Autour de l'exercice 19

ew et lw

Table des matières

I/ Exercice 18

1. Énoncé du problème

- Situation initiale :
  Sur le tas 1 il y a un nombre quelconque de trèfles, et il y a au moins un trèfle. Les autres tas sont vides.
- Situation finale :
  Sur le tas 3, il y a un nombre quelconque de trèfles, sur le tas 2, il y a exactement une carte qui est le plus petit trèfle. Les autres tas sont vides.

2. Algorithme en français

```
1  déplacer la carte au sommet du tas 1 vers le tas 2
2  tant que le tas 1 n'est pas vide
3  faire
4    si la carte au sommet du tas 1 est supérieure à
5     celle au sommet du tas 2
6    alors
7      déplacer la carte au sommet du tas 1 vers le tas 3
8    sinon
9      déplacer la carte au sommet du tas 2 vers le tas 3
10     déplacer la carte au sommet du tas 1 vers le tas 2
11   fin_si
12 fin_tant_que
```

3. Code correspondant en *Pascal*

Listing n°1.

```
1   ...
2   deplacerSommet(1,2);
3   while TasNonVide(1) do
4   begin
5     if superieur(1,2)
6     then
7       begin
8         deplacerSommet(1,3);
9       end
10    else
11      begin
12        deplacerSommet(2,3);
13        deplacerSommet(1,2);
14      end; //if
15  end; //while
```

II/ Vers une première solution de l'exercice 19

1. Énoncé du problème

- Situation initiale :
  Sur le tas 1 il y a un nombre quelconque de trèfles. Les autres tas sont vides.
- Situation finale :

Sur le tas 2, il y a un nombre quelconque de trèfles rangés dans l'autre croissant. (c'est-à-dire les plus petits en dessous les plus grands au dessus). Les autres tas sont vides.

2. Algorithme en français

```
1  Tant_que le tas 1 n'est pas vide
2  faire
3    réaliser le travail de l'execice 18.
4    tant_que le tas 3 n'est pas vide
5    faire
6      deplacer la carte au sommet du tas 3 sur le tas 1.
7    fin_tant_que
8  fin_tant_que
```

3. Code correspondant en *Pascal*

Listing n°2.

```
1   ...
2   while TasNonVide(1) do
3   begin
4     deplacerSommet(1,2);
5     while TasNonVide(1) do
6     begin
7       if superieur(1,2)
8       then
9         begin
10          deplacerSommet(1,3);
11        end
12      else
13        begin
14          deplacerSommet(2,3);
15          deplacerSommet(1,2);
16        end; //if
17    end; //while
18    while TasNonVide(3) do
19    begin
20      deplacerSommet(3,1);
21    end; //while
22  end; //while
```

III/ Procédons à quelques améliorations

1. Constats

On aurait pu remplacer boucle **`while`** commençant en ligne 17 par un appel à la procédure `viderTas`. À chaque étape, on déplace toutes les cartes vers le tas 3 puis on les remet sur le tas 1, on aurait pu faire mieux en profitant de ce déplacement pour mettre la plus petite carte du tas 3 au sommet du tas 2. On peut copier coller, et modifier le travail précédent mais on risque d'oublier de faire une modification. Il est préférable réaliser une procédure.

2. Fabrication d'une procédure

Pour réaliser une procédure, il faut connaître quel sera l'effet de la procédure, et dans quelles conditions on peut l'utiliser, il faut lui trouver un nom, et dé-

terminer quels seront ses paramètres, et leurs types respectifs, et aussi trouver un nom pour chacun des paramètres.

Listing n°3.

```
procedure trouver_la_plus_petite_carte( TasDepart,
                                        TasMini,
                                        TasAutre:TASPOSSIBLES);
begin
  deplacerSommet(TasDepart,TasMini);
  while TasNonVide(TasDepart) do
  begin
    if superieur(TasDepart,TasMini)
    then
      begin
        deplacerSommet(TasDepart,TasAutre);
      end
    else
      begin
        deplacerSommet(TasMini,TasAutre);
        deplacerSommet(TasDepart,TasMini);
      end ; //if
  end; //while
end; //trouver_la_plus_petite_carte
```

3. Code modifié et erroné

Maintenant que la procédure est réalisée, il ne reste plus qu'à l'utiliser.

Listing n°4.

```
...
BEGIN
  initTas(1,'[T]');
  while TasNonVide(1) do
  begin
    trouver_la_plus_petite_carte(1,2,3);
    trouver_la_plus_petite_carte(3,2,1);
  end ; //while
END.
```

Si on teste ce programme, deux cas peuvent se présenter :

- soit le nombre de trèfle et l'exécution se déroule correctement
- soit il est impair et une exception `Tas_Vide` se déclenche.

En tout cas cela permet d'attirer l'attention sur la difficulté de déterminer dans quel cas particulier tester le programme. Si on n'a pas pensé que la parité du nombre de trèfles pose ici un problème, il est difficile de penser à faire le test dans ces cas particuliers là.

Si on y a pensé, on peut remplacer l'instruction d'initialisation par le cas particulier :

Listing n°5.

```
  initTas(1,'[TT]T'); // un nombre impair de trèfles
```

et patatras l'exception `Tas_Vide` ne manquera pas de se déclencher.

4. De l'intérêt de la spécification détaillée d'une procédure

Le problème précédent provient du fait qu'on a utilisé la procédure `trouver_plus_petite_carte` sans respecter ses contraintes d'utilisation. En

effet l'utilisation de cette procédure nécessite que le tas de départ ne soit pas vide. Cette contrainte d'utilisation aurait dû être signalée en commentaire !

En effet lorsqu'on écrit une procédure, on doit[1] préciser ses paramètres, son effet, ses contraintes d'utilisation, des exemples de comportements attendus. Cela donnerait ici :

Listing n°6.

```
1  // trouver_la_plus_petite_carte
2  // parametres:
3  //   TasDepart        : TASPOSSIBLES
4  //   TasMini          : TASPOSSIBLES
5  //   TasAutres        : TASPOSSIBLES
6  // effet:
7  //  trouve la plus petite carte du tas TasDepart,
8  //  pose cette carte au sommet du tas TasMini
9  //  pose toutes les autres cartes du tas TasDepart
10 //  sur le TasAutresCartes.
11 //  après l'appel de la procédure le tas TasDepart est vide.
12 // contrainte d'utilisation:
13 //  le tas TasDepart contient au moins une carte
14 // exemple de comportement attendu:
15 // avant
16 //  Tas 1: '6T ; RT; 4T ; VT'
17 //  Tas 2: ''
18 //  Tas 3: ''
19 // trouver_la_plus_petite_carte(1,2,3);
20 // après
21 //  Tas 1: ''
22 //  Tas 2: '4T'
23 //  Tas 3: 'VT ; RT ; 6T'
24 procedure trouver_la_plus_petite_carte( TasDepart,
25                                          TasMini,
26                                          TasAutre:TASPOSSIBLES);
27 begin
28   deplacerSommet(TasDepart,TasMini);
29   while TasNonVide(TasDepart) do
30   begin
31     if superieur(TasDepart,TasMini)
32     then
33       begin
34         deplacerSommet(TasDepart,TasAutre)
35       end
36     else
37       begin
38         deplacerSommet(TasMini,TasAutre);
39         deplacerSommet(TasDepart,TasMini);
40       end ; //if
41   end; //while
42 end; //trouver_la_plus_petite_carte
```

On peut remarquer que les lignes de 1 à 6 font double emploi avec l'entête de la procédure qui se trouve en ligne 24 à 26. On peut donc les omettre si on le souhaite.

5. Pose d'une rustine

Il suffit de tester la non vacuité du tas 3 avant de procéder à l'appel de la procédure `trouver_la_plus_petite_carte`

1 dans le sens : « *on est moralement obligé de* »

6. Code modifié

Listing n°7.

```
...
BEGIN
  initTas(1,'[T]');
  while TasNonVide(TasCourant) do
  begin
    trouver_la_plus_petite_carte(1,2,3);
    if tasNonVide(3)
    then
     begin
       trouver_la_plus_petite_carte(3,2,1);
     end ; //if
  end ; //while
END.
```

IV/ Variantes aux variables

1. Constat

Dans le code précédent, on a écrit deux appels à la procédure `trouver_la_plus_petite_carte`. Lorsqu'on écrit deux fois des choses analogues, il devient intéressant de se demander si on ne peut pas factoriser le code. On peut souhaiter supprimer l'instruction conditionnelle à l'intérieur de la boucle.

2. Idée

On pourrait utiliser une variable nommée `TasCourant`, dont le contenu nous permettrait de connaître où se trouve les cartes qui restent à trier.

3. Code modifié

Listing n°8.

```
...
var TasCourant,TasAutre: TASPOSSIBLES;
BEGIN
  initTas(1,'[T]');
  TasCourant:=1;
  TasAutre:=3;
  while TasNonVide(TasCourant) do
  begin
    trouver_la_plus_petite_carte(TasCourant,2,TasAutre);
    if TasCourant=1
    then
     begin
       TasCourant:=3;
       TasAutre:=1;
     end
    else
     begin
       TasCourant:=1;
       TasAutre:=3;
     end ; //if
  end ; //while
END.
```

4. Amélioration discutable

On souhaitait supprimer l'instruction conditionnelle, et on en a mis une autre à la place... D'autre part pour la factorisation de code, on a des lignes analogues qui se répètent. Mais cette version présente un intérêt, celui d'avoir remplacé les deux appels à `trouver_la_plus_petite_carte` par un appel un seul avec des paramètres effectifs variables. Elle est préparatoire à la version suivante beaucoup plus astucieuse.

V/ Une version astucieuse

1. Une remarque

Voici un fait très utile :

$$4-1 = 3$$
$$4-3 = 1$$

On peut en effet utiliser cette remarque de la manière suivante si `TasCourant` vaut respectivement 1 ou 3, alors l'expression `4-TasCourant` vaut respectivement 3 ou 1.

2. Code modifié

Listing n°9.

```
1  ...
2  var TasCourant: TASPOSSIBLES;
3  BEGIN
4    initTas(1,'[T]');
5    TasCourant:=1;
6    while TasNonVide(TasCourant) do
7    begin
8      trouver_la_plus_petite_carte(TasCourant,2,4-TasCourant);
9      TasCourant:=4-TasCourant;
10   end ; //while
11 END.
```

On remarque qu'en ligne 8, dans l'appel de procédure, le troisième pamètre effectif est une expression.[2]

VI/ Vers l'exercice 22

Pour aller vers l'exercice 22, il reste du travail, d'abord, comme tous les tas sont utilisés, il va falloir distinguer les couleurs des cartes. On aura besoin de la procédure `ViderCouleur`, et il faudra modifier la procédure `trouver_la_plus_petite_carte` pour lui ajouter un paramètre relatif à la couleur, en plus il pourrait être intéressant de transformer le travail de l'exercice 19 en une procédure. Par ailleurs, on peut créer une fonction `autre` qui permettra de calculer le numéro du tas où l'on va poser les cartes qu'on n'a pas encore triées. Pour sa réalisation, on va utiliser une fonction `suivant` qui permet de calculer le numéro du tas suivant, en imposant que le tas 1 est celui qui suit le tas 4.

2 On verra au second semestre que dans certains cas, on ne peut pas faire cela.

1. la fonction suivant

Listing n°10.

```
1  ...
2  // fonction suivant
3  // paramètre :
4  // tas : TASPOSSIBLES
5  // résultat : TASPOSSIBLES
6  // suivant(1) vaut 2
7  // suivant(2) vaut 3
8  // suivant(3) vaut 4
9  // suivant(4) vaut 1
10 // contrainte d'utilisation:
11 // aucune
12   function suivant(Tas: TASPOSSIBLES):TASPOSSIBLES;
13   begin
14     if Tas=4
15     then
16       begin
17         suivant:=1;
18       end
19     else
20       begin
21         suivant:=Tas+1;
22       end;
23   end; //suivant
```

Il est possible d'écrire une version de `suivant` qui n'utilise pas d'instruction conditionnelle. Il suffit de remarquer que l'expression `(Tas mod 4)+1` vaut ce qu'il faut.

2. la fonction autre

On veut non seulement utiliser le couple de tas (1,3), mais aussi pouvoir utiliser le couple de tas (2,4)

Listing n°11.

```
1  ...
2  // fonction autre
3  // paramètre :
4  // tas : TASPOSSIBLES
5  // résultat : TASPOSSIBLES
6  // autre(1) vaut 3
7  // autre(2) vaut 4
8  // autre(3) vaut 1
9  // autre(4) vaut 2
10 // contrainte d'utilisation:
11 // aucune
12   function Autre(Tas: TASPOSSIBLES):TASPOSSIBLES;
13   begin
14     Autre:=suivant(suivant(Tas));
15   end; //Autre
```

La fonction `autre` utilise la fonction `suivant`, il faut donc que la déclaration de la fonction `suivant` se trouve avant celle de la fonction `autre`.

3. modification de la procédure trouver_la_plus_petite_carte.

Listing n°12.

```
// trouver_la_plus_petite_carte
// parametres:
//   TasDepart        : TASPOSSIBLES
//   TasMini          : TASPOSSIBLES
//   TasAutres        : TASPOSSIBLES
//   c                : COULEURS
// effet:
//  trouve la plus petite carte parmi les cartes de
//  couleur c situées au sommet du TasDepart,
//  pose cette carte au sommet du tas TasMini
//  pose toutes les autres cartes du tas TasDepart
//  sur le TasAutresCartes.
//  après l'appel de la procédure le tas TasDepart est vide,
//  ou bien la carte au sommet n'est pas de couleur c
// contrainte d'utilisation:
//  le tas TasDepart contient au moins une carte
//  le tasAutres est vide
// exemple de comportement attendu:
// avant
//  Tas 1: 'RK ; 6T ; RT; 4T ; VT'
//  Tas 2: ''
//  Tas 3: '7P'
// trouver_la_plus_petite_carte(1,2,3,Trefle);
// après
//  Tas 1: 'RK'
//  Tas 2: '4T'
//  Tas 3: '7P ; VT ; RT ; 6T'
procedure trouver_la_plus_petite_carte( TasDepart,
                                        TasMini,
                                        TasAutre:TASPOSSIBLES ;
                                        c: COULEURS);
begin
  deplacerSommet(TasDepart,TasMini);
  while TasNonVide(TasDepart) and (couleurSommet(TasDepart)=c) do
  begin
    if superieur(TasDepart,TasMini)
    then
      begin
        deplacerSommet(TasDepart,TasAutre);
      end
    else
      begin
        deplacerSommet(TasMini,TasAutre);
        deplacerSommet(TasDepart,TasMini);
      end ; //if
  end; //while
end; //trouver_la_plus_petite_carte
```

4. Adaptation du travail de l'exercice 19 en procédure.

Il faut choisir le nom et les paramètres. Comme on devra trié des tas d'une couleur donnée, on peut décider de nommer la procédure `trier_couleur`, et de la paramétrer par le numéro du tas et la couleur.

Listing n°13.

```
...
// procedure trier_couleur
// paramètres
// tasDepart : TASPOSSIBLES
// c : COULEURS;
// effet
// trie les cartes de couleur c située au sommet du tasDepart
// contrainte d'utilisation
// les autres tas ne doivent pas avoir au sommet de cartes de couleur c
procedure trier_couleur(TasDepart:TASPOSSIBLES;
                        c:COULEURS);
var TasCourant:TASPOSSIBLES;
begin
    TasCourant:=suivant(TasDepart);
    viderCouleur(TasDepart,TasCourant,c);
  while TasNonVide(TasCourant) and (couleurSommet(TasCourant)=c) do
begin
    trouver_la_plus_petite_carte(TasCourant,TasDepart,Autre(TasCourant),c);
    TasCourant:=Autre(TasCourant);
end ; //while
end ; //trierCouleur
```

5. une solution de l'exercice 22.

Il ne reste plus grand chose à faire. le programme principale de la solution de l'exercice 22, va tenir en 10 lignes

Listing n°14.

```
BEGIN
  initTas(1,'[T]');
  initTas(2,'[K]');
  initTas(3,'[C]');
  initTas(4,'[P]');
  trier_couleur(1,TREFLE);
  trier_couleur(2,CARREAU);
  trier_couleur(3,COEUR);
  trier_couleur(4,PIQUE);
END.
```

On peut légèrement améliorer le programme en utilisant deux boucles **for**, pour factoriser les travaux d'initialisations et de déplacements de cartes. On peut d'ailleurs constater que le second paramètre de `trier_couleur` est un peu superflu. En effet, si le tas est vide, il n'y a rien à faire, sinon la carte au sommet du tas peut servir pour déterminer la couleur.

6. des suggestions pour aller plus loin.

- Situation initiale :
  Sur les quatre tas, il y a un nombre quelconque de cartes quelconques.
- Situation finale :
  Les quatre tas sont triés dans l'ordre croissant, et leur contenu est le même que dans la situation initiale, à l'ordre des cartes près.

Dans cet exercice, il n'est plus question de se fier à la couleur, seul le nombre de cartes peut être utile.