Initiation à la Programmation

http://www.fil.univ-lille1.fr/licence

Itération

Récapitulatif

Initiation à la programmation impérative (Pascal)

- ✓ Support : robot manipulateur de cartes
  (4 tas, instructions de déplacement et d'observation des cartes et des tas)
- ✓ Programme: suite d'instructions pour passer d'une SI à une SF
  - Analyse du problème (SI, SF, Algorithme)
  - Codage en Pascal (structure d'un programme)
  - Tests (TP)
- ✓ Instruction conditionnelle
  - Expressions booléennes (`true`, `false`, `and`, `or`, `not`)
  - `if...then...else...`, `if...then...` (alternative)

Aujourd'hui

- ✓ Retour sur l'instruction conditionnelle
- ✓ Les actions répétées sous condition

```
if Condition
then
    begin
        Instructions Alors
    end
else
    begin
        Instructions Sinon
    end;
Suite instructions
```

} N'est exécuté que si la *Condition* est vraie (`true`)

} N'est exécuté que si la *Condition* est fausse (`false`)

} est toujours exécuté quelque soit la valeur de *Condition*

```
if Condition
then
    begin
        Instructions Alors
    end;
Suite instructions
```

} N'est exécuté que si la *Condition* est vraie (`true`)

} est toujours exécuté quelque soit la valeur de *Condition*

2 exercices

```
if not(sommetCARREAU(1))
then
    begin
        deplacerSommet(1,2);
    end;
else
    begin
        deplacerSommet(1,3);
    end;
```

Écrire un algorithme équivalent qui n'utilise pas le connecteur not

```
if sommetCARREAU(1) and sommetCOEUR(2)
then
    begin
        deplacerSommet(1,2);
    end;
```

Écrire un algorithme équivalent qui n'utilise pas le connecteur and

Encore un exercice

```
if sommetTREFLE(1)
then
    begin
        deplacerSommet(1,2);
    end;

if sommetPIQUE(1)
    begin
        deplacerSommet(1,3);
    end;
```

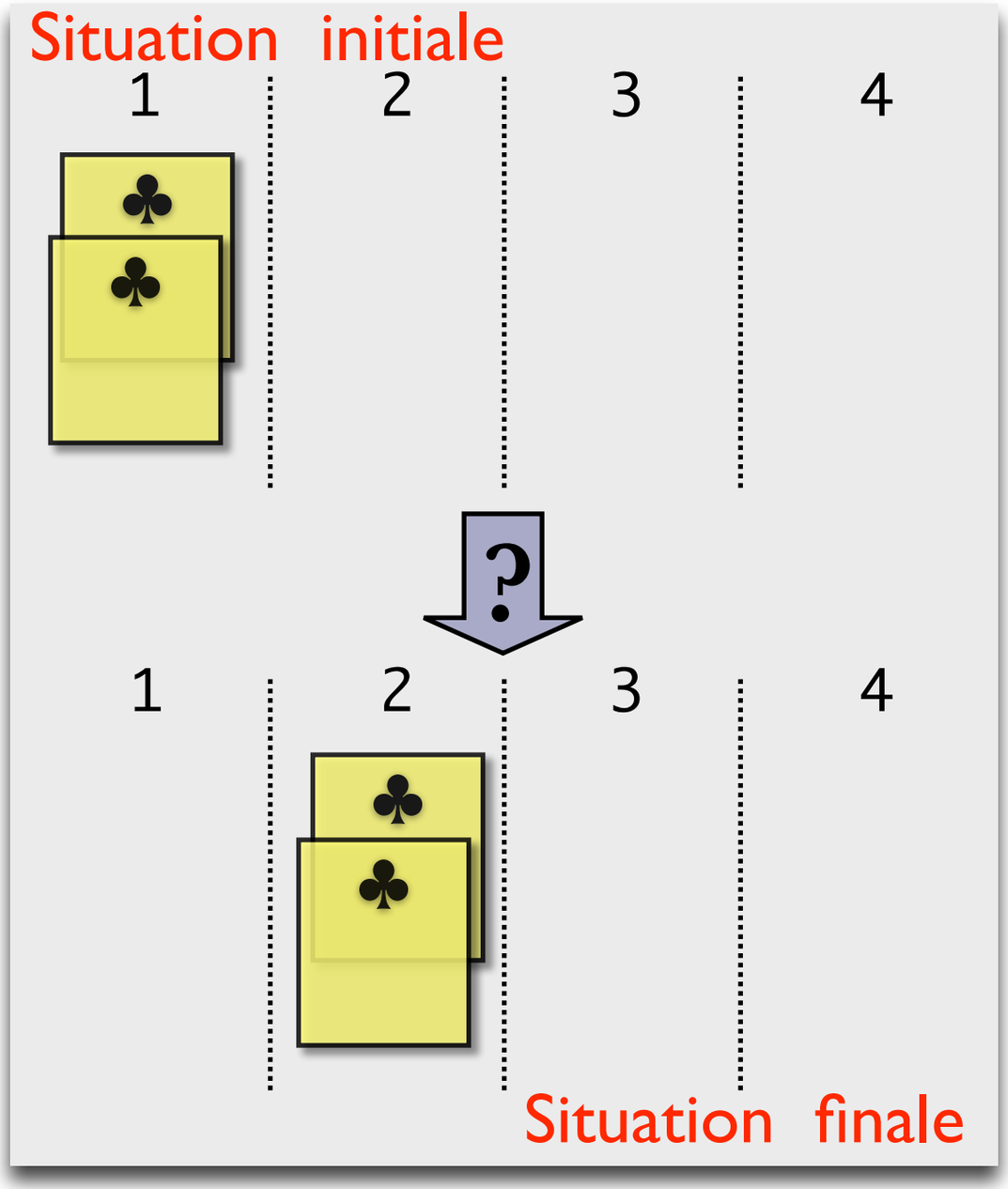
SI: tas 1:'T+P', autres tas vides
SF: tas 2:'[T]', tas 3: '[P]', autres tas vides

Pourquoi l'algorithme ci-contre est faux pour résoudre ce problème.

Retour sur le tout premier problème

Rappel: on ne peut déplacer qu'une seule carte à la fois

SI: tas 1:'TT', autres tas vides
SF: tas 2:'TT', autres tas vides



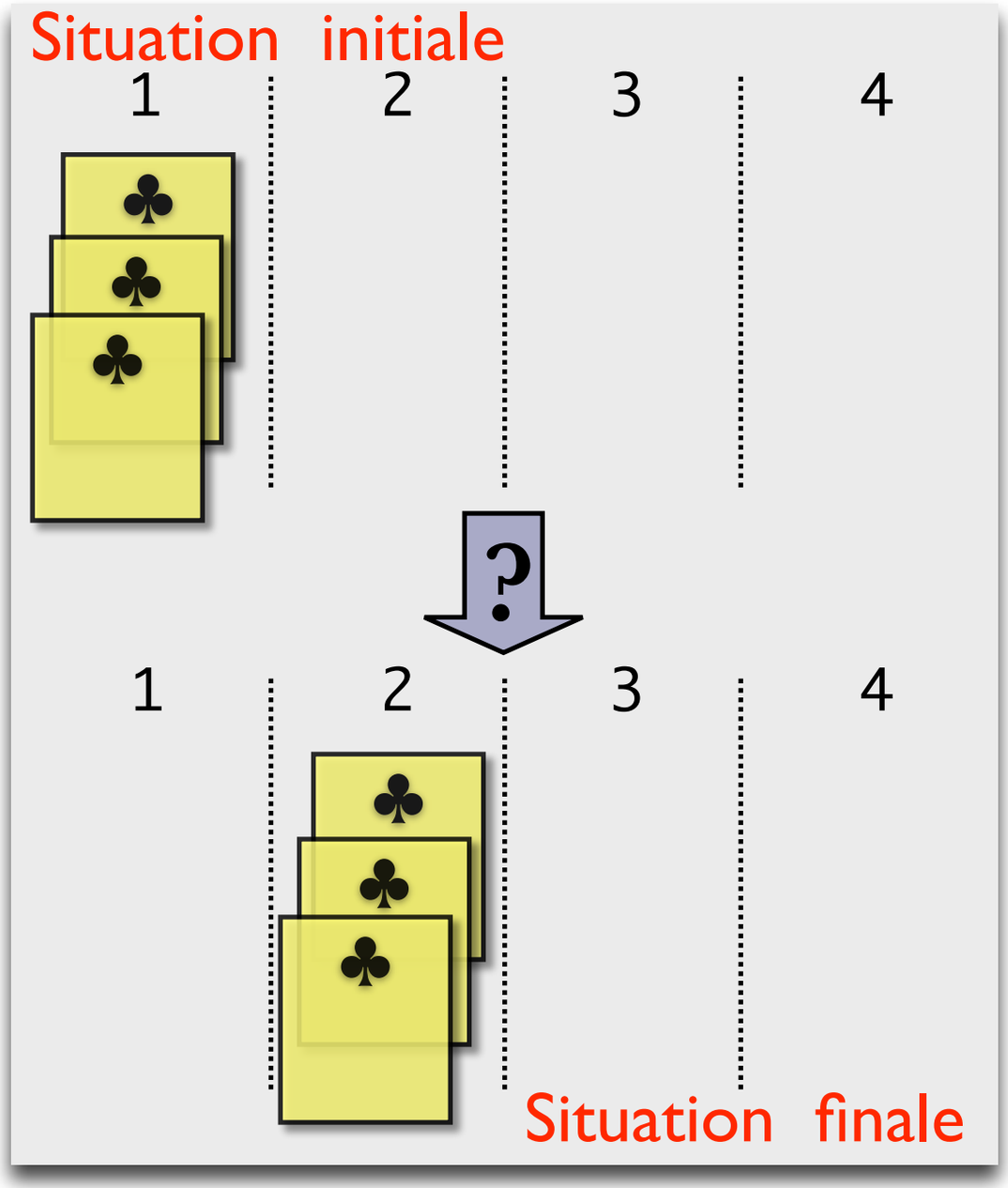
Algorithme:

```
deplacerSommet(1,2);
deplacerSommet(1,2);
```

Retour sur le tout premier problème

Rappel: on ne peut déplacer qu'une seule carte à la fois

SI: tas 1:'TTT', autres tas vides
SF: tas 2:'TTT', autres tas vides



Algorithme:

```
deplacerSommet(1,2);
deplacerSommet(1,2);
deplacerSommet(1,2);
```

Retour sur le tout premier problème

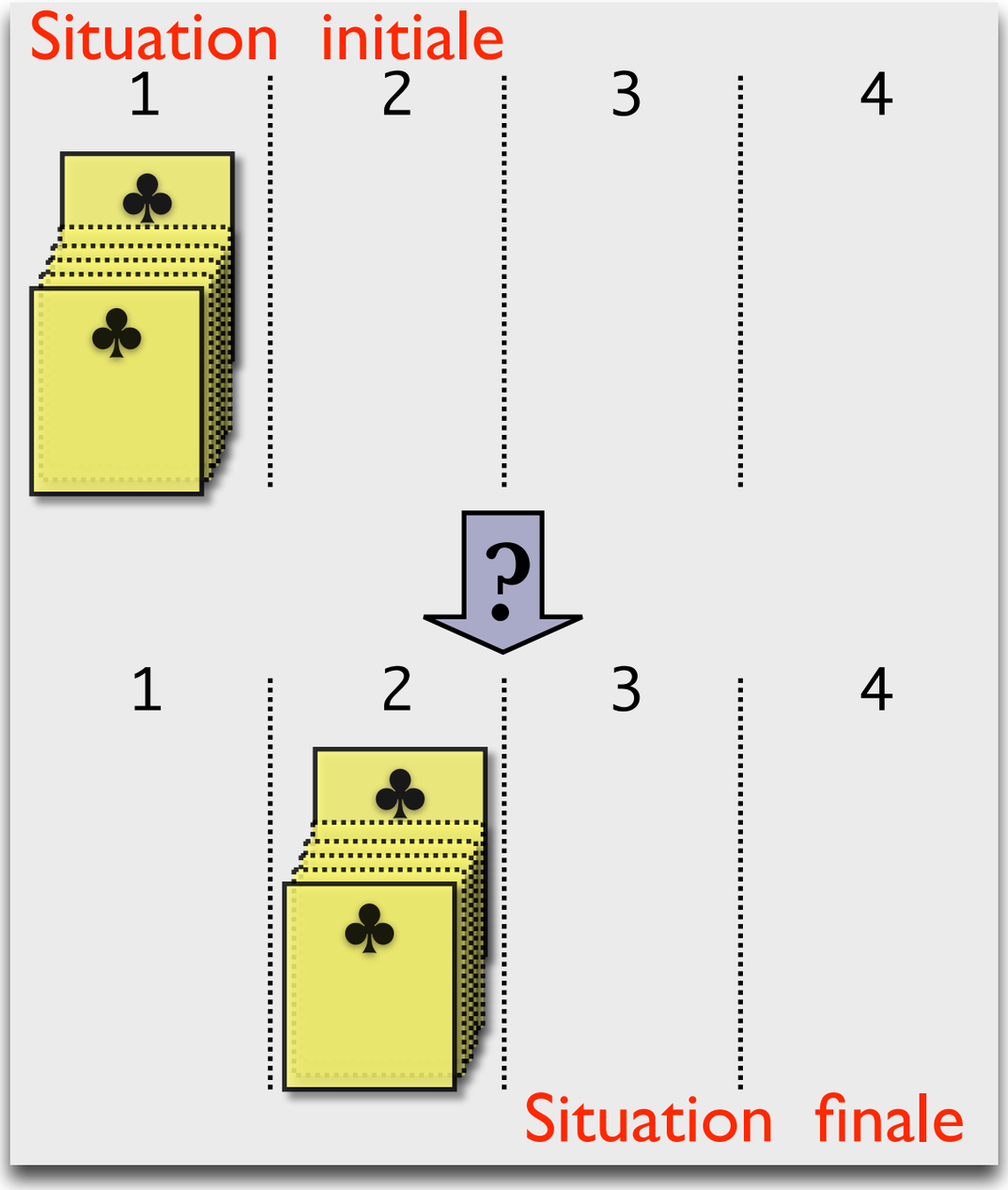
Rappel: on ne peut déplacer qu'une seule carte à la fois

SI: tas 1:'T[100]', autres tas vides
SF: tas 2:'T[100]', autres tas vides

Algorithme:

```
deplacerSommet(1,2);
deplacerSommet(1,2);
...
deplacerSommet(1,2);
```

} 100 x



Retour sur le tout premier problème

- ✓ On ne veut pas écrire 100 fois la même chose
- ✓ On souhaiterait écrire une fois l'instruction et donner le nombre de fois qu'il faut l'exécuter
- ✓ *Exemple: déplacer 100 fois la carte au sommet du tas 1 sur le tas 2*

Pourrait-on utiliser cette méthode pour résoudre le problème suivant ?

SI: tas 1 : '[T]', autres tas vides
SF: tas 2 : '[T]', autres tas vides

Retour sur le tout premier problème

SI: tas 1 : '[T]', autres tas vides
SF: tas 2 : '[T]', autres tas vides

Dans ce cas on ne connait pas le nombre (n) de cartes intialement présentes sur le tas 1.

À priori: déplacer les cartes une par une du tas 1 vers le tas 2
Problème: quelle est la condition qui détermine s'il faut effectivement réaliser un tel déplacement ?

Retour sur le tout premier problème

SI: tas 1 : '[T]', autres tas vides
SF: tas 2 : '[T]', autres tas vides

Dans ce cas on ne connait pas le nombre (n) de cartes intialement présentes sur le tas 1.

À priori: déplacer les cartes une par une du tas 1 vers le tas 2
Problème: quelle est la condition qui détermine s'il faut effectivement réaliser un tel déplacement ?

En particulier:
1. Quand le tas 1 est vide, ne rien faire! (on est déjà dans la situation finale),
2. Quand le tas 1 n'est pas vide, réaliser un déplacement.

Retour sur le tout premier problème

SI: tas 1 : '[T]', autres tas vides
SF: tas 2 : '[T]', autres tas vides

1. Quand le tas 1 est vide, ne rien faire! (on est déjà dans la situation finale),
2. Quand le tas 1 n'est pas vide, réaliser un déplacement.

On a trouvé une expression booléenne (tas 1 non vide) qui détermine à chaque pas (*i.e.* à chaque situation intermédiaire) la tâche suivante à réaliser (nouveau déplacement ou ne plus rien faire).

tant que ...

SI: tas 1 : '[T]', autres tas vides
SF: tas 2 : '[T]', autres tas vides

Algorithme:

```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```

Syntaxe et sémantique

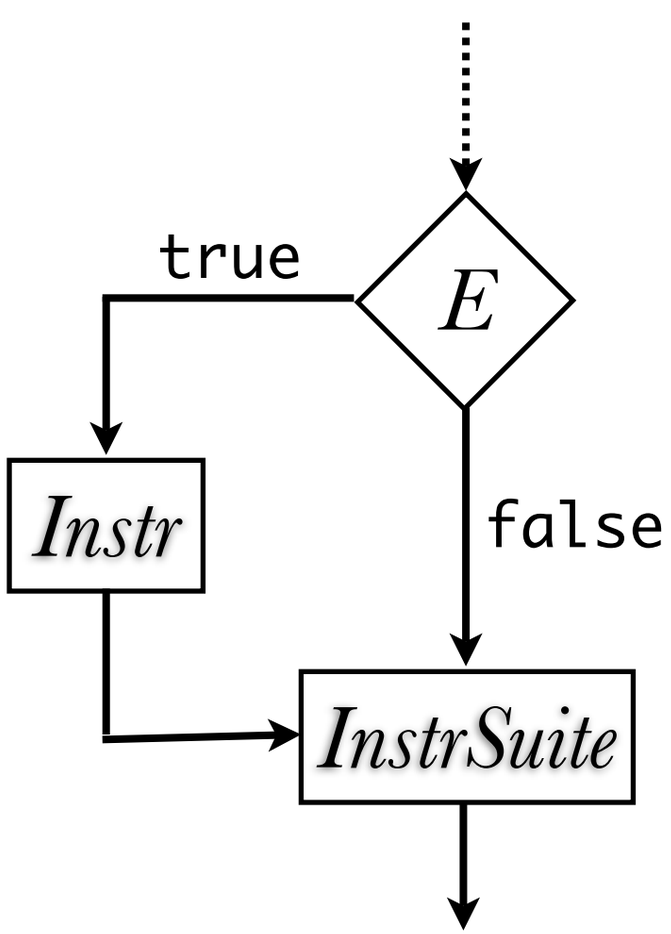
```
while Condition do
begin
    Instructions Tant que
end;
Instructions suivantes
```

} Exécutées que si *Condition* est vrai et tant que celle-ci reste vraie

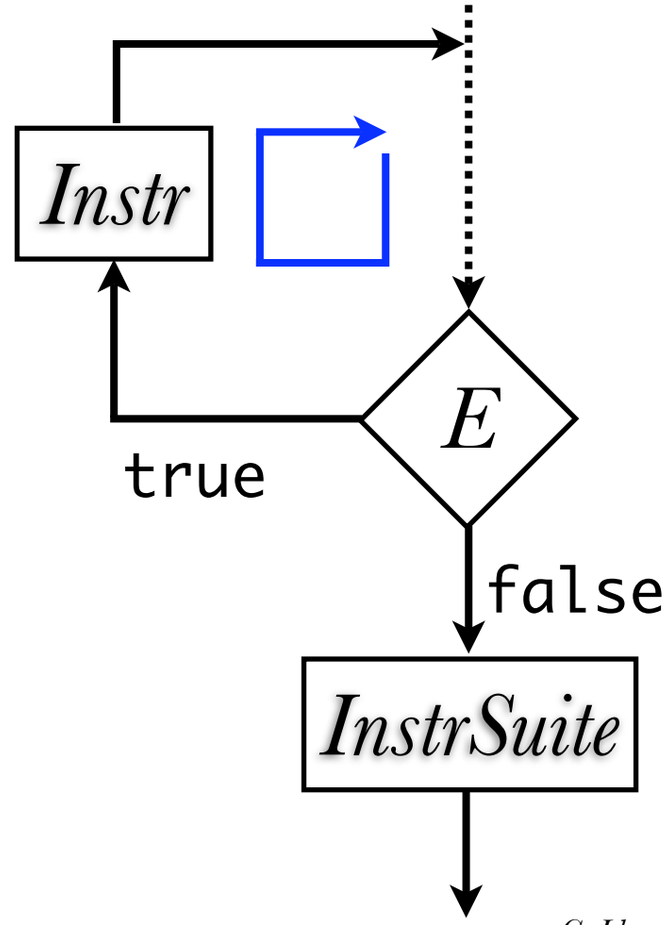
} Exécutées dès que *Condition* est fausse

“si” vs. “tant que”

```
if E then
begin
    Instr
end;
InstrSuite
```

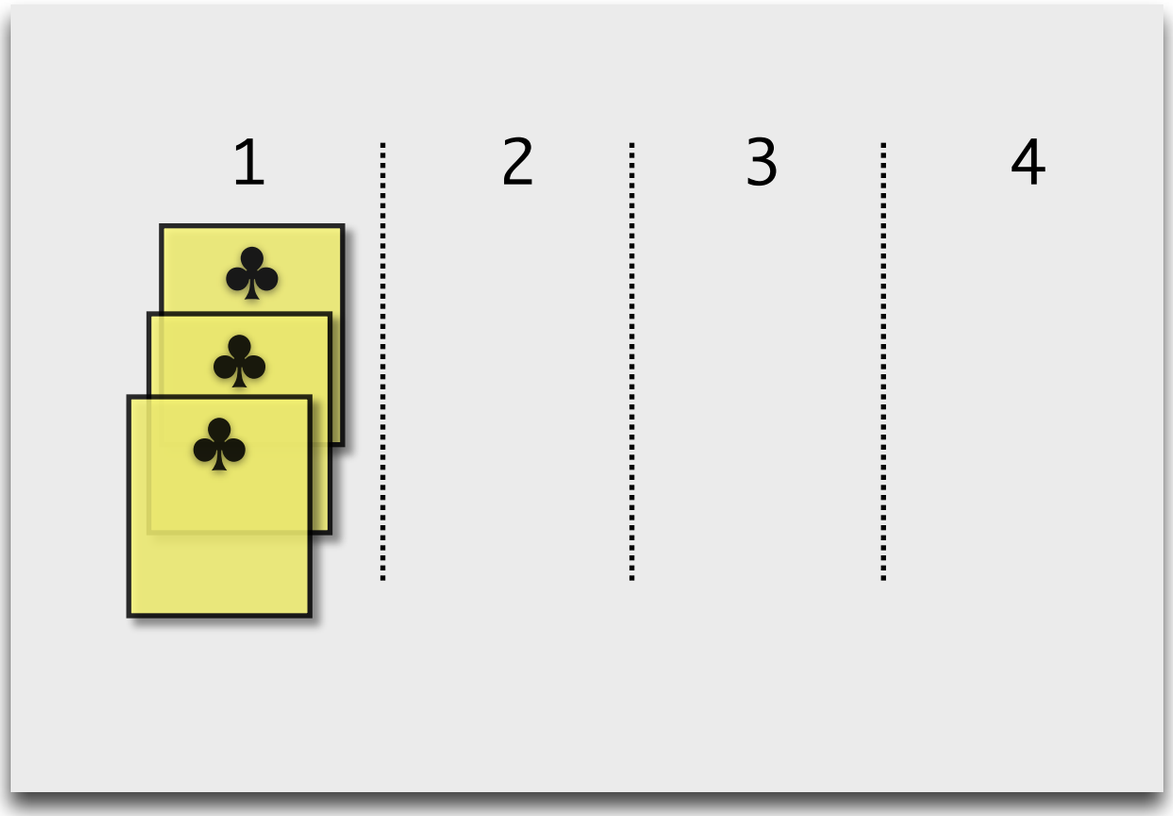



```
while E do
begin
    Instr
end;
InstrSuite
```



Exemple

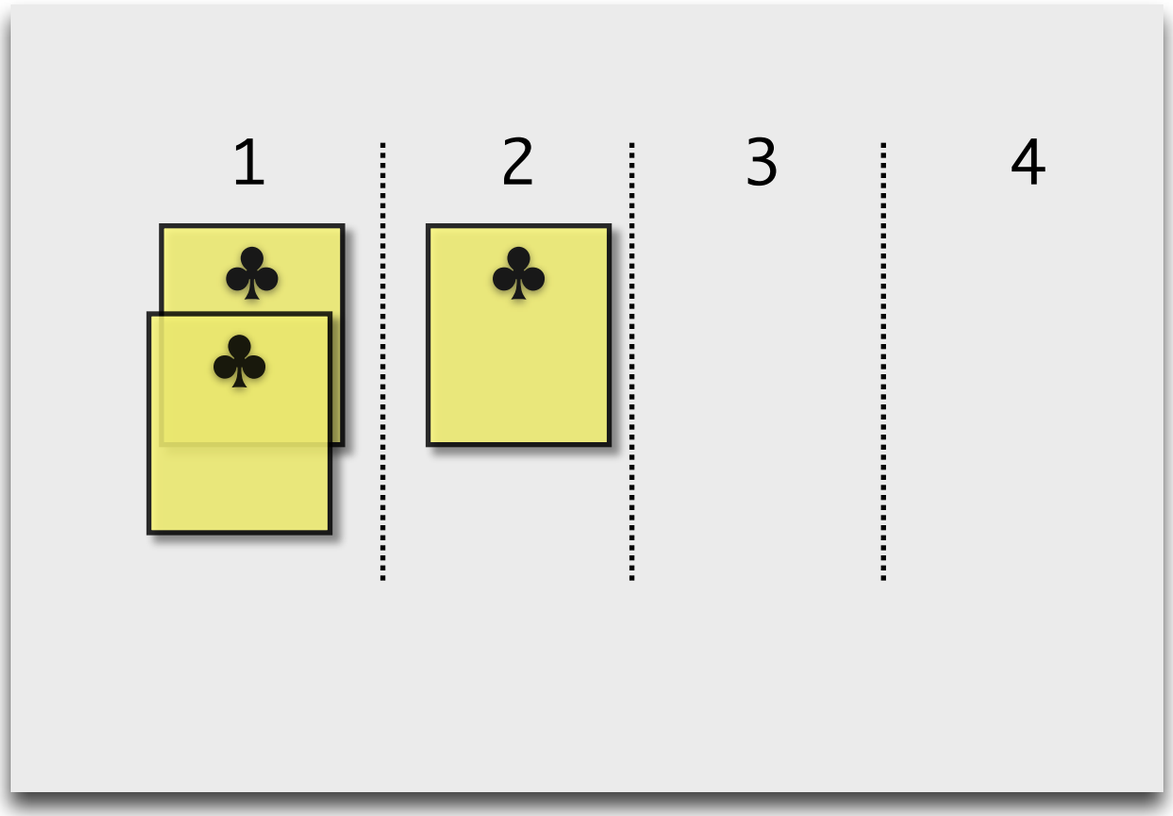
```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



TasNonVide(1) = true

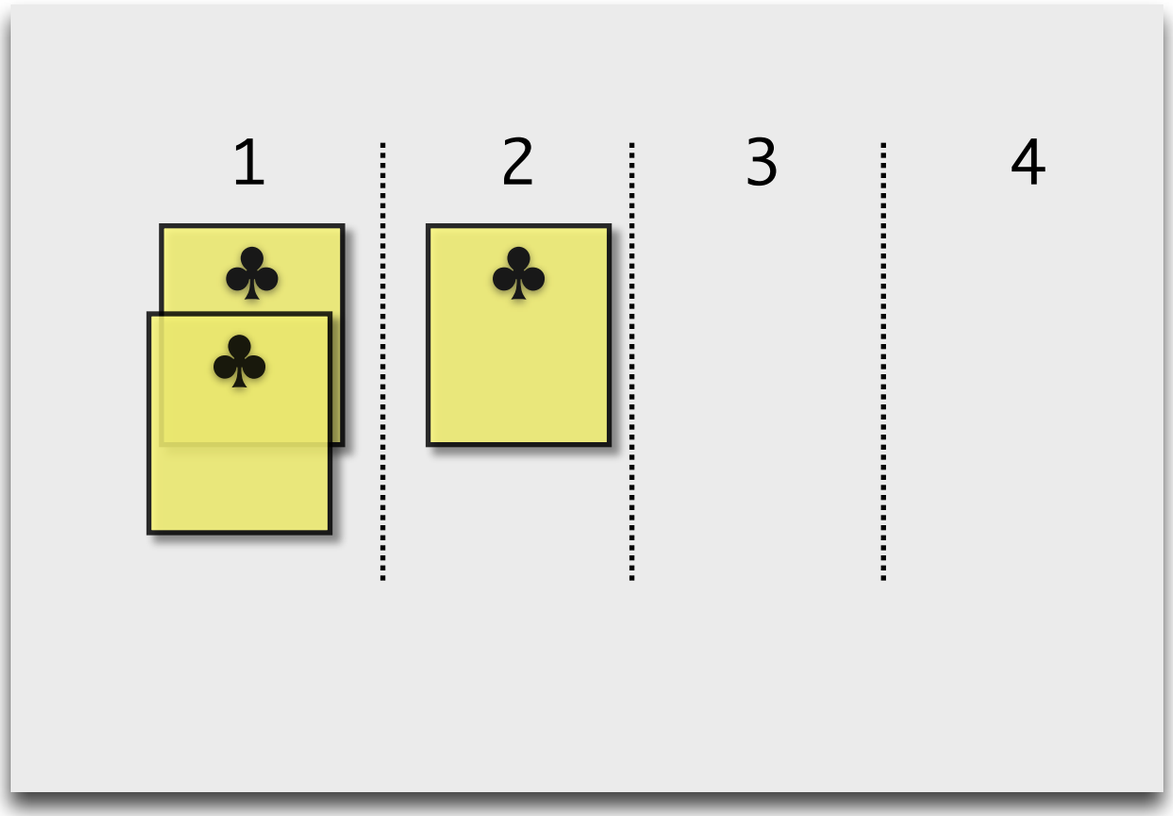
Exemple

```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



Exemple

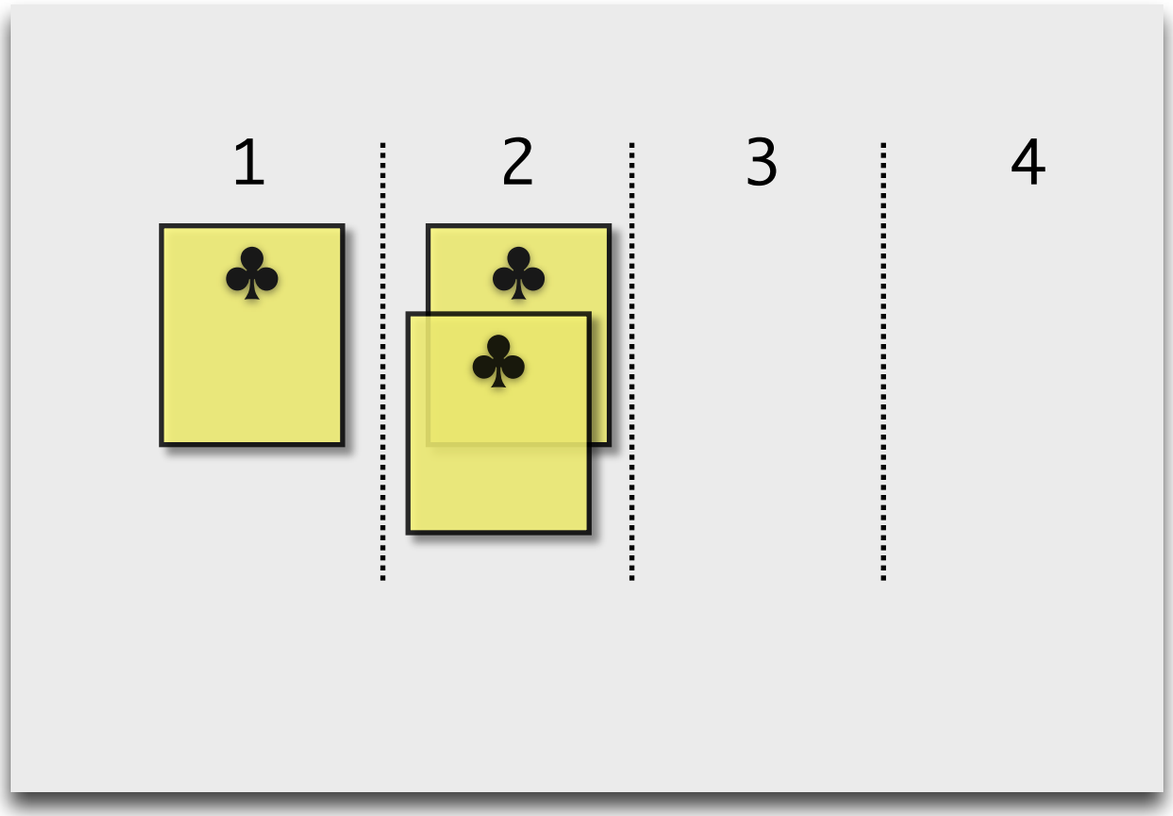
```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



TasNonVide(1) = true

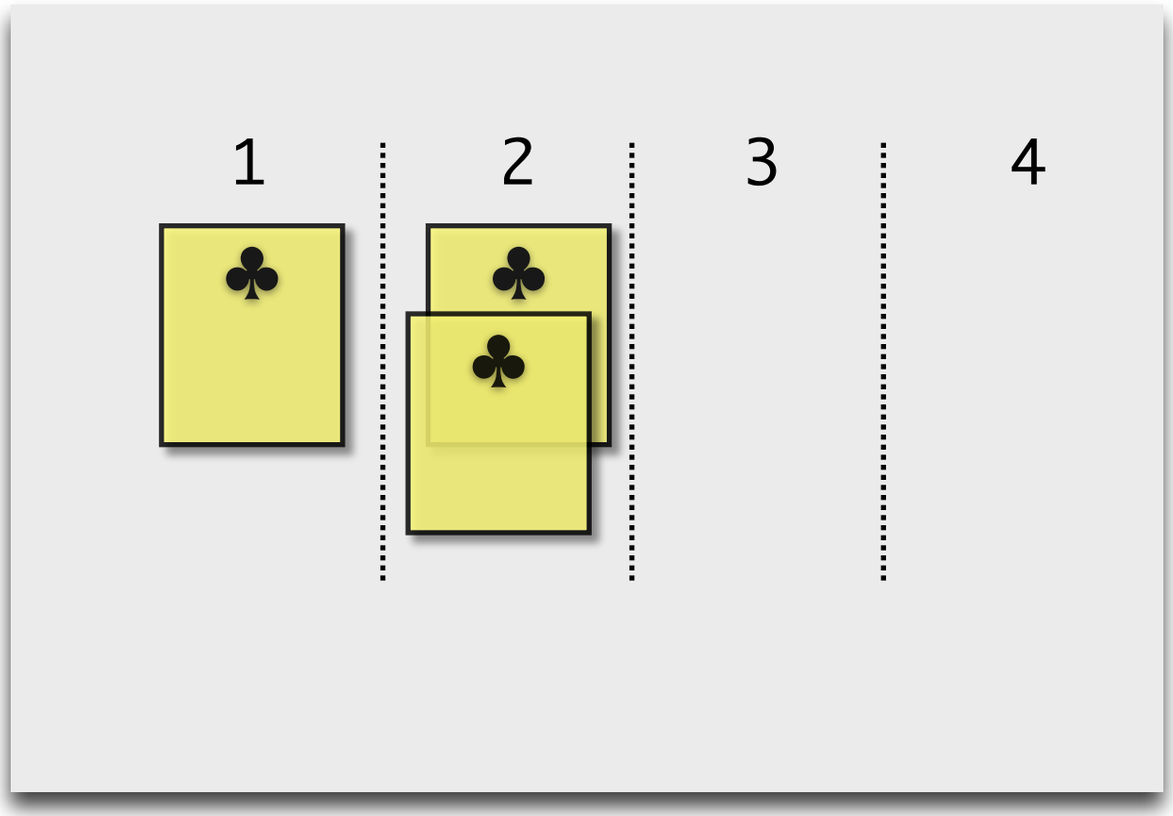
Exemple

```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



Exemple

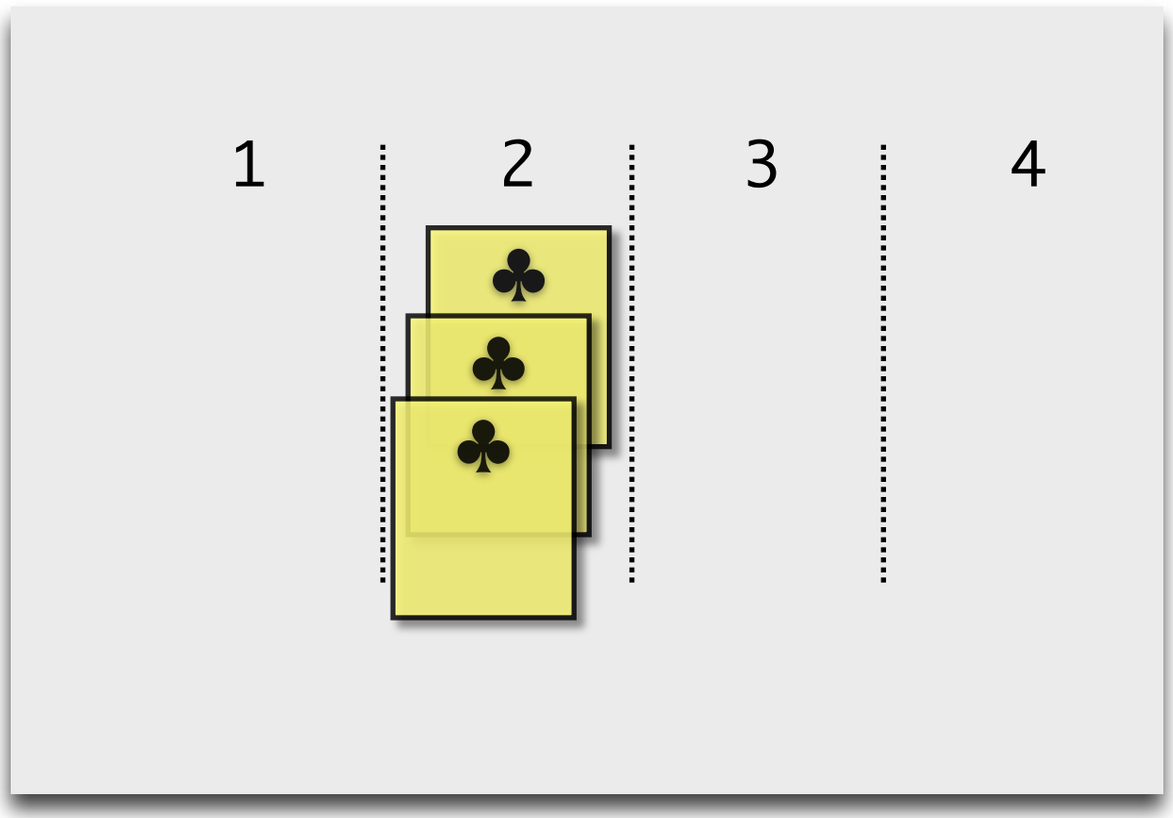
```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



TasNonVide(1) = true

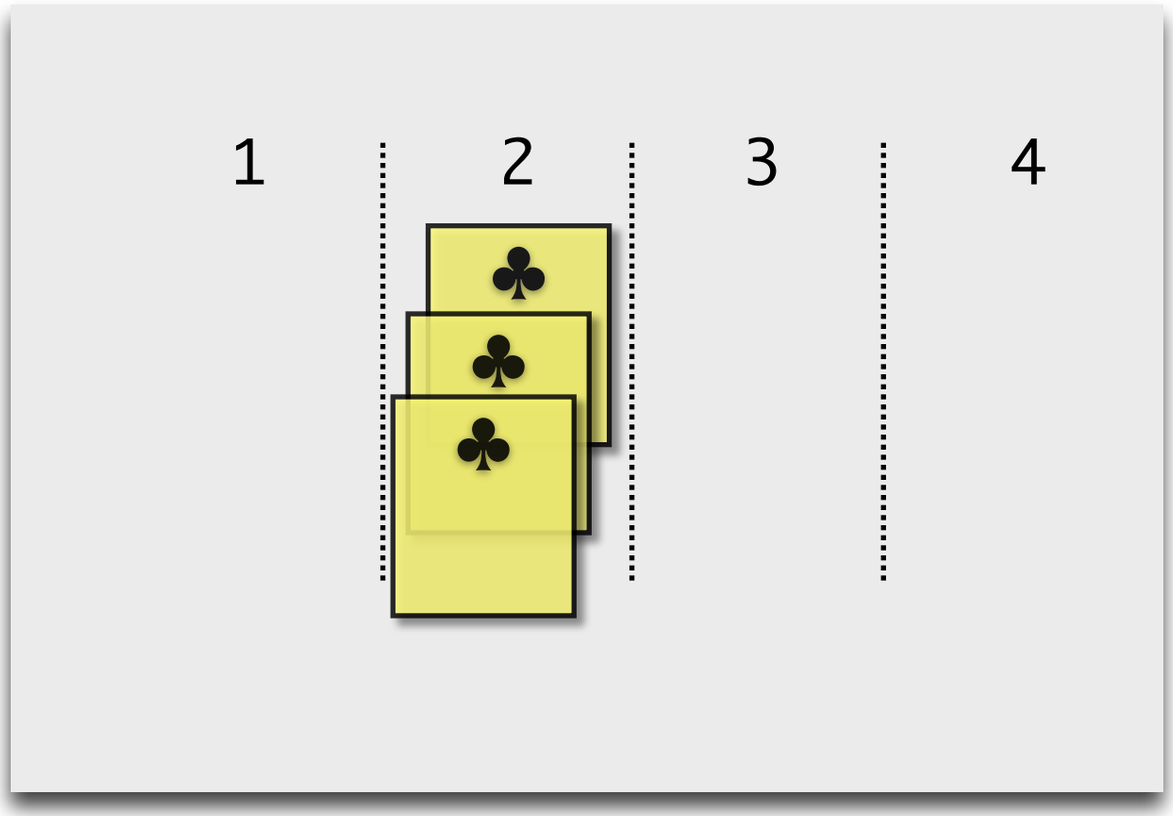
Exemple

```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



Exemple

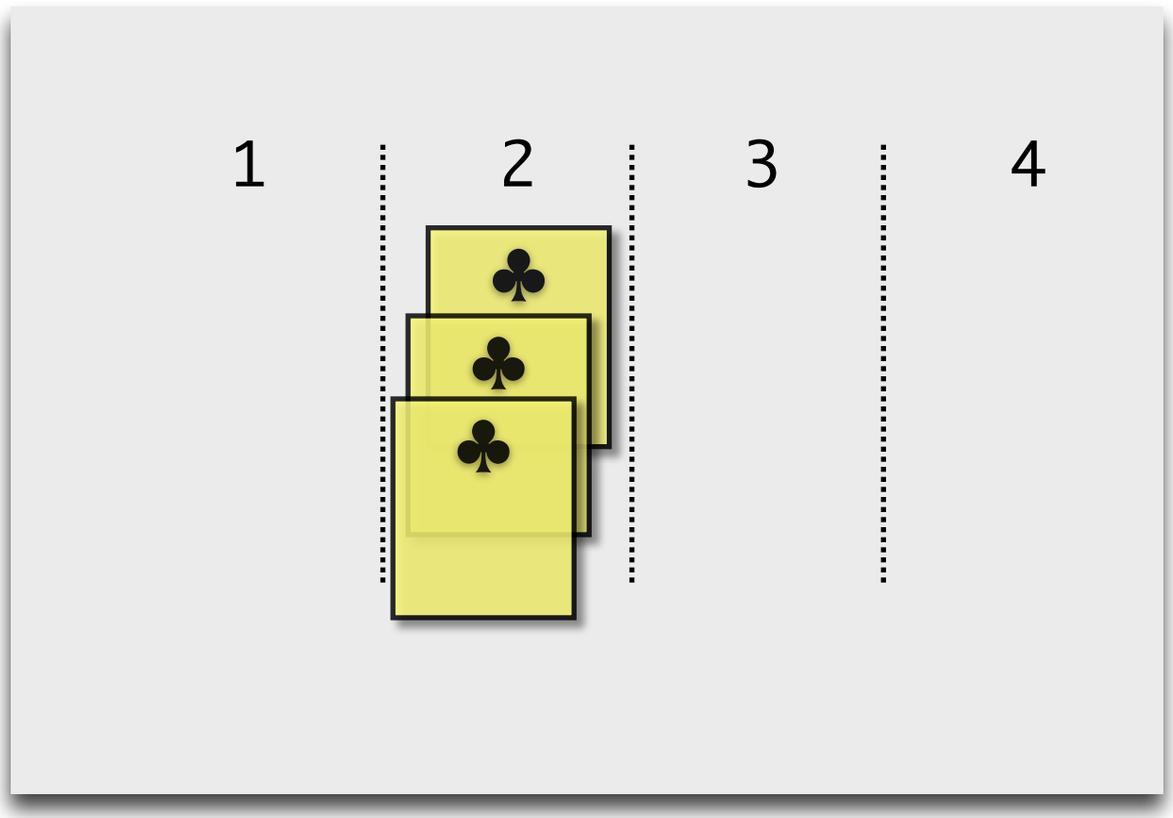
```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
```



TasNonVide(1) = false

Exemple

```
while TasNonVide(1) do
begin
    deplacerSommet(1,2);
end;
suite;
```



Pièges

SI: tas 1:'T', tas 2:'T', autres tas vides

```
while TasNonVide(1) do
begin
    deplacerSommet(2,3);
    deplacerSommet(3,2);
end;
```

Dans quelle situation nous mène cet algorithme ?

Pièges

SI: tas 1:'T', tas 2:'T', autres tas vides

```
while TasNonVide(1) do
begin
    deplacerSommet(2,3);
    deplacerSommet(3,2);
end;
```

Dans quelle situation nous mène cet algorithme ?

Ce programme ne termine jamais !!!

Les instructions dans la boucle doivent garantir qu'on atteindra un jour une situation qui rend la condition fausse (ici: tas 1 vide)

Exercices

Exercice 1

Écrire un algorithme qui résout le problème suivant:
SI : tas 1 : '[T+P]', autres tas vides
SF: tas 1 : '[T]', tas 2 : '[P]', autres tas vides

Exercice 2

Écrire un algorithme qui résout le problème suivant:
SI : tas 1 : '[T][P]', autres tas vides
SF: tas 1 : '[T]', tas 2 : '[P]', autres tas vides

Exercice 3

Écrire un algorithme qui résout le problème suivant:
SI : tas 1 : '[T]P[T]', autres tas vides
SF: tas 1 : 'P[T]', tas 2 : '[T]', autres tas vides

Les procédures

Une procédure est une suite nommée d'instructions

Elle permet d'abstraire cette suite d'instructions et d'y faire référence autant de fois que l'on veut en utilisant simplement son nom.

Les procédures

Une procédure est une suite nommée d'instructions

Exemple: mettre le contenu du tas 1 sur le tas 2
Déclaration de la procédure `ViderTas1SurTas2`:

```
procedure ViderTas1SurTas2;
begin
    while TasNonVide(1) do
    begin
        deplacerSommet(1,2);
    end;
end;
```

Les procédures

Appels de procédure

```
// Remplissage du tas 1
InitTas('TTT',1);

ViderTas1SurTas2;
// ici le tas 1 est vide


// Nouveau remplissage du
// tas 1
InitTas('TTT',1);

ViderTas1SurTas2;
// tas 1 à nouveau vide
// tas 2 ='TTTTTT'
```

Procédures

Déclaration de procédure sans paramètre:

```
procedure nom;
begin
    Instructions;
end;
```

Entête

Procédures

Déclarations de procédures

Programme principal

```
// Auteur
// Date : 28/09/2005
// Objet : exo1 manipulation de cartes
// Etat initial : ...
// Etat final : ...

program nom du program;

uses
    unités séparées par des virgules;

procedure nom de procedure;
begin
    instructions;
end;

begin
    instructions d'initialisation des tas;
    instructions de déplacement;
end.
```