3.Les instructions conditionnelles (les alternatives)

a) Introduction

Les algorithmes comportent généralement deux types d'instructions :

- Les instructions simples : qui permettent la manipulation des variables telles que l'affectation, la lecture et l'écriture.

- Les instructions de contrôle : qui précisent l'enchainement chronologique des instructions simples. C'est en particulier le cas des instructions conditionnelles ou les tests.

3.Les instructions conditionnelles (les alternatives)

b) Structure d'un test

Il existe deux formes de test : forme simple (ou réduite) et forme complète.

Forme simple

Dans cette forme, une action qui correspond à une ou plusieurs instructions, est exécuté si une condition est vérifiée. Sinon l'algorithme passe directement au bloc d'instruction qui suit immédiatement le bloc conditionnel.

Syntaxe :

Si (condition) **Alors**

instruction(s) ; // *action*

- **Finsi**

3.Les instructions conditionnelles (les alternatives)

Remarque :

La condition évaluée après l'instruction « Si » est une variable ou une expression booléenne qui, à un moment donné, est Vraie ou Fausse. par exemple : x=y ; x <= y ; …

Exemple :

x ← 5 ; y ← 9 ;

Si (x = y) **Alors Ecrire** ("x est égale à y") ;

Dans cet exemple, le message « x est égale à y » ne sera pas affiché puisque la condition (x = y) n'est pas vérifiée

3.Les instructions conditionnelles (les alternatives)

Forme complexe

Cette forme permet de choisir entre deux actions selon qu'une condition est vérifiée ou non.

Syntaxe :

Si (condition) **Alors**

instruction(s) 1 ; // *action1*

Sinon instruction(s) 2 ; // *action2*

Finsi

3.Les instructions conditionnelles (les alternatives)

Remarque :

Certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme d'une simple comparaison. Par exemple, la condition x ∈ [0, 1[s'exprime par la combinaison de deux conditions x >= 0 et x < 1 qui doivent être vérifiées en même temps.

Pour combiner ces deux conditions, on utilise les opérateurs logiques. Ainsi, la condition x ∈ [0, 1[pourra s'écrire sous la forme : (x >= 0) ET (x < 1). Cette dernière est appelée une condition composée ou complexe.

3.Les instructions conditionnelles (les alternatives)

Exemple (sur la forme complète d'un test) :

x ← 5 ; y ← 9 ;

Si (x = y) **Alors**

Ecrire ("x est égale à y ") ;

Sinon

Ecrire ("x est différente de y ") ;

Fin SI

Avec cette forme, on peut traiter les deux cas possibles. Si la condition (x=y) est vérifiée, le premier message est affiché, si elle n'est pas vérifiée, le deuxième message est affiché

3.Les instructions conditionnelles (les alternatives)

Tests imbriqués

La forme **« Si … Alors…Sinon »** permet deux choix correspondants à deux traitements différents. Dans d'autres situations, on pourra avoir plus de deux cas ce qui rend cette alternative insuffisante pour traiter tous les cas possibles (voir exemple ci-dessous).

La forme complète permet de choisir entre plusieurs actions en imbriquant des formes simples selon la syntaxe ci-dessous.

3.Les instructions conditionnelles (les alternatives)

Syntaxe :

Si (condition1) **Alors**

instruction(s) 1 ;

Sinon Si (condition2) **Alors**

instruction(s) 2 ;

Sinon Si (condition3) **Alors**

instruction(s) 3 ;

...

Sinon instruction(s) N

Fin si

Fin Si

Fin si

3.Les instructions conditionnelles (les alternatives)

Exemple : Etat de l'eau

Dans les conditions normales de température et de pression, l'eau est sous forme de glace si la température est inférieure ou égale à 0° C, sous forme de liquide si la température est comprise entre 0° C et 100° C et sous forme de vapeur au-delà de 100° C. Ecrivons l'algorithme qui permet de vérifier l'état de l'eau selon sa température.

La solution pourrait être comme suit :

3.Les instructions conditionnelles (les alternatives)

```
Algorithme Etat_Eau ;
 Var t : réel ;
Début
      Ecrire ("Donner la température de l'eau :") ;
     Lire (t) ; Si (t <= 0) Alors
      Ecrire ("Etat solide") ;
      FinSi Si (t > 0 ET t < 100) Alors
     Ecrire ("Etat liquide") ;
     Finsi Si (t >= 100) Alors
      Ecrire ("Etat gazeux") ;
     Finsi
Fin
```

3.Les instructions conditionnelles (les alternatives)

Cet algorithme est correct mais il évalue les trois conditions qui portent sur la même variable et qui sont exclusives. En effet, si (t <= 0), alors on ne peut pas avoir (t>= 0 et t < 100) ni (t > 100). Il est donc inutile d'évaluer les deux dernières conditions si la première est vérifiée, ou d'évaluer la dernière condition si la deuxième est vérifiée. Pour éviter ce cas de figure, il sera préférable d'utiliser des tests imbriqués comme suit :

3.Les instructions conditionnelles (les alternatives)

```
Algorithme Etat_Eau ;
 Var t : réel ;
Début
      Ecrire ("Donner la température de l'eau:") ;
       Lire (t) ;
      Si (t <= 0) Alors
      Ecrire ("Etat solide") ;
            Sinon Si (t < 100) Alors
            Ecrire (" Etat liquide") ;
             Sinon
            Ecrire ("Etat gazeux") ;
            Finsi
      Finsi
Fin
```

3.Les instructions conditionnelles (les alternatives)

Les choix multiples

Il existe une autre variante d'instructions conditionnelles qui permet d'effectuer des actions différentes suivant les différentes valeurs que peut avoir une variable. Cette structure est décrite comme suit :

Syntaxe :

Selon (variable)

valeur1 : instruction(s) 1 ;

valeur2 : instruction(s) 2 ;

...

valeurN : instruction(s) N ;

défaut : instruction(s) par défaut;

FinSelon ;

3.Les instructions conditionnelles (les alternatives)

Remarque :

Dans la structure de test à choix multiples :

- L'action peut être une suite d'instructions ;

- La valeur est une constante de même type que la variable ;

- La partie « défaut » est exécutée si aucun des autres cas n'est vérifié ;

- L'exécution des différents cas (y compris le cas par défaut) est exclusive c'est-à-dire l'exécution d'un seul cas provoque la sortie de cette structure.

Exemple :

Dans ce qui suit, le nom du jour de la semaine correspondant est affiché selon la valeur de la variable « jour ».

jour ← 5 ;

Selon jour

1 : **Ecrire** ("Dimanche") ;

2 : **Ecrire** ("Lundi") ;

3 : **Ecrire** ("Mardi") ;

4 : **Ecrire** ("Mercredi") ;

5 : **Ecrire** ("Jeudi") ;

6 : **Ecrire** ("Vendredi") ;

7 : **Ecrire** ("Samedi") ;

Défaut : **Ecrire** ("Numéro de jour invalide.") ;

FinSelon

Donc, l'expression « Jeudi » est affichée dans ce cas

4. Les instructions itératives (les boucles)

a) Introduction

Considérons le même exemple qu'on a vu précédemment concernant le calcul de la moyenne générale d'un étudiant. Pour se faire, on doit :

- ✓ Lire toutes les notes (et leurs coefficients) de l'étudiant,
- ✓ Calculer la somme des notes,
- ✓ Diviser la somme obtenue sur le nombre (ou sur la somme des coefficients).

4. Les instructions itératives (les boucles)

a) Introduction

Pour **N** d'étudiants, il nous faudra donc répéter **N** fois la même séquence d'instructions. Cet exemple soulève deux questions importantes :

1- Comment éviter d'écrire plusieurs fois la même séquence d'instructions ?

2- Combien de fois doit-on répéter l'exécution de la séquence d'instructions pour obtenir le résultat attendu ?

Pour répondre à ces questions, de nouvelles instructions de contrôle sont introduites. Il s'agit des instructions itératives (appelées aussi les boucles ou les itérations).

4. Les instructions itératives (les boucles)

b) Définition

Une boucle (ou itération) est une instruction de contrôle qui permet de répéter plusieurs fois un ensemble d'instructions. Généralement, deux cas sont distingués :

- Le nombre de répétitions est connu.
- Le nombre des répétitions est inconnu ou variable.

4. Les instructions itératives (les boucles)

C) L'instruction « Pour »

Lorsque le nombre de répétitions est déterminé (connu), l'utilisation de l'instruction « **Pour** » est privilégiée. Une structure de boucle avec l'instruction « **Pour** » s'arrête une fois que le nombre de répétitions est atteint. Cette structure possède un indice (compteur) de contrôle d'itérations caractérisé par :

- **une valeur initiale**,
- **une valeur finale**,
- **un pas de variation**.

4. Les instructions itératives (les boucles)

Syntaxe :

Pour indice **de** début **à** fin **Pas** valeur_du_pas

    instruction(s) ;

FinPour

4. Les instructions itératives (les boucles)

Exemple : un compteur croissant/décroissant

Les deux algorithmes suivants comptent de 1 à N et de N à 1 respectivement.

```
Algorithme compteur_croissant ;
Var  i : entier ;
Const N=100 ;
Début
   Pour i de 1 à N    /* par défaut le  pas = 1 */
         Ecrire (i);
   FinPour
Fin
```

Résultat d'exécution : 1,2, 3, … , 99, 100

```
Algorithme compteur_decroissant ;
Var  i : entier ;
Const N=100 ;
Début
   Pour i de N à 1  /* par défaut le  pas = -1 */
         Ecrire (i);
   FinPour
Fin
```

Résultat d'exécution : 100, 99, 98, … , 2, 1

Si la valeur du « pas » n'est pas précisée dans l'instruction « Pour », elle est par défaut égale à un (1).

4. Les instructions itératives (les boucles)

d) L'instruction « Tant que … Faire »

Cette instruction permet de tester une condition et répéter le traitement associé tant que cette condition est vérifiée.

- **Syntaxe**:
- **Tant que** condition **faire** instruction(s) ;
- **FinTant q**ue

4. Les instructions itératives (les boucles)

d) L'instruction « Tant que … Faire »

Exemple : Réécrivons l'algorithme précédent avec cette instruction.

```
Var i : entier ;
Début
        i ← 1 ;
        Tant que (i<=100) faire
                Ecrire (i) ;  i ← i+1 ;
        FinTq
Fin
```

4. Les instructions itératives (les boucles)

Exemple : Soit l'algorithme suivant :

```
Var n, p : entier ;
Début
   Répéter
        Ecrire ("Donner un nombre :") ; Lire (n) ;
        p ← n*n ; Ecrire (p);
   Jusqu'à (n=0)
   Ecrire ("Fin de l'algorithme") ;
Fin
```

4. Les instructions itératives (les boucles)

Les instructions encadrées par les mots **répéter** et **jusqu'à** constituent le bloc de la boucle qu'il faut répéter jusqu'à ce que la condition (**n=0)** soit vérifiée. Donc le nombre de répétitions de cette boucle dépend des données fournies par l'utilisateur.

4. Exercices

Réécrire l'algorithme précédent avec « Tant que… faire » puis avec « Pour ».

4. Les instructions itératives (les boucles)

La notion du compteur

Un compteur est une variable associée à la boucle dont la valeur est incrémentée de un à chaque itération. Elle sert donc à compter le nombre d'itérations (répétitions) de la boucle. La notion du compteur est associée particulièrement aux deux boucles : « **Répéter…jusqu'à** » et « **Tant que…faire** ». Par contre, dans la boucle « **Pour** », c'est l'indice qui joue le rôle du compteur.

L'utilisation du compteur dans les deux premières boucles est exprimée ainsi :

4. Les instructions itératives (les boucles)

L'utilisation du compteur dans les deux premières boucles est exprimée ainsi :

Bloc de la boucle

```
compt ← 0 ;
Répéter
    instruction(s) ;
    …
    compt ← compt +1 ;
Jusqu'à (condition) ;
```

```
compt ← 0 ;
Tant que (condition) faire
    instruction(s) ;
    …
    compt ← compt +1 ;
FinTant que ;
```

4. Les instructions itératives (les boucles)

Remarque

Il faut toujours initialiser le compteur avant de commencer le comptage. La variable « compt » (utilisée ci-dessus comme compteur), a été initialisée à zéro (0) avant le début de chaque boucle.

L'instruction « compt ← compt +1 » incrémente la valeur de « compt » de un (1). Elle peut être placée n'importe où à l'intérieur du bloc de la boucle.

4. Les instructions itératives (les boucles)

Exemple :

```
i ← 0 ;
Répéter
    Ecrire (i);
    i ← i +1 ;
Jusqu'à (i=5) ;
```

Résultat d'exécution : 0, 1, 2, 3, 4

```
i ← 0 ;
Tant que (i<5) faire
    Ecrire (i);
    i ← i +1 ;
FinTant que ;
```

Résultat d'exécution : 0, 1, 2, 3, 4

4. Les instructions itératives (les boucles)

La notion d'accumulation

Cette notion est fondamentale en programmation. Elle est utilisée notamment pour calculer la somme d'un ensemble de valeurs. L'instruction correspondante se présente ainsi : variable ← variable + valeur ;

Cette instruction consiste à ajouter une valeur à une variable numérique, puis affecter le résultat dans la variable elle-même. En d'autres termes, la nouvelle valeur de variable égale à l'ancienne plus une certaine valeur [4].

4. Les instructions itératives (les boucles)

La notion d'accumulation

Exemple : calcul de la somme de n valeurs données par l'utilisateur :

```
Var  i, n : entier ; som, val : réel ;
Début
    Écrire ("Donner le nombre de valeurs :") ; Lire (n) ;
    som ← 0 ;
    Pour i de 1 à n
            Écrire ("Enter une valeur :") ; Lire (val) ;
            som ← som + val ;
    Finpour
    Écrire ("La somme des valeurs est égale à :", som) ;
Fin
```

4. Les instructions itératives (les boucles)

Les boucles imbriquées

Les boucles peuvent être imbriquées les unes dans les autres. Deux ou plusieurs boucles imbriquées peuvent être aussi les mêmes ou différentes

Exemple :

```
Pour i de 1 à 2
    Écrire ("i = ", i) ;
    Pour j de 1 à 3
        Écrire ("j = ", j) ;
    Finpour
Finpour
```

boucle 2

boucle 1

4. Les instructions itératives (les boucles)

Dans l'exemple ci-dessus, chaque itération de la boucle extérieure (boucle 1) exécute la boucle intérieure (boucle 2) jusqu'à la fin avant de passer à l'itération suivante, et ainsi de suite jusqu'à la fin des deux boucles. Ainsi, le résultat d'exécution peut être représenté comme suit :

i = 1
j = 1
j = 2
j = 3
i = 2
j = 1
j = 2
j = 3

3.Les Tableaux

a) Introduction

Supposons que l'on a besoin de stocker et de manipuler les notes de 100 étudiants. On doit, par conséquent, déclarer 100 variables : n1, n2,..., n100. Vous pouvez remarquer que c'est un peu lourd de manipuler une centaine de variables (avec 100 fois de lecture/écriture...). Imaginons maintenant le cas pour une promotion de 1000 étudiants, alors là devient notre cas un vrai problème