln**

```
var
  a,b,n : CARDINAL;
begin
  write('a = '); readln(a);
  write('b = '); readln(b);
  for n := a to b do begin
    writeln(n,' : ',transforme(n));
  end {for};
end.
```

Exercice 3

Q 1 .

Solution

il suffit de passer la chaîne `'TKCP'` en paramètre de la fonction `crochete`, le résultat obtenu, soit la valeur chaîne `'[T][K][C][P]'`, étant lui-même paramètre de la procédure `InitTas`

```
  InitTas(1,crochete('TKCP'));
```

Q 2 .

Solution

la fonction demandée a comme paramètre un caractère et renvoie un résultat sous forme d'une chaîne de 3 caractères

pour l'obtenir il faut donc concaténer un crochet ouvrant, le caractère passé en paramètre et le crochet fermant

```
  function encadre(const c:CHAR):STRING;
  begin
    encadre:='['+c+']';
  end {encadre};
```

Q 3 .

Solution

la fonction demandée a comme paramètre une chaîne de caractères `s` et renvoie un résultat sous forme d'une chaîne de 3 fois plus de caractères

pour l'obtenir il faut donc appliquer la fonction `encadre` à chaque caractère `s[i]` du paramètre `s`

la boucle **for** ci-dessous permet de balayer chaque caractère de la chaîne et de cumuler par concaténation les résultats successifs

il est nécessaire d'avoir une variable locale `res` pour construire la chaîne résultat, qu'il faut initialiser à chaîne vide

```
  function crochete(const s:STRING):STRING;
  var res:STRING;
      i  : CARDINAL;
  begin
```

```
    res:='';
    for i:=1 to length(s) do
    begin
      res:=res+encadre(s[i]);
    end {for};
    crochete:=res;
  end {crochete}
```

Q 4 .
Solution

`s` et `res` sont des variables associées à des chaînes de caractères, `i` est un indice de boucle positif et `c` est un caractère

la boucle a pour effet de ne garder que les caractères entre crochets situés aux rangs `i= 2, 5, 8, 11`, soit tels que `i mod 3 = 2`

notez que l'indice de boucle varie entre `1` et **`length(s)`** mais redevient indéterminé en sortie de boucle

```
var
  i:CARDINAL;
  s:STRING;
  res:STRING;
  c: CHAR;
```

Q 5 .
Solution

- `s` ne change pas. `s` vaut `'[t][i][m][o]'`.
- `c` vaut `'o'`
- `res` vaut `'timo'`
- `i` possède une valeur indéterminée (non définie par le langage).

Q 6 .
Solution

la fonction demandée a un paramètre de type chaîne de caractères, elle s'appliquera à une chaîne de longueur multiple de 3

et renvoie une chaîne de longueur égale au tiers de celle du paramètre, constituée des caractères du paramètre dont le rang i est tel que i mod $3 = 2$

on reprend donc la séquence d'instructions précédentes dans le corps de la fonction, avec `res` et `i` comme variables locales à la fonction et sans oublier d'attribuer le résultat au nom de la fonction

```
function decrochete(const s:STRING):STRING;
  var res:STRING;
      i  : CARDINAL;
begin
  res:='';
  for i:=1 to length(s) do
  begin
    if (i mod 3)=2 then
    begin
      c:=s[i];
      res:=res+c;
    end {if};
  end {for};
  decrochete:=res;
end {decrochete};
```