2 Analyse par cas

Traitement à faire si…

Comment exprimer le fait de ne faire certains traitements que si une condition particulière est vérifiée ?

J.M. Adam – UGA – UFR SHS – Département IMSS

La sélection sur choix multiples

Selon liste des indicateurs concernés

condition-1: liste d'instructions 1

condition-2: liste d'instructions 2

….

condition-n: liste d'instructions n

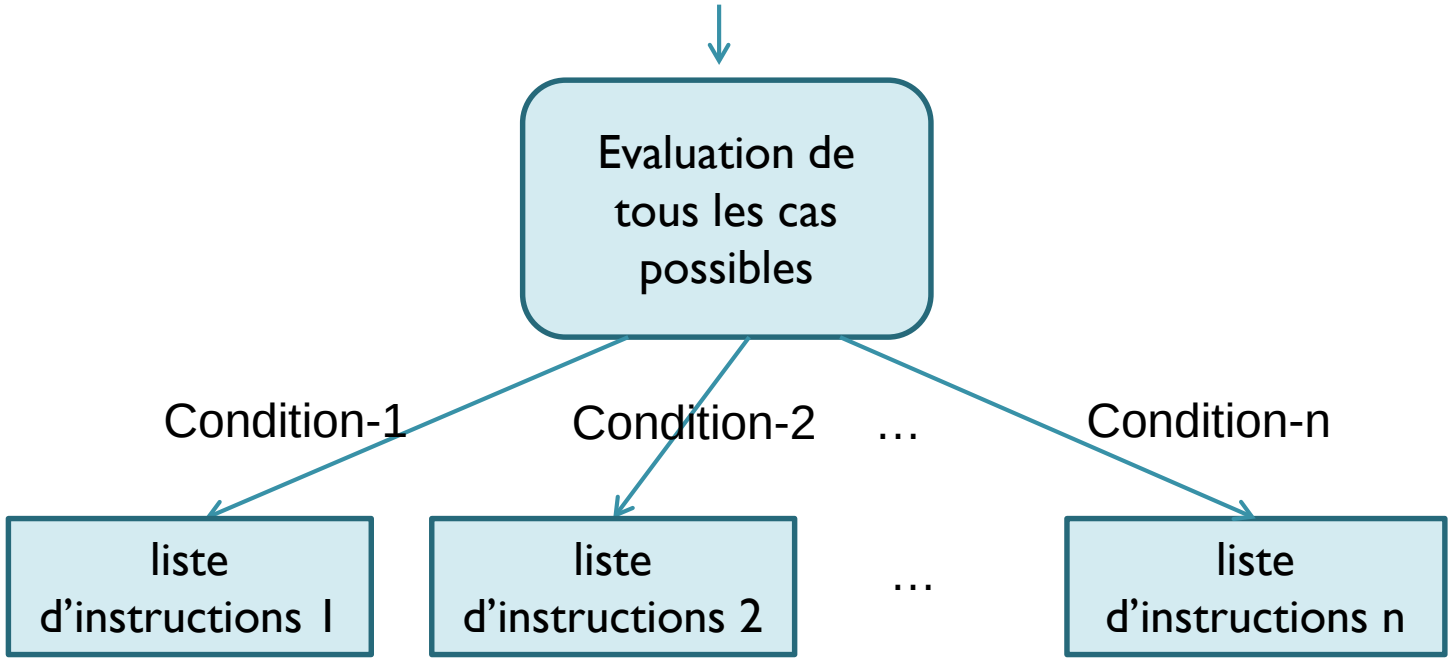
fselon

Expressions logiques

• Les conditions énoncées sont des expressions logiques, elles représentent tous les cas possibles

• Deux conditions distinctes ne doivent jamais être vraies en même temps

Fonctionnement de la sélection sur choix multiples



- Si la condition **i** est vraie, on exécute la liste d'instructions **i**
- **Si aucune des conditions énoncées n'est vraie… erreur !**
- **Si deux conditions ou plus sont vraies… erreur !**

Exemple : afficher la traduction des abréviations M, Mme, Mlle

lexique principal
cl : clavier // périphérique d'entrée
e : écran // périphérique de sortie
abréviation : chaine // donnée : abréviation dont on veut afficher
// la traduction

Algorithme principal
e.afficher("entrez l'abréviation:")
cl.saisir (abréviation)
selon abréviation
    abréviation = "M" : e.afficher("Monsieur ")
    abréviation = "Mme": e.afficher("Madame ")
    abréviation = "Mlle": e.afficher("Mademoiselle ")
fselon

Cherchez l'erreur

lexique principal
cl : clavier // périphérique d'entrée
e : écran // périphérique de sortie
a, b : entiers // données : valeurs à examiner
max : entier // résultat : valeur maximale de a et de b
Algorithme principal
e.afficher("entrez un entier :")
cl.saisir (a) // $a = a_0$, $b = ?$
e.afficher("entrez un autre entier:")
cl.saisir (b) // $a = a_0$, $b = b_0$
selon a, b
    a < b : max ← b
    a > b : max ← a
fselon
e.afficher("maximum : ", max)

L'instruction conditionnelle

si condition **Expression logique**
alors liste d'instructions
fsi

Si la condition prend la valeur **vrai**, la liste d'instructions est exécutée ; si elle prend la valeur **faux**, on ne fait rien.

Exemple d'action conditionnelle

lexique principal
cl : clavier // périphérique d'entrée
e : écran // périphérique de sortie
a, b : entiers // données : valeurs à examiner
max : entier // résultat : valeur maximale de a et de b
Algorithme principal
e.afficher("entrez un entier :")
cl.saisir (a) // $a = a_0$, $b = ?$, $max = ?$
e.afficher("entrez un autre entier:")
cl.saisir (b) // $a = a_0$, $b = b_0$, $max = ?$
max ← b // $a = a_0$, $b = b_0$, $max = b_0$
si a > b
alors max ← a // $a = a_0$, $b = b_0$, $max = a_0$ si $a_0 > b_0$
fsi
// $a = a_0$, $b = b_0$, $max = maximum(a_0 , b_0)$
e.afficher("maximum : ", max)

Les actions alternatives

Si condition **Expression logique**
alors liste d'instructions 1
sinon liste d'instructions 2
fsi

Si la condition prend la valeur **vrai**, la liste d'instructions 1 est exécutée ; si elle prend la valeur **faux**, la liste d'instructions 2 est exécutée.

Exemple d'actions alternatives

lexique principal
cl : clavier // périphérique d'entrée
e : écran // périphérique de sortie
a, b : entiers // données : valeurs à examiner
max : entier // résultat : valeur maximale de a et de b
Algorithme principal
e.afficher("entrez 2 entiers:")
cl.saisir (a, b) // $a = a_0$, $b = b_0$
si a > b
alors max ← a // cas où a > b
sinon max ← b // cas où a ≤ b
fsi
e.afficher("maximum : ", max)

Actions alternatives : attention aux imbrications…

Problème: afficher "Très bien" si une note est supérieure ou égale à 16, "Bien" si elle est supérieure ou égale à 14 et inférieure à 16, "Assez Bien" si elle est supérieure ou égale à 12 et inférieure à 14, "Passable" si elle est supérieure ou égale à 10 et inférieure à 12, et "Insuffisant" dans tous les autres cas.

```
si note ≥ 16
alors e.afficher("Très bien")
sinon si note ≥ 14
        alors e.afficher("Bien")
        sinon si note ≥ 12
                alors e.afficher("Assez Bien")
                sinon si note ≥ 10
                        alors e.afficher("Passable")
                        sinon e.afficher("Insuffisant")
                        fsi
                fsi
        fsi
 fsi
```

Solution basée sur des choix multiples

Problème: afficher "Très bien" si une note est supérieure ou égale à 16, "Bien" si elle est supérieure ou égale à 14 et inférieure à 16, "Assez Bien" si elle est supérieure ou égale à 12 et inférieure à 14, "Passable" si elle est supérieure ou égale à 10 et inférieure à 12, et "Insuffisant" dans tous les autres cas.

```
selon note
    note ≥ 16 :                  e.afficher("Très bien")
    note ≥ 14 et note < 16 :     e.afficher("Bien")
    note ≥ 12 et note < 14 :     e.afficher("Assez Bien")
    note ≥ 10 et note < 12 :     e.afficher("Passable")
    note < 10  :                 e.afficher("Insuffisant")
fselon
```

Actions alternatives : la réalité des langages

Dans les langages de programmation traduction du schéma selon se fait en général par des si successifs de la manière suivante :

```
si note ≥ 16
alors e.afficher("Très bien")
sinon si note ≥ 14
alors e.afficher("Bien")
sinon si note ≥ 12
alors e.afficher("Assez Bien")
sinon si note ≥ 10
alors e.afficher("Passable")
sinon e.afficher("Insuffisant")
fsi
```

La sélection sur choix multiples généralisée

Selon liste des indicateurs concernés

condition 1: liste d'instructions 1

condition 2: liste d'instructions 2

….

condition n: liste d'instructions n

autrement: liste d'instruction n+1

fselon

- Si la condition **i** est vraie, on exécute la liste d'instructions **i**
- Si aucune des **n** conditions énoncées n'est vraie, c'est le cas **autrement** qui est vrai : on exécute la liste d'instructions n+1
- **Si deux conditions ou plus sont vraies… erreur !**

Exemple d'usage de **autrement**

lexique principal
cl : clavier // périphérique d'entrée
e : écran // périphérique de sortie
abréviation : chaine // donnée : abréviation dont on veut afficher
// la traduction
Algorithme principal
e.afficher("entrez l'abréviation:")
cl.saisir (abréviation)
selon abréviation
    abréviation = "M" : e.afficher("Monsieur ")
    abréviation = "Mme": e.afficher("Madame ")
    abréviation = "Mlle": e.afficher("Mademoiselle ")
    autrement : e.afficher("Madame, Monsieur")
fselon

Exemples d'usages inadaptés de **autrement**

```
selon a, b
  | a < b :          max ← b
  | autrement :      max ← a     // c'est de l'abus !
fselon
```

```
selon a, b
  | a < b :          max ← b
  | a ≥ b :          max ← a
  | autrement :      max ← a     // c'est de la superstition…
fselon
```

Exemple d'usage inadapté de **autrement**

```
// a ≠ 0
selon a
    a > 0 : e.afficher("positif")
    a < 0 : e.afficher("négatif")
    autrement : e.afficher("erreur")  // programmation défensive !
fselon
```

Stratégies d'analyse par cas

Problème à résoudre

Ecrire un algorithme qui affiche dans l'ordre croissant trois entiers donnés. On suppose que ces trois entiers sont distincts deux à deux (il n'y a pas de valeurs identiques parmi les trois valeurs à afficher).

lexique principal

```
cl : clavier  // périphérique d'entrée
e : écran     // périphérique de sortie
a, b, c : entiers // données : valeurs à examiner
p, s, t : entier    // résultat : valeurs ordonnées de a, b et c,
                    // de sorte que p < s < t
```

Nous allons étudier diverses manières (stratégies) d'examiner les cas possibles

Stratégie 1 : analyse basée sur l'examen de tous les cas possibles

Les cas possibles :

Notation mathématique	**p**	**s**	**t**
a < b < c	a	b	c
a < c < b	a	c	b
b < a < c	b	a	c
b < c < a	b	c	a
c < a < b	c	a	b
c < b < a	c	b	a

Stratégie 1 : analyse basée sur l'examen de tous les cas possibles

lexique principal
  cl : clavier // périphérique d'entrée
  e : écran // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que p< s< t
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // $a = a_0$, $b = b_0$, $c = c_0$

Spécifique à la stratégie utilisée

  selon a, b, c
    (a < b) et (b < c) : p ← a ; s ← b ; t ← c // $p=a_0, s=b_0, t=c_0$, et $p < s < t$
    (a < c) et (c < b) : p ← a ; s ← c ; t ← b // $p=a_0, s=c_0, t=b_0$, et $p < s < t$
    (b < a) et (a < c) : p ← b ; s ← a ; t ← c // $p=b_0, s=a_0, t=c_0$, et $p < s < t$
    (b < c) et (c < a) : p ← b ; s ← c ; t ← a // $p=b_0, s=c_0, t=a_0$, et $p < s < t$
    (c < a) et (a < b) : p ← c ; s ← a ; t ← b // $p=c_0, s=a_0, t=b_0$, et $p < s < t$
    (c < b) et (b < a) : p ← c ; s ← b ; t ← a // $p=c_0, s=b_0, t=a_0$, et $p < s < t$
  fselon
  // $p < s < t$
  e.afficher(p, ' ', s, ' ', t)

Stratégie 2 : analyse par cas emboîtés

Principe :

- on fait une partition des cas possibles
- puis on analyse les cas possibles dans chaque partition

Pour notre exemple, on distingue:

- Le cas où a > b
- Le cas où a < b
- Dans chaque cas on étudie où peut se situer c par rapport à a et b.

Stratégie 2 : analyse par cas emboîtés

```
lexique principal
  cl : clavier  // périphérique d'entrée
  e : écran     // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que p< s< t
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c)  // a = a0 , b = b0 , c = c0
  selon a, b
    (a < b) : selon a, b, c
                (c < a) :                p ← c ; s ← a ; t ← b   // p=c0, s=a0, t=b0, et p < s < t
                (a < c) et (c < b) : p ← a ; s ← c ; t ← b   // p=a0, s=c0, t=b0, et p < s < t
                (c > b) :                p ← a ; s ← b ; t ← c   // p=a0, s=b0, t=c0, et p < s < t
              fselon
    (a > b) : selon a, b, c
                (c < b)              : p ← c ; s ← b ; t ← a   // p=c0, s=b0, t=a0, et p < s < t
                (b < c) et (c < a) : p ← b ; s ← c ; t ← a   // p=b0, s=c0, t=a0, et p < s < t
                (c > a)              : p ← b ; s ← a ; t ← c   // p=b0, s=a0, t=c0, et p < s < t
              fselon
  fselon
  // p < s < t
  e.afficher(p, '  ', s, '  ', t)
```

Stratégie 3 : Analyses par cas successives

Principe :

- On procède par étapes successives
- Chaque étape nous amène à un état se rapprochant de l'état final

Pour notre exemple on applique l'idée suivante :

1. Placer la plus petite valeur dans p
2. Puis ensuite classer les valeurs s et t en procédant par un échange de valeurs si nécessaire

Stratégie 3 : Analyse par cas successives

```
lexique principal
  cl : clavier          // périphérique d'entrée
  e : écran             // périphérique de sortie
  a, b, c : entiers     // données : valeurs à examiner
  p, s, t : entier      // résultats : valeurs ordonnées de a, b, c, de sorte que p< s< t
  k : entier            // intermédiaire : nécessaire pour échanger s et t
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c)  // a = a0 , b = b0 , c = c0
  selon a, b, c
    (a < b) et (a < c) : p ← a ; s ← b ; t ← c    // p=a0, s=b0, t=c0, et p < s et p < t
    (b < a) et (b < c) : p ← b ; s ← a ; t ← c    // p=b0, s=a0, t=c0, et p < s et p < t
    (c < a) et (c < b) : p ← c ; s ← a ; t ← b    // p=c0, s=a0, t=b0, et p < s et p < t
  fselon
  // p < s et p < t
  si s > t
  alors // échanger les valeurs de s et de t car s > t
     k ← s ; s ← t ; t ← k
     // s < t
  fsi
  // p < s < t
  e.afficher(p, '  ', s, '  ', t)
```

Stratégie 3 : une autre idée

1. On recopie les données dans p s et t
2. On classe les valeurs de p et de s de sorte que p < s
3. On classe les valeurs de s et de t de sorte que s < t
4. On échange les valeurs de p et de s si nécessaire

Stratégie 3 : une autre idée

lexique principal
  cl : clavier // périphérique d'entrée
  e : écran // périphérique de sortie
  a, b, c : entiers // données : valeurs à examiner
  p, s, t : entier // résultats : valeurs ordonnées de a, b, c, de sorte que p< s< t
  k : entier // intermédiaire : nécessaire pour échanger 2 valeurs

Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // $a = a_0 , b = b_0 , c = c_0$
  p ← a ; s ← b ; t ← c // $p = a_0 , s = b_0 , t = c_0$
  si p > s
  alors // échanger les valeurs de p et de s car p > s
    k ← p ; p ← s ; s ← k
  fsi // p < s, $p = \min(a_0 , b_0)$, $s = \max(a_0 , b_0)$, $t = c_0$
  si s > t
  alors // échanger les valeurs de s et de t car s > t
    k ← s ; s ← t ; t ← k
  fsi // s < t , $t = \max(a_0 , b_0 , c_0)$, $p = \min(a_0 , b_0)$, $s = \min(\max(a_0 , b_0), c_0)$
  si p > s
  alors // échanger les valeurs de p et de s car p > s
    k ← p ; p ← s ; s ← k
  fsi
 // p < s et s < t
  e.afficher(p, ' ', s, ' ', t)

Stratégie 4 : Analyse basée sur l'examen des résultats possibles

- Pour notre exemple, cela revient à la solution de la stratégie 1 :

Les résultats possibles :

Conditions correspondant à chaque résultat possible	**p**	**s**	**t**
a < b < c	a	b	c
a < c < b	a	c	b
b < a < c	b	a	c
b < c < a	b	c	a
c < a < b	c	a	b
c < b < a	c	b	a

Question

- On souhaite maintenant généraliser le problème du classement des valeurs a, b, et c dans p, s et t
- Parmi les solutions étudiées, laquelle est la plus adaptée si l'on considère maintenant que a, b et c n'ont pas forcément des valeurs distinctes ?

Mise en œuvre des diverses stratégies sur un exemple plus simple

Calcul du minimum de 3 entiers distincts

Nous présentons une solution mettant en œuvre chacune des 4 stratégies présentées

Calcul du minimum de 3 entiers distincts
Stratégie 1 : examen de tous les cas possibles

```
lexique principal
  cl : clavier          // périphérique d'entrée
  e : écran             // périphérique de sortie
  a, b, c : entiers     // données : valeurs à examiner
  min : entier          // résultat : minimum(a,b,c)
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c)  // a = a0 , b = b0 , c = c0 , a0 ≠ b0 ≠ c0
  selon a, b, c
      (a < b) et (b < c) :  min ← a
      (a < c) et (c < b) :  min ← a
      (b < a) et (a < c) :  min ← b
      (b < c) et (c < a) :  min ← b
      (c < a) et (a < b) :  min ← c
      (c < b) et (b < a) :  min ← c
  fselon
  // min = minimum(a0 , b0 , c0 )
  e.afficher("valeur minimale :", min)
```

Calcul du minimum de 3 entiers distincts
Stratégie 2 : analyse par cas emboîtés

```
lexique principal
  cl : clavier          // périphérique d'entrée
  e : écran             // périphérique de sortie
  a, b, c : entiers     // données : valeurs à examiner
  min : entier          // résultat : minimum(a,b,c)
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // a = a0 , b = b0 , c = c0 , a0 ≠ b0 ≠ c0
  selon a, b
     a < b :  selon a, c
                 a < c : min ← a
                 a > c : min ← c
              fselon
     a > b :  selon b, c
                 b < c : min ← b
                 b > c : min ← c
              fselon
  fselon
  // min = minimum(a0 , b0 , c0 )
  e.afficher("valeur minimale :", min)
```

Calcul du minimum de 3 entiers distincts
Stratégie 3 : analyse par cas successives

```
lexique principal
  cl : clavier            // périphérique d'entrée
  e : écran               // périphérique de sortie
  a, b, c : entiers       // données : valeurs à examiner
  min : entier            // résultat : minimum(a,b,c)
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // a = a0 , b = b0 , c = c0 , a0 ≠ b0 ≠ c0
  min ← a                 // min = a0
  si b < min
  alors  min ← b
  fsi
  // min = minimum(a0 , b0 )
  si c < min
  alors  min ← c
  fsi
  // min = minimum(a0 , b0 , c0 )
  e.afficher("valeur minimale :", min)
```

Calcul du minimum de 3 entiers
Stratégie 4 : examen des résultats possibles

On a 3 résultats possibles : a, b, ou c

```
lexique principal
  cl : clavier          // périphérique d'entrée
  e : écran             // périphérique de sortie
  a, b, c : entiers     // données : valeurs à examiner
  min : entier          // résultat : minimum(a,b,c)
Algorithme principal
  e.afficher("entrez 3 entiers distincts :")
  cl.saisir (a, b, c) // a = a0 , b = b0 , c = c0 , a0 ≠ b0 ≠ c0
  selon a, b, c
      (a < b) et (a < c) :  min ← a
      (b < a) et (b < c) :  min ← b
      (c < a) et (c < b) :  min ← c
fselon
  // min = minimum(a0 , b0 , c0 )
  e.afficher("valeur minimale :", min)
```