COBOL

par **Christian BONNIN**
Ingénieur informaticien de l'Institut Industriel du Nord (IDN)
Licencié ès Sciences
Expert Langages informatiques
Représentant de la France (AFNOR) à l'ISO

1.	**Normalisation**	H 2 140 -	2
2.	**Présentation du langage**	—	2
2.1	Exemple	—	2
2.2	Structure du langage	—	2
2.3	Descriptions de données	—	4
2.4	Instructions de traitement de fichiers	—	5
2.5	Instructions de traitement en mémoire centrale	—	5
	2.5.1 Instructions arithmétiques	—	5
	2.5.2 Fonctions intégrées	—	5
	2.5.3 Mouvements de données	—	5
	2.5.4 Manipulations de chaînes de caractères	—	6
	2.5.5 Instructions conditionnelles	—	6
	2.5.6 Instructions de branchements	—	7
2.6	Programmes contenus	—	7
2.7	Programmation structurée	—	7
2.8	Communications entre programmes non contenus	—	8
3.	**Fichiers COBOL**	—	8
3.1	Fichiers et accès	—	8
3.2	Fichiers relatifs	—	9
3.3	Fichiers indexés	—	9
4.	**Gestion des tables**	—	9
5.	**Éditeur (REPORT WRITER)**	—	10
6.	**Tri (SORT)**	—	10
7.	**Mise au point des programmes COBOL**	—	10
8.	**Perspectives**	—	11
8.1	COBOL orienté objet	—	11
8.2	Faut-il choisir COBOL ?	—	11
Pour en savoir plus		Doc. H 2 140	

C'est vers la fin des années 50 que le gouvernement américain, devant la diversité et l'incompatibilité des ordinateurs, langages machines et assembleurs, a demandé à des spécialistes de bâtir un langage commun, indépendant des machines, orienté vers les applications de gestion et compréhensible pour tous, informaticiens ou non.

*C'est ainsi que naquit le langage de programmation COBOL (**COmmon Business Oriented Language**), basé sur l'anglais évidemment et structuré un peu comme une œuvre littéraire sous forme de chapitres, sections, paragraphes et phrases avec verbes, mots et ponctuation.*

À l'origine, les ambitions pour COBOL étaient assez limitées. Il s'agissait essentiellement de pouvoir traiter de grands fichiers séquentiels, de les mettre à jour, de réaliser des calculs relativement simples afin d'éditer et d'imprimer des milliers de lignes d'états de paie, comptabilité, stock, etc.

Le langage a évolué, COBOL est aujourd'hui capable de traiter des bases de données en accès direct, d'échanger des informations via des lignes de transmission, de générer automatiquement des états. Mais COBOL demeure un langage pour des programmeurs avertis du fonctionnement des ordinateurs.

Des statistiques récentes (publiées par le magazine Fortune sur la base des 200 plus grosses entreprises mondiales) ont montré que le langage COBOL est très largement le plus utilisé au niveau mondial (57 %) dans l'informatique de gestion et qu'il représente d'énormes investissements financiers en développement et maintenance d'applications.

Il n'est donc pas étonnant que les experts en langage COBOL se soient penchés sur l'avenir prometteur de la programmation orientée objet. Une proposition de norme OO-COBOL est donc en cours de validation.

1. Normalisation

Le mécanisme de développement et de normalisation du langage COBOL est en fait assez complexe, bien qu'il ait été allégé récemment afin de réduire les coûts.

Précédemment, les normes portaient le label ANS (American National Standard) et s'appliquaient automatiquement à la planète sous forme de norme ISO (International Organization for Standardization).

C'est ainsi qu'une première norme ANS est apparue en 1968, suivie d'une autre en 1974 ; enfin la dernière porte la date 1985. Onze ans pour adopter une norme, c'est réellement trop long, c'est pourquoi il a été admis qu'entre deux versions seront adoptés des addenda spécifiques tels que celui des fonctions intégrées (Intrinsic functions) publié en 1991 (§ 2.5).

Aujourd'hui, le groupe COBOL SC22-X3J4 de l'ANSI conserve la prépondérance dans l'élaboration des normes mais la prochaine version devrait avoir le label ISO-1989-AAAA (1989 pour COBOL et AAAA pour l'année) et non plus ANS-AAAA. À ce jour une prochaine norme est planifiée pour 1997, elle s'appellerait donc ISO-1989-1997.

Le processus de normalisation est actuellement le suivant.

Chaque nation membre de l'ISO gère un groupe de travail COBOL (CN22-GT4 pour l'AFNOR en France) et adresse une délégation au niveau mondial à l'ISO-SC22-WG4 qui se réunit officiellement au moins deux fois par an, pendant une semaine, pour entériner ou non les propositions de modifications ou extensions de la norme.

Les propositions sont d'abord établies sous la forme de documents de travail explicitant le pourquoi et le comment, adressés au groupe ANSI-X3J4 qui les étudie. Après étude détaillée et amendements par les experts de l'ANSI, la proposition est soumise au groupe ISO-WG4 avant enquête formelle auprès des pays membres de l'ISO pour approbation. Cette phase d'étude est très longue car il faut tenir compte de la syntaxe, des matériels, de la compatibilité, etc. La France s'est distinguée, par exemple en 1993, avec deux propositions adoptées, la simplification de l'instruction STRING et surtout l'adjonction de l'allocation dynamique pour les tables internes.

Jusqu'en 1985, la norme se présentait sous forme de modules, par exemple le module séquentiel indexé, mais il a été décidé de démodulariser la norme afin d'éviter les redondances d'un module à l'autre. Cette tâche est finalement beaucoup plus lourde que prévue, ce qui laisse mal augurer de l'objectif 1997 pour la future norme.

Par ailleurs, on peut se poser des questions sur l'opportunité de cette démodularisation car, par exemple, le futur module Report Writer, très intelligent, en cours d'élaboration, va se trouver éparpillé dans la norme sans homogénéité et pratiquement inintelligible.

Enfin, il est envisagé d'adjoindre à la norme un module Orienté Objet. Une proposition formelle d'OO-COBOL est en cours de validation prioritaire. Mais la complexité du sujet risque de repousser la norme prévue pour 1997 à 1998. Cela n'empêche pas certaines compagnies (MicroFocus par exemple) de proposer des extensions hors norme telles que l'OO-COBOL ou la gestion des écrans de visualisation en espérant que ces extensions seront avalisées par la future norme.

2. Présentation du langage

2.1 Exemple

C'est en nous appuyant sur un exemple que nous allons examiner les caractéristiques essentielles de COBOL.

Le programme choisi pour exemple (figure **1**) consiste à créer un fichier de documentation, en organisation séquentielle indexée, dont les enregistrements ont le format présenté figure **2**. PRV1, PRV2, PRV3 correspondent à des prix de cessions différents pour des commandes ordinaires (PRV1), commandes par quantité (PRV2) et commandes internes (PRV3). Un top 1, 2 ou 3 en tête des valeurs indique si le tarif correspondant est applicable.

Le principe de traitement de ce programme est classique (figure **3**). Dans le paragraphe DEBUT, on ouvre le fichier séquentiel en lecture FILECTU et le fichier séquentiel indexé en sortie FISECIN que l'on veut créer ; on ouvre aussi un fichier d'impression en sortie FICHIMP pour éditer les messages d'erreurs (ligne 61 du programme).

On lit un premier enregistrement du fichier FILECTU en s'assurant qu'il n'est pas vide (lignes 62-63), puis on traite la suite des enregistrements du fichier FILECTU dans le paragraphe BOUCLE (ligne 64 et lignes 71 à 77).

En fin de traitement, lorsqu'il n'y a plus d'enregistrements du fichier FILECTU à traiter, on ferme les fichiers et on arrête le traitement dans le paragraphe FINTRA (lignes 66 à 69).

2.2 Structure du langage

Autrefois, un programme COBOL s'écrivait sur une feuille de programmation appelée bordereau de perforation pour la transcription des instructions sur cartes perforées. Chaque ligne d'instruction COBOL représente une carte perforée de 80 colonnes.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. CREFICIN.

 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. IBM-S34.
 OBJECT-COMPUTER. IBM-S34.

*************************************************************
*                                                           *
*  CREATION D'UN FICHIER SEQUENTIEL INDEXE FISECIN          *
*  EN ENTREE: FILECTU: FICHIER SEQUENTIEL                   *
*  EN SORTIE: FISECIN: FICHIER SEQUENTIEL INDEXE            *
*             FICHIMP: FICHIER EDITION DES ANOMALIES        *
*                                                           *
*************************************************************

 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT FILECTU ASSIGN TO DISK-FILECTU.
     SELECT FISECIN ASSIGN TO DISK-FISECIN
                    ORGANIZATION INDEXED
                    ACCESS SEQUENTIAL
                    RECORD KEY CODREF
                    FILE STATUS FISTAT.
     SELECT FICHIMP ASSIGN TO PRINTER-FICHIMP.

 DATA DIVISION.
 FILE SECTION.
 FD   FILECTU LABEL RECORD STANDARD
              DATA RECORD ENTREE.
 01   ENTREE.
      02 FILLER PIC X.
      02 REF    PIC X(9).
      02 REST1  PIC X(118).

 FD   FISECIN LABEL RECORD STANDARD
              DATA RECORD SORTIE.
 01   SORTIE.
      02 FLAGS  PIC X.
      02 CODREF PIC X(9).
      02 REST2  PIC X(118).

 FD   FICHIMP LABEL RECORD OMITTED
              DATA RECORD LIGNE.
 01   LIGNE     PIC X(132).

 WORKING-STORAGE SECTION.
 01   DIVERS.
      02 EOF    PIC X  VALUE '0'.
      02 FISTAT PIC XX VALUE '00'.

 01   LIDET.
      02 FILLER PIC X(30) VALUE 'ERREUR ECRITURE, STATUS='.
      02 STATED PIC X(10).
      02 FILLER PIC X(30) VALUE 'INDIC ENREGISTREMENT='.
      02 CLESED PIC X(62).

 PROCEDURE DIVISION.
 DEBUT.
     OPEN INPUT FILECTU OUTPUT FISECIN FICHIMP
     READ FILECTU AT END DISPLAY 'FICHIER FILECTU VIDE'
                         PERFORM FINTRA.
     PERFORM BOUCLE THRU FINBOUC UNTIL EOF = '1'.

 FINTRA.
     CLOSE FILECTU FISECIN FICHIMP
     DISPLAY 'FIN DU PROGRAMME CREFICIN'
     STOP RUN.

 BOUCLE.
     MOVE SPACE TO FLAGG
     MOVE REF TO CODREF
     MOVE REST1 TO REST2
     WRITE SORTIE INVALID KEY PERFORM ERREUR.
     IF FISTAT NOT = '00' PERFORM ERREUR.
     READ FILECTU AT END MOVE '1' TO EOF.
 FINBOUC.
     EXIT.

 ERREUR.
     MOVE FISTAT TO STATED MOVE '00' TO FISTAT
     MOVE REF TO CLESED
     WRITE LIGNE FROM LIDET AFTER 2.
```

Figure 1 – Exemple : création d'un fichier de documentation

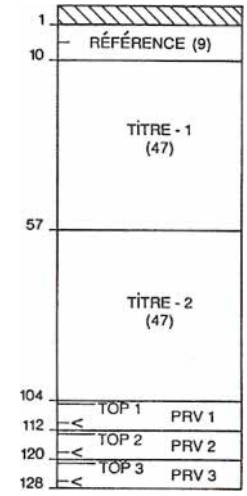

Figure 2 – Enregistrement d'un fichier séquentiel indexé

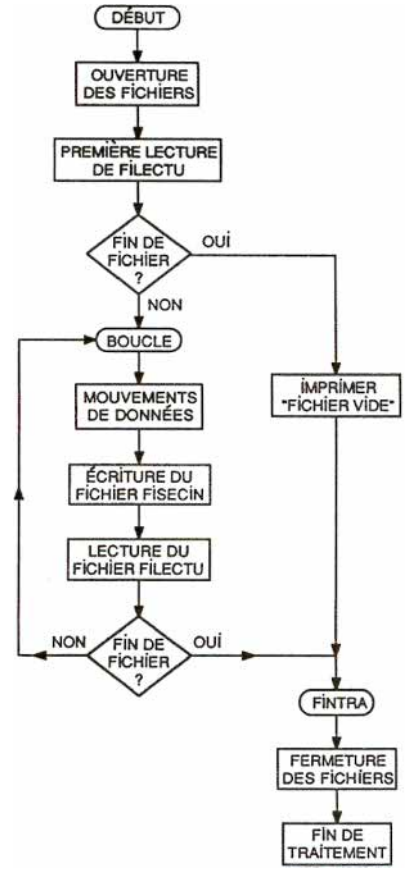

Figure 3 – Organigramme de traitement de l'exemple

— Dans les colonnes 1 à 6 on numérote les lignes (autrefois les cartes) de programme sous la forme de numéro de page et numéro de ligne. Les compilateurs modernes ignorent ces numéros mais respectent l'alignement des instructions à partir des positions 7, 8 et 12.

— La colonne 7 est utilisée en inscrivant :

• un tiret (-) pour indiquer qu'une information (adresse symbolique, constante, mot réservé, etc.) n'a pas pu être mise en entier sur la ligne précédente ;

• un caractère ⋆ ou / pour indiquer que la ligne comporte un commentaire qui ne sera évidemment pas pris en considération par le compilateur pour générer le programme en langage machine (lignes 9 à 16 de l'exemple).

— Les colonnes 8 à 72 sont utilisées pour l'écriture du programme proprement dit. Les noms de divisions, sections, paragraphes sont écrits à partir de la marge A (colonne 8). Les instructions COBOL ne peuvent être écrites qu'après la marge B (colonne 12).

— Les colonnes 73 à 80, autrefois utilisées pour identifier un paquet de cartes, n'ont plus d'intérêt avec les machines modernes.

Un programme COBOL est divisé en quatre grands chapitres appelés DIVISIONS.

● **IDENTIFICATION DIVISION** sert à définir le nom du programme tel qu'il sera reconnu par le système d'exploitation de l'ordinateur. Dans notre exemple (ligne 02), le programme s'appelle CREFICIN.

On peut aussi éventuellement indiquer dans cette division certaines informations d'ordre pratique telles que le nom du programmeur, la date d'écriture du programme, etc. Mais ces informations ne sont pas indispensables.

● **ENVIRONMENT DIVISION** permet de désigner les informations se rapportant à l'environnement dans lequel le programme est exécuté. Lorsque l'on passe d'un compilateur à un autre, cette division doit généralement être modifiée puisqu'elle est très dépendante de la machine et du compilateur.

Dans notre exemple, on désigne la machine de compilation (SOURCE-COMPUTER) et la machine d'exécution (OBJECT-COMPUTER) dans la CONFIGURATION SECTION. Ensuite, après une zone commentaire entourée d'astérisques (parce qu'il faut un ⋆ en colonne 7 pour une zone commentaire), on déclare dans l'INPUT-OUTPUT SECTION les fichiers qui seront utilisés dans le programme.

● **DATA DIVISION** est la division des instructions déclaratives des zones de données réservées en mémoire centrale pour le traitement du programme. À l'intérieur de cette division, on distingue deux sections mais il peut y en avoir d'autres plus spécialisées. Dans notre exemple, nous trouvons deux sections :

— FILE SECTION (section des fichiers) avec la description des caractéristiques des enregistrements et des zones de communication des unités périphériques avec la mémoire centrale, c'est-à-dire les zones de mémoire centrale dans lesquelles le périphérique délivre ses enregistrements ; à partir de la ligne 30, on trouve la description des caractéristiques du fichier appelé FILECTU et des enregistrements qu'il contient (ENTREE) ;

— WORKING-STORAGE SECTION (section des mémoires de travail) où sont décrites les zones de mouvement en mémoire centrale. Comme en FILE SECTION, les données sont référencées par un nom symbolique. C'est ainsi qu'à la ligne 34 on trouve la description d'une donnée REF comportant 9 caractères alphanumériques.

● **PROCEDURE DIVISION** est, comme son nom l'indique, la division du traitement proprement dit.

Il faut d'abord rappeler que chaque division est composée de sections, elles-mêmes composées de paragraphes contenant enfin des phrases qui sont les instructions du programme COBOL.

Chaque division, section ou paragraphe doit être doté d'un nom symbolique écrit en marge A, c'est-à-dire la position 8 d'une ligne de programme COBOL. À l'intérieur d'un paragraphe, les phrases sont écrites librement à partir de la marge B entre les positions 12 et 72. Mais pour des raisons de clarté du texte des instructions, les programmeurs s'imposent généralement des règles d'alignement comme on peut le voir dans notre exemple de programme.

Les instructions COBOL sont des phrases de longueurs variables où les codes opérations sont représentés par des mots clés anglais (traditionnellement les mots clés sont soulignés dans la littérature COBOL, mais pas dans les programmes), par exemple :

— ADD pour additionner (*ADD* BASE *TO* ZOBRUT) ; ADD et TO sont des mots clés COBOL alors que BASE et ZOBRUT sont des adresses symboliques de zones de données appelées dans la pratique Nom-Données ;

— MOVE pour un mouvement de mémoire à mémoire (*MOVE* INDIC *TO* LU) ;

— COMPUTE pour calculer une expression (*COMPUTE* BRUT = BASE + (PH ⋆ HS) ; ici ⋆ signifie multiplier) ;

— READ pour lire un fichier (*READ* FICART) ;

— WRITE pour écrire un enregistrement dans un fichier (*WRITE* IMPRI).

Les mots clés, réservés aux codes opérations, ne peuvent bien sûr être utilisés comme adresses symboliques de Nom-Données, et cela peut constituer un avantage pour le Français qui risquera peu d'utiliser un mot anglais comme nom symbolique.

Il faut aussi noter l'importance particulière de la ponctuation en langage COBOL, par exemple la fin de la clause SELECT (lignes 21 à 25) est désignée par le point en fin de ligne 25. Derrière chaque nom de division, section, paragraphe, il faut accoler un point.

Pour la PROCEDURE DIVISION, il n'est pas indispensable de mettre un point entre des instructions exploitées en séquence normale (exemple lignes 72 à 74) mais il faut mettre obligatoirement un point en fin de paragraphe (ligne 69 pour la fin du paragraphe FINTRA). Il faut aussi mettre un point en cas de rupture de la séquence de traitement, c'est ce que nous verrons avec les instructions conditionnelles.

En COBOL, il n'existe qu'un seul mot clé de calcul (ADD, MULTIPLY...) quel que soit le format des données (binaire, flottant, condensé, etc.) car le compilateur COBOL prend en charge toutes les conversions nécessaires de formats et zones intermédiaires de calcul.

Tout cela dégage considérablement le programmeur des soucis liés à la machine dans l'écriture de la procédure mais, en contrepartie, cela impose au préalable une description très complète des zones de données sous forme de structures.

2.3 Descriptions de données

Une **structure** est un système hiérarchique de noms qui décrit une zone de mémoire interne. Au premier niveau (01), un seul nom (le nom de structure principale) englobe tous les éléments de données mémorisés dans la totalité de la zone de mémoire attribuée à cette structure. Au niveau suivant (02), il est attribué de nouveaux noms à certaines portions de la zone. Le processus se poursuit jusqu'à ce que, au dernier niveau, un nom désigne un seul élément de donnée élémentaire.

Par **exemple**, une date composée de JOUR, MOIS, AN donnerait la structure :

```
01 DATE.
   02 JOUR...
   02 MOIS...
   02 AN...
```

DATE désigne la zone globale de six caractères comportant le JOUR (2 caractères), le MOIS (2 caractères) et l'AN (2 caractères).

Chaque donnée élémentaire est décrite en donnant :

— son image PICTURE ou PIC en anglais :

• A pour un caractère alphabétique,
• X pour un caractère alphanumérique,
• 9 pour un caractère numérique de calcul,

par exemple un nombre de 5 chiffres sera décrit PIC 99999 ou PIC 9(5) ;

— son format :

• DISPLAY pour des caractères en code décimal externe, c'est le format pris par défaut,

• BINARY, COMPUTATIONAL ou COMP pour un nombre binaire pur,

• COMPUTATIONAL-1 ou COMP-1 pour la virgule flottante sur 1 mot de 32 bits,

• COMPUTATIONAL-2 ou COMP-2 pour la virgule flottante sur 2 mots, soit 64 bits,

• PACKED-DECIMAL, COMPUTATIONAL-3 ou COMP-3 pour le décimal condensé interne.

Par **exemple**, on décrira la zone DATE contenant 13-12-86 de la façon suivante :

```
01 DATE
    02 JOUR  PIC  99  VALUE  '13'.
    02 MOIS  PIC  99  VALUE  '12'.
    02 AN    PIC  99  VALUE  '86'.
```

Dans cet exemple, le format DISPLAY est sous-entendu par défaut.

2.4 Instructions de traitement de fichiers

Ce sont les instructions qui permettent d'ouvrir un fichier OPEN en lecture INPUT ou en écriture OUTPUT (ligne 61 de notre exemple). Le pendant est l'instruction CLOSE de fermeture (ligne 67).

Les enregistrements de fichiers en lecture sont rendus disponibles en mémoire centrale, dans les zones décrites en FILE SECTION par un ordre READ *nom-de-fichier*.

Par **exemple** READ FILECTU (ligne 62) alimente la zone ENTREE (lignes 32 à 35). On écrit un enregistrement dans un fichier par un ordre WRITE *nom-d'enregistrement*. Dans notre exemple, WRITE SORTIE (ligne 75) écrit sur le fichier FISECIN la zone SORTIE (lignes 39 à 42).

On utilise READ *nom-de-fichier* parce qu'un fichier peut posséder plusieurs types d'enregistrements et que l'on ignore *a priori* lequel va se présenter à la lecture. Par contre on écrit WRITE *nom-d'enregistrement* pour bien préciser quelle zone on veut inscrire sur le fichier.

Pour éditer un fichier d'impression, on utilise également l'ordre WRITE avec la possibilité de changer de page ou de sauter un certain nombre de lignes blanches.

Par **exemple** WRITE LIGNE AFTER 2 précise que l'on écrit une ligne après un saut de 2 lignes blanches.

Lorsque l'on traite un fichier en accès direct, on peut supprimer un enregistrement DELETE ou réécrire un enregistrement REWRITE précédemment lu avec READ.

On utilise aussi les instructions ACCEPT pour recevoir et DISPLAY pour échanger des messages avec la console de l'ordinateur.

2.5 Instructions de traitement en mémoire centrale

Ces instructions manipulent (calculs, transferts, comparaisons) des données en mémoire centrale de travail (WORKING-STORAGE) ou de communication avec les fichiers (FILE SECTION).

2.5.1 Instructions arithmétiques

Elles sont de deux sortes :

— les opérations simples ADD addition, SUBSTRACT soustraction, MULTIPLY multiplication et DIVIDE division,

Par **exemple** :

```
ADD QTE1 QTE2 GIVING TOTAL
```

Ajouter QTE1 et QTE2 pour donner TOTAL ;

— les expressions arithmétiques avec l'ordre calculer COMPUTE et les opérateurs + addition, – soustraction, ⋆ multiplication, / division et ⋆⋆ puissance,

Par **exemple** :

```
COMPUTE DELTA = (B ⋆⋆ 2) – (4 ⋆ A ⋆ C)
```

pour la célèbre formule $\Delta = B^2 - 4AC$.

Un énorme avantage fourni par COBOL est d'affranchir le programmeur des conversions de formats de données.

Par **exemple** si B est en binaire, A en décimal externe et C en décimal condensé, toutes les conversions sont automatiquement réalisées pour fournir DELTA en virgule flottante si tel est le cas.

2.5.2 Fonctions intégrées

Les processus de normalisation étant très longs (de cinq à dix ans), il a été décidé de publier des addenda à la norme existante, aujourd'hui ANSI 1985, en période intermédiaire, lorsqu'il existait un consensus général.

C'est ainsi qu'a été publié en 1991 l'addendum des fonctions intégrées « Intrinsic functions » (tableau 1) adopté en 1989.

En fait, ces fonctions sont semblables à celles de FORTRAN et PL/I.

A priori, en gestion, les fonctions les plus intéressantes sont les fonctions qui traitent les dates et les chaînes de caractères.

Il est couramment admis que les fonctions intégrées apportent un gain de productivité de l'ordre de 25 % lorsqu'il s'agit de programmer des procédures équivalentes en COBOL classique.

Il faut prendre garde que les fonctions intégrées n'ont été adoptées par CODASYL qu'en 1991. Certains compilateurs en sont dotés mais pas toujours avec une fiabilité parfaite.

Un compilateur se distingue particulièrement des autres, celui de MicroFocus qui fonctionne sur ordinateur personnel. Il permet de développer, compiler et tester sur l'ordinateur personnel puis de transférer le programme sur un grand ordinateur dont le COBOL (IBM en particulier) est compatible.

2.5.3 Mouvements de données

L'instruction la plus utilisée est l'instruction MOVE. Elle permet de transférer le contenu d'une zone TOTAL1 dans une zone TOTAL2 en écrivant MOVE TOTAL1 TO TOTAL2. Comme pour les instructions arithmétiques, les conversions de formats numériques sont réalisées automatiquement.

On peut également réaliser l'affectation directe d'une valeur ou d'un littéral alphanumérique dans une zone de données.

Par **exemple** MOVE « DECEMBRE » TO MOIS inscrit le littéral DECEMBRE dans la zone MOIS.

Enfin, il est possible d'affecter des valeurs grâce à des constantes figuratives telles que ZERO et SPACE qui permettent d'affecter la valeur zéro à une donnée numérique ou encore de mettre des zéros ou des espaces dans une zone alphanumérique.

Tableau 1 – Nouvelles fonctions intégrées (BUILT-IN)

FORMAT GÉNÉRAL
FUNCTION Nom-Fonction [(Argument-1 [Argument-i]...)]

ACOS (Arc cosinus)	Valeur en radians de l'arc cosinus.
ANNUITY (Arg-1 Arg-2)	Ratio des annuités sur Arg-2 périodes pour un taux Arg-1.
ASIN (arc sinus)	Valeur en radians de l'arc sinus.
ATAN (Arc tangente)	Valeur en radians de l'arc tangente.
CHAR (Arg-1)	Caractère de position Arg-1 de la collating séquence (1).
COS (Arg-1)	Cosinus de l'arc Arg-1 exprimé en radians.
CURRENT-DATE	Date et heure présentes.
DATE-OF-INTEGER (Arg-1)	Date YYYYMMDD correspondant au nombre de jours Arg-1 depuis le 31/12/1600.
DAY-OF-INTEGER (Arg-1)	Date YYYYDDD correspondant au nombre de jours Arg-1 depuis le 31/12/1600.
FACTORIAL (Arg-1)	Factorielle du nombre Arg-1.
INTEGER (ARG-1)	Plus grande valeur entière de Arg-1.
INTEGER-OF-DATE (Arg-1)	Nombre de jours entre une Date Arg-1 = YYYYMMDD et le 31/12/1600.
INTEGER-OF-DAY (Arg-1)	Nombre de jours entre une Date Arg-1 = YYYYDDD et le 31/12/1600.
INTEGER-PART (Arg-1)	Partie entière de Arg-1.
LENGHT (Arg-1)	Nombre de caractères d'une chaîne Arg-1.
LOG (Arg-1)	Logarithme à base « e » de Arg-1.
LOG10 (Arg-1)	Logarithme à base 10 de Arg-1.
LOWER-CASE (Arg-1)	Transforme une chaîne Arg-1 en caractères minuscules.
MAX (Arg-1 [Arg-i] ...)	Valeur maximale de la liste Arg-i.
MEAN (Arg-1 [Arg-i] ...)	Moyenne arithmétique des Arg-i.
MEDIAN (Arg-1 [Arg-i] ...)	Valeur de l'Arg-m, milieu de la liste des Arg-i.
MIDRANGE (Arg-1 [Arg-i] ...)	Moyenne des 2 valeurs extrêmes de Arg-i (mini et maxi).
MIN (Arg-1 [Arg-i] ...)	Plus petite valeur des Arg-i.
MOD (Arg-1 Arg-2)	Résultat de (Arg-1 modulo Arg-2).
NUMVAL (Arg-1)	Valeur numérique d'une chaîne numérique éditée Arg-1.
NUMVAL-C (Arg-1 [Arg-2])	Valeur numérique d'une chaîne numérique éditée Arg-1 comprenant des signes et caractères monétaires.
NUMVAL-F (Arg-1)	Valeur numérique d'une chaîne numérique éditée Arg-1 en virgule flottante.
ORD (Arg-1)	Position ordinale du caractère Arg-1 dans la collating séquence.
ORD-MAX (Arg-1 [Arg-i] ...)	Position ordinale maximale de Arg-i dans la collating séquence.
ORD-MIN (Arg-1 [Arg-i] ...)	Position ordinale minimale de Arg-i dans la collating séquence.
PRESENT-VALUE (Arg-1 Arg-2 Arg-i ...)	Valeur finale pour des périodicités Arg-2, Arg-i pour un taux Arg-1.
RANDOM (Arg-1)	Nombre aléatoire calculé à partir de Arg-1.
RANGE (Arg-1 [Arg-i] ...)	Différence entre Arg-M (maximum) et Arg-m (minimum).
REM (Arg-1 Arg-2)	Reste de la division de Arg-1 par Arg-2.
REVERSE (Arg-1)	Inverse l'ordre des caractères d'une chaîne Arg-1.
SIN (Arg-1)	Sinus d'un angle Arg-1 en radians.
SQRT (Arg-1)	Racine carrée de Arg-1.
STANDARD-DEVIATION (Arg-1 [Arg-i] ...)	Déviation standard des nombres Arg-i.
SUM (Arg-1 [Arg-i] ...)	Somme algébrique des Arg-i.
TAN (Arg-1)	Tangente de l'arc Arg-1 exprimé en radians.
UPPER-CASE (Arg-1)	Transforme une chaîne Arg-1 en caractères majuscules.
VARIANCE (Arg-1 [Arg-i] ...)	Variance des Arg-i = carré de la déviation standard.
WHEN-COMPILED	Date et Heure de compilation du programme.

(1) Collating sequence : codification de base de l'ordinateur (EBCDIC, ASCII, etc.)

2.5.4 Manipulations de chaînes de caractères

Depuis la norme COBOL 74, il est devenu possible de manipuler des chaînes de caractères alphanumériques. On dispose pour cela du mot clé STRING qui permet de concaténer plusieurs zones de données dans une seule zone et du mot clé UNSTRING qui, symétriquement, permet de faire éclater une zone de données en plusieurs zones élémentaires. On peut noter que C. Bonnin a obtenu une simplification de STRING pour la prochaine norme.

2.5.5 Instructions conditionnelles

Au cours du déroulement d'un programme, on est très souvent amené à prendre, en fonction d'une condition, des décisions qui influent sur la suite du traitement, par exemple effectuer ou non un calcul suivant que la condition est vérifiée ou non. COBOL permet d'effectuer de tels choix à l'aide de l'instruction conditionnelle IF.

Exemple : IF A = B GO TO CALCUL1 ELSE GO TO CALCUL2.

Si A = B aller au paragraphe CALCUL1 sinon aller au paragraphe CALCUL2. Il existe une forme simplifiée (ligne 76 du programme) (cf. § 2.5.6) où la ponctuation joue un rôle primordial car si FISTAT n'a

pas la valeur 00, il faut effectuer le paragraphe ERREUR (lignes 81 à 84), dans le cas contraire, la présence du point indique qu'il faut simplement passer en séquence à l'instruction suivante (ligne 77).

2.5.6 Instructions de branchements

Outre l'instruction GO TO qui désigne explicitement l'instruction (le nom de paragraphe) qui doit être exécutée ensuite, il existe aussi les ordres PERFORM, EXIT et STOP.

Il arrive fréquemment qu'une même séquence d'instructions intervienne à différents endroits dans le cours d'un programme. Il vient alors immédiatement à l'esprit de n'écrire qu'une seule fois une séquence répétitive et d'y dérouter le traitement à chaque fois que le besoin s'en fait sentir.

Une telle séquence d'instructions peut avoir été écrite et compilée séparément du programme principal COBOL ; on l'appelle alors un sous-programme externe, auquel il est fait référence par une instruction CALL.

La séquence d'instructions peut aussi avoir été écrite et compilée dans le programme COBOL et on l'appelle alors séquence indépendante sur laquelle on se branche par une instruction PERFORM.

Dans notre exemple, on voit à la ligne 76 l'instruction PERFORM ERREUR qui fait exécuter le paragraphe ERREUR avant de revenir en séquence à la ligne 77. Mais on a aussi l'exemple d'une boucle itérative (ligne 64) qui commande d'exécuter le paragraphe BOUCLE (jusqu'à l'instruction EXIT) tant que la donnée EOF n'aura pas la valeur 1. Cette valeur 1 est mise dans l'instruction 77 de lecture lorsque l'on arrive à la fin du fichier FILECTU.

L'instruction STOP RUN (ligne 69) termine le traitement. Il n'y a plus d'autres instructions exécutées après cette instruction.

L'instruction EXIT ne sert qu'à préciser, sans ambiguïté, la fin d'un paragraphe (ligne 79).

2.6 Programmes contenus

L'évolution des techniques de programmation structurée a conduit les experts à introduire dans COBOL des programmes contenus, plus ou moins copiés sur les procédures internes de PASCAL ou PL/I.

Pour simplifier l'exposé, nous partirons de l'exemple présenté figure 4. Dans cet exemple, le programme PRINC contient les programmes SP1 et SP3. Le programme SP1 contient le programme SP2.

Comme pour les programmes externes (compilés séparément), les programmes contenus peuvent être appelés par un ordre CALL ou supprimés de la mémoire centrale par un ordre CANCEL.

Dans un programme contenu, les divisions ENVIRONMENT, DATA et même PROCEDURE sont facultatives, bien que l'on ne comprenne pas très bien ce que signifie un programme sans PROCEDURE.

Attribut GLOBAL

Un Nom-Donnée affecté de l'attribut GLOBAL est connu du programme dans lequel il est déclaré ainsi que de tous les programmes contenus dans ce programme, à condition que le Nom-Donnée ne soit pas déclaré une seconde fois.

Par **exemple**, ZONA décrit dans PRINC avec l'attribut GLOBAL est connu dans PRINC, SP2 et SP3. Par contre ZONA est LOCAL dans le programme SP1 où il est déclaré une seconde fois (figure 4).

Un Nom-Donnée déclaré sans attribut est par défaut LOCAL, donc connu d'un seul programme.

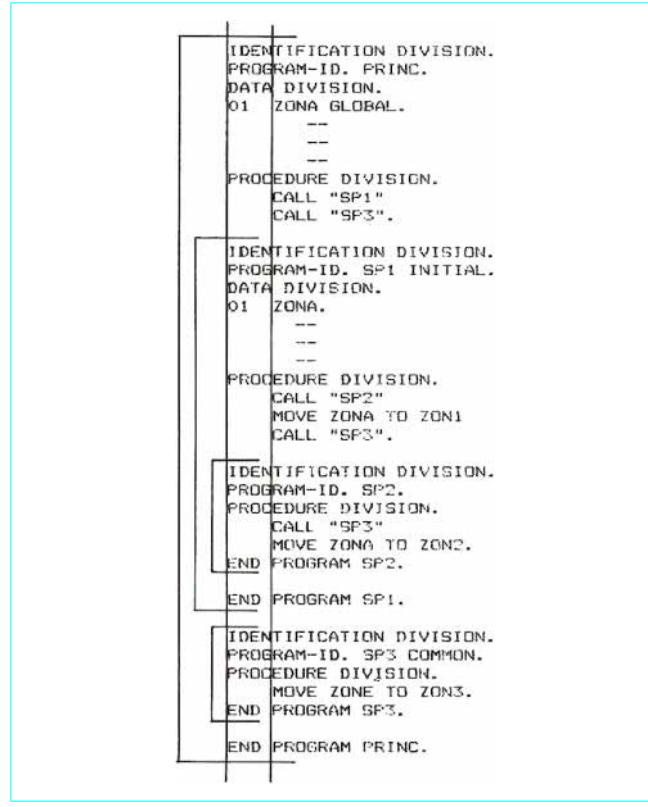

Figure 4 – Exemple de programmes contenus

Attribut COMMON

Normalement, un programme contenu ne peut être appelé que par un programme qui le contient au niveau immédiatement supérieur.

L'attribut COMMON affecté au nom d'un programme contenu permet, en revanche, d'appeler ce programme depuis n'importe quel autre programme de l'entité compilée.

Par **exemple**, le programme SP3 est connu de PRINC, SP1 et SP2 alors que SP1 n'est connu que de PRINC et que SP2 n'est connu que de SP1.

Attribut INITIAL

Jusqu'à présent, les données déclarées en COBOL ont toujours été de type statique, c'est-à-dire que l'état initial est affecté une seule fois à la compilation (par exemple la clause VALUE).

L'attribut INITIAL de la norme 85 affecté à un programme signifie que les données de ce programme seront de type automatique. Dans ce cas, les valeurs initiales sont affectées à chaque appel du programme, les fichiers sont remis à l'état initial, etc.

2.7 Programmation structurée

Outre les programmes contenus qui permettent un découpage fonctionnel des programmes, la norme 85 introduit les mots clés et terminateurs d'instructions nécessaires à la pratique de la programmation structurée.

Terminateurs

END-ADD, END-CALL, END-COMPUTE, END-DELETE, END-DIVIDE, END-EVALUATE, END-IF, END-MULTIPLY, END-READ, END-RECEIVE, END-RETURN, END-REWRITE, END-SEARCH, END-START, END-STRING, END-SUBTRACT, END-UNSTRING, END-WRITE

Ces terminateurs permettent d'écrire des groupes de séquences d'instructions à la suite sans faire appel à des PERFORM de paragraphe indépendants.

```
Exemple : READ FICHENT AT END CLOSE FICHENT
                  IF INDICA < 9999
                        THEN MOVE...
                        ELSE MOVE...
                  END-IF
          END-READ.
```

Instruction CONTINUE

Cette instruction neutre permet de passer « proprement » en séquence.

```
Exemple : IF A > B
              THEN MOVE C TO D
              ELSE CONTINUE
          END-IF.
```

PERFORM en séquence

Ce nouveau PERFORM permet d'exécuter en séquence un groupe d'instructions et non pas seulement un groupe d'instructions indépendant.

```
Exemple : PERFORM UNTIL EOF = '1'
                    MOVE ZOLEC TO ZONEDIT
                    READ FICHENT AT END MOVE '1' TO EOF.
          END-PERFORM.
```

PERFORM AFTER et BEFORE

Le PERFORM UNTIL traditionnel de COBOL fonctionne à l'envers des conventions de la programmation structurée. La norme 85 essaie de rattraper l'erreur en introduisant le PERFORM AFTER *test* en supplément du PERFORM BEFORE *test* jusqu'alors sous-entendu.

```
Exemple : PERFORM PAR1 THRU PAR2
                WITH TEST AFTER VARYING I FROM 1
                BY 1 UNTIL I > 20.
```

Cela permet d'exécuter au minimum une fois le PERFORM qui fait exécuter les paragraphes PAR1 et PAR2 en faisant progresser (VARYING) I de 1 en 1 jusqu'au maximum de 20.

Instruction EVALUATE

Cette instruction qui correspond au CASE de la programmation structurée permet de rassembler dans une seule instruction une sélection de traitements correspondant aux différentes valeurs que peut prendre une expression.

```
Exemple : Si A = 1 faire le traitement-1
          Si A = 2 faire le traitement-2
          Si A = 3 faire le traitement-3
          Sinon faire le traitement-n.
que l'on écrirait : EVALUATE A
                    WHEN '1' PERFORM TR1
                    WHEN '2' PERFORM TR2
                    WHEN '3' PERFORM TR3
                    WHENOTHER PERFORM TRN.
```

2.8 Communications entre programmes non contenus

Données EXTERNAL

Lorsque l'on souhaite partager des données ou fichiers entre programmes compilés séparément, on peut leur donner l'attribut EXTERNAL. Il est évidemment nécessaire que les données aient toujours le même nom et la même description dans chaque programme.

Passage d'arguments par valeur

Jusqu'à présent les arguments étaient passés d'un programme à l'autre dans l'ordre CALL uniquement par la référence des données (BY REFERENCE). Il est maintenant possible de passer directement les valeurs des arguments (BY CONTENT).

3. Fichiers COBOL

Le type de fichier le plus courant est le fichier séquentiel où les articles ou enregistrements sont classés dans l'ordre où ils ont été écrits. En général l'ordre est déterminé par un indicatif (REF ligne 34 pour l'article ENTREE du fichier FILECTU décrit sur les lignes 30 à 35).

Une fois un fichier créé, ses articles ne peuvent être relus que dans cet ordre, l'un après l'autre et éventuellement réécrits après modification. Si l'on désire traiter le fichier dans un ordre différent, il faut au préalable trier le fichier sur un autre indicatif (un code service par exemple).

Si l'on veut accéder à un article particulier, il faut lire tous les articles qui le précèdent, ce qui peut être très long pour un fichier de plusieurs millions d'articles. Par conséquent, il est impensable d'utiliser un fichier séquentiel pour une application transactionnelle avec accès aléatoire aux enregistrements.

3.1 Fichiers et accès

COBOL propose deux organisations de fichiers qui permettent l'accès direct aux articles : l'organisation **relative** et l'organisation **indexée**.

Dans ces deux organisations, on accède aux articles en désignant un indicatif de recherche encore appelé **clé d'accès** (KEY). Le mode d'accès aux articles peut être :

— SEQUENTIAL (séquentiel) : on accède alors aux articles dans l'ordre croissant de l'indicatif ;

— RANDOM (sélectif) : la séquence selon laquelle on accède aux articles est aléatoire, contrôlée par le programmeur qui place la valeur de l'indicatif recherché dans la zone KEY, clé d'accès du fichier ;

— DYNAMIC (sélectif/séquentiel) : c'est un mixage des deux précédents : on peut accéder sélectivement à un article déterminé puis traiter le fichier en séquence à partir du dernier article *accédé*. Par exemple, si l'on ne veut traiter que la ville d'Orléans, on se positionne sur l'indicatif postal 45000 puis on traite le fichier en séquence. On dispose à cet effet d'une instruction particulière START qui positionne sur l'article recherché et de READ NEXT qui demande l'article suivant en ordre séquentiel.

L'organisation du fichier, le mode d'accès et la clé doivent être désignés dans l'ENVIRONMENT DIVISION. Les lignes 21 à 24 de notre exemple montrent :

— ligne 21 : le fichier FISECIN est assigné à une unité DISK ;
— ligne 22 : l'organisation est INDEXED ;
— ligne 23 : le mode d'accès est SEQUENTIAL ;
— ligne 24 : la clé d'accès est la zone CODREF ;
— ligne 25 : le FILE STATUS s'appelle FISTAT.

Ce FILE STATUS (état du fichier) est une zone numérique de deux caractères, qu'il faut déclarer en DATA DIVISION (ligne 51), dans laquelle le système de gestion des fichiers vient mettre une valeur 00 lorsque l'opération de traitement de fichier (ouverture, lecture, etc.) s'est bien déroulée et une valeur différente dans le cas contraire ; cela permet de contrôler le bon déroulement des opérations avec les fichiers.

3.2 Fichiers relatifs

L'organisation relative consiste à désigner un enregistrement logique de fichier sur mémoire à accès sélectif par le numéro relatif de l'enregistrement, compté à partir de un pour le premier enregistrement du fichier. Un enregistrement logique est donc défini exclusivement par son numéro d'ordre. L'organisation relative est la plus performante des organisations en accès sélectif mais est difficile à réaliser dans la pratique car il est assez rare de pouvoir désigner un enregistrement par son numéro d'ordre. On peut toutefois imaginer une société vendant un maximum de 99999 pièces répertoriées par un matricule allant de 00001 à 99999. Le fichier relatif comporte alors 99999 enregistrements, avec des enregistrements fictifs pour les matricules qui ne correspondent pas à des pièces réelles.

L'argument de positionnement ou recherche d'un enregistrement est fourni à COBOL dans une zone de données RELATIVE KEY décrite en WORKING-STORAGE SECTION sous forme d'un nombre entier non signé.

Exemple : si l'on remplaçait le fichier indexé FISECIN de notre exemple par un fichier relatif FIRELA, les lignes 21 à 25 du programme s'écriraient alors comme suit :

```
SELECT FIRELA ASSIGN TO DISK-FIRELA
              ORGANIZATION RELATIVE
              ACCESS SEQUENTIAL
              RELATIVE KEY CLEF
              FILE STATUS FISTAT.
```

3.3 Fichiers indexés

L'organisation séquentielle indexée, plus complexe en principe, est cependant la plus utilisée car finalement la plus facile à adapter dans la pratique. Cette organisation consiste à permettre la recherche d'un enregistrement logique par consultation de tables successives. On peut comparer ce processus de recherche à la consultation du Bottin.

Un index général nous indique le volume à consulter puis une table des matières nous désigne le chapitre, puis éventuellement la page. Enfin, une recherche séquentielle dans la page nous fournit le renseignement souhaité. Dans la littérature informatique, on lit le plus souvent qu'il s'agit d'une structure en arbre avec des troncs et des branches.

Les accès sont réalisés à l'aide d'une clé d'enregistrement (RECORD KEY) présente physiquement dans l'enregistrement logique et de tables de clés qui concrétisent la structure en arbre.

Il résulte que chaque fichier logique séquentiel indexé est le plus souvent scindé en deux fichiers physiques.

- **Fichier données** : il comporte les informations contenues dans les enregistrements logiques. Il faut noter que, dans nombre de systèmes, les enregistrements supprimés sont distingués par la valeur HIGH-VALUE (constante figurative) dans le premier caractère de l'enregistrement. On peut donc avoir intérêt à réserver ce premier caractère si l'on souhaite échanger des fichiers séquentiels indexés avec d'autres systèmes.

- **Fichier clés** : il contient les clés et les adresses physiques des enregistrements du fichier données.

En fait, cette organisation physique complexe se traduit par une organisation logique simple pour le programmeur qui n'a pratiquement qu'à désigner le Nom-Donnée qui sert de clé dans l'enregistrement logique du fichier.

Les enregistrements du fichier doivent être classés, à la création, en ordre de séquence croissante des clés. Ensuite, le fichier peut être traité en séquence ou bien en accès aléatoire en désignant une valeur précise de la clé d'accès. Si aucun enregistrement ne possède la valeur de clé recherchée, cela génère une condition de clé invalide (INVALID KEY ligne 75) qu'il faut traiter dans un paragraphe particulier (ERREUR dans notre exemple).

Clés secondaires : à l'origine, il était spécifié que la clé devait être unique et qu'il ne devait pas y avoir deux enregistrements ayant la même valeur de clé. Mais les systèmes de gestion de fichiers modernes acceptent maintenant des clés secondaires uniques ou multiples. Par exemple, un fichier de personnel ordonné suivant les matricules croissants comme clé principale peut aussi être classé suivant l'ordre alphabétique des noms de personnes comme clé secondaire.

On déclare la clé secondaire dans l'ENVIRONMENT DIVISION sous la forme ALTERNATE RECORD KEY IS NOM et on ajoute WITH DUPLICATES s'il existe plusieurs personnes du même nom, donc si la clé secondaire est multiple. Si l'on utilise plusieurs clés dans un programme pour un même fichier, il faut l'indiquer dans l'ordre READ de lecture pour que le compilateur s'y retrouve.

4. Gestion des tables

Il était déjà facile d'utiliser des tables avant l'apparition de la norme 74 mais celle-ci a introduit de nouvelles fonctions, fondamentales pour leur gestion. Rappelons qu'une table est un groupe ordonné de données composé d'éléments simples ayant tous des attributs identiques. Chacun des éléments de la table est désigné à l'aide d'un numéro d'ordre appelé indice sous la forme ELEMENT (*indice*). L'indice peut être utilisé sous forme d'un nombre entier, par exemple ELEMENT (5) pour désigner le cinquième élément d'une table. L'indice peut aussi être utilisé sous forme d'un Nom-Donnée, par exemple SALAIRE (ECHELLE) avec ECHELLE prenant exclusivement des valeurs entières. Puisque COBOL autorise jusqu'à trois indices pour une table nous pouvons écrire dans le cas le plus général ELEMENT (*indice-1, indice-2, indice-3*). Depuis 1974, COBOL offre la possibilité de remplacer les indices par des **index** jouant pratiquement le même rôle mais implicitement réservés par COBOL lors de la déclaration de la table.

Par **exemple** :

```
01   TABLE.
     02   TABLEAU OCCURS 2 INDEXED BY IND1.
          04   VECTEUR OCCURS 2 INDEXED BY IND2.
               06   ELEM OCCURS 3 INDEXED BY IND3 PIC 9(8).
```

suffit à déclarer une table et les trois index IND1, IND2, IND3.

À première vue, index et indices paraissent jouer exactement le même rôle. En fait le contenu de l'indice est un nombre entier désignant le numéro d'ordre de l'élément compté à partir de 1 pour le premier ; par contre, le contenu de l'index est le nombre de positions de mémoires (octets par exemple) désignant le déplacement de l'élément par rapport au début de la table. On peut résumer en disant que l'indice désigne un nombre d'éléments alors que l'index est un *offset* (décalage) par rapport à une adresse interne de mémoire. On donne une valeur à un index par l'instruction SET IND1 TO 1 qui ici positionne l'index sur la première occurrence des éléments de la table.

Pour rechercher dans une table un élément qui satisfait à une ou plusieurs conditions données, on utilise l'instruction SEARCH.

Exemple : recherche d'un numéro de compte analytique à partir d'une table de codes divisions et codes services.

Description de la table en WORKING-STORAGE SECTION :

```
01   TABLANA.
     02   ANALI OCCURS 1000 ASCENDING KEY DIVIS SERVI
          INDEXED BY INDI.
          04   DIVIS PIC 99.
          04   SERVI PIC 9(4).
          04   COMPT PIC 9(4).
          04   LIBEL PIC X(20).
```

La clause OCCURS indique que la table TABLANA est composée de 1 000 groupes de données ANALI (OCCURS 1000) classés suivant la séquence ascendante (ASCENDING KEY) de la division (DIVIS) et du service (SERVI) dans la division. L'index de la table s'appelle INDI. Chaque groupe ANALI comporte les éléments de données DIVIS, SERVI, COMPT et LIBEL.

Exemple : la recherche dans la table en PROCEDURE DIVISION s'écrit :

```
SEARCH ALL ANALI AT END GO TO ERREUR1
   WHEN DIVIS = DIV AND SERVI = SERV
      MOVE COMPT (INDI) TO NUCOMPTE.
```

On recherche (SEARCH) parmi tous (ALL) les groupes de données ANALI ceux qui correspondent à la condition : DIVIS doit être égal au contenu de la zone DIV et SERVI égal au contenu d'une zone SERV. Lorsque l'égalité est trouvée, on transfère le numéro de compte de la table COMPT (INDI) dans une zone NUCOMPTE. Si, en fin d'exploration de la table, l'instruction SEARCH n'a pas trouvé d'égalité, alors il faut se dérouter pour exploiter un paragraphe ERREUR1.

5. Éditeur (REPORT WRITER)

L'éditeur COBOL est un module généralisé qui permet de générer automatiquement des états en déchargeant le programmeur des problèmes de ruptures, comptages de lignes, éditions d'en-têtes, cumuls de nombres intermédiaires et généraux, etc.

En édition, l'unité homogène est la ligne d'imprimante mais l'idée de base de l'éditeur COBOL est de rassembler les lignes en groupes d'éditions.

Un ensemble ordonné de lignes éditées successivement et dépendant d'une condition unique constitue un groupe d'édition.

Pour un état, le programme EDITEUR doit comporter :

— une déclaration de fichier dans l'ENVIRONMENT et la DATA DIVISION ;

— une description d'état dans une section spéciale de la DATA DIVISION, la REPORT SECTION avec les groupes en-tête, détail, bas de page, fin d'état, etc. ;

— les verbes de génération d'état en PROCEDURE DIVISION :

- INITIATE au début de l'édition,
- GENERATE pour générer les groupes détail,
- TERMINATE en fin d'édition.

L'éditeur offre aussi la possibilité de produire des lignes récapitulatives en calculant automatiquement des sous-totaux à chaque fois qu'un CONTROL est détecté, c'est-à-dire en français une rupture dans l'édition. Le plus haut niveau de CONTROL est le niveau FINAL de fin d'état.

Pour chaque ligne détail générée, COBOL évalue les différents tests CONTROL et, lorsqu'une rupture est décelée pour l'un d'entre eux, il y a naturellement génération de la cascade des niveaux de rupture qui lui sont inférieurs.

À chaque niveau de rupture ou conditionnement correspond un groupe d'édition CONTROL, la correspondance étant donnée lors de la description des groupes.

Par **exemple**, pour CONTROL IS PAYS DEPT CANTON VILLE, une rupture détectée sur DEPT déclenchera les contrôles inférieurs CANTON et VILLE, d'où la génération des groupes d'édition de totalisation associés à VILLE puis CANTON puis DEPT.

L'éditeur COBOL est réellement un générateur d'édition très puissant mais dont la logique déroute bien souvent le programmeur, ce qui lui vaut des dénigrements totalement injustifiés.

6. Tri (SORT)

La plupart du temps, lorsque l'on veut bâtir un état, il faut au préalable trier le fichier de base ordonné par exemple sur les matricules de l'individu, alors que l'état est demandé par code de service.

Le programmeur doit alors intercaler des tris standards entre chaque programme ou recourir au tri COBOL, qui permet d'inclure les programmes généralisés de tri dans un programme COBOL.

En fait, disons plus justement qu'on permet l'introduction de séquences de programmes COBOL, en début ou fin de tri, dans le but de modifier ou d'éliminer des enregistrements avant ou après tri.

Le programme avec tri est bâti différemment des programmes habituels (dans lesquels généralement on lit, on calcule, on écrit) car ici on ne lit ni n'écrit, simplement on trie (SORT) et éventuellement on greffe une procédure de traitement en début de tri (INPUT PROCEDURE) ou fin de tri (OUTPUT PROCEDURE).

Autre originalité des programmes de tri, l'apparition d'une description de fichier de tri SD (Sort Description) en FILE SECTION au lieu des FD (File Descriptions) habituelles, car il ne s'agira pas de décrire réellement un fichier mais les enregistrements à trier. L'ordre SORT, placé dans la PROCEDURE DIVISION, désigne les enregistrements à trier ainsi que les arguments de tri.

Par **exemple** : SORT FICHTRI ON ASCENDING MAJEUR DESCENDING INTER ASCENDING MINEUR.

DESCENDING signifie que le classement doit être décroissant alors que ASCENDING indique un classement croissant.

Il est également possible de fusionner des fichiers, déjà triés sur des arguments identiques, en utilisant l'instruction MERGE.

7. Mise au point des programmes COBOL

Les compilateurs COBOL donnent généralement de nombreuses indications sur les erreurs détectées dans les programmes qui leur sont soumis. Mais le compilateur ne détecte pas les erreurs au-delà du contrôle des règles d'écriture d'un programme COBOL.

Lorsqu'il a corrigé les erreurs de syntaxe avec l'aide du compilateur, le programmeur doit ensuite corriger les erreurs de logique et de calcul à l'aide de jeux de données d'essais fournis au programme.

8. Perspectives

8.1 COBOL orienté objet

Les méthodes de programmation orientées objet ouvrent des horizons totalement nouveaux dans le domaine du développement d'applications informatiques de gestion.

Il est donc normal que le langage majeur de programmation de gestion, le langage COBOL, s'engage dans la programmation orientée objet, mais non sans quelques problèmes à résoudre.

Le groupe ANSI-X3J4, qui a la charge de l'élaboration de la future norme COBOL, aujourd'hui planifiée pour 1997, a décidé de créer en 1992 un sous-groupe X3J4.1 OO-COBOL (Object Oriented COBOL), dédié à la programmation objet.

Les experts de X3J4.1 s'activent considérablement, car ils souhaitent que l'OO-COBOL soit partie intégrante de la norme COBOL 1997.

À cet effet, leurs travaux ont été présentés aux derniers séminaires du groupe ISO-SC22-WG4 COBOL.

Les principes et spécifications de base de la programmation objet ont été clairement explicités. En revanche, leur application au langage COBOL n'est pas évidente sur plusieurs points, par exemple :

— la définition des classes objets méthodes, envisagée sous la forme de programmes COBOL imbriqués, a été remise en cause car jugée trop compliquée à visualiser et à appréhender ;

— l'appel de méthodes dans le cours du programme COBOL doit-il être réalisé sous la forme d'appel de sous-programme, de fonction intégrée ou d'opérateur ? ;

— les notions de fichiers et de persistance des données n'existant pas en programmation orientée objet, faut-il refondre complètement COBOL ?

Sur une vingtaine de problèmes importants recensés, seule la moitié a pu être étudiée par les experts de WG4. Or, plus que jamais, il faut s'attendre à ce que, dans le futur, les produits programmes mis en vente sur le marché ne soient pas intégrables aux applications OO-COBOL s'ils ne sont pas conformes aux normes internationales, d'où l'importance des études en cours.

Après de nombreuses discussions, au sein de l'ANSI et de l'ISO, une proposition formelle d'OO-COBOL a été soumise aux experts. Il apparaît que cette proposition est très complexe et l'on peut légitimement douter de sa praticabilité pour les programmeurs lambda. Les experts de l'ISO ayant voté majoritairement (pas la France) pour que l'OO-COBOL soit inclus obligatoirement dans la future norme plutôt que sous forme d'addendum, il est à craindre que cela repousse la parution de la norme prévue pour 1997 d'au moins une année.

8.2 Faut-il choisir COBOL ?

Avant de décider, il faut évaluer les différents critères de choix.

Capacités de traitement

Le langage est bien adapté aux applications de gestion et son évolution, certes lente, suit l'évolution des nouveautés technologiques.

Adaptation aux matériels

On peut dire qu'il n'existe plus aujourd'hui de réelle limite inférieure pour installer un compilateur COBOL sur les micro-ordinateurs qui offrent des capacités comparables aux machines les plus puissantes.

Performances des compilateurs et des programmes

Le temps est révolu où un programme Assembleur était dix fois plus performant qu'un programme COBOL. Aujourd'hui les performances sont sensiblement équivalentes avec, de plus, une maintenance très simplifiée des programmes.

Coût des compilateurs

Il existe maintenant une longue expérience de fabrication des compilateurs COBOL, ce qui garantit un coût très raisonnable à l'achat.

Apprentissage de COBOL

La majorité des programmeurs sur le marché de l'emploi pratique le COBOL. La formation d'un programmeur COBOL n'est pas onéreuse car, en un mois, il est possible de former un programmeur débutant pouvant écrire des programmes simples.

Portabilité de COBOL

Grâce aux organismes de normalisation, les programmes COBOL sont de plus en plus compatibles d'un ordinateur à l'autre.

Pérennité de COBOL

L'énorme quantité de sites informatiques utilisant COBOL et le coût qu'implique un changement de langage pour convertir les bibliothèques de programmes assurent la pérennité du langage.

Programmation orientée objet

COBOL sera très certainement, d'ici peu, le seul langage évolué permettant une réelle programmation évolutive orientée objet.

COBOL

par **Christian BONNIN**
Ingénieur informaticien de l'Institut Industriel du Nord (IDN)
Licencié ès Sciences
Expert Langages informatiques
Représentant de la France (AFNOR) à l'ISO

Bibliographie

BONNIN (C.). – *COBOL et CICS.* Eyrolles (1993).
BONNIN (C.). – *COBOL MicroFocus.* Eyrolles (1993).
BONNIN (C.). – *COBOL ANS 85.* Eyrolles (1992).
SILVERIO (N.). – *Programmer en COBOL ANS 85.* Eyrolles (1993).

Normes

ANSI X 3.23	1985	Programming language COBOL
ISO 1989 :	1985	Langages de programmation - COBOL
NF EN 21989	1-94	Langages de programmation - COBOL