Initiation à la Programmation

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

Structures conditionnelles

Booléens et conditionnelles

L'ordinateur est capable de prendre des décisions. Ses choix seront les bons si le programmeur a prévu tous les cas qui se présenteront au moment de cette prise de décision

Les Booléens

Le mathématicien ***Boole*** a élaboré une théorie mathématiques : l'*algèbre de boole* sur un ensemble comportant 2 valeurs :

Vrai / Faux

ou tout autres entités fondamentalement opposées comme *Oui / Non* ou 1/0 ou encore TRUE / FALSE

Exemple avec les cartes...

Application aux cartes

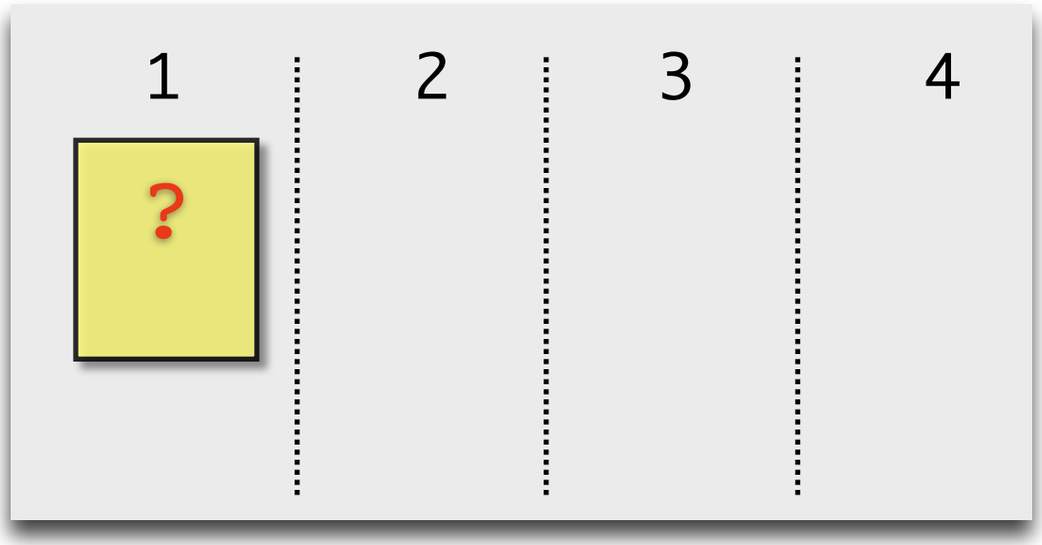
Hypothèse : il y a (au moins) une carte sur le tas 1
Proposition : « la carte au sommet du tas 1 est un ♥»
(En Pascal: `CouleurSommet(1) = COEUR`)

L'évaluation de la proposition mène à 2 valeurs possibles :
Vrai ou Faux

⇨ Satisfait les conditions exprimées par l'algèbre de boole

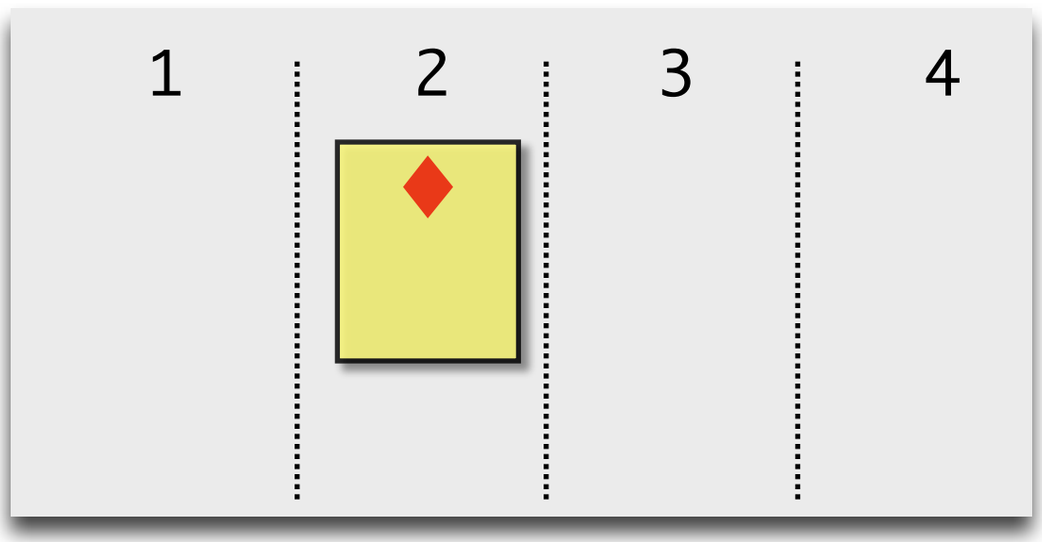
Exemple

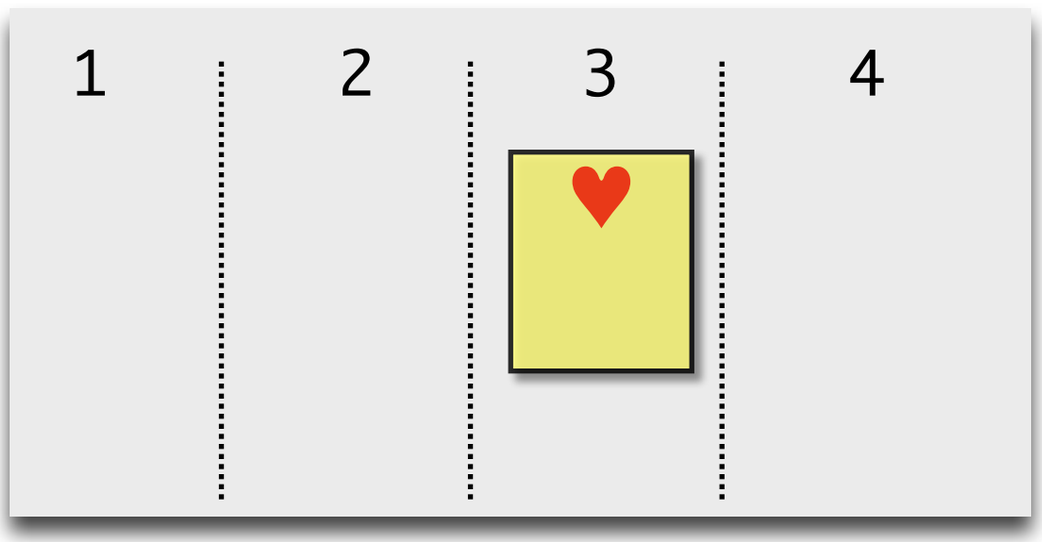
Situation initiale



?

Situation finale





Situation initiale

? = une carte rouge (♦ ou ♥) (« choisie » aléatoirement par la machine.)
Initialisation du tas 1: `InitTas(1,'C+K')`

Situation finale

Si tas 1 initialisé avec ♥, le mettre sur tas 3
Si tas 1 initialisé avec ♦, le mettre sur tas 2

Algorithmes

```
if couleurSommet(1) = COEUR
then
    begin
        deplacerSommet(1,3);
    end
else
    begin
        deplacerSommet(1,2);
    end;
```

Algorithmes

```
if couleurSommet(1) = COEUR
then
    begin
        deplacerSommet(1,3);
    end
else
    begin
        deplacerSommet(1,2);
    end;
```

Test si la couleur du premier sommet est un ♥

Algorithmes

```
if couleurSommet(1) = COEUR
then
    begin
        deplacerSommet(1,3);
    end
else
    begin
        deplacerSommet(1,2);
    end;
```

Déplace la carte au sommet du tas 1 sur le sommet du tas 3

Algorithmes

```
if couleurSommet(1) = COEUR
then
   begin
      deplacerSommet(1,3);
   end
else
   begin
      deplacerSommet(1,2);
   end;
```

```
if couleurSommet(1) = CARREAU
then
   begin
      deplacerSommet(1,2);
   end
else
   begin
      deplacerSommet(1,3);
   end;
```

Sémantique

```
if Condition
then
    begin
        Instructions Alors
    end
else
    begin
        Instructions Sinon
    end;
Instructions suivantes
```

} 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*

Sémantique

```
if Condition
then
    begin
        Instructions Alors
    end
else
    begin
        Instructions Sinon
    end;
Instructions suivantes
```

Pas de ';'

} 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*

Sémantique

Syntaxe alternative

```
if Condition
then
    begin
        Instructions Alors
    end;
Instructions suivantes
```

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

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

Remarque sur la syntaxe alternative

```
if couleurSommet(1) = COEUR
then
    begin
        deplacerSommet(1,3);
    end;

if couleurSommet(1) = CARREAU
then
    begin
        deplacerSommet(1,2);
    end;
```

Cet algorithme est faux!

Pourquoi ?...

Expressions booléennes

Les conditions sont des expressions booléennes

Nous avons vu jusqu'à présent des expressions simples du type

```
couleurSommet(1) = COEUR
```

Il est possible des composer des expressions pour en former d'autres à l'aide d'opérateurs (ou connecteurs) logiques.

Connecteurs logiques: la négation

```
if couleurSommet(1) = COEUR
then
   begin
      deplacerSommet(1,3);
   end
else
   begin
      deplacerSommet(1,2);
   end;
```

Connecteurs logiques: la négation

```
if couleurSommet(1) = COEUR
then
   begin
      deplacerSommet(1,3);
   end
else
   begin
      deplacerSommet(1,2);
   end;
```

```
if not(couleurSommet(1) = COEUR)
then

      ??

else

      ??
```

Connecteurs logiques: la négation

```
if couleurSommet(1) = COEUR
then
   begin
      deplacerSommet(1,3);
   end
else
   begin
      deplacerSommet(1,2);
   end;
```

```
if not(couleurSommet(1) = COEUR)
then
   begin
      deplacerSommet(1,2);
   end
else
   begin
      deplacerSommet(1,3);
   end;
```

Connecteurs logiques: la négation

Le connecteur de négation est le `not`

Si `E` est une expression booléenne, alors `not(E)` est aussi une expression booléenne avec le sens suivant:

E	not(E)
true	false
false	true

Table de vérité du connecteur `not`

Connecteurs logiques: la négation

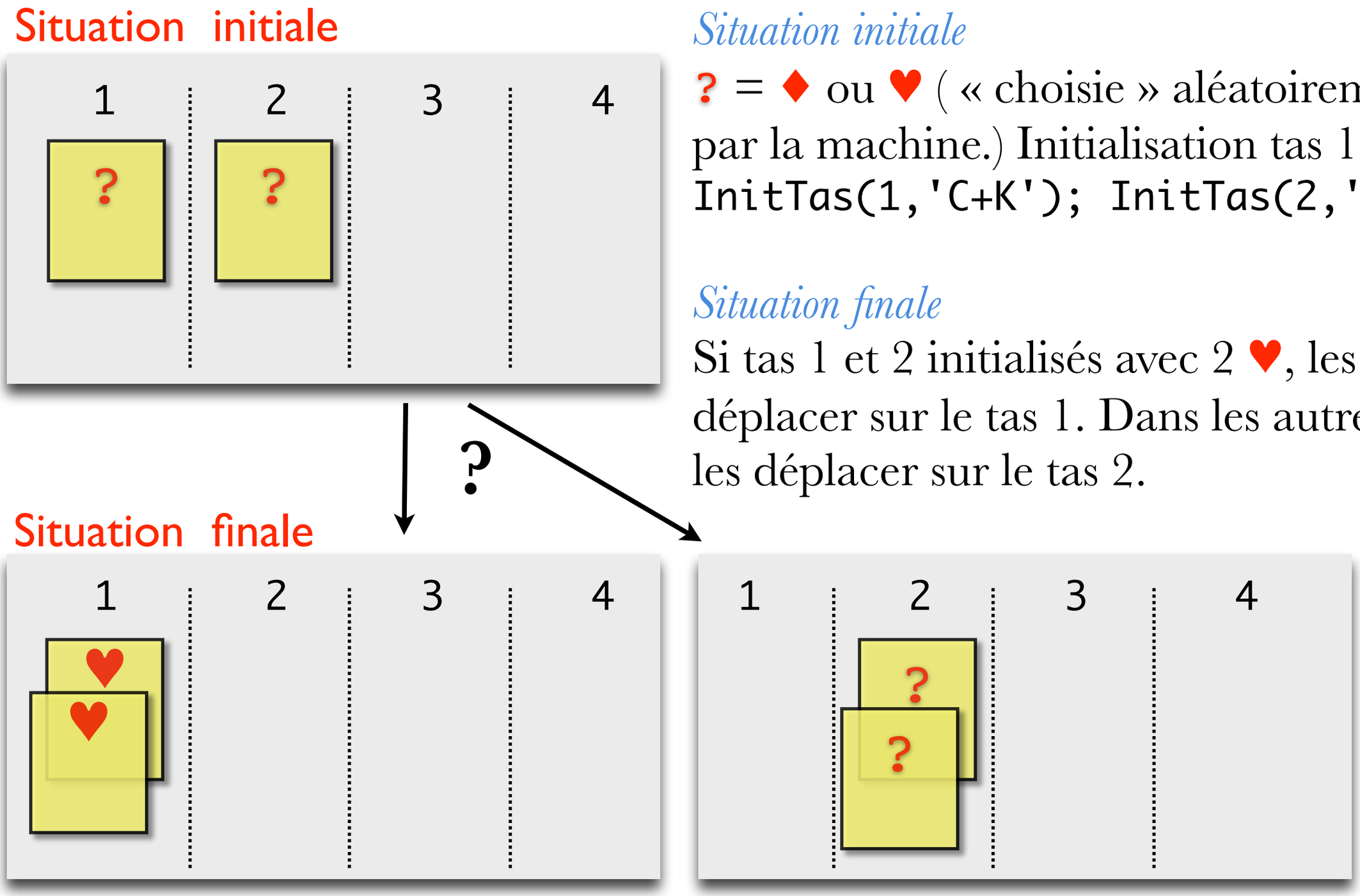
```
if Expression booléenne
then
    begin
        Instructions 1
    end
else
    begin
        Instructions 2
    end;
```

Connecteurs logiques: la négation

```
if Expression booléenne
then
   begin
        Instructions 1
   end
else
   begin
        Instructions 2
   end;
```

```
if not(Expression booléenne)
then
   begin
        Instructions 2
   end
else
   begin
        Instructions 1
   end;
```

Conjonction et disjonction



Situation initiale

? = ♦ ou ♥ (« choisie » aléatoirement par la machine.) Initialisation tas 1 et 2:

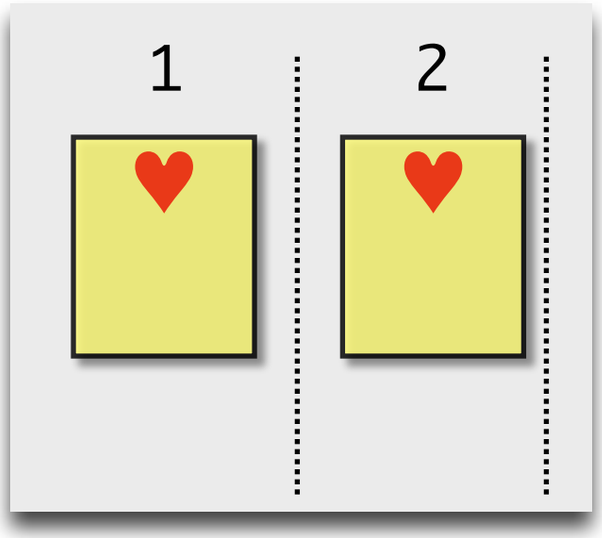
```
InitTas(1,'C+K'); InitTas(2,'C+K');
```

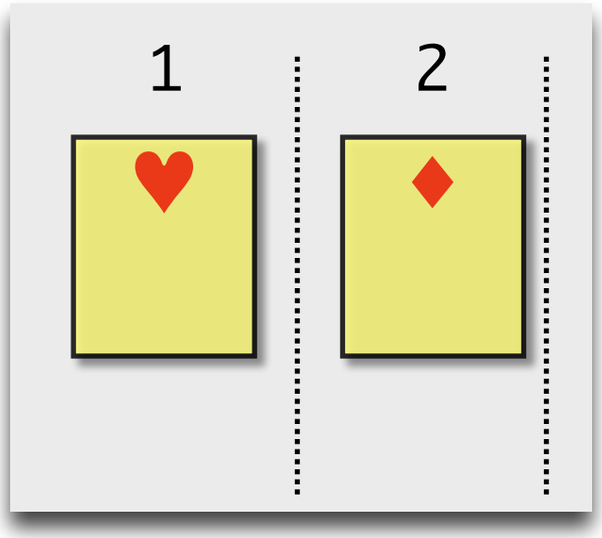
Situation finale

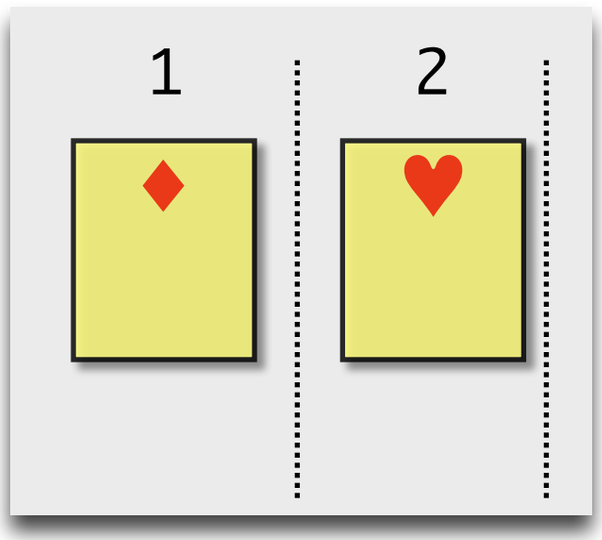
Si tas 1 et 2 initialisés avec 2 ♥, les déplacer sur le tas 1. Dans les autres cas, les déplacer sur le tas 2.

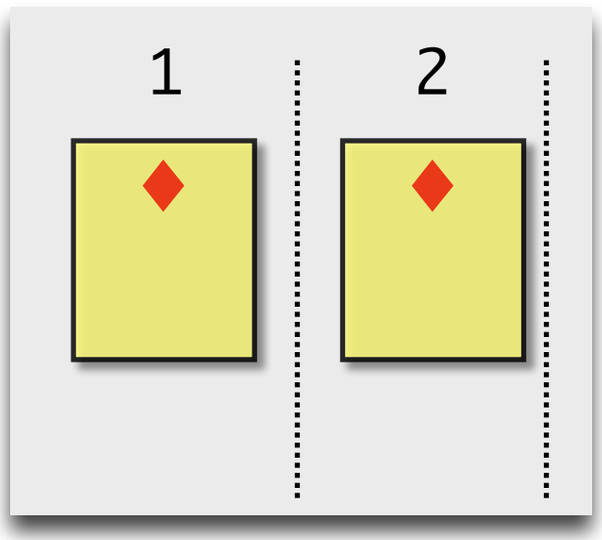
Conjonction et disjonction

Analyse : 4 cas possibles d'initialisation









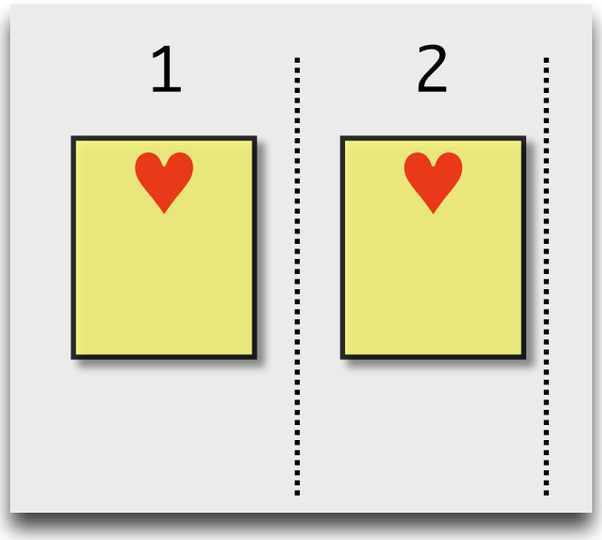
Quel(s) cas d'initialisation mènent au regroupement des carte sur le tas 1 ?

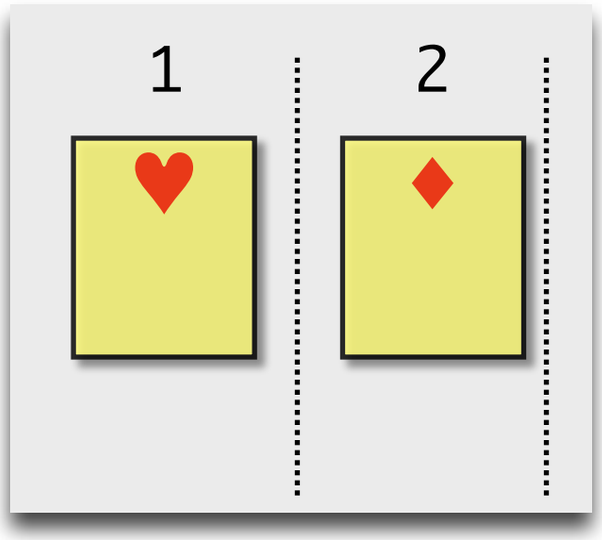
Quel(s) cas d'initialisation mènent au regroupement des carte sur le tas 2 ?

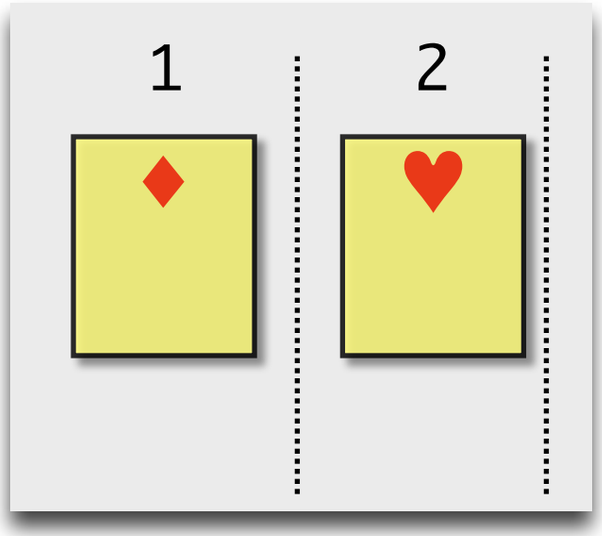
Comment exprimer les propriétés qui les caractérisent ?

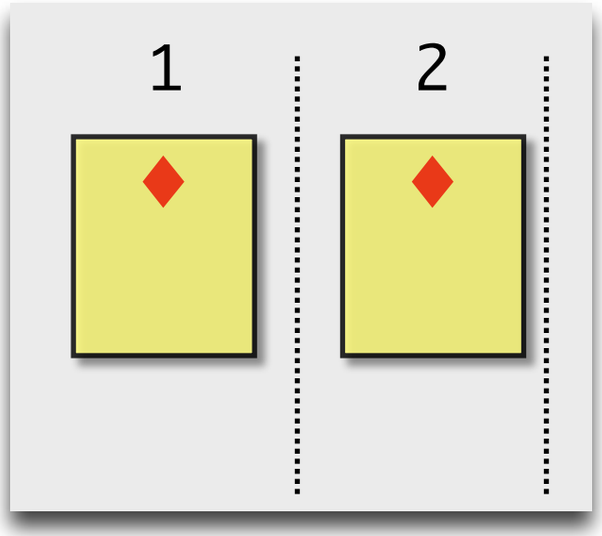
Conjonction et disjonction

Analyse : 4 cas possibles d'initialisation









Regroupement sur tas 1 si:

un ♥ sur le tas 1
et ♥ un sur le tas 2

Regroupement sur tas 2 si:

un ♦ sur le tas 1
ou un ♦ sur le tas 2

Conditions exhaustives et exclusives

Conjonction et disjonction

```
if (couleurSommet(1) = COEUR) and (couleurSommet(2) = COEUR)
then
    begin
        deplacerSommet(2,1);
    end
else
    begin
        deplacerSommet(1,2);
    end;
```

Conjonction et disjonction

```
if (couleurSommet(1) = CARREAU) or (couleurSommet(2) = CARREAU)
then
   begin
      deplacerSommet(1,2);
   end
else
   begin
      deplacerSommet(2,1);
   end;
```

Méthodologie

1. Étude des cas

   - ✓ distinguer les différents cas
   - ✓ vérifier que ces cas sont exhaustifs (on n'oublie rien)
   - ✓ vérifier que ces cas sont exclusifs (pas de redondance)
   - ✓ un exemple de comportement pour chaque cas

2. Pour chacun des cas

   - ✓ établir un test (expression booléenne) permettant de distinguer le cas
   - ✓ déterminer la séquence d'instructions pour ce cas

3. Construire un jeu de tests pour s'assurer de la validité du programme: au moins un test par cas envisagé

Conjonction et disjonction

- ✓ Connecteur de conjonction : `and`
- ✓ Connecteur de disjonction : `or`
- ✓ Si `E` et `E'` sont des expressions booléennes, alors `E and E'` et `E or E'` sont des expressions booléennes avec le sens suivant:

E	E'	E and E'	E or E'
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Table de vérité des connecteurs `and` et `or`

Conjonction et disjonction

E	E’	E and E’	E or E’
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

Exercice:

écrivez les tables de vérité des expressions booléennes suivantes

non(E) et non(E’) non(E ou E’)

Qu’en déduisez-vous ?

Conjonction et disjonction

E	E’	E and E’	E or E’
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

E	E’	not(E)	not(E’)	not(E) and not(E’)	not(E or E’)
true	true	false	false	false	false
true	false	false	true	false	false
false	true	true	false	false	false
false	false	true	true	true	true

Conjonction et disjonction

E	E’	E and E’	E or E’
true	true	true	true
true	false	false	true
false	true	false	true
false	false	false	false

E	E’	not(E)	not(E’)	not(E) and not(E’)	not(E or E’)
true	true	false	false	false	false
true	false	false	true	false	false
false	true	true	false	false	false
false	false	true	true	true	true

Les expressions `not(E) and not(E’)` et `not(E or E’)` sont donc équivalentes (*loi de De Morgan*)

Les Booléens: erreurs fréquentes

Ne pas oublier les parenthèses!

```
if couleurSommet(1) = TREFLE and couleurSommet(2) = TREFLE...
```

est faux car ‘and’ est prioritaire sur ‘=’, et donc équivalent à

```
if couleurSommet(1) = (TREFLE and couleurSommet(2)) = TREFLE...
```

or `(TREFLE and couleurSommet(2))` n’a aucun sens ...

Il faut donc écrire

```
if (couleurSommet(1) = TREFLE) and (couleurSommet(2) = TREFLE)...
```

Les Booléens: erreurs fréquentes

Confusion “et/ou” booléen et “et/ou” du langage naturel !

On ne peut pas écrire

```
if couleurSommet(1) and couleurSommet(2) = TREFLE ...
```

qui est équivalent à

```
if (couleurSommet(1) and couleurSommet(2)) = TREFLE ...
```

où `(couleurSommet(1) and couleurSommet(2))` n’a aucun sens.

Il faut donc écrire

```
if (couleurSommet(1) = TREFLE) and (couleurSommet(2) = TREFLE)...
```