CHAPITRE 6

LES STRUCTURES ITÉRATIVES

OBJECTIFS

- DÉFINIR LES STRUCTURES ITÉRATIVES DU LANGAGE PASCAL.
- PRÉCISER LE CONTEXTE D'UTILISATION DE CES STRUCTURES PAR LEUR MISE EN APPLICATION DANS DES SITUATIONS CONCRÈTES DE RÉSOLUTION DE PROBLÈME.

Dans ce chapitre, nous présentons les structures itératives du langage PASCAL. Par *structure* ou *instruction itérative*, nous entendons une instruction qui permet de répéter un certain nombre de fois une série d'instructions simples ou composées constituant un bloc, d'où leur appellation équivalente de *boucle*. Le langage PASCAL possède trois types d'instructions itératives ou boucles : POUR, TANT QUE et RÉPÉTER. Le fonctionnement et le contexte d'utilisation de ces instructions font l'objet des sections qui suivent.

6.1 LA STRUCTURE ITÉRATIVE « POUR »

La structure itérative POUR correspond, en PASCAL, au mot réservé FOR. Il en existe deux types dont les formats sont :

Premier format

```
FOR variable := expression1 TO expression2 DO instruction;
```

Deuxième format

```
FOR variable := expression2 DOWNTO expression1 DO instruction;
```

où *variable* est un identificateur qui désigne le nom d'une variable, *expression1* et *expression2* sont deux expressions représentant respectivement la borne inférieure et la borne supérieure de l'intervalle des valeurs que peut prendre l'identificateur *variable*, et *instruction* une instruction simple ou composée du langage.

Dans le premier format, lorsque la boucle s'exécute la première fois, *variable* prend la valeur *expression1* et tant que celle-ci reste inférieure ou égale à *expression2*, *instruction* est exécutée, puis la valeur de *variable* est augmentée de 1. La fin de la boucle est atteinte lorsque la valeur de *variable* dépasse celle de *expression2*. La figure 6.1 présente l'organigramme d'une instruction FOR dans ce format.

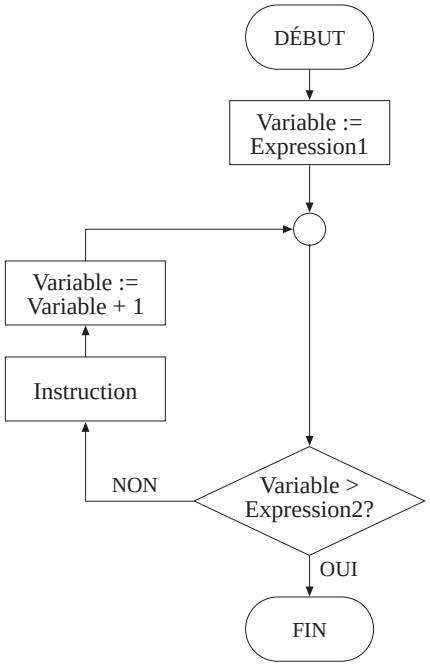

FIGURE 6.1
ORGANIGRAMME D'UNE BOUCLE FOR ... DO.

Pour illustrer notre propos, considérons l'instruction suivante :

FOR *i* := 1 TO 9 DO WRITE(*i* : 2);

Par analogie avec le premier format, on a :

variable = *i*
expression1 = 1
expression2 = 9
instruction = WRITE(*i* : 2)

Dans cette instruction, la variable *i* prend en ordre croissant des valeurs allant de 1 à 9; et pour chaque valeur de cette variable, l'instruction WRITE(*i* : 2) est exécutée, produisant ainsi le résultat suivant :

1 2 3 4 5 6 7 8 9

Considérons, à titre d'exemple, le programme de l'exemple 6.1. Dans ce programme, la variable *compte* qui contrôle l'exécution de la boucle prend des valeurs croissantes allant de 1 à 3, respectivement bornes inférieure et supérieure de variation. Quant à l'instruction qui fait l'objet de la répétition, elle est composée de la séquence d'instructions simples encadrées par le premier BEGIN suivant le mot réservé DO et le END correspondant.

Exemple 6.1

```
PROGRAM Prog61;
USES FORMS;
CONST   dix = 10;
        cinq = 5;
        deux = 2;
VAR     reel1, reel2 : REAL;
        compte, entier1, entier2 : LONGINT;

BEGIN
    READLN(entier1, entier2);
    READLN(reel2, reel1);
    FOR compte := 1 TO 3 DO
    BEGIN
        WRITELN('PASSE NUMERO', compte : 2);
        WRITELN;
        WRITELN('LA VALEUR DU PREMIER ENTIER EST ', entier1 : cinq);
        WRITELN('LA VALEUR DU DEUXIEME ENTIER EST ', entier2 : 5);
        WRITELN('LA VALEUR DU PREMIER REEL EST ', reel1 : 2*cinq : 4);
        WRITELN('LA VALEUR DU DEUXIEME REEL EST ', reel2 : dix : 4);
        WRITE('LA SOMME DES DEUX REELS EST ');
        WRITELN(reel1 + reel2 : dix : deux);
        WRITE('LA DIFFERENCE DES DEUX REELS EST ');
        WRITELN(reel1 – reel2 : dix : deux);
        WRITE('LE PRODUIT DES DEUX ENTIERS EST ');
        WRITELN(entier1 * entier2 : dix);
        WRITE('LE RESULTAT DE LA DIVISION DU REEL 1 PAR ');
        WRITELN('LE 2 EST ', reel1/reel2 : dix : 4*deux);
        WRITE('LE RESULTAT DE LA DIVISION DE L''ENTIER 1 PAR ');
        WRITELN('LE 2 EST ', entier1 DIV entier2 : deux);
        WRITE('LE RESTE DE LA DIVISION DE L''ENTIER 1 PAR ');
        WRITELN('LE 2 EST ', entier1 MOD entier2 : deux);
        WRITELN; WRITELN; WRITELN;
    END;
END.
```

Si, à l'exécution de ce programme, les deux lignes de données suivantes sont entrées :

```
44662  22330
35678.1234  45678.8766
```

la sortie du programme correspondra à :

```
PASSE NUMERO 1

LA VALEUR DU PREMIER ENTIER EST 44662
LA VALEUR DU DEUXIEME ENTIER EST 22330
LA VALEUR DU PREMIER REEL EST 45678.8766
LA VALEUR DU DEUXIEME REEL EST 35678.1234
LA SOMME DES DEUX REELS EST **81357.00
LA DIFFERENCE DES DEUX REELS EST **10000.75
LE PRODUIT DES DEUX ENTIERS EST *997302460
LE RESULTAT DE LA DIVISION DU REEL 1 PAR LE 2 EST 1.28030491
LE RESULTAT DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
LE RESTE DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2

PASSE NUMERO 2

LA VALEUR DU PREMIER ENTIER EST 44662
LA VALEUR DU DEUXIEME ENTIER EST 22330
LA VALEUR DU PREMIER REEL EST 45678.8766
LA VALEUR DU DEUXIEME REEL EST 35678.1234
LA SOMME DES DEUX REELS EST **81357.00
LA DIFFERENCE DES DEUX REELS EST **10000.75
LE PRODUIT DES DEUX ENTIERS EST *997302460
LE RESULTAT DE LA DIVISION DU REEL 1 PAR LE 2 EST 1.28030491
LE RESULTAT DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
LE RESTE DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2

PASSE NUMERO 3

LA VALEUR DU PREMIER ENTIER EST 44662
LA VALEUR DU DEUXIEME ENTIER EST 22330
LA VALEUR DU PREMIER REEL EST 45678.8766
LA VALEUR DU DEUXIEME REEL EST 35678.1234
LA SOMME DES DEUX REELS EST **81357.00
LA DIFFERENCE DES DEUX REELS EST **10000.75
LE PRODUIT DES DEUX ENTIERS EST *997302460
LE RESULTAT DE LA DIVISION DU REEL 1 PAR LE 2 EST 1.28030491
LE RESULTAT DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
LE RESTE DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
```

Selon le deuxième format de la boucle FOR, la valeur initiale de *variable* est *expression2* qui doit être cependant supérieure ou égale à *expression1*, pour que *instruction* soit exécutée. Contrairement au format précédent, chaque exécution de *instruction* fait décrémenter de 1 la valeur de *variable*, jusqu'à ce que cette dernière atteigne une valeur inférieure à *expression1*, ce qui met fin à l'exécution de cette boucle. La figure 6.2 présente l'organigramme d'une instruction FOR dans ce format.

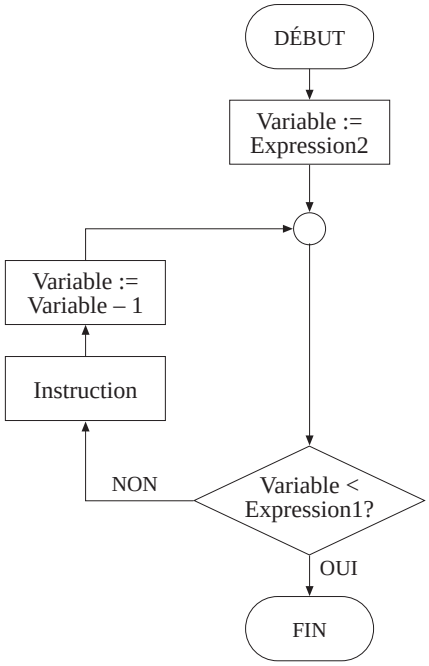

FIGURE 6.2
ORGANIGRAMME D'UNE BOUCLE FOR ... DOWNTO.

Pour illustrer notre propos, considérons l'instruction suivante :

FOR *i* := 9 DOWNTO 1 DO WRITE(*i* : 2);

Dans cette instruction, la variable *i* prend des valeurs décroissantes allant de 9 à 1; et pour chaque valeur de cette variable, l'instruction WRITE(*i* : 2) est exécutée, produisant ainsi le résultat suivant :

9 8 7 6 5 4 3 2 1

Le programme de l'exemple 6.2 permet de calculer les cotes d'une série de 10 étudiants, à partir de la note obtenue par chacun d'entre eux, laquelle est lue par le biais du clavier. Dans ce programme, la variable de contrôle *compte* prend des valeurs décroissantes allant de 10 à 1, alors que l'instruction à répéter est composée de la séquence délimitée par le premier BEGIN suivant le DO et le END correspondant. À l'intérieur de cette instruction composée, l'instruction simple :

temp := 1 + (10 – compte);

permet de déterminer l'ordre croissant de l'étudiant dont on demande la note par l'instruction suivante :

WRITELN('INDIQUEZ LA NOTE DE L"ETUDIANT', temp : 2);

En effet :

pour compte = 10, on a : temp = 1 + (10 – 10) = 1 + 0 = 1
pour compte = 9, on a : temp = 1 + (10 – 9) = 1 + 1 = 2
pour compte = 8, on a : temp = 1 + (10 – 8) = 1 + 2 = 3

et ainsi de suite.

Exemple 6.2

```
PROGRAM Notmultip;
USES FORMS;
VAR     compte, note, temp : INTEGER;
        cote : CHAR;

BEGIN
    FOR compte := 10 DOWNTO 1 DO
    BEGIN
        temp := 1 + (10 – compte);
        WRITELN('INDIQUEZ LA NOTE DE L"ETUDIANT', temp : 2);
        READLN(note);
        IF (note >= 0) AND (note <= 100) THEN
        BEGIN
            CASE (note DIV 10) OF
                                10,  9  : cote := 'A';
                                    8   : cote := 'B';
                                    7   : cote := 'C';
                                    6   : cote := 'D';
                        5, 4, 3, 2, 1, 0  : cote := 'E';
```

```
            END;
            WRITELN('LA COTE DE L''ETUDIANT EST : ', cote);
        END
        ELSE WRITELN('CETTE NOTE N''EST PAS CORRECTE!!!');
    END;

END.
```

6.2 LA STRUCTURE ITÉRATIVE « TANT QUE »

La structure itérative TANT QUE correspond en PASCAL au mot réservé WHILE. Elle a le format suivant :

WHILE condition DO instruction;

où *condition* est une expression booléenne et *instruction* une instruction simple ou composée du langage. Lorsque cette condition est vérifiée, l'exécution de l'instruction est déclenchée. Cette exécution est répétée aussi longtemps que la condition demeure vraie, conformément au cycle représenté à la figure 6.3. Lorsque cette condition devient fausse, cela consacre la fin de l'exécution de la boucle.

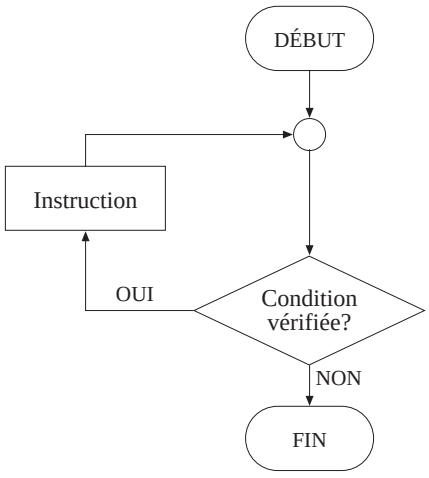

FIGURE 6.3
ORGANIGRAMME D'UNE BOUCLE TANT QUE.

Pour illustrer notre propos, considérons la séquence suivante :

```
i := 1;
WHILE (i <= 9) DO
BEGIN
        WRITE(i : 2);
        i := i + 1;
END;
```

En se référant au format de la boucle WHILE, (*i* <= 9) est la condition et la séquence :

```
BEGIN
        WRITE(i : 2);
        i := i + 1;
END;
```

l'instruction, qui dans ce cas est une instruction composée. La première instruction :

```
i := i + 1;
```

permet d'initialiser la variable *i* et, ainsi, effectuer le premier test de la condition. Toutes les fois que la condition est vérifiée, l'instruction WRITE(*i* : 2) est exécutée et la valeur de *i* incrémentée, produisant ainsi le résultat suivant :

```
1 2 3 4 5 6 7 8 9
```

Considérons le programme de l'exemple 6.3, qui est une modification du programme de l'exemple 6.1. Dans ce programme, la variable *compte* à partir de laquelle est définie la condition de la boucle (compte <= 3) est initialisée à 1. Quant à l'instruction qui fait l'objet de la répétition, elle est composée de la séquence d'instructions simples encadrées par le premier BEGIN suivant le mot réservé DO et le END correspondant.

Exemple 6.3

```
PROGRAM Prog63;
USES FORMS;
CONST   dix = 10;
        cinq = 5;
        deux = 2;
VAR     reel1, reel2 : REAL;
        compte, entier1, entier2 : LONGINT;
```

```
BEGIN
      READLN(entier1, entier2);
      READLN(reel2, reel1);
      compte := 1;
      WHILE compte <= 3 DO
      BEGIN
            WRITELN('PASSE NUMERO', compte : 2);
            WRITELN;
            WRITELN('LA VALEUR DU PREMIER ENTIER EST ', entier1 : cinq);
            WRITELN('LA VALEUR DU DEUXIEME ENTIER EST ', entier2 : 5);
            WRITELN('LA VALEUR DU PREMIER REEL EST ', reel1 : 2*cinq : 4);
            WRITELN('LA VALEUR DU DEUXIEME REEL EST ', reel2 : dix : 4);
            WRITE('LA SOMME DES DEUX REELS EST ');
            WRITELN(reel1 + reel2 : dix : deux);
            WRITE('LA DIFFERENCE DES DEUX REELS EST ');
            WRITELN(reel1 – reel2 : dix : deux);
            WRITE('LE PRODUIT DES DEUX ENTIERS EST ');
            WRITELN(entier1 * entier2 : dix);
            WRITE('LE RESULTAT DE LA DIVISION DU REEL 1 PAR ');
            WRITELN('LE 2 EST ', reel1/reel2 : dix : 4*deux);
            WRITE('LE RESULTAT DE LA DIVISION DE L''ENTIER 1 PAR ');
            WRITELN('LE 2 EST ', entier1 DIV entier2 : deux);
            WRITE('LE RESTE DE LA DIVISION DE L''ENTIER 1 PAR ');
            WRITELN('LE 2 EST ', entier1 MOD entier2 : deux);
            WRITELN; WRITELN; WRITELN;
            compte := compte + 1;
      END;
END.
```

Notez qu'à l'exécution de ce programme, pour les valeurs entrées au début, la sortie est en tout point identique à la sortie du programme de l'exemple 6.1 :

```
PASSE NUMERO 1

LA VALEUR DU PREMIER ENTIER EST 44662
LA VALEUR DU DEUXIEME ENTIER EST 22330
LA VALEUR DU PREMIER REEL EST 45678.8766
LA VALEUR DU DEUXIEME REEL EST 35678.1234
LA SOMME DES DEUX REELS EST **81357.00
LA DIFFERENCE DES DEUX REELS EST **10000.75
LE PRODUIT DES DEUX ENTIERS EST *997302460
```

```
LE RESULTAT DE LA DIVISION DU REEL 1 PAR LE 2 EST 1.28030491
LE RESULTAT DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
LE RESTE DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2


PASSE NUMERO 2

LA VALEUR DU PREMIER ENTIER EST 44662
LA VALEUR DU DEUXIEME ENTIER EST 22330
LA VALEUR DU PREMIER REEL EST 45678.8766
LA VALEUR DU DEUXIEME REEL EST 35678.1234
LA SOMME DES DEUX REELS EST **81357.00
LA DIFFERENCE DES DEUX REELS EST **10000.75
LE PRODUIT DES DEUX ENTIERS EST *997302460
LE RESULTAT DE LA DIVISION DU REEL 1 PAR LE 2 EST 1.28030491
LE RESULTAT DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
LE RESTE DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2


PASSE NUMERO 3

LA VALEUR DU PREMIER ENTIER EST 44662
LA VALEUR DU DEUXIEME ENTIER EST 22330
LA VALEUR DU PREMIER REEL EST 45678.8766
LA VALEUR DU DEUXIEME REEL EST 35678.1234
LA SOMME DES DEUX REELS EST **81357.00
LA DIFFERENCE DES DEUX REELS EST **10000.75
LE PRODUIT DES DEUX ENTIERS EST *997302460
LE RESULTAT DE LA DIVISION DU REEL 1 PAR LE 2 EST 1.28030491
LE RESULTAT DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
LE RESTE DE LA DIVISION DE L'ENTIER 1 PAR LE 2 EST *2
```

La boucle FOR du programme de l'exemple 6.2 peut être remplacée par une boucle WHILE. Pour cela, trois modifications sont nécessaires : l'initialisation de la variable *compte* à 10, l'énoncé d'une condition portant sur cette variable (compte >= 1) et la décrémentation de la variable *compte*; ce qui donne lieu au programme de l'exemple 6.4.

Exemple 6.4

```
PROGRAM Notmultip;
USES FORMS;
VAR   compte, note, temp : INTEGER;
       cote : CHAR;

BEGIN
      compte := 10;
      WHILE compte >= 1 DO
      BEGIN
            temp := 1 + (10 – compte);
            WRITELN('INDIQUEZ LA NOTE DE L"ETUDIANT', temp : 2);
            READLN(note);
            IF (note >= 0) AND (note <= 100) THEN
            BEGIN
                  CASE (note DIV 10) OF
                                          10, 9  : cote := 'A';
                                              8  : cote := 'B';
                                              7  : cote := 'C';
                                              6  : cote := 'D';
                                  5, 4, 3, 2, 1, 0  : cote := 'E';
                  END;
                  WRITELN('LA COTE DE L"ETUDIANT EST : ', cote);
            END
            ELSE WRITELN('CETTE NOTE N"EST PAS CORRECTE!!!');
            compte := compte – 1;
      END;
END.
```

6.3 LA STRUCTURE ITÉRATIVE « RÉPÉTER »

La structure itérative RÉPÉTER correspond, en PASCAL, au mot réservé REPEAT. Elle a le format suivant :

REPEAT instruction UNTIL condition;

où *condition* est une expression booléenne et *instruction* une ou plusieurs instructions simples ou composées du langage. Comme le montre l'organigramme de la figure 6.4, *instruction* s'exécute tant et aussi longtemps que *condition* reste vraie. Lorsque cette condition devient fausse, cela consacre la fin de l'exécution de la boucle.

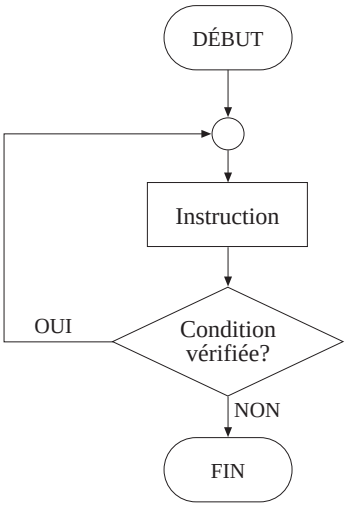

FIGURE 6.4
ORGANIGRAMME D'UNE BOUCLE REPEAT.

Pour illustrer notre propos, considérons la séquence suivante :

```
i := 1;
REPEAT
        WRITE(i : 2);
        i := i + 1;
UNTIL i > 9;
```

En se référant au format de la boucle REPEAT, ($i > 9$) est la condition et la séquence :

```
WRITE(i : 2);
  i := i + 1;
```

représente les instructions qui seront répétées jusqu'à ce que la condition devienne fausse. La première instruction :

```
i := i + 1;
```

permet d'initialiser la variable *i*. L'instruction WRITE(i : 2) est exécutée et la valeur de *i* incrémentée jusqu'à 10, produisant ainsi le résultat suivant :

1 2 3 4 5 6 7 8 9

L'exécution du programme de l'exemple 6.5 (programme modifié de l'exemple 6.3) affichera des résultats identiques à ceux du programme de l'exemple 6.3. Dans le programme, la variable *compte* à partir de laquelle est définie la condition de la boucle (compte > 3) est initialisée à 1. Les instructions situées entre le REPEAT et le UNTIL sont répétées tant et aussi longtemps que la condition est fausse.

Exemple 6.5

```
PROGRAM Prog65;
USES FORMS;
CONST   dix = 10;
        cinq = 5;
        deux = 2;
VAR     reel1, reel2 : REAL;
        compte, entier1, entier2 : LONGINT;

BEGIN
    READLN(entier1, entier2);
    READLN(reel2, reel1);
    compte := 1;
    REPEAT
        WRITELN('PASSE NUMERO', compte : 2);
        WRITELN;
        WRITELN('LA VALEUR DU PREMIER ENTIER EST ', entier1 : cinq);
        WRITELN('LA VALEUR DU DEUXIEME ENTIER EST ', entier2 : 5);
        WRITELN('LA VALEUR DU PREMIER REEL EST ', reel1 : 2*cinq : 4);
        WRITELN('LA VALEUR DU DEUXIEME REEL EST ', reel2 : dix : 4);
        WRITE('LA SOMME DES DEUX REELS EST ');
        WRITELN(reel1 + reel2 : dix : deux);
        WRITE('LA DIFFERENCE DES DEUX REELS EST ');
        WRITELN(reel1 – reel2 : dix : deux);
        WRITE('LE PRODUIT DES DEUX ENTIERS EST ');
        WRITELN(entier1 * entier2 : dix);
        WRITE('LE RESULTAT DE LA DIVISION DU REEL 1 PAR ');
        WRITELN('LE 2 EST ', reel1/reel2 : dix : 4*deux);
        WRITE('LE RESULTAT DE LA DIVISION DE L''ENTIER 1 PAR ');
        WRITELN('LE 2 EST ', entier1 DIV entier2 : deux);
        WRITE('LE RESTE DE LA DIVISION DE L''ENTIER 1 PAR ');
        WRITELN('LE 2 EST ', entier1 MOD entier2 : deux);
        WRITELN; WRITELN; WRITELN;
        compte := compte + 1;
    UNTIL compte > 3;
END.
```

La boucle WHILE du programme de l'exemple 6.4 peut être remplacée par une boucle REPEAT; nous aurons alors le programme de l'exemple 6.6.

Exemple 6.6

```
PROGRAM Notmultip;
USES FORMS;
VAR   compte, note, temp : INTEGER;
       cote : CHAR;

BEGIN
      compte := 10;
      REPEAT
            temp := 1 + (10 – compte);
            WRITELN('INDIQUEZ LA NOTE DE L"ETUDIANT', temp : 2);
            READLN(note);
            IF (note >= 0) AND (note >= 10) THEN
            BEGIN
                  CASE (note DIV 10) OF
                                          10, 9  : cote := 'A';
                                              8  : cote := 'B';
                                              7  : cote := 'C';
                                              6  : cote := 'D';
                                  5, 4, 3, 2, 1, 0  : cote := 'E';
                  END;
                  WRITELN('LA COTE DE L"ETUDIANT EST : ', cote);
            END
            ELSE WRITELN('CETTE NOTE N"EST PAS CORRECTE!!!');
            compte := compte – 1;
      UNTIL compte < 1;
END.
```

6.4 LE CONTEXTE D'UTILISATION ET L'IMBRICATION DE BOUCLES

Les programmes, que nous avons présentés dans les exemples précédents, montrent qu'il est relativement aisé de changer une structure itérative pour une autre à l'intérieur d'un même programme. Ceci semble établir une équivalence tout au moins fonctionnelle entre les trois types de boucle.

En dépit de cette constatation, il existe dans certains cas des raisons bien pratiques qui justifient l'utilisation d'une structure plutôt que d'une autre. En effet, lorsque le nombre d'itérations est connu à l'avance, il est préférable d'utiliser la boucle FOR; c'est le cas du problème d'attribution de cote à une dizaine d'étudiants d'un même cours.

Dans les cas plus fréquents où ce nombre n'est pas connu, comme lors de la consultation de fichiers sur disque, les boucles REPEAT et WHILE présentent plus d'intérêt. La différence essentielle entre REPEAT et WHILE réside dans le fait que la première exécute les instructions avant de vérifier la condition, alors que la deuxième procède inversement. Ainsi, dans un seul programme, il est courant de rencontrer les trois structures; elles sont parfois même imbriquées.

L'imbrication de boucles correspond à la situation où certaines instructions qui forment le corps d'une boucle sont elles-mêmes des boucles. Pour illustrer ce concept, nous allons présenter deux méthodes de tri parmi les plus connues : le tri par sélection et le tri par insertion.

6.4.1 Le tri par sélection

Le *tri par sélection* est une méthode qui consiste à ramener itérativement en dernière position d'une liste d'éléments le maximum de cette liste; à chaque itération, le maximum trouvé est enlevé de la liste. La notion de maximum fait référence à un ensemble ordonné qui sert de domaine aux éléments de la liste et à l'ordre croissant de tri. Un tel ensemble peut désigner, par exemple, les entiers naturels, les caractères d'un alphabet ou tout autre ensemble muni d'une relation d'ordre total. Ainsi, l'algorithme est basé sur la recherche du maximum d'une liste, et ce maximum, une fois trouvé, est retiré de la liste.

Soit à trier, en ordre croissant, la liste des six entiers de la figure 6.5. Selon la méthode de tri par sélection, on cherchera à ramener en dernière position de la liste le plus grand entier de la liste courante. Pour cela, on compare successivement l'élément d'indice i à l'élément d'indice $(i + 1)$: si élément[i] est plus grand que élément[$i + 1$], alors on permute les deux éléments et on rend inaccessible le plus petit des deux. La figure 6.6 présente les configurations obtenues au cours de la première itération, pour cinq $(6 - 1)$ comparaisons. La configuration finale ramène le plus grand entier (le maximum) de la liste courante, c'est-à-dire 24, en dernière position; ce maximum ne sera pas considéré dans la prochaine itération.

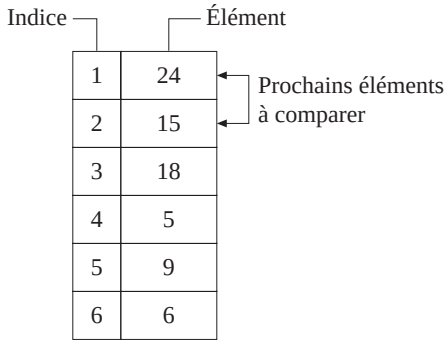

FIGURE 6.5
LISTE D'ENTIERS À TRIER PAR SÉLECTION.

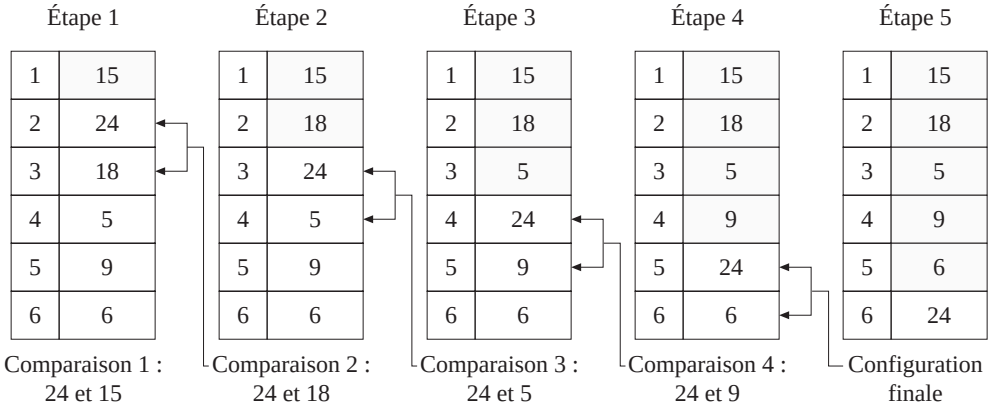

FIGURE 6.6
CONFIGURATIONS DE LA LISTE AUX DIFFÉRENTES ÉTAPES DE LA PREMIÈRE ITÉRATION.

La deuxième itération manipule une liste courante de cinq éléments, représentée à la figure 6.7. La figure 6.8 présente les configurations obtenues au cours de la deuxième itération, pour quatre (5 – 1) comparaisons. La configuration finale ramène le plus grand entier (le maximum) de la liste courante, c'est-à-dire 18, en dernière position. Ce maximum ne sera pas considéré dans la prochaine itération.

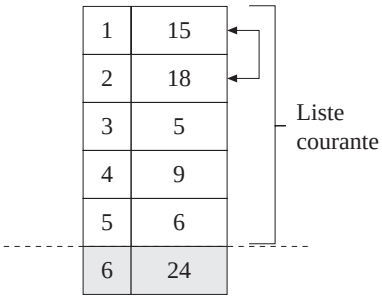

FIGURE 6.7

CONFIGURATION DE LA LISTE DES ÉLÉMENTS AU DÉBUT DE LA DEUXIÈME ITÉRATION.

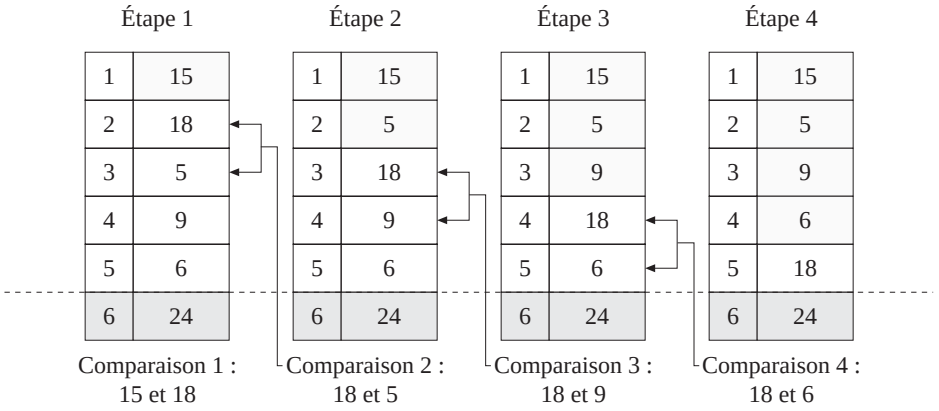

FIGURE 6.8

CONFIGURATION DE LA LISTE AUX DIFFÉRENTES ÉTAPES DE LA DEUXIÈME ITÉRATION.

La troisième itération manipule une liste courante de quatre éléments, représentée à la figure 6.9. La figure 6.10 présente les configurations obtenues au cours de la troisième itération, pour trois (4 – 1) comparaisons. La configuration finale ramène le plus grand entier (le maximum) de la liste courante, c'est-à-dire 15, en dernière position. Ce maximum ne sera pas considéré dans la prochaine itération.

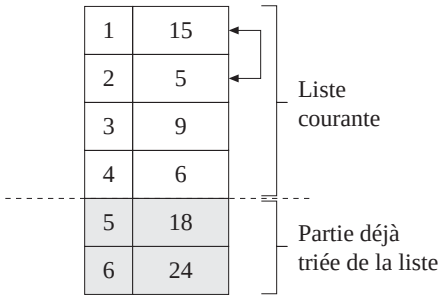

FIGURE 6.9

CONFIGURATION DE LA LISTE DES ÉLÉMENTS AU DÉBUT DE LA TROISIÈME ITÉRATION.

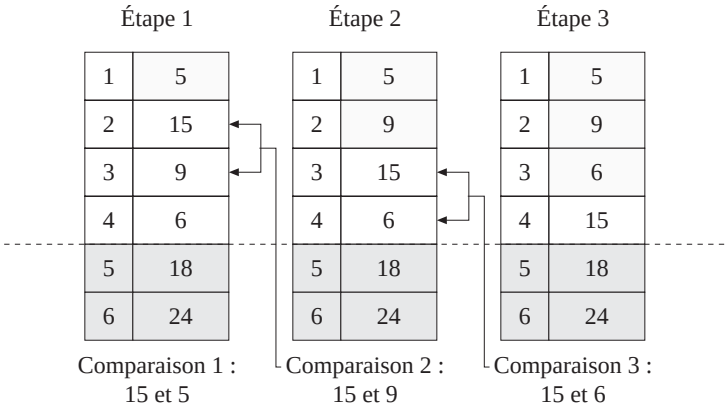

FIGURE 6.10

CONFIGURATION DE LA LISTE AUX DIFFÉRENTES ÉTAPES DE LA TROISIÈME ITÉRATION.

La quatrième itération manipule une liste courante de trois éléments, représentée à la figure 6.11. La figure 6.12 présente les configurations obtenues au cours de la quatrième itération, pour deux (3 – 1) comparaisons. La configuration finale ramène le plus grand entier (le maximum) de la liste courante, c'est-à-dire 19, en dernière position. Ce maximum ne sera pas considéré dans la prochaine itération.

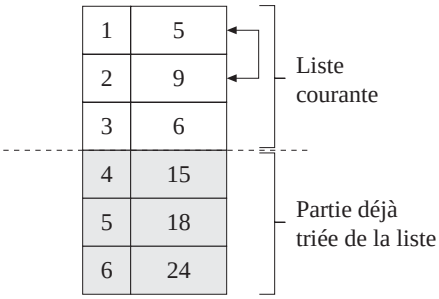

FIGURE 6.11

CONFIGURATION DE LA LISTE DES ÉLÉMENTS AU DÉBUT DE LA TROISIÈME ITÉRATION.

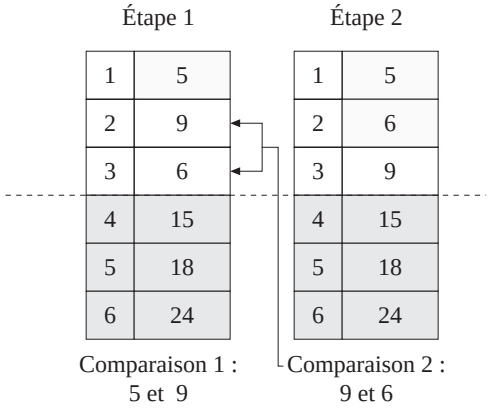

FIGURE 6.12

CONFIGURATION DE LA LISTE AUX DIFFÉRENTES ÉTAPES DE LA QUATRIÈME ITÉRATION.

La cinquième et dernière itération manipule une liste courante de deux éléments, représentée à la figure 6.13. La figure 6.14 présente les configurations obtenues au cours de la cinquième itération, pour une (2 – 1) comparaison. La seule configuration alors obtenue est finale et correspond à la liste triée recherchée.

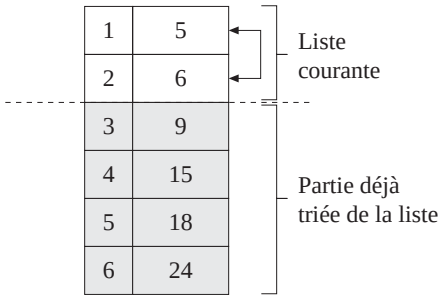

FIGURE 6.13
CONFIGURATION DE LA LISTE DES ÉLÉMENTS AU DÉBUT DE LA CINQUIÈME ITÉRATION.

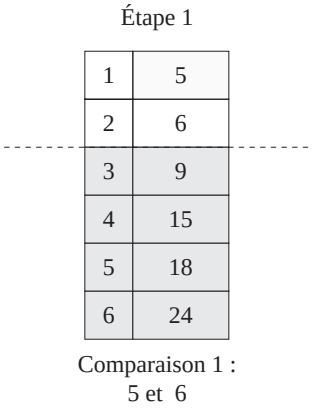

FIGURE 6.14
LISTE ENTIÈREMENT TRIÉE OBTENUE À LA FIN DE LA CINQUIÈME ITÉRATION.

Le programme de l'exemple 6.7 constitue une implantation de la méthode de tri (en ordre croissant) par sélection appliquée à une liste d'entiers. Dans ce programme comme dans tout programme en PASCAL, on notera que les paires de symboles (* et *) ou les accolades { } encadrent du texte placé à titre de commentaire.

Exemple 6.7

```
PROGRAM Triselec;
USES FORMS;
CONST
      nombremax = 25;
TYPE
      tableau = ARRAY[1..nombremax] OF INTEGER;
VAR
      lesnombres : tableau;
      minj, compteur, minx, i, j, nombre : INTEGER;

BEGIN
      (* Lecture des nombres a trier *)
      WRITELN('QUEL EST LE NOMBRE DE DONNEES A TRIER?');
      READLN(nombre);
      IF (nombre > nombremax) THENnombre := nombremax;
      WRITELN('ENTREZ LES NOMBRES');
      FOR compteur := 1 TO nombre DO
            READLN(lesnombres[compteur]);
      (* Tri des nombres lus *)
      FOR i := 1 TO (nombre – 1) DO
      BEGIN
            minj := i;
            minx := lesnombres[i];
            FOR j := (i + 1) TO nombre DO
            BEGIN
                  IF (lesnombres[j] < minx) THEN
                  BEGIN
                        minj := j;
                        minx := lesnombres[j];
                  END;
            END;
            lesnombres[minj] := lesnombres[i];
            lesnombres[i] := minx;
      END;
      (* Ecriture des nombres tries *)
      FOR i := 1 TO nombre DO WRITELN(lesnombres[i]);
END. (* Fin du programme *)
```

6.4.2 Le tri par insertion

La méthode de *tri par insertion* (ordre croissant), comme son nom l'indique, consiste à insérer chaque élément d'indice *i* de la liste à trier entre deux autres éléments d'indices respectifs *j* et (*j* + 1), tels que :

$$\text{élément}[j] \leq \text{élément}[i] \leq \text{élément}[j + 1]$$

Pour démarrer l'algorithme, il est commode d'ajouter un dernier élément à liste des éléments à trier. Cet élément fictif doit avoir une valeur plus grande que toutes celles de la liste réelle.

À titre d'exemple, soit à trier en ordre croissant la liste des entiers de la figure 6.15. Pour commencer, on se donne un élément (maximum) fictif de valeur 100 qui est placé en dernière position du tableau. Comme le montre la figure 6.16, l'insertion du premier élément de la liste à trier, soit 24, se fait à la sixième position, entre l'élément 16 et l'élément fictif 100.

Indice	Élément
1	24
2	15
3	18
4	5
5	9
6	16

FIGURE 6.15
LISTE D'ENTIERS À TRIER PAR INSERTION.

L'insertion du deuxième élément de la liste à trier, soit 15, se fait à la quatrième position, entre l'élément 9 et l'élément 16 de la dernière configuration. Le troisième élément de la liste, soit 18, est inséré à la cinquième position, entre l'élément 16 et l'élément 24 de la dernière configuration. Le quatrième, le cinquième et le sixième éléments ont gardé leur position respective étant donné que la liste était déjà complètement ordonnée. Ainsi, les dernières tentatives d'insertion n'ont donné lieu à aucun déplacement d'éléments et se révèlent donc inutiles. Un bon algorithme de tri par

insertion doit éviter de telles tentatives. Le programme de l'exemple 6.8 constitue une implantation de la méthode de tri par insertion appliquée à une liste d'entiers.

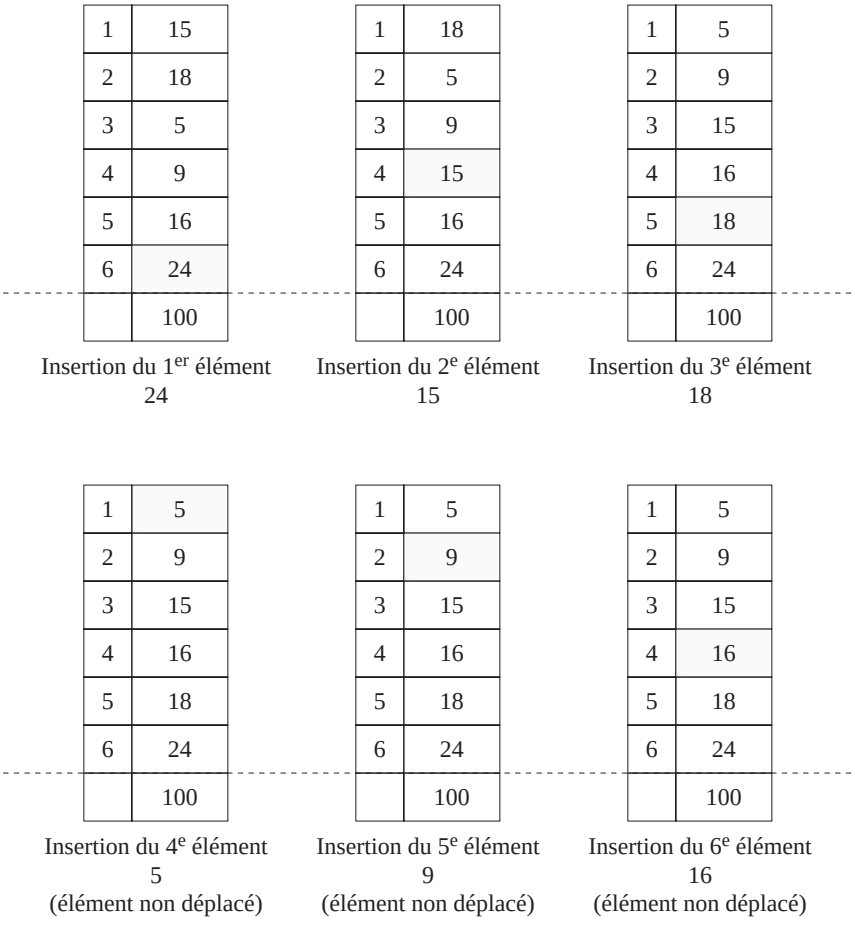

FIGURE 6.16

CONFIGURATION DE LA LISTE À L'INSERTION DES DIFFÉRENTS ÉLÉMENTS.

6.5 LA TRACE D'UN PROGRAMME

On désigne par *trace* d'un programme l'opération qui consiste à suivre instruction par instruction, à partir d'un jeu de données, le déroulement d'un programme. C'est donc une façon pratique de vérifier, sans ordinateur, le bon fonctionnement d'un programme

en tenant compte des résultats qu'il produit aux différentes étapes de son exécution. Concrètement, cela conduit à observer les valeurs prises par les différentes variables lors de l'exécution des diverses instructions du programme.

Exemple 6.8

```
PROGRAM Trinsert;
USES FORMS;
CONST
      nombremax = 25;
TYPE
      tableau = ARRAY[1..nombremax] OF INTEGER;
VAR
      tabnombres : tableau;
      indice, tampon, compteur, nombre, i, j : INTEGER;

BEGIN
       j : = 0;
      WRITELN('QUEL EST LE NOMBRE DE DONNEES A TRIER?');
      READLN(nombre);
      IF (nombre > nombremax) THEN nombre := nombremax;
      WRITELN('ENTREZ LES NOMBRES');
      FOR compteur := 1 TO nombre DO READLN(tabnombres[compteur]);
      (* Tri des nombres lus *)
      FOR indice := 2 TO nombre DO
      BEGIN
            tampon := tabnombres[indice];
            j := indice – 1;
            WHILE ((j > 0) AND (tabnombres[j] >= tampon)) DO
            BEGIN
                  tabnombres[j + 1] := tabnombres[j];
                  j := j – 1;
            END;
            tabnombres[j + 1] := tampon;
      END;
      { Ecriture des nombres a trier }
      FOR i := 1 TO nombre DO WRITELN(tabnombres[i]);
END. (* Fin du programme *)
```

Pour illustrer notre propos, considérons le programme de l'exemple 6.9 pour le calcul de la moyenne arithmétique de nombres réels et faisons une trace de ce programme en

donnant pour nombres réels 23.5, 12.5, 6.8 et 17.2. Même si le langage PASCAL ne permet pas de numéroter les lignes de programme, à des fins pédagogiques nous l'avons fait dans le programme. Comme on peut le constater, les instructions exécutables sont celles qui sont comprises exclusivement entre les lignes 5 à 17, BEGIN et END n'étant pas des instructions exécutables.

Exemple 6.9

```
PROGRAM Moyenne;                                                    {1}
USES FORMS;
VAR                                                                 {2}
      i, n : INTEGER;                                               {3}
      nombre, somme, moyenne : REAL;                                {4}

BEGIN                                                               {5}
      WRITELN('INDIQUER LE NOMBRE DE REELS');                       {6}
      READ(n);                                                      {7}
      somme := 0;                                                   {8}
      WRITELN('INDIQUER LES ', n, 'REELS');                         {9}
      FOR i := 1 TO n DO                                            {10}
      BEGIN                                                         {11}
            READ(nombre);                                           {12}
            somme := somme + nombre;                                {13}
      END;                                                          {14}
      moyenne := somme/n;                                           {15}
      WRITELN('LA MOYENNE DES REELS EST ', moyenne);                {16}
END.                                                                {17}
```

Ainsi, la trace est obtenue en relevant l'effet de chaque instruction exécutable sur les différentes variables *n*, *i*, *somme* et *moyenne*. Au départ, on considère que toutes les variables ont une valeur indéterminée. Lorsqu'une variable prend une valeur déterminée, elle la maintient jusqu'à ce que l'exécution d'une instruction vienne la modifier. Le tableau 6.1 présente la trace du programme de l'exemple 6.9. À la fin de la trace, on a pour valeur des variables :

n = 4
i = 5
nombre = 17.2
somme = 60.0
moyenne = 15.0

TABLEAU 6.1

TRACE DU PROGRAMME DE L'EXEMPLE 6.9

Numéro de ligne de l'instruction exécutée	*n*	*i*	nombre	somme	moyenne
6	–	–	–	–	–
7	4	–	–	–	–
8		–	–	0	–
9		–	–		–
10		1	–		–
12			23.5		–
13				23.5	–
10		2			–
12			12.5		–
13				36.0	–
10		3			–
12			6.8		–
13				42.8	–
10		4			–
12			17.2		–
13				60.00	–
10		5			–
15					15.0