CHAPITRE 5

LES INSTRUCTIONS DE BASE ET LES STRUCTURES DE SÉLECTION

OBJECTIFS

- PRÉSENTER LES ÉNONCÉS DE LECTURE, D'ÉCRITURE ET D'AFFECTATION DU LANGAGE PASCAL.
- DISTINGUER LES INSTRUCTIONS COMPOSÉES DES INSTRUCTIONS SIMPLES À PARTIR DESQUELLES ELLES SONT CONSTRUITES.
- DÉFINIR LES STRUCTURES DE SÉLECTION SIMPLE ET DE SÉLECTION MULTIPLE DANS LE CONTEXTE DE CE LANGAGE.

Dans tout langage de programmation, il existe un certain nombre d'instructions ou d'énoncés qui en constituent la base. C'est le cas, en PASCAL, des énoncés de lecture et d'écriture, des énoncés d'affectation, des instructions simples et composées, ainsi que des structures de sélections simple et multiple. Ces instructions de base font l'objet du présent chapitre.

5.1 LES ÉNONCÉS DE LECTURE ET D'ÉCRITURE

Le but de tout programme informatique est de produire, à partir d'un ensemble de données constituant les entrées (INPUT), un ensemble de résultats ou sortie (OUTPUT). Les données peuvent provenir aussi bien d'un clavier d'ordinateur que d'un fichier sur disque ou sur bande magnétique. Quant aux résultats, ils peuvent être affichés sur un écran, imprimés ou inscrits dans un fichier. La manipulation de ces données et résultats s'effectue par des énoncés mettant en jeu des procédures standards de lecture et d'écriture.

5.1.1 Les procédures de lecture

Le langage PASCAL comprend deux procédures standards de lecture : READ et READLN. La première a pour format général :

READ(fichier, var1, var2, ..., var*n*);

Les identificateurs qui apparaissent entre les parenthèses constituent les *paramètres* ou *arguments* de la procédure READ. Le premier paramètre, *fichier*, qui spécifie le fichier d'entrée n'est pas toujours présent. En effet, si les données sont lues à partir d'un clavier considéré comme un fichier standard (INPUT), ce paramètre devient facultatif : c'est l'option par défaut. L'exemple 5.1 en est une illustration.

Exemple 5.1

Les deux instructions suivantes sont équivalentes :

– READ(INPUT, var1, var2, ..., var*n*);

– READ(var1, var2, ..., var*n*);

Les paramètres *var1*, *var2*, ..., *var*n correspondent à des variables qui peuvent être de type entier, réel, caractère, intervalle ou même chaîne de caractères. Les valeurs respectives de ces variables, lorsqu'elles sont entrées à partir d'un clavier, doivent être séparées l'une de l'autre par au moins un espace, la fin de l'entrée étant indiquée par un retour de chariot (RC). Le nombre de valeurs entrées doit être égal au nombre de variables spécifiées comme arguments. L'exemple 5.2 montre l'effet d'un énoncé de lecture sur les variables servant d'arguments.

Mentionnons que l'énoncé READ(fichier, var1, var2, ..., var*n*) est équivalent à la séquence suivante :

```
BEGIN
    READ(fichier, var1);
    READ(fichier, var2);
    ...
    READ(fichier, varn);
END;
```

Exemple 5.2

Considérons les déclarations de variables suivantes :

```
VAR  var1 : INTEGER;
     var2 : REAL;
     var3 : CHAR;
     var4 : STRING[3];
```

Si les valeurs suivantes sont lues à partir d'un ficher *fich* :

345 3.1416 9 NON

l'énoncé :

READ(fich, var1, var2, var3, var4);

attribuera aux variables les valeurs suivantes :

var1 = 345
var2 = 3.1416
var3 = 9
var4 = 'NON'

Cependant, dans le cas où chaque variable est lue séparément, un retour de chariot est nécessaire, après chaque valeur entrée à partir du clavier, pour déclencher la lecture de celle-ci. L'exemple 5.3 montre l'effet d'une séquence d'instructions de lecture sur les variables tenant lieu d'arguments.

L'énoncé READLN permet également de lire des données à partir d'un clavier. La seule différence qui existe entre cet énoncé et READ, c'est que READLN commande un changement de ligne après la lecture de la liste des variables spécifiées comme paramètres. Ce changement de ligne ne prend effet qu'à la prochaine lecture occasionnée par un autre énoncé READ ou READLN. Ce dernier a pour format général :

READLN(fichier, var1, var2, ..., var*n*)

Exemple 5.3

Considérons les déclarations de variables suivantes :

```
VAR  var1 : INTEGER;
     var2 : REAL;
     var3 : CHAR;
     var4 : STRING[3];
```

Si les valeurs suivantes, suivies à chaque fois d'un retour de chariot, sont entrées :

```
345
3.1416
9
NON
```

l'énoncé :

```
BEGIN
    READLN(var1);
    READLN(var2);
    READLN(var3);
    READLN(var4);
END;
```

attribuera aux variables les valeurs suivantes :

```
var1 = 345
var2 = 3.1416
var3 = 9
var4 = 'NON'
```

Comme avec l'énoncé READ, les identificateurs qui apparaissent entre les parenthèses constituent les paramètres de la procédure READLN. Aussi, lorsque les données sont lues à partir d'un clavier considéré comme un fichier standard (INPUT), le premier paramètre (fichier) qui précise le fichier d'entrée devient facultatif, comme le montre l'exemple 5.4.

Exemple 5.4

Considérons les déclarations de variables suivantes :

```
VAR  var1 : INTEGER;
     var2 : REAL;
     var3 : CHAR;
     var4 : STRING[3];
```

Si les quatre lignes de valeurs suivantes sont entrées :

```
345
3.1416
9
NON
```

la séquence d'énoncés suivante :

```
READLN(var1);
READLN(var2);
READLN(var3);
READLN(var4);
```

attribuera aux variables les valeurs suivantes :

```
var1 = 345
var2 = 3.1416
var3 = 9
var4 = 'NON'
```

Mentionnons que l'énoncé READLN(fichier, var1, var2, ..., var*n*) est équivalent à la séquence suivante :

```
BEGIN
    READ(fichier, var1);
    READ(fichier, var2);
    ...
    READLN(fichier, varn);
END;
```

L'exemple 5.5 en est une illustration.

Exemple 5.5

Considérons les déclarations de variables suivantes :

```
VAR  var1 : INTEGER;
     var2 : REAL;
     var3 : CHAR;
     var4 : STRING[3];
```

Si les trois lignes de valeurs suivantes sont entrées :

```
345
3.1416   9
NON
```

la séquence d'énoncés :

```
BEGIN
   READLN(var1);
   READLN(var2, var3);
   READLN(var4);
END;
```

attribuera aux variables les valeurs suivantes :

```
var1 = 345
var2 = 3.1416
var3 = 9
var4 = 'NON'
```

5.1.2 Les procédures d'écriture

En PASCAL, il existe deux procédures standards pour afficher des données à l'écran, pour écrire celles-ci dans un fichier ou les imprimer. Il s'agit de WRITE et de WRITELN.

L'énoncé WRITE a pour format général :

WRITE(fichier, var1, var2, ..., var*n*);

Les identificateurs qui apparaissent entre les parenthèses constituent les paramètres de la procédure WRITE. Le premier paramètre, *fichier*, qui spécifie le fichier de sortie n'est pas toujours présent. En effet, si la sortie s'effectue à l'écran considéré comme un fichier standard (OUTPUT), ce paramètre devient facultatif. C'est ce que montre l'exemple 5.6.

Exemple 5.6

Les deux instructions suivantes sont équivalentes :

– WRITE(OUTPUT, var1, var2, ..., var*n*);

– WRITE(var1, var2, ..., var*n*);

Les paramètres *var1*, *var2*, ..., *var*n correspondent à des variables qui peuvent être de type entier, réel, caractère ou même chaîne de caractères. Les valeurs respectives de ces variables sont affichées à l'écran en format libre, c'est-à-dire à l'endroit où l'on se trouve sur la ligne au moment de l'écriture. Cependant, il est possible de spécifier la forme des données ou résultats à afficher en associant à chaque argument de la procédure WRITE un ou plusieurs paramètres qui contrôle la zone d'écriture.

Dans le cas d'un argument de type entier, caractère ou chaîne de caractères, on ne peut associer qu'un seul paramètre de contrôle de zone à cet argument. Ce paramètre, qui ne peut être que de type entier, spécifie le nombre minimum de caractères ou d'espaces nécessaires pour afficher ou imprimer cet argument. Dans le cas où l'espace indiqué par le paramètre se révèle insuffisant, l'argument sera tronqué de manière à pouvoir rentrer dans une plage de taille égale à celle du paramètre. Si, par contre, cet espace est plus grand que l'argument, des caractères blancs ou espaces, en l'occurrence des astérisques, sont ajoutés en avant de ce dernier à l'affichage ou à l'impression, comme l'illustre l'exemple 5.7. Il convient de noter toutefois qu'en PASCAL la troncature ne fonctionne pas avec les fichiers texte. En effet, si on ne spécifie pas un espace suffisant, Pascal utilise tout l'espace nécessaire pour écrire ce qu'il y a à écrire.

Exemple 5.7

Soit la déclaration suivante :

```
CONST   taille = 'JE NE SUIS PAS ASSEZ GRAND';
        var1 = 36745;
```

la séquence d'instructions

```
WRITE(taille : 36);
WRITE(var1 : 4);
```

provoquera comme sortie :

**********JE NE SUIS PAS ASSEZ GRAND6745

Dans cet exemple, l'entier 36 est le paramètre de contrôle de zone associé à l'argument *taille*, et 4 celui qui est associé à l'argument *var1*. Étant donné que la constante *taille* est une chaîne de caractères de longueur 26, dix espaces sont ajoutés en avant pour compléter la plage de longueur 36 qui a été prévue. Par contre, la plage de 4 étant insuffisante pour l'écriture de l'entier 36745 comportant cinq caractères, il y a eu troncature vers la gauche.

Dans le cas d'un argument de type réel, on doit associer deux paramètres de contrôle de zone à cet argument, conformément au format suivant :

WRITE(var1 : A : B);

Ces paramètres, tous deux de type entier, spécifient le nombre minimum de caractères ou d'espaces nécessaires pour afficher ou imprimer cet argument. Le premier, A, indique la longueur totale de plage nécessaire pour représenter le nombre réel au complet, c'est-à-dire la partie entière, le point décimal et la partie décimale. Quant au paramètre B, il spécifie simplement la taille que doit avoir la partie décimale; le point décimal occupe un caractère. Ainsi, le nombre de caractères de la partie entière C est donné par la relation suivante :

$$C = (A - B - 1).$$

Dans le cas où l'espace indiqué par le paramètre B est plus grand que la longueur de la partie décimale, des zéros sont ajoutés à la fin de celle-ci à l'affichage ou à l'impression. Si ce paramètre est plus petit, la partie décimale sera tronquée à partir de la droite de manière à pouvoir rentrer dans une plage de taille égale à B.

Dans le cas où l'espace indiqué par le paramètre A est plus grand que la longueur de la partie entière, des caractères blancs ou espaces, en l'occurrence des astérisques, sont ajoutés en avant de celle-ci à l'affichage ou à l'impression. Si ce paramètre est plus petit, la partie entière sera tronquée à partir de la gauche de manière à pouvoir rentrer dans une plage de taille égale à C. L'exemple 5.8 illustre différents cas d'affichage de réels.

Exemple 5.8

Soit la déclaration suivante :

```
VAR  var1, var2, var3 : REAL;
```

où var1 vaut 234.5678
var2 vaut 56789.231
var3 vaut 23.245678

la séquence d'instructions :

```
WRITE(var1 : 8 : 2);
WRITE(var2 : 9 : 5);
WRITE(var2 : 9 : 2);
WRITE(var3 : 9 : 5);
WRITE(var3 : 9 : 7);
```

provoquera comme sortie en PASCAL :

```
**234.56
789.23100
*56789.23
*23.24567
3.2456780
```

Note : Le caractère « * » égale un espace blanc.

Mentionnons que l'énoncé

WRITE(fichier, var1, var2, ..., var*n*)

est équivalent à la séquence suivante :

```
BEGIN
    WRITE(fichier, var1);
    WRITE(fichier, var2);
    WRITE(fichier, varn);
END;
```

L'énoncé WRITELN permet également d'écrire des données à l'écran, à l'imprimante ou dans un fichier. La seule différence qui existe entre cet énoncé et WRITE, c'est que

WRITELN commande un changement de ligne après l'écriture de la liste des variables spécifiées comme paramètres. Ce changement de ligne ne prend effet qu'à la prochaine écriture occasionnée par un autre énoncé WRITE ou WRITELN. Ce dernier a pour format général :

WRITELN(fichier, var1, var2, ..., var*n*);

Comme avec l'énoncé WRITE, les identificateurs qui apparaissent entre les parenthèses constituent les paramètres de la procédure WRITELN. Aussi, lorsque les données sont lues à partir d'un clavier considéré comme un fichier standard, le premier paramètre (fichier) qui spécifie le fichier de sortie devient facultatif, comme l'illustre l'exemple 5.9.

Exemple 5.9

Soit la déclaration de variables suivante :

```
VAR  var1 : INTEGER;
     var2 : REAL;
     var3 : CHAR;
     var4 : STRING[30];
```

où var1 vaut 23456
var2 vaut 234.56
var3 vaut '='
var4 vaut 'LE RESULTAT DE LA DIVISION DE'

la séquence d'instructions :

```
WRITELN(var4 : 30);
WRITE(var1 : 5, 'PAR 100 EST ', var3);
WRITE(' ', var2 : 6 : 2);
```

provoquera comme sortie en PASCAL :

```
LE RESULTAT DE LA DIVISION DE
23456 PAR 100 EST = 234.56
```

Examinons le programme de l'exemple 5.10.

Exemple 5.10

```
PROGRAM Prog51;
USES FORMS;
CONST   dix = 10;
        cinq = 5;
        deux = 2;
VAR     reel1, reel2 : REAL;
        entier1, entier2 : LONGINT;
BEGIN
    READLN(entier1, entier2);
    READLN(reel2, reel1);
    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);
END.
```

Si, au moment de l'exécution du programme de l'exemple 5.10, les deux lignes de données suivantes sont entrées :

```
44662        22330
35678.1234  45678.8766
```

la sortie du programme, en PASCAL, correspondra à :

```
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
```

5.2 LES ÉNONCÉS D'AFFECTATION

On désigne par *énoncé d'affectation* une instruction qui, comme son nom l'indique, permet d'affecter une valeur à une variable. Si on considère une variable comme une adresse en mémoire, la valeur de cette variable n'est autre que le contenu de cette adresse.

La forme générale d'une instruction ou énoncé d'affectation est :

Variable := Expression;

La notion de *variable* fait référence à un identificateur qui peut être de type entier, réel, booléen, caractère, ensemble, tableau, enregistrement ou pointeur. Le symbole := a été défini antérieurement comme l'opérateur d'affectation et signifie que l'expression constituant le membre de gauche doit être évaluée, puis affectée à la variable du membre de droite. Quant à l'*expression*, elle peut être de type arithmétique, booléen ou relationnel; elle peut aussi porter sur des fonctions mathématiques ou spéciales intégrées au langage PASCAL.

Il convient de noter que l'évaluation de toute expression doit donner lieu à une valeur de type compatible avec celui de la variable à laquelle cette expression doit être affectée. Le cas le plus évident de compatibilité de types est celui dans lequel la variable et la valeur de l'expression sont de même type.

5.2.1 L'affectation d'expressions arithmétiques

On désigne par *expression arithmétique* toute expression construite à partir d'un ou de plusieurs opérateurs arithmétiques. La liste de ces opérateurs a été dressée au chapitre 3.

Par ailleurs, il est possible de regrouper ces opérateurs en deux catégories : les *opérateurs additifs* et les *opérateurs multiplicatifs*. Dans la première catégorie, on trouve l'addition (+) et la soustraction (–), alors que la deuxième regroupe la multiplication (*), la division (/), la division entière (DIV) et le reste d'une division entière (MOD).

En l'absence de parenthèses spécifiant la priorité des opérations, une expression arithmétique est évaluée de la gauche vers la droite, en traitant d'abord les opérateurs multiplicatifs, puis les opérateurs additifs.

Considérons le programme de l'exemple 5.11. L'énoncé d'affectation donnant la valeur de la variable *result1* n'utilise pas de parenthèses. En tenant compte de la préséance des opérateurs multiplicatifs sur les additifs, il peut être réécrit de la manière suivante :

result1 := *a* + (*e* / *d*) – (*b* MOD *a*) + (*c* * *a*) – ((*d* * *b*) DIV *d*);

Exemple 5.11

```
PROGRAM Arithm;
USES FORMS;
VAR   a, b, c, d, e : INTEGER;
      result1, result2 : REAL;

BEGIN
      a := 40;
      b := 50;
      c := 50;
      d := 10;
      e := 100;
      result1 := a + e / d – b MOD a + c * a – d * b DIV d;
      result2 := (a + e)/d – b MOD a + c * (a – d) * (b DIV d);
END.
```

En remplaçant les variables par leurs valeurs respectives, on obtient :

result1 := *a* + (10.0) – (10) + (2000) – ((500) DIV 10);
result1 := 40 + 10.0 – 10 + 2000 – 50;

Par la suite, on peut dire que l'expression arithmétique vaut 1990.0, laquelle valeur est affectée à la variable *result1*.

Dans le cas de l'énoncé d'affectation mettant en jeu la variable *result2*, l'évaluation de l'expression doit tenir compte de la priorité des opérations, telle que celle-ci est indiquée par les parenthèses. En remplaçant les variables par leurs valeurs respectives, cet énoncé devient :

```
result2 := (40 + 100)/10 – 50 MOD 40 + 50 * (40 – 10) * (50 DIV 10);
result2 := (140.0)/10 – (50 MOD 40) + 50 * (30) * (5);
result2 := 14.0 – 10 + (50 * 30) * 5;
result2 := 14.0 – 10 + 1500 * 5;
```

Dans ce cas, l'expression arithmétique vaut 7504.0, qui est aussi la valeur de la variable *result2*.

5.2.2 L'affectation d'expressions booléennes

On dit d'une expression qu'elle est *booléenne* lorsqu'elle est construite à partir des constantes booléennes (TRUE, FALSE), des opérateurs booléens (NOT, AND, OR), ou des fonctions booléennes standards (ODD, EOLN, EOF). De plus, l'évaluation de cette expression doit donner comme valeur soit *vrai* (TRUE) soit *faux* (FALSE).

Tout comme les opérateurs arithmétiques, en l'absence de parenthèses, les opérateurs logiques ou booléens sont assujettis à un système de priorité. Selon ce système, la priorité la plus forte est accordée à l'opérateur de négation (NOT), puis vient l'opérateur de conjonction (AND) et enfin l'opérateur de disjonction (OR).

Considérons le programme de l'exemple 5.12. L'énoncé d'affectation donnant la valeur de la variable *result1* n'utilise pas de parenthèses. En tenant compte de la préséance des opérateurs, il peut être réécrit de la manière suivante :

```
result1 := a OR (e AND d) OR (NOT b) AND a OR (c AND a);
```

En remplaçant les variables par leurs valeurs respectives, on obtient successivement :

```
result1 := a OR (FALSE) OR (TRUE) AND a OR (FALSE);
result1 := a OR FALSE OR TRUE AND a OR FALSE;
result1 := (a OR FALSE) OR (TRUE AND a) OR FALSE;
result1 := (TRUE OR FALSE) OR (TRUE AND TRUE) OR FALSE;
```

```
result1 := (TRUE) OR (TRUE) OR FALSE;
result1 := (TRUE OR TRUE) OR FALSE;
result1 := TRUE OR FALSE;
```

Exemple 5.12

```
PROGRAM Logique;
USES FORMS;
VAR   a, b, c, d, e : BOOLEAN;
      result1, result2 : BOOLEAN;

BEGIN
      a := TRUE;
      b := FALSE;
      c := FALSE;
      d := FALSE;
      e := TRUE;
      result1 := a OR e AND d OR NOT b AND a OR c AND a;
      result2 := (a OR e) AND d OR (NOT b) AND (a OR c) AND a;
END.
```

L'évaluation de l'expression booléenne est TRUE, laquelle valeur est affectée à la variable *result1*.

Dans le cas de l'énoncé d'affectation mettant en jeu la variable *result2*, l'évaluation de l'expression doit tenir compte de la priorité des opérations indiquée par les parenthèses. En remplaçant les variables par leur valeur respective, cet énoncé devient :

```
result2 := (TRUE OR TRUE) AND FALSE OR (NOT FALSE) AND
           (TRUE OR FALSE) AND TRUE;
result2 := (TRUE) AND FALSE OR (TRUE) AND (TRUE) AND TRUE;
result2 := TRUE AND FALSE OR TRUE AND TRUE AND TRUE;
result2 := (TRUE AND FALSE) OR (TRUE AND TRUE) AND TRUE;
result2 := (FALSE) OR (TRUE) AND TRUE;
result2 := FALSE OR TRUE AND TRUE;
result2 := (FALSE) OR (TRUE AND TRUE);
result2 := (FALSE) OR (TRUE);
```

Dans ce cas, l'évaluation de l'expression booléenne est TRUE, qui est aussi la valeur de la variable *result2*.

Quant aux fonctions booléennes standards du PASCAL, on peut les regrouper en deux catégories : ODD qui est à argument de type entier, EOLN et EOF qui sont à arguments de type fichier. ODD(entier) retourne TRUE si la valeur de la variable *entier* est impaire et FALSE dans le cas contraire.

EOLN(fichier1) retourne TRUE lorsque le caractère *retour de chariot* indiquant la fin d'une ligne de *fichier1* est rencontré, alors que EOF(fichier2) retourne cette même valeur à la rencontre d'un caractère de fin de *fichier2*.

5.2.3 L'affectation d'expressions relationnelles

On désigne par *expression relationnelle* toute expression construite à partir des opérateurs relationnels et dont l'évaluation génère la valeur *vrai* (TRUE) ou *faux* (FALSE). La liste de ces opérateurs, qui servent essentiellement à établir des comparaisons entre des expressions de même type, a été présentée au chapitre 3. En PASCAL, ils sont au nombre de six et correspondent aux symboles suivants :

=	pour égal à
< >	pour différent de
<	pour inférieur à
<=	pour inférieur ou égal à
>	pour supérieur à
>=	pour supérieur ou égal à

À titre d'exemple, le programme de l'exemple 5.13 affichera comme résultat d'exécution ce qui suit :

```
false true true
```

représentant les valeurs respectives des variables booléennes *test1*, *test2* et *test3*. Ces valeurs découlent de l'évaluation des expressions relationnelles affectées à ces trois variables.

Exemple 5.13

```
PROGRAM Relation;
USES FORMS;
VAR   a, b, c, d : INTEGER;
      test1, test2, test3 : BOOLEAN;

BEGIN
      a := 40;
      b := 50;
      c := 50;
      d := 10;
      test1 := (a + b) <> (a + c);
      test2 := (a + d) = c;
      test3 := (c >= b);
      WRITELN(test1 : 6, test2 : 5, test3 : 5);
END.
```

5.2.4 L'affectation d'expressions spéciales

Dans ce contexte, nous désignons par *expression spéciale* une fonction standard du langage PASCAL qui correspond à une fonction mathématique classique ou à une fonction spécifique générant une valeur numérique. La liste de ces fonctions standards a été donnée au chapitre 4.

En exécutant le programme de l'exemple 5.14, les instructions d'écriture afficheront la ligne de résultats suivante :

```
324.6  –324  –325  16  2  15  3  2
```

Exemple 5.14

```
PROGRAM Fonction;
USES FORMS;
VAR   x, valabs : REAL;
      y, carre, racine : INTEGER;
      tronc, arrond, preced, suivant, meme : INTEGER;
```

```
BEGIN
        x := –324.6;
        y := 4;
        valabs := ABS(x);
        tronc := TRUNC(x);
        arrond := ROUND(x);
        carre := SQR(y);
        racine := ROUND(SQRT(y));
        preced := PRED(carre);
        suivant := SUCC(racine);
        meme := PRED(SUCC(racine));
        WRITE(valabs : 5 : 1, tronc : 5, arrond : 5, carre : 3);
        WRITELN(racine : 2, preced : 3, suivant : 2, meme : 2);
END.
```

5.3 LES INSTRUCTIONS COMPOSÉES

Jusqu'ici, nous n'avons examiné que des instructions simples ou élémentaires du langage PASCAL, séparées les unes des autres par un point-virgule. Toutefois, il existe également dans ce langage des instructions composées (ou blocs) constituées d'une séquence d'instructions simples encadrées par les mots clés BEGIN (début) et END (fin). La figure 5.1 présente le diagramme syntaxique d'une instruction composée. On notera qu'il peut y avoir imbrication d'instructions composées.

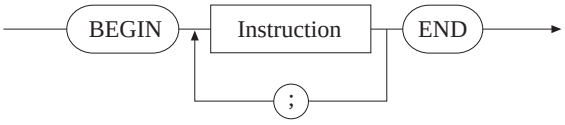

FIGURE 5.1
DIAGRAMME SYNTAXIQUE D'UNE INSTRUCTION COMPOSÉE.

5.4 LA SÉLECTION SIMPLE

On désigne par *sélection simple* une structure qui permet de tester une expression booléenne appelée *condition* qui, si elle est vraie, déclenche l'exécution d'une action. Aussi l'appelle-t-on *instruction conditionnelle*. Comme l'illustre la figure 5.2, il existe en PASCAL deux types de sélection simple. Le premier a pour format :

IF expression THEN instruction1;

alors que le deuxième se présente sous la forme suivante :

IF expression THEN instruction1 ELSE instruction2;

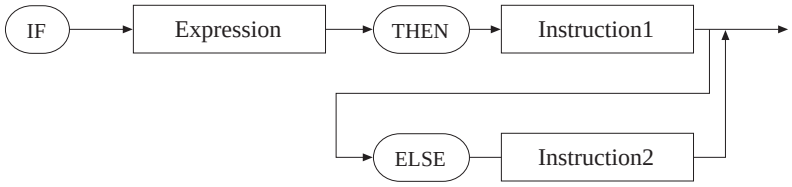

FIGURE 5.2
DIAGRAMME SYNTAXIQUE D'UNE SÉLECTION SIMPLE.

Dans ces formats, *expression* représente une condition pouvant être évaluée *vraie* ou *fausse*, *instruction1* représente une instruction simple ou composée qui n'est exécutée que si la condition exprimée par *expression* est vraie; *instruction2* est aussi une instruction simple ou composée exécutée seulement si *expression* est évaluée *fausse*. Mentionnons que la notion d'instruction doit être considérée dans son sens le plus large, pouvant même désigner une sélection simple. On parle alors d'imbrication de sélections.

Considérons le programme de l'exemple 5.15 dans lequel la sélection simple intègre une instruction composée dont le début et la fin sont indiqués par des commentaires appropriés placés entre accolades. Étant donné que la condition de déclenchement ($x > y$) n'est pas vérifiée, c'est l'instruction qui suit ELSE qui sera exécutée. Ainsi, le résultat de l'exécution du programme sera l'affichage du message suivant :

La condition n'est pas vérifiée!!!

Exemple 5.15

```
PROGRAM Selsimple;
USES FORMS;
VAR   x, valabs : REAL;
      y, carre, racine : INTEGER;
      tronc, arrond, preced, suivant, meme : INTEGER;

BEGIN
      x := –324.6;
      y := 4;
      IF x > y THEN
      BEGIN  {Debut du bloc}
           valabs := ABS(x);
           tronc := TRUNC(x);
           arrond := ROUND(x);
           carre := SQR(y);
           racine := ROUND(SQRT(y));
           preced := PRED(carre);
           suivant := SUCC(racine);
           meme := PRED(SUCC(racine));
           WRITE(valabs : 5 : 1, tronc : 5, arrond : 5, carre : 3);
           WRITELN(racine : 2, preced : 3, suivant : 2, meme : 2);
      END {Fin du bloc}
      ELSE WRITELN('LA CONDITION N''EST PAS VERIFIEE!!!');
END.
```

5.5 LA SÉLECTION MULTIPLE

Il arrive souvent que le domaine de choix offre plus de deux possibilités qui sont mutuellement exclusives, auquel cas on peut parler de *sélection multiple*. La figure 5.3 en présente le diagramme syntaxique qui correspond à la structure suivante :

```
CASE  select OF
          choix1 : instruction1;
          choix2 : instruction2;
          .....
          choixm : instructionm;
END;
```

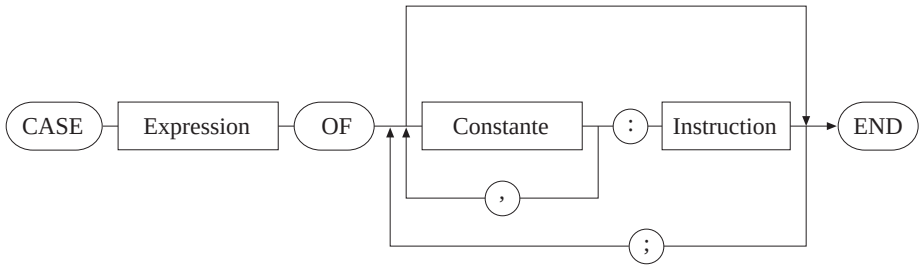

FIGURE 5.3
DIAGRAMME SYNTAXIQUE D'UNE SÉLECTION MULTIPLE.

Notons que l'identificateur *select* est une expression qui peut être de type entier, caractère ou chaîne de caractères, mais jamais de type réel. Aussi l'appelle-t-on un sélecteur. De plus, *choix1*, *choix2*, ..., *choix*m désignent des constantes qui sont du même type que le sélecteur.

Lorsque la valeur du sélecteur correspond à un des choix possibles, seule l'instruction simple ou composée correspondant à ce choix est exécutée; après quoi l'exécution se poursuit avec la plus prochaine instruction qui suit le END de la structure de sélection multiple. Si, par contre, le sélecteur ne prend aucune valeur du domaine de choix, aucune instruction de cette sélection ne sera exécutée.

Mentionnons enfin qu'il est toujours possible d'exprimer une sélection multiple sous la forme d'une série de sélections simples imbriquées, comme le montrent les programmes des exemples 5.16 et 5.17 considérés comme deux versions équivalentes. On notera qu'il n'y a pas de point-virgule (;) avant le mot clé ELSE, et que ce séparateur est facultatif lorsqu'il est suivi par un END.

Exemple 5.16

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

BEGIN
      READLN(note);
      IF (note >= 0) AND (note <= 100) THEN
```

```
        BEGIN
                CASE (note DIV 10) OF {(note DIV 10) est le selecteur}
                                        10, 9  : cote := 'A';
                                            8  : cote := 'B';
                                            7  : cote := 'C';
                                            6  : cote := 'D';
                                5, 4, 3, 2, 1, 0   : cote := 'E';
                 END;
        END
        ELSE WRITELN('CETTE NOTE N"EST PAS CORRECTE !!!');
END.
```

Exemple 5.17

Ce programme fait afficher un message demandant une note à l'usager, après avoir effacé le contenu de l'écran. Puis le programme détermine une cote (A, B, C, D ou E) correspondant à la note et affiche cette cote et la note entrée par l'usager.

```
PROGRAM Notsimple;
USES FORMS;
VAR note : INTEGER;
      cote : CHAR;

BEGIN

      WRITELN('Entrer la note (%) de l"etudiant');
      READLN(note);
      IF (note >= 0) AND (note <= 100) THEN
      BEGIN {Debut de la serie de selection simple imbriquee}
            IF note >= 90 THEN
                  cote := 'A'
            ELSE  IF note >= 80 THEN
                        cote := 'B'
                  ELSE  IF note >= 70 THEN
                              cote := 'C'
                        ELSE  IF note >= 60 THEN
                                    cote := 'D'
                              ELSE cote := 'E';
      END {Fin de la serie de selection simple imbriquee}
      ELSE WRITELN('CETTE NOTE N"EST PAS CORRECTE !!!');
      WRITELN('La note', note,  ' correspond a: ',cote);
      READLN;
END.
```