Itération, boucles Tant Que

1 Itération

1.1 Un problème

- État initial : Tas 1 un nombre quelconque de cartes, les autres tas vides.
- État final : tas 1, 3 et 4 vides, le tas 2 contient toutes les cartes du tas 1

1.2 Algorithme

La résolution du problème nécessite de déplacer (une à une) toutes les cartes du tas 1 vers le tas 2. Ce qu'on peut décrire par

```
tant que le tas 1 n'est pas vide faire
  déplacer la carte au sommet du tas 1 vers le tas 2
fin tant que
```

1.3 Sortie d'une boucle `tant que`

Une boucle **`tant que`** se termine lorsque sa condition n'est plus satisfaite. Par conséquent à la sortie de la boucle, on est assuré que la condition est fausse

```
  tant que <condition> faire
     actions
  fin tant que
  {<condition> est fausse}
```

1.4 La boucle Tant Que en Pascal

```
while <condition> do
  (* bloc d'instruction*)
```

Exemple traduction en Pascal de l'algorithme décrit à la section 1.2.

```
while TasNonVide(1) do
begin
  DeplacerSommet(1,2);
end {while}
```

1.5 Méthodologie

La conception d'une boucle **`tant que`** nécessite plusieurs points d'attention :

- la situation avant d'entrer dans la boucle (pré-condition) est–elle celle souhaitée ?
- la situation à la sortie de la boucle (post-condition) est–elle bien celle souhaitée ?
- la boucle ne risque–t–elle pas d'être infinie (problème de l'arrêt) ?

Exemple de boucle infinie

```
  while TasNonVide(1) do begin
    DeplacerSommet(1,2);
    DeplacerSommet(2,1);
  end {while};
```

Exemple de pré-condition pouvant être non satisfaite

```
while SommetTrefle(1) do begin
  DeplacerSommet(1,2);
end {while};
```

2 Exercices

Exercice 1. Décrivez les relations satisfaites entre a, b et c à l'issue des boucles qui suivent

1. tant que $a = b$ faire
   actions
   fin tant que

   tant que $a \leq b$ faire
   actions
   fin tant que

3. tant que $(a \neq b)$ ou $(a > c)$ faire
   actions
   fin tant que

Exercice 2. Les exercices de manipulation de cartes.