Séminaire d'algorithmique: Cours 4 Informatique Différenciée - EISTI - ING 1

Ecole Internationale des Sciences du Traitement de l'Information

Principe itératif

Pourquoi utiliser une structure itérative (boucle) ?

Répéter une suite d'instructions un nombre de fois (in)déterminé.

Première solution

Utiliser une structure conditionnelle autant de fois que nécessaire :

Exemple : saisie d'une réponse

```
variables rep:chaine
ecrire("Entrer une réponse")
lire(rep)
si (rep ≠ "Oui" et rep ≠ "Non") alors
  ecrire("Entrer une réponse")
  lire(rep)
fsi
...
si (rep ≠ "Oui" et rep ≠ "Non") alors
  ecrire("Entrer une réponse")
  lire(rep)
fsi
Suite Instructions
```

Problème : on ne peut pas prévoir le nombre de saisies

Instruction d'itération

Comment ?

Syntaxe d'une boucle "tant que" :

```
tant que Condition faire
  Instructions
ftq
```

Exemple : saisie d'une réponse

```
variables rep:chaine
ecrire("Entrer une réponse")
lire(rep)
tant que (rep ≠ "Oui" et rep ≠ "Non") faire
  ecrire("Entrer une réponse")
  lire(rep)
ftq
Suite Instructions
```

Boucles pour compter

Pourquoi

Dans de nombreux cas, une boucle sert principalement à compter, c'est à dire à effectuer la suite d'instructions un nombre bien défini de fois. On pourra utiliser dans ce cas une structure particulière adéquate : la boucle "pour".

Syntaxe

```
pour Compteur ← Initial à Final pas ValeurDuPas
  Instructions
fpour
```

- Initial contient la valeur initiale du compteur
- Final contient la valeur finale du compteur
- ValeurDuPas contient la valeur de l'incrémentation du compteur à chaque boucle

Boucles pour compter

Exemple : écrire les chiffres de 1 à 9

```
compteur ← 1
tant que compteur < 10 faire
  ecrire(compteur)
  compteur ← compteur + 1
ftq
```

```
pour compteur ← 1 à 9 pas 1
  ecrire(compteur)
fpour
```

Boucles imbriquées

Pourquoi

Comme pour les structures conditionnelles, il est possible d'imbriquer des structures itératives pour combiner plusieurs itérations dépendantes les unes des autres.

Exemple : écrire les tables de multiplication

```
pour Compteur1 de 1 à 9 pas 1
  pour Compteur2 de 1 à 9 pas 1
    ecrire(Compteur1*Compteur2)
  fpour
fpour
```

Pour obtenir un affichage correcte, il faut bien faire attention au placement des saut de lignes, en utilisant par exemple la procédure ecrirenl :

```
pour Compteur1 de 1 à 9 pas 1
  pour Compteur2 de 1 à 9 pas 1
    ecrire((Compteur1*Compteur2)&" ")
  fpour
  ecrirenl()
fpour
```

Instructions d'itération

Pièges à éviter

La manipulation des boucles impose une rigueur particulièrement exigente afin d'éviter les erreurs suivantes :

- Ecrire une boucle dont la condition d'entrée ne sera jamais vraie. La suite d'instructions ne sera jamais exécutée, et ne sert donc à rien.
- Encore plus grave : écrire un boucle dont la condition d'entrée ne sera jamais fausse. La suite d'instructions sera exécutée indéfiniment et finira par provoquer le "plantage" de la machine (problème de mémoire en général)

Ces deux pièges doivent impérativement être évités pour toute écriture de structure itérative, ce qui va être facilité par l'utilisation d'assertions un peu particulières : les invariants de boucle.

Notion d'invariant de boucle

Définition

Un invariant de boucle est une assertion particulière associée à une boucle. Il s'agit donc d'une formule booléenne toujours vraie quelque soit l'itération de la boucle.

Pourquoi

L'apport essentiel de l'invariant de boucle est de permettre une démonstration formelle du résultat produit par une boucle.

Invariant de boucle

Exemple : division euclidienne

```
B ← b
R ← a
Q ← 0
{a = B * Q + R}
tant que R ≥ B faire {(a = B * Q + R) ∧ (R ≥ B)}
  R ← R − B
  Q ← Q + 1
ftq
{(a = B * Q + R) ∧ (R < B)}
```

Preuve de l'invariant

- Conditions initiales : $a = b * 0 + a = a$
- Soit R', B', Q' les valeur modifiées par la boucle de R, B, Q :
  - $R' = R - B$ et $Q' = Q + 1$
  - donc $B' * Q' + R' = B * (Q + 1) + R - B = B * Q + B + R - B = B * Q + R$
  - de plus, $R - B$ diminue strictement (si $B \neq 0$)
- En sortie de boucle, on a $a = B * Q + R$ et $R < B$ $\square$

Invariant de boucle

Exemple : calcul de puissance

```
A ← a
N ← n
R ← 1
{A^N * R = a^n}
tant que N > 0 faire
   si pair(N) alors
      A ← A*A
      N ← N/2
      {(A^N * R = a^n) ∧ (N > 0)}
   sinon
      R ← R*A
      N ← N−1
      {(A^N * R = a^n) ∧ (N > 0)}
   fsi
ftq
{(A^N * R = a^n) ∧ (N = 0)}  donc  {R = a^n}
```

Notion de coût

Définition

Le coût d'un algorithme peut être déterminé par le nombre d'instructions nécessaires pour le dérouler.

- Calcul très simple pour les instructions séquentielles (comptage)
- Calcul assez simple pour les instructions conditionnelles (on considère le pire des cas)
- Calcul plus compliqué pour les instructions itératives (il faut évaluer le nombre de boucles dans le pire des cas) :
  - Simple en cas de boucle "pour" : valeur du compteur
  - Nécessite un peu de réflexion pour une boucle "tant que" : nombre d'étapes avant que la condition soit vraie
  - Peut être délicat à calculer pour les boucles imbriquées : dépend du degré d'imbrication et de l'indépendance des conditions d'arrêt des boucles.

Coût d'un algorithme

Exemple : division euclidienne

```
B ← b    // 1 instruction
R ← a    // 1 instruction
Q ← 0    // 1 instruction
{a = B * Q + R}
tant que R ≥ B faire             // 1 instruction
   {(a = B * Q + R) ∧ (R >= B)}
   R ← R − B                     // 1 instruction
   Q ← Q + 1                     // 1 instruction
ftq                              // Q boucles
{(a = B * Q + R) ∧ (R < B)}  // Total: 3 * Q + 3
```

Le coût ne doit dépendre que des paramètres d'entrée de l'algorithme, ici a et b. On exprime donc Q en fonction de a et b à la fin de l'algorithme :

- D'après l'assertion finale, $a = B * Q + R$ et $R < B$
- Donc $Q = (a - R)/B < (a - B)/B = (a - b)/b$
- Le coût est donc majoré par $3 * (a - b)/b + 3$