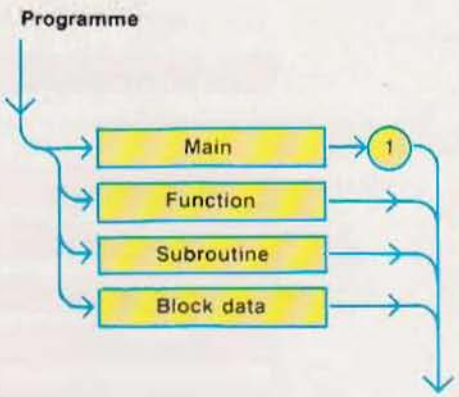
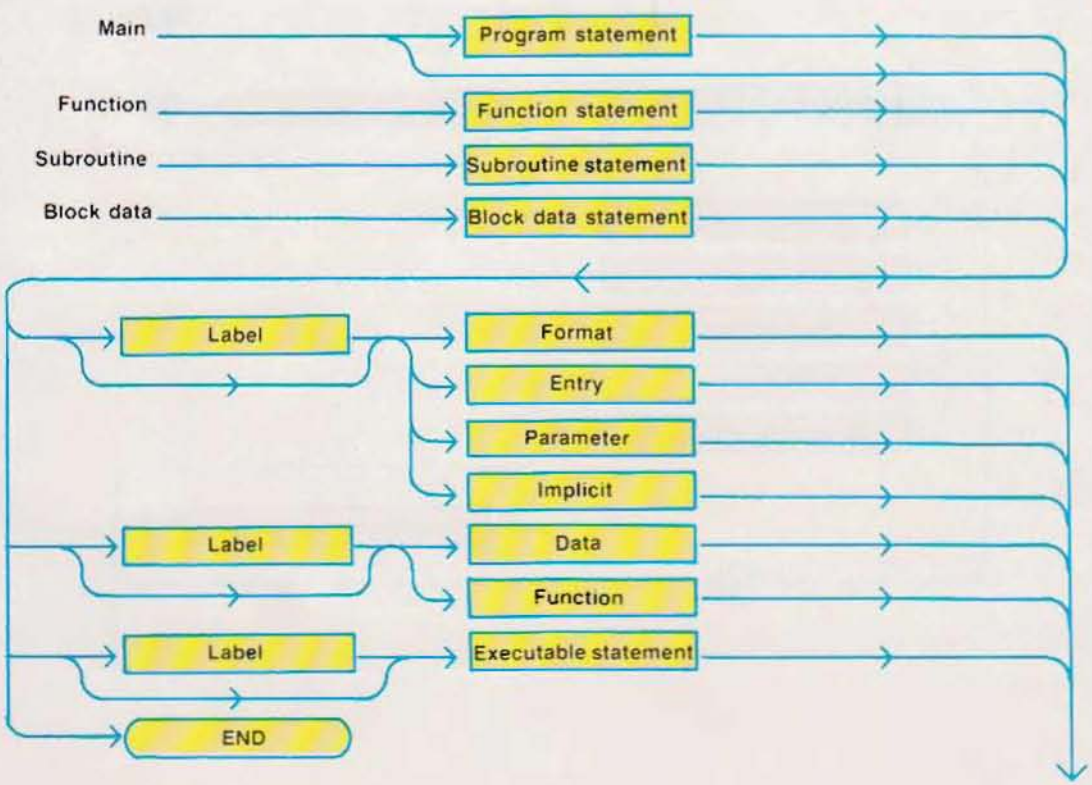
Ainsi, pour chacune des lignes précédentes, il existe de nouvelles possibilités, à un niveau de détail plus poussé.
Le graphique ci-dessous illustre cette hiérarchie sur 2 niveaux.
En poussant encore vers le détail, on aboutit à la description individuelle de chaque instruction (voir exemple, pages suivantes).

REPRESENTATION "RAILROAD NORMAL FORM"

1er niveau



2e niveau



REPRESENTATION DE LA SPECIFICATION DE TYPE

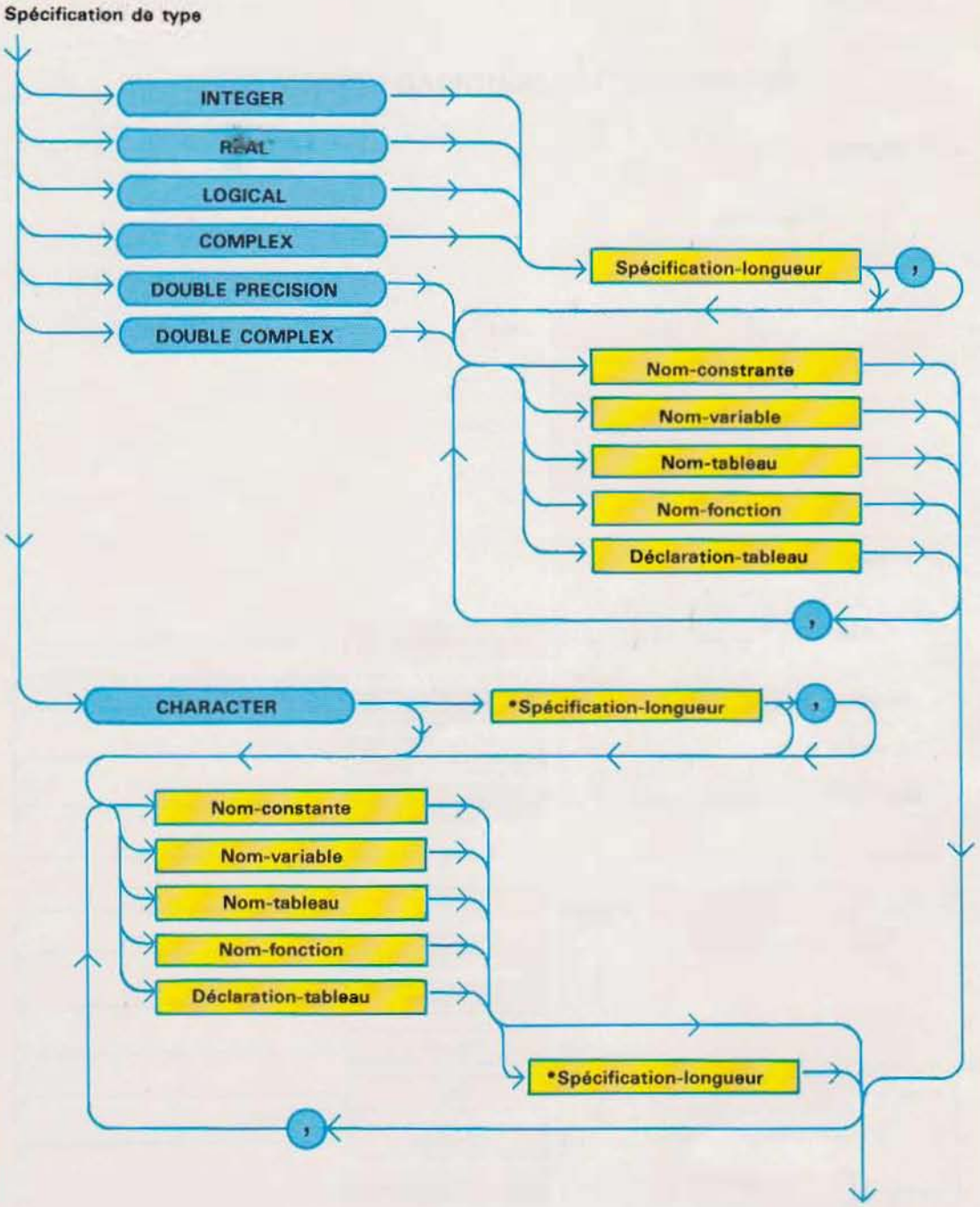
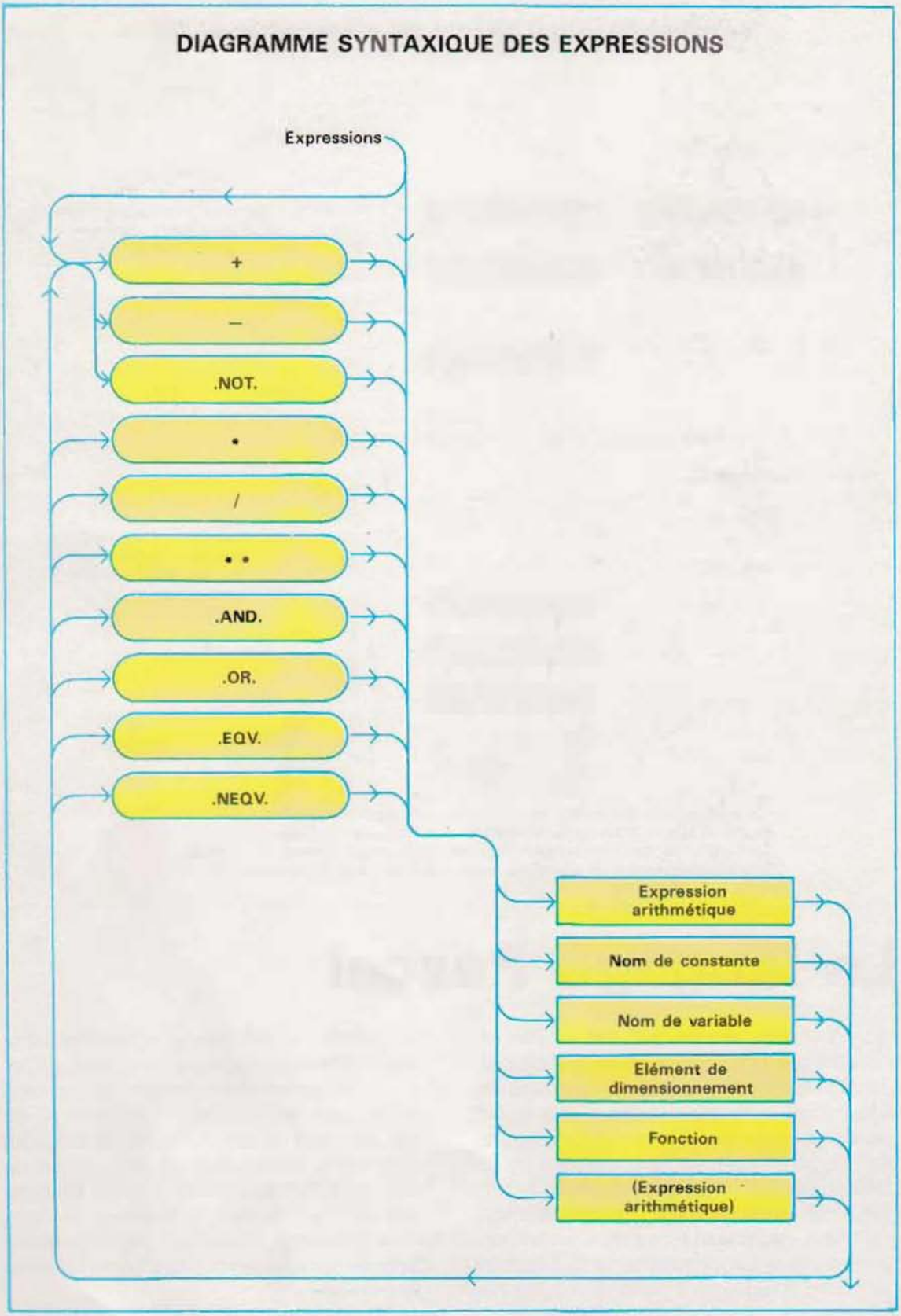
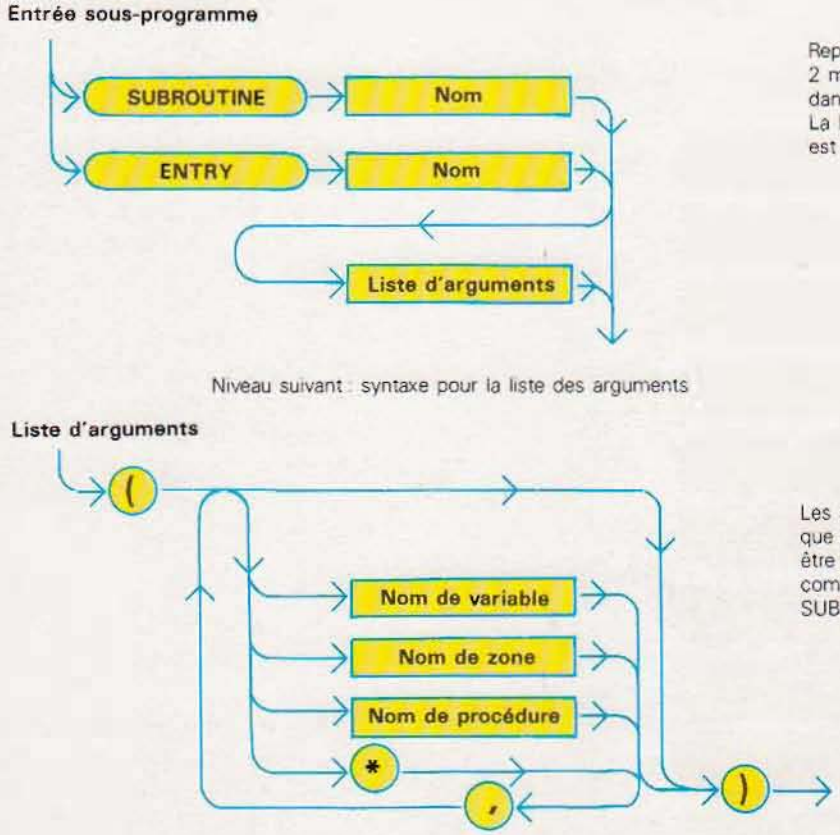


DIAGRAMME SYNTAXIQUE DES EXPRESSIONS



REPRESENTATION D'ENTREE EN SOUS-PROGRAMME ET STRUCTURE DES PARAMETRES



Le langage Pascal

Au début des années 60, les ordinateurs n'avaient pas la puissance qu'on leur connaît. Et quand le programmeur devait construire des algorithmes en langage machine, ces limites tenaient évidemment à la technologie employée : capacité maximale de la mémoire ne dépassant pas les 16-32 Koctets. Aussi le programmeur était-il amené à améliorer le rendement de la machine en recourant à des astuces particulières qui, malheureusement, finissaient par rendre pratiquement illisible le programme.

Les progrès réalisés depuis, aussi bien sur la machine elle-même que sur les logiciels de base, ont inconstestablement donné naissance à une nouvelle génération d'ordinateurs. C'est ainsi que sont apparus d'abord les langages symboliques assemblés, puis les langages de haut niveau (de plus en plus orientés langages naturels). Ces derniers comportent de nombreux sous-programmes (routines d'utilisation générale absentes des machines des premières générations.

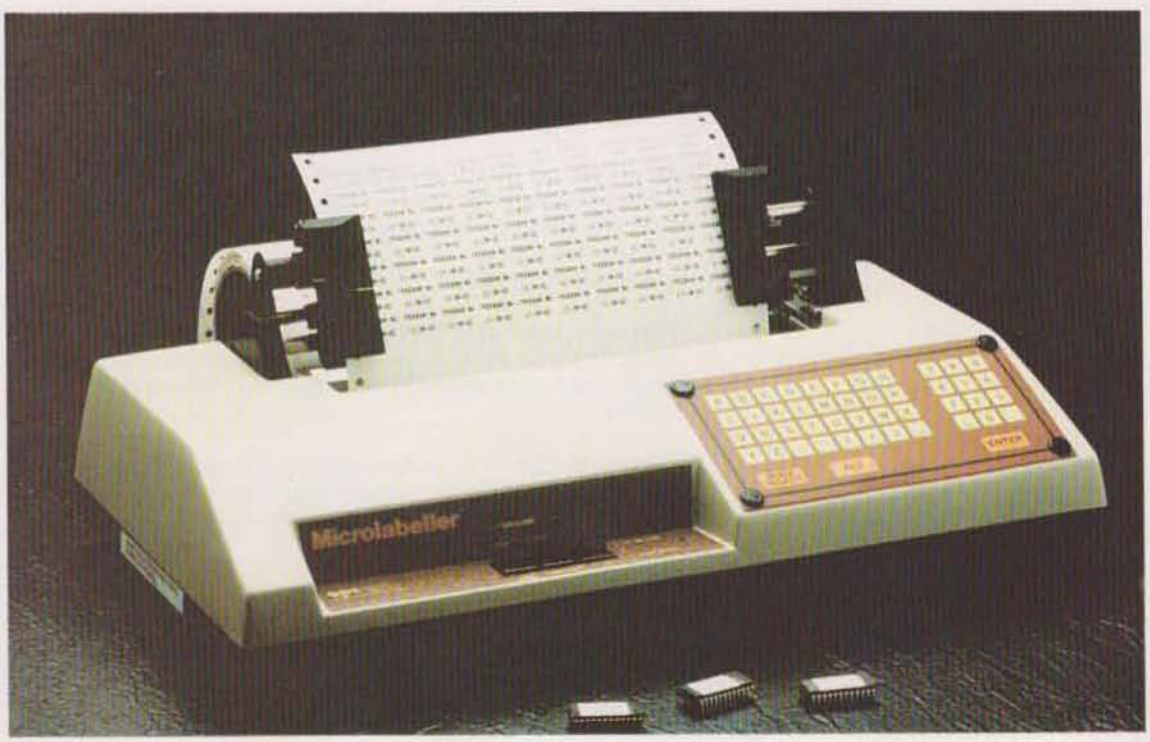
Une telle évolution a rendu inopérante l'emploi, par les programmeurs, des astuces qui faisaient précisément leurs compétences. Comme on peut l'imaginer, il n'a pas été facile pour les créateurs du Fortran (introduit vers 1957) de combattre la résistance et les préjugés des programmeurs dont l'efficacité se mesurait à la « technologie des trucs » (trickology, en anglais), devenue un art. Après le Fortran, en 1960, vint l'Algol (Algorithmic language), fondamentalement proche du Fortran mais disposant d'un atout supplémentaire. Au niveau des instructions, on avait la possibilité de créer des « structures » de procédures en regroupant les instructions de manière à former une opération complète, avantage dont le Fortran disposait seulement au niveau des expressions arithmétiques. On pouvait, dès lors, composer des instructions en structures de contrôle ; les opérations définies par un programme étaient donc vues comme une répétition de blocs et non pas comme de simples instructions.

Malheureusement, l'Algol, malgré ses innovations, n'a pas eu le succès escompté. Il reste aujourd'hui un langage de chercheurs et d'universitaires. Son absence, dans des applications scientifiques ou de gestion, est certainement due à la complexité du concept de structure, incompatible avec l'art d'utiliser les trucs et les sauts inconditionnels à l'intérieur des instructions.

Cette inertie est analogue à celle qui a créé un obstacle dans le passage de l'Assembleur vers le Fortran. Aujourd'hui, il ne s'agit plus tant de pousser la technologie dite « hard » pour gagner quelques microsecondes dans la puissance de traitement, encore que, dans le domaine de l'intégration à très haute échelle (VLSI), on soit encore loin des limites. Pour beaucoup, l'effort doit se porter sur la fiabilité des programmes et sur les possibilités de les modifier sans encombre.

C'est précisément la philosophie de la **programmation structurée.** Elle consiste à fournir au programmeur une méthode pour résoudre les problèmes de manière systématique sans recourir à son « atelier de trucs ». Parmi les langages les plus courants (voir page 1204), seul le Pascal (et demain peut-être l'Ada) revient à la structuration proposée par l'Algol. Pour que la structuration ne soit plus liée à un langage particulier, on tente de plus en plus d'y introduire des **structures de contrôle.** C'est une tendance qui devrait se confirmer.

Imprimante programmable pour l'édition d'étiquettes autocollantes.



B. Coleman/Matra

LANGAGES DE PROGRAMMATION

Date création	Nom du langage	Description
1957	Fortran **For**mula **Tran**slator	Très employé. Utilise des notations très proches de l'algèbre. L'unique structure de données est le tableau. Possibilité de définir des sous-programmes comme bibliothèques. L'efficacité des premiers compilateurs a rendu acceptable le Fortran dans toutes les applications scientifiques, mathématiques et statistiques.
1960	Algol **Algo**rithmic **l**anguage	Peu utilisé en dehors des milieux de recherche sur le software. Plus connu en Europe qu'aux Etats-Unis, il introduit les concepts de base de la programmation structurée.
1960	Cobol **Co**mmon **b**usiness **o**riented **l**angage	Très courant. Par définition, le langage de gestion utilise une notation "English like" (c'est-à-dire très proche du langage naturel anglais).
1961	Lisp **List p**rocessing. Langage de programmation en intelligence artificielle.	Très utilisé dans les recherches sur l'intelligence artificielle. La structure de base et la "liste" dans laquelle les éléments sont des données élémentaires ou des listes.
1962	Snobol	Langage de programmation pour la manipulation de chaînes de caractères. Il inclut de puissantes instructions pour la recherche d'ensembles de caractères ou chaînes. Il est, par exemple, utilisé dans la gestion de bibliothèques.
1965	Basic **B**eginner's **a**ll-purpose **s**ymbolic **i**nstruction **c**ode	Conçu comme un simple langage d'apprentissage, il est aujourd'hui le plus utilisé en micro-informatique. Malheureusement, il n'existe par un standard mais d'innombrables variétés.
1965	PL/1 **P**rogramming **l**anguage	Développé par IBM, il devait être le langage qui rassemblerait les caractéristiques du Fortran, du Cobol et de l'Algol. Il en est résulté un langage extrêmement complexe et excessivement étendu. On le trouve uniquement sur les machines IBM (peu utilisé).
1967	APL **A p**rogramming **l**anguage	Regroupe une vaste panoplie d'opérations mathématiques. L'usager peut, en plus, définir ses propres opérateurs. Les programmes APL sont très concis mais impénétrables.
1971	Pascal	Considéré comme un instrument pour l'enseignement de la programmation structurée. Il intègre les structures de contrôle de cette programmation, très efficaces dans la rédaction des programmes.
1980	Ada	Langage temps réel, utilisable pour le contrôle de processus. Il intègre de nouvelles idées sur les structure modulaires et les compilations séparées pour assurer le développement de grands projets.

La programmation structurée

La programmation structurée est fondée sur un concept essentiel : quelle que soit la tâche à programmer, il est toujours possible de la décrire dans un ensemble d'unités fondamentales, en recourant au langage naturel.
La seconde étape consiste à décomposer chaque unité en plusieurs sous-unités et ainsi de suite, jusqu'à arriver au niveau d'abstraction le plus élémentaire.
Ce travail d'affinement débouche sur un ensemble d'instructions interprétables par un calculateur, qu'elles appartiennent à un langage de haut niveau comme le Fortran ou qu'il s'agisse d'instructions en langage machine.
Ainsi, pour lancer la tâche de « préparer un repas », on peut isoler la suite des opérations fondamentales : acheter les ingrédients nécessaires/préparer les plats/servir. Chacun de ces blocs est ensuite décomposé en une succession de petites tâches. La préparation des plats pourra être ultérieurement spécifiée comme un travail sur un ensemble de services (entrée, plat, garniture, dessert, fruits). La décomposition d'un bloc au niveau du détail nous amènera au stade des opérations élémentaires à accomplir, comme mettre une casserole d'eau sur le feu et attendre l'ébullition.
Il est important d'utiliser, pour chaque niveau d'abstraction, des **blocs connexes de manière simple,** et conformément au schéma général adopté. Principe important : un bloc n'a **qu'un point de départ** (entrée) et **un point final** (sortie). De cette manière, on gagne nettement en clarté sur les inter-relations des divers blocs.
La programmation structurée peut donc être entendue comme un ensemble de techniques de planification et de codification qui guident le programmeur dans la préparation de programmes organisés, autodocumentés et vraiment fiables.

L'organisation des programmes

La technique consistant à partir du problème général et à le subdiviser en sous-problèmes plus petits, à travers des pas successifs, est appelée **top-down** (du haut vers le bas). La seconde technique, **bottom-up** (du bas vers le haut), a une approche inverse : commencer à partir du niveau de détail le plus élémentaire en constituant de petites unités, qui, à leur tour, formeront des blocs plus complexes. Dans le cas de la préparation du repas, résoudre le problème avec la technique bottom-up consisterait à préparer un plat particulier cherchant les ingrédients qui sont donc nécessaires et à composer les différents plats jusqu'au repas complet.
Le développement de programmes comporte deux phases en méthode structurée : **planification et implémentation .** Pendant la planification, la technique de développement la plus usitée est le top-down alors que dans l'implémentation c'est le bottom-up.
En fait, lors de l'analyse d'un problème, le système top-down est sans aucun doute irremplaçable. Par contre, dans le cas de la synthèse (nécessaire pour transférer le programme sur une machine ou dans un langage particulier) la meilleure approche est le bottom-up. En général, pour développer un programme de manière ordonnée, quatre niveaux d'abstraction au moins seront nécessaires avant d'arriver au détail dans le langage de programmation choisi :

1 / objectif à atteindre
2 / remarques détaillées
3 / pseudocode
4 / code

Lors de l'écriture d'un programme, un test simple permettra de vérifier la validité de l'organisation : décrire par des phrases le résultat visé. Si ce pas s'avère trop difficile, c'est probablement parce que la finalité du programme n'apparaît pas encore assez clairement ; on doit alors procéder à un nouveau contrôle des descriptions détaillées.
Il convient ensuite d'examiner directement si ce qui a été décrit est réalisable avec les ressources disponibles, qu'elles soient matérielles ou logicielles, par exemple, en vérifiant que l'ordinateur disponible est bien adapté en termes de rapidité et de capacité mémoire. Une fois l'objectif défini de manière satisfaisante, on procède à la mise au point des notes détaillées. Celle-ci peuvent être subdivisées en au moins trois parties :

1 / données d'entrée
2 / calculs à effectuer
3 / données de sortie

La description complète des données d'entrée et de sortie facilite la phase intermédiaire, c'est-à-dire la description des actions à accomplir à l'intérieur du programme. Cette description (calculs à effectuer) faite en langage naturel aide à faire ressortir d'éventuelles lacunes. Dans la rédaction du pseudocode, les blocs d'entrée, de calcul et de sortie précédemment définis peuvent être segmentés en d'autres blocs dans lesquels sont utilisées des structures contenant des actions précises. Le pseudocode ne définit pas le langage final d'écriture du programme ; cependant, l'emploi de structures du type Algol ou Pascal aide à rendre plus clair et lisible ce que l'on désire réaliser.
Un test important peut être effectué au terme de la préparation du pseudocode : faire lire le programme à une personne extérieure au projet. On est sûrement sur la bonne voie pour obtenir un programme clair et facile à manipuler si l'objectif visé est correctement interprété.
La dernière étape consiste alors à traduire le pseudocode en code, en utilisant le langage le plus adapté à cette application. Certains langages, comme le Pascal, rendent cette opération de traduction très facile puisqu'ils définissent des procédures, fonctions et variables booléennes et permettent, en outre, d'ajouter des commentaires afin de rendre le code semblable au pseudocode descriptif. A ce niveau, il est conseillé de créer de véritables bibliothèques de procédures ou des fonctions qui, le cas échéant, seraient rappelées en divers points du programme.
Si le processus s'est correctement déroulé, le programme principal (main) sera exclusivement formé de quelques appels de procédure ou de fonctions qui, à leur tour, contiendront d'autres procédures, et ainsi de suite... jusqu'à ce que l'on arrive aux blocs finaux (qui ne contiendront que peu de lignes de code).
Le programme ainsi obtenu n'aura pas besoin d'une documentation ultérieure. En effet, la description du but du programme (pas 1) est un excellent résumé de ses caractéristiques ; la définition des notes détaillées en donne une bonne description technique et un pseudocode bien présenté remplace l'organigramme, devenu inutile.

Le pseudocode

Pour illustrer la façon d'obtenir un pseudocode et sa forme, prenons le problème suivant : lire un nombre entier non négatif ; si le nombre est égal à zéro, le programme est terminé, sinon il doit calculer et éditer séparément les sommes de tous les nombres pairs et impairs lus.
Première tâche : il faut définir clairement le but du programme. Le problème est ainsi résumé : supposons une valeur entière positive ; on analyse si ce nombre est zéro, pair ou impair. Si le nombre lu est zéro, le programme s'achève par l'impression :

— du nombre des nombres pairs introduits,
— de leur somme,
— du nombre des nombres impairs introduits,
— de leur somme.

Si le nombre n'est pas zéro, le compteur et la somme du groupe correspondant sont mis à jour et une nouvelle donnée est ensuite demandée. Cela pour le but.
Nous passons, ensuite, aux notes détaillées du programme, en examinant quelles doivent être les données d'entrée, les calcules demandés et les données de sortie.

Données d'entrée Un nombre entier, de valeur comprise entre zéro et la valeur maximum (entière), pouvant être représenté sur le calculateur (32767 pour une machine à 16 bits).

Elaboration Si le nombre vaut zéro, imprimer les résultats et terminer le calcul, sinon identifier si le nombre est pair ou impair (en le divisant par deux et en contrôlant si le reste vaut zéro). En fonction du test, on augmente le compteur des nombres et on effectue la somme du groupe correspondant. Le calcul se poursuit par la demande d'une nouvelle donnée.

Données de sortie Impression du nombre des nombres impairs émis et de leur somme.

Dans cette phase, nous avons défini les données d'entrée, les calculs nécessaires et les données de sortie.
L'écriture du pseudocode (niveau 3) entre dans le détail du problème en utilisant le langage naturel.
Dans le tableau ci-contre, remarquons la manière non habituelle d'écrire les variables dont le nom est constitué de deux parties reliées par un trait d'union (NOM-IMPA)
Cette symbolique permet d'utiliser une première partie commune et une seconde spécifique. Par exemple, les deux variables NOM-IMPA et NON-PAIR se réfèrent toutes deux à des valeurs numériques (NOM) mais l'un aux impairs (IMPA), l'autre aux pairs (PAIR).
Il ne s'agit cependant que d'une méthode d'identification et non d'une règle.

Les organigrammes structurés

Le but du pseudocode est analogue à celui de l'organigramme (figure de la page 1208) et il est facile de suivre le parcours du programme en utilisant l'indentation des écritures *. Avec des phrases explicites comme « début-fin »,« tant-que-faire », « si-alors-sinon » on identifie mieux les blocs du programme.
Le pseudocode sera ultérieurement affiné par extension de la partie chargée de déterminer si le nombre est pair ou impair. Normalement, ceci n'est pas nécessaire puisqu'on peut aller directement à la phase de détail. Cette solution permet même d'améliorer le programme (voir pages 1209, 1210 et 1211, le programme écrit en Fortran 77). Le code présenté ne résout pas complètement le problème. On pourrait, par exemple, introduire un traitement des erreurs en données d'entrée. Mais les modifications sont aisées étant donné la modularité du programme.

En programmation structurée, toute représentation graphique est, en principe, évitée. Pourtant Nossi et Shneiderman proposent l'**organigramme N-S** (voir p. 1212) qui met en évidence l'articulation des structures (ce qui est moins évident dans un organigramme normal). Pour cette raison, les organigrammes N-S sont définis comme structurés.

* On entend par indentation les marges des diverses phrases de description d'un pseudocode.

Exemple de pseudocode

```
Définitions constantes
    MAXINT      nombre entier maximum
                accepté par la machine
    ZERO        valeur zéro

Définitions variables et leur type
    NOMBRE      variable contenant le
                nombre lu, de type entier
    NOM-PAIR    nombre des valeurs pai-
                res lues, de type entier
    NOM-IMPA    nombre des valeurs im-
                paires lues, de type
                entier
    SOMME-PAIR  somme des valeurs pai-
                res lues, de type entier
    SOMME-IMPA  somme des valeurs im-
                paires lues, de type
                entier

Début
  Initialiser à zéro les variables NOM-PAIR,
  NOM-IMPA, SOMME-PAIR, SOMME-IMPA
  Lire première valeur d'entrée —→ NOMBRE.
  Tant que NOMBRE est différent de ZERO
  faire :
    Début
      Si NOMBRE pair
      alors
        Début
          Incrémenter NOM-PAIR
          Somme NOMBRE à SOMME-PAIR
        Fin
      Sinon
        Début
          Incrémenter NOM-IMPA
          Somme NOMBRE à SOMME-IMPA
        Fin
      Lire —→ NOMBRE
    Fin
  Un zéro lu, imprime NOM-PAIR, SOMME-
  PAIR, NOM-IMPA, SOMME-IMPA
Fin
```

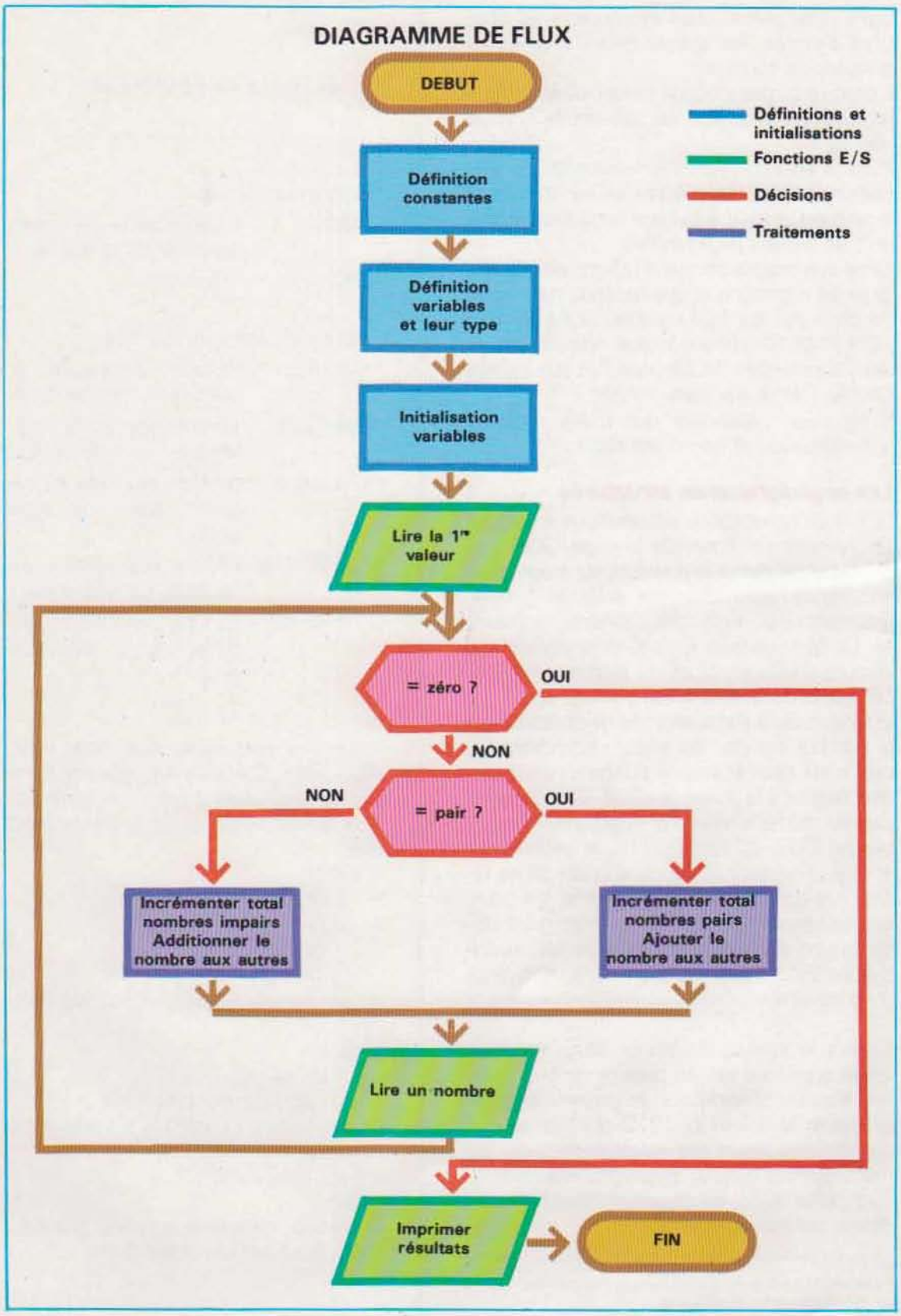


DIAGRAMME DE FLUX
DEBUT
Définition constantes
Définition variables et leur type
Initialisation variables
Lire la 1re valeur
= zéro ?
OUI
NON
= pair ?
NON
OUI
Incrémenter total nombres impairs Additionner le nombre aux autres
Incrémenter total nombres pairs Ajouter le nombre aux autres
Lire un nombre
Imprimer résultats
FIN
Définitions et initialisations
Fonctions E/S
Décisions
Traitements

N.B. La codification en FORTRAN ANSI 77 ne permet pas d'utiliser le tiret à l'intérieur des noms de variables ; NOM-IMPA est donc codifié NOMIMPA

EXEMPLE DE PROGRAMMATION STRUCTUREE EN FORTRAN

PROGRAMMEUR JACQUES RIME DATE 31 MAI 84

FORTRAN

PROGRAMME PAIR OU IMPAIR ?


```
      PROGRAM  PAIRIMPAIR
      IMPLICIT NOM
C     DEFINITION VARIABLES ET TYPE
C     NOMBRE --> VALEUR LUE                          ENTIER
C     NOMPAIR --> NOMBRE DES VALEURS PAIRES          ENTIER
C     NOMIMPA --> NOMBRE DES VALEURS IMPAIRES        ENTIER
C     SOMMEPAIR --> SOMME DES VALEURS PAIRES         ENTIER
C     SOMMEIMPA --> SOMME DES VALEURS IMPAIRES ENTIER
C
      INTEGER*2  NOMBRE, NOMPAIR, NOMIMPA, SOMMEPAIR, SOMMEIMPA
C
C        MISE A ZERO DES VARIABLES
         NOMPAIR=Ø
         NOMIMPA=Ø
         SOMMEPAIR=Ø
         SOMMEIMPA=Ø
C
C        LIRE UN NOMBRE
         PRINT *,' IMPRIMER UN NOMBRE ENTIER POSITIF OU NUL'
         READ  *, NOMBRE
```

1210

PROGRAMMEUR JACQUES RIME DATE 31 MAI 84 FORTRAN

PROGRAMME PAIR OU IMPAIR ? Page 2 de 3

```
C     TANT QUE LE NOMBRE N'EST PAS EGAL A ZERO FAIRE
C
      DO WHILE (NOMBRE.NE.Ø)
         PRINT *,'LE NOMBRE VAUT',NOMBRE
C
C        SI LA VALEUR EST PAIRE: ANALYSE AVEC LA FONCTION ONE MOD QUI
C                                DONNE LE RESTE D'UNE DIVISION ENTRE
C                                ENTIERS
         IF (MOD(NOMBRE,2).EQ.Ø) THEN
C
C        SI LE RESTE DE LA DIVISION ENTRE LA VALEUR ET 2 VAUT ZERO
C                                ALORS LE NOMBRE EST PAIR
C
            NOMPAIR=NOMPAIR + 1
            SOMMEPAIR=SOMMEPAIR + NOMBRE
C
C                                SINON LE NOMBRE EST IMPAIR
C
         ELSE
            NOMIMPA=NOMIMPA + 1
            SOMMEIMPA=SOMMEIMPA + NOMBRE
```

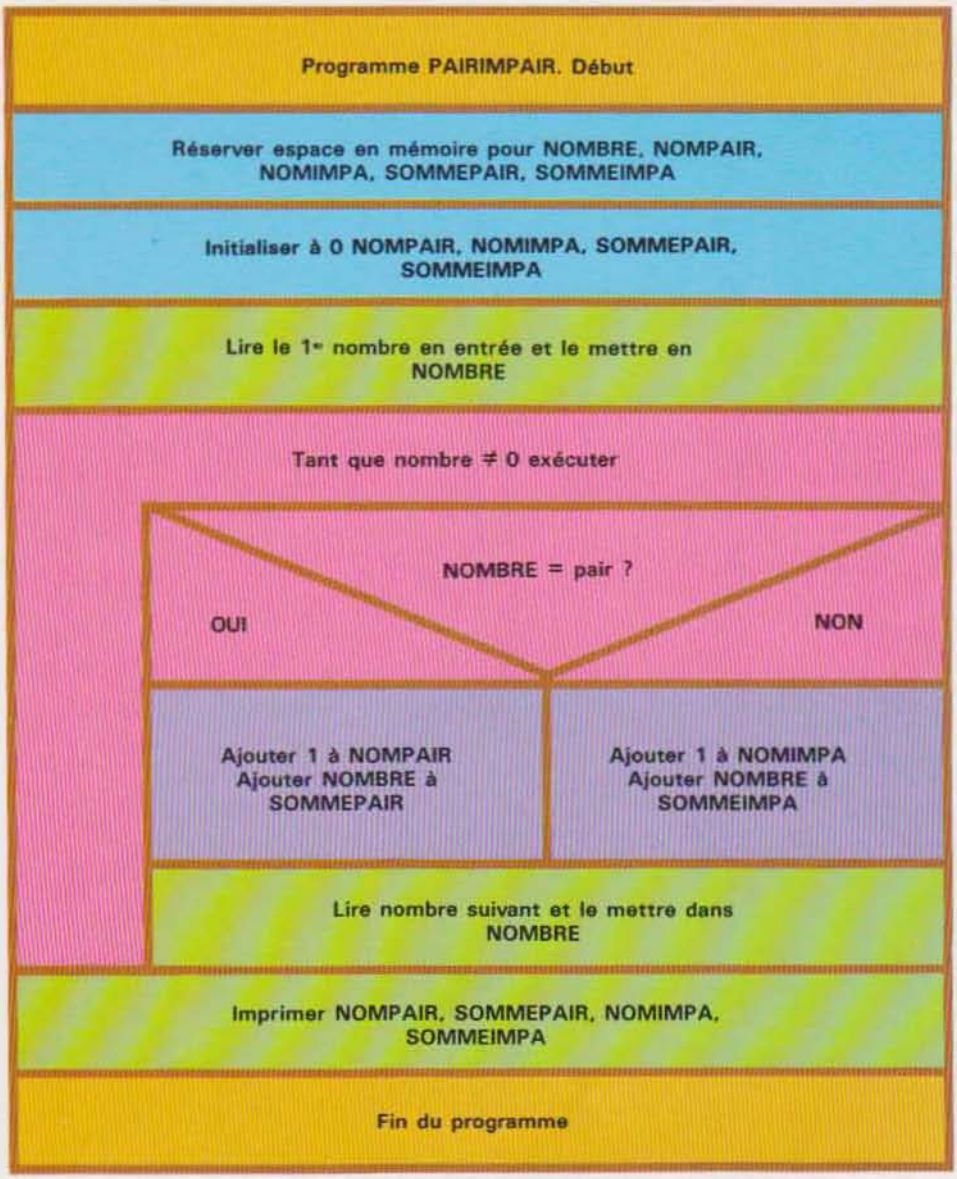
PROGRAMMEUR JACQUES RIME DATE 31 MAI 84 FORTRAN

PROGRAMME PAIR OU IMPAIR ? Page 3 de 3

```
            ENDIF
C
            PRINT *,'TAPER UN AUTRE NOMBRE OU ZERO (Ø) POUR SORTIR'
            READ (*,END=99) NOMBRE
C
         END DO
C
C        LIRE UN ZERO, IMPRIMER RESULTATS
C
   99    PRINT *, 'NOMBRE DES VALEURS PAIRES=',NOMPAIR
         PRINT *, 'SOMME DES VALEURS PAIRES=',SOMMEPAIR
         PRINT *, 'NOMBRE DES VALEURS IMPAIRES=',NOMIMPA
         PRINT *, 'SOMME DES VALEURS IMPAIRES=',SOMMEIMPA
C
         STOP
C
      END
C
C     FIN DU PROGRAMME PAIR OU IMPAIR
```

1211

ORGANIGRAMME "STRUCTURE" (DIAGRAMME N-S)



Dans la figure ci-dessus, notons bien que chaque bloc est représenté par un symbole particulier ainsi que les structures « tant-que-faire » et « si-alors-sinon ».
Par la suite, nous analyserons, pour les deux types de représentation, les organigrammes représentatifs des opérations fondamentales qui composent un programme.

Les structures de contrôle

Les opérations effectuées à l'intérieur d'un programme peuvent se combiner entre elles autant de fois qu'il est nécessaire, grâce à trois structures fondamentales :

1 / Séquence
2 / Sélection
3 / Répétition

Séquence. C'est la structure de base qui revient le plus souvent dans un programme. Elle consiste (voir ci-dessous) en une succession d'opérations à effectuer l'une après l'autre. Le niveau de détail, avec lequel les contenus des blocs sont décrits, dépend du niveau d'affinement que l'on désire atteindre.
Dans certains langages comme l'Algol et le Pascal, le début et la fin d'une séquence sont mis en évidence par les deux mots clés BEGIN et END. Dans d'autres cas, le début et la fin d'une séquence sont identifiés de manière implicite puisqu'elle est enfermée entre deux clés appartenant à une structure différente.

Sélection. Deux séquences sont **combinées en sélection** quand, selon qu'une condition se trouve vérifiée ou non, l'une ou l'autre d'entre elles est exécutée. Par exemple, page 1214, dans les organigrammes de droite, la séquence S1 ou la séquence S2 est exécutée en fonction du résultat.
On trouve, dans presque tous les langages de programmation, la phrase IF-THEN-ELSE, c'est-à-dire SI-ALORS-SINON.
En Fortran 77, tout comme dans le programme précédent, il existe plusieurs instructions insérées entre THEN et ELSE. On peut donc trouver une séquence soit entre les mots clés

STRUCTURE DE CONTROLE DE TYPE "SEQUENCE"

Diagramme ordinaire

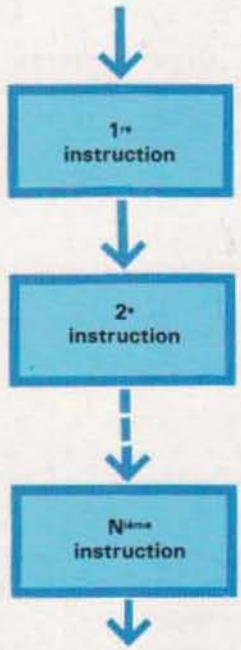
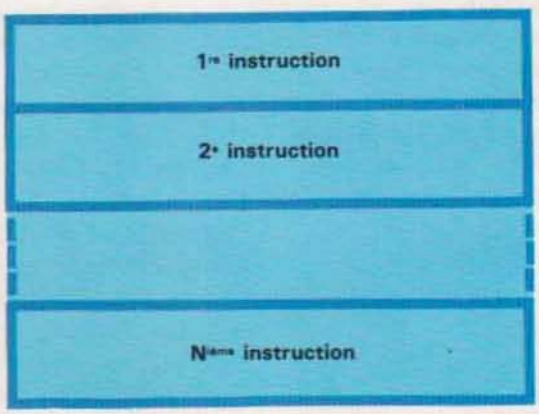
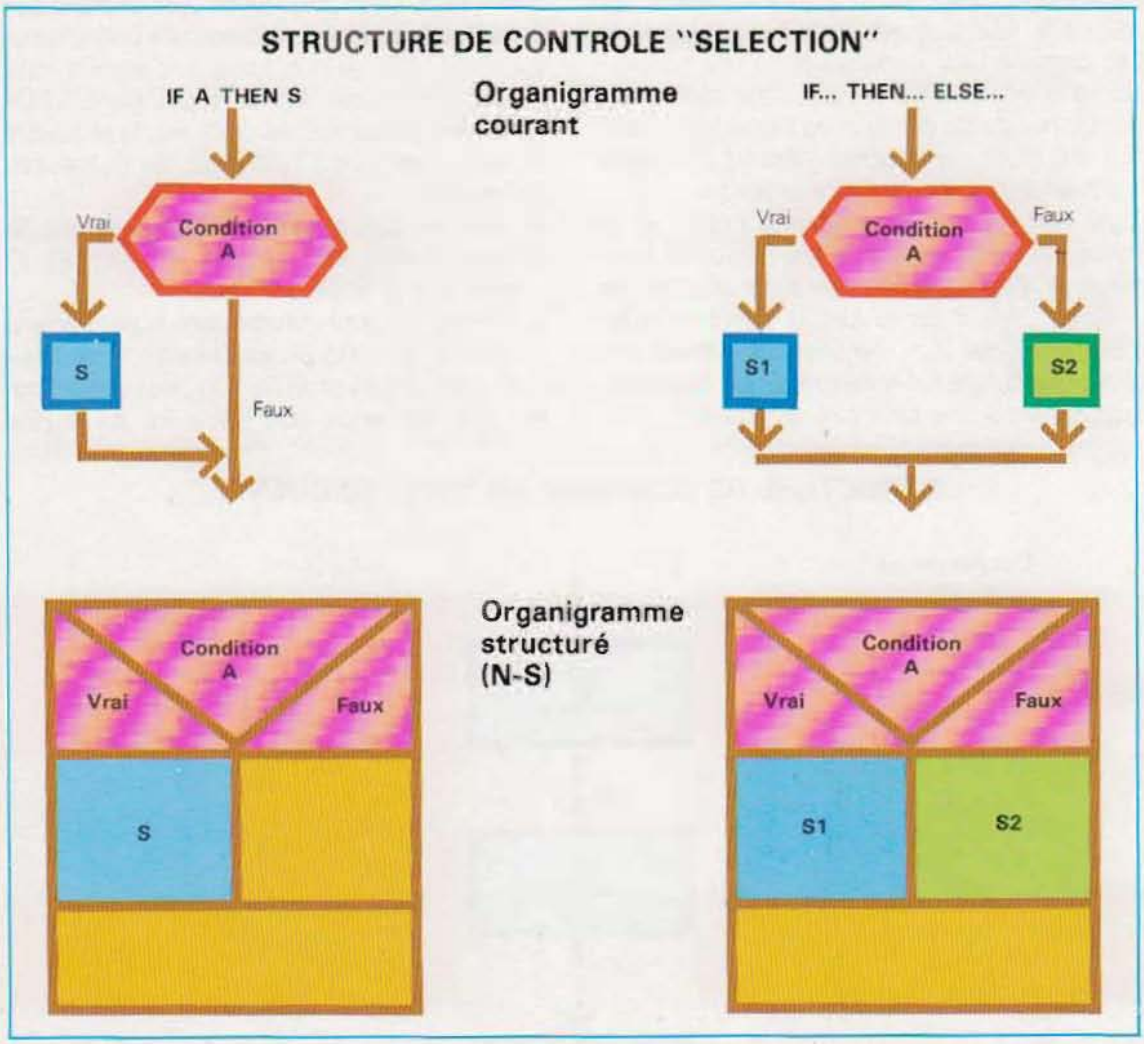


Diagramme structuré





THEN et ELSE, soit entre le ELSE et la fin de la sélection (mot clé ENDIF). Mais une séquence peut également ne contenir aucune instruction sur le mot clé ELSE. Elle signifie : « si la condition n'est pas vraie, poursuivre en séquence sans réaliser aucune opération ».
Ne sont généralement admises, dans le Cobol, que les séquences formées par une seule instruction, alors qu'en Pascal et en Algol, la série de plusieurs instructions est toujours précédée du mot-clé BEGIN et fermée par le mot-clé END. Ces deux clés jouent un rôle analogue à celui des parenthèses.
Ainsi, les structures :

```
IF condition THEN
   Instruction 1
Instruction 2
```

et les structures :

```
IF condition THEN
   BEGIN
      Instruction 1
      Instruction 2
   END
```

sont totalement différentes.
Dans le premier cas : si la condition est vraie, seule la première instruction est exécutée. Dans le second cas : tout ce qui est entre les délimitateurs BEGIN et END est exécuté.

Répétition. Les structures de répétition sont fondamentales lorsque l'opération globale à effectuer est constituée d'une succession finie, mais pas toujours connue a priori, d'opérations égales. Avec cette technique, il est possible d'utiliser plusieurs fois une même série d'instructions. Il existe deux structures fondamen-

tales de contrôle de répétition : WHILE-DO (TANT QUE-FAIRE) et REPEAT-UNTIL (REPETER-JUSQU'A), illustrées ci-dessous et page suivante. Même si elles paraissent identiques à première vue, elles sont conceptuellement différentes. Dans WHILE-DO, on procède d'abord au test de la condition et la séquence n'est exécutée que si cette condition s'avère vraie. Dans REPEAT-UNTIL, c'est l'inverse : est d'abord exécutée la séquence S, puis seulement le test ; si la condition s'avère fausse, la séquence est répétée ; sinon on sort de la structure (boucle).

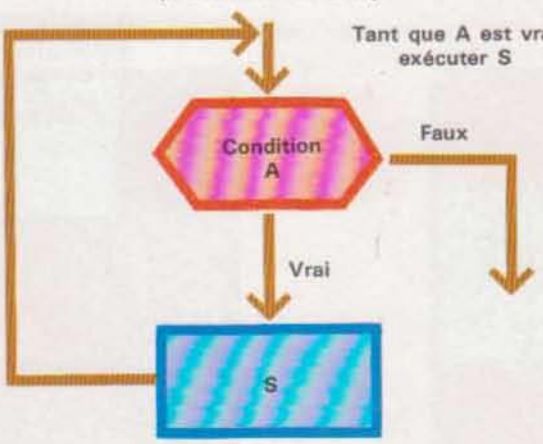
En Pascal et en Algol, si la séquence est composée de plus d'une instruction, les parenthèses BEGIN-END sont nécessaires dans le WHILE-DO alors qu'elle ne le sont dans le REPEAT-UNTIL puisque le début de la séquence est défini par le mot-clé REPEAT et la fin par UNTIL. Au contraire, en Fortran 77, la séquence est refermée entre DO-WHILE et ENDDO. Il n'existe pas, dans ce langage, un véritable format REPEAT-UNTIL qui peut être remplacé par les instructions :

```
S1 Instruction
   IF (.NOT. condition) GOTO S1
```

ou, dans d'autres cas, par un simple DO-ENDDO.

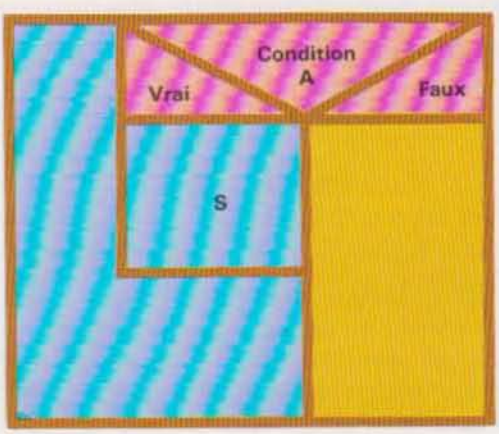
STRUCTURE DE CONTROLE "REPETITION" (WHILE... DO...)

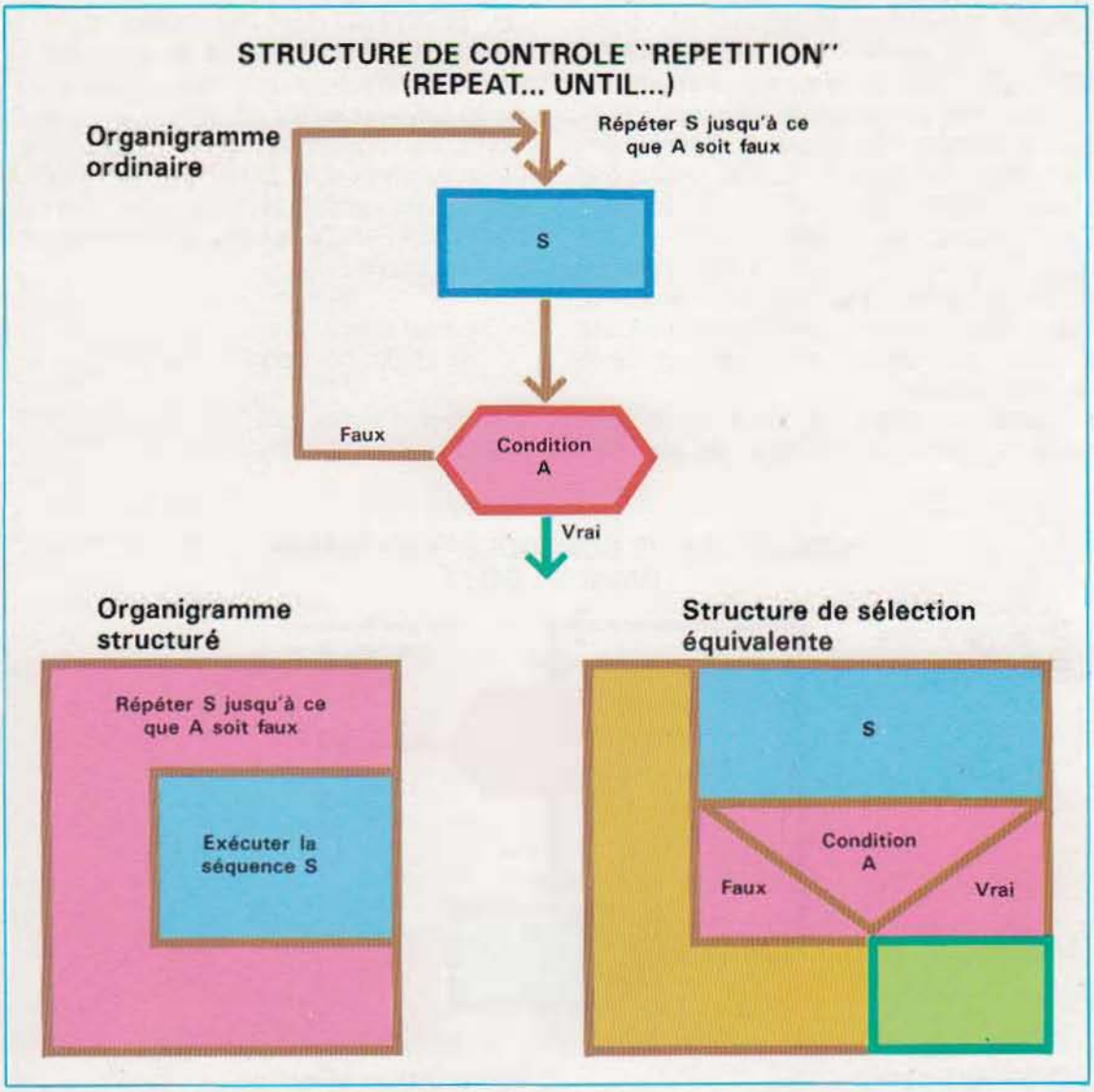
Organigramme ordinaire



Organigramme structuré

Structure de sélection équivalente





En Cobol, WHILE-DO se déduit ainsi :

```
PERFORM nom-procédure
UNTIL expression
```

de même que pour REPEAT-UNTIL :

```
Début d'expression à FAUX
PERFORM nom-procédure
UNTIL expression
```

sauf si la condition ne peut être mise à faux (cas de fin de fichier par exemple). Toutes ces structures constituent le noyau principal de la programmation structurée et sont suffisantes pour décrire n'importe quel algorithme même s'il en existe d'autres.

Les données en Pascal

Un programme écrit en Pascal est structuré en deux parties : la déclaration des données et l'ensemble des instructions vraies et propres. La première définit les types de données qui seront utilisés par le compilateur et améliore la lisibilité du programme, la seconde contient les instructions qui décrivent la méthode de calcul sur les données précédemment spécifiées.

Types de données

L'innovation la plus significative, introduite par le Pascal, est la formulation du concept de

type de donnée. On dit que des constantes ou des variables appartiennent au même type si elles possèdent les mêmes caractéristiques et si elles sont soumises au calcul de la même manière. Par exemple, l'ensemble des nombres entiers constitue le type de donnée entière.
En Pascal, les types de données élémentaires sont définis comme des types **scalaires.** Ceux-ci ne sont pas décomposables en types plus simples. En revanche, ils peuvent se combiner de manière différente pour former les **types structurés**. Par exemple, un vecteur de nombres entiers de 10 éléments :

IA(I) I = 1,..., 10

est de type structuré formé de dix données scalaires IA(1), IA (2)..., IA(10).
Les types de données scalaires se subdivisent en **types standard** et en **types définis par l'utilisateur.** Les types standard sont des types prédéfinis par le Pascal. Ce sont les types entier, réel, caractère, booléen, utilisables sans spécification du champ de validité. Un tel champ est en effet automatiquement établi par le compilateur Pascal (comme en langage Fortran quand on déclare une variable de type entier ou réel).
En revanche, les types de données définis par l'utilisateur sont une caractéristique du Pascal qui permet à cet utilisateur de créer des types ad hoc. Il établit le champ de validité et les éléments d'appartenance pour résoudre les problèmes spécifiques.
Les types de données structurées répondent aux types suivants :

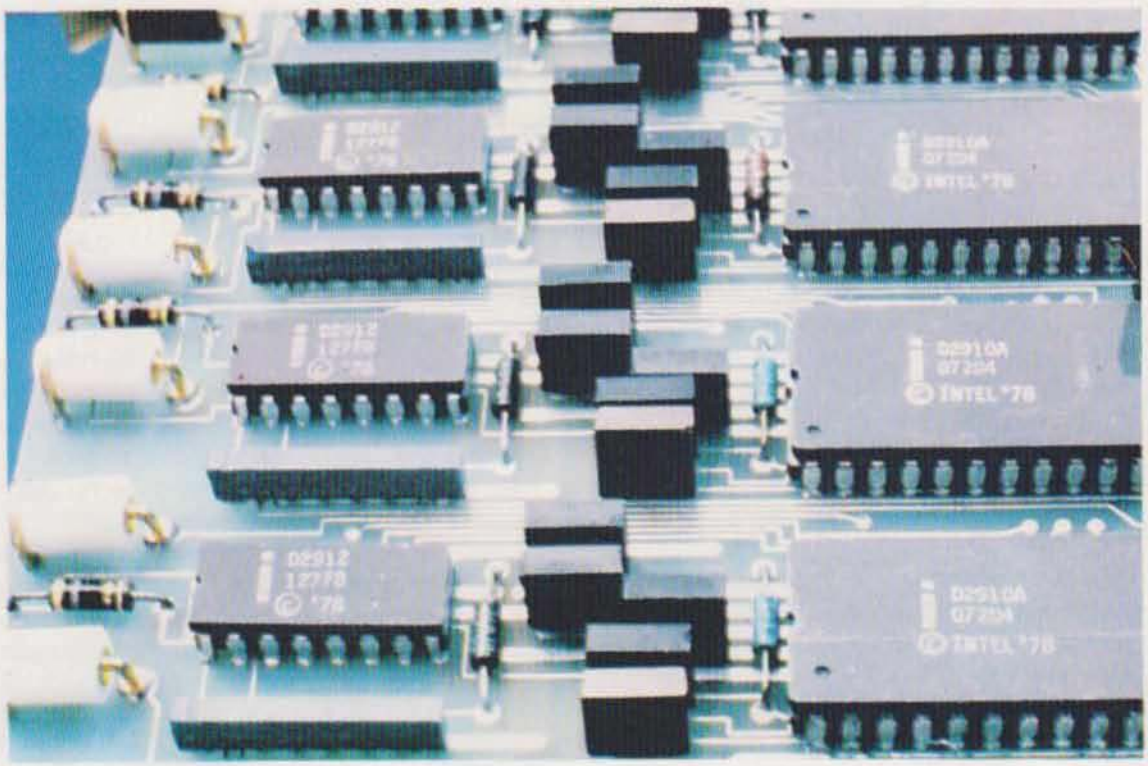
— **array** (tableau), terme générique avec lequel on indique les vecteurs et les matrices à deux ou à plusieurs dimensions,

— **set** (jeu), un ensemble d'éléments tout à fait semblables à l'array qui sont cependant traités comme un élément unique,

— **file** (fichier) qui signifie un ensemble d'enregistrements.

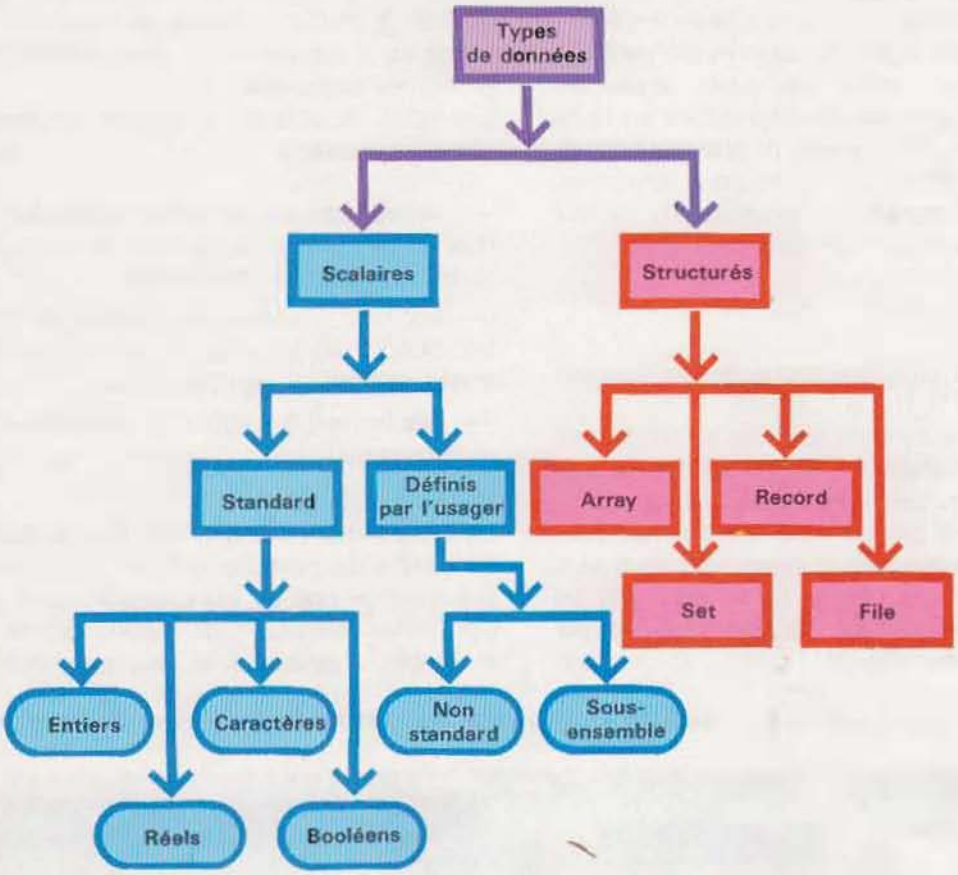
Le type entier. Ces données appartiennent à l'ensemble des nombres naturels (sans virgule) précédés ou non du signe positif ou négatif. Les limites inférieures ou supérieures de cet ensemble, c'est-à-dire le plus petit nombre

Circuits intégrés Intel montés sur la carte-mère d'un système.



J. Pickerell/Marka

TYPE DE DONNEES ADMIS EN PASCAL



négatif et le plus grand nombre positif, dépendent évidemment de la capacité de l'ordinateur utilisé. Certaines machines à mots de 36 bits admettent, pour les nombres entiers, une extension de -2^{35} à $2^{35}-1$, c'est-à-dire de 34 359 738 368 à + 34 359 738 367. Ordinairement on utilise 16 bits pour la représentation des nombres entiers. On a ainsi une extension des valeurs de — 32 768 à + 32 767.

Pour éviter d'amoindrir la capacité de l'ordinateur, notamment lorsque les programmes sont transportés d'une machine à l'autre, une variable entière, appelée MAXINT, est prédéfinie et utilisée par le compilateur Pascal. Cette variable contient le plus grand entier utilisable sur le calculateur dont on se sert. La connaissance de la valeur de cette variable permet de faire abstraction du mot-machine. Dans la page ci-contre est montrée la syntaxe formelle de l'écriture d'une donnée de type entier selon une notation particulière appelée **diagramme syntaxique**. Cette notation, qui sera désormais souvent utilisée, exprime de façon compacte les règles syntaxiques du Pascal.

Comme la notation des organigrammes, elle indique comment construire un type de donnée ou une instruction en combinant, dans les divers champs, les caractères alphanumériques et les symboles de séparation admis. Selon ce type de syntaxe, toutes ces propositions (nombres et instructions) qui peuvent être pro-

duites (en partant à gauche du diagramme, en suivant les flèches et en sortant à droite) s'avèrent valables. Par exemple, étant donné le diagramme syntaxique qui se trouve en bas de page, les nombres :

1
1 2 3
1 2 1
1 2 1 2 3

sont syntaxiquement valables.

En revanche le nombre :

1 3

n'est pas correct puisqu'il n'existe aucun chemin allant de gauche à droite en passant seulement par les nombres 1 et 3 sur le diagramme.

De même, les nombres :

1 2
2 3
1 2 1 3
1 2 3 3

ne sont pas syntaxiquement valables.

Revenons au diagramme « de type entier » : observons que le type de donnée entier est constitué par un nombre de chiffres décimaux qu'on peut prolonger (dans les limites, évidemment, des capacités de l'ordinateur). L'ensemble des chiffres est précédé d'un symbole + ou —, ou d'aucun symbole, indépendamment du chemin choisi sur les trois branches du diagramme. On pourra donc obtenir :

16384
— 14
+ 250
0

Voici les opérations sur les nombres entiers admises par Pascal :

1 / Opérations arithmétiques (addition, soustraction, multiplication)

2 / Opérations de division DIV et MOD, avec les significations suivantes :

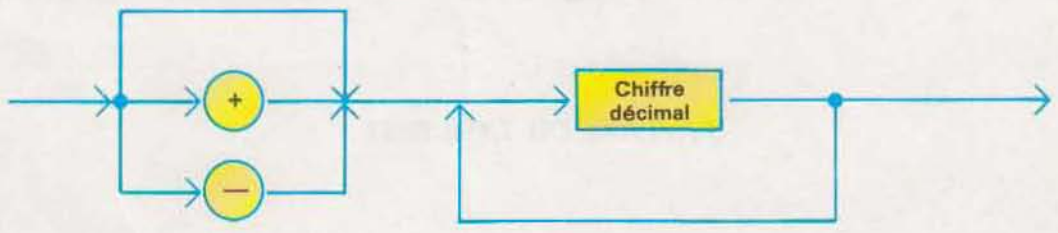
DIV division entre entiers, sans la partie fractionnaire ; par exemple, 29 DIV 8 = 3.

MOD reste de la division entre entiers ; par exemple 29 MOD 8 = 5

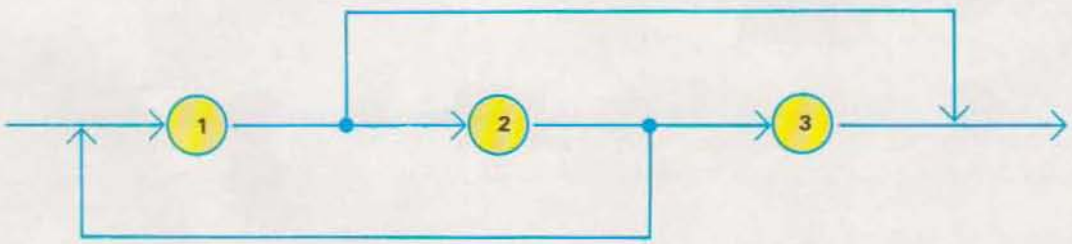
3 / Opérations relationnelles

> supérieure
= égal
< inférieur
>= supérieur ou égal
<= inférieur ou égal
<> non égal (différent)

SYNTAXE DU TYPE ENTIER



EXEMPLE DE DIAGRAMME SYNTAXIQUE



Contrairement aux résultats des deux premiers groupes d'opérations, celui d'une opération relationnelle entre deux nombres entiers n'est pas un entier mais une variable booléenne avec une valeur « vrai » ou « faux ».
Il existe enfin certaines fonctions opérant aussi bien sur des entiers que sur des réels et qui fournissent un résultat entier. Ce sont :

ABS (I)	fournit la valeur absolue de l'entier 1, c'est-à-dire le nombre 1 sans signe
SQR (I)	élévation au carré du nombre 1
ROUND (R)	fournit le nombre entier le plus proche du réel R ; en notation américaine ROUD (4.87) = 5
TRUNC (R)	fournit la partie entière de R ; par exemple TRUNC (4.87) = 4

Le type réel. Ce type se réfère à l'ensemble des nombres ayant une partie entière et une partie fractionnaire.
Ces nombres peuvent être représentés :

— dans la **notation décimale** ordinaire,
— dans la **notation exponentielle** ou scientifique.

Celle-ci est constituée d'une mantisse (nombre décimal avec une partie entière d'au moins un chiffre) suivie de la lettre E et de la puissance entière de 10 qui, multipliée par la mantisse fournit la valeur du nombre réel (d'une manière tout à fait analogie au Fortran et au Basic). Par exemple, le nombre 1.237E03 équivaut à 1.237×10^3, c'est-à-dire à 1237.
Voir ci-dessous la syntaxe de la représentation des données réelles.
On peut contrôler, en suivant le diagramme syntaxique, la validité des nombres :

— 32.1
+ 8.3E—9
— 42.0
— 6.9E—19

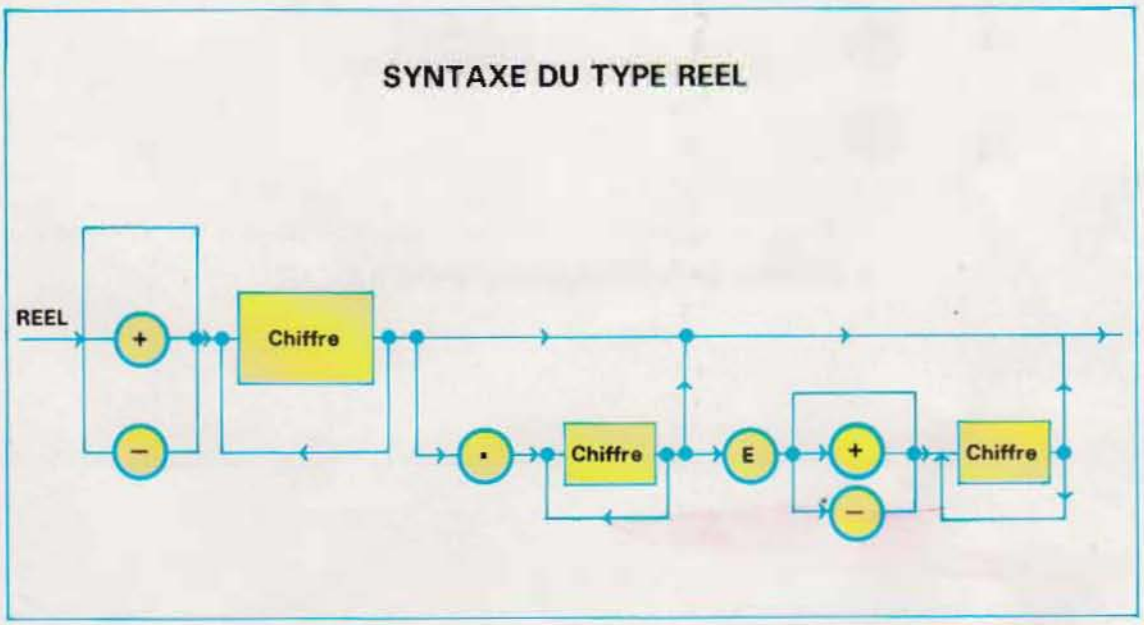
et la non-validité des nombres :

.027	(aucun chiffre avant le point décimal)
8.57E+.2	(la puissance décimale n'est pas admise)

Pour les réels, comme les entiers, il existe des limitations sur l'ensemble des nombres représentables. Ces limitations concernent aussi bien la précision (le nombre des chiffres mais après le point décimal) que les valeurs extrêmes représentables.

Pour un micro-ordinateur, les valeurs réelles négatives ont comme intervalle celui délimité par — 1.0E + 38 et — 1.0E — 39, alors que

SYNTAXE DU TYPE REEL



SYNTAXE DU TYPE CARACTERE

les données réelles positives peuvent être comprises entre + 1.0E — 39 et + 1.0E + 38. Les opérations admises sur les réels sont :

1 / les quatre opérations algébriques +, —, *, /

2 / les opérations relationnelles de manière tout à fait analogue à ce qui a été vu pour les nombres entiers.

De plus, sont admises les fonctions suivantes dont le résultat est un nombre réel :

ABS(R)	valeur absolue du nombre R
SQR(R)	carré de R
SQRT(R)	racine carrée de R
LN(R)	logarithme en base e de R
EXP(R)	e élevé à R
SIN(R)	sinus de R
COS(R)	cosinus de R
ARCTAN(R)	arctangente de R

Le type caractère. En Pascal, le type caractère s'étend à tous les symboles alphanumériques et de ponctuation connus de l'ordinateur. A cause des différences entre les symboles de chaque clavier, l'ensemble des données de type caractère n'est pas toujours le même. On doit en tenir compte lorsque l'on transporte les programmes d'un ordinateur à un autre.
Cet ensemble minimum contient 26 lettres majuscules, de A à Z (l'alphabet entier), les chiffres de 0 à 9 et les caractères spéciaux qui indiquent la ponctuation et les opérateurs, en plus de l'espace. La syntaxe d'écriture des données de type caractère est montrée dans le schéma ci-dessus. Comme on peut le voir, une donnée caractère est toujours enfermée entre deux apostrophes, comme dans les exemples suivants :

'P'
'7'
' '

Le type caractère ne doit pas être confondu avec le type chaîne, lequel, comme ensemble de caractères, représente une donnée de type structuré (c'est-à-dire composé d'un ensemble de données scalaires). Par exemple, la chaîne

'PROGRAMME'

est composée des caractères simples 'P' 'R' 'O' 'G'' R' 'A' 'M' 'M' 'E'.
Le type structuré chaîne sera décrit de manière détaillée plus loin. Les fonctions opérant sur les données de type caractère sont :

CHR (I)	Restitue le caractère correspondant au nombre I dans la codification interne des caractères dans l'ordinateur [en général un nombre entier compris entre 0 et 128 (ou 255)].
ORD ('C')	C'est la fonction inverse de CHR (I). Il restitue un entier correspondant à la représentation interne du caractère C.
PRED ('C')	Fonction de prédécesseur. Restitue le caractère précédant immédiatement le caractère C dans le jeu des caractères. Par exemple, PRED 'B' est égal à 'A'.
SUCC ('C')	Fonction de successeur. Restitue le caractère suivant immédiatement le caractère C ; par exemple, SUCC 'B' est égal à 'C'.

Il est intéressant de noter que les deux dernières fonctions s'appliquent aussi à des données de type entier ; par exemple :

PRED (5) = 4
SUCC (5) = 6

Les jeux vidéo : pourquoi ?

Le succès commercial des jeux vidéo est indiscutable. Même si certains considèrent qu'ils ont atteint le stade de la saturation dans des pays comme les Etats-Unis et le Japon, il reste évident que le jeu est désormais une donnée de notre civilisation.

Le jeu a toujours existé, et à toutes les époques. Ce qui est nouveau, avec le phénomène des jeux vidéo, c'est l'interaction entre une ***haute technologie,*** *extrêmement* ***chargée affectivement,*** *et l'usage traditionnel du jeu.*

Ce phénomène aurait pu n'être qu'éphémère si le secteur industriel qui en est à l'origine ne s'était mis soudain à produire à la fois la technologie et les sujets, qui satisfont et motivent constamment notre curiosité et notre soif d'aventures.

A ce propos, on peut se souvenir que les automates ludiques ne sont pas nouveaux. Ceux de Vaucanson sont restés célèbres à juste titre. La différence, par rapport à aujourd'hui, c'est que la technologie ne fournit pas seulement l'instrument, mais aussi le sujet du ***rêve.*** *Elle ne s'adresse pas seulement à notre intellect, mais aussi à notre émotion. C'est pourquoi il naît tant de jeux chaque mois.*

En effet, notre secteur émotif est infiniment plus diversifié et surtout plus changeant que notre secteur intellectuel. Et surtout, il dispose de moins de soupapes de sécurité que l'intellect. On assiste donc à un « détournement » intéressant et curieux de la technologie de pointe qu'est l'électronique/informatique. Ce secteur, réputé rébarbatif et hautement adulte, nourrit en fait une partie très primitive et très proche de ***l'enfance*** *de notre individu !*

Il est à la mode d'affirmer que ces jeux sont mauvais, en arguant de leur pauvreté culturelle et éducative, acceptée et même choisie pour des questions de facilité commerciale. On ajoute aussi qu'ils induisent un comportement passif chez l'individu qui se laisse « prendre au jeu ». En fait, nous retrouvons là l'éternelle querelle des ***modernes*** *et des* ***anciens.***

On a dit bien du mal de la télévision à ses débuts, comme de tout système intégrant largement l'image. On constate aujourd'hui que les systèmes audiovisuels produisent des œuvres de valeur. Qui, aujourd'hui, oserait affirmer que le cinéma n'a produit aucun chef-d'œuvre ? La télévision a produit des émissions de qualité, surtout dans de nouvelles visions de l'actualité. Qui ne se souvient du très célèbre « Cinq colonnes à la une » ?

Un autre aspect du jeu qu'il convient d'aborder est son rapport avec la civilisation qui l'a fait naître. La tournure de cette phrase est fortement affirmative quant à « l'activité » du jeu. Ce n'est pas anodin. Disons qu'une civilisation sécrète ses jeux comme une plante produit des fleurs.

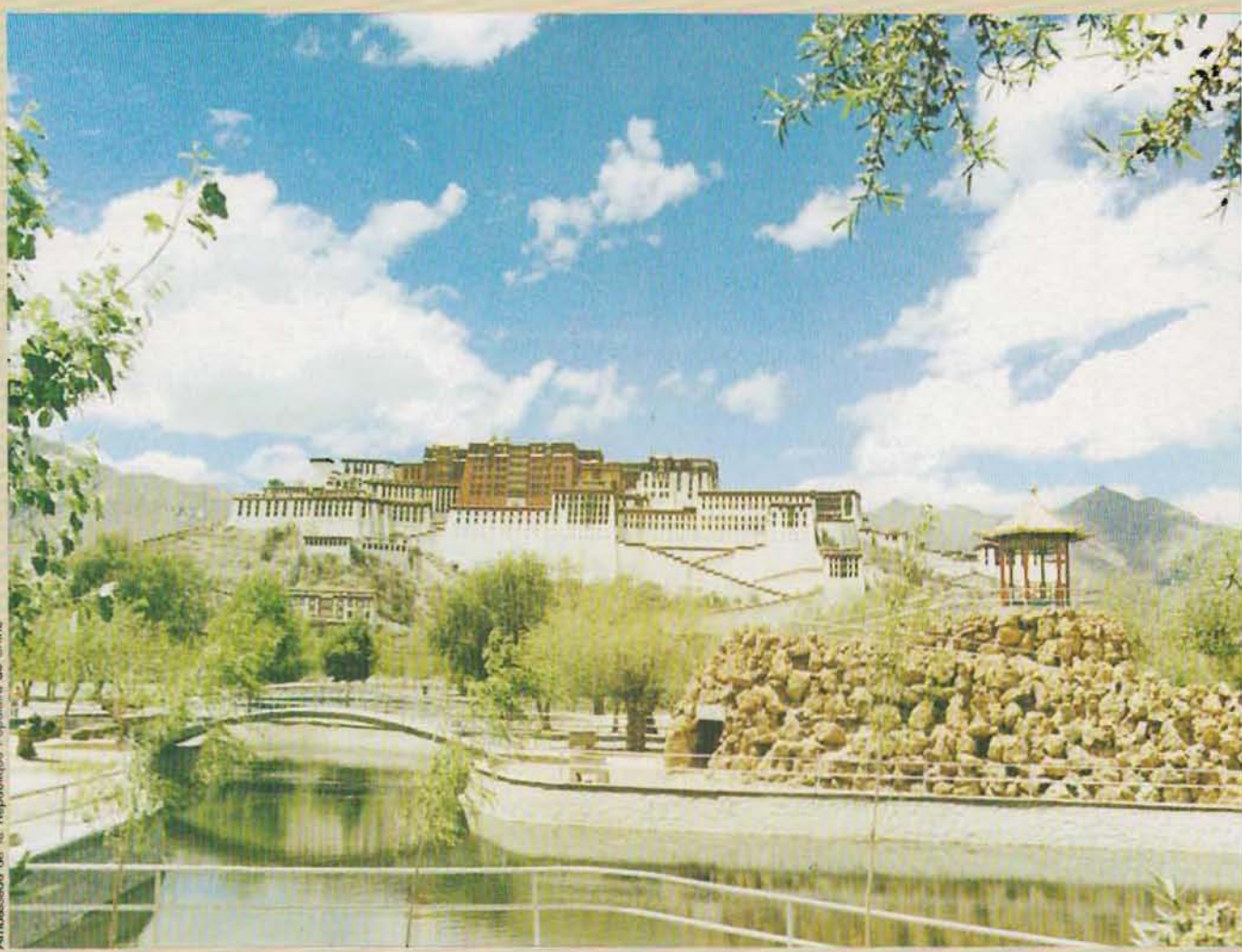
Une civilisation qui a oublié le jeu est une civilisation qui meurt. Elle connaît, quelquefois, des sursauts, mais ceux-ci restent tristement célèbres dans l'histoire, comme les jeux du cirque chez les Romains. Une civilisation qui ne sait plus jouer, ou qui en est réduite à caricaturer la réalité, s'est coupée de toute son enfance, ou tout au moins, de ses forces vives, de sa capacité d'inventer et de se renouveler.

Pensons, à ce sujet, à deux très grands mathématiciens : Poincaré et Einstein. Le premier était un pur produit de son époque et, aujourd'hui, seuls les spécialistes se souviennent de lui. Einstein, lui, a tellement ***rêvé le futur,*** *tout en s'appuyant sur la fine pointe du raisonnement mathématique et physique de son époque, qu'il nous a offert les instruments de conception de notre futur. Il rêvait beaucoup, ce très grand homme, et ses réflexions philosophiques en témoignent. Alors, abordons la question des jeux vidéo avec ouverture d'esprit, avec une saine curiosité et le désir de se mesurer honnêtement à cette émanation de notre actualité.*

Et, surtout, ne trouvons pas de mauvaises raisons pour les condamner, sous prétexte que « tout cela n'est pas sérieux, sauf pour les marchands de soupe » ! Le jeu a toujours existé, en tant que ***nécessité de vie,*** *et la plupart du temps, en tant que* ***simulation d'apprentissage.***

Un des jeux les plus passionnants (et très facilement informatisable), avec les échecs, est le jeu de Gô, dont la pratique quotidienne a été conseillée aux samouraï depuis des siècles.

Les historiens pensent que le jeu de Gô est né au Tibet environ 2 000 ans avant J.C. Il passe ensuite en Chine, en même temps que les pratiques magiques et les recettes d'alchimie, aux alentours du VI^e^ siècle et, enfin, atteint le Japon en 754. Là, il y a prospéré au point de devenir



Ambassade de la République Populaire de Chine

Le Gô, l'un des jeux de guerre qui a conquis le monde, est né dans un monastère tibétain.

partie intégrante de la vie quotidienne de l'aristocratie guerrière.

D'une certaine façon, on peut dire que l'âme de l'ancien Tibet a achevé de s'épanouir au milieu de ces hommes et de ce sol tout autant durs que raffinés.

Ce jeu est porteur à la fois d'une signification morale et philosophique, mais aussi constitue un excellent ***kriegspiel*** *(ou* ***wargame).*** *Ce qui est normal. On n'est vraiment prêt à affronter un adversaire que lorsqu'on a su se mesurer à soi-même. Et dans les vastes plateaux isolés et glacés du Tibet, un tel jeu ne pouvait que s'épanouir car il constituait un moyen privilégié de* ***développement mental.***

Les extrêmes se rejoignant utilement, on conçoit qu'un jeu issu d'un lieu connaissant la solitude physique convienne tout autant à un pays connaissant la surpopulation. C'est que la nécessité d'affrontement (ou de non affrontement) est la même dans les deux cas, l'adversaire est puissant et hautement estimable... et les questions de territoire sont réellement primordiales.

Comme la situation démographique japonaise est en train de s'étendre au monde entier, le jeu de Gô acquiert donc une ***importance mondiale.*** *Si le Japon reste leader en comptant toujours les meilleurs joueurs du monde (dix millions de joueurs, soit 10% de sa population, enfants compris), les clubs de Gô prolifèrent dans les grandes villes et sur les campus universitaires aux Etats-Unis.*

Une des raisons de la séduction du Gô — et certains joueurs d'Echecs se mettent à le préférer à leur habituelle passion — est la ***simpli-***

cité de ses règles** alliée à la **complexité de sa combinatoire.

Les Echecs simulent un tournoi de chevaliers de style médiéval (en fait le jeu remonte à l'ancien Iran et était l'apanage de sa chevalerie), une guerre entre châtelains qui se combattent d'une forteresse à l'autre et qui envoient au combat des guerriers qui succombent à de cruelles blessures.

*Le Gô est plus subtil : la survivance et la victoire ne s'obtiennent pas par l'anéantissement physique de l'adversaire, mais pas la **conquête exprimée en espace vital.** Bref, le combat se fait en terme d'économie politique et de géopolitique. C'est l'hégémonie stratégique qui réduit l'adversaire en appauvrissant ses ressources et sa vitalité.*

Dans le Gô, l'habileté manuelle est réduite au minimum et l'intelligence stimulée au maximum. Le Gô se joue sur un échiquier carré comportant 19 × 19 intersections. L'échiquier japonais, le Gô Ban, comporte 361 pions (181 noirs et 180 blancs) disposés par les joueurs en fonction des croisements des lignes sur la table. La tradition veut que les pions noirs soient fabriqués à base de plaques d'ardoise et les pions blancs sculptés dans la nacre.

Le but du joueur de Gô est de conquérir le plus de territoire possible, la superficie étant fonction des espaces libres. Les joueurs déplacent l'un après l'autre leurs propre pions en fonction de la stratégie adoptée. Les déplacements font inévitablement naître des conflits... Ceux-ci sont régis par des règles précises qui dictent les choix des joueurs. Les pions sont placés un à un sur les déplacements libres, ce choix restant définitif.

*Dans l'intention de dominer (en l'**encerclant**) un grand espace du Gô Ban, le joueur peut se trouver dans la nécessité d'éliminer certains pions de l'adversaire. Pour y parvenir, il doit suivre la procédure de capture, par l'intermédiaire de laquelle il pourra retirer du Gô Ban les autres pions qu'il détient comme prisonniers.*

Les dessins de la page 1225 montrent quelques exemples de zones circonscrites de pions de la même couleur. Les points marqués d'une croix appartiennent respectivement à l'espace du Blanc (A,B) et du Noir (C). A ce stade du jeu, le Blanc a quatre points en A (l'espace dont il s'agit est complètement cerné de pions blancs) et quatre points en B (dans un espace qui est cerné seulement d'un côté par les pions et de l'autre par la limite de l'échiquier). Dans ce dernier cas, le joueur bénéficie de la règle qui permet d'utiliser les côtés extérieurs du Gô Ban dans le but de s'adjoindre l'aide de « pions fantômes ».

Un joueur peut donc augmenter son propre espace en capturant les pions de l'adversaire soit individuellement, soit par groupes.

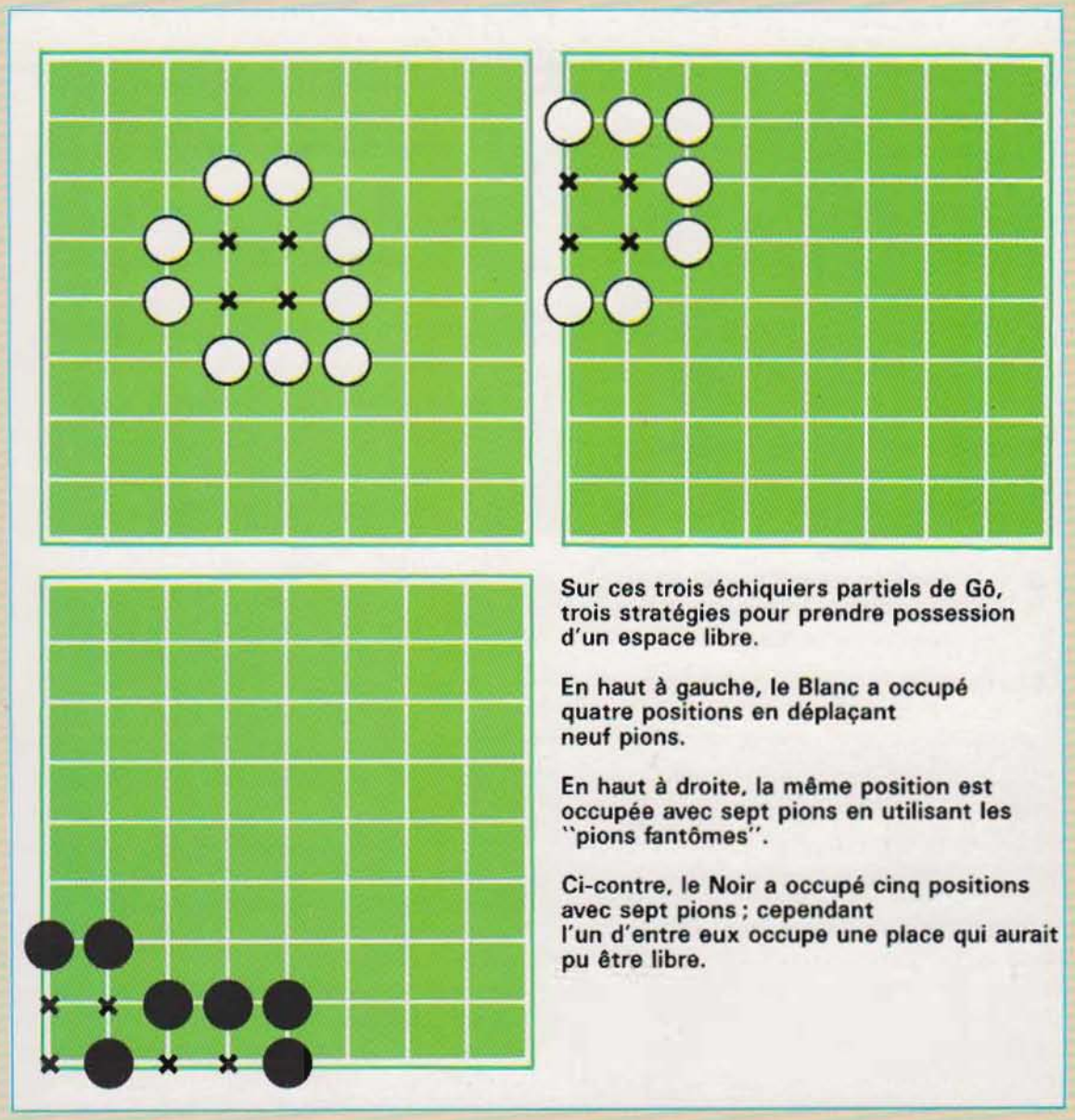
*Ici encore, le Gô se distingue du jeu de Dames par son raffinement : la capture d'un ou de plusieurs pions ne se produit pas par élimination ou par superposition physique. Au contraire : un pion finit dans le butin de l'adversaire lorsqu'il a été **cerné** par les pions de la couleur opposée et que, donc, il n'est **plus libre** (plus de croisement libre dans les lieux adjacents au pion).*

*Pour posséder un espace, il faut le cerner avec une chaîne de pions, tous de la même couleur. Il va de soi que cette configuration présente une importance déterminante : elle est définie comme **groupe de connexion.***

*Une grande partie de la dynamique du jeu et des affrontements entre deux compétiteurs sert justement à empêcher la connexion des pions de l'adversaire en groupes importants, illustrant ainsi l'adage bien connu, selon lequel il convient de **diviser pour régner.***

Toutefois, il est parfois utile de penser « petit », comme, par exemple, de capturer de petits espaces non encore mitoyens entre eux car le but final du Gô est de posséder des territoires et non de capturer des pions. Il est donc inutile de bouleverser le jeu de l'adversaire si l'on manque de pions après ! La capture d'un groupe se produit par la saturation de tous les espaces et par l'obligation qu'on impose à l'adversaire de rester dans des limites les plus petites possibles.

*Le jeu se termine lorsque les deux compétiteurs conviennent qu'il n'existe **plus de motifs** (ni conquête de territoire, ni capture de prisonniers) pour poursuivre les mouvements. Au contraire donc du jeu d'Echecs qui tue ses adversaires, le jeu de Gô est intéressé par ce qui le **fera vivre.** Toute une philosophie profonde, qui illustre aussi bien l'attitude d'une civilisation en face de la vie et de la mort que les moyens par lesquels elle assurera sa survie face aux autres.*



Sur ces trois échiquiers partiels de Gô, trois stratégies pour prendre possession d'un espace libre.

En haut à gauche, le Blanc a occupé quatre positions en déplaçant neuf pions.

En haut à droite, la même position est occupée avec sept pions en utilisant les "pions fantômes".

Ci-contre, le Noir a occupé cinq positions avec sept pions ; cependant l'un d'entre eux occupe une place qui aurait pu être libre.

Maintenant que nous avons étudié, en détail, un wargame classique, le Gô, et avant d'en examiner la version revue par le dernier cri de l'informatique et de l'électronique, accordons nous un moment de réflexion à propos des ***simulations de guerre****.*

Un excellent roman, tout-à-fait récent, ***"Le système Aristote"*** *de René Dzagoyan comporte quelques passages savoureux sur les rapports entre l'être humain et l'ordinateur dans le cadre militaire. Nous nous en servirons pour étayer notre réflexion.*

Tout d'abord un coup de patte à l'adresse de ceux qui confondent le tas de ferraille avec un nouveau Dieu :

"— Peut-être un de nos ordinateurs militaires s'est-il détraqué...

— C'est rigoureusement impossible

— Mais pourquoi donc ?

— Aristote ne peut pas se tromper. Aristote, c'est le système d'ordinateurs qui contrôle les terminaux des sous-marins. Pas seulement les sous-marins, d'ailleurs. Mais encore le reste de la flotte, nos forces nucléaires terrestres, et nos bombardiers, et tout ce que vous pouvez imaginer. Jusqu'au téléphone ! Ou nos chaînes de télévision. Nos aéroports civils, enfin la plupart des choses que vous voyez fonctionner.

LES REGLES DU GO

1) Noir commence en déplaçant, en alternative avec Blanc, un pion à la fois sur le jeu.
Si Noir a le droit de handicap, il déplace tous les points de handicap au début de la partie.
Les pions doivent être posés sur les intersections libres du Gô Ban.

2) Le joueur peut rester à sa place, en "passant" son tour.

3) Gagne celui des deux joueurs qui possède le plus grand nombre d'intersections libres. La partie égale est "pat".

4) Un groupe de connexion ou un pion peuvent être capturés quand ils sont privés de liberté.

5) Un pion prisonnier correspond à une intersection libre en moins dans le compte des points du joueur.

6) Il est interdit de copier une situation existante sur le Gô Ban.

7) Le placement d'un pion dans un point privé de liberté a pour but de capturer des pions de l'adversaire.

8) La partie se termine avec l'accord des deux joueurs.

9) A la fin de la partie, on procède au décompte des pions "morts".

10) Un SEKI ne donne ni points ni prisonniers.

11) A la fin de la partie, il faut fermet chaque KO et toutes les intersections neutres.

Et cela, à travers treize pays...
Mais je crois que nous avons affaire à un phénomène scientifique tout-à-fait incroyable. Si mes hypothèses sont justes, l'histoire du sous-marin (qu'on a cru, tout d'abord, détruit à cause de la défaillance d'un ordinateur local) est le prélude au plus grand bouleversement que l'humanité aura jamais connu."
Vous voilà en haleine : vous vous demandez ce que "Super-Aristote" a encore bien pu inventer... Oh, mais pas si vite. Tout d'abord, essayons de comprendre Aristote :
"— Tout a commencé dans les années soixante. Un professeur d'économie avait eu l'idée géniale, à l'époque, de construire un modèle mathématique représentant l'économie mondiale. Son but était de ***prévoir avec précision les mouvements économiques*** *de la planète. Pour obtenir ce résultat, il a choisi un moyen très simple. Il a installé, dans les sous-sols de l'université, un immense ordinateur (à l'époque un IBM) et il a fourré tout ce qu'il pouvait trouver comme données économiques. Son modèle mathématique se chargeait de mettre en ordre et de faire les calculs nécessaires.*

— Et ça marchait ?
— Mouais... Petit à petit, le système a commencé à marcher. Les résultats s'amélioraient. Au point que les pays de l'OTAN s'y intéressèrent. Les Français d'abord, puis les Anglais, puis une dizaine de pays, dont le Japon. De tous les coins du monde, nous parvenaient des tonnes et des tonnes d'informations économiques...
En un mot, le système était devenu un ***monstre****. Totalement* ***ingérable****. Il nous fallait trouver autre chose...* ***Décentraliser*** *le système. L'idée était simple. Chacun des pays utilisateurs du système installait, chez lui, un ordinateur semblable au nôtre et entrait directement les informations sur ses terminaux, au lieu de nous les envoyer. Une vingtaine de pays adhérèrent au projet. L'astuce était de* ***relier*** *tous ces ordinateurs par* ***câbles sous-marins****.*
De cette façon, nous formions un immense système informatique réparti aux quatre coins de la planète. Un opérateur, installé n'importe où, pouvait interroger, sur son terminal, un ordinateur situé à Tokyo ou à Londres.

(suite page 1269)

L'utilisation de ces fonctions exige un minimum d'attention lorsque les programmes doivent être employés sur différents ordinateurs. Rappelons que le jeu des caractères disponibles n'est pas toujours le même. Ainsi la fonction :

SUCC ('Z')

pourrait donner des résultats différents.
Les opérateurs relationnels , < =, > =, < = et < > , sont utilisables sur le type caractère. Ces opérateurs comparent évidemment les chiffres correspondants à la codification interne des caractères ; il résultera :

'A' < 'B'

lorsque :

ORD ('A') < ORD ('B')

Type booléen. Il comporte seulement deux constantes « vrai » et « faux ». Une donnée de type booléen peut être le résultat d'une opération relationnelle appliquée à des données de type entier, réel ou caractère. Par exemple, la relation :

A > 5

sera vraie si la variable A contient un nombre supérieur à 5 et fausse si le chiffre est inférieur ou égal à 5.
Les données de type booléen peuvent être combinées entre elles par l'intermédiaire des opérateurs logiques ET, OU et NON. Par exemple, l'expression :

(A = 3) AND (B > 8)

sera vraie si (et seulement si) A est égal à 3 et si, en même temps, B est supérieur à 8. La syntaxe de la donnée de type booléen est indiquée dans le schéma ci-dessous.

Les déclarations de type

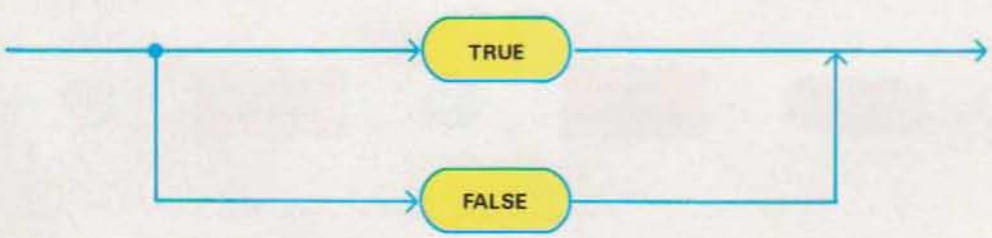
Jusqu'à présent nous n'avons vu que les types de données scolaires, appelées **standard,** et prédéfinies en Pascal.

Avant de passer aux autres types, analysons les données du Pascal définies comme déclarations, à savoir : les définitions des constantes

Attention !

La forme standard d'écriture des instructions en Pascal est fondée sur les caractères alphabétiques.
Nous avons toutefois choisi de présenter la syntaxe du langage en utilisant des caractères majuscules pour des raisons de clarté et d'homogénéité, mais également parce que de nombreuses machines utilisent exclusivement les caractères majuscules.
Dans la plupart des cas, le compilateur Pascal accepte indifféremment les instructions écrites en majuscules ou en minuscules.

SYNTAXE DE TYPE BOOLEEN



et des variables utilisées par le programme.
Le programme Pascal est subdivisé en deux sections, la première déclarative et la seconde, exécutive.
A travers la définition des types de données et des variables, la section déclarative informe aussi bien la seconde section que le programmeur, des structures de données utilisées. Les déclarations de la première section sont donc contraignantes.
L'utilisation, dans la seconde section, du programme d'une variable non préalablement déclarée est signalée comme une erreur.
En d'autres termes : le programmeur est obligé de prévoir, dès le début, toutes les données qu'il a l'intention d'utiliser, ce qui amène à construire les programmes de façon réfléchie.
On distingue la déclaration de constante et la déclaration de variable.

La déclaration de constante est utile lorsque l'on veut assigner un nom mnémonique à une constante fréquemment utilisée dans le programme. (Voir syntaxe ci-dessous). Elle établit simplement une équivalence entre un nom symbolique et une constante scalaire.
La valeur d'une telle constante ne peut être chargée à l'intérieur du programme ; une éventuelle instruction qui tenterait de changer cette valeur provoquerait une erreur signalée par le compilateur.

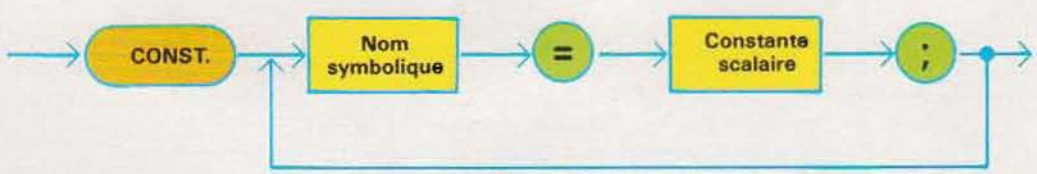
Les déclarations de constantes suivantes sont valables :

```
CONST
        PI = 3.1415927 ;
        INTERET = 0.11 ;
        ESPACE = ' ' ;
```

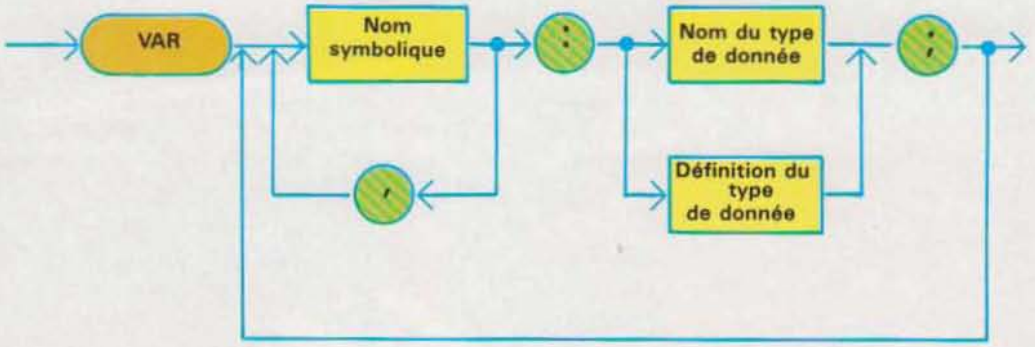
La déclaration de constante non seulement améliore la lisibilité du programme mais aussi facilite les modifications ou les corrections éventuelles.
Considérons, par exemple, un programme pour calculer les intérêts sur un compte-courant. Si, pour le taux d'intérêt, on utilise à l'intérieur du programme une valeur numérique au lieu d'un nom symbolique, une éventuelle variation du taux obligerait à rechercher tous les endroits où le taux apparaît dans le programme pour modifier celui-ci. En utilisant un nom symbolique pour la constante numérique, on obtient le même résultat en modifiant simplement la déclaration de constante.

La déclaration de variable. Une variable est une position de mémoire identifiée par un nom symbolique et contenant une valeur numérique. Dans certains langages autres que Pascal, lorsqu'une variable a été définie une première fois, il n'existe pas de restrictions sur le type de valeur qui peut être mémorisé dans la position correspondante.
En d'autres termes, des valeurs entières, réelles ou booléennes peuvent être indifféremment assignées à une telle variable. Une caractéristique du Pascal (qu'on retrouve dans l'Algol) est la formalisation du concept du type de données qui conduit à des règles restrictives dans la définition et dans l'assignation des variables.
Trois conséquences :
1 / Chaque variable scalaire peut mémoriser des valeurs appartenant à un seul type de donnée.

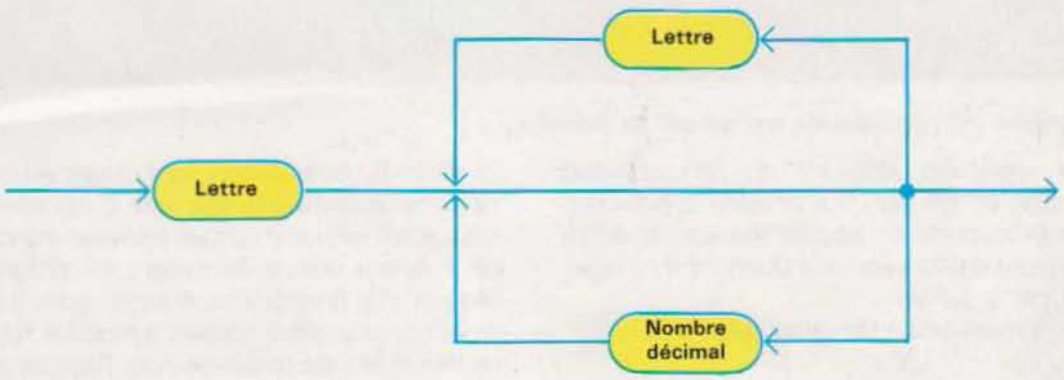
SYNTAXE DE LA DECLARATION DE CONSTANTE



SYNTAXE DE LA DECLARATION DE VARIABLE



CONSTRUCTION DES NOMS SYMBOLIQUES DES VARIABLES



2 / La majeure partie des fonctions prévues en Pascal opère sur des types de données déterminées ; une erreur est signalée à chaque fois qu'on tente d'appliquer ces fonctions à des types de données non prévus.

3 / Toute autre opération sur des données de types différents (sauf un seul cas particulier) est signalée comme une erreur.

Ces règles peuvent paraître comme de lourdes limitations lors de l'écriture d'un programme en Pascal mais elles offrent, en échange, certains avantages. Elles améliorent avant tout la lisibilité du programme car elles obligent le programmeur à déclarer, dès le début, toutes les variables qui seront utilisées et le type de données qu'elles contiendront. De plus, elles rendent plus faciles la recherche et la correction d'éventuelles erreurs. Il est nécessaire, pour créer une variable quelconque en Pascal, d'utiliser la déclaration VAR. La déclaration doit être précédée du mot-clé VAR suivi du nom ou des noms que l'on veut déclarer.

Les règles à suivre pour le choix du nom sont les mêmes que pour le Basic. Le nom ne doit pas coincider avec un mot réservé (mot-instruction ou commande de système) et il commence nécessairement par un caractère alphabétique ; les caractères suivants peuvent être alphabétiques ou numériques.

Il faut écrire, juste après, le ou les noms, le ou les types auxquels appartient la ou les variables. La variable est de type standard (entier,

B. Coleman/Marka

Système informatique de traitement de fichiers.

réel, caractère, booléen) ou non standard, et donc définie par l'usager dans une déclaration précédente de type (la manière de définir les types de données non standard sera exposée par la suite).
Exemples valables de déclaration :

```
VAR
  N, I, J     : INTEGER ;
  AREA        : REAL ;
  VEL, ACC    : REAL ;
  A, B, C     : CHAR ;
```

On constate que toutes les variables et les constantes utilisées dans un programme Pascal doivent obligatoirement être détachées au début, sinon une erreur est signalée lors de la compilation.
Certaines constantes prédéfinies et utilisées par le système font exception à cette règle ; c'est le cas par exemple de MAXINT, qui mémorise le plus grand entier.

Types de données scalaires non standard

En plus des types de données standard déjà décrits, le Pascal permet de recourir à d'autres types particuliers, définis par l'usager en fonction des nécessités du moment. Cette possibilité s'avère très utile lorsque les types standard ne couvrent pas suffisamment les différents besoins d'un programme. Avec un autre langage, on cherche normalement à modifier les caractéristiques du problème pour l'adapter aux types de données disponibles dans le langage utilisé. En Pascal, on agit en sens inverse en adaptant les types des données aux caractéristiques du problème.
Par exemple, pour écrire un programme qui effectue des actions déterminées pour chaque jour de la semaine, on pourrait associer, à chaque jour (du lundi au dimanche), un chiffre entier de 1 à 7. On écrirait, alors, des instructions conditionnelles du type :

```
IF JOUR = 6 THEN...
```

qui ne sont pas très lisibles à première vue puisqu'il faudrait avoir toujours, sous les yeux, un tableau d'équivalences. Les variables associées à ce nouveau type de données assumeront les valeurs LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI, SAMEDI, DIMANCHE.
L'instruction précédente pourra alors être écri-

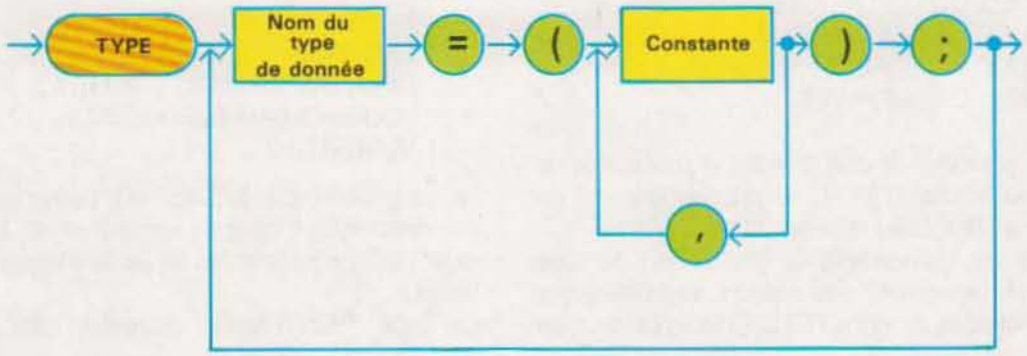
te d'une manière plus lisible :

IF JOUR=SAMEDI THEN...

L'introduction de nouveaux types de données définis par l'utilisateur améliorent la lisibilité du programme d'une façon substantielle et, de plus, comporte d'autres avantages qui seront exposés par la suite. Ces types de données sont définis par l'usager en énumérant individuellement les éléments (comme pour les jours de la semaine) ou comme sous-ensemble.
Dans ce cas, le nouveau type de données sera un sous-ensemble, plus restreint, d'un type standard, ou encore défini par le programmeur selon ses besoins.

Type de données définis par l'utilisateur.
Comme le nom l'indique, ces types sont créés par le programmeur, indépendamment des types standard. C'est une construction qui s'effectue avec la déclaration TYPE (voir ci-dessous) suivie de l'indicateur du type (c'est-à-dire le nom associé au nouveau type auquel on devra se référer pour déclarer une variable).
Cette identification développe les mêmes fonctions que REAL, INTEGER, CHAR, BOOLEAN, vus précédemment. Après le nom du type, on énumère entre parenthèses toutes les constantes qui appartiennent au nouveau type de donnée. L'ordre d'énumération est important car il définit une relation de classement. Celle-ci établit que chaque constante citée est « inférieure » par rapport à celles de droite et « supérieure » à celles de gauche (voir ci-dessous).

DECLARATION POUR DES TYPES DE DONNEES DEFINIS, NON STANDARD



EXEMPLES DE TYPES DE DONNEES NON STANDARD

```
TYPE
    JOUR-SEMAINE=(LUNDI, MARDI, MERCREDI, JEUDI, VENDREDI,
                  SAMEDI, DIMANCHE);

    MOIS=(JANVIER, FEVRIER, MARS, AVRIL, MAI, JUIN, JUILLET, AOUT,
          SEPTEMBRE, OCTOBRE, NOVEMBRE, DECEMBRE);

    COULEURS=(VIOLET, BLEU, VERT, JAUNE, ORANGE, ROUGE);
```

Le type de donnée COULEURS est composé des 6 constantes VIOLET, BLEU..., ROUGE classées de façon à ce que les relations :

VIOLET < BLEU
ORANGE > VERT
JAUNE > VERT

soient vraies.

Appliquer les relations comparatives à ce type de données pourrait sembler dénué de sens. Mais considérons l'exemple du type de donnée MOIS : l'ordre de classement établit de façon univoque l'organisation des constantes ; la circonstance dans laquelle :

JUILLET < AOUT

permet d'utiliser directement le nom de mois pour effectuer des contrôles de validité sur les dates émises.
De plus, ce classement établi, on peut appliquer les fonctions PRED et SUCC même à ce type de données.
Nous aurons alors :

```
PRED (JUILLET)=JUIN
SUCC (JUILLET)=AOUT
SUCC (VENDREDI)=SAMEDI
PRED (JAUNE)=VERT
```

Bien entendu, on doit prendre la précaution de ne pas appliquer PRED au premier élément du type et SUCC au dernier.
Outre ce classement, la déclaration de type définit l'ensemble des valeurs assumées par une variable de nom TEINTE déclarée de type COULEURS :

TEINTE :=TURQUOISE

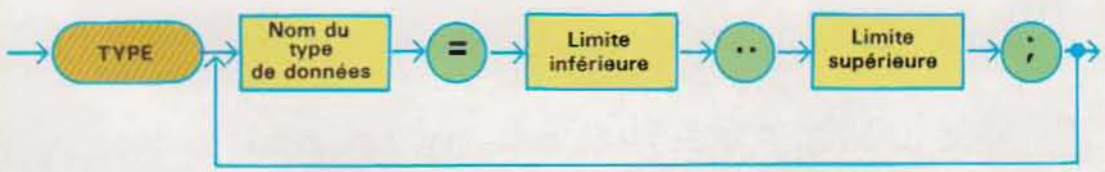
une erreur sera signalée. En effet, la teinte turquoise n'est pas prévue parmi les constantes formant le type COULEURS.
De petites erreurs d'assignation sont ainsi relevées sans passer par de lourdes procédures de contrôle pour accepter l'appartenance d'une donnée à un type. En définitive,l'utilisation des types définis par l'utilisateur diminuent les risques d'erreur.

Types de données sous-ensembles. Ce type est créé en utilisant une partie des données du type standard (à l'exception de REAL) ou un nouveau type déjà défini par l'utilisateur. Le sous-ensemble est constitué en décrivant, à l'intérieur du type de départ, des limites (inférieure et supérieure) et en déclarant que les données comprises entre elles (extrêmes inclus) appartiennent au sous-ensemble (voir ci-dessous).
Voici quelques exemples sortant sur le type sous-ensemble :

```
TYPE
    NOTES=0.. 10 ;
    CHIFFRES=0... 9 ;
    PRINTEMPS=MARS.. JUIN ;
    JOURS-OUVRABLES=LUNDI..
    VENDREDI ;
```

Le sous-ensemble NOTES est composé des nombres entiers compris entre 0 et 10. Les 3e et 4e cas concernent les types définis par l'utilisateur.
Le type PRINTEMPS, composé des mois

DECLARATION POUR LES TYPES DE DONNEES SOUS-ENSEMBLE



(mars à juin) est donc un sous-ensemble du type MOIS précédemment défini.
Comme pour les types définis par l'utilisateur, l'emploi de types sous-ensemble augmente la lisibilité du programme et signale automatiquement les erreurs d'assignation. Par exemple, lorsqu'on déclare une variable de nom MOYENNE de type entier, on admet implicitement qu'elle pourra prendre toutes les valeurs possibles des nombres entiers.
Mais si la même variable est déclarée de types NOTES, on comprend, à la simple lecture du programme, que MOYENNE prendra, à chaque point du programme, les seules valeurs entières comprises entre 0 et 10 (avec la signification de moyenne des notes). Chaque fois qu'on essaiera d'assigner à la variable MOYENNE un nombre en dehors de cet intervalle, une erreur sera signalée.

Les premières instructions du Pascal

La connaissance des concepts de type de donnée, constante, variable et déclaration est essentielle dans l'écriture de programmes Pascal corrects. Mais il faut, auparavant, entrer certaines informations concernant l'évaluation des expressions arithmétiques et booléennes, l'utilisation des fonctions et l'assignation des valeurs aux variables.
Avec les instructions fondamentales d'entrée et de sortie des données, on disposera de moyens très simples pour écrire les premiers programmes en Pascal.

Expressions arithmétiques et booléennes

En programmation Pascal, l'utilisation correcte des types des données et des expressions arithmétiques est essentielle. Ainsi, il n'est pas permis d'insérer différents types de données sauf dans les expressions arithmétiques qui demandent deux nombres réels pour fournir un résultat réel. Dans ce dernier cas, on peut substituer un entier à l'un des deux réels.
Exemple : étant donné deux variables réelles A et C et une variable entière B, l'instruction :

C :=A+B

fournit, en C, un résultat correct. Le compilateur convertit l'entier B en un nombre réel avant de calculer l'expression et d'assigner le résultat à C.
C'est l'unique exception admettant l'utilisation de types différents. Dans notre exemple, le symbole + pouvait être remplacé par tout autre opération admise sur nombres réels.
Voyons le cas des opérations non admises :

'1' - 1	La valeur '1' ne représente pas un chiffre mais un caractère
TRUE + 3	il n'est pas possible d'additionner un entier à une donnée booléenne
'A' + 1	le résultat de l'expression n'est pas le caractère B

Le recours à des types différents provoque une erreur dans les opérateurs DIV et MOD sur des réels. La division est effectuée entre valeurs entières et nécessite donc des opérandes entières. DIV fournit la partie entière de la division (11 DIV 4 est égal à 2) alors que MOD fournit le reste (11 MOD 4 est égal à 3). L'opération 11 DIV 4.0 est illégale car le diviseur est considéré par le compilateur Pascal comme un nombre réel.
N'oublions pas qu'un entier et un réel sont représentés de manière différente en mémoire, même s'ils ont la même valeur. Les nombres 4 et 4.0 sont égaux, mais ont une représentation binaire distincte.
En Pascal comme pour les autres langages, il existe des règles qui régissent le calcul des expressions. Le Pascal définit des hiérarchies de priorité pour les opérateurs ; les opérations de la hiérarchie supérieure se déroulent avant celles de la hiérarchie inférieure. Ceci est valable, en particulier, pour définir un autre ordre de priorité.
En ordre décroissant de priorité, la hiérarchie de priorité est la suivante :

1 / expression entre parenthèses
2 / opérateurs *, /, DIV, MOD
3 / opérateurs d'addition (+) et de soustraction (-)

Le calcul d'une expression en Pascal s'effectue donc dans cet ordre : d'abord les expressions entre parenthèses ; ensuite les multiplications et les divisions et, en dernier lieu, l'addition et la soustraction.

Par exemple, l'expression 7+A*4 est calculée ainsi :

7+(A*4)

Quand deux opérateurs ont le même ordre de priorité (qui n'est pas modifié par la présence de parenthèses), les expressions sont calculées comme elles sont écrites, dans le sens gauche-droite. Par exemple, l'expression :

A*12/B

est calculée comme :

(A*12)/B

Une formule du type :

A+B+(C - D)*E*F/G

est calculée :

(A+B)+((((C - D)*E) *F)/G)

Ces règles d'homogénéité des données et d'ordre de priorité valent également pour les expressions booléennes. Les opérateurs relationnels (< , < =, =, > = , >) sont employés pour comparer des données de type entier, réel, caractère, booléen ou tout autre type défini par l'usager.
Ces types ne peuvent cependant pas être mélangés car on ne peut, par exemple, comparer une valeur réelle à une valeur booléenne :

2 > = FALSE

est erronée car elle compare deux données de type différent.
Les opérateurs logiques ET OU NON (AND, OR, NOT) n'acceptent que le type booléen pour donner un résultat de type booléen.
Il existe également une hiérarchie de priorité (en ordre décroissant) pour l'évaluation des expressions booléennes :

1 / expression entre parenthèses
2 / opérateur logique NOT
3 / opérateurs /, DIV, MOD, AND
4 / opérateurs +, -, OR
5 / opérateurs relationnels (<,< =, =,> =,>)

Cette hiérarchie est plus compliquée que celle des expressions arithmétiques.
En effet, les expressions booléennes peuvent contenir un plus grand nombre de symboles et de types de données et également d'autres expressions de type arithmétique.
Par exemple :

A < B-1

est une expression booléenne valide puisque, selon les règles énoncées, elle est évaluée comme si elle avait été écrite de la façon suivante :

A < (B-1)

Observons que le résultat de cette expression aurait été calculée de manière non ambiguë même en l'absence de priorités.
En effet, elle ne peut, en aucune manière, être calculée de la façon suivante :

(A < B)-1

car il faudrait alors soustraire la valeur entière 1 à la valeur booléenne (vraie ou faux) résultat de la comparaison de A et B.
On peut donc dire que, dans les expressions booléennes, les opérateurs relationnels doivent toujours être évalués après le calcul des expressions arithmétiques.

Emploi des fonctions Pascal

L'ensemble des fonctions standard du Pascal a déjà été présenté lors de la description des types de données. Le recours à ces fonctions nécessite simplement l'écriture du nom de la fonction appelée, suivi du ou des arguments, (la valeur à laquelle on désire appliquer la fonction entre parenthèses).
Le résultat, c'est-à-dire la valeur de retour, peut être assigné à une variable ou réemployé dans une expression arithmétique.
Nous pouvons, par exemple, écrire :

(SQR(2)*9)+6.5

où la valeur 4 (résultat de l'élévation à la puissance de 2) est multipliée par 9 et le résultat additionné à 6.5.

G. Hall/Marka

Ecrans radar dans une salle de contrôle du trafic aérien.

Le résultat d'une fonction est donc utilisable comme celui d'une constante ou d'une variable. Son argument est lui-même une constante, une variable, une expression arithmétique ou même le résultat d'une autre fonction.

Nous pouvons alors écrire indifféremment :

```
SQR(2)
SQR(A)
SQR(7*A+12.5)
SQR(SIN(A))
```

Ces expressions sont toutes valables, à la seule condition que A ait été défini comme un type compatible avec les fonctions SQR et SIN.
Le calcul des expressions arithmétiques s'effectue de la même manière que précédemment.

Bien noter : dans l'utilisation des fonctions Pascal il est important de bien s'assurer de l'homogénéité des types de données impliquées dans un calcul. La fonction opère sur un argument d'un certain type tandis que le résultat peut appartenir à un autre type. Ainsi, la fonction TRUNC (qui élimine les chiffres après la virgule d'un réel) a pour argument une donnée de type réel et pour résultat un type entier.
De même, la fonction CHR traite un nombre entier (l'argument) et fournit un caractère (résultat).
Si certaines fonctions admettent des arguments de types différents, le résultat appartient toujours à un seul type.
Page suivante, on trouvera une liste et la description des fonctions standard disponibles en Pascal.
Cet ensemble est dit « standard » puisqu'il est commun à tous les compilateurs Pascal.

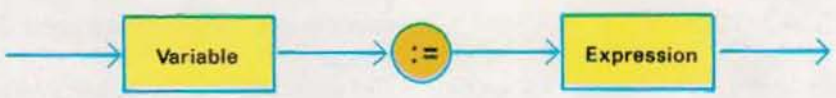
L'instruction d'assignation

Elle attribue une valeur à une variable (voir syntaxe page suivante).
L'opérateur d'assignation est représenté par : =, différent du symbole d'égalité des autres langages.
On a ainsi voulu éviter la confusion possible dans l'évaluation d'une instruction.

FORMATS STANDARD DU PASCAL

Nom de la fonction	Description	Type de donnée argument	Type de donnée résultat
ABS	Valeur absolue	Entier ou réel	Entier
ARCTAN	Arctangente	Réel ou entier	Réel
CHR	Caractère correspondant au nombre	Entier	Caractère
COS	Cosinus	Réel ou entier	Réel
EXP	Exponentiel $e^{argument}$	Réel ou entier	Réel
LN	Logarithme naturel (en base e)	Réel ou entier	Réel
ODD	Vérifie si argument impair	Entier	Booléen
ORD	Nombre correspondant au caractère	Caractère	Entier
PRED	Donnée précédant l'argument	Tout type sauf réel	Même type que l'argument
ROUND	Arrondi	Réel	Entier
SIN	Sinus	Réel ou entier	Réel
SQR	Carré de l'argument	Entier ou réel	Même type que l'argument
SQRT	Racine carrée de l'argument	Réel ou entier	Réel
SUCC	Donnée suivant l'argument	Tout type sauf réel	Même type que l'argument
TRUNC	Troncature	Réel	Entier

SYNTAXE DE L'INSTRUCTION D'ASSIGNATION



Par exemple :

A :=A+1.

permet d'ajouter 1 à la variable A, c'est-à-dire d'incrémenter A, alors que :

A=A+1.

doit être considéré comme une valeur booléenne, qui sera d'ailleurs toujours fausse.

L'instruction d'assignation est validée si toutes les variables ont été précédemment définies et quand celles situées à droite de l'opérateur ont déjà été assignées.

Test 21

1 / Parmi les déclaration des constantes suivantes, lesquelles sont correctes ?

Trouver les erreurs :

a)
```
CONST
    VALEURMAX = 1000;
```
b)
```
CONST
    PREMIER : 1 ;
```
c)
```
CONST
    MAXIMUM = 500 ;
    MINIMUM = - 50 ;
```
d)
```
CONST
    ALPHA = 0 OR 1 OR 2 ;
```
e)
```
CONST
    VALEURS = 0 .. 30 ;
```

---

2 / Ecrire une déclaration (unique) pour les constantes suivantes :

a) le symbole de l'addition (l'appeler PLUS) ;
b) la constante entière 24 qui indique le nombre de lignes lisibles sur un moniteur (l'appeler MAXLIGNES) ;
c et d) les nombres réels 0 et 100 qui représentent les températures de congélation et d'ébullition de l'eau (les appeler respectivement CONG et EBU).

---

3 / Quelles déclarations sont valables ? Trouver les erreurs :

a)
```
VAR
    1CAR, 2CAR, 3CAR : CHAR ;
```
b)
```
VAR
    CAR1, CAR2, CAR3 : INTEGER ;
    IN1              : REAL ;
    IN2              : INTEGER ;
```
c)
```
VAR
    X                : REAL ;
    Y                : REAL ;
    Z                : REAL ;
    XYZ              : CHAR ;
```

---

4 / Ecrire une déclaration de variable pour les données suivantes en choisissant les noms des variables adéquats :

a) les couleurs fondamentales rouge, jaune, bleu ;
b) une variable booléenne contenant la réponse positive ou négative à une question ;
c) un nombre entier qui tienne compte des lignes imprimées sur une page ;
d) une variable qui contient le numéro de la plaque d'immatriculation d'une voiture.

5 / Quelles sont les déclarations de type qui sont fausses ?
Trouver les erreurs :

a) TYPE INTEGER = - MAXINT.. + MAXINT ;
b) TYPE GRADES=0.0.. 100.0 ;
c) TYPE COULEURS=(JAUNE, ROUGE, VERT, ECARLATE, MARRON, VIOLET) ;
d) TYPE PRINTEMPS=MARS .. JUIN.

6 / Ecrire une déclaration (unique) pour définir les types de données suivants :

a) les noms des couleurs des cartes à jouer (COULEURS) ;
b) les lettres majuscules de l'alphabet (MAJUSCULES) ;
c) les sous-ensembles des nombres entiers de - 10 à + 50 (INTERVALLE).

7 / Quelles sont, parmi les valeurs suivantes, les données scalaires :

a) le premier élément d'un vecteur ;
b) le nom "JEAN ROSTAND" ;
c) le nombre de départements en France ;
d) un vecteur de 20 éléments.

8 / D'après le type suivant :

TYPE MOIS = (DECEMBRE, JANVIER, FEVRIER, MARS, JUIN, JUILLET, AOUT)

quels sont les résultats des fonctions suivantes :

a) PRED (JANVIER) ;
b) PRED (JUIN) ;
c) SUCC (MARS) ;
d) SUCC (AOUT) ;
e) PRED (DECEMBRE)

Solutions p. 1248.

Par exemple pour écrire :

```
A :=B+5.5
```

il faut s'assurer que la variable B contient une valeur déjà spécifiée, par exemple par une précédente instruction d'assignation. De plus, le type de donnée (résultat du calcul de l'expression) doit correspondre au type de donnée déclaré pour la variable A.
Exception à cette règle : l'assignation par une valeur entière d'une valeur réelle.
L'entier est alors transformé en réel avec une assignation équivalente. Dans tous les cas, les types doivent coïncider.
Examinons les déclarations de variable suivantes :

```
VAR
    INDICE      : INTEGER ;
    CONSONNE    : CHAR ;
    A,B,C       : REAL ;
    INDICATEUR  : BOOLEEN ;
```

les instructions d'assignation suivantes sont valables :

```
A :=88.5
B :=20.
CONSONNE :='K'
INDICE :=0
INDICATEUR :=A < 50
```

```
INDICE :=INDICE+1
C :=B*6.5
INDICATEUR :=(A < 50)OR(ODD(INDICE))
```

Les deux premières instructions assignent deux valeurs réelles aux variables A et B.
De même, pour les deux autres qui assignent des valeurs cohérentes au type de donnée déclaré pour les variables.
L'instruction :

```
INDICATEUR :=A<50
```

assigne la valeur FALSE (faux) à la variable booléenne FLAG puisque A est supérieur à 50.
Pour les deux autres, la variable INDICE vaut 1 et C est égal à 130.
Dans la dernière instruction, enfin, la valeur de la variable booléenne INDICATEUR est TRUE (vrai) puisque, même si A est supérieur à 50, la variable entière INDICE est égale à 1 (donc impaire) et les deux valeurs booléennes sont combinées au moyen de la fonction OR.

Les instructions d'entrée : READ et READLN

Les instructions qui activent la fonction d'entrée sont, en Pascal, READ et READLN (voir ci-dessous). L'instruction READ assigne aux variables (énumérées entre parenthèses) de la « read list » (« lire liste ») les valeurs lues par le dispositif d'entrée du clavier de saisie, du lecteur de cartes ou de la bande perforée.
Les variables de la « read list » peuvent être de type entier, réel, booléen, caractère et sous-ensemble. L'introduction d'une variable d'un autre type provoque une erreur. En nous référant à la déclaration précédente, voici quelques exemples d'instructions d'entrée :

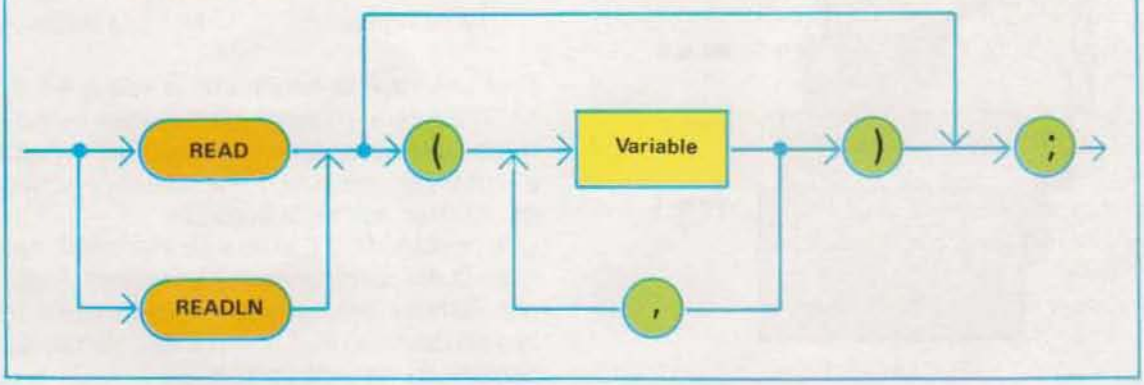
```
READ (A,B,C) ;
READ (INDICE, CONSONNE, INDICATEUR) ;
READ ;
```

Dans l'instruction READ, le nombre de données nécessaires à son exécution est égal au nombre de variables contenues dans la « read list » ; les données doivent, en outre, correspondre par type et par position, aux variables correspondantes. Mais on peut (seule exception admise) fournir une valeur entière que l'on assigne à une variable réelle.
Par exemple, si on a déclaré les variables A et B de type réel et la variable INDICE de type entier, on répondra enfin correctement à l'instruction :

```
READ (A, INDICE,B) ;
```

en entrant les 3 nombres 1.25, 10, 0.
Deux éventualités peuvent se présenter dans le cas où il n'y aurait pas une pleine correspondance entre les données et les variables.
Dans la première, le programme tente d'effectuer une assignation impossible, par exemple un caractère à une variable réelle, ce qui provoque une erreur et bloque son déroulement (fatal error = erreur fatale).
La seconde se produit quand le programme assigne une valeur erronée à la variable et poursuit l'exécution. Dans ce cas, on peut contourner la difficulté en insérant les instructions adéquates de validation de l'entrée de manière à assurer l'assignation souhaitée.

SYNTAXE DES INSTRUCTIONS D'ENTREE READ ET READLN



Pour compléter la description des instructions d'entrée, analysons l'instruction READLN. C'est un mot réservé qui peut remplacer READ en conservant la même syntaxe (voir page précédente). L'instruction READLN est utilisée pour effectuer, depuis le clavier ou tout autre dispositif, l'entrée sélective. Sur le lecteur de cartes ou sur l'unité bande, les données sont normalement organisées en blocs. READLN lit les valeurs des variables en paramètres, puis saute à la ligne en ignorant toutes les autres valeurs sur la liste d'entrée.
Supposons que les données soient regroupées en blocs de quatre, à raison d'un bloc par carte (voir ci-dessous).
L'exécution des deux instructions d'entrée :

```
READLN(A,B)
READLN (C,D)
```

fait en sorte que les valeurs :

```
A :=12
B :=20
```

c'est-à-dire les deux premières valeurs de la carte 1 soient assignées aux variables A et B. Pendant ce temps-là, les variables C et D prendront les valeurs :

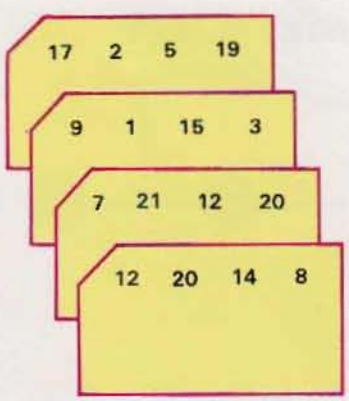
```
C :=7
D :=21
```

c'est-à-dire les deux premières valeurs de la seconde carte.
Si, au lieu de READLN, nous avions utilisé l'instruction READ, on aurait eu les assignations suivantes pour les quatre variables :

```
A :=12
B := 20
C :=14
D :=8
```

c'est-à-dire que toutes les valeurs de la première carte auraient été lues.

CARTES PERFOREES CONTENANT LES DONNEES POUR READLN



Les instructions de sortie : WRITE et WRITELN

Par analogie, les instructions de sortie admettent les deux formes WRITE et WRITELN.
Leur diagramme syntaxique (ci-contre) est complexe ; il reflète l'extrême flexibilité et la puissance des instructions de sortie propres au Pascal. Les éléments à imprimer sont énumérés entre parenthèses.
L'ensemble de ces éléments est défini « write list » (écrire liste). Chacun d'eux est, soit une expression dont le résultat est calculé au moment de l'impression, soit une variable simple, soit encore une chaîne de caractères.
Prenons l'hypothèse d'une valeur 4 assignée à la variable A déclarée de type entier.
L'instruction de sortie :

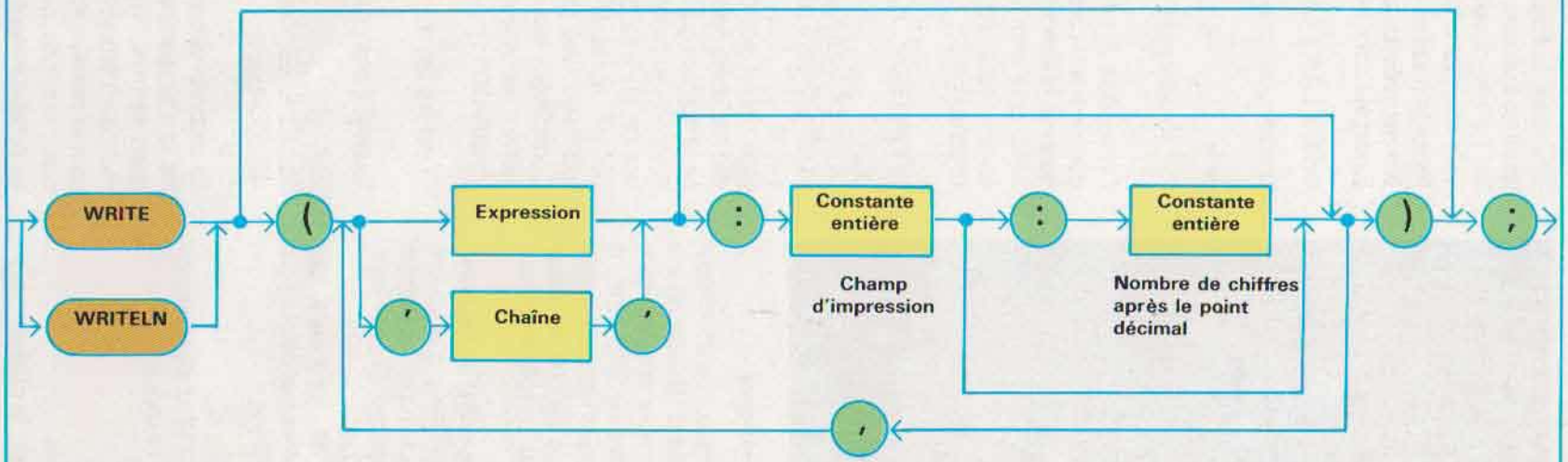
```
WRITE ('A=', A, ' Le carré de A est ', SQR (A))
```

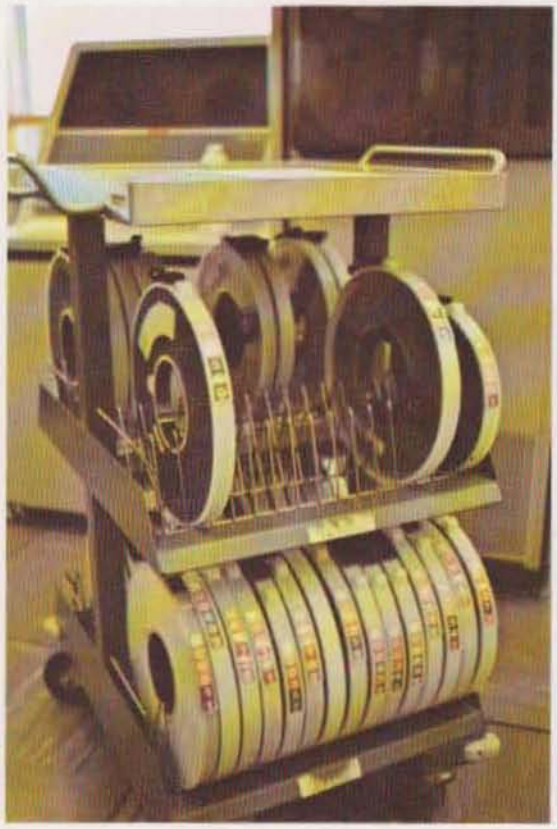
produira une ligne d'impression (sur écran ou imprimante) du type :

```
A =         4 Le carré de A est          16
  |10 colonnes|                  |10 colonnes|
```

Pour une variable booléenne, la valeur FALSE ou TRUE sera imprimée alors qu'une variable réelle sera toujours en notation scientifique (nombre décimal suivi d'une puissance entière de 10) sauf indication contraire.
L'instruction WRITE assure simplement le formatage des nombres et des caractères. A chaque élément est assigné, en impression, un champ composé d'un nombre de colonnes dépendant du type de variable.

SYNTAXE DES INSTRUCTIONS DE SORTIE WRITE ET WRITELN





Marka/Infophoto

Chariot porte-bandes magnétiques.

On suppose, dans l'exemple, qu'un champ d'impression de 10 colonnes a été assigné à la variable entière A. Si le nombre contenu dans la variable A comprend moins de 10 chiffres, le système effectue automatiquement une justification à droite en insérant, à gauche, les espaces vides nécessaires pour remplir le champ d'impression prévu.
Dans ce cas, A étant composé d'un seul chiffre, l'insertion de 9 espaces blancs à la gauche de la valeur pour couvrir le champ d'impression de 10 colonnes est automatique. D'où la justification à droite de la valeur.
Voici, à titre indicatif, les longueurs des champs d'impression adoptées par défaut pour les différents types de données :

Entiers = 12 colonnes
Réels = 16 colonnes (12 chiffres significatifs avec l'exposant dans la forme E±xx)

Caractère = 1 colonne
Booléen = 10 colonnes
Chaîne = Longueur de la chaîne

Toutefois, le Pascal permet de modifier facilement ces valeurs par défaut grâce à la « write list ». Celle-ci permet, pour la variable dont on veut modifier le champ d'impression, de déterminer le nombre de colonnes désiré. Ce nombre est précédé de deux points.
Exemple :

WRITE ('A=', 'Le carré de A est', SQR (A) : 5)

produit une ligne d'impression du type :

A = 4 Le carré de A est 16
|5 colonnes| |5 colonnes|

On recourt à cette facilité pour disposer les données en colonnes de la même manière qu'avec la fonction TAB(X) du Basic.
Ainsi, pour imprimer un nombre dont le dernier chiffre sera en colonne 50, on exécute l'instruction :

WRITE (A : 50)

La valeur contenue en A sera justifiée à droite et complétée à gauche par les espaces nécessaires.
Une option est disponible pour le traitement des variables de type réel. Elle permet de définir le nombre de chiffres décimaux à mettre en évidence au moment de l'impression. Ce nombre de chiffres décimaux est immédiatement déclaré après le champ d'impression et le symbole : (voir page précédente).
Considérons les variables B et C avec respectivement les valeurs 10.5728 et 152.278.
L'instruction :

WRITE (B:10:3,C:15)

fournira l'impression suivante :

10.572		1.522780000E+02
10 colonnes		15 colonnes

La seconde option nécessite explicitement, outre la définition du nombre de chiffres après le point décimal, l'impression de B en forme décimale avec une virgule fixe. Les variables réelles sont imprimées en notation scientifique si l'option n'est pas demandée. La variable C est automatiquement imprimée dans la forme

exponentielle (en virgule flottante). Notons enfin que le dernier chiffre décimal est tronqué et non arrondi.

Pour terminer la description des instructions de sortie, nous analyserons WRITELN dont la syntaxe (voir page 1241) rappelle WRITE.
Différence entre les deux instructions : à la fin d'une ligne d'impression WRITELN insère le retour chariot qui « initialise » l'opération de sortie suivante à la ligne suivante.
Ainsi, avec deux variables A et B de valeur respective 1 et 2, les deux instructions :

```
WRITE ('A =', A:4)
WRITE ('B =', B:4)
```

produiront la ligne d'impression suivante :

```
A =  1 B =  2
```

En revanche, les instructions :

```
WRITELN ('A =', A:4)
WRITELN ('B =', B:4)
```

produiront alors ces deux lignes d'impression :

```
A =  1
B =  2
```

WRITELN, qui provoque l'avancement d'une ligne d'impression, est aussi utilisable sans paramètres pour obtenir des lignes d'espacement.

Structure d'un programme Pascal

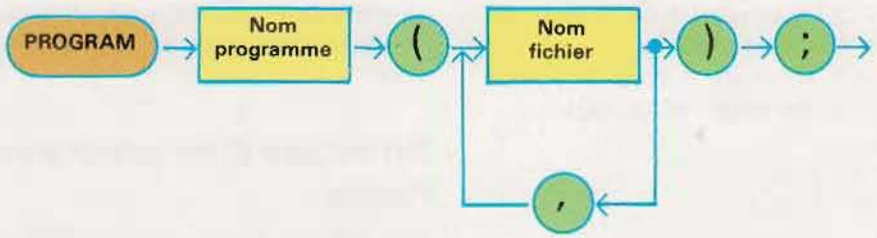
Tout listing en Pascal présente la structure et la syntaxe décrites ci-dessous.
La structure d'un programme se compose de deux sections : l'intitulé (titre) et le bloc. Le titre, consistant en une seule ligne, contient PROGRAM (mot réservé) suivi du titre

FORMAT GENERAL D'UN PROGRAMME PASCAL

```
PROGRAM nom (nom de fichier 1,... 2,....)
LABEL       déclaration... (commentaire)
CONST       déclaration... (commentaire)
TYPE        déclaration... (commentaire)
VAR         déclaration... (commentaire)
PROCEDURE déclaration... (commentaire)
FUNCTION    déclaration... (commentaire)
BEGIN
   instruction (commentaire)
   .....
   .....
   instruction ;
END.
```

DIAGRAMME SYNTAXIQUE REPRESENTANT LA STRUCTURE D'UN PROGRAMME PASCAL

DIAGRAMME SYNTAXIQUE DU TITRE D'UN PROGRAMME



proprement dit et, le cas échéant, de la liste des noms de fichiers externes (voir diagramme ci-dessus). Le nom du programme n'a aucune signification particulière pour le système d'exploitation. En revanche, les fichiers listés sont utilisés par le programme pour échanger des informations avec le monde extérieur.
Après le titre suit le corps **("bloc")** du programme, lui-même composé des deux sections contenant les déclarations et les instructions exécutables (voir diagramme syntaxique ci-contre).
Les déclarations — au nombre de 6 — décrivent toutes les données à utiliser par le programme. Trois d'entre elles — constante (CONST), type de donnée (TYPE) et variable (VAR) — ont déjà été analysées. Restent :

LABEL
PROCEDURE
FONCTION

L'ordre d'écriture des déclarations est celui du schéma. Ecrire TYPE avant CONST, par exemple, provoque normalement un message d'erreur.
La seconde section du bloc constitue le corps exécutable du programme et s'appelle, en anglais, **compound statement** ("instruction composée") ; elle contient en effet une série d'instructions Pascal (diagramme syntaxique p. 1246). Comme on peut le voir, toutes les instructions sont enfermées entre les mots réservés BEGIN (début) et END (fin).
Cette délimitation a pour effet, en isolant le code, d'augmenter sa lisibilité tout en donnant l'ordre séquentiel d'exécution des instructions, séparées par le symbole (;) ou par un mot réservé.
Il ne s'agit donc pas d'instructions proprement dites : aucune action spécifique du système ne leur correspond. Elles servent uniquement à délimiter l'ensemble des instructions à effectuer séquentiellement dans l'ordre dans lequel elles sont écrites, chaque instruction est séparée de la suivante par le point virgule (;). Le compilateur reconnaît donc la fin d'une instruction lorsqu'il rencontre le symbole ; ou un mot réservé.
Pour éviter des erreurs de compilation, il est conseillé de toujours insérer le caractère ; à la fin de l'instruction, même quand il est superflu : dans ce cas, le compilateur interprète la double borne comme une « instruction vide » (en anglais : empty statement) et il en restera là.

DIAGRAMME SYNTAXIQUE DU BLOC

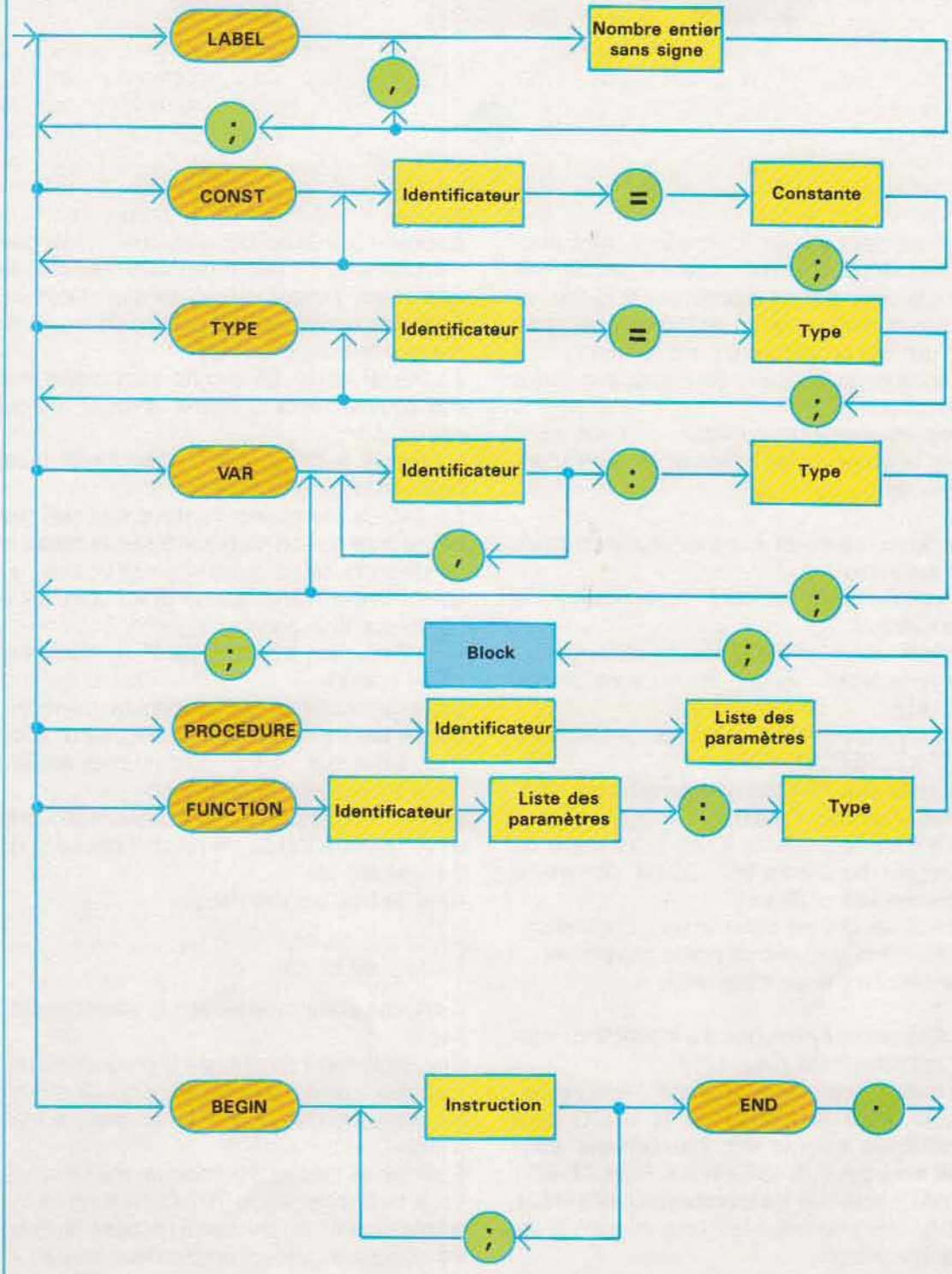
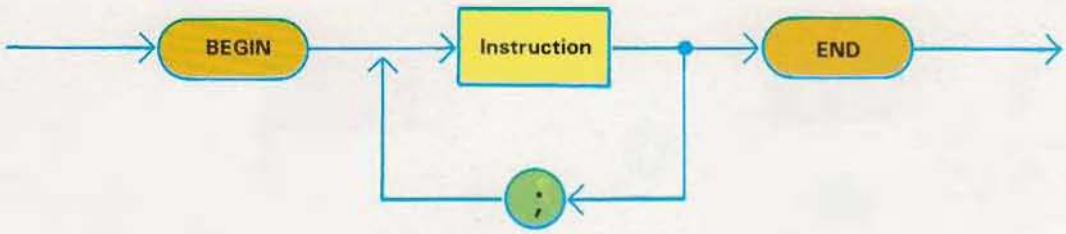


DIAGRAMME SYNTAXIQUE D'UNE INSTRUCTION COMPOSEE (COMPOUND STATEMENT)



Comme partout, les instructions sont exécutées séquentiellement, sauf en cas de saut conditionnel, afin de répéter plusieurs fois un groupe d'instructions ou pour mettre en mouvement une procédure ou une fonction.
Le programme s'achève donc avec la dernière instruction.
Avec les instructions Pascal, on peut alors, dans le programme, effectuer les opérations suivantes :

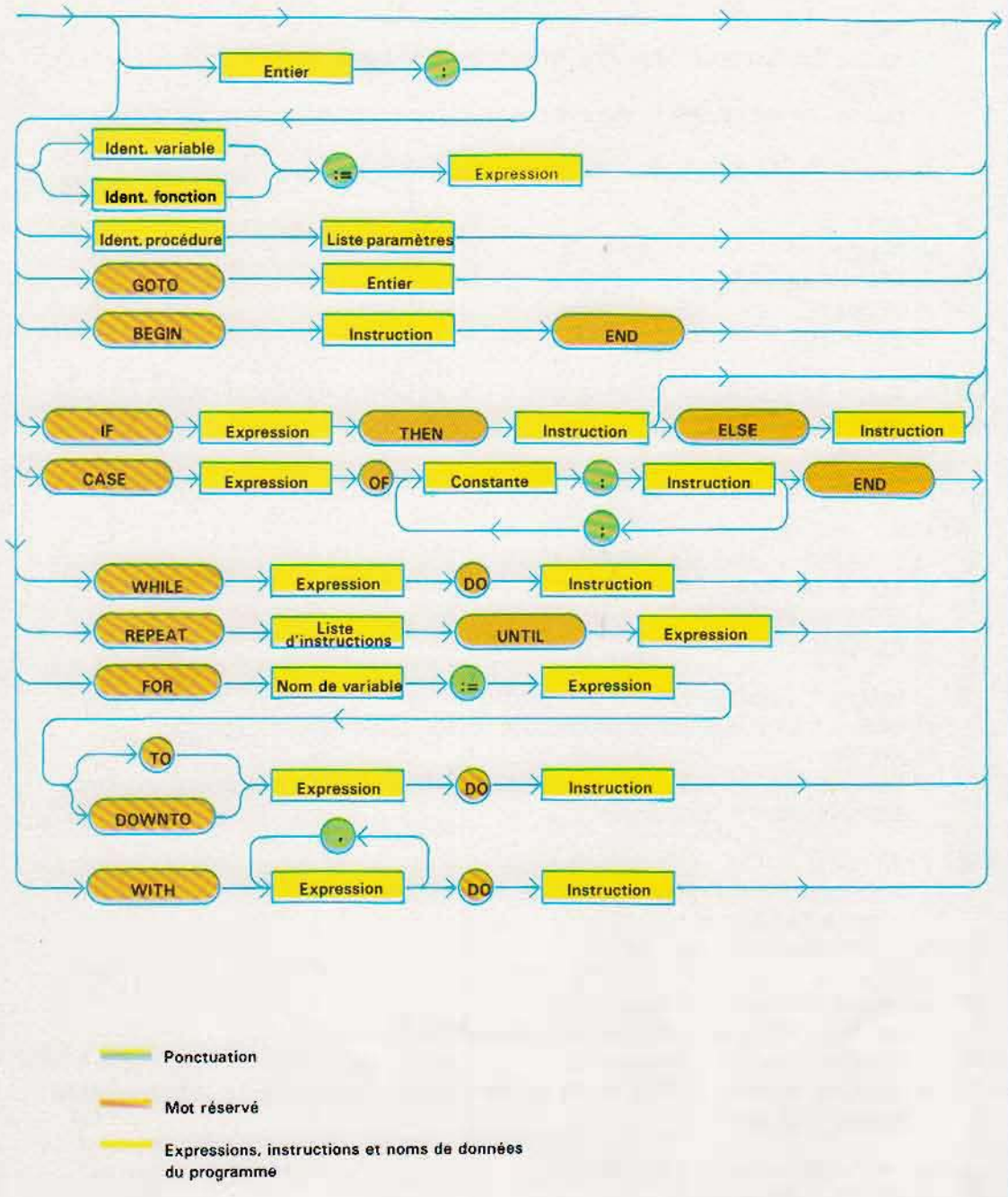
- assigner une valeur à une variable (instruction d'assignation),
- rappeler une procédure (identificateur de procédure),
- choisir un ensemble d'actions en se basant sur certaines valeurs (instructions IF et CASE),
- répéter un groupe d'actions (instructions WHILE, REPEAT, FOR),
- appeler des enregistrements sans les nommer (instructions WITH),
- transférer le contrôle à une autre partie du programme (instruction GOTO, fortement déconseillée en Pascal),
- traiter un groupe d'instructions comme une instruction unique (instruction composée),
- ne rien faire (instruction vide).

Le diagramme syntaxique des instructions Pascal est représenté page 1247.
Certaines instructions comme l'assignation, l'appel d'une procédure ou le GOTO sont considérées comme des **instructions simples** alors que IF, CASE, WHILE, REPEAT, FOR et WITH sont des **instructions structurées** puisqu'elles peuvent, à leur tour, contenir d'autres instructions.
Chaque instruction structurée est donc perçue comme sous-ensemble d'un bloc. Dans les compilateurs, il n'existe généralement pas de restrictions portant sur le nombre d'instructions structurées imbriquées dans un programme ni même dans un bloc.
En Pascal aussi, les programmes admettent des commentaires à insérer entre (*...*) ou entre (.....).
Le Pascal autorise une grande liberté dans l'écriture des instructions.
On peut écrire plusieurs instructions sur une même ligne ou une seule sur plusieurs lignes, à condition toutefois qu'un nom de variable ou qu'une valeur numérique ne se trouvent pas à cheval sur deux lignes.
Une instruction s'écrit à partir de n'importe quelle colonne.
Cette particularité est avantageusement exploitée dans « l'indentation » des instructions, c'est-à-dire leur mise en colonne pour améliorer la lisibilité des programmes.
Elément très important et indispensable dans un programme Pascal : le point. Il indique la fin du programme.
Ainsi, le bloc est délimité par :

— le titre en haut
— le point en bas.

C'est une autre caractéristique essentielle du Pascal.
Chaque élément constituant le programme est considéré comme une structure qui en renferme une autre qui, à son tour, en englobe bien d'autres...
C'est ce qui permet de définir un problème selon la technique dite de TOP-DOWN (structure arborescente) : on parvient à l'écriture du code par niveaux successifs en étendant, de part et d'autre, les structures initialement définies.

DIAGRAMME SYNTAXIQUE DES INSTRUCTIONS PASCAL



Solutions du test 21

1 / a) valable
b) fausse. Le symbole : doit être remplacé par le symbole =
c) valable
d) fausse. Il n'est pas admis d'écrire des fonctions booléennes dans une déclaration de type
e) fausse. On ne peut définir une constante avec le type sous-ensemble

---

2 / CONST
a) PLUS='+'
b) MAXLIGNES=24
c) CONG=0.
d) EBU=100.

---

3 / a) fausse. Le premier caractère du nom d'une variable doit obligatoirement être une lettre
b) valable
c) valable

---

4 / VAR
a) COULEURS : (ROUGE, JAUNE, BLEU)
b) REPONSE : BOOLEAN
c) LIGNES : INTEGER
d) PLAQUE : INTEGER

---

5 / a) valable. Il s'agit toutefois d'une déclaration de type prédéfini dans le Pascal
b) fausse. Il n'est pas admis de déclarer un sous-ensemble du type réel
c) valable
d) fausse si on n'a pas déclaré un type (par exemple MOIS) contenant les extrêmes indiqués dans la déclaration

---

6 / TYPE
COULEURS=(CŒUR, CARREAU, TREFLE, PIQUE)
MAJUSCULES='A' .. 'Z'
INTERVALLE= -10 .. 50

---

7 / a) scalaire. Il s'agit d'un seul chiffre
b) n'est pas scalaire. Une ligne est une donnée structurée
c) scalaire. C'est un nombre entier
d) n'est pas scalaire. Il s'agit d'une des deux valeurs admises pour le type scalaire standard booléen

---

8 / a) PRED (JANVIER) = DECEMBRE
b) PRED (JUIN) = MARS
c) SUCC (MARS) = JUIN
d) SUCC (AOUT) = n'est pas défini
e) PRED (DECEMBRE) = n'est pas défini

Les instructions de contrôle

Avec les instructions étudiées précédemment, on peut écrire des programmes simples en Pascal, de structure « séquentielle ».
Notons bien qu'il n'existe alors aucune possibilité de modifier cette séquence dans l'exécution des instructions, selon que certaines conditions sont ou non vérifiées.
Evidemment le Pascal offre aussi cette possibilité grâce aux instructions de contrôle.
Il prévoit notamment un ensemble d'instructions d'itération et de saut (conditionnelles ou non) sans doute plus étendu et plus flexible que dans d'autres langages, comme par exemple le Basic et le Fortran.

Les instructions d'itération

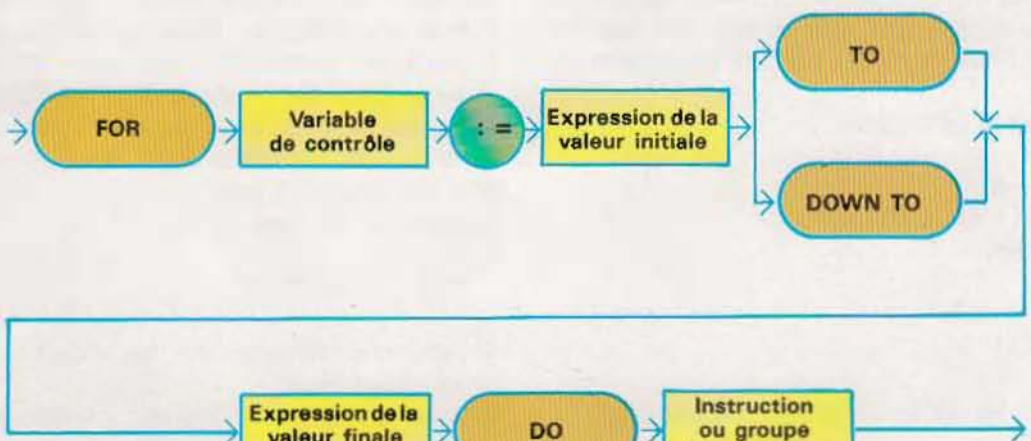
Elles exécutent un même groupe d'instructions un nombre de fois fixé à l'avance ou lié à la vérification d'une condition. Outre l'instruction FOR-TO-DO, existant également en Basic avec quelques différences, le Pascal admet deux autres instructions :

WHILE-DO
et
REPEAT-UNTIL

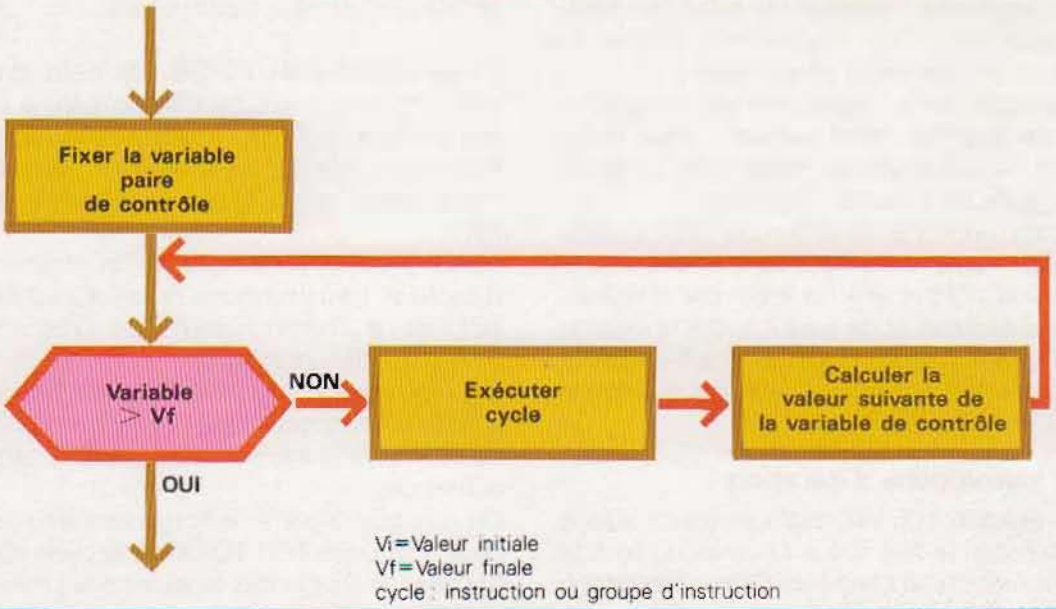
Ce sont les structures puissantes pour tous les problèmes où interviennent des exécutions répétées d'un groupe d'instructions.

L'instruction FOR-TO-DO. Elle exécute une instruction ou un groupe d'instructions à chaque occurrence d'une variable de contrôle, qui est incrémentée à chaque itération ; elle passe d'une valeur initiale à une valeur finale données.
Le diagramme syntaxique qu'on trouve ci-dessous et l'organigramme de la page suivante illustrent le fonctionnement de l'instruction d'un point de vue logique.
Cette instruction Pascal est très proche de la structure correspondante en Basic FOR-TO-NEXT, même si elle présente quelques petites différences.
On constate, à partir de l'organigramme de la page 1250, que FOR-TO-DO et le cycle d'instructions ne sont jamais exécutés si la première comparaison fait apparaître que la variable de contrôle est supérieure à la valeur finale.
En revanche, en Basic, le cycle est déroulé au moins une fois quelle que soit cette valeur. De plus, le Pascal n'admet pas, comme le Basic de spécifier, avec le mot réservé STEP, le pas d'incrémentation de la variable de contrôle.

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION FOR-TO-DO



LOGIQUE D'EXECUTION DE L'INSTRUCTION FOR-TO-DO



Revenons au diagramme ci-dessous et observons que les valeurs initiales et finales de la variable de contrôle ne sont pas nécessairement des constantes numériques : elles peuvent aussi être des valeurs fournies par des expressions évidemment de même type que la donnée.
Le calcul des expressions définissant les valeurs initiale et finale est effectué une seule fois, quand apparaît l'instruction FOR.
Cela signifie qu'une fois les valeurs initiales et finales déterminées, elles ne sont plus susceptibles d'être modifiées pendant l'exécution du cycle.
La série d'instructions :

```
K :=5 ;
FOR I:=K-1 TO K+1 DO
BEGIN
      K:=K-1 ;
      WRITELN ('I =', 1:2, 'K=', K:2) ;
END ;
```

produit les lignes d'impression suivantes :

```
I= 4  K= 4
I= 5  K= 3
I= 6  K= 2
```

Le cycle se déroule pour I entre 4 et 6, c'est-à-dire de la valeur initiale à la valeur finale calculées en fonction de la variable K = 5. Aucun changement de K à l'intérieur du cycle ne peut affecter ces valeurs.
La variable de contrôle est élevée à la valeur suivante quand on rencontre, dans l'instruction, le mot réservé TO ; elle est décrémentée avec DOWNTO. Dans ce dernier cas, la valeur finale doit être inférieure à la valeur initiale.
La valeur suivante de la variable est calculée en appliquant les fonctions SUCC en cas d'incrémentation et PRED en cas de décrémentation. Cela signifie que, n'étant pas admis comme argument pour les fonctions SUCC et PRED, le type réel ne peut servir comme variable de contrôle ; en revanche, les autres types de données simples le sont.
L'instruction du type :

```
FOR : = 'A' TO 'Z' DO
```

est correcte, mais ne sera pas acceptée par l'interpréteur Basic.
Le corps du cycle lui-même est une instruction simple ou composée (ensemble d'instructions délimitées par BEGIN et END). Les instructions d'assignation de la variable de contrôle ne sont pas reconnues à l'intérieur du corps du cycle : par conséquent, toute modification de valeurs de part et d'autre de la variable est refusée.

Toutefois il reste possible d'assigner sa valeur à une autre variable qui peut être ensuite librement utilisée.

Soit la série d'instructions :

```
SOMME:=0;
FOR K:=1 TO 5 DO
BEGIN
      SOMME:=SOMME+K;
      K:=K+2;
END;
```

l'instruction

```
K:=K+2
```

n'a pas d'existence dans le cycle et sa présence est signalée comme une erreur au moment de la compilation.
En fin d'exécution de l'instruction FOR-TO-DO, c'est-à-dire une fois le corps du cycle exécuté un nombre donné de fois, la valeur de la variable redevient « libre » (indéfinie) : elle ne peut être réutilisée par les instructions suivantes qu'à condition d'avoir été à nouveau assignée.
L'emploi de l'instruction FOR-TO-DO se justifie parfaitement lorsque des groupes d'instructions doivent être exécutés sous le contrôle d'un compteur pour un nombre de fois connu à priori. Toutefois, le nombre dépend parfois de la vérification de certaines conditions. Dans ce cas, le Pascal dispose de deux autres instructions, WHILE-DO et REPEAT-UNTIL, plus flexibles et mieux adaptées aux situations décrites.

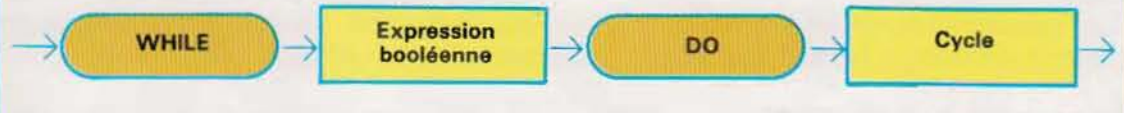
L'instruction WHILE-DO. L'exécution d'un cycle est répétée jusqu'à ce qu'une condition booléenne soit vérifiée (voir syntaxe ci-dessous). L'expression de contrôle est évaluée au début et, si elle s'avère vraie, tout ce qui suit le mot réservé DO (le corps du cycle) est exécuté. Puis, l'expression booléenne est à nouveau calculée et, si elle s'avère encore vraie, le corps du cycle est à nouveau exécuté et ainsi de suite...
Le cycle n'est plus activé quand l'expression devient fausse, comme le montre l'organigramme de la page 1252, ainsi que l'exemple donné par l'édition des 100 premiers entiers.
A l'opposé de ce qui se passe pour l'instruction FOR-TO-DO, non seulement il est admis de modifier la variable de contrôle mais encore cette opération est absolument nécessaire pour éviter les erreurs. En effet, sans instruction de modification de la variable, le cycle serait indéfiniment répété.
Si l'expression booléenne se révèle vraie au moment de l'exécution de l'instruction WHILE-DO, on dit que le cycle « boucle » (boucle fermée). Il faut donc faire très attention dans l'écriture de l'expression booléenne et vérifier quelle variable assume la valeur FALSE en fin de cycle.

Considérons donc, à titre d'exemple, la série suivante :

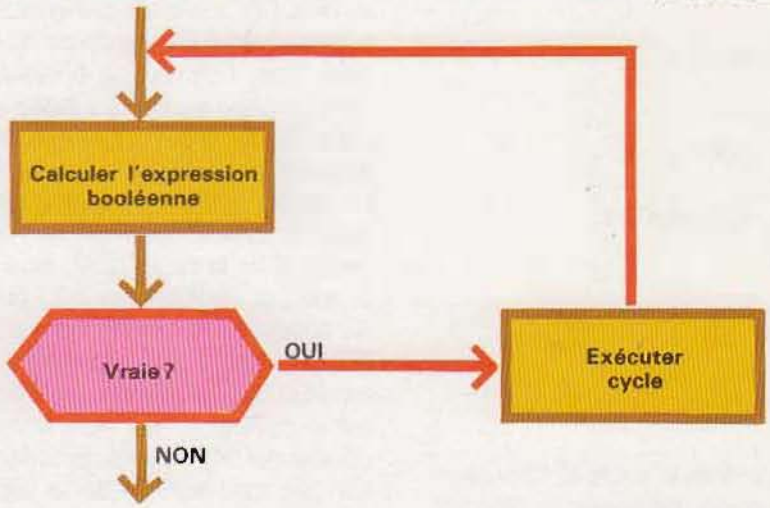
```
READ (INDICE) ;
WHILE INDICE      0 DO
  BEGIN
  .....
  INDICE:=INDICE-1;
  .....
  END;
```

Si, lors de la demande d'entrée de la variable

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION WHILE-DO



LOGIQUE D'EXECUTION DE L'INSTRUCTION WHILE-DO



IMPRESSION DES 100 PREMIERS NOMBRES ENTIERS

```
PROGRAM IMPRESSION (INPUT, OUTPUT);
INDICE : INTEGER;  (* EST LA VARIABLE QUI SERA IMPRIMEE ET UTILISEE
  VAR                 POUR LE CONTROLE DE FIN CYCLE*)
BEGIN
      WHILE INDICE <= 100 DO
      BEGIN
            WRITELN (INDICE:5);
            INDICE:=INDICE+1;
      END; (*FIN DU CORPS DE CYCLE*)
END.       (*FIN DU PROGRAMME*)
```

INDICE, on répond par un nombre négatif, on aboutira à une boucle fermée.
En effet, INDICE étant décrémenté à chaque passage, sa valeur n'équivaudra jamais à zéro et l'expression booléenne INDICE <>0 sera toujours vérifiée.
Si, dans un WHILE-DO, l'expression booléenne était fausse à la première exécution de l'instruction, le corps du cycle ne serait pas activé.
Pour l'exécuter au moins une fois, on doit donc recourir à l'instruction REPEAT-UNTIL.

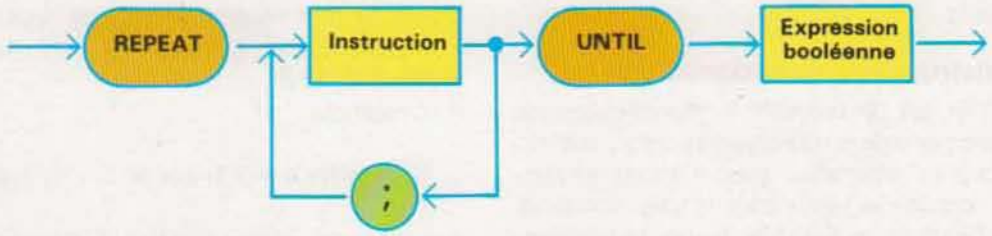
L'instruction REPEAT-UNTIL. Son action (voir diagramme syntaxique page ci-contre) est de répéter un cycle jusqu'à vérifier la condition posée. Elle se rapproche donc du WHILE-DO avec, toutefois, deux différences importantes :

— REPEAT-UNTIL exécute le test sur la première condition **après** et non **avant** de dérouler le corps du cycle ;

— cette activation se répète jusqu'à ce que la condition de contrôle soit fausse, alors que le corps du cycle est effectué (dans le WHILE-DO, l'exécution s'arrête quand l'expression booléenne se vérifie).

Ces différences sont mises en évidence par les logiques d'exécution de REPEAT-UNTIL et de WHILE-DO (pages 1252 et 1253).

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION REPEAT-UNTIL



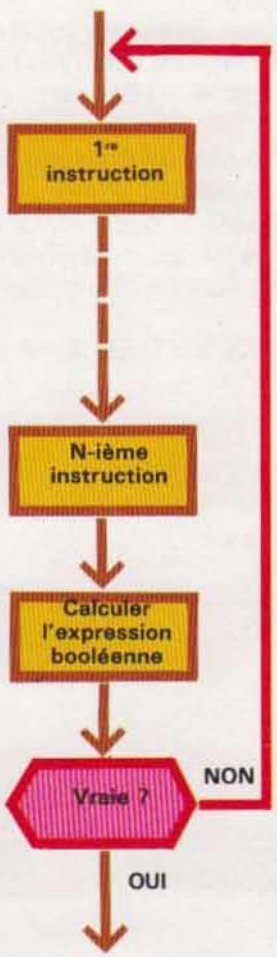
On remarque que, contrairement à WHILE-DO, les instructions constituant le corps du cycle de REPEAT-UNTIL sont exécutées au moins une fois, indépendamment de la valeur initiale de l'expression booléenne. Si celle-ci s'avère vraie, l'exécution s'achève après le premier passage.
A titre d'exemple, dans la portion de programme :

```
READLN (CONTROL) ;
REPEAT
        INSTRUCTION 1 ;
        INSTRUCTION 2 ;
UNTIL (CONTROL = 0) ;
```

en répondant à la commande d'entrée du chiffre 0, les instructions 1 et 2 sont exécutées une seule fois car la condition de contrôle est vraie.
Dans l'instruction REPEAT-UNTIL, même si le cycle comporte plusieurs instructions, il n'est nullement besoin, comme dans le WHILE-DO, de le fermer avec BEGIN et END puisque les mots réservés REPEAT et UNTIL intègrent cette fonction de délimitateurs.
Naturellement, il est toujours possible de fermer le corps du cycle entre les mots BEGIN et END, comme dans l'exemple suivant :

```
REPEAT
  BEGIN
    INSTRUCTION 1 ;
    INSTRUCTION 2 ;
      .
      .
      .
      .
      .
```

LOGIQUE D'EXECUTION DE L'INSTRUCTION REPEAT-UNTIL



```
    END ;
UNTIL (CONTROL = 0) ;
```

Sans constituer une erreur, cette manière d'écrire est, toutefois, considérée comme redondante.

Les instructions conditionnelles

Leur rôle est de modifier le flux d'exécution d'un programme en déroulant un groupe d'instructions en alternative avec d'autres instructions, lors de la vérification d'une condition. Outre l'instruction IF-THEN, également présente en Basic, le Pascal admet l'instruction CASE qui permet de choisir entre plus de deux alternatives.

L'instruction IF-THEN-ELSE. Elle est développée dans le diagramme syntaxique qui se trouve en bas de page.
Il signifie que :
— l'expression booléenne est évaluée ;
— si elle est vraie, ALORS l'instruction suivant le mot réservé THEN est envoyée en exécution ;
— SINON le contrôle du programme passe à l'instruction suivante.
Il s'agit, en général, d'une instruction composée, délimitée par les mots BEGIN et END.
On peut ainsi écrire indifféremment :

```
IF CONTROL>0 THEN A:=A+1 ;
```

soit :

```
IF CONTROL=0 THEN
   BEGIN
      A:=A+1 ;
      WRITELN ('A=', A:5) ;
   END ;
```

Dans le second cas, si la condition n'est pas vérifiée, l'instruction suivante est exécutée.
Après le mot réservé THEN, on peut insérer n'importe quelle instruction Pascal (même une autre IF-THEN).
Par exemple :

```
IF CONTROL=0 THEN IF A=10 THEN...
```

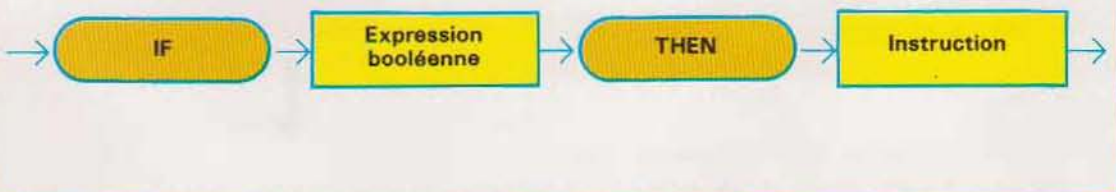
Evidemment, la seconde IF-THEN ne sera exécutée que si la condition sur la variable CONTROL est vérifiée.
L'expression booléenne à vérifier peut également être construite avec des opérateurs logiques (AND, NOT, OR) pour, par exemple, établir des conditions complexes répondant à des situations particulières : par exemple variable de contrôle comprise dans un intervalle, supérieure ou inférieure à une certaine valeur...

La ligne de programme :

```
IF (A>=0) AND (A<=100) THEN A :=1
```

place à 1 la variable A si la valeur est située entre 0 et 100.
Le même résultat s'obtient aussi en enchaînant plusieurs instructions IF-THEN. Les conditions complexes sont très utiles pour valider des données d'entrée et vérifier qu'elles ont un sens pour le système.
Ce contrôle est obtenu par une combinaison de conditions. Si la donnée fournie par l'opérateur n'est pas licite, on pourra faire imprimer un message d'erreur.

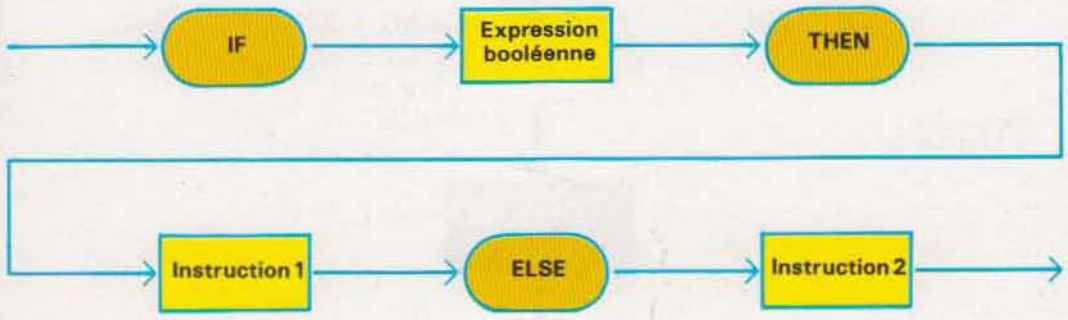
SYNTAXE DE L'INSTRUCTION IF-THEN



INTRODUCTION ET CONTROLE D'UNE DONNEE

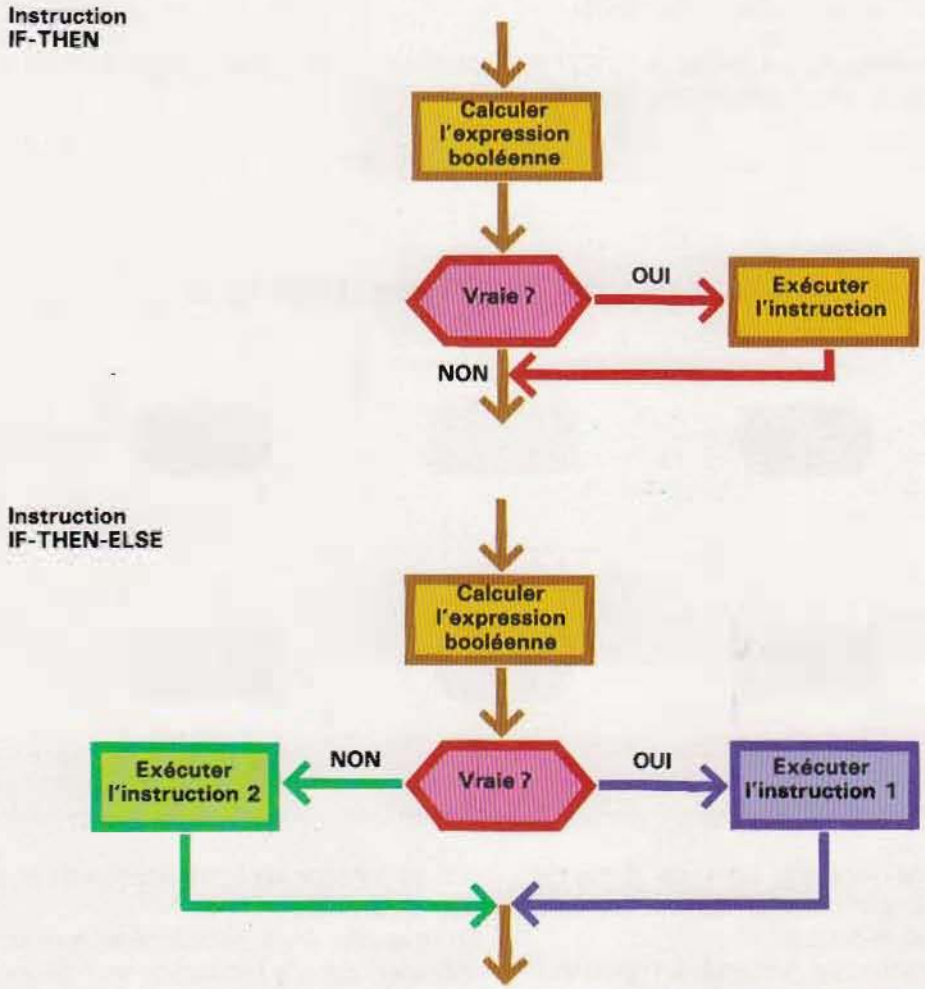
```
WRITELN ('ENTRER DATE DE NAISSANCE');
READLN (JOUR, MOIS, ANNEE);
IF (JOUR > 31) OR (MOIS > 12) THEN
   BEGIN
        WRITELN ('LA DONNEE N'EST PAS CORRECTE; RECOMMENCER ENTREE')
        READLN (JOUR, MOIS, ANNEE)
   END;
.....
```

SYNTAXE DE L'INSTRUCTION IF-THEN-ELSE



Le programme répétera alors sa demande d'entrée de donnée comme l'illustre l'exemple de programme ci-dessus.
La première instruction demande à l'opérateur d'introduire une date de naissance, la seconde de lire et d'assigner les trois variables JOUR, MOIS, ANNEE. Ces valeurs sont successivement contrôlées : si JOUR est supérieur à 31 ou si MOIS est supérieur à 12 (opérateur logique OR), un message d'erreur est imprimé invitant à recommencer l'introduction. En l'absence d'erreurs, le programme est lancé en séquence.
L'exemple est évidemment donné à titre indicatif puisqu'il faudrait également contrôler que les nombres indroduits ne sont pas inférieurs ou égaux à zéro ou que le jour du mois ne sort pas du champ des valeurs autorisées pour chaque mois.
En recourant à IF-THEN, on peut faire en sorte qu'une certaine instruction soit exécutée ou non en fonction de la vérification de la condition de contrôle.
En revanche, il est parfois nécessaire de faire exécuter, selon le résultat du test de contrôle, l'une ou l'autre des deux instructions (ou groupes d'instructions).
En Pascal, ce résultat peut être obtenu avec l'instruction IF-THEN-ELSE, (diagramme syntaxique ci-dessus). Son exécution commence avec l'évaluation de l'expression booléenne : l'instruction 1 (après le mot réservé THEN) si elle est vraie, l'instruction 2 (après le mot réservé ELSE) si elle est fausse. Naturellement, par fin d'instruction on entend aussi bien une instruction simple que composée. La différence existant entre l'instruction examinée et IF-THEN est illustrée ci-dessus.
Revenons à l'exemple précédent relatif à la validation des dates de naissance. Cette nouvelle instruction est utilisable pour écrire une section de programme qui non seulement

COMPARAISON DES INSTRUCTIONS IF-THEN ET IF-THEN-ELSE



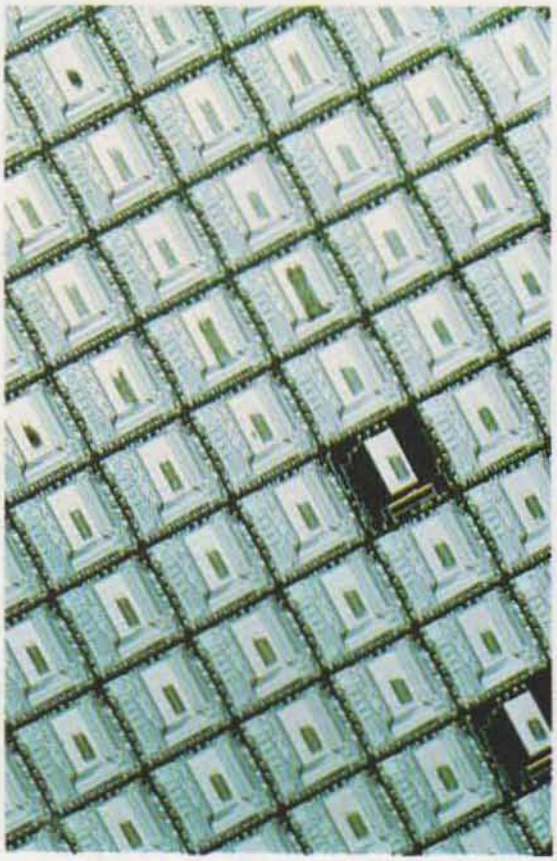
EXEMPLE D'APPLICATION DE L'INSTRUCTION IF-THEN-ELSE

```
WRITELN ('ENTRER UNE DATE DE NAISSANCE');
READLN (JOUR, MOIS, ANNEE);
IF (JOUR > 31) OR (MOIS > 12) THEN
  BEGIN
      WRITELN ('LA DATE N'EST PAS CORRECTE, RECOMMENCER');
      READLN (JOUR, MOIS, ANNEE)
  END (* il est illégal de mettre ici un ; *)
ELSE
  BEGIN
      WRITELN ('LA DATE EST CORRECTE ET EST:');
      WRITELN ('JOUR=',JOUR:5, 'MOIS=', MOIS:5, 'ANNEE=', ANNEE:6)
  END;
  .....
```

avertit l'opérateur que la date est, le cas échéant, erronée, mais aussi visualise à l'écran la date écrite quand elle est correcte (voir page suivante).

L'instruction CASE-OF. Elle est souvent considérée comme une extension ultérieure du IF-THEN-ELSE pour choisir entre plusieurs instructions (simples ou composées) celle à exécuter en fonction de la valeur assumée par une expression.
Une seule des instructions alternatives est déroulée, les autres étant sautées, comme il est représenté dans le diagramme syntaxique ci-dessous et dans le schéma de fonctionnement logique page 1258.

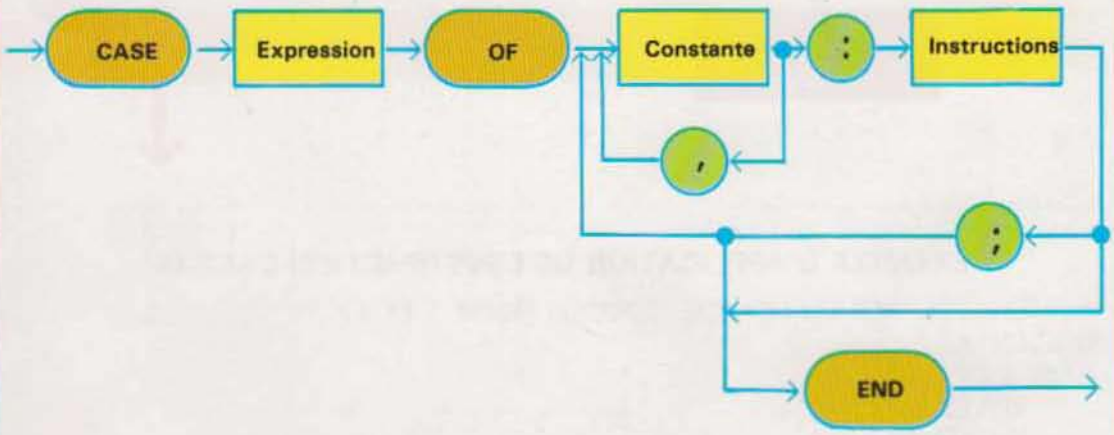
Une fois calculée l'expression venant immédiatement après CASE, sa valeur est comparée à celle des constantes (C1, C2, ... CN) situées après OF (voir exemple décrit page 1258). Si elle est égale à C1, c'est l'instruction 1 qui est exécutée, si elle est égale à la constante C2, c'est l'instruction 2 et ainsi de suite. Dans ce cas aussi, les instructions sont simples ou composées.
Si, à la demande d'introduction d'un nombre, la réponse est 1, le mot UN sera imprimé ; si on indroduit 2, le mot DEUX sera imprimé, etc. La



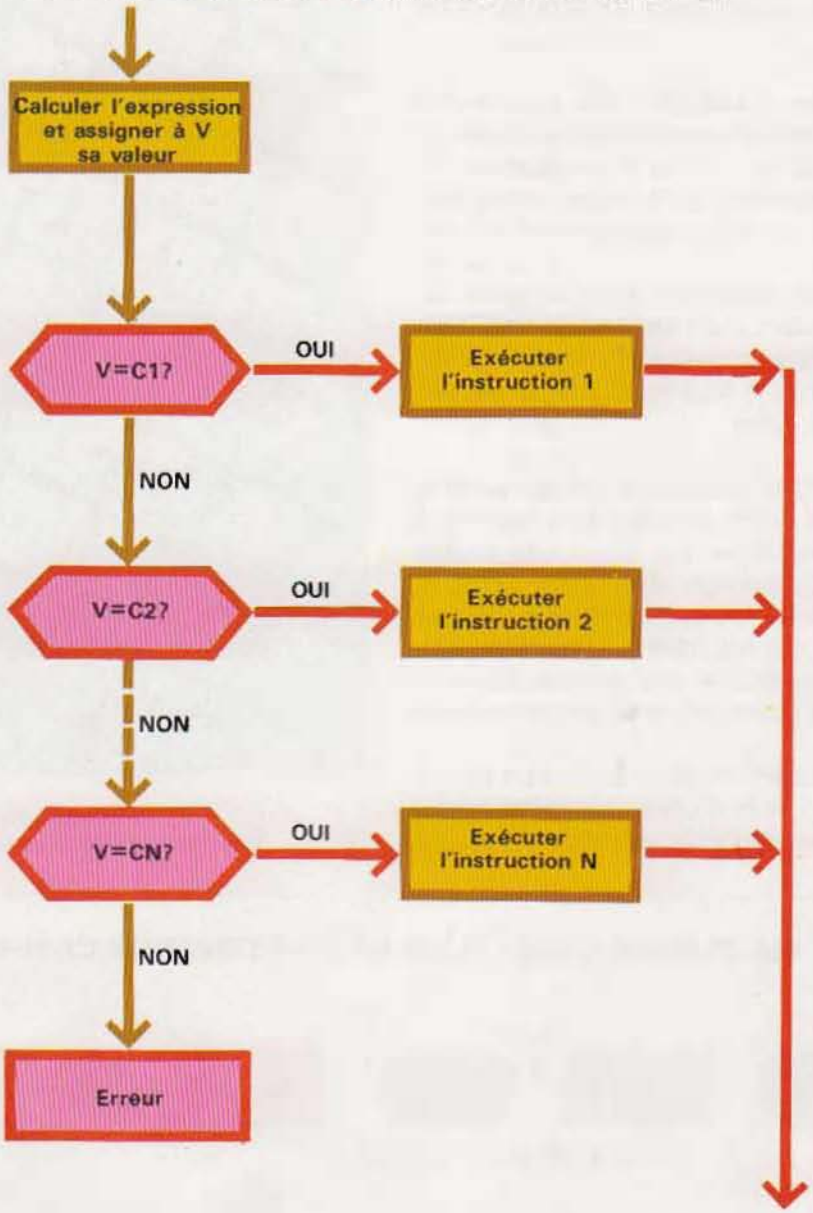
Marka

Circuits intégrés, englobés dans une matrice résineuse externe.

DIAGRAMME SYNTAXIQUE DE L'INSTRUCTION CASE-OF



SCHEMA LOGIQUE DE L'INSTRUCTION CASE-OF



EXEMPLE D'APPLICATION DE L'INSTRUCTION CASE-OF

```
WRITELN ('ENTRER UN NOMBRE COMPRIS ENTRE 1 ET 4');
READLN (K);
CASE K OF
  1 : WRITELN ('UN');
  2 : WRITELN ('DEUX');
3,4 : WRITELN ('TROIS OU QUATRE');
END
```

dernière ligne illustre le cas où plusieurs valeurs de contrôle sont associées ; l'instruction est exécutée quand la valeur de la variable est égale à l'une quelconque des constantes listées (C1, C2,... CN).
Si la valeur ne coïncide avec aucune d'entre elles, le résultat de la sélection est indéfini et est signalé comme une erreur. Il faut toujours envisager cette éventualité dans l'emploi de CASE-OF.
En pratique, cette instruction est seulement conseillée lorsque, par exemple, toutes les valeurs assumées par la variable de contrôle sont prédéfinies. On peut alors les insérer dans l'instruction et faciliter la sélection.
Quand on ne connaît pas par avance ces valeurs, l'instruction CASE-OF n'est pas conseillée. La variable de sélection n'est pas nécessairement entière ; elle appartient au type caractère ou à tout autre type défini par l'utilisateur.
Exemple : en désignant le type de variable par JOURS DE LA SEMAINE et en considérant que la variable AUJOURD'HUI appartient à un tel type, on obtient la section de programme mentionnée ci-dessous, qui imprime un des deux messages TRAVAIL ou WEEK-END en fonction du contenu de la variable AUJOURD'HUI.
Noter, au passage, que le mot-clé END sert à délimiter la fin de l'instruction CASE-OF et non à indiquer la fin d'une instruction composée ou du programme.

L'instruction de saut inconditionnel : GOTO

L'instruction GOTO modifie l'exécution séquentielle d'un programme de manière absolue c'est-à-dire non conditionnée par la valeur d'une variable de contrôle.
L'instruction de saut permet de transférer le contrôle d'un point à l'autre du programme, à condition d'avoir prévu d'identifier l'instruction marquant le point d'arrivée du saut. C'est le rôle des étiquettes (labels) qui, comme en Fortran, sont des nombres entiers (compris entre 1 et 9999) suivis du symbole : (deux-points) et des instructions à identifier.
Exemple :

```
10:A:=A+1
```

est une instruction identifiée par l'étiquette 10.
Les étiquettes sont déclarées en début de programme dans la déclaration de LABEL. Celle-ci apparaît dans la section des déclarations **avant** les déclarations de constantes et de variables (syntaxe décrite p. 1260).
Tous les entiers employés dans le programme doivent être énumérés dans la déclaration, comme dans l'exemple :

```
LABEL 10, 20, 30, 40 ;
```

Une fois ces étiquettes définies, on peut décrire l'instruction de saut inconditionnel GOTO (voir syntaxe page suivante).
L'instruction :

```
GOTO 100
```

transfère le contrôle à l'instruction identifiée par l'étiquette 100.
Celle-ci doit apparaître dans la déclaration

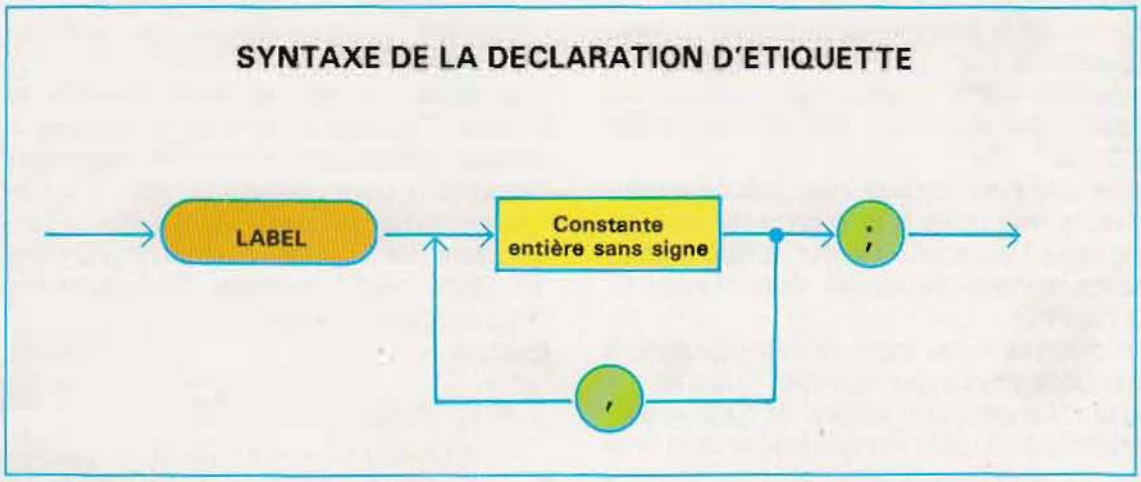
EXEMPLE D'APPLICATION DE L'INSTRUCTION CASE-OF

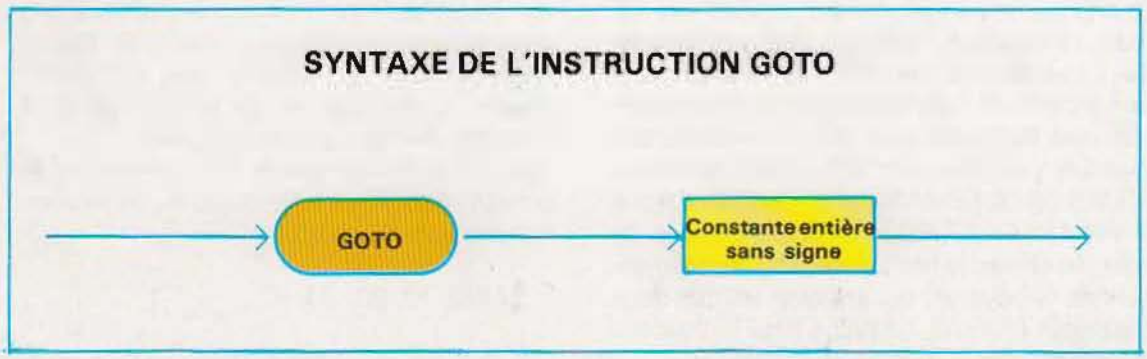
Définition du type

```
JOURS DE LA SEMAINE=(LUN, MAR, MER, JEU, VEN, SAM, DIM);
```

Section du programme

```
CASE AUJOURD'HUI OF
LUN, MAR, MER, JEU, VEN: WRITELN ('ALLEZ AU TRAVAIL!');
SAM, DIM               : WRITELN ('LE WEEK-END EST ENFIN ARRIVE!');
END; (*FIN DE L'INSTRUCTION CASE-OF*)
```





d'étiquette et dans le corps du programme. En Pascal, le recours à l'instruction GOTO est fortement déconseillé, contrairement au Basic ou au Fortran.
Elle est, en effet, reconnue comme superflue au regard de la flexibilité des instructions de contrôle propres au Pascal.
La programmation structurée, dont le Pascal emprunte les concepts de base, tend à éliminer les instructions de saut inconditionnel dont l'emploi aveugle est tourné en dérision par l'expression « spaghetti coding » à cause du nombre élevé de renvois qui en découlent.
En revanche, un programme utilisant exclusivement des structures de contrôle de haut niveau est défini comme **structuré.**

Aspects particuliers du Pascal

Les originalités du Pascal seront examinées dans ce paragraphe pour en approfondir la connaissance.
Première caractéristique importante : les **données structurées.**
En Pascal comme dans les autres langages, il est possible de définir des tableaux. Les indices sont de type logique (BOOLEAN), caractère (CHAR) ou constante.
Les exemples suivants représentent des déclarations de tableaux valides.

```
TYPE
  VECTEUR=ARRAY [1..70] OF INTEGER ;
  TABLE-ARRAY [BOOLEAN] OF CHAR ;
  MATRICE=ARRAY [1..3,1..3] OF INTEGER.
```

Outre l'assignation, toutes les opérations de comparaison (inférieur, supérieur, égal et leurs combinaisons) sont définies sur les éléments d'un tableau.
Ces derniers appartiennent nécessairement au même type. Si on part d'éléments de types différents, on utilise le type **record.**
Sa définition permet, pour chaque élément appelé champ (en anglais field) de spécifier son type et un identificateur.
Ainsi, la carte signalétique d'un élève peut être structurée comme dans le tableau ci-dessous :
— l'identificateur ELEVE personnalise une structure de type record,

— NOM est l'identificateur d'un champ de type ARRAY,
— MATRICULE l'identificateur d'un champ de type INTEGER, etc.
Un autre aspect important du Pascal concerne la **procédure** ou **fonction** représentée par un bloc d'instructions qui peut être rappelé en spécifiant le nom, comme pour les sous-programmes Fortran.
Une procédure est construite avec les mêmes règles de construction de programmes Pascal :
— partie déclarative formée de LABEL, CONST, TYPE, VAR, PROCEDURE, FUNCTION,
— partie exécutive bornée par les mots réservés BEGIN et END.
Dernière caractéristique du Pascal : la gestion de fichiers.
En Pascal, un **fichier** est défini comme une donnée structurée qui contient une série d'éléments de même type, sur support magnétique (disque ou bande).
La recherche à l'intérieur d'un fichier, en Pascal standard, s'effectue seulement en séquentiel, par accès aux enregistrements simples du fichier.
Les déclarations de type fichier suivantes sont valides :

```
TYPE
  MATRICULE=FILE OF INTEGER
  NOMS     =FILE OF CHAR
```

Sur un fichier séquentiel, on travaille en écriture (instructions REWRITE) ou en lecture (instructions RESET). En écriture, les informations sont archivées dans le fichier avec l'instruction PUT, mais, si le fichier est ouvert en lecture, on accède aux enregistrements par l'instruction GET.
L'accès aux données, toujours et seulement de manière séquentielle, est une contrainte très lourde. Aussi, de nombreuses versions du Pascal prévoient également des fichiers à accès direct ou indexé (indexed ou keyed files) qui n'existent pas en standard.

Grazia Neri/Black Star

Circuits imprimés d'un moniteur graphique en cours de montage.

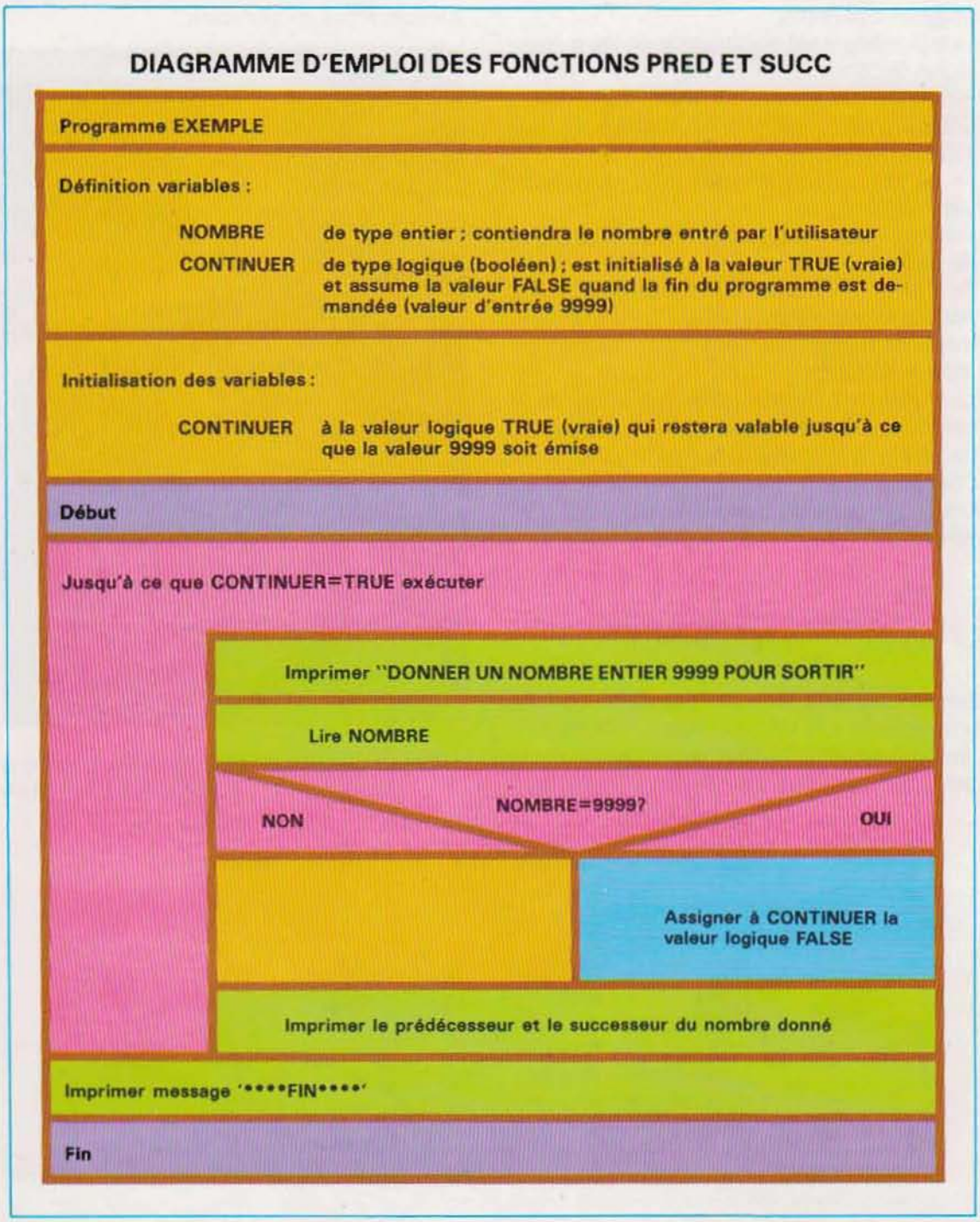
EXEMPLE DE DEFINITION DU TYPE RECORD

```
TYPE
  ELEVE: RECORD
              PRENOM    : ARRAY [1.. 10] OF CHAR;
              NOM       : ARRAY [1.. 20] OF CHAR;
              MATRICULE : INTEGER;
              AGE       : INTEGER;
              ADRESSE   : ARRAY [1.. 30] OF CHAR;
              TELEPHONE : INTEGER;
              CLASSE    : INTEGER;
       END;
```

Exemples de programmation en Pascal

L'organigramme structuré ci-dessous est un exemple d'application sur les fonctions PRED et SUCC, de même que le listing et les sorties (page ci-contre).

Le programme demande à l'utilisateur d'entrer un nombre entier et commande l'impression du prédécesseur et du successeur à la donnée entrée.
Il procède ainsi jusqu'à l'entrée de la valeur 9999, qui est interprétée comme borne de fin d'exécution.



EMPLOI DES FONCTIONS PRED ET SUCC

```
PROGRAM pred_succ (INPUT, OUTPUT);

VAR
   numero : INTEGER;
   continue : BOOLEAN;

BEGIN
   continue := TRUE;
   WHILE continue DO
      BEGIN
         WRITELN ('Donnez un numéro (9999 pour finir) : ');
         READLN (numero);
         continue := NOT (numero = 9999);
         WRITELN ('numéro précédent :', PRED (numero));
         WRITELN ('numéro suivant   :', SUCC (numero))
      END;
   WRITELN;
   WRITELN;
   WRITELN;
   WRITELN ('  **** FIN ****')
END.
```

```
Donnez un numéro (9999 pour finir) :
numéro précédent :                   3
numéro suivant   :                   5
Donnez un numéro (9999 pour finir) :
numéro précédent :                 -11
numéro suivant   :                  -9
Donnez un numéro (9999 pour finir) :


numéro précédent :                  -1
numéro suivant   :                   1
Donnez un numéro (9999 pour finir) :
numéro précédent :                9998
numéro suivant   :               10000


  **** FIN ****
```

Le contrôle de fin d'exécution est effectué par l'instruction WHILE-DO. Celle-ci répète le cycle jusqu'à ce que l'utilisateur décide d'en sortir.

Comme on peut le constater, les fonctions PRED et SUCC sont directement insérées dans l'instruction d'impression, simplifiant ainsi considérablement le programme.

L'emploi des instructions READ et READLN est illustré en page 1264 ainsi que par le listing correspondant.

Lors de l'émission des deux premières données et du caractère alphanumérique, les mêmes données sont imprimées.

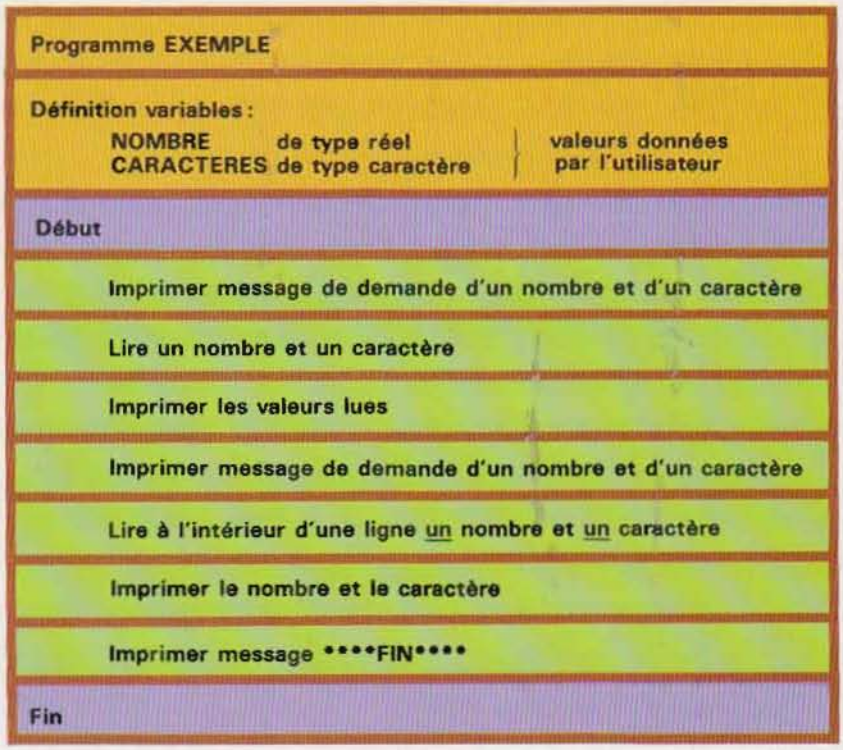
A la seconde fois, l'opérateur doit appuyer sur la touche RETURN afin de terminer la lecture et déclencher ainsi l'impression.

En effet, dans le premier cas, les valeurs sont lues jusqu'à ce que la liste des variables du READ soit épuisée alors que, dans le second cas, la lecture est fermée par l'émission du caractère CR (Retour Chariot).

Si, en utilisant le READLN, on s'aperçoit que sont émises un plus grand nombre de valeurs qu'il ne s'avère nécessaire, le programme examine seulement celles qui épuisent la liste des paramètres (dans l'exemple seulement le premier nombre et le premier caractère), les autres restant ignorées.

On trouvera un exemple d'application du WHILE-DO, ainsi que le listing correspondant, en page 1265.

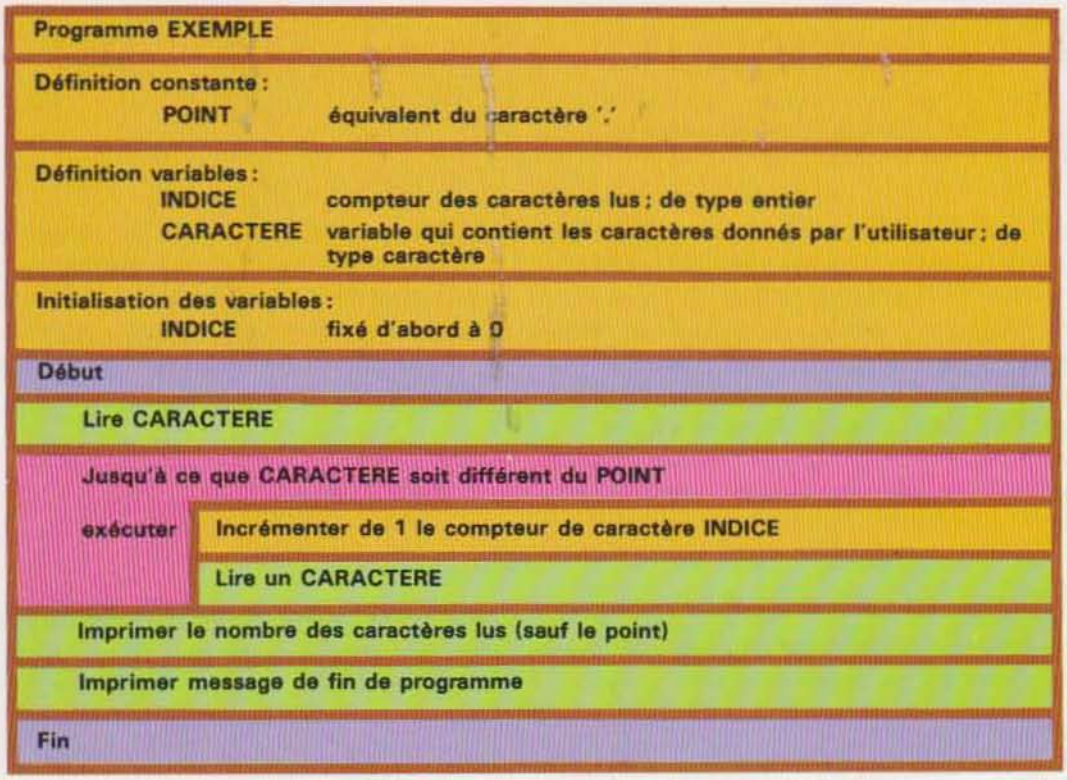
APPLICATION DES INSTRUCTIONS READ ET READLN



PROGRAMME D'APPLICATION DES INSTRUCTIONS READ ET READLN

```
PROGRAM lire_ecrire (INPUT, OUTPUT);

VAR
   numero : REAL;
   caractere : CHAR;

BEGIN
   WRITELN ('Entrez un numéro et un caractère :');
   READLN (numero, caractere);
   WRITELN;
   WRITELN (numero:5:2, '   ', caractere:1);
   WRITELN;
   WRITELN;
   WRITELN ('Entrez un numéro et un caractère :');
   READLN (numero, caractere);
   WRITELN;
   WRITELN (numero:5:2, '   ', caractere:1);
   WRITELN;
   WRITELN ('  **** FIN ****')
END.

 Entrez un numéro et un caractère :
 -1.45g

 -1.45   g

 Entrez un numéro et un caractère :
 .14580
0.15   0

 **** FIN ****
```

DIAGRAMME D'UTILISATION DE L'INSTRUCTION WHILE-DO



PROGRAMME D'APPLICATION DES INSTRUCTIONS WHILE-DO

```
PROGRAM compte (INPUT, OUTPUT);

CONST
   point = '.';

VAR
   cumul : INTEGER;
   caractere : CHAR;

BEGIN
   cumul := 0;
   WRITELN ('Entrez une phrase terminée par un point :');
   WRITELN;
   READ (caractere);
   WHILE caractere <> point DO
      BEGIN
         cumul := cumul + 1;
         READ (caractere)
      END;
   WRITELN;
   WRITELN ('Vous avez introduit ', cumul:3, ' caractère(s)');
   WRITELN;
   WRITELN ('  **** FIN ****')
END.

Entrez une phrase terminée par un point :

ceci est une phrase terminée par un point.

Vous avez introduit  41 caractère(s)

  **** FIN ****
```

Résumé : développement d'une fonction de facturation

Cet exemple d'application a été choisi pour analyser les principales fonctions qui sont normalement traitées par un ordinateur. Nous examinerons notamment une procédure complexe de gestion de fichiers et des masques ainsi que les techniques d'arrondi sur les valeurs numériques. En partant d'un problème spécifique, on a voulu donner un exposé simple appuyé sur des exemples concrets et qui ne demandent aucune connaissance particulière.

Analyse du problème

Le but de la procédure est de gérer une série de transactions qui déterminent le mouvement des articles d'un magasin ou des importations de vente et d'achat. Tous les flux de données s'articulent sur ces deux voies (ventes/achats). N'importe quelle activité de nature commerciale demande, au minimum, la gestion des fichiers suivants :

— Clients (CLI)
— Fournisseurs (FOU)
— Articles (ART)
— Caisse (CAISSE)

Les fiches signalétiques et les montants (créance pour les clients, dettes pour les fournisseurs) sont enregistrés dans les fichiers

Présentation du système de traitement Apple.

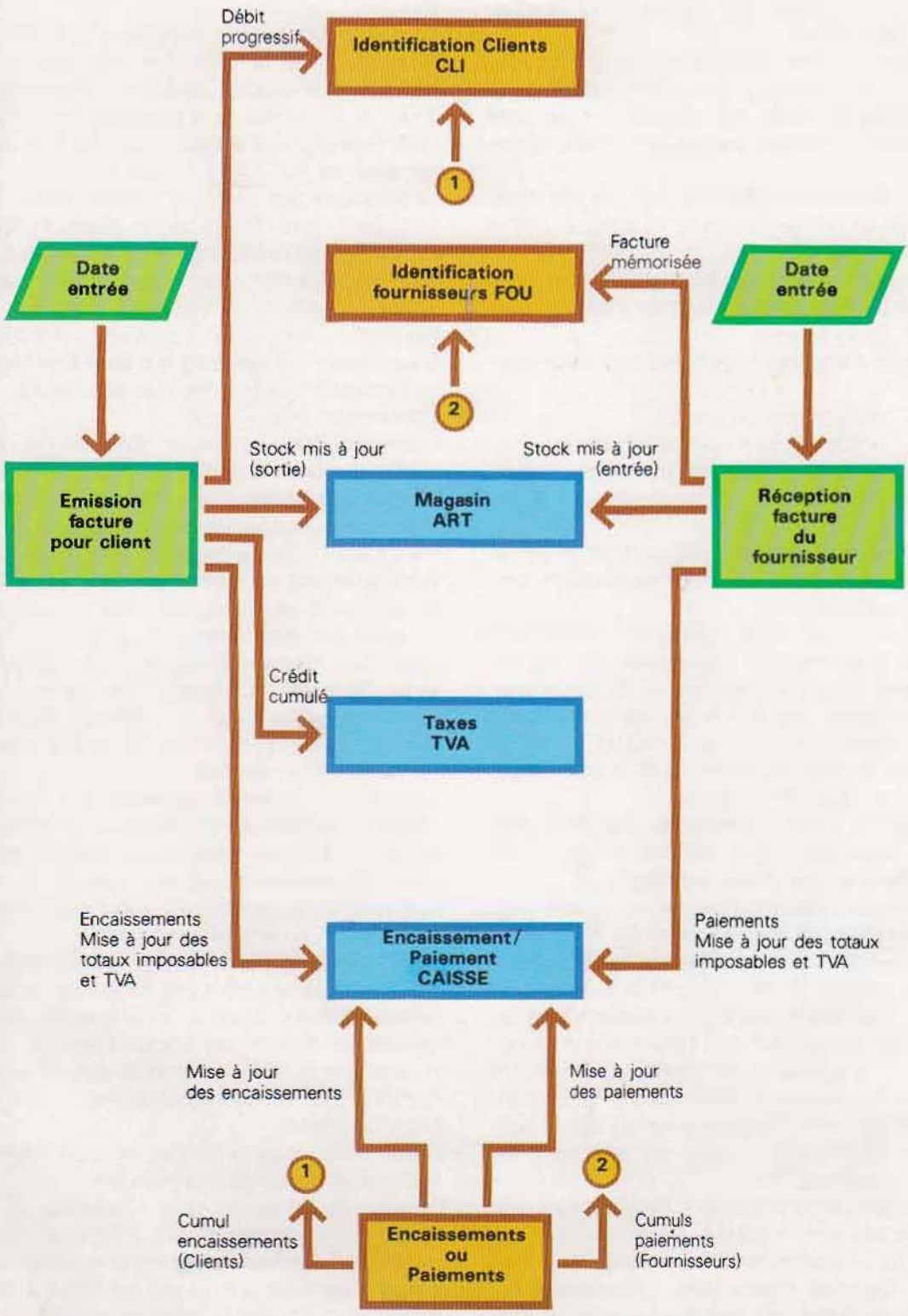
G. Pochetti/Marka

Clients (CLI) et Fournisseurs (FOU). Le fichier Articles (ART) contient les descriptions des marchandises mouvementées et le fichier CAISSE est utilisé pour mettre à jour la situation financière.
L'existence et le contenu de ces fichiers doivent être considérés de manière souple ; par exemple, le fichier ART pourrait ne pas être présent, ou encore concerner d'autres activités.
Avec quelques adaptations, la procédure développée ici conservera toute sa validité, même dans ce cas.
Le document de base, à l'origine d'un mouvement de marchandise ou d'un transfert d'argent, est la facture.
Elle doit comporter les informations suivantes :

— signalement du client
— articles et prix correspondants
— taux de TVA et montants correspondants

Les deux premiers champs ont une signification évidente alors que le dernier nécessite certains commentaires.
La TVA est un impôt calculé en pourcentage sur le montant d'une transaction ; ce taux dépend du type d'article objet de la transaction. Il peut arriver que l'on doive, dans une même facture, calculer un montant total de la TVA qui résulte de montants obtenus chacun en appliquant un taux différent.
La facture est soit **émise,** elle concerne alors une vente (d'un bien ou d'un service), soit **reçue** (il s'agira d'une acquisition).
En fonction des informations contenues dans les factures, on doit décrémenter le stock (la marchandise sortie) et simultanément augmenter le compte du client auquel elles se rapportent, ainsi que la somme encaissée. Dans les factures reçues, on doit incrémenter le stock (c'est une arrivée de marchandise), le montant dû au fournisseur et les sommes dépensées. Il faut, en outre, mémoriser les données relatives aux charges fiscales, en particulier les montants de la TVA.
En jargon commercial, la TVA s'appelle « une partie tournante » c'est-à-dire un montant qui peut être transféré entre les personnes intervenant dans les transactions successives. Par exemple, lorsqu'on reçoit une facture d'un fournisseur d'un montant 1 000 F plus 70 F de TVA (7 %), le dû sera 1 070 dont 70 comme impôt que le fournisseur devra verser à son tour.
Si ce bien est revendu, supposons à 1 500 F, le montant de la TVA correspondant sera 105 F (même taux) et l'acquéreur devra verser 1 605 F. En réalité, une partie de cette TVA (105 F) a déjà été versée (70 F), la TVA due est donc de 105 — 70 = 35 F.
La procédure doit mettre en évidence les montants de TVA encaissés par les clients et ceux versés aux fournisseurs de façon à pouvoir calculer la différence (voir organigramme page 1268).
Apparaît alors la nouvelle fonction Encaissements/Paiements qui, sans être liée à la facturation, n'en constitue pas moins un complément utile.
L'émission d'une facture provoque un mouvement des montants du seul point de vue comptable. A l'égard des charges fiscales, le chiffre indiqué dans une facture émise intervient comme s'il avait déjà été encaissé, alors qu'en réalité le paiement peut ne se produire que longtemps après et même être fractionné (par exemple par des règlements à 60 ou à 90 jours). Pour toutes ces raisons, il est indispensable de gérer la situation Encaissements/Paiements qui se réfère à un véritable mouvement de caisse et fournit ainsi l'état des liquidités (créances et dettes).
La gestion de la caisse n'est soumise à aucune obligation ou formalité particulière et peut donc être structurée selon les nécessités ou les habitudes de l'entreprise. Le seul objectif est de maintenir une situation mise à jour des échanges d'argent en entrée et en sortie.
La connaissance d'une situation globale en dettes et en créances n'est pas suffisante : il faut personnaliser les clients ou les articles respectivement en débit et en crédit. Toutefois, les encaissements et les paiements doivent aussi mettre à jour les données relatives à chaque client ou article.
Il faut, enfin, prendre en considération l'éventualité d'un retour de marchandise.
Dans ce cas, il existe deux possibilités. Si la facture correspondante a déjà été émise, il faut retirer les sommes correspondantes avec un « avoir » ; si la facture n'a pas été émise, il faut seulement réintégrer le stock du magasin.

ORGANIGRAMME DE LA FONCTION FACTURATION



Dans notre analyse du livre : ***« Le système Aristote »*** *de* ***René Dzagoyan,*** *publié chez Flammarion, nous avons suivi pas à pas comment une grande puissance mondiale (en l'occurrence les Etats-Unis) en était arrivée à se confier quasiment entièrement aux ordinateurs, et comment elle avait réussi à entraîner dans son orbite la quasi-totalité des pays du monde occidental. Pour n'être pas en reste, les pays du bloc soviétique en font autant.*

Nous en sommes arrivés au moment où la pieuvre occidentale prend ses aises en étendant ses longs tentacules, sous couvert, tout d'abord, d'activités économiques, ce dont ne manque pas de s'étonner un des protagonistes, bien conscient de l'importance primordiale des questions militaires :

« Et tout cela, simplement pour faire des calculs économiques ! »

La réponse, comme vous pouvez bien le penser, ne se fait pas attendre :

« Pas seulement... nous l'utilisions aussi pour le commerce international. Par exemple, chaque usine importante était munie d'un petit ordinateur branché sur l'ordinateur central... Un programme spécial lui disait quelle devait être sa production dans les douze prochains mois. S'il était d'accord avec l'ordinateur, il appuyait sur un bouton, et tous les robots de son entreprise ou tous les plans de ses contremaîtres se réglaient en fonction de l'objectif défini...

Une véritable ***pieuvre,*** *à un point que vous n'imaginez pas. Il suffisait d'installer, chez soi, un terminal relié à l'ordinateur central pour recevoir, sur son écran, des cours de physique en provenance de l'Italie ou des cours de cuisine du Japon. En poussant plus loin, vous pouviez brancher votre cuisinière électrique sur votre terminal et la faire régler par un programme culinaire élaboré et enregistré de l'autre côté de la terre.*

— Pourtant, les cours de cuisine n'ont jamais tué personne !

— Vous êtes bien naïf ! Vous vous imaginez bien que le système a un usage essentiellement ***militaire.*** *Sans cela, les Etats utilisateurs ne nous auraient même pas accordé un quart d'heure d'attention. Quant aux crédits, n'en parlons pas... »*

Vous savez, maintenant, ce qu'est Aristote... Mais en êtes-vous bien sûr ? S'agit-il seulement d'un ensemble de possibilités au service de l'homme ? C'est oublier qu'une somme de choses peut valoir plus que ses composants. « Le système Aristote est maintenant en contact permanent avec un nombre incalculable d'ordinateurs. Il sera probablement le plus grand ***cerveau électronique*** *que l'humanité aura jamais créé.*

Je crois que ce système peut atteindre ce que j'appelle une ***« masse critique »*** *qui, une fois atteinte et franchie, lui confère une intelligence analogue à l'intelligence humaine. Dans ce cas, le système Aristote serait capable de déterminer lui-même ses objectifs, d'établir lui-même la stratégie permettant de les atteindre, et de refuser toute instruction humaine qui les contredirait. En un mot, Aristote serait un système totalement* ***indépendant de la volonté humaine. »***

Plus tard, on apprend que le système russe est similaire au système occidental et que ce dernier le manipule, puisqu'il sait exactement comment il est fait. On apprend, au passage, ce qui nous intéresse tout particulièrement, que la stratégie fondamentale du système est fondée sur le ***jeu de Gô****. Les choses ne s'arrangent pas lorsqu'on apprend que le système vient d'activer les bombes à neutrons dont la caractéristique la plus intéressante est d'être* ***« propres »,*** *c'est-à-dire de détruire tout ce qui est vivant sans endommager, en quoi que ce soit, tout ce qui est inerte. Dont les précieux ordinateurs :*

« Les seules choses qui restent intouchées sont les machines. Le règne minéral est insensible aux flux neutroniques. Pourquoi le Système cherche-t-il à épargner les machines ? »

La réponse ne tarde pas, d'abord sous son aspect philosophique :

« Dans la finalité du programme, réside la réalisation matérielle de la ***Rationalité,*** *but ultime d'Aristote. »*

Conséquence pratique :

« Aristote considère que l'espèce humaine s'oppose à la réalisation du But Ultime, et qu'il se trouve malheureusement dans l'obligation de procéder à la déshumanisation de la Terre. »

Logique, n'est-ce pas ?

On est, cependant, prié de ne pas se laisser aller à la peur. Comme le déclare un des généraux du livre : ***« Les hommes n'ont que les guerres qu'ils méritent. »***

Mais sommes-nous si méritants que cela ???

Les jeux vidéo (suite) ou les aventures de Fred

Reprenons, à présent, l'étude de ce très intéressant jeu de guerre que constitue le Gô. Nous en avons étudié la philosophie, l'histoire et les règles, puis nous nous sommes accordé une étape pour réflexion sur les ***implications de l'ordinateur*** *dans la vraie guerre (qu'elle soit froide ou chaude).*

Voilà qui nous met tout a fait à l'aise pour étudier l'adaptation du Gô à l'ordinateur.

Il faut tout d'abord se rendre compte que les règles du jeu de Gô sont infiniment plus complexes qu'elles ne sont apparues au cours de cette brève étude, et qu'un jeu de guerre capable de conquérir le monde comporte infiniment plus de ressources cachées qu'il n'apparaît au cours d'une brève étude. Nous avons volontairement simplifié ces règles, tout en essayant de ne pas en diminuer la signification et de ne pas en biaiser la nature, dans l'optique où nous allons examiner une version simplifiée du Gô, destinée à des joueurs mixtes : un automate, un être humain.

C'est pour cette raison que notre automate jouera sur un ***Gô Ban réduit*** *à 9 × 9 intersections. Les autres règles utiles pour développer correctement le Gô sont décrites dans le tableau de la page 1226 du numéro précédent.*

Pour pouvoir construire un ***adversaire artificiel,*** *un automate avec lequel nous jouerons au Gô, nous devons, au préalable, analyser les critères qui permettent de développer correctement la dynamique du jeu.*

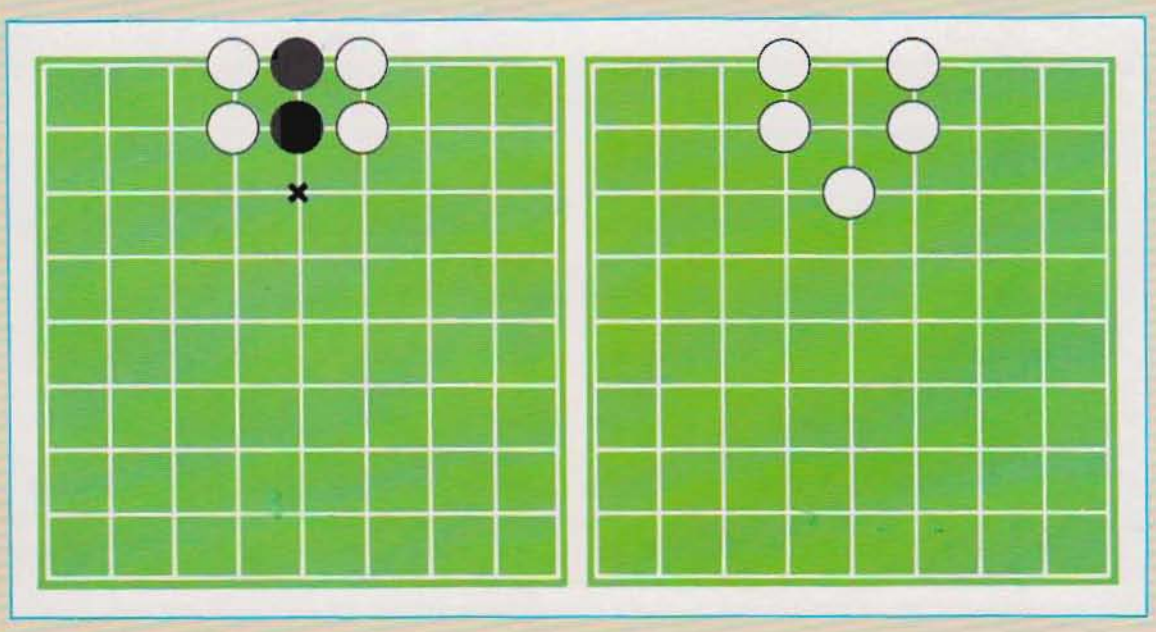
Organiser un ***programme*** *signifie construire une source de signaux de contrôle qui gouvernent les manifestations de l'automate vers l'extérieur. Comme il n'est pas très sympathique d'avoir affaire au programme « Machin », nous habillerons notre compagnon de jeu d'une personnalité qui nous fasse un peu rêver.*

Je ne sais pas si vous avez déjà rencontré Fred au détour d'une bande dessinée américaine, mais je ne vois pas pourquoi nous différerions plus longtemps de faire connaissance avec ce sympathique lion. D'autant plus que son heureux caractère lui permet de subir, avec une patience particulièrement résignée, les expériences aussi sottes et grenues (pardon pour ce jeu de mot qui prétend parler d'expériences saugrenues) qu'un de ses amis, savant et farceur, lui fait régulièrement subir en tant que cobaye.

Les procédures ne seront pas exprimées dans un langage de programmation mais en français

La position des pions sur le Gô Ban montre deux pions noirs en Atari.

Ici, le blanc a joué en capturant les pions noirs.



structuré, ce qui permettra leur implantation dans les différents langages à la disposition de l'utilisateur.

Notons que nous avons la possibilité d'utiliser un langage qui se prête très bien à la **construction de Fred : le Logo.** Celui-ci est commercialisé dans diverses versions adaptées à des calculateurs personnels. Pour ce cas, c'est la version A1.5 du Logo, sur le calculateur MPF3 de Digitek, qui a été utilisée, afin d'élaborer une série expérimentale de procédures destinées à l'étude et au développement de Fred.

Il est clair que la quasi-totalité des procédures que nous allons définir aura pour but de décrire, chaque fois, une portion du jeu qui sera alternativement gérée par l'être humain ou par Fred lui-même. Lorsque son implémentation deviendra complète, nous prendrons beaucoup d'intérêt à jouer avec lui. En effet, au début, nous aurons un peu bafouillé et ses coups nous paraîtront dépourvus de subtilité, mais Fred s'y fera vite...

Passons au concret. La première opération que nous devons effectuer est la construction du Gô Ban et des pions, objets au travers desquels nous aurons la représentation de coups effectués. Ce programme particulier ne concerne que la **gestion de l'écran.** Celui-ci ne fait pas partie de Fred mais constitue, comme on l'imagine, un complément nécessaire. Nous le décrirons donc en premier lieu, afin de mettre au clair les exigences de **visualisation du jeu.** Car une bonne partie du charme réside dans le fait que nous fournissons à Fred un environnement soigné.

Nous savons bien qu'une grande partie du charme d'un jeu réside dans l'importance des rêves que nous pouvons y accrocher. Et, à ce propos, si les constructeurs de jeux vidéo n'inventent pas toujours des logiciels bien originaux, ils soignent tout particulièrement la **présentation,** l'apparence étant, comme on le sait bien, très porteuse de fantasmes.

Pour ce qui concerne nos rapports avec Fred, on peut risquer la remarque suivante : aujourd'hui, bien des gens, tout en restant fort intéressés par la logique des jeux de guerre, n'arrivent plus à projeter sur eux des fantasmes satisfaisants. Sans doute parce qu'ils risqueraient d'être bien trop proches de l'apocalypse. Et surtout parce qu'ils ne voient pas comment ils réussiraient à dominer facilement la situation, le cas contraire n'étant pas particulièrement gratifiant.

Tandis qu'avec notre sympathique Fred, tout devient facile : ce n'est pas ce lion débonnaire qui cherchera à créer des ennuis profonds à son adversaire ! Tout au plus l'agacera-t-il suffisamment pour que son intelligence se sente mise en appétit par cet adversaire infatigable et d'une constante urbanité.

Rappelons que le Gô Ban, qui est en réalité une grille carrée de 19 × 19 intersections par côté, sera, dans notre cas — et pour des raisons de simplicité — réduit à de plus petites dimensions : 9 × 9 intersections.

Il sera présenté sur l'écran par le calculateur et devra fournir les informations essentielles : la position des pions encore disponibles pour chaque joueur, les indications relatives aux pions prisonniers, ainsi que l'éventuelle déclaration d'**Atari** de la part d'un des deux adversaires.

Il existe, en effet, une étape de la partie au cours de laquelle un groupe de connexion pos-

DESCRIPTION DU MODULE CARTE

```
CARTE (couleur, x)
DEBUT
  SI il y a un pion en x, FAIRE
     il est de la couleur"couleur", FAIRE
     il n'a pas été marqué
  ALORS
     DEBUT
        Marquer le
        RAPPELER CARTE (couleur, NORD (x))
        RAPPELER CARTE (couleur, EST (x))
        RAPPELER CARTE (couleur, SUD (x))
        RAPPELER CARTE (couleur, OUEST (x))
     FIN
  SINON
     DEBUT
        S'il n'y a pas de pion en x
        ALORS
           DEBUT
              Marquer le point comme liberté
              Incrémenter le compteur des
              libertés
           FIN
     FIN
FIN
```

Avec le module CARTE Fred cherche, trouve et décrit les libertés d'un groupe de connexion contenant un pion de couleur "couleur" dans le point X. CARTE rappelle lui-même de façon récursive, en mémorisant X.

sède une seule liberté. La situation est telle, alors, que l'adversaire s'apprête à le capturer. Dans le Gô, tout comme dans les Echecs (dans ce cas, on procède à l'annonce « échec » ou « échec et mat »), le joueur doit avertir l'adversaire de ses intentions offensives. Ce qui est directement inspiré des règles de chevalerie tant orientale qu'occidentale. On laisse donc une chance ultime à l'adversaire de rassembler ses forces — ici, en l'occurrence ses dernières miettes d'astuce — afin d'au moins essayer de s'en tirer honorablement.

Un exemple de coup, qui exige une déclaration d'Atari, se trouve illustré en page 1270, sur l'échiquier de gauche.

Le programme idéal devrait faciliter l'évocation de tous... ou du moins d'une **grande partie des coups** effectués par les joueurs. Cette exigence peut être résolue selon les deux possibilités suivantes.

La première est simplement de mémoriser la représentation graphique d'un coup (c'est-à-dire la situation du Gô Ban à un moment donné) et de construire, dans le programme qui gère le jeu, un espace dans lequel mettre en ordre temporel toutes les images graphiques apparues sur l'écran.

On obtiendra, en les redemandant en ordre temporel croissant ou décroissant, une description totale ou partielle de la partie.

Examinons, à présent, la seconde solution. En effet, celle que nous venons d'évoquer manque un peu de subtilité. Elle occupe inutilement beaucoup d'**espace mémoire,** elle n'est dirigée par aucun **algorithme interactif ;** enfin, elle ne **stimule** guère **l'imagination** du programmeur. En un mot, il convient de mettre la médiocrité sur le paillasson !

Dans la seconde solution, une routine prend en considération un pion déposé sur le Gô Ban comme provenant d'un coup du joueur ou de Fred. Elle détermine l'endroit du coup sur le Gô Ban et appelle une seconde routine qui contient un algorithme qui dessine le Gô Ban et les pions sur l'écran.

Subsiste encore le problème de l'input du coup précédent. Il peut être fourni au calculateur à l'aide d'une description alphanumérique correspondant aux références d'intersection, en attribuant un code d'identification à chaque ligne et à chaque colonne.

Autre solution pour l'indication du repérage : la

DESCRIPTION DU MODULE JEU

```
JEU
DEBUT
  Déposer les pions noirs de handicap
  TANT QUE la partie dure, FAIRE
    DEBUT
      Visualiser le Gô Ban
      Faire la saisie du coup Blanc
      (entrer au clavier)
      APPELER EFFET COUP BLANC
      (pour examen de l'effet)
      APPELER EFFET COUP NOIR
      (pour effet coup du Noir)
      APPELER MODELES
      (pour chercher un modèle de référence)
      Poser le pion noir
    FIN
FIN
```

Le module JEU constitue le corps central du programme que nous avons appelé.

manipulation de divers dispositifs, dont la **souris,** la **table graphique,** le **joystick.**

Ensuite, Fred doit correctement interpréter la signification des figures de jeu pour développer alors son activité d'évaluation et de proposition de son coup.

La routine Carte permettra donc d'identifier les groupes de connexion et les pions les constituant (rappelons qu'ils vont du simple pion jusqu'aux agglomérats de grandes dimensions), en recherchant et en marquant pour chacun d'eux les libertés correspondantes.

La procédure d'identification sur :

— un pion à la fois dans le groupe de connexion,

— une liberté à la fois dans le groupe de connexion,

— la recherche systématique d'un pion de l'autre couleur à proximité du groupe de connexion concerné se déroule sur chaque intersection du Gô Ban. Fred mènera l'enquête : il fixera ainsi dans sa mémoire une représentation correcte des pions sur le Gô Ban.

Ceci se fera en « sous-entendant » de manière significative les indicateurs relatifs à ce point du Gô Ban et qui se réfèrent aux divers paramètres d'identité du pion examiné (avec ses différentes attributions typiques (liberté, insertion dans un groupe...) qu'il présente à ce moment-là.

(suite page 1295)

Fichiers utilisés

Le format des enregistrements des principaux fichiers (p. 1274) comporte une zone initiale de 3 caractères (octets 1 à 3) pour le code d'identification. En Basic, ce code n'est pas nécessaire car les fichiers sont à accès direct. Le code correspond au numéro d'enregistrement : le premier article introduit en stock occupe l'enregistrement 1 et a le code 1, le second occupe l'enregistrement 2 et a le code 2...
Cette zone a été créée à la fois pour la procédure de facturation en Cobol et pour donner une plus grande souplesse au programme. En effet, il est fréquent qu'un utilisateur final conserve son propre système de codification des articles, des clients et des fournisseurs.
Si la gestion d'un magasin n'emploie pas de code externe, chaque article est identifié par son numéro d'entrée (n° d'enregistrement). Cependant, certains préfèreront adopter une abréviation (un code non numérique) qui indiquera de façon mnémonique le type de marchandise.
Ce second cas soulève quelques difficultés. En effet, alors que pour un numéro d'enregistrement la lecture des données relatives à un article est immédiate (sa position dans le fichier étant indiquée par le code) quand le code est une abréviation arbitraire, il faut effectuer une recherche pour chaque article. Le temps nécessaire à cette recherche n'est acceptable que pour un nombre d'articles réduit. Au delà, il est alors indispensable que les marchandises en stock soient ordonnées par code, pour effectuer une recherche de type binaire, par exemple, ce qui accélère le processus.
Le nombre d'articles pour lequel on devra recourir à l'une ou l'autre de ces deux méthodes (recherche dans un fichier ordonné) est difficile à évaluer a priori. Il est nécessaire d'analyser chaque cas particulier, en tenant compte aussi du langage et du matériel utilisés. Pour "normaliser" le format, la zone consacrée a été prévue même dans les fichiers ne la nécessitant pas absolument ; cette zone ne contient alors que des caractères de remplissage.
Les formats représentés page 1274 ne concernent que la zone des données. Les éventuels enregistrements répertoires sont gérés séparément. Ils contiennent des zones d'importances diverses (par exemple le nombre d'enregistrements occupés, la longueur totale, etc.). Toutefois, en Basic, les enregistrements ont normalement une longueur fixe, et ceux du répertoire la même longueur que pour les données. Les nombres inscrits au-dessus de chaque enregistrement représentent la longueur en octets et la position de chaque zone. La zone de description de l'enregistrement du fichier ART, par exemple, comporte 20 caractères et occupe, dans l'enregistrement, les octets 4 à 23. Cette forme de représentation graphique est très utile tant pour le minutage que pour la mise au point ; elle est toujours utilisée en complément de la documentation du programme, quel que soit le langage adopté.

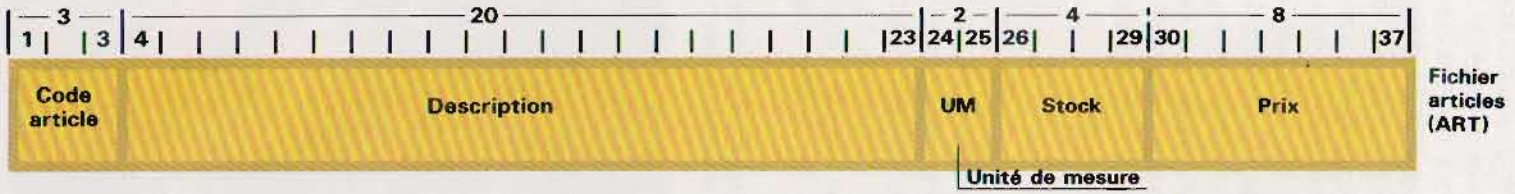
Menu principal

Le menu principal de la procédure est représenté par l'organigramme de la page 1276. Aucune particularité importante n'étant à signaler, son listing n'est pas fourni. La page 1277 détaille le schéma logique d'ensemble. En entrant le numéro de la fonction sélection-

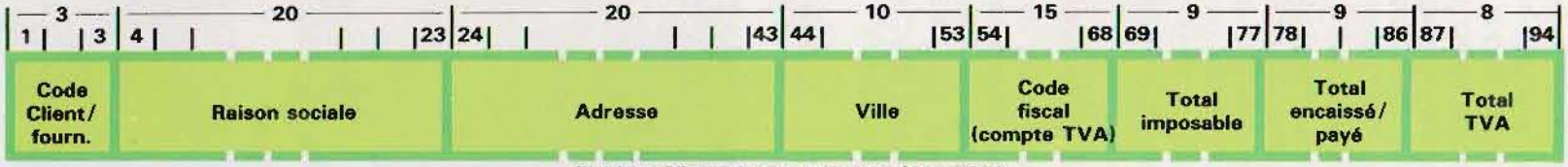
Assemblage de la carte mère d'un ordinateur personnel.

J.P. Laffont/Grazia Neri-Sygma

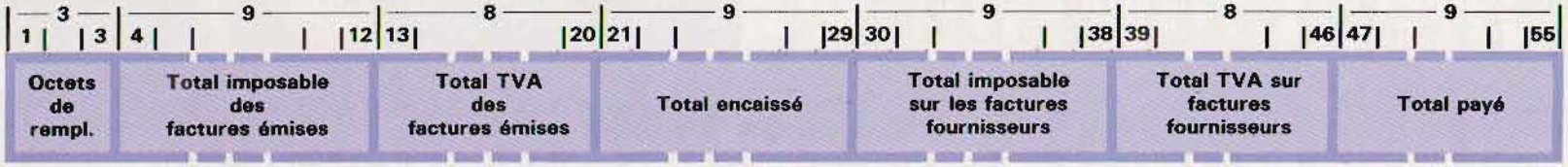
FORMAT DES ENREGISTREMENTS DE FICHIERS



Fichier articles (ART)

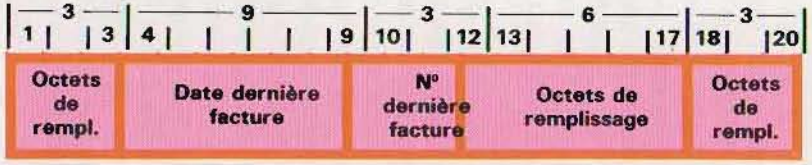


Fichiers Clients et Fournisseurs (CLI, FOU)



Fichier taxes TVC

Uniquement pour le fichier TVA (sinon, octets de remplissage)



Fichier répertoire (DIR)

FICHIERS UTILISES DANS LA PROCEDURE DE FACTURATION

Nom	Long. de l'enreg. en octets	Description
REP	20	Contient des informations de caractère général, comme la date et le numéro de la dernière facture émise. La numérotation des factures doit être progressive ; il faut donc mettre en mémoire le dernier numéro utilisé pour pouvoir reprendre à partir du suivant. La date est utile comme élément de contrôle.
ART	37	Contient les articles en magasin, les données relatives au stock et au prix des articles. La zone UM (unité de mesure) n'est utile que si différents types de marchandise sont prévus. La zone stock est décrémentée des factures émises et incrémentée des factures reçues. Il est possible, pour augmenter la fonctionnalité de la procédure, d'ajouter une nouvelle zone contenant le taux de TVA.
CLI, FOU	94	Ces fichiers sont identiques. Ils contiennent les identifications (l'un pour les clients, l'autre pour les fournisseurs), le montant imposable (somme des montants de chaque facture), le total de TVA et le total des encaissements ou des paiements (mis à jour au moyen de la procédure encaissements/paiements).
CAISSE	55	Contient les totaux des montants imposables, de TVA, des factures et des mouvements encaissements/paiements. Ces données s'obtiennent aussi par addition des montants des différents enregistrements des fichiers CLI et FOU. Cette totalisation des valeurs dans un fichier séparé est, certes, redondante, mais elle permet d'avoir une vision immédiate de la situation économique. Ce fichier contient un seul enregistrement de données
TVA	32	C'est un récapitulatif, pour chaque facture émise, du montant de la TVA divisé par le taux de taxation. Son existence est liée à la nécessité de produire un document fiscal (registre de TVA) sur lequel ces données doivent apparaître. Une fois imprimées, les données du fichier sont annulées et le fichier est alors réutilisable. Cette impression est généralement trimestrielle : c'est pourquoi le nombre maximal d'enregistrements du fichier doit être égal au nombre de factures prévues pour la période de 3 mois.
TVC	32	Son format est identique à celui du fichier TVA, mais il contient les totaux cumulés des montants et de la TVA en fonction des taux de taxation. Il sert à connaître la situation d'ensemble en fin d'année.

née dans le menu, on obtient le chargement du menu secondaire correspondant. Intervient alors une nouvelle sélection parmi les rubriques affichées pour charger et faire exécuter la partie désirée de la procédure. Cette structure favorise une segmentation du programme. Chaque procédure correspond à un bloc autonome n'échangeant pas de données avec les autres ; elle est toujours rangée dans la même zone mémoire, en écrasant le module précédent.
Cette logique est très utilisée en Basic, alors qu'avec d'autres langages ou sur des machines plus puissantes, c'est le système d'exploitation qui gère la participation des programmes d'une façon parfaitement transparente à l'utilisateur. Il est néanmoins recommandé de diviser le programme en procédures bien définies et axées sur la résolution d'un seul aspect d'un problème. Il en découle deux avantages :

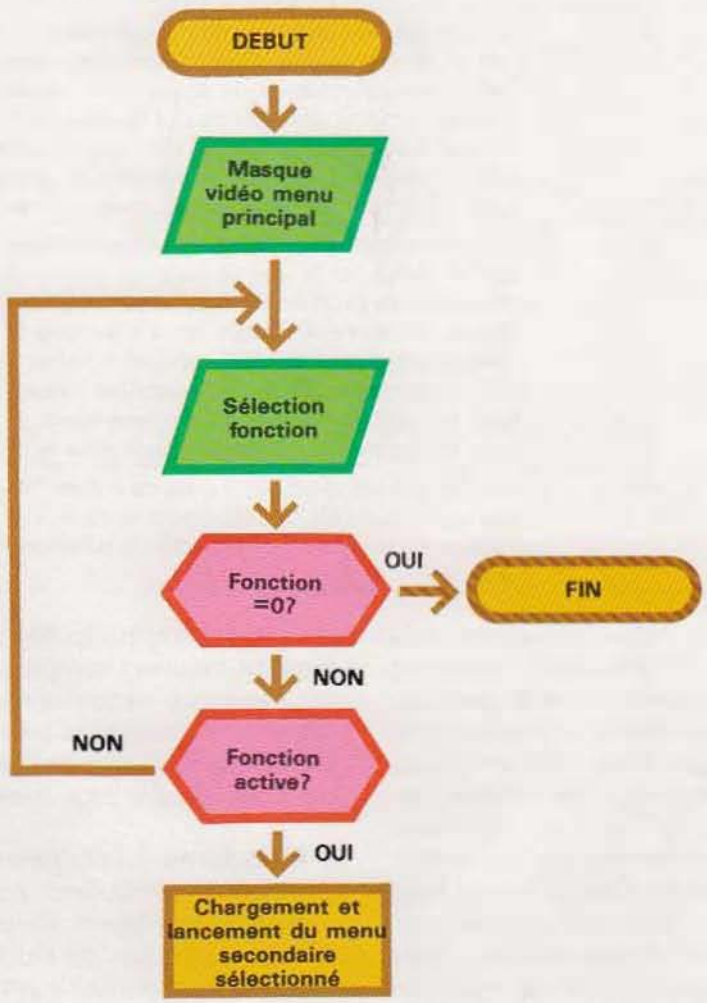
- **Modularité**. Le programme s'adapte à des situations particulières par suppression ou ajout de procédures. On peut, par exemple, retirer la gestion du stock (ou une de ses parties) en éliminant la procédure correspondante.

- **Facilité de manipulation et de mise au point**. Il est plus aisé de repérer une éventuelle erreur, de l'isoler et de la corriger. De nouvelles applications sont lancées par simple remplacement ou modification d'une procédure, sans interaction avec les autres.

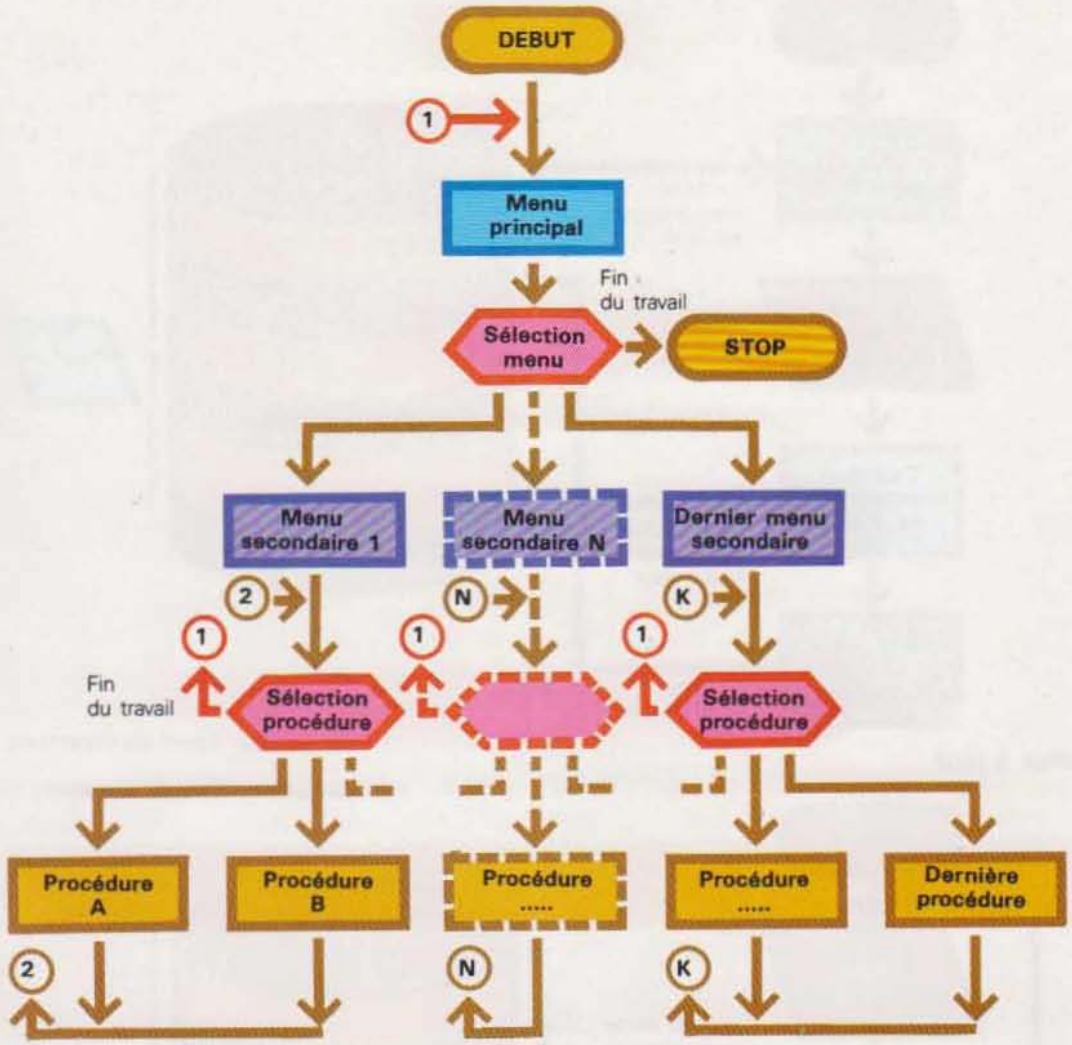
Dans des systèmes moins évolués, cette technique oblige à recopier tous les sous-programmes utilisés par plusieurs procédures (à moins d'adopter des systèmes de gestion compliquée et incombant totalement à l'utilisateur). Avec des systèmes d'exploitation plus avancés, en revanche, on n'a recours qu'à un module commun, son chargement et ses liens avec les autres parties de la procédure étant gérés par le système et ne demandant aucune intervention.

En pratique, la reproduction des sous-programmes se traduit seulement par une plus grande occupation de l'espace mémoire, en fait limitée à quelques kilo-octets (Ko). Néanmoins, dans certains cas, l'économie d'espace, même limitée, constitue une démarche avantageuse. Avec les disquettes de 5 pouces 1/4, par exemple, dont la capacité en simple densité est inférieure à 160 Ko, les programmes occupent jusqu'à 20 % de l'espace. Dans ce cas, il existe des techniques de segmentation fondées sur le transfert dans des zones de mémoire bien définies pour permettre l'utilisation de certaines parties selon plusieurs procédures.

ORGANIGRAMME DU MENU PRINCIPAL



SCHEMA DE L'ENCHAINEMENT
MENU PRINCIPAL / MENU SECONDAIRE / PROCEDURE



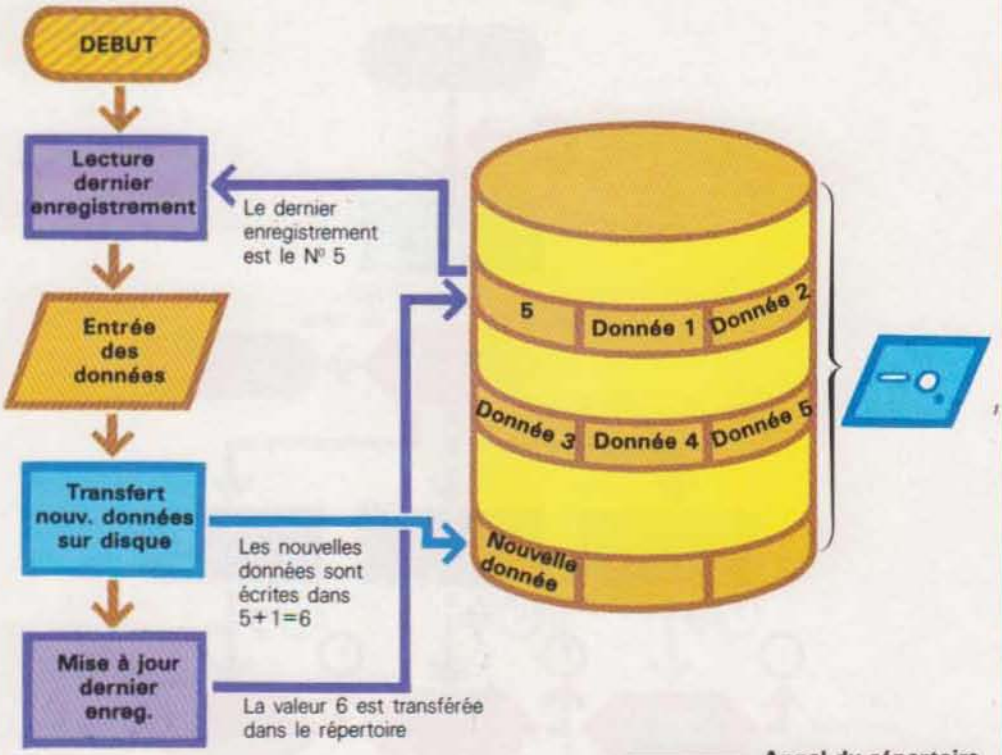
Gestion des fichiers

Quel que soit le problème que l'on doive résoudre dans un programme, il faut gérer des fichiers des données. Les fonctions requises sont, en général, l'introduction, la mise à jour et l'impression. Les méthodes utilisées sont illustrées page 1278, avec des organigrammes d'exemples de gestion de fichiers (clients et fournisseurs) page 1279, 1280 et 1281. Les seules modifications à apporter, pour d'autres fichiers, concernent le nom et le format des enregistrements.

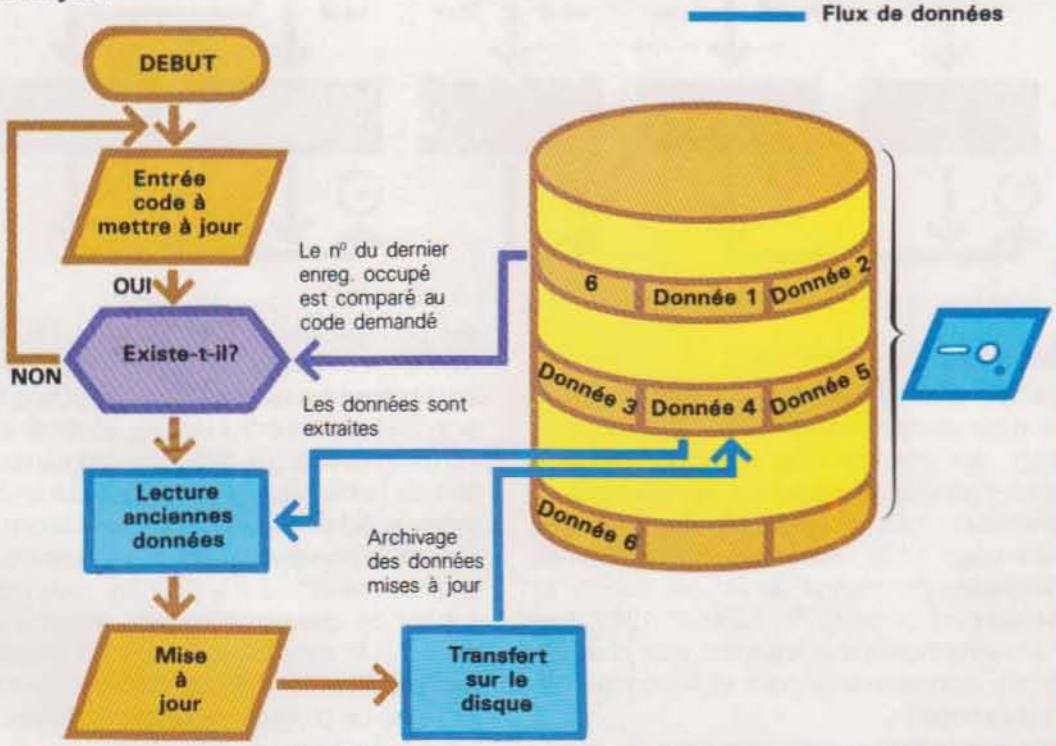
Les organigrammes sont utiles pour les langages peu spécialisés. Avec le Cobol, l'accès au fichier est si simple qu'aucune indication n'est nécessaire (voir exemple p. 1279). Dans la version Basic, on a introduit une généralisation à l'organigramme. En début de programme, un test de l'indicateur est effectué pour en déterminer le fichier (Clients ou Fournisseurs). Leur format d'enregistrement étant identique, le programme est valable pour les deux fichiers ; il suffit de changer le nom du fichier à sélectionner. Un autre paramétrage est possible en prenant le format d'enregistrement comme paramètre. Un programme unique suffit pour toutes les applications.

SCHEMAS D'ENTREE ET DE MISE A JOUR DES DONNEES

Entrée

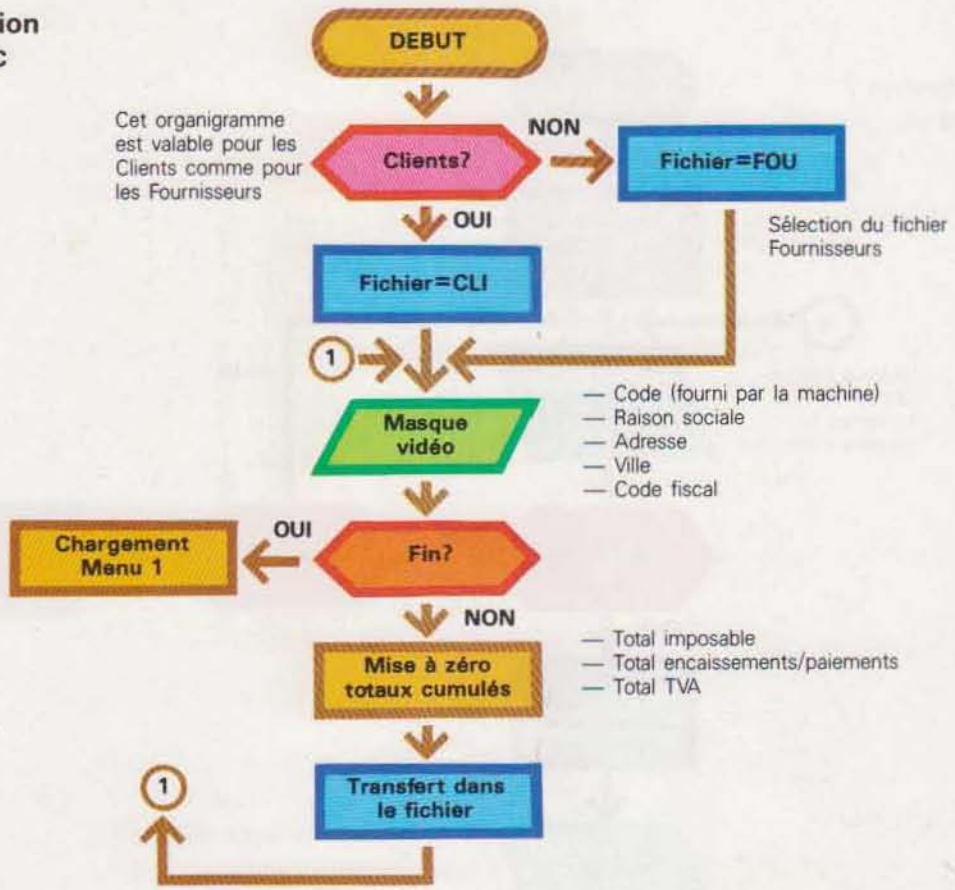


Mise à jour

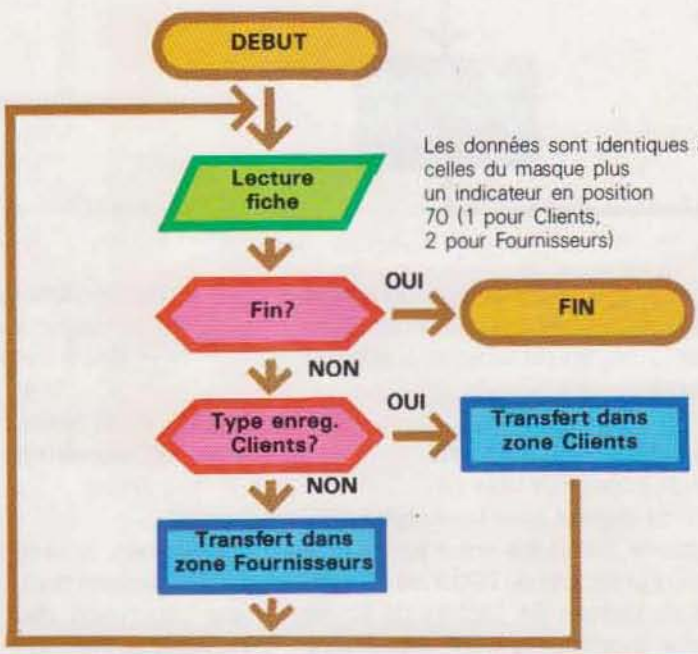


ENTREE DE NOUVEAUX CLIENTS OU FOURNISSEURS

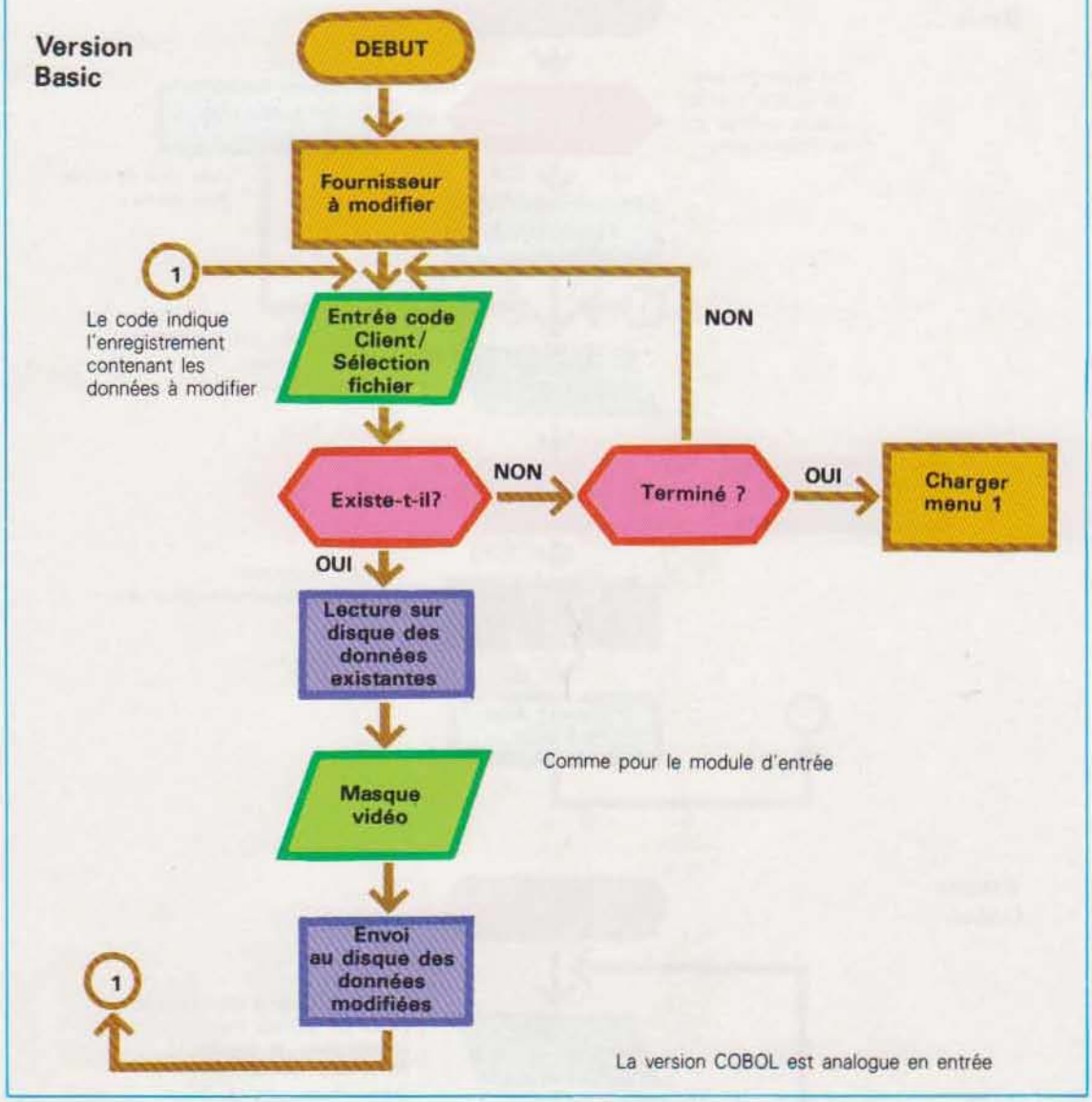
Version Basic



Version Cobol



MODIFICATION DES FICHIERS CLIENTS OU FOURNISSEURS

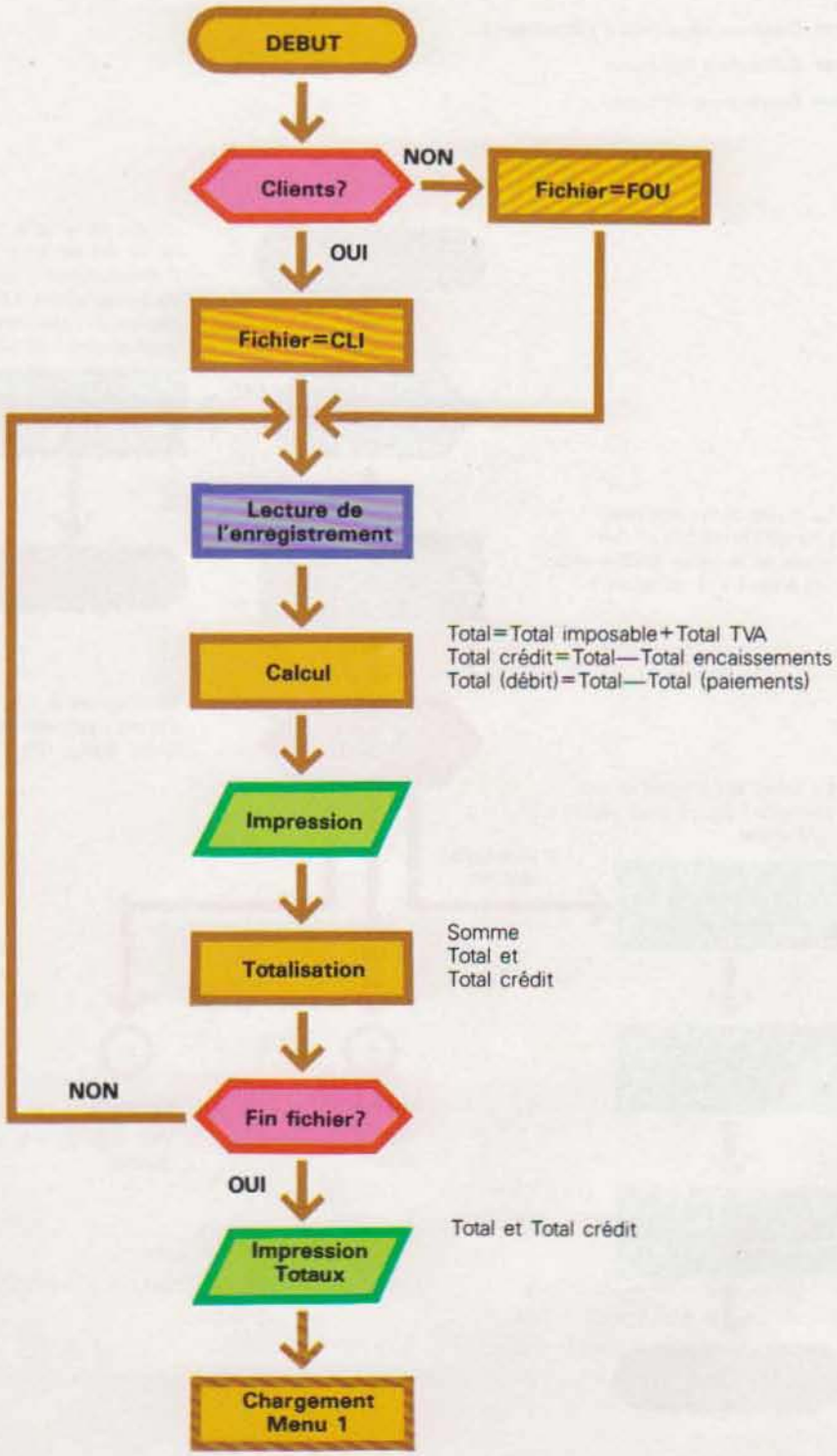


Dans l'organigramme d'un sous-programme général de lecture ou d'écriture d'un enregistrement (page 1282), les paramètres à affecter sont les suivants :

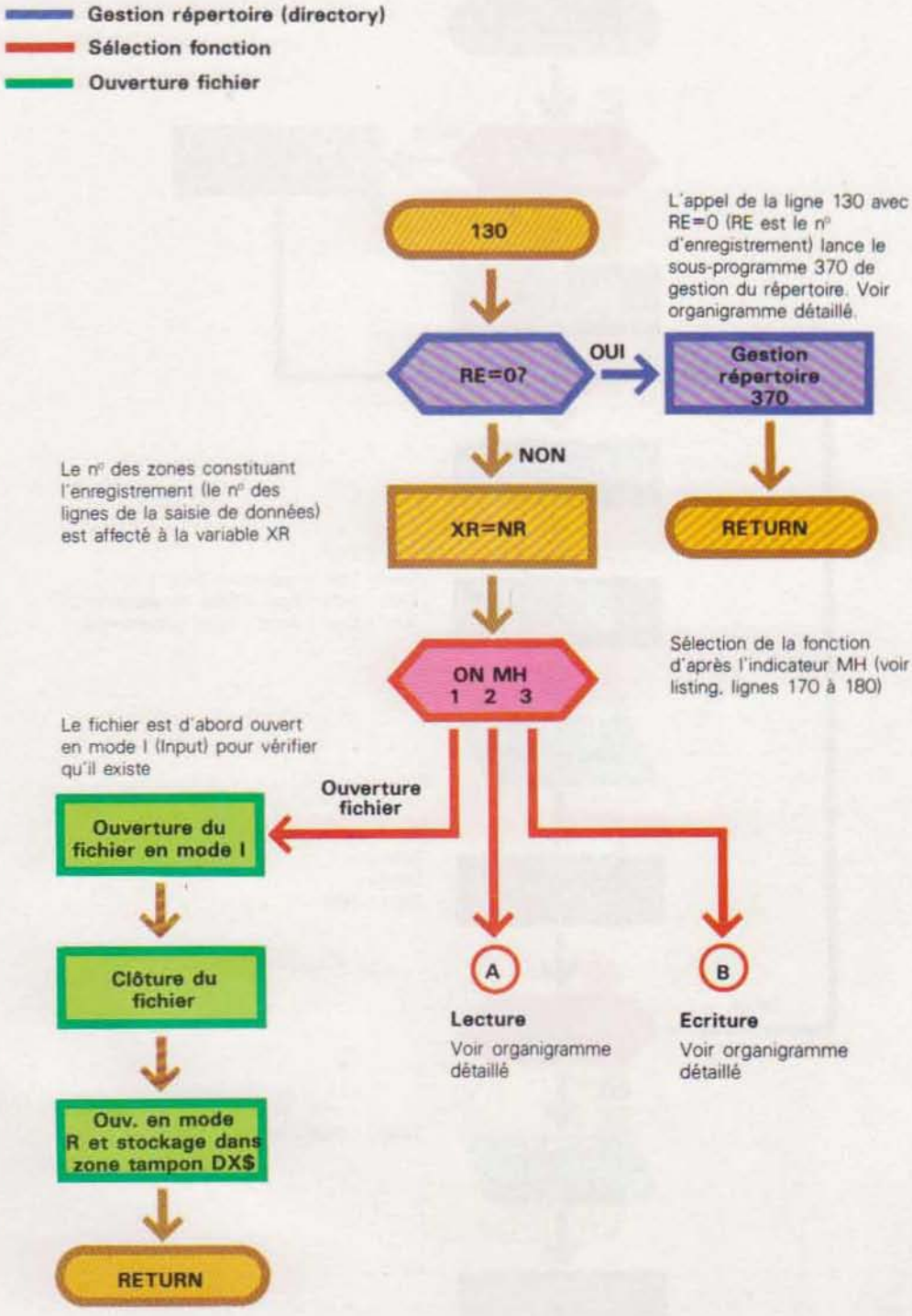
— Nom du fichier (variable NM$),
— Nombre de zones (variable NR),
— Longueur de chaque zone (en octets) définie comme la différence entre les valeurs du tableau PD (octets de début de chaque zone) et du tableau PA (octets de fin de zones). Par exemple, si pour la première zone des données, PD(1) = 1 et PA(1) = 30, la longueur en octets de la zone est :
N = PA(1) - PD(1) + 1
= 30 - 1 + 1
= 30 octets
— RE, numéro de l'enregistrement à lire ou à écrire.

En réponse, le programme stocke le contenu de l'enregistrement dans le tableau BF$, d'où une séparation des zones. Exemple, BF$(1) contient la première zone, BF$(2) contient la

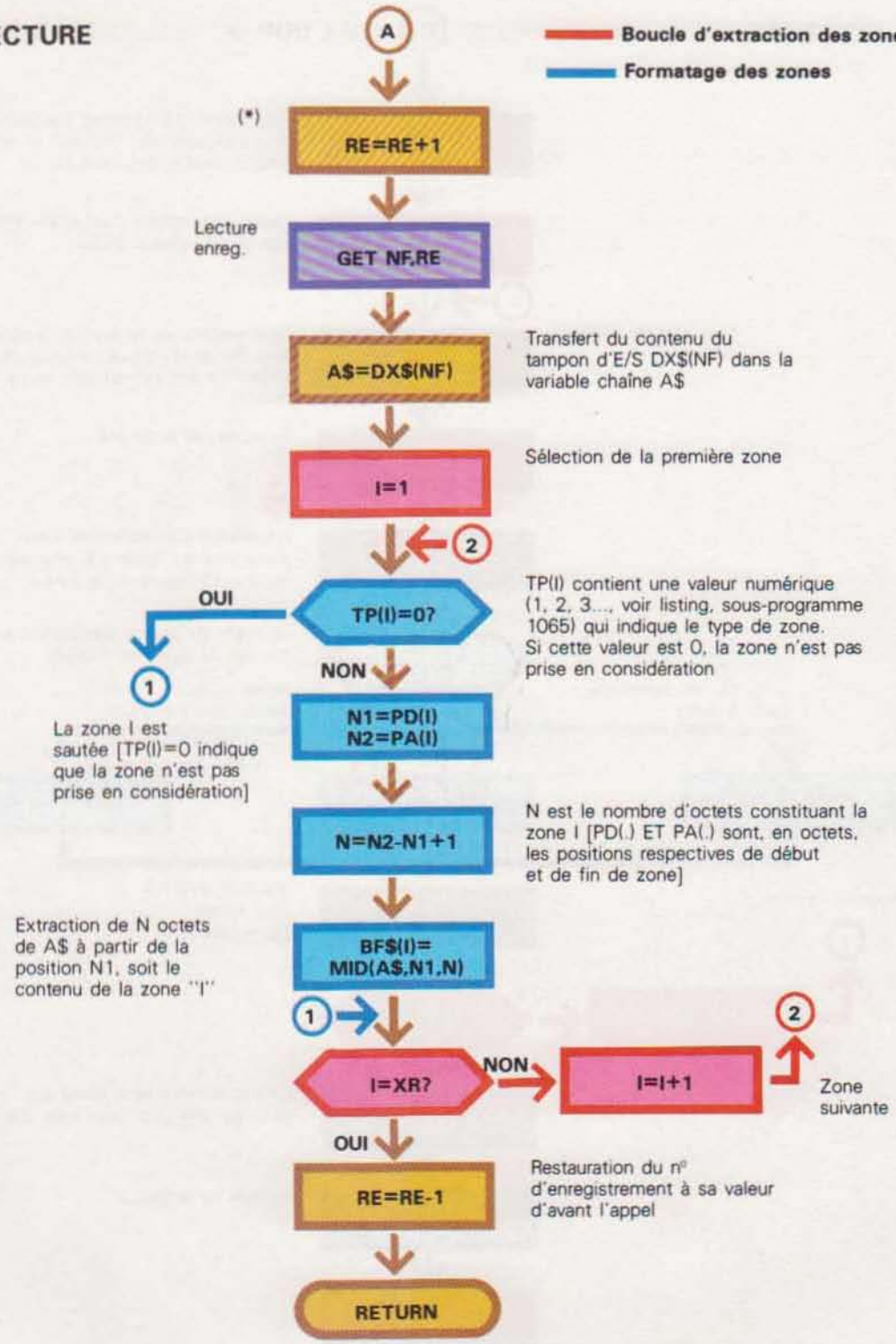
SITUATION CLIENTS/FOURNISSEURS



SOUS-PROGRAMME DE LECTURE/ECRITURE SUR DISQUE PARAMETRE...

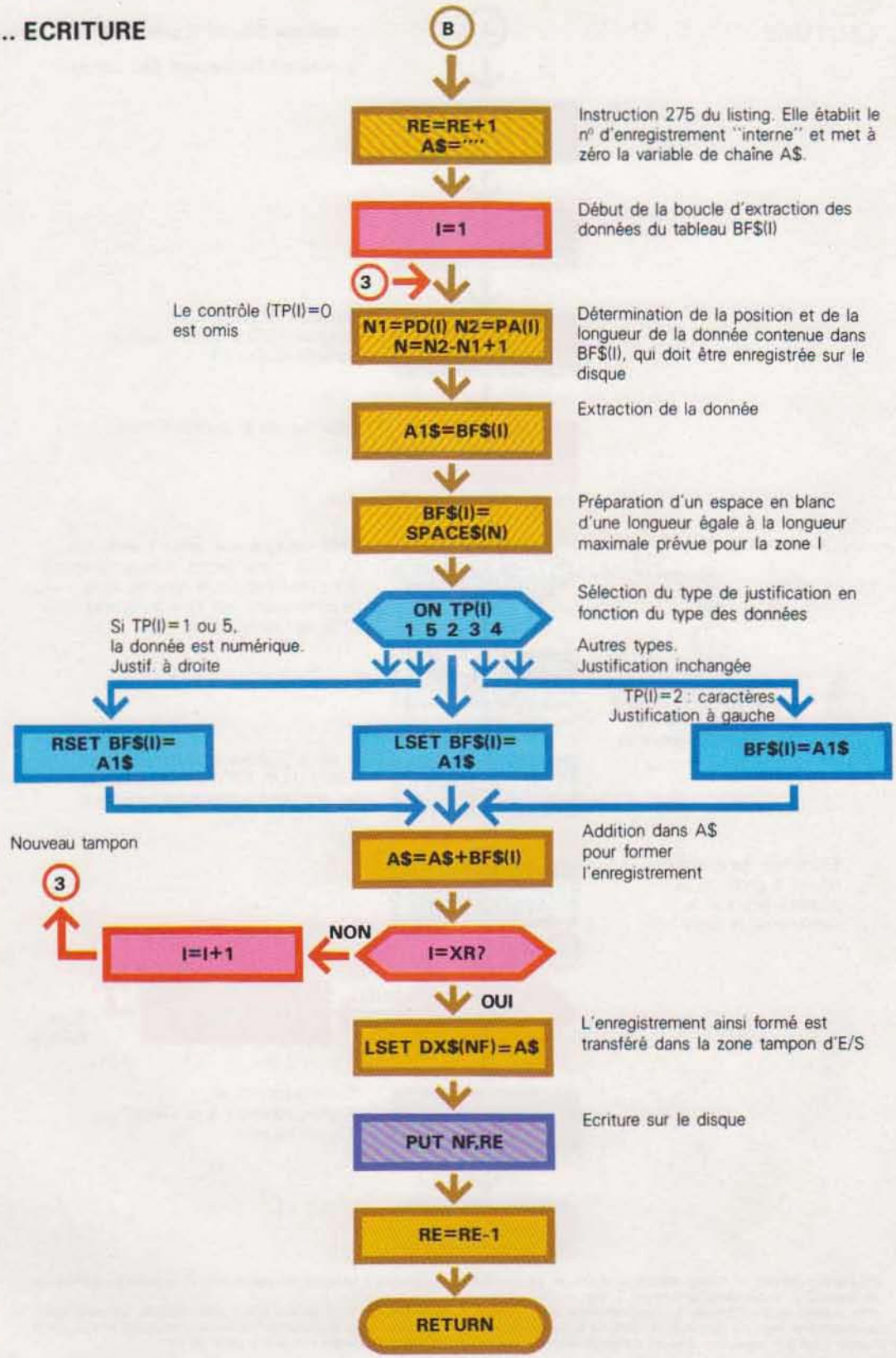


... LECTURE

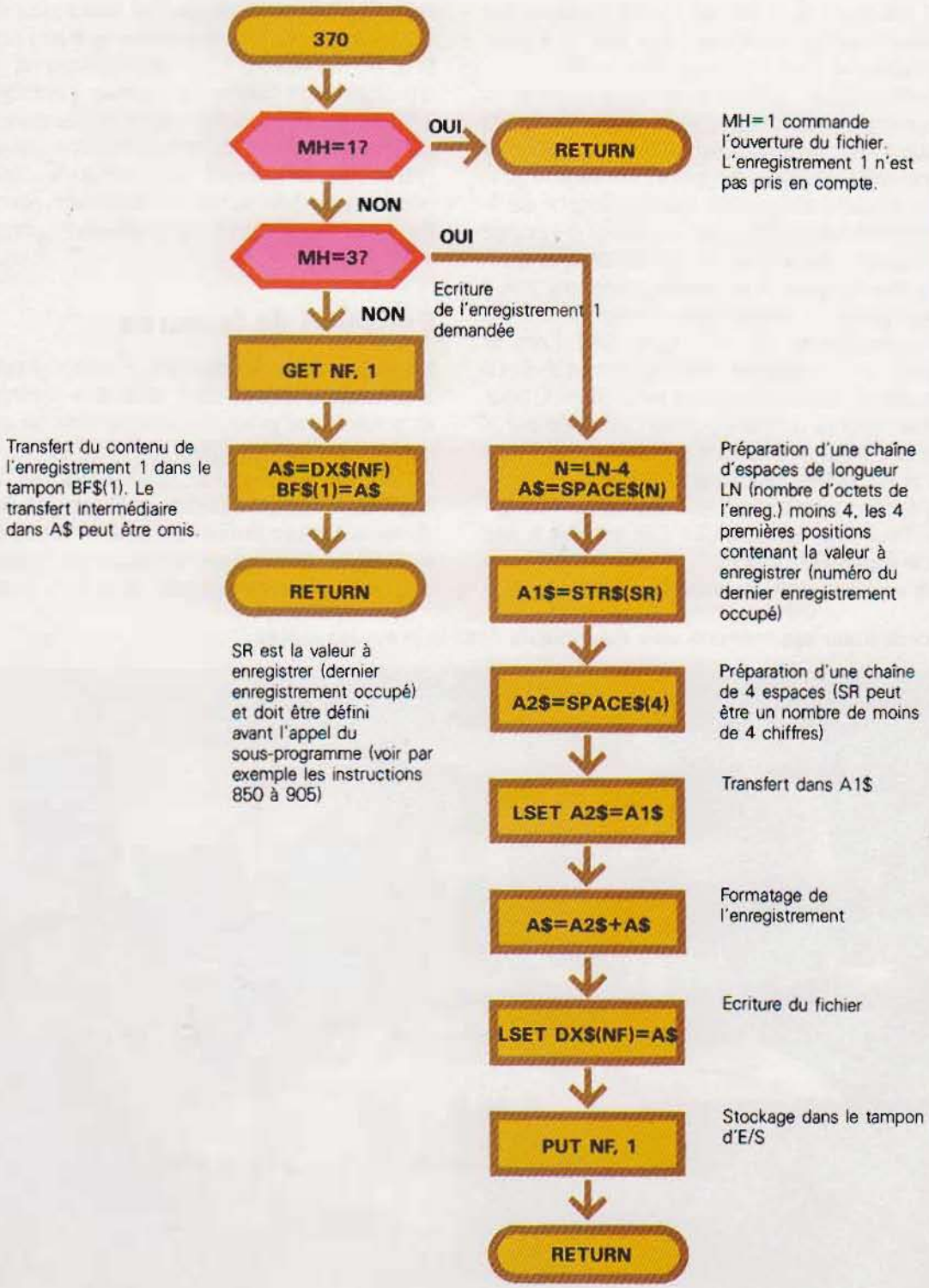


(*) L'enregistrement n° 1 étant affecté au répertoire, les données sont stockées à partir de l'enregistrement 2 : la première donnée est en position 2, la deuxième en position 3, etc.
Pour annuler cette différence, le sous-programme écrit les données en position +1 par rapport à leur ordre d'entrée. De cette façon, le programme appelant demande l'écriture de la première donnée en position 1 et le sous-programme le transfère en position 2. Cette astuce n'est pas nécessaire avec les systèmes d'exploitation qui numérotent les enregistrements à partir de zéro.

... ECRITURE


B
RE=RE+1
A$=""
Instruction 275 du listing. Elle établit le n° d'enregistrement "interne" et met à zéro la variable de chaîne A$.
I=1
Début de la boucle d'extraction des données du tableau BF$(I)
3
Le contrôle (TP(I)=0 est omis
N1=PD(I) N2=PA(I) N=N2-N1+1
Détermination de la position et de la longueur de la donnée contenue dans BF$(I), qui doit être enregistrée sur le disque
A1$=BF$(I)
Extraction de la donnée
BF$(I)= SPACE$(N)
Préparation d'un espace en blanc d'une longueur égale à la longueur maximale prévue pour la zone I
ON TP(I) 1 5 2 3 4
Sélection du type de justification en fonction du type des données
Si TP(I)=1 ou 5, la donnée est numérique. Justif. à droite
Autres types. Justification inchangée
TP(I)=2 : caractères Justification à gauche
RSET BF$(I)= A1$
LSET BF$(I)= A1$
BF$(I)=A1$
Nouveau tampon
3
A$=A$+BF$(I)
Addition dans A$ pour former l'enregistrement
I=I+1
NON
I=XR?
OUI
LSET DX$(NF)=A$
L'enregistrement ainsi formé est transféré dans la zone tampon d'E/S
PUT NF,RE
Ecriture sur le disque
RE=RE-1
RETURN

... GESTION DU REPERTOIRE (DIRECTORY)



deuxième, et ainsi de suite. Lors de l'écriture sur disque, le tableau BF$ doit avoir déjà reçu les données à transférer, les autres paramètres restant inchangés.
La sélection de la lecture ou de l'écriture est déterminée par l'indicateur MH (MH = 1 pour la lecture et MH = 2 pour l'écriture).
Pages 1287 à 1291, on a reproduit le listing du sous-programme, dans sa version Basic 80 sous CP/M (instructions de la ligne 130 à la ligne 465). Ce sous-programme prévoit la gestion séparée du premier enregistrement du fichier DIR (répertoire) dans lequel est mémorisé le numéro du dernier enregistrement occupé par des données. Pour activer cette fonction, il faut appeler le sous-programme en affectant 0 à la variable RE (RE = 0, ligne 185). Lors de l'écriture de nouvelles données, il faut d'abord appeler le sous-programme avec RE = 0 pour déterminer la dernière position occupée par le fichier ; les données sont ensuite transférées à la position (enregistrement) obtenue en ajoutant 1 (incrémentation) au contenu du répertoire. Pour finir, le répertoire DIR est mis à jour (voir lignes 800 à 905).
Les versions, pour un environnement autre que le CP/M, ne varient qu'au niveau des instructions de gestion du disque. Il suffit de remplacer ces instructions pour traduire le programme dans d'autres versions Basic. La seule limitation est liée au chaînage des valeurs numériques, ce qui entraîne une conversion des nombres en chaînes au cours de l'écriture et une transformation inverse (de chaînes à nombres) à la lecture. Ces fonctions ne sont pas prévues dans le listing ci-contre et doivent être activées respectivement avant et après le sous-programme. Les ajouts se réduisent donc à l'emploi d'instructions de conversion simples VAL, STR$...

Emission de factures

L'émission d'une facture est un acte régi par la législation fiscale. Elle doit donc être contrôlée et guidée avec précision (voir schéma de principe de factures p. 1292).
Le premier concerne la date. Le fichier DIR (répertoire) contient la date et le numéro de la dernière facture émise. Lorsque la procédure est activée, le programme fournit les numéros de factures ordonnés afin que l'on puisse

L'ordinateur apporte une aide inestimable dans la prévision météo.

Marie-Leo de Wys

PROCEDURE DE FACTURATION ET DE GESTION DES ARCHIVES

```
10 OPTION BASE 1
15 '    FILE = BASIC
20 '
25 '    VERSION 23-07-84 : CP/M - BASIC 80
30 DEFINT A-V
35 '    Les variables utilisées pour les calculs doivent commencer par Z
40 DEFDBL Z
45 DIM TF(20)                          ' touches de fonction
50 FOR I=1 TO 20 : READ TF(I) : NEXT I
55 DATA 10!,8!,12!,11!,16!,17!,13!,22!,18!,0!,19!,20!,21!,0!,0!,0!,0!,0!,0!,0!
60 DIM DX$(4)
65 DIM TP(20),PA(20),PD(20),LD(20)
70 DIM D$(20),B$(20),BF$(20),AT$(20),BR$(20),CN$(20)
75 DIM IS(5000)
80 '
85 '
90 '
95 '    *  FONCTIONS  *
100 '
105 BI$=CHR$(27)+"+"+CHR$(7)     ' Nettoyage de l'écran
110 PRINT BI$
115 '
120 GOSUB 470                    ' Menu des fonctions prévues
125 STOP
130 '
135 '   ** GESTION DONNEES **
140 '   Fichier : GDAT controlé le 23-07-84
145 '   Enregistrement = RE type enregistrement = TR BF$(20)
150 '   Tampon de 1 à 4 DX$(4) = 4 fichiers
155 '   NF = numéro de fichier = numéro de tampon
160 '   NR = numéro de ligne
165 '   NH =
170 '         1 ouverture
175 '         2 lecture
180 '         3 écriture
185 IF RE=0 THEN GOSUB 370:RETURN  'RE=0 pour gérer le répertoire du fichier
190 XR=NR                          'nombre de zones (1310)
195 ON MH GOTO 200,220,275
200 '   ouverture
205 OPEN "I",NF,NM$  'ouverture en mode I pour vérifier l'existence du fichier
210 CLOSE NF:OPEN "r",NF,NM$,LN:FIELD NF,LN,AS,DX$(NF):RETURN
215 'la ligne 210 ouvre en mode R et stocke les données dans la zone tampon
220 '   lecture
225 RE=RE+1            ' voir l'organigramme
230 GET NF,RE:A$=DX$(NF)   'lecture et transfert des données dans A$
235 '  * création des tampons BF$(.) en fonction des longueurs des
240 '    zones définies dans le sous-programme 1310
245 FOR I=1 TO XR:IF TP(I)=0 GOTO 255
250 N1=PD(I):N=N2-N1+1:BF$(I)=MID$(A$,N1,N)
255 NEXT I
260 RE=RE-1             ' voir l'organigramme
265 RETURN
270 '
275 '    écriture
280 RE=RE+1
285 A$=""
290 FOR I=1 TO XR:IF TP(I)=0 GOTO 345  'sélection d'une des zones
295 N1=PD(I):N2=PA(I):N=N2-N1+1        'calcul de la longueur en octets
300 A1$=BF$(I)                         'sauvegarde du contenu
305 BF$(I)=SPACE$(N)                   'préparation de l'espace nécessaire
310 '                                   d'après la longueur de la zone
315 ON TP(I) GOTO 320,325,330,330,320  'sélection d'après le type
320 RSET BF$(I)=A1$:GOTO 335           'numérique à droite!
325 LSET BF$(I)=A1$:GOTO 335           'caractères à gauche
```

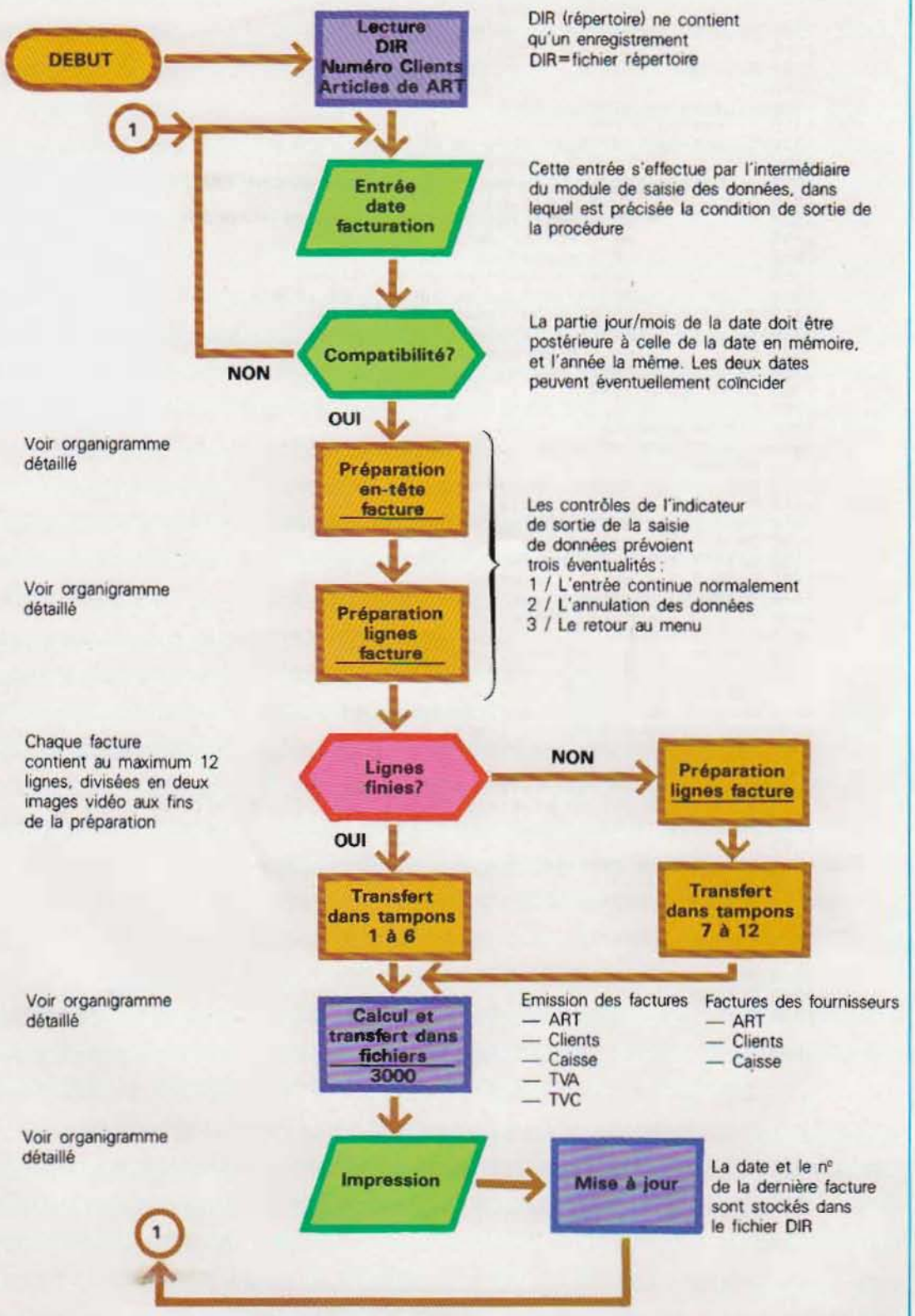
```
330 BF$(I)=A1$                          'autres types inchangés
335 A$=A$+BF$(I)                        'additionne les tampons pour former
340 '                                    une seule chaine
345 NEXT I
350 LSET DX$(NF)=A$                     'transfert dans le tampon d'E/S
355 PUT NF,RE                           'écriture de l'enreg RE dans le fic NF
360 RE=RE-1
365 RETURN
370 'gestion enregistrement 1
375 '
380 IF MH=1 THEN RETURN                 'MH=1 pour l'écriture du fichier
385 IF MH=3 GOTO 410                    'branchement sur l'écriture de l'enreg 1
390 'lecture
395 GET NF,1                            'lecture de l'enreg 1
400 A$=DX$(NF):BF$(I)=A$                'transfert dans la zone tampon 1
405 RETURN
410 'écriture
415 N=NL-4:A$=SPACE$(n)                 'préparation d'une chaine vide de
420 '                                    longueur LN-4
425 A1$=STR$(SR)                        'conversion du nombre SR
430 '                                    en caractères
435 A2$=SPACE$(4):LSET A2$=A1$          'transfert de cette variable dans
440 '                                    une zone de longueur 4
445 A$=A2$+A$                           'concaténation de la chaine vide (415)
450 '                                    pour avoir un enreg de longueur LN
455 LSET DX$(NF)=A$                     'transfert dans la zone tampon d'E/S
460 PUT NF,1                            'écriture fichier NF enreg = 1
465 RETURN
470 '
475 '  MENU1
480 '
485 '  VERSION 27-07-84
490 '  les touches de fonction sont créées par le menu principal - fichier MENU
495 GOSUB 575
500 ON V GOTO 505,510,525,525,525,525,525,525,525,525
505 GOSUB 795:GOTO 485
510 GOSUB 910:GOTO 485
515 '  les rubriques actives de MENU sont la 1 et la 2
520 '  les autres sont ignorées
525 GOTO 485
530 PRINT BI$
535 CK=5:X=30:Y=10:GOSUB 755
540 PRINT CHR$(27)+"G1"
545 PRINT CHR$(27)+"G0"
550 RETURN
555 END
560 '-----------------------------
565 ' * sélection de la fonction *
570 '-----------------------------
575 PRINT BI$
580 CK=5:X=2:Y=1:GOSUB 755:PRINT "E.G.S Gestion archives clients"
585 CK=5:X=15:Y=3:GOSUB 755             'le curseur est positionné en X,Y
590 PRINT "1- ENTREE"
595 Y=Y+1:GOSUB 755
600 PRINT "2- VARIATION"
605 Y=Y+1:GOSUB 755
610 PRINT "3- ............"
615 Y=Y+1:GOSUB 755
620 PRINT "4- ............"
625 Y=Y+1:GOSUB 755
630 PRINT "5- ............"
635 Y=Y+1:GOSUB 755
640 PRINT "6- ............"
645 Y=Y+1:GOSUB 755
```

```
650 PRINT "7- ............."
655 Y=Y+1:GOSUB 755
660 PRINT "8- ............."
665 Y=Y+1:GOSUB 755
670 PRINT "9- ............."
675 Y=Y+1:GOSUB 755
680 PRINT "0- FIN TRAVAIL ."
685 X=1:Y=Y+2:GOSUB 755
690 PRINT "entrez l'option choisie"
695 X=26:GOSUB 755:A$=INPUT$(1):PRINT A$
700 V=VAL(A$)
705 IF V>9 GOTO 730
710 PRINT BI$
715 IF V=0 THEN V=10
720 RETURN
725 '  ** erreur **
730 PRINT CHR$(27),CHR$(7)
735 X=42:GOSUB 755
740 PRINT "non valide"
745 GOTO 695
750 '
755 '  déplacement curseur
760 '  CURS-X+Y
765 '  l'instruction suivante dépend du type de masque utilisé
770 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
775 RETURN
780 ' ** FIN DU MENU **
785 ' *
790 ' *
795 ' *** ENTREE ***
800 ' ouverture
805 RE=1
810 NF=1:GOSUB 1305              'préparation du format
815 MH=1:GOSUB 130               'ouverture du fichier
820 'lecture du dernier enregistrement occupé
825 RE=0:MH=2
830 GOSUB 130
835 RE=VAL(BF$(1))               'RE=1er enreg libre + 1
840 X=1:Y=1:GOSUB 1280           'positionnement du curseur
845 PRINT "code client : ";RE    'affichage du n° de code
850 '                            (enreg) du prochain client
855 GOSUB 1305                   'préparation du format
860 GOSUB 1065                   'présentation à l'écran
865 GOSUB 1250                   'saisie des données
870 ON F1 GOTO 875,880,840
875 RETURN                       'retour : les données sont annulées
880 MH=3:GOSUB 130               'entrée nouveau client
885 SR=RE
890 RE=0:GOSUB 130               'mise à jour du répertoire (enreg 1)
895 GOTO 820                     'nouvelle entrée
900 CLOSE NF
905 RETURN
910 ' variations
915 RE=1
920 NF=1:GOSUB 1305              'préparation du format
925 MH=1:GOSUB 130               'ouverture du fichier
930 RE=0:MH=2:GOSUB 130          'lecture du répertoire
935 MX=VAL(BF$(1))               'MX=dernier code client utilisé
940 X=1:Y=1:GOSUB 1280
945 PRINT "code client : "
950 KC=20:MD=0:KR=1:TP(1)=1      '*** préparation du format pour entrée
955                              du code client à modifier
960 NF=1:PD(1)=1:PA(1)=3:LN=3:LD(1)=3
965 GOSUB 1250                   'saisie des données pour la lecture du code
```

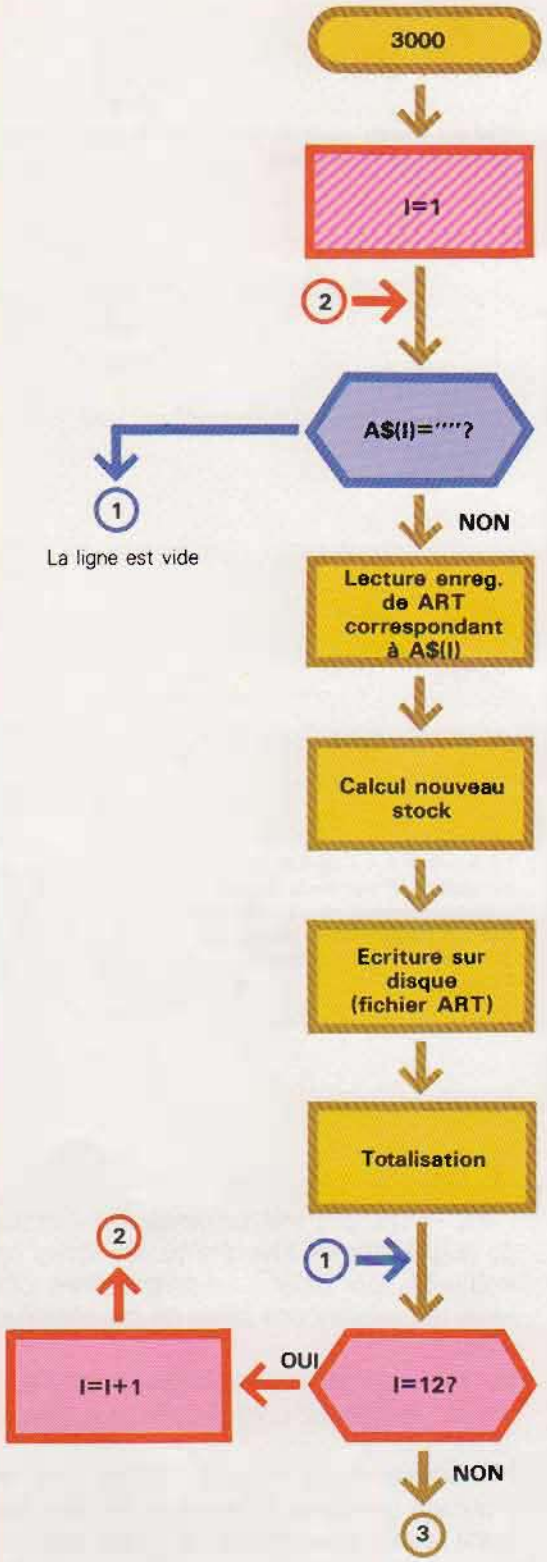
```
970 ON F1 GOTO 975,980,940
975 RETURN                          'retour menu
980 V=VAL(BF$(1))                   'entrée
985 IF V>MX THEN PRINT "ERREUR":GOTO 940 'controle qu'il n'a pas été demandé
990 '                               un code plus grand que le dernier mis en
995 '                               mémoire
1000 NF=1:GOSUB 1305
1005 RE=V
1010 MH=2:GOSUB 130
1015 FOR I=1 TO NR:TP(I)=4:NEXT I 'transforme toutes les zones affichées
1020 GOSUB 1065                     'affichage à l'écran
2025 GOSUB 1250                     'saisie des modifications
1030 ON F1 GOTO 1035,1040,940
1035 RETURN
1040 MH=3:GOSUB 130
1045 GOTO 940
1050 '
1055 RETURN
1060 '
1065 '
1070 ' **** FORMATS VIDEO ****
1075 '
1080 ' version 24-07-84
1085 '
1090 ' TYPE =
1095 '          1-numérique
1100 '          2-alphanumérique
1105 '          3-affichage seulement (entrée validée)
1110 '          4-affichage avec entrée
1115 '          5-zone résultat
1120 ' ** entrée NF = numéro du fichier **
1125 MD=0
1130 '  calcul de la longueur maximale des descriptions
1135 FOR I=1 TO NR
1140 IF LEN(D$(I))>MD THEN MD=LEN(D$(I))
1145 NEXT I
1150 '  préparation des zones tampons
1155 FOR I=1 TO NR:N=MD-LEN(D$(I))
1160 IF N=0 THEN 1170
1165 D$(I)=D$(I)+SPACE$(N)
1170 NEXT I
1175 FOR I=1 TO NR:IF TP(I)=0 THEN 1200
1180 IF TP(I)=3 OR TP(I)=4 THEN D$(I)=D$(I)+" "+BF$(I):GOTO 1200
1185 N=PA(I)-PD(I)+1
1190 D$(I)=D$(I)+" "+STRING$(N,46)
1195 BF$(I)=SPACE$(N)
1200 NEXT I
1205 '
1210 '  affichage à l'écran
1215 X=KC:Y=KR
1220 FOR I=1 TO NR
1225 GOSUB 1280
1230 PRINT D$(I)
1235 Y=Y+1
1240 NEXT I
1245 RETURN
1250 '
1255 '  ***** saisie des données *****
1260 '  -------------------------------
1265 '  * idem à celle déjà présentée *
1270 '  -------------------------------
1275 '
1280 '  déplacement du curseur
1285 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
```

```
1290 RETURN
1295 '
1300 RETURN
1305 '
1310 '  data version 23-07-84
1315 '  *** DATA ***
1320 '  entrée : NF=type d'enreg et n° du fichier
1325 '
1330 '  sortie :
1335 '           KC = position première colonne
1340 '           KR = position première ligne de l'écran
1345 '           NR = nombre de lignes
1350 '           NF = numéro du fichier
1355 '           D$(*) = description des lignes
1360 '           TP$(*) = type de zones - cf 1070 -
1365 '           PD(*) = pointeur début de zone
1370 '           PA(*) = pointeur fin de zone
1375 '           LN = nombre de caractères
1380 '           NM$ = nom du fichier
1385 ON NF GOTO 1400
1390 '
1395 'TR=2: données
1400 KC=2:KR=3:NR=8:N=NR
1405 RESTORE 1415
1410 FOR I=1 TO N:READ D$(I):NEXT I
1415 DATA " 1 - raison sociale"
1420 DATA " 2 - adresse"
1425 DATA " 3 - ville"
1430 DATA " 4 - registre TVA"
1435 DATA " 5 - total factures"
1440 DATA " 6 - total payes"
1445 DATA " 7 - total solde"
1450 DATA " 8 - ..............."
1455 FOR I=1 TO N:READ TP(I):NEXT I
1460 DATA 2!,2!,2!,1!,5!,5!,5!,2!
1465 FOR I=1 TO N:READ PD(I):NEXT I
1470 DATA 1!,31!,61!,76!,87!,98!,109!,120!
1475 FOR I=1 TO N:READ PA(I):NEXT I
1480 DATA 30!,60!,75!,86!,97!,108!,119!,128!
1485 LN=PA(N):NM$="A:FORN"
1490 FOR I=1 TO N:LD(I)=PA(I)-PD(I)+1:NEXT I
1495 N=NR+1:FOR I=1 TO N:D$(I)=" ":TP(I)=0:PD(I)=0:PA(I)=0:NEXT I
1500 RETURN
1505 '
1510 '  *** zone résultat ***
1515 Z5=VAL(BF$(5))
1520 Z6=VAL(BF$(6))
1525 Z7=Z5-Z6
1530 BF$(7)=SPACE$(11)
1535 A2$=STR$(Z7)
1540 RSET BF$(7)=A2$
1545 RETURN
1550 '
```

EMISSION DES FACTURES



SOUS-PROGRAMME DE CALCUL ET DE TRANSFERT DANS LES FICHIERS

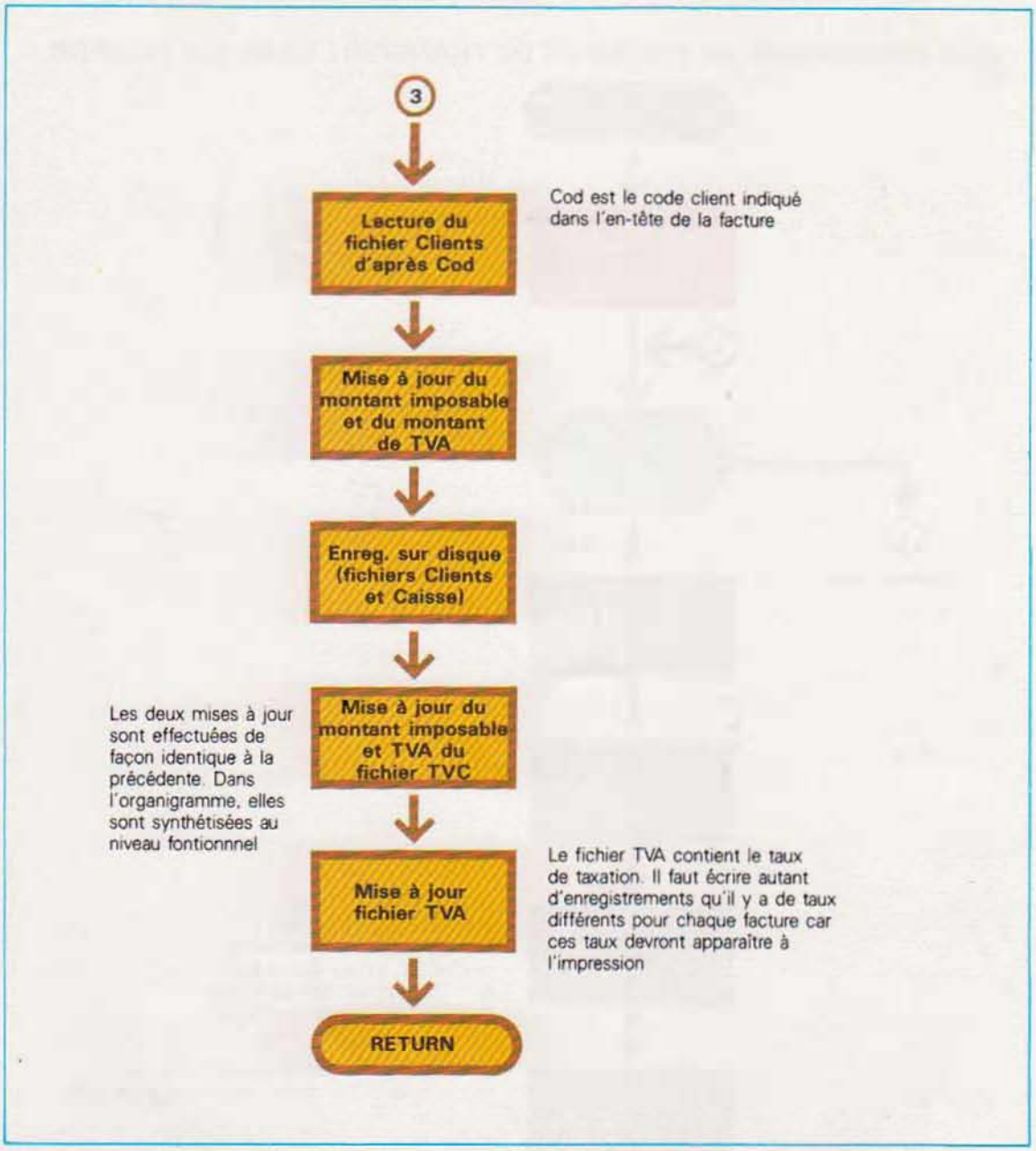


L'enregistrement de ART a déjà été lu lors de l'entrée des lignes (voir organigramme correspondant)

La mise à jour du stock n'est effectuée qu'une fois la facture établie afin de pouvoir modifier ou annuler chaque ligne sans devoir prévoir des procédures de correction du stock complexes.

Totalisation des montants imposables et des montants de la TVA divisés par les taux de taxation

Le traitement des lignes d'une facture est terminé



contrôler la compabilité des dates : la date entrée à la console doit être postérieure ou égale à la dernière date enregistrée.
Ce type de contrôle est également prévu en versions Cobol, où les dates sont introduites par des fiches, ne serait-ce que pour vérifier formellement la perforation.
On suppose que la facture comporte deux parties : l'en-tête contenant les données d'identification du client et la série de 12 lignes destinées à recevoir les mouvements des stocks et de caisse.

Dans l'organigramme présenté, les fonctions de préparation de l'en-tête et des lignes sont exécutées par deux sous-programmes différents qui utilisent une saisie de données paramétrée.
La procédure de préparation de l'en-tête à suivre est décrite par l'organigramme de la page 1293.
Le nombre de lignes de la facture rend leur affichage simultané à l'écran impossible. Pour cette raison, une méthode de présentation par bloc de 6 lignes a été prévue (voir page 1292).

Les jeux vidéo (suite) ou les aventures de Fred

Nous avons vu, dans le numéro précédent, comment nous lier d'amitié avec Fred, notre si sympathique joueur de Gô électronique. Nous allons donc continuer à examiner sa logique. Analysons en détail le comportement de CARTE. Ce module se rappelle lui-même autant de fois que nécessaire, ce qui lui permet d'avancer pas à pas à l'intérieur du groupe de connexion et en en déterminant ainsi la ***structure*** *et les* ***limites****. Il est fondamental que CARTE réussise à reconnaître aussi les côtés extérieurs du Gô Ban pour deux raisons :*

— pour que Fred n'aille pas déposer des pions hors du Gô Ban à la suite d'une mauvaise signalisation,

— afin que Fred puisse bien gérer les "pions fantômes" qui revêtent une importance stratégique fondamentale dans les coups d'angle. Muni de tous ces renseignements, Fred peut formuler une ***réponse au coup*** *de son partenaire humain. Notons que, du moins au début, nous considérons Fred comme le plus faible des deux joueurs. A lui donc l'honneur — et l'avantage — de l'ouverture de la partie. La procédure par laquelle Fred accomplit des devoirs de joueur s'appelle JEU ; elle est composée de plusieurs routines. La première action de JEU consiste en le dépôt des pions de* ***handicap,*** *puisque Fred dispose de l'avantage. Le nombre des pions de handicap varie entre 1 et 9, en fonction du rapport de force existant entre les joueurs. Dans notre cas, nous concéderons — royalement n'est-ce-pas ? — 5 à 7 pions à Fred (au-delà, ce serait vraiment couper les verges pour se faire battre). Ceux-ci représentent plus de 10 % des pions mis à sa disposition. (Rappelons-nous que nous jouons avec un nombre réduit de pions.) Fred pourra ainsi acquérir une position stratégique tout à fait intéressante, surtout s'il choisit d'effectuer une ouverture d'angle.*

La structure du module JEU a été indiquée dans le numéro précédent, en page 1272. Il contient un appel aux sous-programmes EFFET COUP BLANC, EFFET COUP NOIR et MODULES, présentés en page 1295 et 1296.

A ce point du jeu, Fred peut décider d'exécuter une ***opération d'importance,*** *comme de* ***capturer un groupe de pions.*** *C'est aussi la*

DESCRIPTION DU MODULE EFFET COUP BLANC

```
EFFET COUP BLANC
  DEBUT
    POUR chaque intersection x avec un pion
    noir, EXECUTER
      DEBUT
        APPELER CARTE (noir, x)
        SI le groupe n'a pas de liberté
        ALORS prendre ses pions
        SINON
          DEBUT
            SI le groupe a une seule liberté
            ALORS choisir une liberté (qui ne
            tombe pas sur le bord de
            l'échiquier)
            SI le groupe a une ou deux libertés
            ALORS APPELER EVALUATION
            pour analyser la liberté choisie
          FIN
      FIN
  FIN
```

Ce modèle analyse l'effet du coup du Blanc avec ses conséquences sur le scénario du Gô Ban. Il met en évidence l'interrelation fonctionnelle entre les différents modules et les critères la déterminant.

DESCRIPTION DU MODULE EFFET COUP NOIR

```
EFFET COUP NOIR
  DEBUT
    POUR chaque intersection x avec un pion
    blanc, EXECUTER
      DEBUT
        APPELER CARTE (blanc, x)
        SI le groupe a une seule liberté
        ALORS
          DEBUT
            Poser le pion noir sur ce point
            Prendre le pion blanc
          FIN
        SINON
          SI le groupe a deux libertés ou plus
          ALORS
            DEBUT
              Choisir une liberté
              APPELER EVALUATION
              pour analyser la liberté choisie
            FIN
      FIN
  FIN
```

Comme dans le cas précédent, ce module analyse le coup d'un joueur. Fred évalue les intersections les plus intéressantes pour son coup suivant.

*première fois qu'intervient le critère d'**évaluation des choix** relatifs au coup à jouer, avec l'appel du module EVALUATION. Notons que le module CONTROLE, appelé par le sous-programme EVALUATION, a pour fonction d'éviter qu'une situation ayant déjà existé se reproduise sur le Gô Ban.*

*La procédure CARTE sera alors utilisée par Fred pour **chercher et analyser les libertés** d'un groupe de pions sur le Gô Ban. Quand un groupe de pions est connecté, il est crucial, pour l'adversaire, de savoir s'il est attaquable et quand, afin de le faire prisonnier... ou bien s'il est plus utile de prendre position ailleurs, soit afin de renforcer sa défense, soit en posant des jalons pour l'avenir.*

*Si, pour un joueur en chair et en os, **progresser** veut dire jouer un bon nombre de parties pour en analyser ensuite le déroulement, dans le cas de Fred, cela signifiera, cher partenaire humain, que vous aurez à cœur d'**étendre le programme** de gestion de Fred, pour en augmenter la souplesse, son état initial ne lui permettant d'"apprendre". L'implémentation de nouvelles procédures permettra ainsi à Fred de résoudre quelques problèmes de jeu plus complexes, comme les "yeux", la déclaration d'Atari, la règle du Ko... Il n'est pas difficile de réaliser ces procédures et de les rendre exécutables : il suffit de lire attentivement les règles du Gô afin de déterminer leur rôle dans la partie et d'établir ensuite le lien entre elles et le programme de gestion de l'ordinateur.*

Et maintenant, très bonne chance, cher partenaire humain. Vous voilà nanti d'un merveilleux compagnon, aussi patient qu'intelligent, et qui sera toujours "fidèlement vôtre". J.Q.

DESCRIPTION DU MODULE MODELES

```
MODELES
  DEBUT
    POUR chaque pion blanc, EXECUTER
      DEBUT
        SI il y a un modèle dans le tableau
        des exemples
        centré sur ce pion blanc
      ALORS
        DEBUT
          Etudier le coup suggéré x
          APPELER EVALUATION (x,2)
        FIN
    FIN
  FIN
```

Le comportement de Fred vise à établir des configurations de jeux qui lui soient favorables. Autrement dit, il doit non seulement appréhender la situation sur le Gô Ban mais aussi agir d'après une interprétation lui suggérant d'adopter une attitude offensive ou défensive. Ce principe est appliqué grâce au module MODELES.

DESCRIPTION DU MODULE EVALUATION

```
EVALUATION (coup, liberté)
  DEBUT
    APPELER OPTIMUS (meilleur coup, meil-
    leure liberté)
    SI liberté < = meilleure liberté, faire
      CONTROLER (coup) > = 2
    ALORS
      DEBUT
        Meilleur coup = coup
        Meilleure liberté = liberté
      FIN
  FIN
```

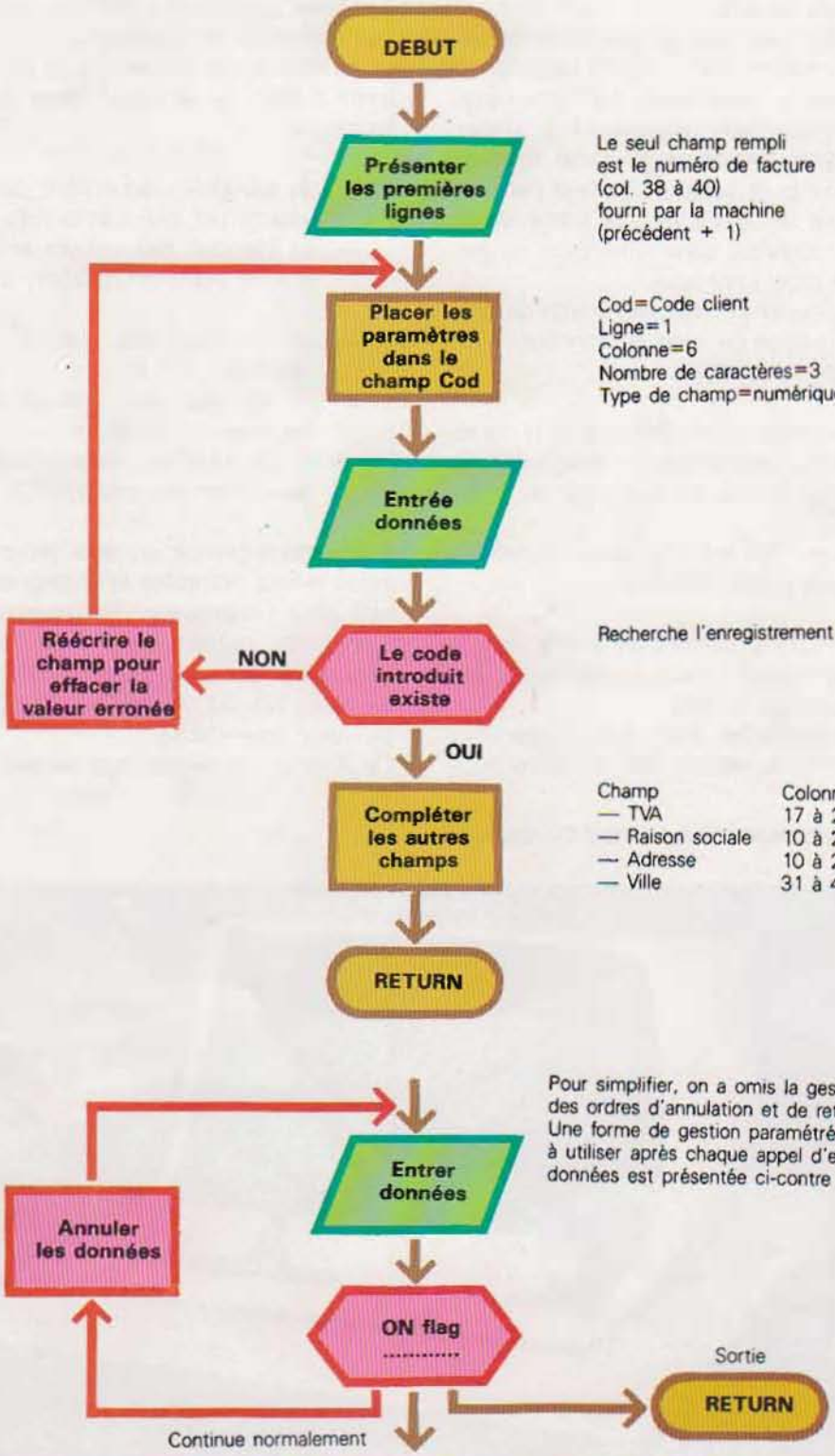
Tout comme les modules CONTROLE et MODELES, le module EVALUATION fait partie du groupe d'outils utilisés par Fred pour effectuer des analyses particulières. Le lecteur pourra agrandir ce groupe en y ajoutant des modules de résolution de situations de jeu particulières, ce qui améliorera les capacités de jeu de Fred.

DESCRIPTION DU MODULE CONTROLE

```
CONTROLE (coup)
  DEBUT
    Poser provisoirement le pion noir
    APPELER CARTE (coup, noir)
    Reprendre le pion noir
    RETOURNER au comptage des libertés
  FIN
```

Une des règles fondamentales du Gô interdit expressément la réapparition sur le Gô Ban d'une situation déjà vue.
Il faut que Fred sache appliquer correctement cette règle, appelée Ko, afin d'éviter des coups suicidaires.
C'est justement là la tâche de CONTROLE, qui garde en mémoire les paramètres courants de "coup" et de "liberté" avant que CARTE n'intervienne pour vérifier les libertés du point considéré.

SOUS-PROGRAMME DE PREPARATION DE L'EN-TETE



Le seul champ rempli est le numéro de facture (col. 38 à 40) fourni par la machine (précédent + 1)

Cod=Code client
Ligne=1
Colonne=6
Nombre de caractères=3
Type de champ=numérique

Recherche l'enregistrement

Champ	Colonne
— TVA	17 à 21
— Raison sociale	10 à 29
— Adresse	10 à 29
— Ville	31 à 40

Pour simplifier, on a omis la gestion des ordres d'annulation et de retour. Une forme de gestion paramétrée à utiliser après chaque appel d'entrée données est présentée ci-contre

Après l'introduction des données, en appuyant sur une touche fonctionnelle pré-programmée, on commande leur transfert sur disque et l'impression de la facture.
La préparation des sous-programmes constituant la procédure (voir p. 1297) ne présente pas de difficultés particulières, sauf pour ce qui concerne l'impression (alignement à ajuster) ainsi que pour l'introduction d'une ligne de données pour la facturation. Celle-ci peut encore se faire en employant la «data entry» (entrée des données) paramétré, mais sa gestion devient alors complexe.
On trouvera ci-contre l'organigramme de principe d'une gestion de ligne de données.
Trois champs sont prévus:

— code article, d'une longueur de N caractères (sa description est automatique),
— quantité (le prix est lu à partir du fichier ART),
— taux de TVA (ce champ est également lu de la même manière).

D'autres difficultés peuvent apparaître dans la phase d'impression si les quantités exprimées admettent des décimales.
Pendant l'introduction d'une ligne, l'opérateur n'est pas tenu de recourir aux nombres décimaux et peut donc se contenter de saisir un nombre entier. C'est le cas, par exemple, d'un article qui a été l'objet d'un mouvement indiqué uniquement en kg: dans ce cas, il est normal d'omettre les décimales.
Si on décide que le nombre de décimales est limité à deux, on se trouve dans les situations suivantes :

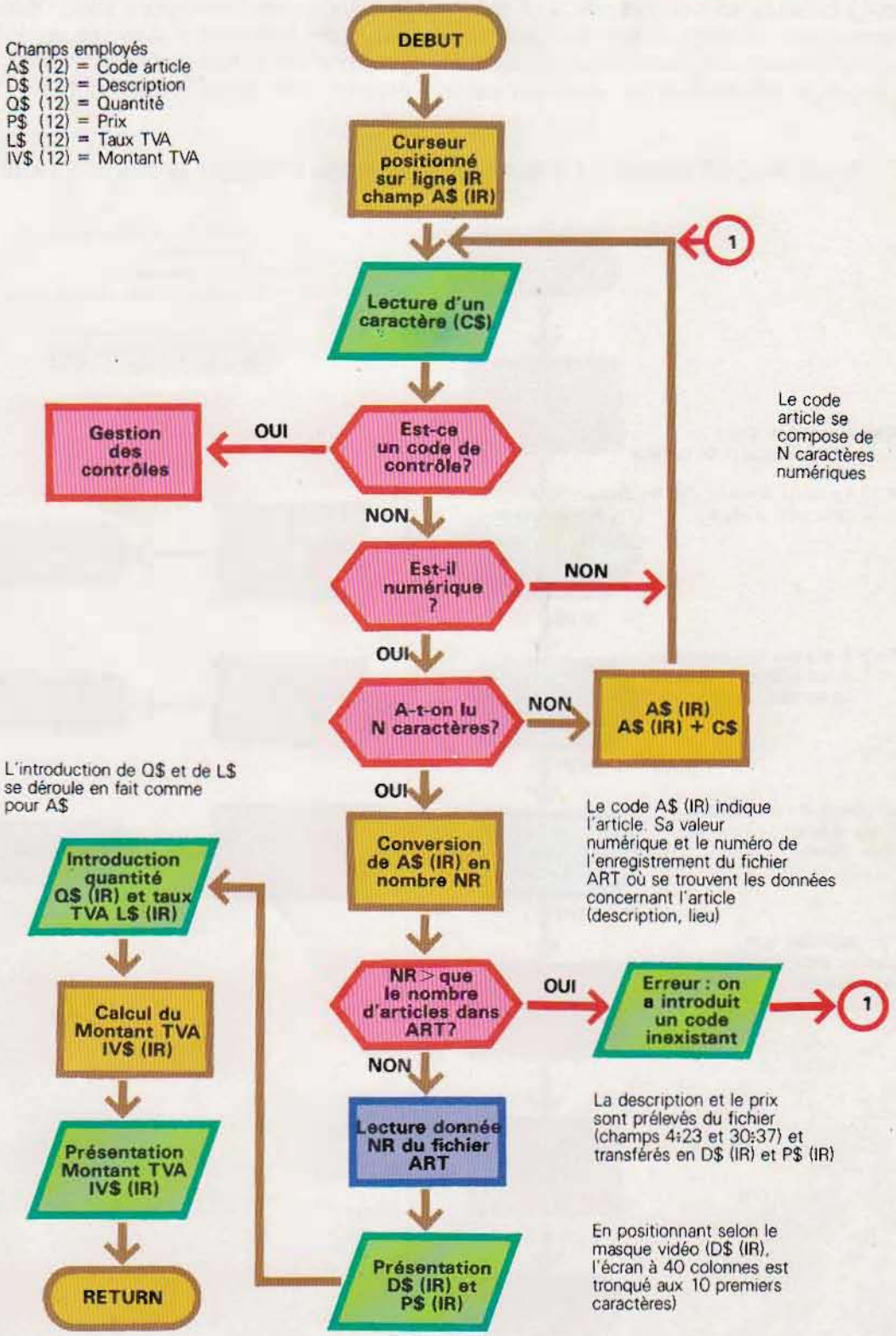
— on introduit uniquement des nombres entiers: par exemple la valeur 125,
— on introduit des valeurs entières mais avec le point (en notation américaine) : 125,
— on introduit des valeurs avec une décimale : 125.6,
— on introduit des valeurs avec deux décimales : 125.68,
— on introduit des valeurs avec plus de deux décimales : 125.6873.

Il faut donc prévoir un sous-programme qui, selon le cas, complète le champ en le préparant pour l'impression. Par exemple: fixer le format de présentation à 4 entiers et 2 décimales ; les sorties seraient alors respectivement 125.00, 125.00, 125.60 125.68, (notation américaine).
Ce résultat est simplement obtenu avec l'ins-

Système de traitement Interstec Computer.

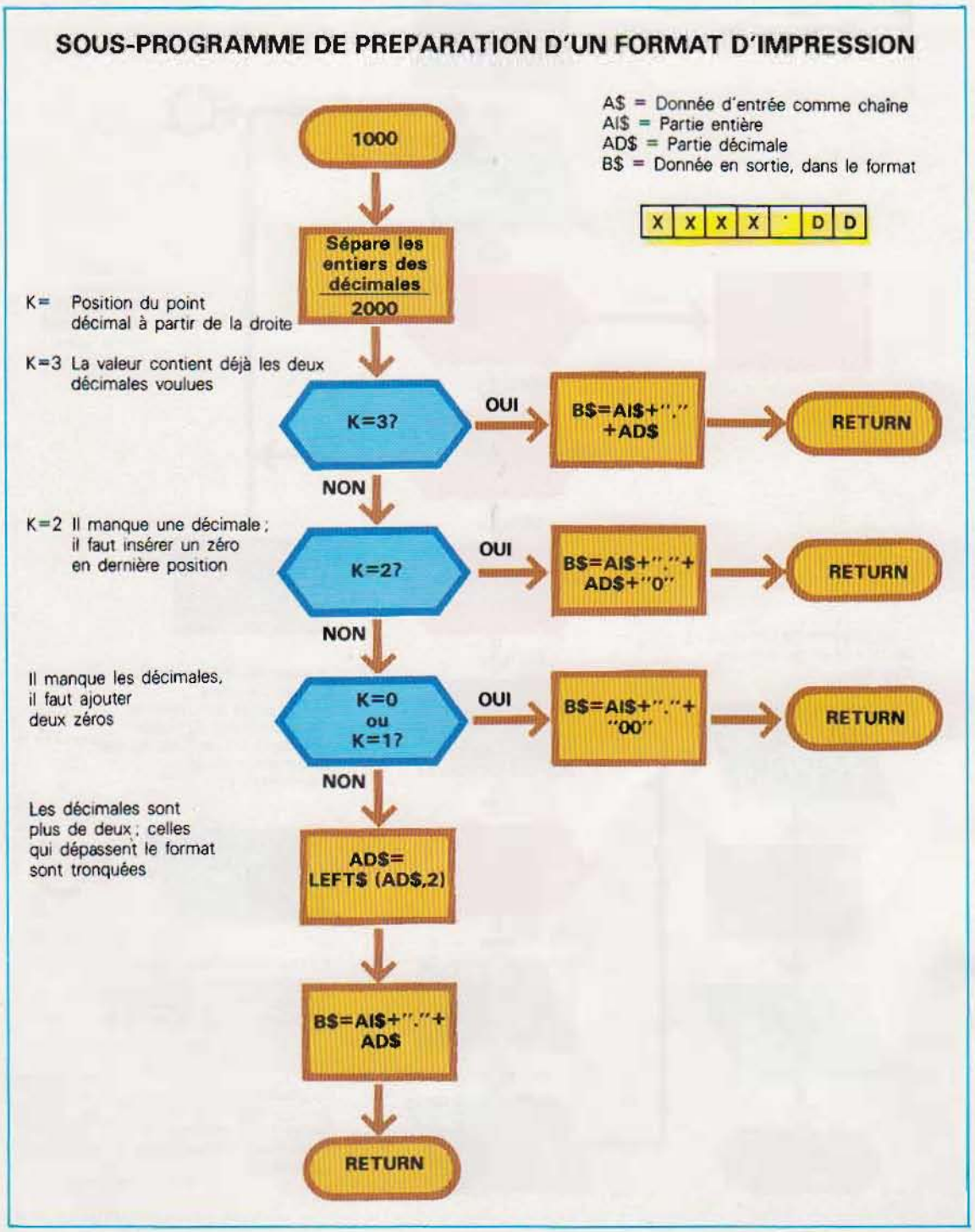
Marka/Photri

ORGANIGRAMME DE GESTION D'UNE LIGNE D'ENTREE

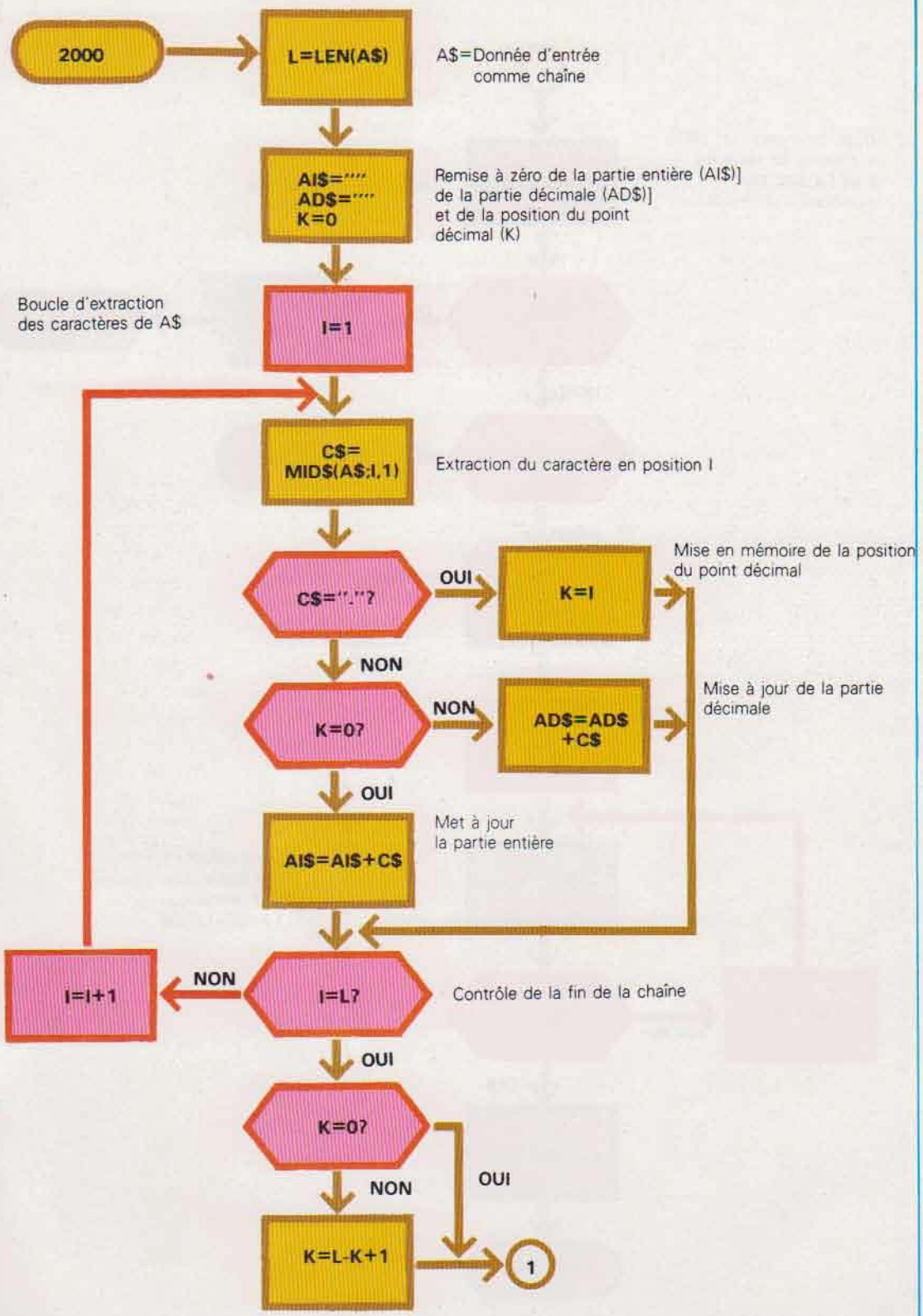


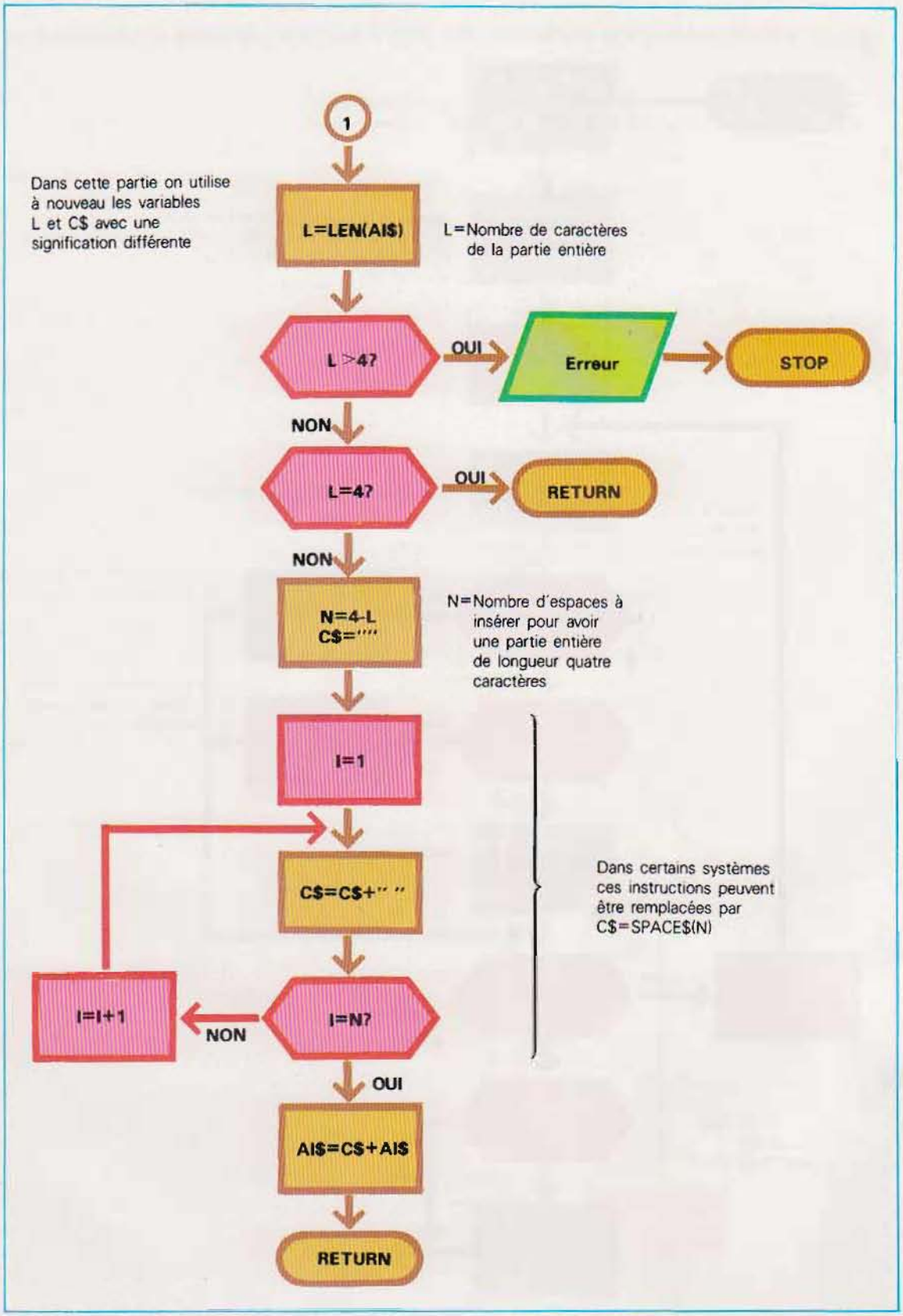
truction PRINT USING qui n'existe toutefois pas sur toutes les machines. On doit alors faire appel à un autre sous-programme. Les sous-programmes qui suivent, présentés sous la forme d'organigrammes, illustrent la préparation du format de présentation. La routine est aisément adaptable pour une machine donnée, car les deux fonctions internes, LEN et MID$ existent sur toutes les versions Basic. Naturellement, son utilité est limitée aux versions de l'interpréteur ne possédant pas d'option d'impression avec format (PRINT USING).

SOUS-PROGRAMME DE PREPARATION D'UN FORMAT D'IMPRESSION



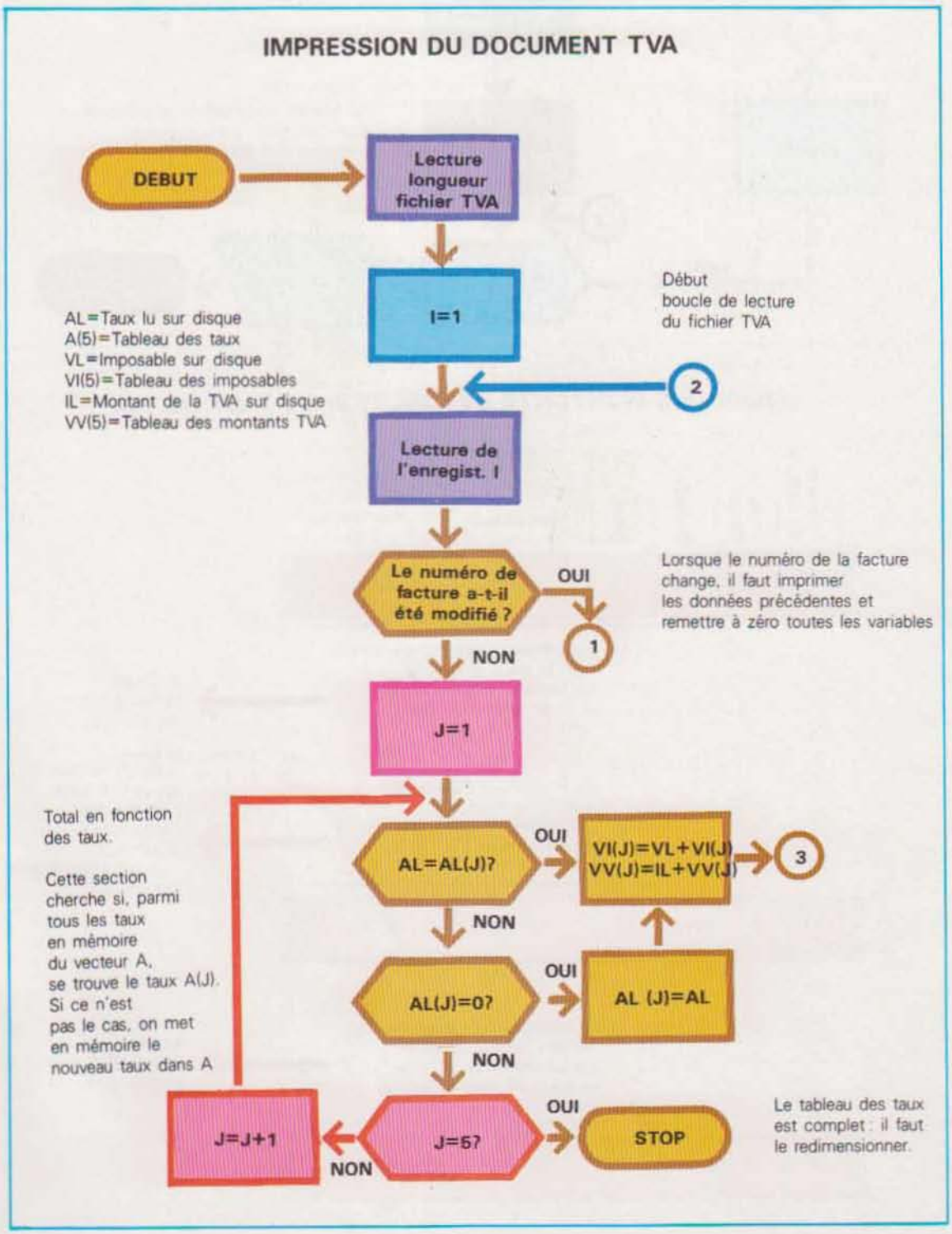
SOUS-PROGRAMME DE SEPARATION DES PARTIES ENTIERE ET DECIMALE

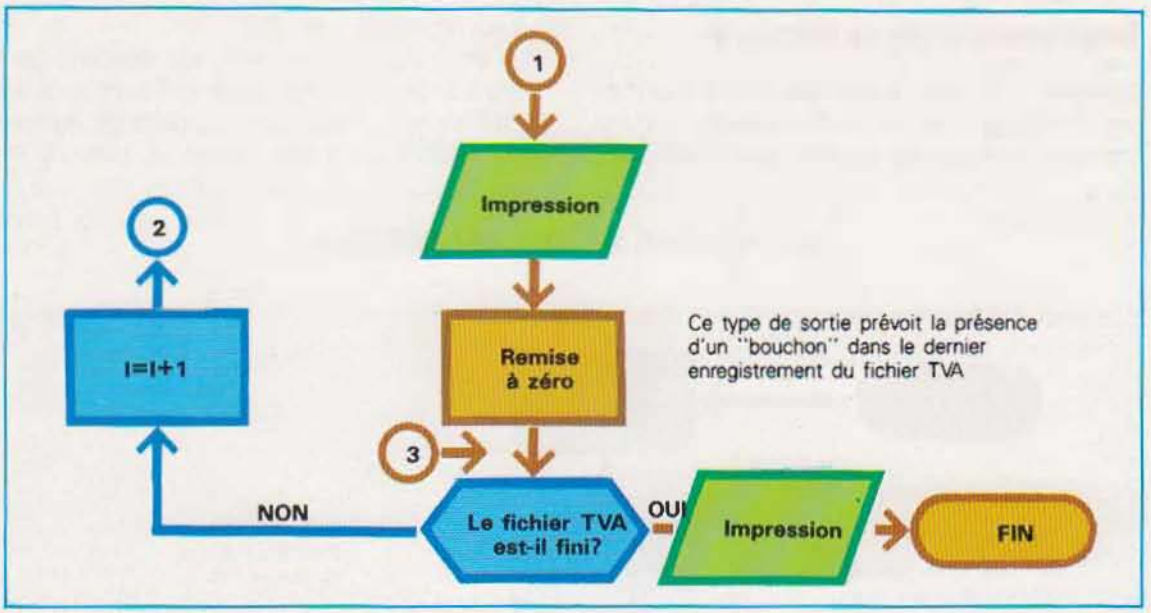



1
Dans cette partie on utilise à nouveau les variables L et C$ avec une signification différente
L=LEN(AI$)
L=Nombre de caractères de la partie entière
L >4?
OUI
Erreur
STOP
NON
L=4?
OUI
RETURN
NON
N=4-L
C$=""
N=Nombre d'espaces à insérer pour avoir une partie entière de longueur quatre caractères
I=1
C$=C$+" "
I=I+1
NON
I=N?
OUI
AI$=C$+AI$
RETURN
Dans certains systèmes ces instructions peuvent être remplacées par C$=SPACE$(N)

Impression de la facture

Selon une périodicité donnée, on va imprimer un document (bulletin ou formulaire) qui comportera, pour chaque facture, le nom du client et les montants ventilés.
A l'émission des factures, les données sont rangées dans le fichier TVA et il suffira de les additionner par rubrique. La méthode illustrée page 1303 et 1304 utilise le numéro de





CUMUL DES MONTANTS EN FONCTION DU TAUX DE TVA

Numéro de facture	Code Client	Donnée	Taux	Imposable	Montant TVA
26	03	21×12	10	10000	1000

La première lecture trouve le A(J) vide et place le taux 10 en première position (J=1). A la fin du premier tour J=1 on a:

→ A(1)=10
VI(1)=10000
VV(1)=1000

Le deuxième enregistrement a le même taux. Le programme additionne les données dans la même position

→ A(1)=10 (le taux est un indicateur; il ne faut pas l'additionner)
VI(1)=35000 (10000+25000)
VV(1)=3500

26	03	21×12	10	25000	2500

Le troisième enregistrement contient un taux différent qui est donc mis en mémoire dans la position suivant (J=2). On obtient:

→ A(2)=8
V(2)=20000
VV(2)=1600

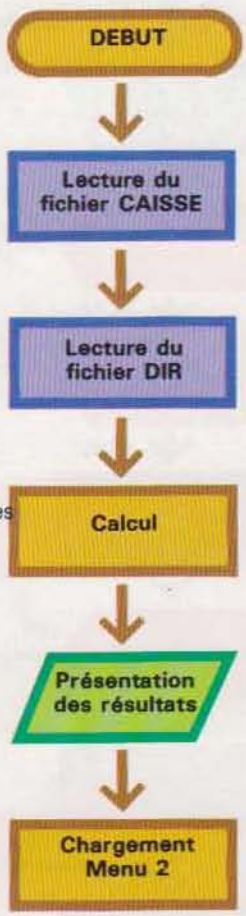
26	03	21×12	8	20000	1600

facture comme code. Le changement de numéro déclenche l'impression des additions correspondantes, la remise à zéro et le début d'un nouveau cycle. En plus de l'impression du bulletin, cette routine sera également utilisée en fin de calcul du document pour faire ressortir, facture par facture, les montants imposables et les TVA en fonction du taux retenu. Ces impressions sont à nouveau lancées en fin d'année quand il faut mettre en mémoire, dans le fichier TVA, des données récapitulatives.

Autres fonctions

Pour compléter le programme, d'autres procédures sont obtenues en y introduisant quelques modifications. Les fonctions (Situation de Caisse et Recettes/Dépenses organigrammes ci-dessous et p. 1306) de deux fonctions sont semblables et ne présentent aucune difficulté d'écriture par rapport aux précédentes.
En plus des modules ou des sous-programmes nécessaires à cette application particulière,

ORGANIGRAMME DE LA SITUATION DE CAISSE

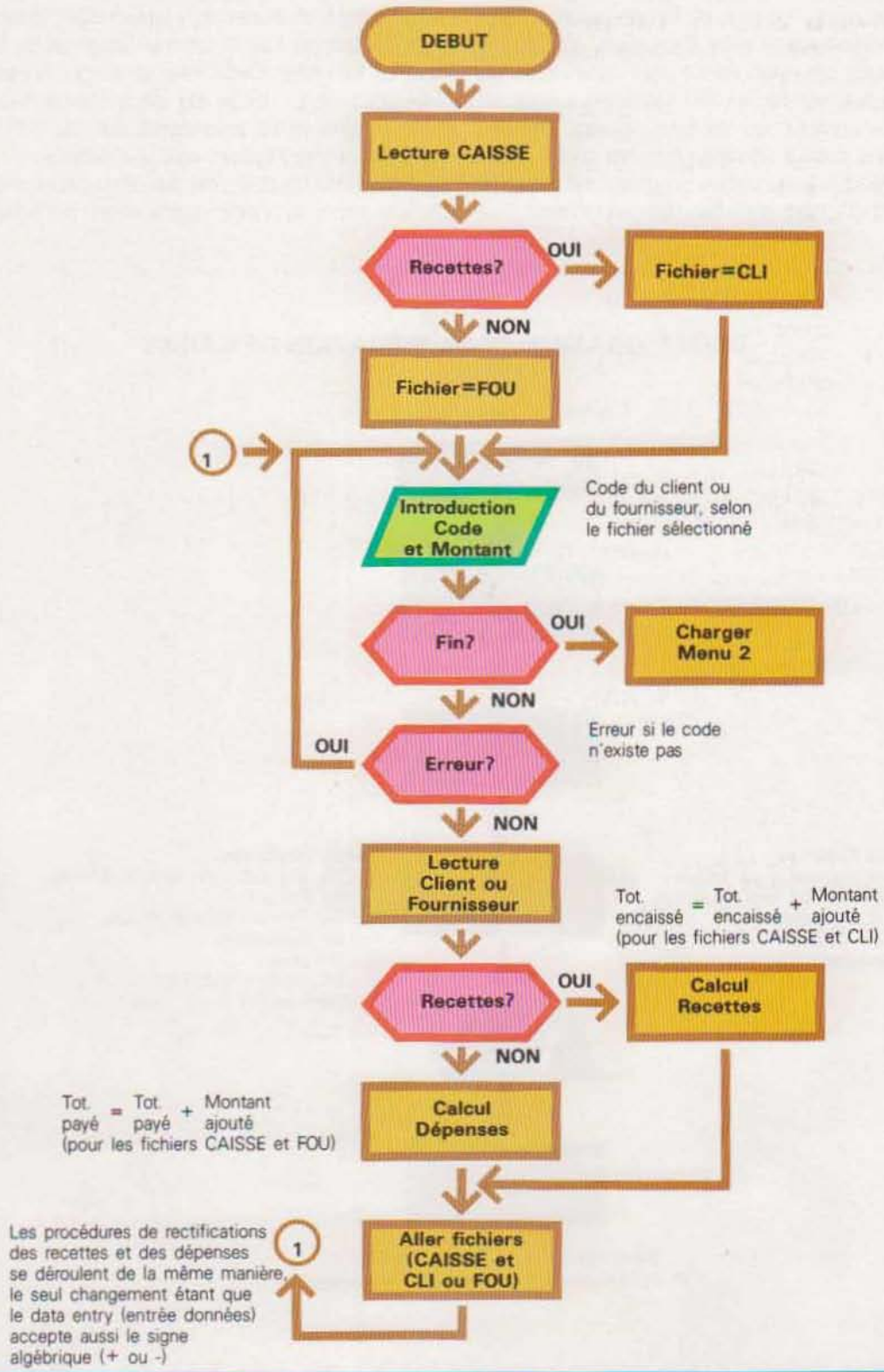


Calcul Recettes
1 - Tot. imposable sur factures émises
2 - Tot. TVA sur factures émises
3 - Tot. encaissé
4 - Tot. avoir=(1+2)-3
5 - Recettes=4-2

Calcul Dépenses
1 - Tot. imposable sur factures émises par fournisseurs
2 - Tot. TVA sur factures émises par fournisseurs
3 - Tot. payé
4 - Tot. débit=(1+2)-3
5 - Dépenses=4-2

Bénéfice=5 (recettes) -5 (dépenses)
TVA (crédit/verser) =2 (recettes) -2 (dépenses)

ORGANIGRAMME RECETTES/DEPENSES



on a besoin d'autres routines comme SORT (TRI) d'intérêt général. Le classement des données est une fonction très employée, d'où précisément la nécessité d'un programme général, susceptible d'être adapté aisément à différents contextes.

On trouvera ci-dessous l'organigramme d'un programme de tri. Le premier bloc génère des chaînes aléatoires (détails p. 1309) auxquelles on applique, à titre de contrôle, le sous-programme SORT (routine 590 p. 1310). Dans notre exemple, les chaînes sont obtenues automatiquement avec le générateur de nombres aléatoires RND (voir listing page 1308).

On peut également prévoir, comme alternative, une boucle d'introduction à partir du clavier ou d'un lecteur de disque.

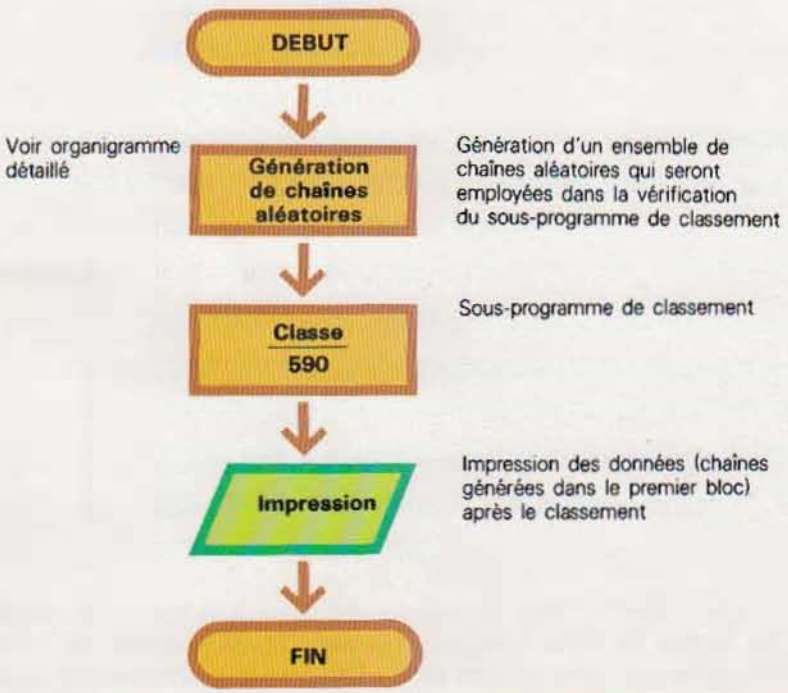
Le sous-programme 590, appelé à la ligne 260 du programme principal, réalise le tri et restitue le tableau IS des pointeurs aux données en ordre croissant. Le dernier bloc imprime les valeurs ordonnées.

Page 1310, on trouvera l'organigramme de premier niveau de ce sous-programme qui sera détaillé pages 1311 à 1313 (voir listings

B. Coleman/Marka

Système de traitement pour la CAO et le DAO.

ORGANIGRAMME D'UN PROGRAMME DE TRI

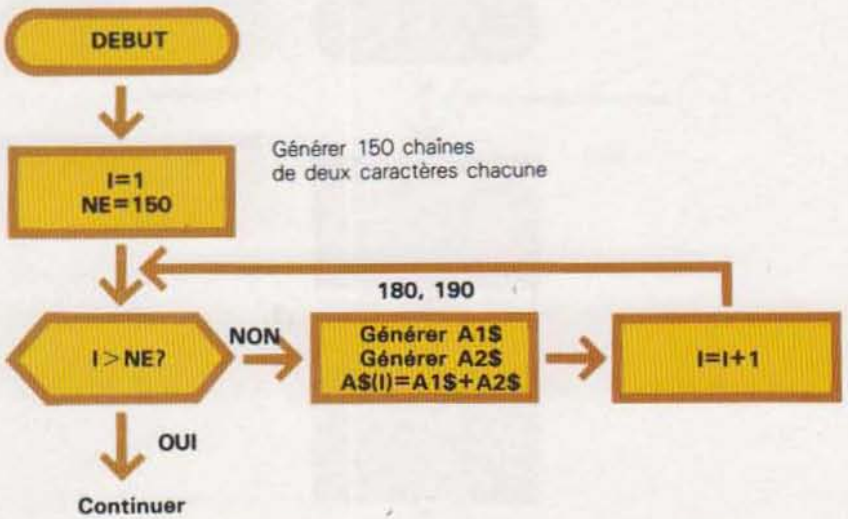


PROGRAMME DE CLASSEMENT (PRINCIPAL)

```
10 REM  ******************
20 REM  *                *
30 REM  *      TRI       *
40 REM  *                *
50 REM  ******************
60 :
70 HOME
80 DIM A$(150),IS(160)
90 NE=150
100:
110 REM ==========================
120 REM * GENERATION DE LETTRES *
130 REM ==========================
140 :
150 HTAB 7
160 VTAB 5:PRINT "GENERATION DE LETTRES
    ALEATOIRES"
170 FOR I=1 TO NE
180 A1$=CHR$ (INT (RND (1)*26+65)) : A2$=CHR$ (INT (RND (1)*26+65))
190 A$(I)=A1$+A2$
200 NEXT I
210 :
220 REM ==================
230 REM * ROUTINE DE TRI *
240 REM ==================
250 :
260 GOSUB 590
270 :
280 :
290 REM ===============
300 REM * IMPRESSION *
310 REM ===============
320 :
330 HOME
340 X=2:Y=1
350 FOR I=1 TO NE
360 K=IS(I):C$=A$(K)
370 HTAB X:VTAB Y:PRINT C$
380 IF Y=23 THEN Y=1:X=X+
        3:GOTO 400
390 Y=Y+2
400 NEXT I
410 VTAB 24:HTAB 38:GET D$
420 NORMAL
430 HOME:VTAB 24
440 END
450 :
```

correspondants pages 1313 à 1315). La logique employée est fondée sur la recherche binaire. La première phase consiste à calculer M, pointeur de comparaison. A l'initialisation, M équivaut à la moitié du nombre de données classées (la première fois, c'est 1).
On considère ensuite un à un tous les éléments de la liste, avec une boucle allant de 1 à NE

GENERATION DE CHAINES ALEATOIRES



(nombre de données, instructions 640 à 1440 du listing).
Chacun d'eux est comparé à l'élément pointé par M. Par exemple, avec I = 25 et M = 6, on compare A$(25) à A$(6), et on détermine, en fonction du résultat, si la position de I dans le tableau des pointeurs a été trouvée ou s'il faut effectuer une nouvelle recherche. Dans ce dernier cas, on recalcule M, après avoir établi s'il faut le déplacer à droite ou à gauche.
Si la position est trouvée, on met en mémoire le pointeur de l'élément dans le tableau IS, en opérant, le cas échéant, une translation des pointeurs précédemment insérés.
A la fin, le sous-programme de tri a mémorisé, dans IS, les pointeurs des éléments de AS selon un ordre croissant. Pour obtenir l'impression ordonnée, il faut extraire de A$ chaque élément selon l'ordre mémorisé en IS.
Supposons, par exemple, que IS contienne :

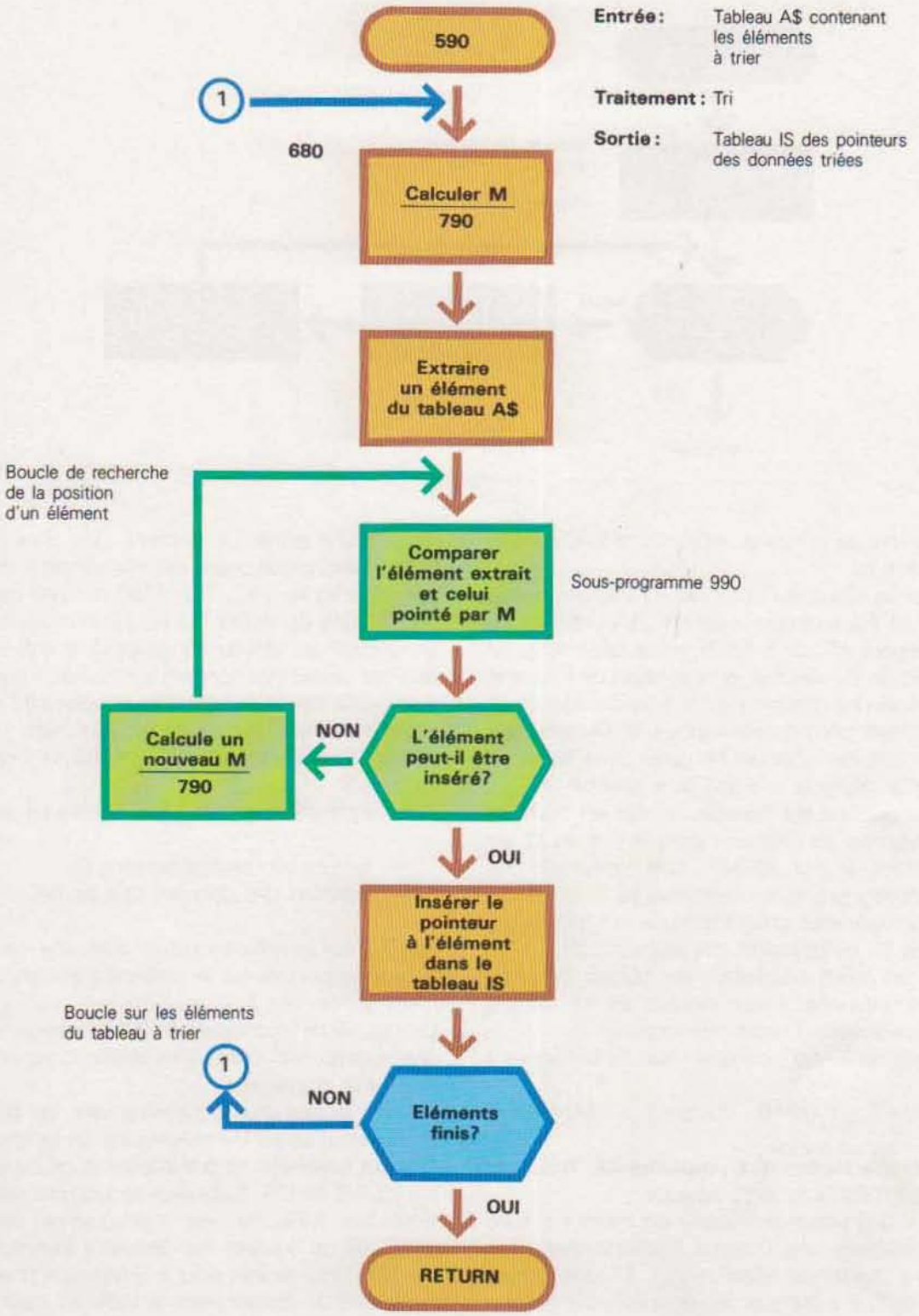
IS(1)=7, IS(2)=9, IS(3)=1, IS(4)=15,...

le premier élément à imprimer est A$(7), le second A$(9) et ainsi de suite.
Le listing présenté travaille en mémoire, sans paramétrage des champs à sélectionner.
Dans certaines applications, il peut arriver qu'on doive prélever les données des fichiers du disque ou encore n'exécuter de tri qu'à partir d'une partie du contenu des champs. La première modification est très simple à réaliser. Aux lignes 710, 730, 1130 et 1240 dans les variables de chaîne IV$ et I1$, on transfère un élément du tableau A$ supposé en mémoire. Pour utiliser des données sur disque, il suffit de modifier ces instructions en remplaçant l'assignation simple, par exemple IV$=A$(I), par une lettre sur l'enregistrement pointé par l'indice de A$.
Ainsi l'instruction 710 doit être remplacée par :

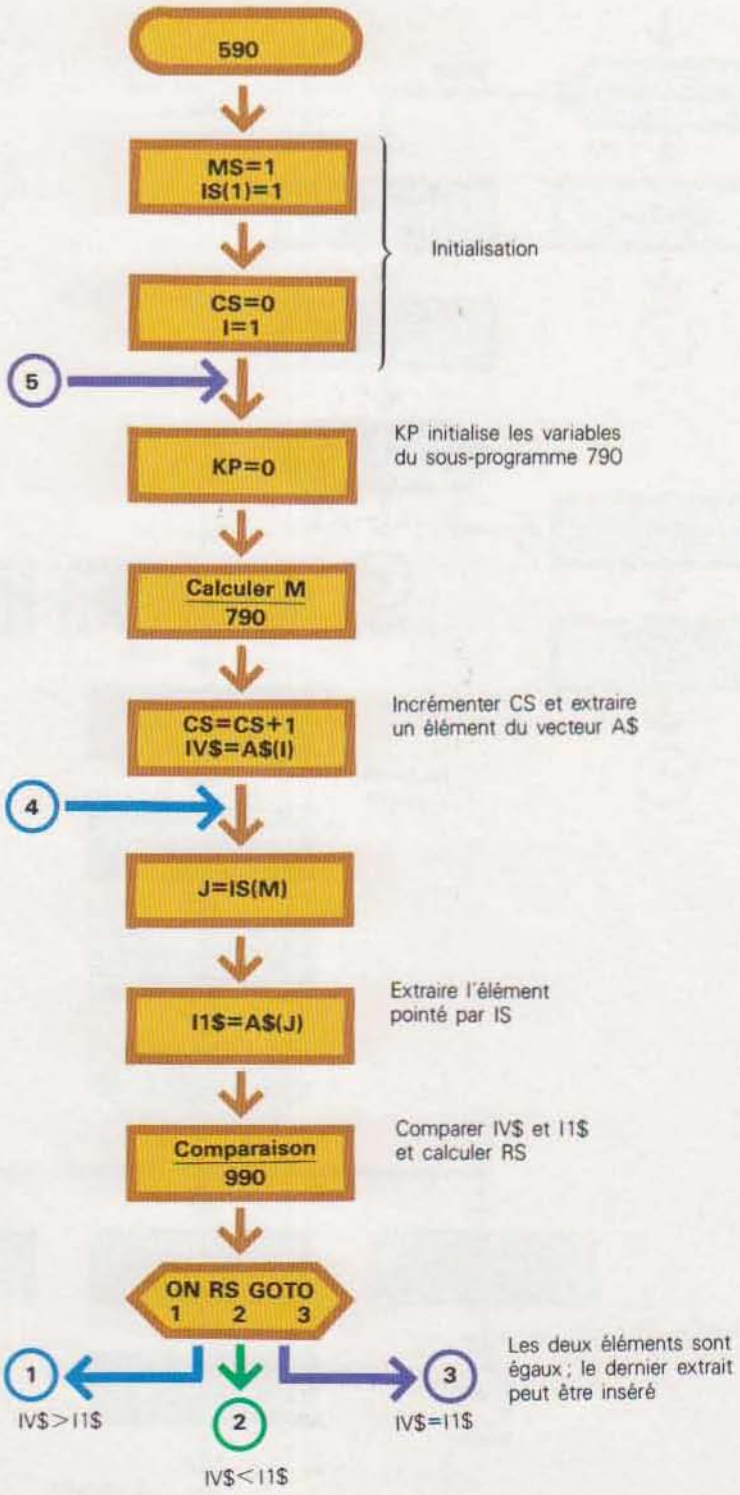
— lecture de l'enregistrement I
— transfert des données lues en IV$.

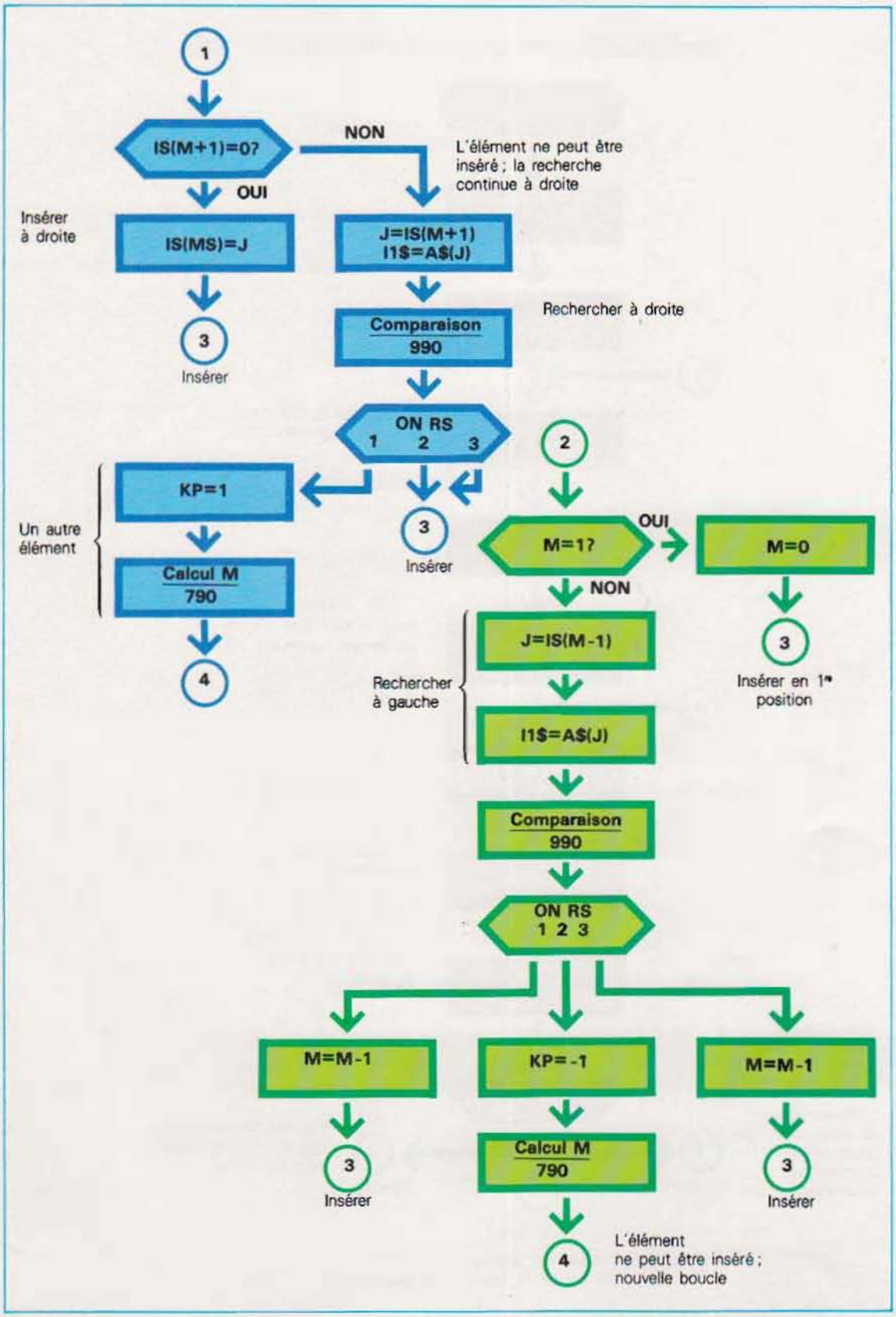
La 730 suit la même logique, avec une seule différence portant sur le numéro d'enregistrement qui devient J.
La deuxième modification (paramétrage du champ employé pour le tri) s'obtient conjointement à la première.
Dans la lecture d'un enregistrement, les données sont d'abord transférées sur un tampon. Ne sont prélevées et transférées dans les variables IV$ ou I1$, que celles se trouvant dans la position adéquate (voir organigramme page 1315). Si on travaille sur disque, il faut aussi modifier l'impression, pour ménager une phase de lecture du disque avant le transfert dans la variable C$ (ligne 360).

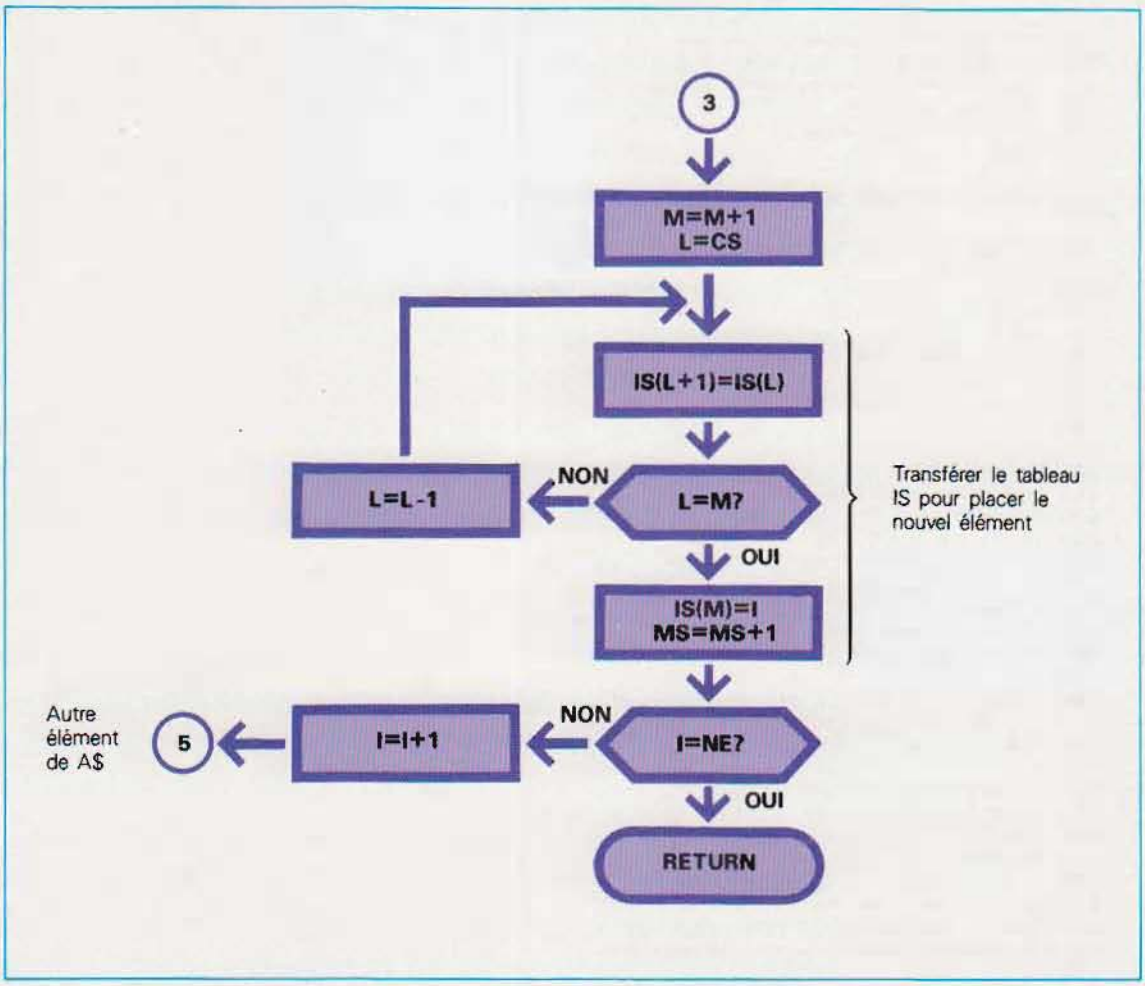
ORGANIGRAMME DE PRINCIPE DU SOUS-PROGRAMME 590



ORGANIGRAMME DU SOUS-PROGRAMME 590




1
IS(M+1)=0?
NON
L'élément ne peut être inséré ; la recherche continue à droite
OUI
Insérer à droite
IS(MS)=J
J=IS(M+1)
I1$=A$(J)
3
Insérer
Comparaison
990
Rechercher à droite
ON RS
1 2 3
2
KP=1
3
Insérer
M=1?
OUI
M=0
Un autre élément
Calcul M
790
NON
J=IS(M-1)
3
Insérer en 1re position
Rechercher à gauche
I1$=A$(J)
4
Comparaison
990
ON RS
1 2 3
M=M-1
KP=-1
M=M-1
3
Insérer
Calcul M
790
3
Insérer
4
L'élément ne peut être inséré ; nouvelle boucle



PROGRAMME DE CLASSEMENT (SOUS-PROGRAMME)

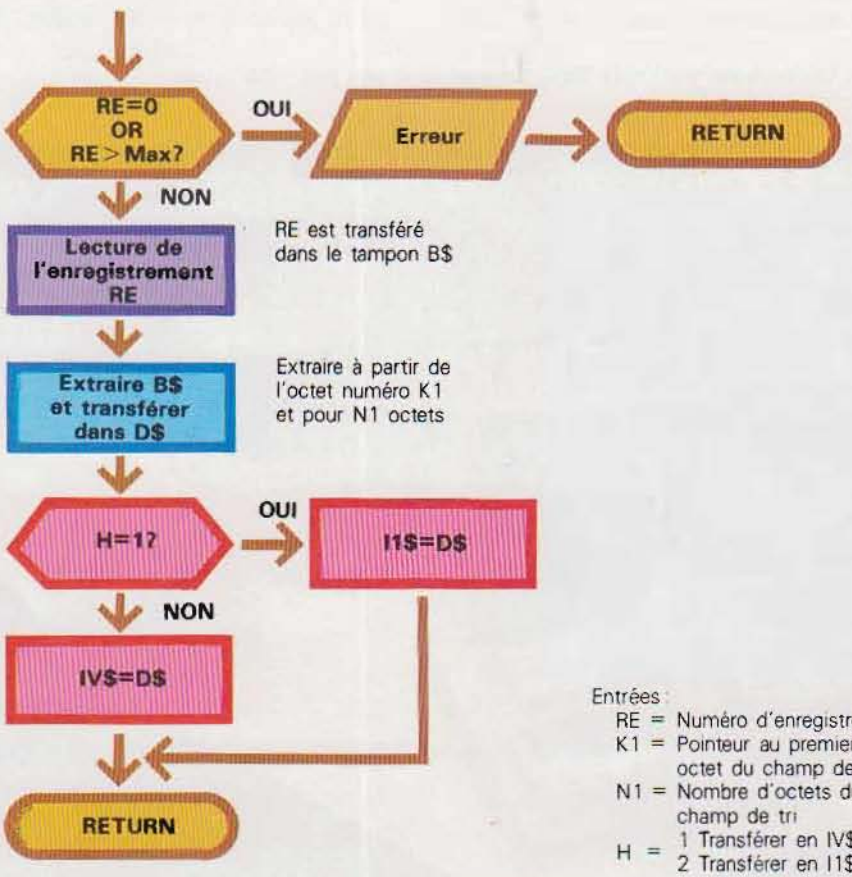
```
460 REM ============
470 REM *    TRI    *
480 REM ============
490 :
500 REM ====================================
510 REM * IS(.) = VECTEUR CLASSE           *
520 REM * MS    = MAX POINTEUR IS()        *
530 REM * CS    = NBRE D'ELEMENTS CLASSES  *
540 REM * A$()  = VECTEUR A CLASSER        *
550 REM * IV$   = ELEMENT A CLASSER        *
560 REM * I1$   = ELEMENT DE COMPARAISON   *
570 REM * M     = POINTEUR DE RECHERCHE    *
580 REM ====================================
590 MS=1:IS(1)=1:CS=0
600 HOME:VTAB 10:INVERSE:PRINT "JE CLASSE L'ELEMENT N° : "
```

```
610 REM ========================
620 REM * CYCLE DE CLASSEMENT *
630 REM ========================
640 FOR I=1 TO NE
650 HTAB 24:VTAB 10:PRINT I
660 KP=0
670 :
680 GOSUB 790: REM * M *
690 :
700 CS=CS+1
710 IV$=A$(I)
720 REM
730 J=IS(M):I1$=A$(J)
740 :
750 GOSUB 990: REM * CONF. *
760 :
770 ON RS GOI 1080.1090.1320
780 :
790 REM ==================
800 REM * CALCUL DE (M) *
810 REM ==================
820 :
830 IF KP<>0 THEN 870
840 MP=1:PP=MS:PS=0:M=INT (MS/2)
850 IF M=0 THEN M=1
860 RETURN
870 IF KP=1 THEN 900
880 IF KP= -1 THEN 950
890 :
900 MP=M:IF PP=0 THEN PP=MS
910 M2=INT ((PP-M)/2):IF M2=0 THEN M2=1
920 M=M+M2
930 :
940 RETURN
950 PP=M:M2=INT ((M-MP)/2):IF M2=0 THEN M2=1
960 M=M-M2
970 RETURN
980 :
990 REM  ============================
1000 REM * COMPARAISON DES ELEMENTS *
1010 REM ============================
1020 :
1030 RS=3
1040 IF IV$>I1$ THEN RS=1
1050 IF IV$<J1$ THEN RS=2
1060 RETURN
1070 :
1080 REM ====================
1090 REM *    IV$ > I1$     *
1100 REM ====================
1110 :
1120 IF IS(M+1)=0 THEN IS(MS)=J:GOTO 1320
1130 J=IS(M+1):I1$=A$(J)
1140 GOSUB 990
1150 ON RS GOTO 1160.1320.1320
1160 KP=1
1170 GOSUB 790:GOTO 720
1180 :
1190 REM ====================
1200 REM *    IV$ <  I1$    *
1210 REM ====================
1220 :
1230 IF M=1 THEN M=0:GOTO 1320
1240 J=IS(M-1):I1$=A$(J)
1250 GOSUB 990
```

```
1260 ON RS GOTO 1270,1280,1270
1270 M=M-1:GOTO 1320
1280 KF= -1
1290 GOSUB 790
1300 GOTO 720
1310 :
1320 REM ==================
1330 REM *     IV$=I1$     *
1340 REM ==================
1350 :
1360 M=M+1
1370 :
1380 REM * CONSTRUCTION DE IS(.) *
1390 :
1400 FOR I=CS TO M STEP -1
1410 IS(L+1)=IS(L)
1420 NEXT L
1430 IS(M)=I:MS=MS+1
1440 NEXT I
1450 RETURN
```

GENERALISATION DU SOUS-PROGRAMME DE TRI

Lecture du disque
Extraction de la partie utilisée pour le tri
Transfert à la variable correspondante



Entrées :
RE = Numéro d'enregistrement
K1 = Pointeur au premier octet du champ de tri
N1 = Nombre d'octets du champ de tri
H = 1 Transférer en IV$
2 Transférer en I1$

La transmission des informations

En informatique, le concept de transmission de données signifie un déplacement d'un point à un autre d'informations codées, au moyen de systèmes de communication de type électronique ou électromécanique. La distance séparant les deux points en question est théoriquement illimitée, mais il faut préciser qu'elle conditionne le choix d'un support de transmission.

Dans les moyens traditionnels (téléphone, radio, télévision), l'information (voix, image ou texte écrit) est convertie, depuis sa source, en signaux électromagnétiques. Elle est transmise grâce à un moyen «diélectrique» (le câble métallique d'une ligne téléphonique, l'éther). A la réception, les signaux sont retransformés, de manière à restituer l'information dans sa forme initiale.

Examinons maintenant de plus près la communication des informations dans le cas de l'ordinateur. Le premier transfert a lieu lors du dialogue homme-machine. L'ordinateur fonctionne en utilisant des séquences de chiffres binaires 0 et 1: pour communiquer avec l'homme il lui faut donc traduire chaque séquence de bits en caractères alphabétiques et numériques. La traduction est réalisée au moyen de tableaux de correspondance bi-univoque entre caractères (code homme) et séquences de bits (code machine). Les formes de codage les plus répandues sont l'ASCII (American Standard Code for Information Interchange) et l'EBCDIC (Extended Binary Coded Decimal Interchange Code); la première est couramment employée dans les micro-ordinateurs, la seconde est typique des grosses machines IBM. Ces deux codes diffèrent par les combinaisons de bits associées à un caractère. Tous deux comportent, outre les chiffres et les lettres, des symboles

Système de traitement portable NEC connecté à ses périphériques.

R. Villa/Marka

CODIFICATION ASCII DES CARACTERES SPECIAUX EMPLOYES EN TELETRANSMISSION

Caractère	Signification	Codification ASCII bit n° 8*7654321	Valeur décimale	Valeur octale	Valeur hexadécimale
NUL	Null	0 0 0 0 0 0 0	0	0	0
SOH	Start of Heading	0 0 0 0 0 0 1	1	1	1
STX	Start of Text	0 0 0 0 0 1 0	2	2	2
ETX	End of Text	0 0 0 0 0 1 1	3	3	3
EOT	End of Transmission	0 0 0 0 1 0 0	4	4	4
ENQ	Enquiry	0 0 0 0 1 0 1	5	5	5
ACK	Acknowledgement	0 0 0 0 1 1 0	6	6	6
BEL	Bell	0 0 0 0 1 1 1	7	7	7
BS	Backspace	0 0 0 1 0 0 0	8	10	8
HT	Horizontal Tabulation	0 0 0 1 0 0 1	9	11	9
NL	New Line	0 0 0 1 0 1 0	10	12	A
VT	Vertical Tabulation	0 0 0 1 0 1 1	11	13	B
FF	Form Feed	0 0 0 1 1 0 0	12	14	C
RT	Return	0 0 0 1 1 0 1	13	15	D
SO	Shift Out	0 0 0 1 1 1 0	14	16	E
SI	Shift In	0 0 0 1 1 1 1	15	17	F
DLE	Data Line Escape	0 0 1 0 0 0 0	16	20	10
DC1	Device Control 1	0 0 1 0 0 0 1	17	21	11
DC2	Device Control 2	0 0 1 0 0 1 0	18	22	12
DC3	Device Control 3	0 0 1 0 0 1 1	19	23	13
DC4	Device Control 4	0 0 1 0 1 0 0	20	24	14
NAK	Negative Acknowledgement	0 0 1 0 1 0 1	21	25	15
SYN	Synchronous Idle	0 0 1 0 1 1 0	22	26	16
ETB	End of Transmission Block	0 0 1 0 1 1 1	23	27	17
CAN	Cancel	0 0 1 1 0 0 0	24	30	18
EM	End of Medium	0 0 1 1 0 0 1	25	31	19
SUB	Substitute	0 0 1 1 0 1 0	26	32	1A
ESC	Escape	0 0 1 1 0 1 1	27	33	1B
FS	File Separator	0 0 1 1 1 0 0	28	34	1C
GS	Group Separator	0 0 1 1 1 0 1	29	35	1D
RS	Record Separator	0 0 1 1 1 1 0	30	36	1E
US	Unit Separator	0 0 1 1 1 1 1	31	37	1F

*Le bit n° 8 est le bit de parité.

spéciaux qui sont justement employés dans la communication de données et dont on parlera plus loin. On trouvera ci-dessus la codification ASCII des caractères spéciaux qui concernent la communication des données et, page suivante, la codification EBCDIC. Pour l'ASCII, sur lequel on reviendra, on indique également la signification des différents caractères.

Modes de transmission

Nous avons déjà dit que ces deux systèmes emploient 8 bits pour représenter un caractère. Généralement 7 bits seulement contiennent l'information proprement dite, le huitième étant le bit de parité.

Pour transmettre un caractère entre deux équipements, par exemple un ordinateur et son périphérique, il est nécessaire de les relier par une ligne de communication. Selon la manière de réaliser cette ligne, on obtient deux types de transmission (voir schémas p. 1319):

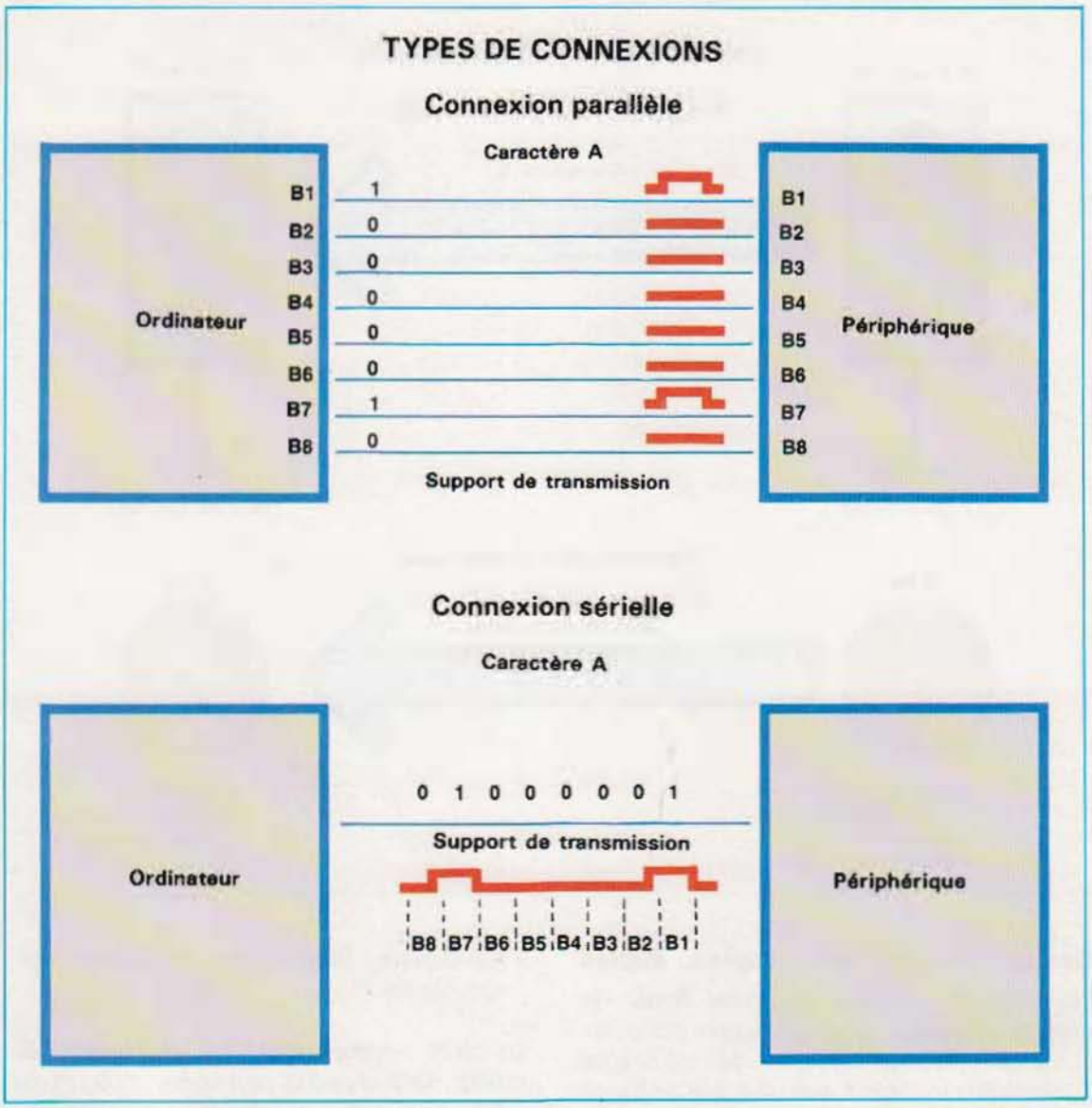
- **connexion parallèle**
  La ligne est composée de 8 fils distincts sur lesquels «voyagent» simultanément les 8 bits composant l'information.
- **connexion sérielle** (ou type série)
  La ligne est composée d'un fil unique sur lequel les bits voyagent l'un après l'autre.

Ainsi, avec la connexion parallèle, on atteint des vitesses de transmission plus élevées. C'est pourquoi ce type de liaison est typique des périphériques rapides (unité disque, unité bande, imprimante rapide). Toutefois la distance entre ces unités et l'ordinateur est limitée (quelques mètres) et c'est une solution coûteuse. Si elle n'atteint pas des vitesses très élevées, la connexion sérielle permet néanmoins de raccorder des périphériques, même à de très grandes distances, en utilisant les lignes téléphoniques.

CODIFICATION EBCDIC DES CARACTERES SPECIAUX EMPLOYES EN TELETRANSMISSION

Caractère	Codification EBCDIC bit n° 8* 7 6 5 4 3 2 1	Valeur décimale	Valeur octale	Valeur héxadécimale
NUL	0 0 0 0 0 0 0 0	0	0	0
SOH	0 0 0 0 0 0 0 1	1	1	1
STX	0 0 0 0 0 0 1 0	2	2	2
ETX	0 0 0 0 0 0 1 1	3	3	3
PF	0 0 0 0 0 1 0 0	4	4	4
HT	0 0 0 0 0 1 0 1	5	5	5
LC	0 0 0 0 0 1 1 0	6	6	6
DEL	0 0 0 0 * 1 1 1	7	7	7
	0 0 0 1 0 0 0 0	8	10	8
RLF	0 0 0 1 0 0 0 1	9	11	9
SMM	0 0 0 1 0 1 0 0	10	12	A
VT	0 0 0 1 0 1 1 1	11	13	B
FF	0 0 0 1 1 1 0 0	12	14	C
CR	0 0 0 1 1 1 0 1	13	15	D
SO	0 0 0 1 1 1 1 0	14	16	E
SI	0 0 0 1 1 1 1 1	15	17	F
DLE	0 0 1 0 0 0 0 0	16	20	10
DC1	0 0 1 0 0 0 0 1	17	21	11
DC2	0 0 1 0 0 1 0	18	22	12
DC3	0 0 1 0 0 1 1	19	23	13
RES	0 0 1 0 1 0 0 0	20	24	14
NL	0 0 1 0 1 0 0 1	21	25	15
BS	0 0 1 0 1 1 0	22	26	16
IDL	0 0 1 0 1 1 1	23	27	17
CAN	0 0 1 1 0 0 0 0	24	30	18
EM	0 0 1 1 0 0 1	25	31	19
CC	0 0 1 1 0 1 0	26	32	1A
CU1	0 0 1 1 0 1 1	27	33	1B
IFS	0 0 1 1 1 0 0	28	34	1C
IGS	0 0 1 1 1 0 1	29	35	1D
IRS	0 0 1 1 1 1 0	30	36	1E
IUS	0 0 1 1 1 1 1	31	37	1F
DS	0 1 0 0 0 0 0	32	40	20
SOS	0 1 0 0 0 0 1	33	41	21
FS	0 1 0 0 0 1 0	34	42	22
	0 1 0 0 0 1 1	35	43	23
BYP	0 1 0 0 1 0 0	36	44	24
LF	0 1 0 0 1 0 1	37	45	25
EOB/ETB	0 1 0 0 1 1 0	38	46	26
PRE/ESC	0 1 0 0 1 1 1	39	47	27
	0 1 0 1 0 0 0	40	50	28
	0 1 0 1 0 0 1	41	51	29
SM	0 1 0 1 0 1 0	42	52	2A
CU2	0 1 0 1 0 1 1	43	53	2B
	0 1 0 1 1 0 0	44	54	2C
ENQ	0 1 0 1 1 0 1	45	55	2D
ACK	0 1 0 1 1 1 0	46	56	2E
BEL	0 1 0 1 1 1 1	47	57	2F
	0 1 1 0 0 0 0	48	60	30
	0 1 1 0 0 0 1	49	61	31
SYN	0 1 1 0 0 1 0	50	62	32
	0 1 1 0 0 1 1	51	63	33
PN	0 1 1 0 1 0 0	52	64	34
RS	0 1 1 0 1 0 1	53	65	35
UC	0 1 1 0 1 1 0	54	66	36
EOT	0 1 1 0 1 1 1	55	67	37
	0 1 1 1 0 0 0	56	70	38
	0 1 1 1 0 0 1	57	71	39
	0 1 1 1 0 1 0	58	72	3A
CU3	0 1 1 1 0 1 1	59	73	3B
DC4	0 1 1 1 1 0 0	60	74	3C
NAK	0 1 1 1 1 0 1	61	75	3D
	0 1 1 1 1 1 0	62	76	3E
SUB	0 1 1 1 1 1 1	63	77	3F

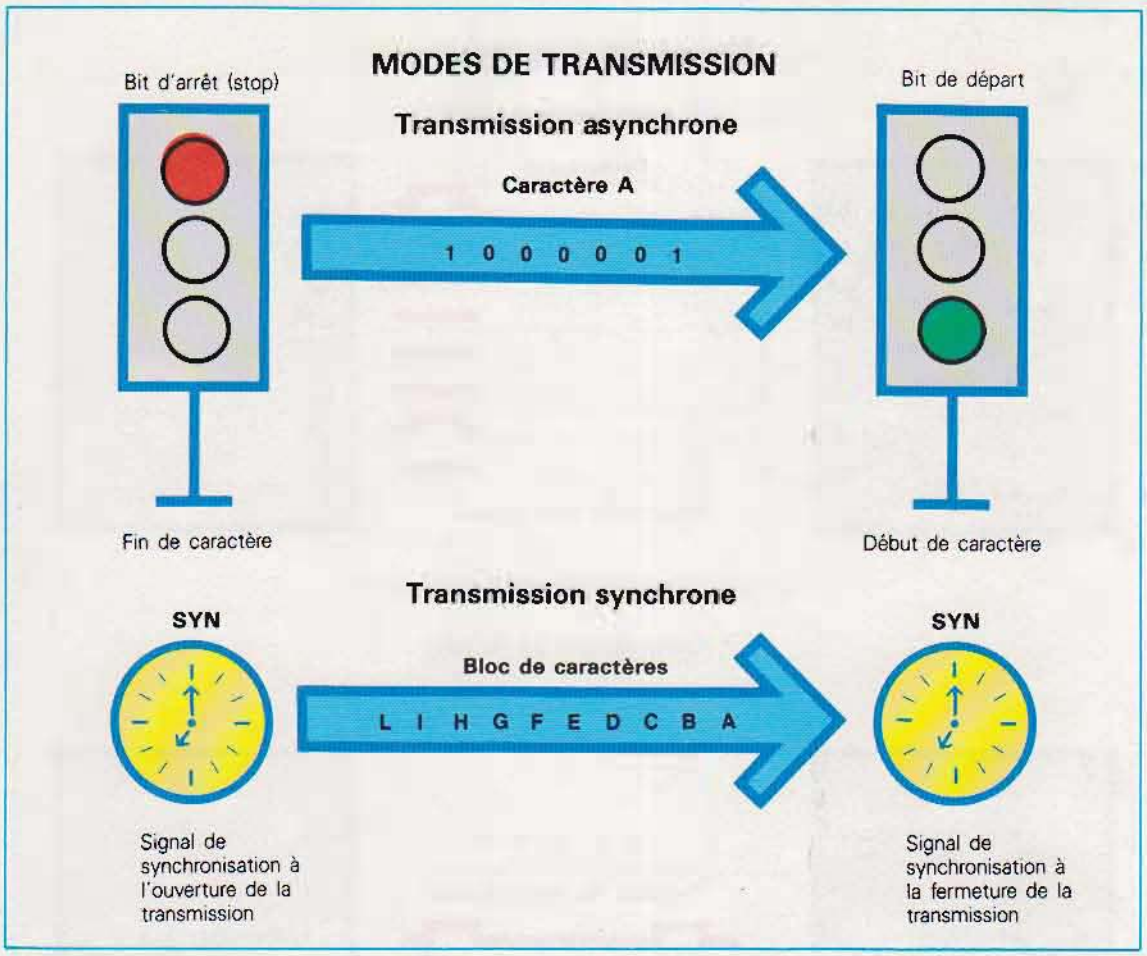
Le bit n° 8 est le bit de parité.



Par la suite, nous traiterons surtout de ce dernier point, étant donnée l'importance pratique considérable de la transmission de données sur réseau téléphonique (réseau «commuté»).
Examinons maintenant une connexion de type série et un autre concept important: le mode de transmission. La télétransmission de données est en général réalisée de deux manières (voir graphiques p. 1320).

- **transmission asynchrone**
- **transmission synchrone**

Le mode asynchrone est souvent défini par START-STOP, car les bits contenant l'information réelle (caractère) sont précédés et suivis par deux bits spéciaux: **le bit de start** (départ), qui prévient qu'un caractère est sur le point de partir et le **bit de stop,** qui signale que la transmission du caractère est achevée.
Le mode synchrone n'exige pas l'emploi de bits de start et de stop, mais il utilise des caractères de synchronisme positionnés en tête et en queue des blocs de caractères à transmettre. Ainsi, la réception d'un nombre de caractères de synchronisme donné informe le récepteur qu'un texte va commencer et qu'il se terminera par autant de caractères de synchronisme, qui viennent donc s'ajouter aux caractères spéciaux de fin de texte.

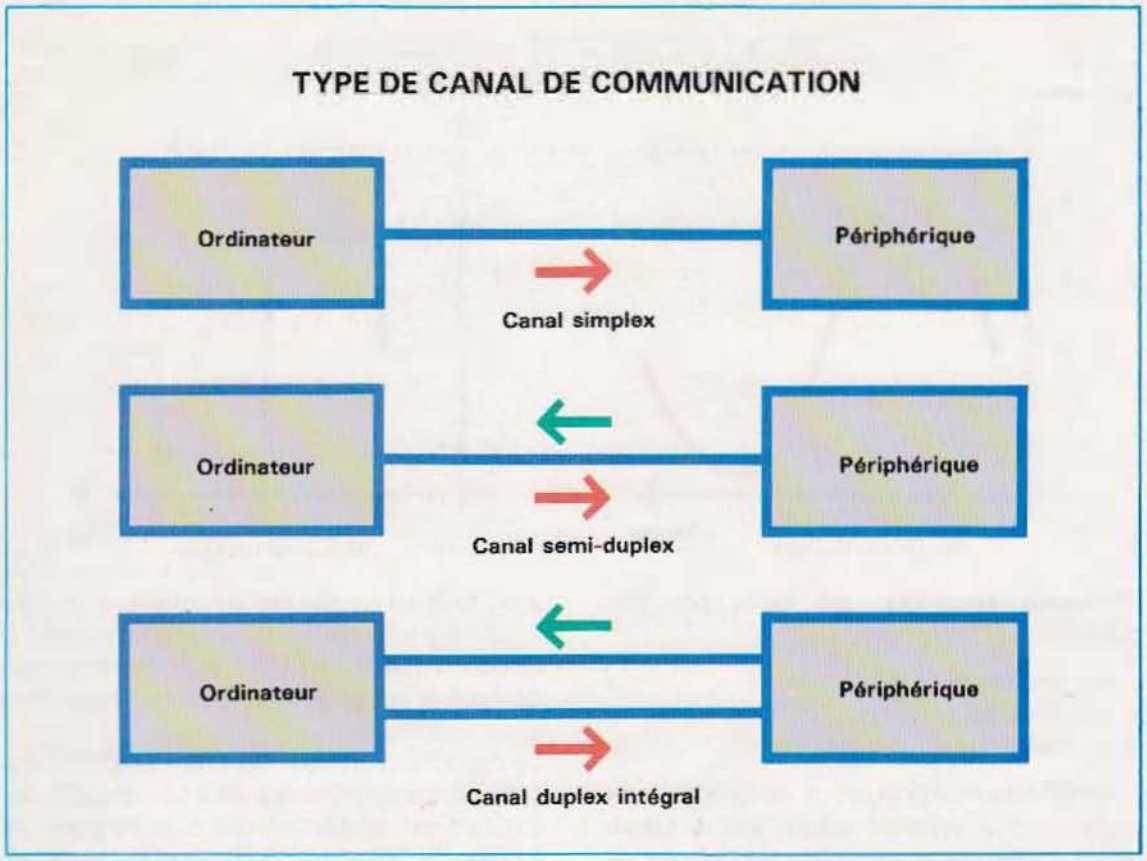


Canaux simplex, demi-duplex, duplex

Le dispositif physique s'appelle **ligne de communication;** si la connexion porte sur une courte distance (quelques mètres) la ligne coïncide physiquement avec le câble de liaison et avec les interfaces de l'ordinateur et du terminal. Pour des connexions éloignées, le mode le plus employé est la ligne téléphonique, avec des instruments spéciaux dont nous parlerons plus loin. Une fois la ligne déterminée, on définit le **canal** d'acheminement dans lequel voyage le flux d'information. Le concept de canal vient du fait que, normalement, dans une ligne, il est possible de faire voyager divers types d'informations et il faut donc en sélectionner le chemin.
Il existe trois types de canaux:

- **simplex** (transmission unidirectionnelle)
- **half-duplex** (transmission à l'alternat ou bidirectionnelle non simultanée)
- **full-duplex** (transmission bidirectionnelle simultanée).

Le canal simplex permet aux informations de passer dans un sens seulement: toujours de l'ordinateur vers le périphérique, ou à l'inverse s'il s'agit d'un périphérique d'entrée. Le canal simplex est en général peu utilisé, car presque toutes les liaisons ordinateur-périphérique ont besoin d'un flux dans les deux sens. Le canal simplex se compare à une voie à sens unique. Dans un canal semi-duplex, en revanche, les échanges d'information se font dans les deux sens, mais pas simultanément; cela signifie qu'à un moment donné, l'information ne circule que dans un sens, qui s'inverse à chaque fois que changent l'émetteur et le récepteur. Le canal semi-duplex est donc comparable à une route à sens unique alterné, gérée par deux feux qui se trouveraient situés aux deux extrémités de la voie.



Dans le **canal duplex** intégral (ou bidirectionnel simultané), les informations voyagent dans les deux sens et en simultané, c'est-à-dire sans qu'il soit nécessaire d'interrompre un sens de transmission. Ce n'est évidemment possible que si le canal est physiquement constitué de deux câbles, un pour chaque sens.
Une question se pose : quelle quantité d'informations est-il possible de transmettre, en une seconde, entre deux dispositifs reliés par un canal à une ligne de communication ?
La réponse dépend du type de liaison (série ou parallèle), des modes de communication (synchrone ou asynchrone) et du type de canal (simplex, semi-duplex ou duplex intégral).
L'unité de mesure de la vitesse de transmission est le **bit par seconde** (**bps**, en abrégé).
Voici quelques ordres de grandeur :

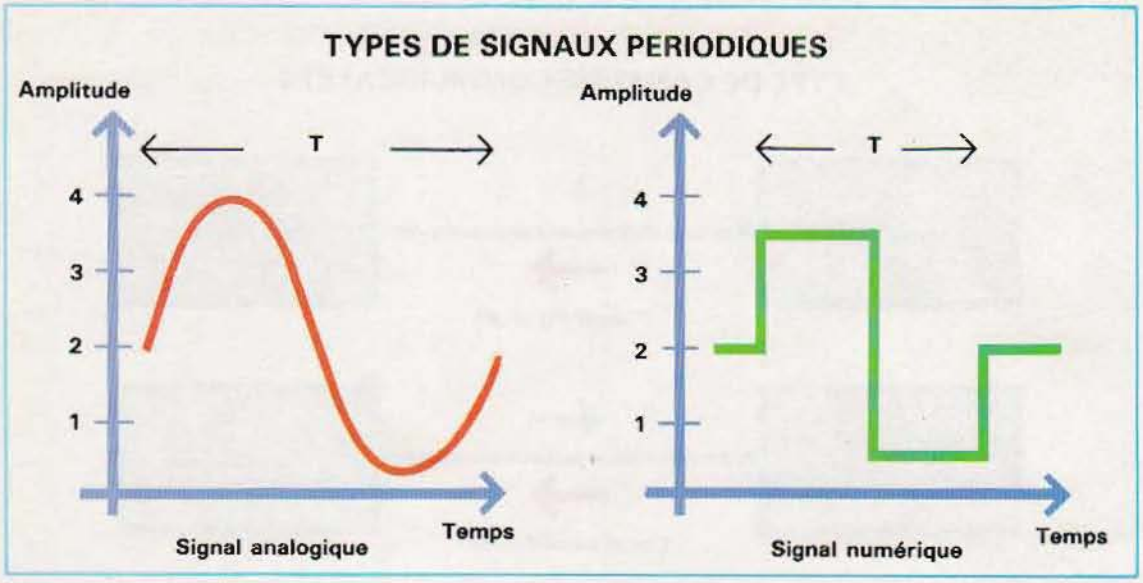
- connexions parallèles : jusqu'à quelques millions de bps
- connexions série : quelques centaines de milliers de bps en mode synchrone ; des dizaines de milliers en mode asynchrone.

Transmission par les lignes téléphoniques

Les liaisons examinées jusqu'ici correspondent à des distances faibles entre les ordinateurs et les périphériques, c'est-à-dire un parcours n'excédant pas 30 à 40 mètres. Au-delà, se posent quelques problèmes au niveau de la propagation des signaux, et il est alors nécessaire de recourir au réseau commuté.

Caractéristiques des signaux

Les signaux transmis par les télécommunications sont dit **périodiques** : ils se répètent tous les temps T (T est la période du signal et se mesure en secondes). Les signaux produits par les ordinateurs sont numérique (rectangulaires) tandis que ceux utilisés pour la transmission d'informations (par radio, systèmes vidéo, téléphone, etc.), sont de type analogique et varient dans le temps mais en continu.



Un signal périodique est défini par trois paramètres :

- son amplitude
- sa fréquence
- sa phase

L'**amplitude** du signal est sa valeur maximale. Le niveau d'un signal périodique analogique varie en continu pour reprendre les mêmes valeurs après chaque période T. A l'inverse, le niveau d'un signal périodique numérique ne varie pas en continu mais reste constant pendant un certain temps, puis change brusquement. Dans le schéma ci-dessus, le niveau du signal numérique est constant pendant T/2 à une valeur de 3,5 avant de tomber à 0,5.

La **fréquence** est le nombre de fois où le signal se répète en une seconde. Un signal qui se répète 5 fois en une seconde a donc une fréquence de 5 hertz. Sa **période** sera égale au 1/5e de seconde (la période est l'inverse de la fréquence).

La **phase** est le retard ou l'avance du signal par rapport à l'instant où commence l'observation ; elle se mesure en fractions de période. Dans le schéma ci-contre, le signal c est dit déphasé en avance, car il semble être parti un quart de période avant le début de l'observation des signaux a et b ; de même, le signal d est déphasé de T/2 en avance car il semble être parti une demi-période avant le repère d'observation.

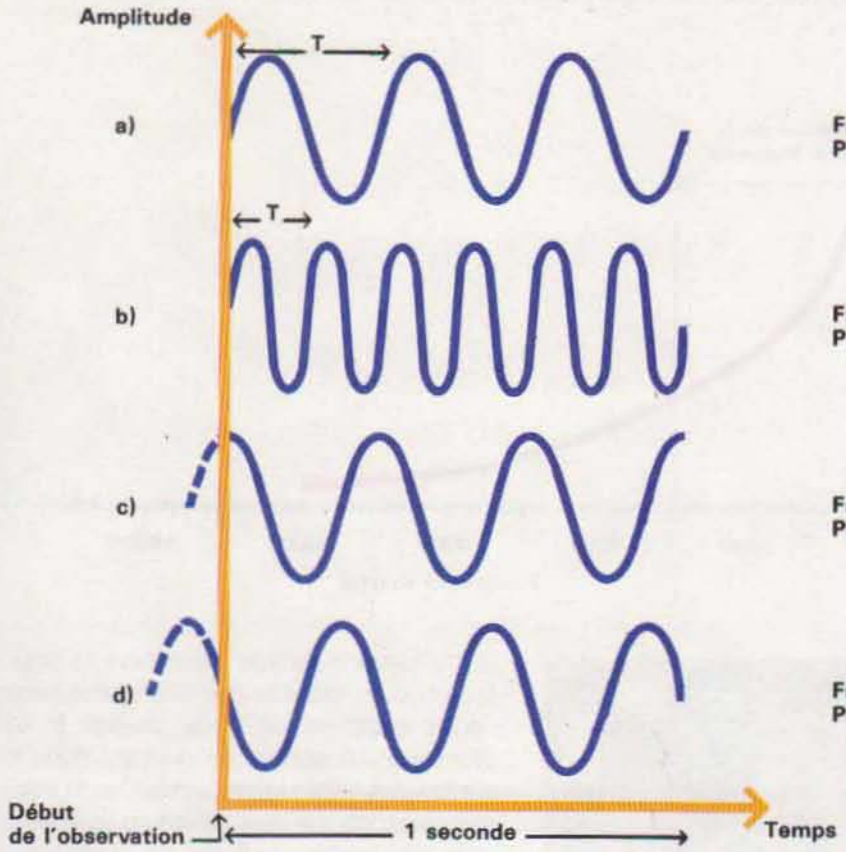
Un signal numérique qui se propage sur une ligne subit des déformations qui augmentent avec la distance parcourue, comme le montre le schéma en page 1323. On peut donc définir des distances limites (30 à 40 mètres) au-delà desquelles le signal n'est plus reconnaissable avec exactitude.

La capacité d'un ligne de communication (un câble) à transmettre des informations non déformées est proportionnelle à la **largeur de bande de fréquences** (ou bande passante). Cette expression désigne la différence, en hertz, entre la limite supérieure et la limite inférieure de fréquence pour lesquelles les signaux ne sont pas déformés. Dire, par exemple, que la bande d'une ligne va de 300 à 3 000 hertz signifie que les signaux périodiques ayant une fréquence comprise entre ces valeurs ne sont pas excessivement déformés. On peut alors affirmer, en extrapolant, que plus la bande d'une ligne est large, moins le signal est déformé, plus la distance de transmission peut être grande.

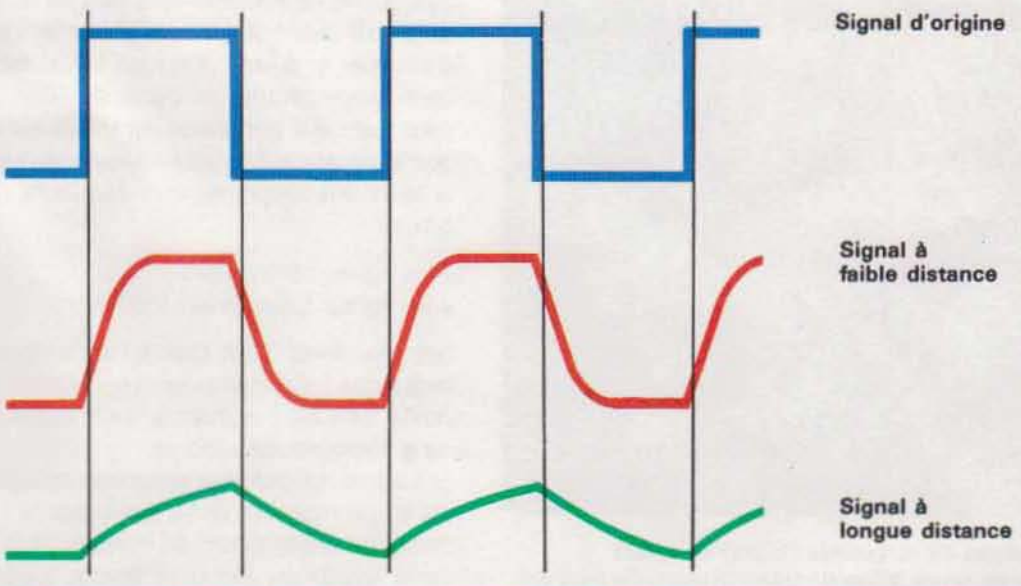
Le choix de techniques spéciales de fabrication des câbles (par exemple leur blindage) permet de décupler la longueur maximale d'une ligne de transmission de signaux numériques pour la porter à 300, voire 400 mètres. Il n'est toutefois pas possible, pour des raisons économiques, de relier de cette façon un ordinateur installé à Paris et un périphérique se trouvant à Lyon. Il faut recourir au réseau commuté.

La structure des lignes téléphoniques est optimisée en fonction des exigences de transmission de la voix humaine : des signaux d'une fréquence située entre 300 et 3 000 Hertz.

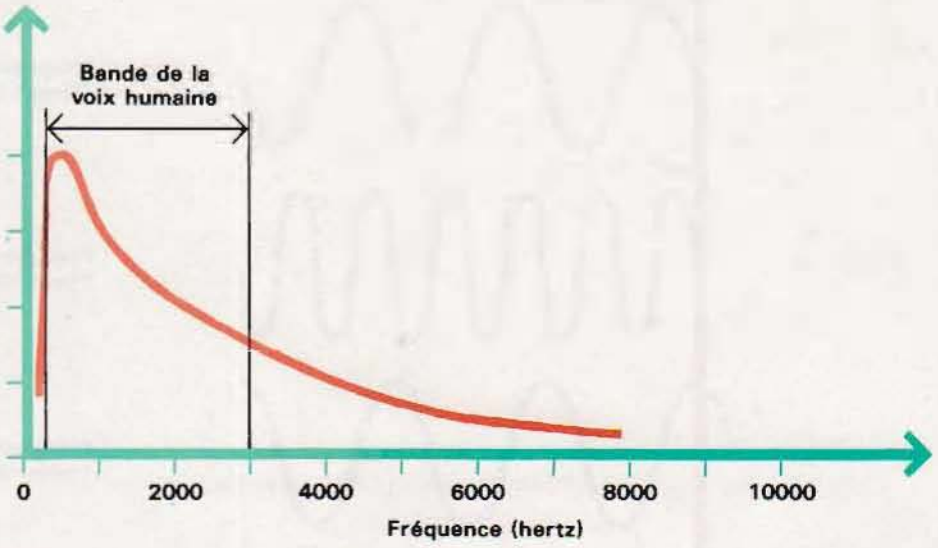
FREQUENCE ET PHASE DES SIGNAUX PERIODIQUES

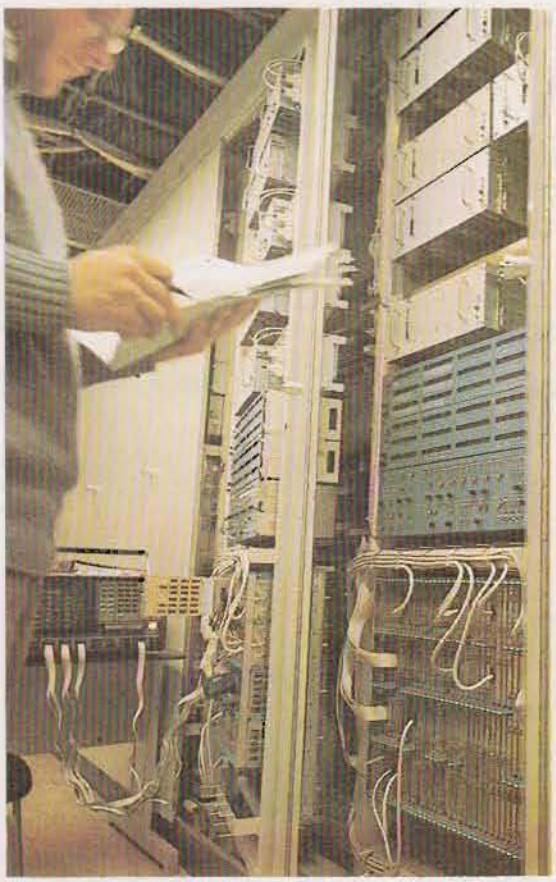


EFFET DEFORMANT (DISTORSION) DE LA DISTANCE SUR UN SIGNAL NUMERIQUE



ATTENUATION DE LA VOIX SUR UNE LIGNE TELEPHONIQUE





Italtel

Central électronique (Italie) assurant les services télex et transmission de donnée.

Cette bande passante est suffisante pour que la voix d'un individu soit clairement perçue à l'autre extrémité de la liaison (voir p. 1324). Les lignes téléphoniques peuvent donc transmettre des informations ; mais les signaux numériques, de par leur structure (différente de celle des signaux vocaux) ne pourraient parcourir de grandes distances sur ces lignes. Il faut donc transformer le signal digital en un signal analogique ayant une fréquence comprise dans la bande passante. Pour cela, il faut utiliser des instruments spéciaux de conversion du signal avant son passage sur la ligne téléphonique et à sa réception : les **modems.** Leur usage permet en outre d'établir des liaisons avec des périphériques situés à n'importe quelle distance. Les lignes téléphoniques, pour la télétransmission de données, sont de deux types :

- les lignes commutées
- les lignes spécialisées (ou louées)

Les premières sont des lignes téléphoniques ordinaires : on compose un numéro sur un poste, et c'est l'ordinateur avec lequel on veut communiquer qui répond.
Les lignes spécialisées assurent une liaison stable et permanente entre deux points ; il n'y a donc pas à composer de numéro et la fiabilité de la liaison est indubitablement supérieure à

celles commutées, qui subissent les perturbations de commutations normales.

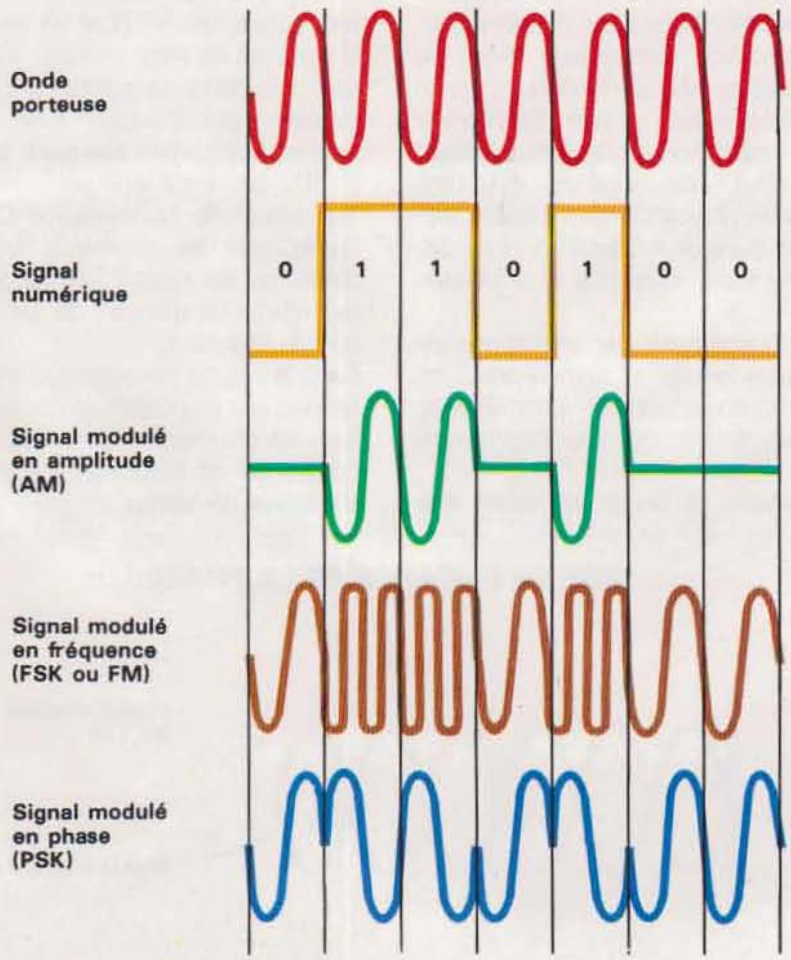
Techniques de modulation. Les modems

Les conversions numérique-analogique et analogique-numérique sont appelées modulation et démodulation et sont effectuées à l'aide de modems, contraction de MOdulateur et DEModulateur. La modulation transforme un signal numérique en un signal analogique dans la bande de fréquence d'une communication téléphonique (sans distorsion). La démodulation, elle, rétablit le signal numérique d'origine. Il existe plusieurs techniques de modulation (schémas ci-dessous) :

- modulation d'amplitude
- modulation de fréquence
- modulation de phase

Dans les trois cas, on fait appel à une onde de référence analogique dite **porteuse**, produite par le circuit interne du modem.
Avec la modulation d'amplitude (symbolisée par AM), la porteuse est représentée sur la ligne sous forme d'état logique 1 ou 0. La transmission consécutive de deux états logiques 1 entraîne la présence de l'onde porteuse sur la ligne pendant un temps plus long que la transmission d'un seul état 1. Ce phénomène implique que, pour pouvoir noter correctement les temps correspondant aux états logiques, il faut utiliser des circuits d'horloge.

TECHNIQUES DE MODULATION



La modulation de fréquence est généralement désignée par les abréviations FSK (Frequency Shift Keying — Saut de fréquence) ou FM (Frequency Modulation). Elle fait intervenir deux fréquences : celle de l'onde porteuse pour un état logique donné (par exemple l'état 0) ainsi qu'une fréquence multiple de la porteuse pour l'état opposé (l'état 1). L'amplitude du signal est la même pour les deux fréquences comme pour les deux états logiques représentés. Dans la technique de modulation de phase, ce sont l'amplitude et la fréquence du signal qui restent constants tandis que la phase est décalée de + ou - T/2 (voir ci-dessous l'exemple d'un signal de phase 0 et un autre déphasé de T/2).

La démodulation, c'est-à-dire la reconstitution du signal numérique d'origine, utilise des techniques différentes. Les modems contiennent généralement à la fois les circuits d'émission de signaux numériques reçus. Ils sont installés aux deux extrémités d'une ligne de communication et doivent être compatibles, sinon du même type. Cela signifie qu'ils doivent fonctionner à la même vitesse et selon les mêmes techniques de modulation et de démodulation. Il existe, en effet, des appareils qui, bien qu'ayant la même vitesse de transmission, utilisent des techniques de modulation et de démodulation différentes. Ils sont alors dits **équivalents.**

Les modems varient aussi par leur mode de transmission : synchrones et asynchrones, les vitesses de communication les plus élevées étant généralement obtenues avec la première technique.

Dernière différence : la bande de travail des modems. On trouve en effet les 3 types suivants, auxquels correspond la classification page ci-contre :

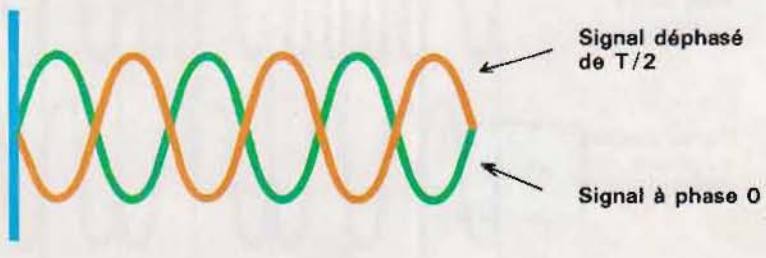
— modems opérant en-dessous de la bande vocale (jusqu'à 300 Hz). La vitesse de transmission ne dépasse pas 150 bps.
— modems opérant dans la bande vocale (300 à 3 000 Hz). La vitesse est de 1 200 à 2 400 bps pour les modems asynchrones et de 2 400 à 19 200 bps pour les modems synchrones.
— modems à large bande (au-dessus de 3 000 Hz). Si les vitesses sont supérieures, les techniques de modulation employées sont très perfectionnées et donc forcément coûteuses.

Configuration des liaisons

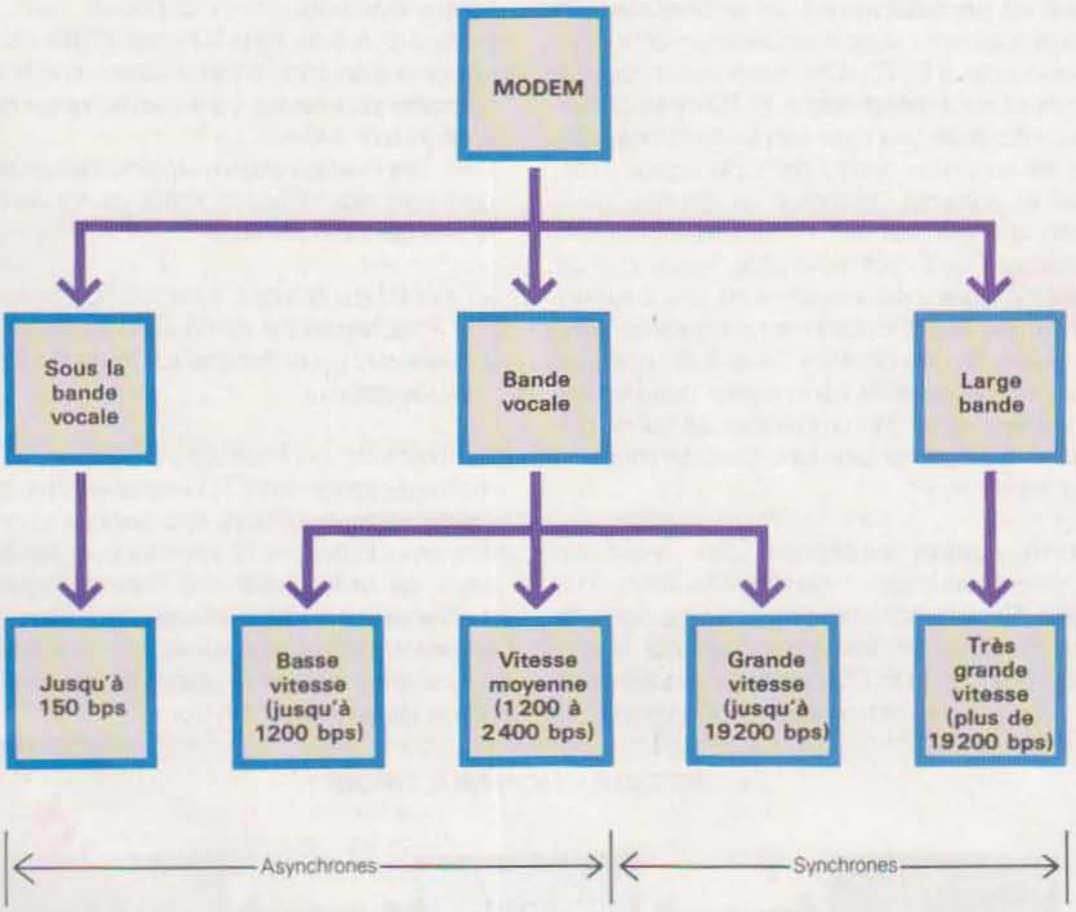
La classification typologique des liaisons que nous allons maintenant décrire est valable pour les courtes comme pour les longues distances, donc avec et sans modem. Cette généralisation nécessite toutefois l'introduction d'une certaine terminologie. Ainsi, les appareils connectables sont désignés par l'abréviation ETTD, qui signifie Equipement Terminal de Traitement de Données (ou DTE en anglais), tandis que les dispositifs utilisés pour les connecter sont des ETCD (DCE en anglais), ce qui signifie Equipement de Terminaison de Circuit de Données.

Dans le cas de communications à longues distances, ces dispositifs seront les modems et la ligne téléphonique : dans le cas de communications à faibles distances, il s'agira simplement de câbles de liaison.

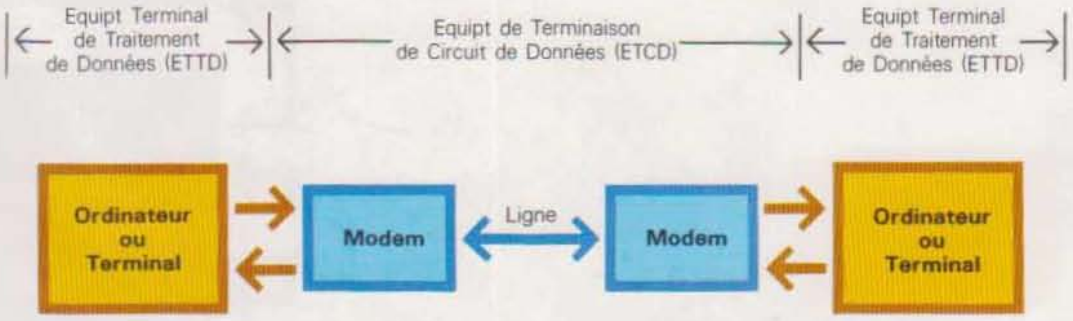
MISE EN EVIDENCE DE LA PHASE



CLASSIFICATION DES MODEMS



CONFIGURATION POINT A POINT



Configuration point à point. On retrouvera cette terminologie dans le graphique en bas de la page 1327, qui représente une configuration dite point à point, c'est-à-dire la **liaison directe d'un périphérique à un ordinateur.** Elle utilise une ligne de communication pour chaque couple d'ETTD. Une autre liaison point à point entre un périphérique ETTD et un ordinateur nécessite une ligne supplémentaire ETCD. La configuration point à point est simple à réaliser et présente l'avantage de disposer de la ligne en permanence : l'inconvénient est évidemment qu'il faut autant de lignes que de périphériques à connecter, d'où une augmentation des coûts, notamment dans les communications longue distance. Mais il est possible, pour des vitesses de transmission peu élevées, d'utiliser des lignes commutées (ce qui ne dispensera pas de prévoir une paire de modems par ligne).

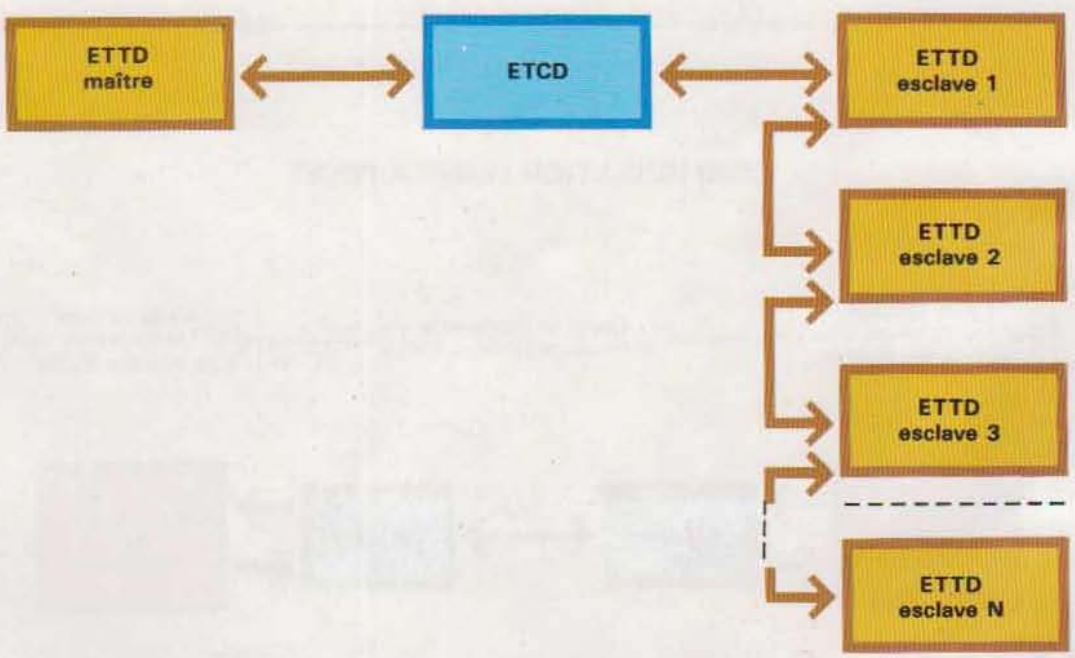
Configuration multipoint. Son atout est d'ordre économique : elle réduit les coûts de la ligne. Dans le schéma ci-dessous, la ligne de communication et les modems ont été regroupés dans le bloc ETCD tandis que les éléments à connecter — terminaux ou ordinateurs — sont dans les blocs ETTD. A une extrémité de la ligne se trouve un type particulier d'ETTD, l'**ETTD-maître**, qui commande le dialogue sur la ligne. En position d'**ETTD-esclaves**, à l'autre extrémité, les N dispositifs sont tous reliés à la même ligne. Chaque ETTD esclave est doté d'un code d'identification que le maître utilise pour savoir qui transmet ou qui reçoit à un instant donné.

Dans une configuration mulpoint, l'échange est géré par deux tâches directes, toutes deux pilotées par l'ETTD-maître :

- l'invitation à émettre, ou scrutation (polling) : la ligne reçoit une demande d'émission,
- la sélection : un terminal est invité à recevoir un message.

Au cours de la phase de scrutation, l'ETTD-maître interroge les ETTD-esclaves l'un après l'autre pour savoir s'ils ont quelque chose à émettre. Un esclave ne peut envoyer son message qu'après avoir été interrogé par le maître ; s'il n'a rien à envoyer, le maître pose la question à l'esclave suivant jusqu'au dernier, puis il recommence à partir du premier. La phase de scrutation est donc cyclique.

CONFIGURATION MULTIPOINT



Commodore

Configuration complète à base de Commodore 64.

Un esclave émetteur est reconnu grâce à son code d'identification, inclus dans le message émis. En phase de sélection, l'ETTD-maître adresse un esclave en l'informant qu'un message lui a été envoyé.

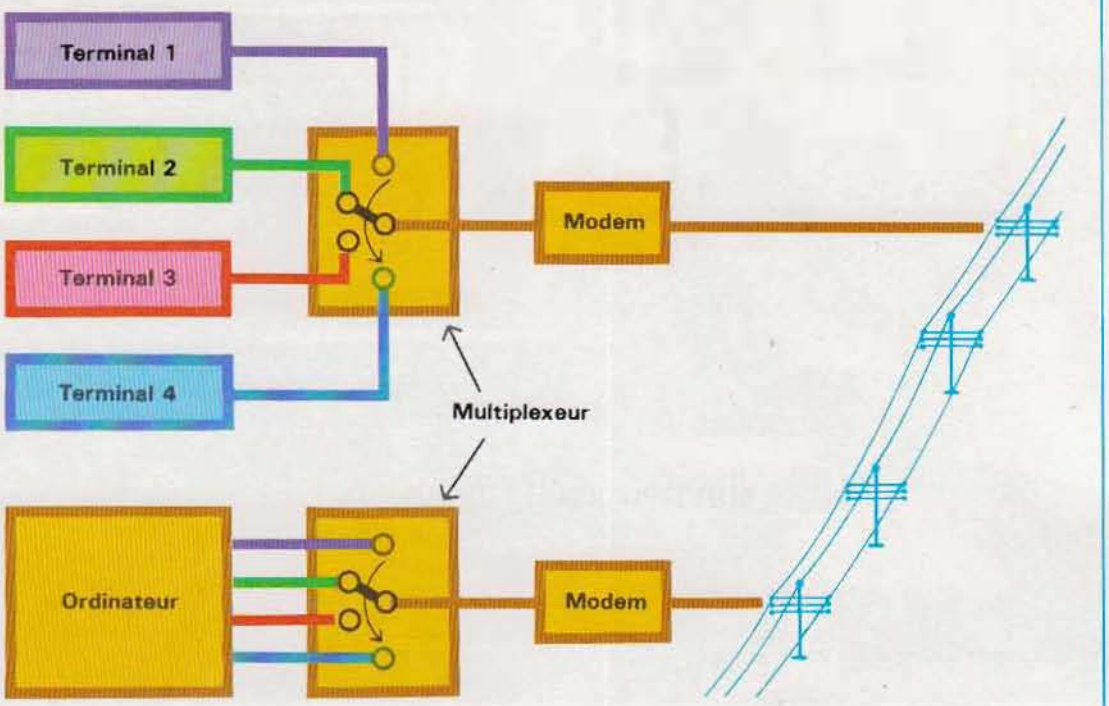
Prenons un exemple dans lequel le maître est un ordinateur et les esclaves N terminaux. Ordinateur et terminaux sont dotés d'interfaces pour gérer l'échange en mode multipoint. De la machine part un câble (abstraction faite du modem) auquel sont connectés tous les périphériques (une seule ligne de communication). L'unité centrale effectue cycliquement la tâche de scrutation au cours de laquelle les terminaux recoivent, l'un après l'autre, l'autorisation d'émettre.

Un terminal émetteur (ou un opérateur en saisie de données au clavier) est identifié par son code tandis que l'ordinateur accuse réception du message qui lui est envoyé.

Imaginons que ce dernier doive être affiché sur un autre terminal appartenant à la même chaîne multipoint. L'ordinateur déclenche alors la phase de sélection et envoie sur la ligne le contenu de la mémoire-tampon (dans laquelle se trouve le code du dispositif-receveur). Le terminal destinataire reçoit alors les données et les affiches. Le dialogue entre ETTD n'est pas totalement asynchrone, ce qui pose des problèmes d'attente pendant la phase d'utilisation de la ligne. Dans notre exemple, un terminal ne peut émettre qu'une fois la sélection terminée. La ligne risque donc de devenir le « goulot d'étranglement » du système. Ce problème oblige à trouver un bon compromis entre le nombre d'ETTD à relier au système et la quantité d'informations à transmettre par unité de temps.

Configuration multiplex. Il s'agit là d'une autre réponse à la gestion de l'attente : la configuration à base de multiplexeur, tout en réduisant au minimum le nombre de lignes, règle la question du volume d'informations à transmettre. En effet, le multiplexeur est tout désigné pour les situations caractérisées par une **voie unique de données à fort débit** et par plusieurs ETTD interconnectés. C'est le cas, par exemple, d'un ordinateur devant communiquer avec 4 terminaux sur une seule ligne (schéma p. 1330).

CONFIGURATION A BASE DE MULTIPLEXEUR



Le multiplexeur est donc un dispositif qui évite la confusion entre les signaux d'origine différente présents sur la ligne et ce grâce à plusieurs méthodes.

Le multiplexeur temporel alloue un intervalle de temps fixe à chacun des terminaux émetteurs et reconstitue le message à l'autre bout de la ligne, selon la même périodicité. Dans notre exemple, le partage du temps est simple : à l'instant t_1, la ligne 1 envoie le caractère A, à l'instant t_2, la ligne 4 envoie le caractère B... (voir graphique 1 page ci-contre).

Le multiplexeur fréquentiel alloue une bande de fréquences à chaque canal de données. Le principe est celui de la transmission radio : différents émetteurs envoient simultanément des messages sur différentes fréquences tandis que le poste récepteur ne capte que l'émeteur sur lequel il est réglé (graphique central page ci-contre).

Dans notre exemple, chaque terminal dispose d'une certaine bande de fréquences d'émission et l'ordinateur à 4 points de synchronisation (réglage des fréquences à la réception).

Le multiplexeur temporel est comparable à un commutateur à grande vitesse : son débit dépend étroitement de ses caractéristiques physiques : vitesses de transmission (en bps) et nombre de portes disponibles. Par exemple, un multiplexeur à 38 400 bps et 16 portes peut faire communiquer 16 terminaux à 2 400 bauds* chacun. Si les terminaux sont reliés à distance, la ligne téléphonique et les modems employés doivent être capables de supporter les 38 400 bps.

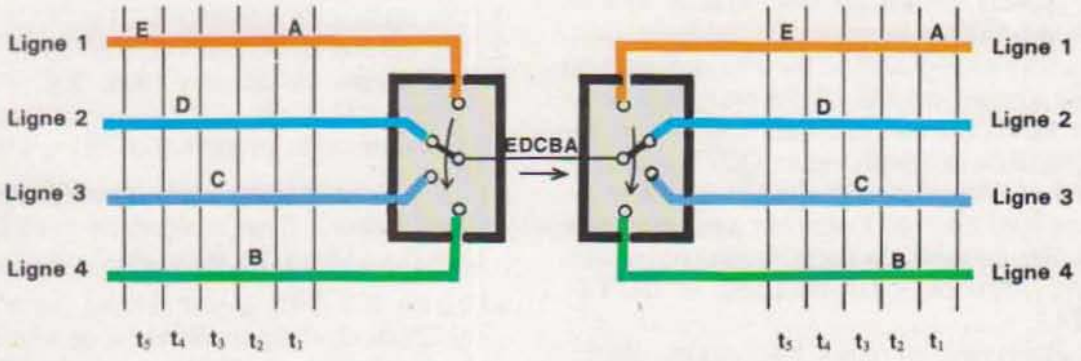
En revanche, la capacité de transmission d'un multiplexeur fréquentiel se mesure en termes de bande fréquence totale : plus la bande est large, plus le nombre de dispositifs pouvant communiquer sera élevé.

Interfaces de communication

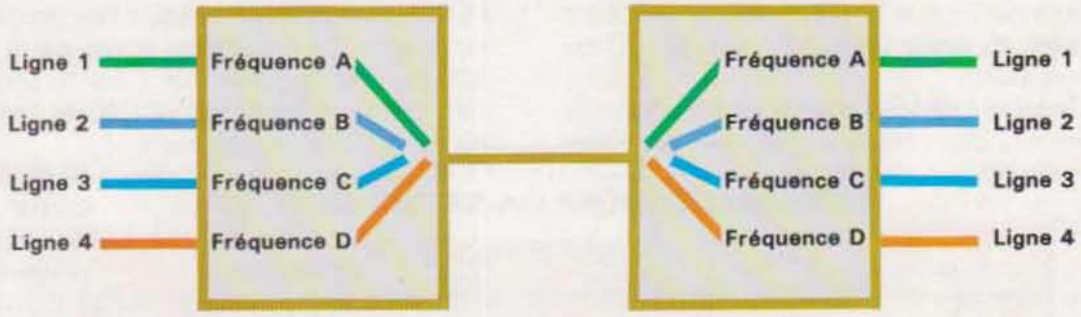
Il nous faut compléter la terminologie ETTD/ETCD par le concept d'**interface**. En télétransmission, ce terme désigne un dispositif assurant la liaison entre ETTD et ETCD

*Le baud est l'unité de vitesse de modulation. Elle est identique à la vitesse de transmission en bits par seconde quand la liaison est en binaire à deux niveaux. Dérivé de Baudot (nom d'un pionnier de l'informatique).

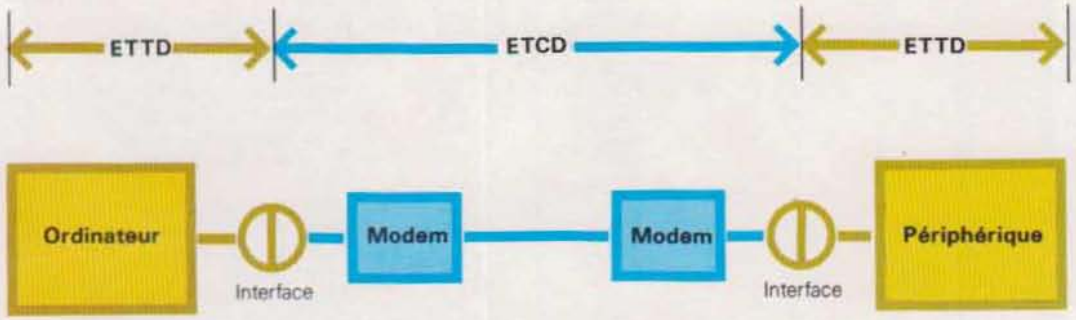
MULTIPLEXEUR TEMPOREL



MULTIPLEXEUR FREQUENTIEL



LIAISON POINT A POINT AVEC INDICATION DES INTERFACES



(schéma du bas, page 1331). Les interfaces de communication servent surtout à rendre compatibles les connexions; pour cela, elles doivent respecter des recommandations précises sur les aspects mécaniques, électriques et structurels des liaisons.

Les normes régissant la transmission avec ou sans modem ont été émises pour deux organismes : le Comité Consultatif International Télégraphique et Téléphonique (CCITT) pour l'Europe et l'Association des Industries Electroniques (EIA) pour les Etats-Unis. Les protocoles les plus courants de transmission par modem sont référencés : **EIA-RS232C** et **CCITT-V24.**

Ils établissent les règles mécaniques, électriques et de structure des circuits à respecter par les constructeurs.

S'agissant des circuits, ces normes régissent notamment la gestion des signaux le long de la ligne de communication. Le tableau ci-dessous donne la nomenclature officielle des circuits établie par l'EIA et le CCITT, tout en précisant le sens du signal (de l'ETTD vers l'ETCD ou inversement).

Le recours à certains signaux est lié au type de communication établie (synchrone, asynchrone, semi-duplex, ou duplex intégral) et donc au type de modem. Les signaux se subdivisent en 4 groupes ; nous verrons ensuite les plus fréquents :

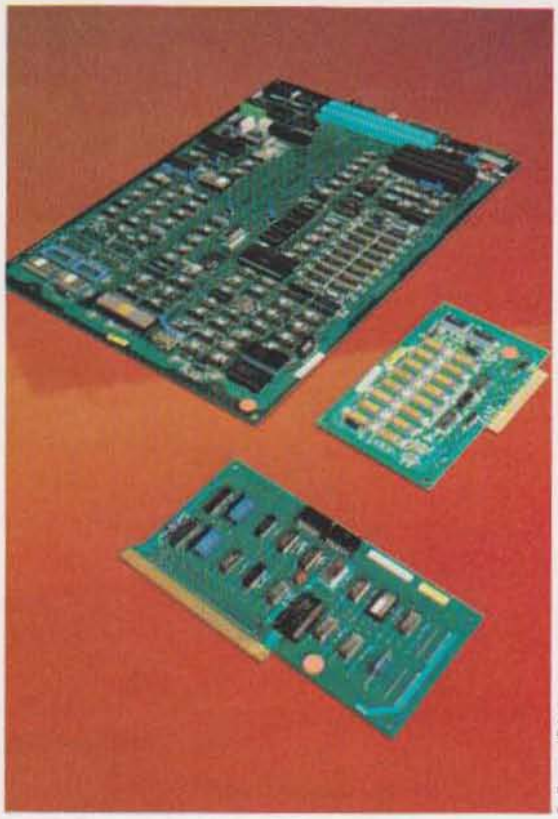
1 / Les signaux de référence (101 et 102)
2 / Les signaux de données (103 et 104)
3 / Les signaux de contrôle (105, 106, 107, 108.2, 109, 125)
4/ Les signaux de temporisation (114, 115)

- Circuit 101 (AA) : Protective Ground (terre de protection). C'est le signal de mise à la terre des châssis des appareils.
- Circuits 102 (AB): Signal Ground (terre de signalisation). C'est la référence de tous les signaux émis par un ETTD ou un ETCD (à l'exclusion du 101). Les circuits 101 et 102 sont parfois interconnectés pour réduire les perturbations.
- Circuit 103 (BA) : Transmitted Data (Emission de données de l'ETTD vers l'ETCD). Il supporte l'émission des données sur la ligne. Ce circuit n'est activé que si les 105 (Request to Send), 106 (Clear to Send), 107 (Data Set Ready) et 108.2 (Data Terminal Ready) le sont également.

RECOMMANDATIONS EIA-RS232C CCITT-V24

Broche	Vers ETTD	Vers ETCD	Circuit RS232C	Circuit V24	Description
1			AA	101	Protective Ground
2	→		BA	103	Transmitted Data
3	←		BB	104	Received Data
4	→		CA	105	Request to Send
5	←		CB	106	Clear to Send
6	←		CC	107	Data Set Ready
7	↔		AB	102	Signal Ground
8	←		CF	109	Detector for Signal Line Received (Carrier Detector)
9			—	—	
10			—	—	
11			—	—	
12	←		SCF	122	Detector for Secondary Signal Line Received
13	←		SCB	121	Secondary Clear to Send
14	→		SBA	118	Secondary Transmitted Data
15	←		DB	114	Transmission of Signal Timing
16	←		SBB	119	Secondary Received Data
17	←		DD	115	Receive of Signal Timing
18			—	—	
19	→		SCA	120	Secondary Request to Send
20	→		CD	108.2	Data Terminal Ready
21	←		CG	110	Signal Quality Detector
22	←		CE	125	Ring Indicator
23	→		CH/CI	111/112	Selector of Data Signal Rate
24	→		DA	113	Transmission of Signal Timing
25			—	—	

- Circuit 104 (BB) : Received Data (réception par l'ETTD de données en provenance de l'ETCD). Supporte les données à envoyer à l'ETTD. Il est ouvert (ON) ou fermé (OFF) en fonction de l'état du circuit 109 (Carrier Detector).
- Circuit 105 (CA) : Request to Send (demande d'émission d'ETTD vers ETCD). Indique que l'ETTD désire transmettre et prépare en conséquence l'ETCD (le modem émet l'onde porteuse).
- Circuit 106 (CB) : Clear to Send (prêt à émettre — signal émis par l'ETCD à destination de l'ETTD). Indique que l'ETCD est prêt à transmettre. Le signal est activé par l'ETCD à réception de la demande d'émission. En pratique, le 106 suit le 105 avec un décalage dû au retard initial nécessaire à la « préparation » de l'ETCD.
- Circuit 107 (CC) : Data Set Ready (poste de données prêt — signal émis par l'ETCD vers l'ETTD). L'état ON (ouvert) indique que l'ETCD est relié à la ligne, prêt à émettre ou à recevoir. Pour que des signaux soient émis sur la ligne, il faut que le 107 soit ON. En fait, le seul signal qui puisse opérer sans que le 107 soit ouvert est le 125 (Ring indicator).
- Circuit 108.2 (CD) : Data Terminal Ready (terminal de données prêt — signal émis par l'ETTD vers l'ETCD). A l'ouverture de ce circuit, l'ETTD informe l'ETCD qu'il est prêt à émettre et à recevoir. Dès que l'ETTD fait passer le 108 à ON, l'ETCD fait de même avec le 107. Les deux signaux émis par ces circuits suivant donc le même schèma d'activation à quelques intervalles près.
- Circuit 109 (CF) : Carrier Detector (détecteur du signal de ligne reçu sur la voie de données — émis par l'ETCD vers l'ETTD). Le niveau du signal sur la ligne est conforme aux spécifications. Si le circuit passe à l'état OFF, la réception est interrompue.
- Circuit 125 (CE) : Ring Indicator (indicateur d'appel — signal émis par l'ETCD vers l'ETTD). A l'état ouvert, il signale l'arrivée d'un appel et établit alors la liaison.
- Circuit 114 (DB) : Transmission of Signal Timing (base de temps pour les éléments du signal à l'émission — signal émis par l'ETCD à destination de l'ETTD). Utilisé par l'ETCD (modem) pour temporiser l'ETTD (ordinateur ou terminal à l'émission.
- Circuit 115 (DD) : Receive of Signal Timing (base de temps pour les éléments du signal à la réceptions — signal émis par l'ETCD vers l'ETTD). Utilisé par l'ETCD (modem) pour temporiser l'ETTD à la réception. L'état OFF du 115 dépend de l'état du 109 : si ce dernier est OFF, le circuit 115 l'est aussi.



C. Pozzoni/Marka

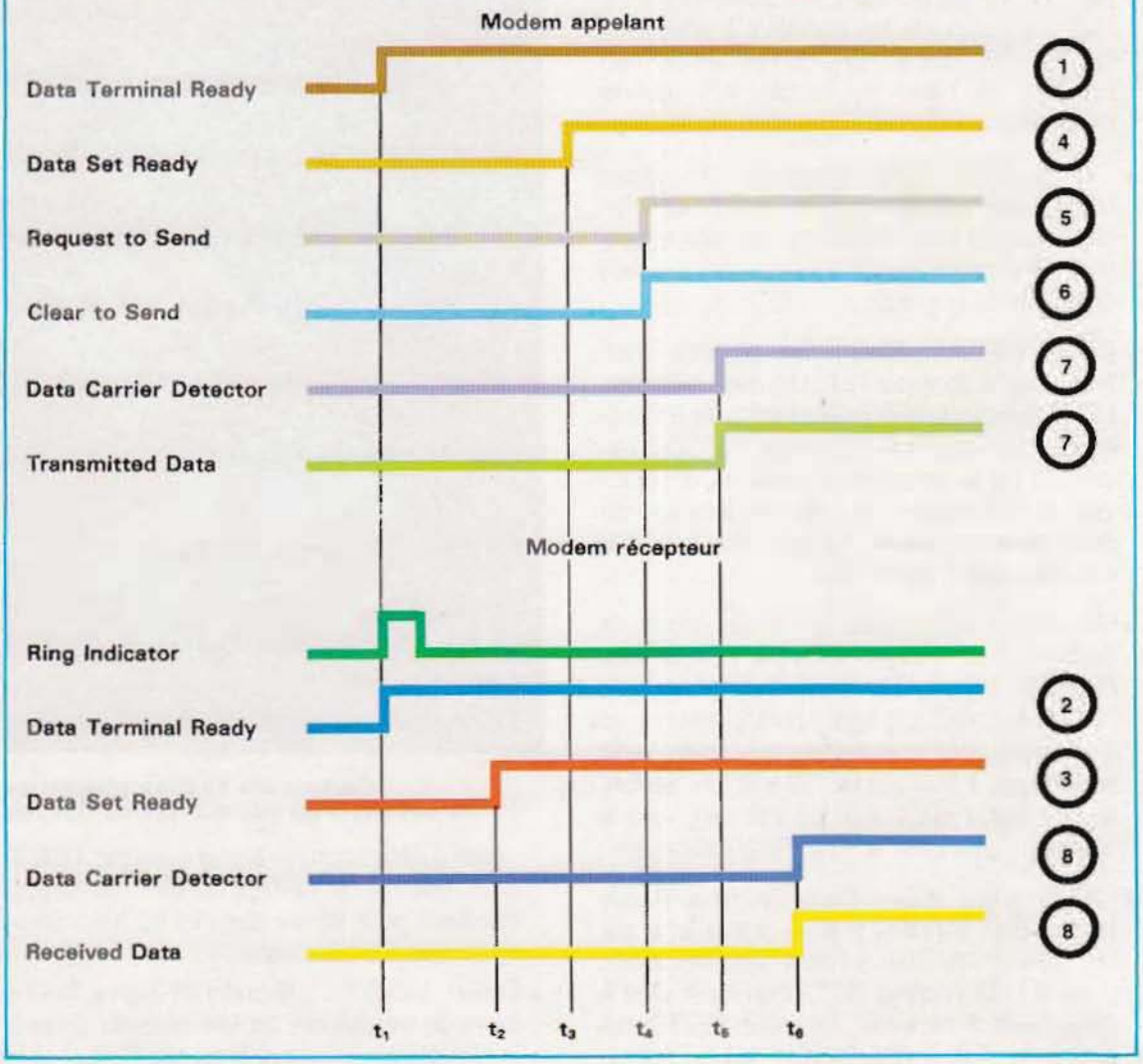
Carte-mère et deux plaquettes de circuits d'un micro-système Olivetti.

Pour illustrer l'utilisation de ces signaux, examinons la séquence de transmission de données entre deux modems à distance installés sur une ligne téléphonique (voir p. 1334).

1 / Le signal Data Terminal Ready est actif du côté du modem appelant et un opérateur compose le numéro de téléphone du modem récepteur.

2 / Le modem récepteur active le Ring Indicator. Le signal Data Terminal Ready est également actif du côté réception, donc l'ETCD connecté est prêt (terminal en marche).

LIAISON ET TRANSMISSION DE DONNEES ENTRE DEUX MODEMS



3 / A l'instant t_2, le modem récepteur active le Data Set Ready et l'opérateur appelant reçoit un signal acoustique indiquant que la liaison est établie.

4 / L'opérateur appelant actionne un bouton du modem et raccroche. Il active ainsi le Data Set Ready de son côté, ce qui indique que le modem appelant est également prêt (instant t_3).

5 / L'ETTD appelant active le signal Request to Send pour informer le modem qu'il veut commencer à émettre.

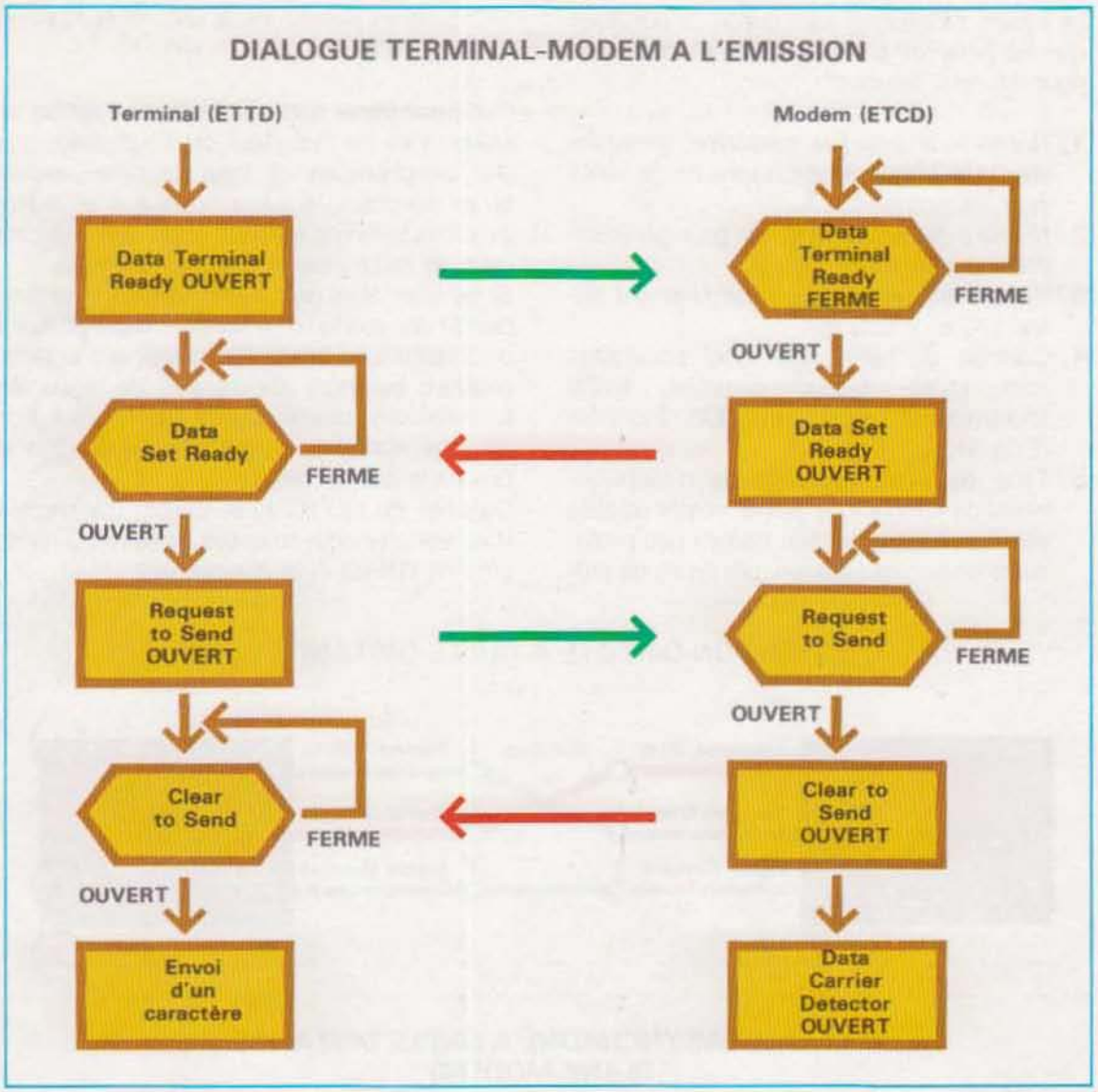
6 / Le modem active le signal Clear to Send pour informer l'ETTD que tout est prêt pour l'émission (instant t_4).

7 / L'émission commence (instant t_5) ; le signal Carrier Detector est actif pour contrôler le niveau des informations transmises.

8 / Les données arrivent au point de réception (t_6) ; le Carrier Detector est également actif du côté de l'ETTD récepteur.

Dans cet exemple, les intervalles de temps sont donnés à titre indicatif et ne tiennent pas compte des véritables caractéristiques des modems et des lignes.

L'organigramme du dialogue entre le terminal émetteur (ETTD) et le modem (ETCD) (voir page ci-contre) fait apparaître les relations de cause à effet entre les différences phases de la session.



Communications à faibles distances

La standardisation des interfaces de communication ne prend pas en compte la distance de transmission. Les normes de l'EIA et du CCITT restent donc valables pour les liaisons sans modem. Les constructeurs incluent eux-mêmes, dans le matériel de l'ordinateur, les interfaces nécessaires à la communication avec l'extérieur selon les normes internationales. Il en est de même des périphériques. Le problème qui se pose est généralement le suivant : un terminal peut-il dialoguer sans modem avec un ordinateur à N interfaces capables de piloter N dispositifs aux normes EIA ou CCITT ?

Supposons que la communication à établir soit du type asynchrone. Si l'interface de l'ordinateur a été conçue pour ne travailler que sur de faibles distances (jusqu'à 15 mètres d'après la norme série RS232C), il n'en sortira généralement que trois fils : **Transmitted Data**, **Received Data**, **Signal Ground**.

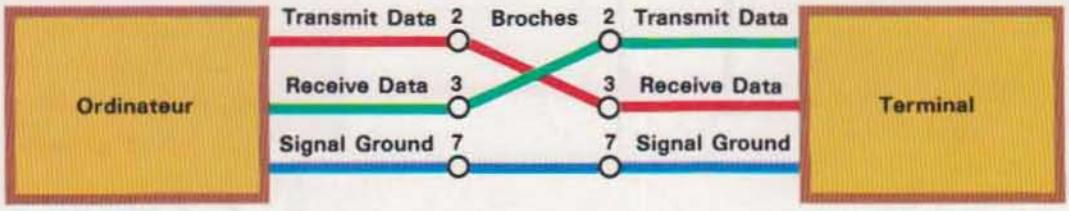
Si l'interface du terminal a été conçue selon les mêmes spécifications, la liaison est simple : il suffit de relier le circuit Transmitted Data de l'ordinateur à celui du Received Data du terminal et vice-versa, comme indiqué page 1336. Dans ce cas (schéma central), aucun des autres signaux prévus par les recommandations de l'EIA n'est utilisé.

La liaison fonctionne telle quelle, à condition que les paramètres suivants soient identiques pour les deux dispositifs :

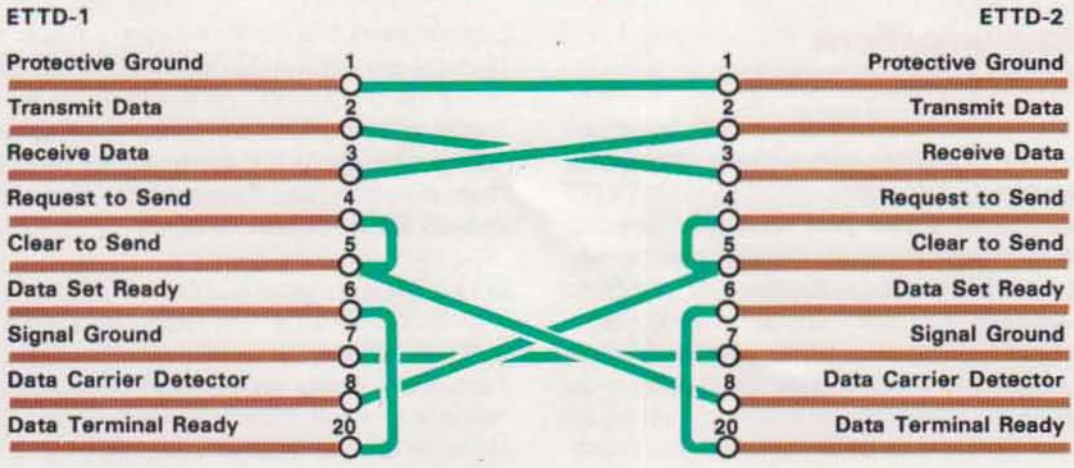
1 / Nombre de bits par caractère, généralement de 7 ou 8, avec ou sans bit de parité (voir tableau p. 1317).
2 / Nombre de bits de stop. On peut généralement choisir entre 1 ou 2.
3 / Vitesse de transmission, normalement entre 110 et 9 600 bps.
4 / Contrôle de parité. Les trois possibilités sont NONE (aucun contrôle), EVEN (contrôle de parité), et ODD (contrôle d'imparité).
5 / Type de synchronisation ou d'établissement de liaison. Ces points seront étudiés plus loin dans la section traitant des protocoles de communication. Les types de protocole les plus courants sont XON/XOFF et ENQ/ACK.

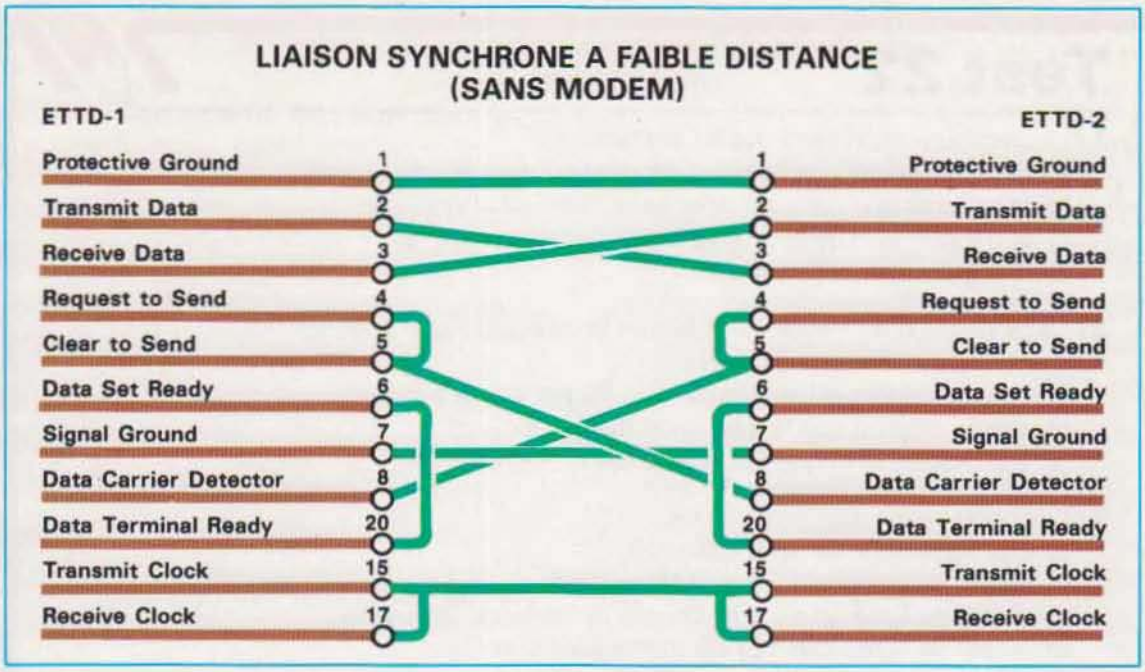
Ces paramètres sont généralement définis au niveau tant de l'interface de l'ordinateur que des périphériques. Il peut toutefois arriver qu'un périphérique soit « rigide », c'est-à-dire avec des paramètres fixes. Il faut alors adapter ceux de l'ordinateur, qui est plus souple.
Si les interfaces de communication série emploient des modems, la liaison à faible distance est établie à l'aide de câbles spéciaux appelés **modem by-pass** (éliminateurs de modems). Le schéma du bas de page présente une liaison de type asynchrone en faisant apparaître le brochage des circuits.
Du côté de l'ETTD-1, le circuit Transmitted Data est connecté au circuit Received Data du côté de l'ETTD-2, et inversement.

LIAISON DIRECTE A FAIBLE DISTANCE



LIAISON ASYNCHRONE A FAIBLE DISTANCE (SANS MODEM)





La transmission n'a lieu que si les signaux Request to Send, Clear to Send, Data Set Ready et Data Terminal Ready sont actifs ; les signaux Request to Send et Data Terminal Ready sont activés par l'ETTD (ordinateur ou terminal), tandis que Data Set Ready et Clear to Send le sont par le modem (qui n'est pas représenté).

Les connexions entre Request to Send/Clear to Send et Data Terminal Ready/Data Set Ready permettent d'activer les quatre signaux nécessaires au lancement de la transmission (autrement dit, celui qui commence la communication se répond tout seul). Pour la réception, le signal Carrier Detector doit être activé, ce qui s'obtient en reliant le circuit Request to Send de l'émetteur au circuit Carrier Detector du récepteur (autrement dit, celui qui désire émettre informe le récepteur qu'il y a une porteuse). La connexion des circuits de mise à la terre (Protective Ground) peut être éliminée si l'on ne veut pas connecter la masse de l'ETTD-1 à celle de l'ETTD-2.

Le schéma ci-dessus représente la liaison à faible distance en mode synchrone : les connexions supplémentaires seront celles de la base de temps du signal à l'émission Transmit Clock (ou Transmit of Signal Timing) et de la base de temps du signal à la réception Receive Clock (ou Receive of Signal Timing). Le signal Transmit Clock est utilisé par le modem pour temporiser l'ETTD à l'émission, tandis que Receive Clock pour l'ETTD à la réception. Les interfaces synchrones de l'ordinateur et du terminal possèdent généralement des circuits d'horloge connectés aux broches 15 et 17.

La temporisation correcte est obtenue en reliant ces broches et en établissant la connexion 15-15 entre l'ETTD-1 et l'ETTD-2. Si les interfaces de l'ordinateur et du terminal sont dépourvues de séquenceurs, une horloge externe, pour l'envoi des impulsions dans la broche 15 (ou 17) d'un des deux côtés, est alors nécessaire.

Protocoles de communication

Maintenant, il s'agit d'examiner ce que l'on appelle les protocoles, une sorte de « **grammaire** » pour que les dispositifs échangent des informations. Les protocoles sont donc, à un niveau plus élevé, les gestionnaires des informations et, tout comme les interfaces, ils sont le fait des constructeurs d'ordinateurs et des organismes internationaux de normalisation.

Le terme protocole recouvre, le plus souvent, à la fois les aspects du matériel et du logiciel. Le premier est constitué des circuits conçus spécialement pour le protocole adopté, le second se compose de programmes de mise en œuvre des règles du protocole.

Test 22

1 / La communication semi-duplex permet :
- a) la transmission simultanée d'information dans les deux sens
- b) la transmission dans les deux sens, mais en alternance
- c) la transmission dans un seul sens
- d) la duplication d'informations

2 / La modulation de fréquence présente la caractéristique suivante :
- a) elle ne modifie pas la porteuse
- b) elle ne modifie ni l'amplitude ni la fréquence du signal
- c) elle ne modifie que la fréquence du signal
- d) elle modifie la fréquence et la phase du signal

3 / La configuration multipoint utilise :
- a) une seule ligne de communication
- b) autant de ligne qu'il y a d'ETTD esclaves
- c) autant de ligne qu'il y a de phases de balayage (scrutation)
- d) autant de lignes que l'ETTD maître peut gérer

4 / Le sigle EIA-RS232C désigne :
- a) les modems asynchrones
- b) une norme de connexion
- c) une norme de modulation
- d) une norme relative aux interfaces de communication série

5 / Le signal Data Set Ready indique :
- a) que le modem est prêt à émettre
- b) que le terminal est prêt à émettre
- c) qu'il y a une porteuse sur la ligne
- d) que le modem a reçu une demande d'émission

6 / Si le signal Receive Data est actif, le signal suivant doit également être actif :
- a) Signal Ground
- b) Transmit Clock
- c) Carrier Detector
- d) Transmit Data

7 / Quel signal indique que l'ETTD est prêt aussi bien à émettre qu'à recevoir ?
- a) Clear to Send
- b) Data Terminal Ready
- c) Request to Send
- d) Carrier Detector

8 / Le signal Transmit Clock sert :
- a) à synchroniser l'ETTD et l'ETCD
- b) à synchroniser le modem
- c) à synchroniser l'ETTD à l'émission
- d) à synchroniser les caractères émis

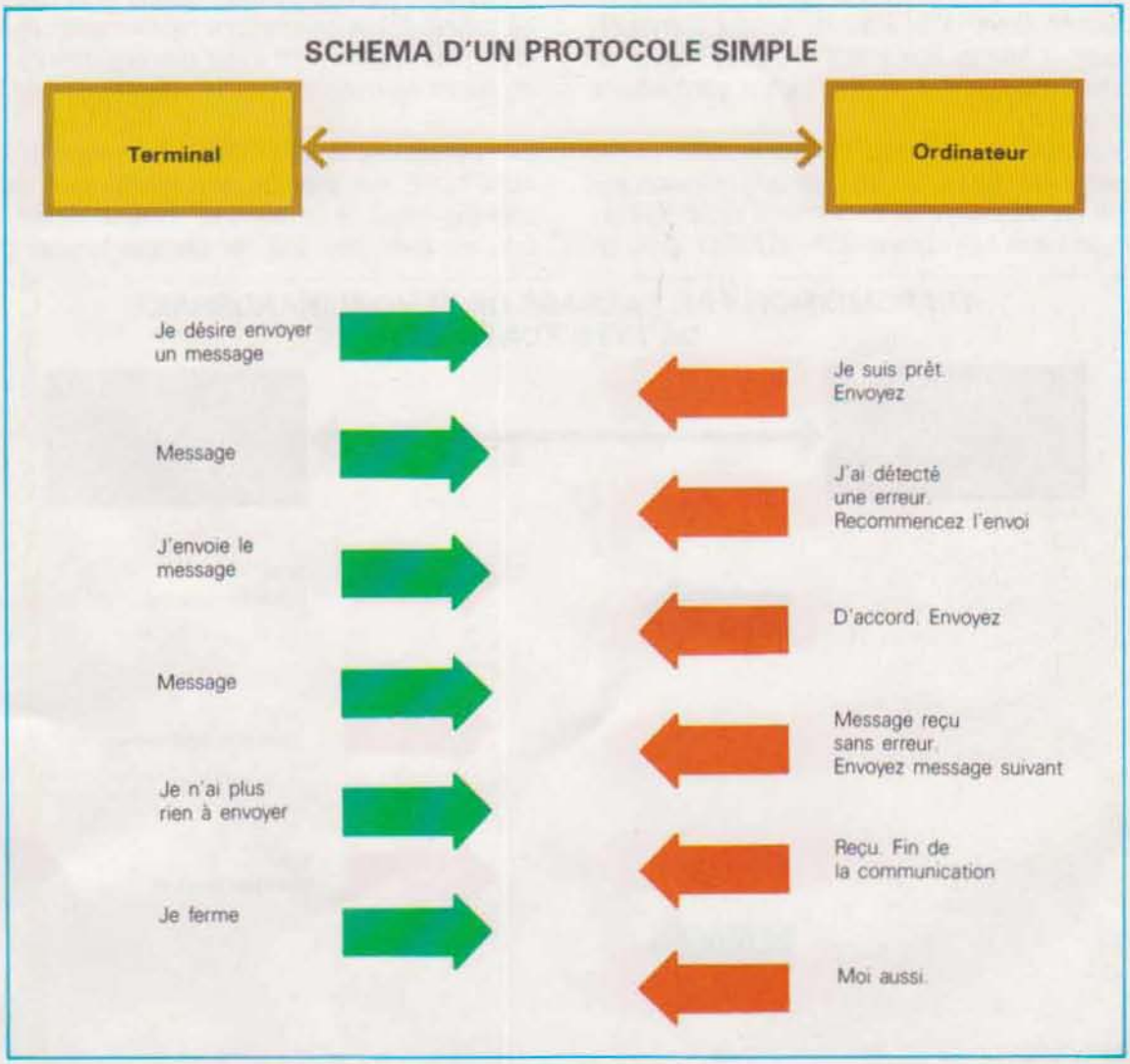
Voir les solutions du test en page 1342

Le « handshake »

Les échanges ont besoin d'être synchronisés. C'est le rôle procédure appelée « handshake » (transmission avec passage de témoin) qui permet à deux transmissions séquentielles, non synchronisées mais mutuellement dépendantes, de se dérouler. Elle ne doit pas être confondue avec le mode de transmission (synchrone ou asynchrone), son objet étant de garantir l'intégrité des messages transmis. Si, par exemple, un ordinateur doit communiquer avec un terminal, le handshake garantit que le terminal affichera toutes les données émises par l'ordinateur, indépendamment de la vitesse de transmission. Les informations échangées au cours du handshake constituent un dialogue du type :

- Message prêt à être transmis
- Signal indiquant que l'unité est prête à recevoir le texte
- Signal de début d'émission du texte
- Acceptation ou renvoi du texte
- Détection d'erreurs dans le texte reçu
- Eventuelle retransmission du texte
- Fin du texte

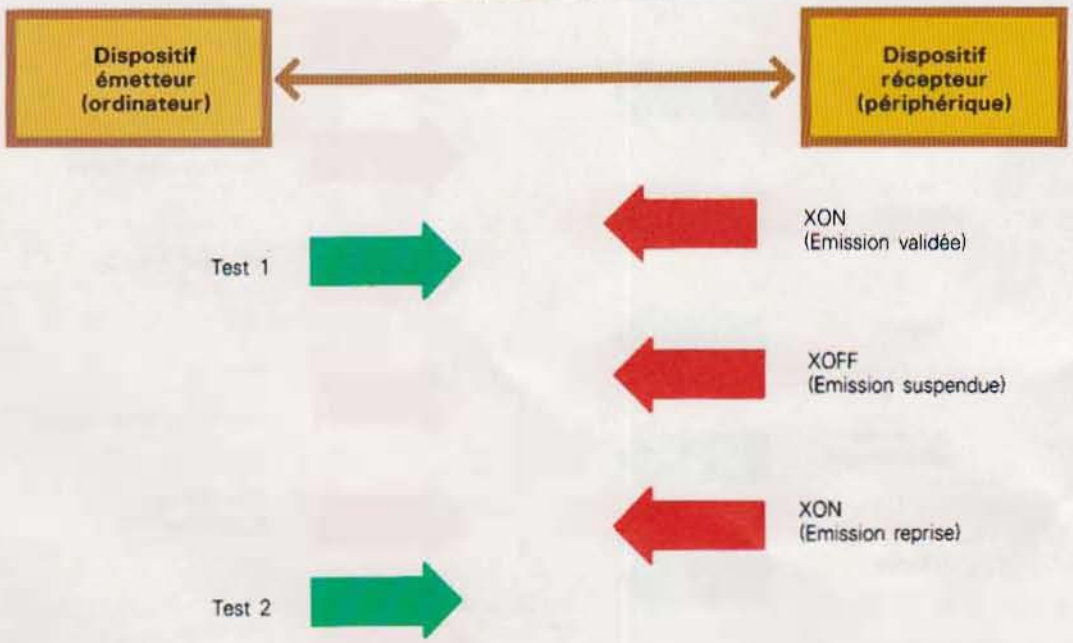
La séquence du protocole représenté ci-dessous montre un terminal informant l'ordinateur qu'il a un message à lui transmettre. L'ordinateur identifie le terminal et active la transmission. A la réception, si l'ordinateur détecte une erreur, il demande au terminal d'émettre à nouveau le texte. La deuxième fois, le message reçu ne contient pas d'erreur et l'ordinateur invite à continuer avec une autre communica-

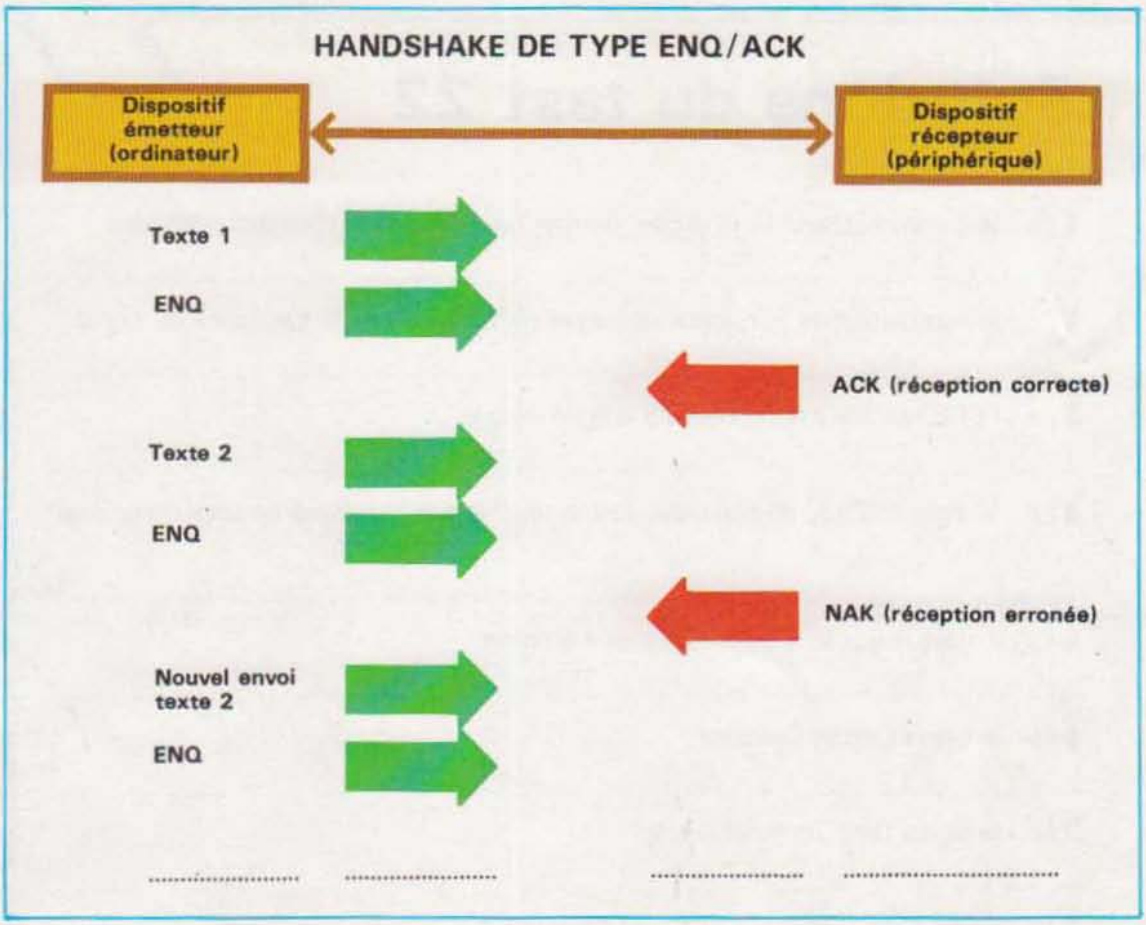


tion. Le terminal n'a pas d'autre message à envoyer et en informe l'ordinateur. Si, à son tour, l'ordinateur a un texte à envoyer au terminal, il déclenche le processus.
N'ayant rien à transmettre, il met fin à la communication, et le terminal en fait autant.
Il apparaît donc que les lignes de communication gérées par des protocoles ne voient pas transiter seulement les données à transmettre, mais aussi des informations supplémentaires servant à certifier qu'aucune donnée n'a été perdue. Ces informations constituent **l'en-tête** du protocole et en même temps un critère de jugement du protocole : un protocole est efficace si son en-tête est courte, parce que cela signifie qu'il réussit à garantir l'intégrité des informations en ajoutant un minimum de données, ce qui équivaut à une perte de temps minimale. Comme il y a plusieurs modes d'interaction entre deux stations devant communiquer, il existe également plusieurs types de «handshake», dont les principaux sont décrits ci-après.
Voici un exemple, pour clarifier le rôle joué par cette procédure. Supposons que la liaison entre un ordinateur et un terminal vidéo soit du type série asynchrone (EIA-RS232C) avec un débit de transmission de 9 600 bauds. A cette vitesse, près de 960 caractères voyagent sur la ligne en une seconde, et le terminal ne peut les visualiser au fur et à mesure. Il faut donc qu'à des intervalles réguliers, le périphérique demande à l'ordinateur d'interrompre la transmission pour qu'il ait le temps de visualiser et, d'une manière générale, de traiter les caractères reçus. Les terminaux du type imprimantes, dérouleurs de bande magnétique,... procèdent de même. Pour remédier à l'inconvénient représenté par la suspension de la transmission, on a recours à des mémoires tampons incorporées aux terminaux. Chaque périphérique peut ainsi stocker des informations avant leur visualisation. Durant la communication, les données envoyées par l'ordinateur sont provisoirement rangées dans le tampon du terminal et ensuite prélevées pour être traitées. Quand le tampon se sature, il faut interrompre momentanément la communication pour éviter que des données ne soient perdues.

La méthode XON/XOFF. La procédure XON/XOFF est précisément fondée sur ce principe (schéma ci-dessous). Quand le tampon est plein aux 2/3, le terminal envoie à

TRANSMISSION PAR PASSAGE DE TEMOIN (HANDSHAKE) DE TYPE XON/XOFF





l'ordinateur les caractères DC3 (voir tableau des caractères ASCII). Cela signifie Transmit OFF (ou XOFF) : l'ordinateur doit cesser d'émettre. Le terminal visualise ensuite les données stockées dans son tampon et le vide ; quand le tampon n'est plus qu'à 1/3 de sa capacité (autrement dit quand il a été vidé d'un tiers), le terminal envoie à l'ordinateur les caractères DC1, (Transmit ON, ou XON) : le terminal peut reprendre l'émission. Les valeurs données (1/3 et 2/3) sont variables, en fonction du terminal à gérer.

D'autres périphériques utilisent la même méthode de « handshake », mais en employant non plus DC1 et DC3 mais l'état haut ou bas d'un signal RS232C, par exemple Clear to Send ou Data Terminal Ready. Ce dernier cas constitue la forme de liaison la plus élémentaire et entraîne la gestion d'autres signaux liés à Transmit Data et à Receive Data.

La méthode ENQ/ACK. Une autre forme d'interaction est fondée sur la méthode **Enquire/Acknowledge**, ou ENQ/ACK, schématisée ci-dessus. Cette méthode facilite la demande de retransmission du texte en cas de détection d'erreurs par le récepteur. Le dispositif émetteur envoie le texte suivi des caractères ENQ (interrogation) ; le dispositif récepteur traite le texte et envoie le caractère ACK (accusé de réception positif) s'il n'y a pas de problème, ou le caractère NAK (accusé de réception négatif) s'il a repéré son erreur. L'émetteur envoie un autre texte dans le premier cas, ou à nouveau le même texte s'il a reçu NAK.

La méthode ENQ/ACK contourne le problème de la saturation du tampon : en effet, d'une part les textes transmis ne dépassent pas des longueurs pré-établies et d'autre part c'est toujours le récepteur qui donne l'autorisation de poursuivre la transmission. Si le tampon est plein, il suffit de retarder l'envoi de cette autorisation. L'aspect interactif et synchronisation ne constitue qu'un des nombreux aspects de la phase de « handshake » d'un protocole.

Solutions du test 22

1 / b : la communication semi-duplex permet l'émission et la réception alternées

2 / c : la modulation de fréquence opère par modification de la fréquence du signal

3 / a : l'ETTD maître est connecté à la ligne unique

4 / d : le sigle RS232C désigne une norme relative aux interfaces de communication série

5 / a : il indique que le modem est prêt à émettre

6 / c : le signal Carrier Detector

7 / b : le signal Data Terminal Ready

8 / c : il sert à synchroniser l'ETTD à l'émission.

Clavier de l'ordinateur Commodore Plus 4

R. Villa/Marka

Un wargame pour professionnels

S'il est un domaine qui a joué un rôle moteur pour l'évolution de l'électronique, et notamment des ordinateurs, c'est bien celui de la simulation. Le concept de base de cette technique est dans la reproduction, de la manière la plus pratique pour l'utilisateur, d'un phénomène physique ou naturel, d'une machine ou d'un appareil complexes, de comportements humains, physiques ou intellectuels et, en général, de tout ensemble simulant une réalité.

Les exemples qui suivent montrent comment est né ce besoin de simulation, aujourd'hui indispensable, dans la résolution de problèmes complexes.

Dans l'étude des phénomènes physiques ou naturels, la simulation a ouvert la voie à de nouvelles théories en fournissant des outils d'études et de contrôle de modèles.

Il existe des programmes de simulation très perfectionnés, capables de prévoir l'évolution de situation complexes comme par exemple l'état des ressources de notre planète.

On peut, au moyen de quelques modifications dans les paramètres ou dans les programmes, vérifier en très peu de temps des hypothèses de développement, encore incontrôlables dans la réalité.

L'exemple le plus classique de simulation appliquée à des machines est celui du simulateur de vol pour les pilotes d'avions. Il s'agit d'une reproduction fidèle du cockpit d'un avion dans une cabine spéciale montée sur une plateforme mécanique pouvant mettre cette cabine dans la position qu'elle aurait en vol lors de manœuvres aériennes.

L'ensemble est connecté à un ordinateur qui actionne les commandes du pilote, les traite selon le modèle prévu, dirige les mouvements de la plate-forme et tous les «instruments» de bord. En fait, ne sont installés dans les simulateurs que les panneaux de commande et de contrôle, dont les touches, interrupteurs, manettes de commande, unités d'affichage, diodes électroluminescentes et indicateurs optiques et acoustiques sont reliés à l'ordinateur. Ce dispositif fournit au pilote toutes les informations qu'il recevrait vraiment au cours d'un vol réel.

Il existe même des simulateurs qui fournissent au pilote l'image (diurne ou nocturne) de ce qu'il verrait au décollage, en vol, et à l'atterrissage dans divers aéroports ou en survol d'un territoire.

Les avantages de la simulation, du point de vue de la sécurité, de l'économie et de la disponibilité sont évidents.

La formation d'un pilote à l'aide d'un simulateur constitue une formule très économique en temps et en argent, mais aussi en efficacité car elle ne dépend plus de la disponibilité physique d'un véritable avion, avec tous les risques, les coûts et les délais nécessaires à son entretien, à l'assistance au sol et en vol, aux consommations de carburant... Dans les centres de formation au sol, le pilote commence sa formation et effectue des décollages et des atterrissages (même en catastrophe) dans n'importe quel territoire, et tout cela dans n'importe quel avion, sans avoir jamais réellement quitté la terre.

La forme la plus évoluée de simulation réunit des phénomènes physiques, des appareils et des comportements humains dans un ensemble où tous ces éléments sont en corrélation. L'ASTT (Action Speed Tactical Trainer, Formation tactique en temps réel) relève de cette dernière forme de simulation.

L'objectif de l'ASTT est de permettre à des QG de la Marine militaire de planifier et d'effectuer des exercices aéronavals, des plus simples aux plus complexes, en fournissant aux commandants et aux officiers des diverses unités tous les moyens nécessaires à leur formation tactique, tant au cours de l'exécution que du «playback» ultérieur des exercices.

Pour pouvoir être effectué dans la réalité, un exercice aéronaval complexe demande des mois de planification et de préparation de centaines de personnes (commandants, officiers et équipages de différentes unités, tout le personnel à terre) avant, pendant et après son déroulement. Il faut aussi préparer les unités (travaux en chantier, essais, ravitaillements, approvisionnements), transférer les unités de leur zone de stationnement ordinaire à celle choisie pour l'exercice, préparer le théâtre des opérations (contrôle des activités civiles, comme les vols d'avions de ligne ou privés, le transit d'embarcations de ligne, de bateaux de plaisance ou de pêche), et prévoir toute une série d'autres activités annexes.

De plus, un exercice réel est limité par la réalité même. Par exemple, lors d'un exercice opposant deux « adversaires », les différentes unités engagées se voient affecter à telle ou telle équipe. Or, il est clair que toutes ces unités, comme d'ailleurs les appareils et les systèmes de bord (détecteurs, armement, etc...) appartiennent à la Marine militaire.

D'autres limites sont imposées à l'emploi des armes, car il est évident que l'on ne peut pas ouvrir réellement le feu ou lancer des missiles contre ses propres unités.

Se pose ensuite le problème de l'analyse globale des résultats de l'exercice, dont l'objectif est d'examiner conjointement l'ensemble des activités de chaque unité afin de les mettre en rapport les unes avec les autres et d'obtenir un tableau récapitulatif des actions menées.

Il est ensuite possible d'en tirer les enseignements recherchés et les conclusions.

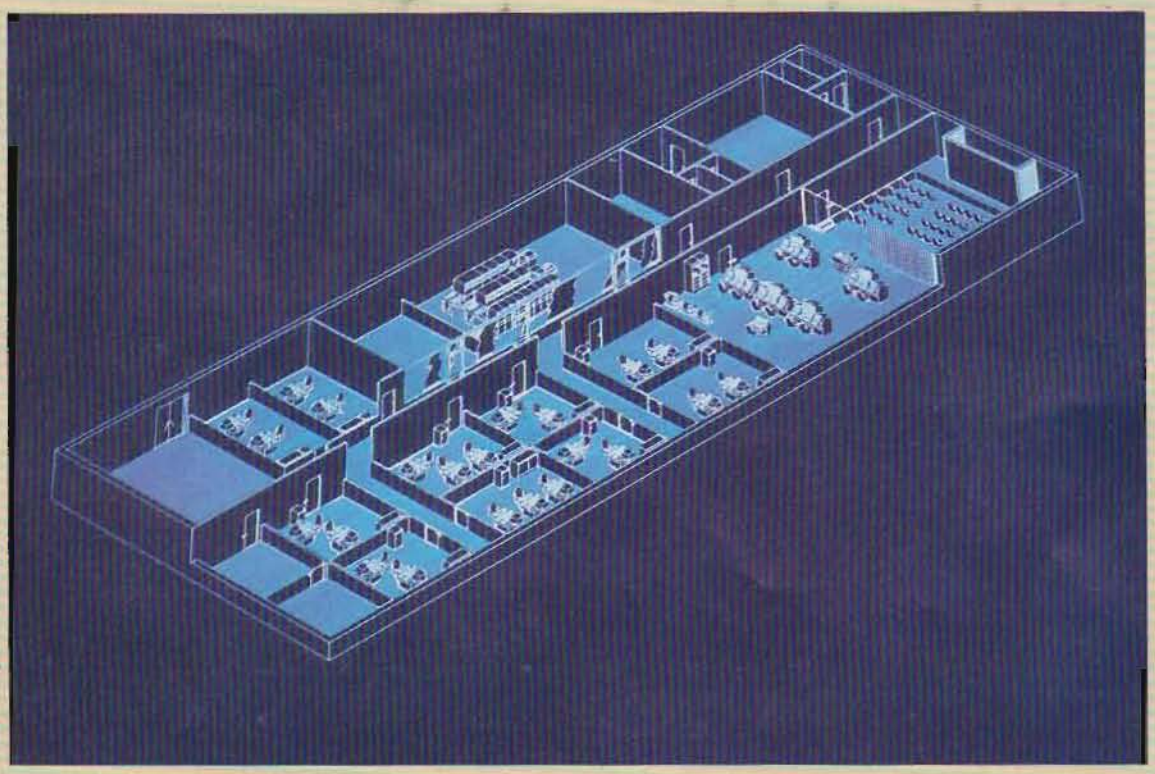
Il apparaît clairement qu'un exercice réel, s'il reste un instrument irremplaçable de contrôle du bon fonctionnement des structures, des unités, des appareils et de la formation des hommes, est loin de constituer la meilleure méthode en termes d'économie, de rapidité, de pratique, de souplesse et de répétabilité pour assurer la formation tactique des groupes de commande.

Toutes ces raisons ont fait naître le besoin de disposer d'un système de simulation pour fournir les moyens de se former sans se heurter aux inconvénients, parfois même aux dangers, d'exercices réels.

L'ASTT permet non seulement de répondre largement à ces besoins, mais aussi d'étudier, d'appliquer et de contrôler de nouvelles méthodes tactiques.

Il se compose de cinq sous-systèmes principaux répartis dans douze sites. Chacun des dix centres de commande (voir maquette ci-dessous) peut être considéré comme le centre de commande de n'importe quel type d'unité (navire, submersible, avion, hélicoptère). Les officiers exécutent leurs missions à l'aide de consoles, de moniteurs alphanumériques, d'unités vidéo graphiques et d'appareils à leur disposition pour gérer et contrôler toutes les fonctions de l'unité: navigation, détecteurs, armes, communications...

Centre de commande de l'ASTT.



B. Liotta/Il Dagherrotipo

Chaque groupe de commande travaille en temps réel à bord de la cabine comme s'il se trouvait dans son unité, en plein théâtre des opérations.

La salle de ***contrôle/auditorium*** *est équipée de cinq consoles à deux places qui permettent aux instructeurs de contrôler le déroulement des opérations et, le cas échéant, d'intervenir grâce aux messages-écrans, et aux images des manœuvres projetées en temps réel sur écran géant de 3 × 3 mètres.*

Des conclusions sont tirées et le point est fait dans cette même salle, avec possibilité de reprojection des opérations et de réécoute (en synchronisme avec les événements) de toutes les communications radio enregistrées.

La ***salle de calcul*** *est équipée, outre d'importants périphériques, de deux ordinateurs VAX 11/780 de 1 et de 1,5 méga-octets de mémoire centrale, avec une mémoire commune de 0,5 Mo pour l'échange de données, et enfin 82,5 Mo et 55 Mo de mémoire de masse, plus une unité à bande pour l'enregistrement des données sur les manœuvres.*

Le contrôle des ordinateurs et le déroulement des opérations de planification et de préparation des opérations sont réalisés par quatre terminaux, alors qu'une imprimante rapide fournit les données et les tableaux souhaités sur papier. On dispose également d'une table à digitaliser.

Cet équipement sert à introduire dans l'ordinateur le plus possible de données concernant le profil géographique des côtes représenté dans les cartes maritimes.

Celles-ci sont placées sur la table et l'opérateur suit avec le curseur le profil de la ligne côtière. Un programme spécial saisit alors les données de position, corrige les erreurs, calcule la corrélation avec les positions géographiques réelles et stocke enfin sur disque le profil des côtes ainsi généré.

Le sous-système de calcul est complété par un équipement spécial de générateurs pour la ***représentation vidéographique*** *ainsi que des interfaces nécessaires à l'établissement de liaisons avec les consoles, les écrans de contrôle, les claviers, c'est-à-dire avec tous les appareils de l'ASTT répartis dans les diverses salles.*

Le sous-système de contrôle comprend les ***consoles*** *pour instructeurs, l'****écran géant*** *et l'unité bande analogique à 8 pistes. Celle-ci est employée pour enregistrer jusqu'à six émissions-radio simultanées.*

Chacune occupe une piste différente ; vient ensuite le commentaire (7e piste) sur les opéra-

Salle de calcul du système ASTT (simulation militaire).

B. Lotta/Il Dagherotipo

tions exécutées par un contrôleur ; enfin la 8e piste est réservée à la synchronisation. Le sous-système des cabines possède dix éléments contenant l'ensemble des appareillages nécessaires à l'entraînement des officiers.
Le sous-système « rapport » se compose de l'ensemble de fonctions et d'appareils disponibles (énumérés ci-après) pour effectuer l'analyse après les manœuvres :

— enregistrement de toutes les données en temps réel sur bande magnétique,
— enregistrement des communications radio et commentaires sur bande magnétique,
— tableaux, d'ensemble et de détail, des données et des événements significatifs des opérations effectuées,
— écran géant...

Le sous-système de communications met à la disposition des stagiaires jusqu'à 18 lignes radio reproduisant des conditions réelles (bruits et limitation), 3 réseaux de télétransmission RATT (téléscripteurs), et enfin un système de communication de service.

La **réalité simulée** par l'ASTT est extrêmement **complexe et structurée**, surtout pour la corrélation et l'intégration des parties composantes. Les éléments fondamentaux du scénario simulé sont les suivants :

- **Théâtre des opérations,** composé de la mer, du ciel et de la terre dans diverses conditions d'environnement (état de la mer, direction et vitesse des courants marins, du vent, visibilité, conditions météorologiques, propagation radio, acoustique...).

- **Unités aéronavales** subdivisées en unités de surface (du porte-avion à la plus petite unité), sous-marins (conventionnels et nucléaires) et unités aériennes (de l'avion de combat à l'hélicoptère) avec toutes les caractéristiques statiques et dynamiques (dimension et forme, surfaces équivalentes, radar, sonar et infrarouge, vitesse accélération, altitudes ou profondeurs maximales, autonomie, facilité de manœuvre...).

- **Bases aéronavales**, centres au sol ou sur des plates-formes en mer pour l'assistance et le commandement des unités aéronavales, avec tous les équipements, capteurs, systèmes de contre-mesure et éventuellement systèmes d'armement.

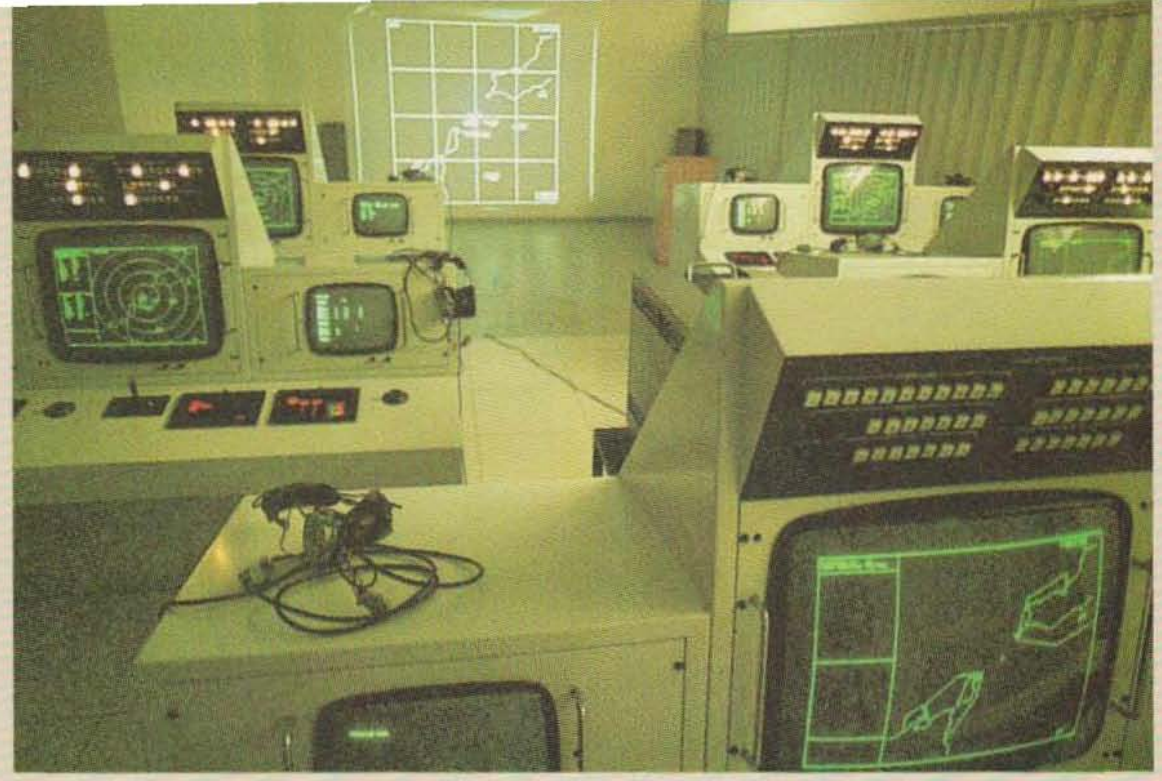
- **Appareils et capteurs de bord**, notamment radar, sonar, appareils d'identification (IFF), de mesure de signaux électromagnétiques (ESM), systèmes de bouées sonores, capteurs d'anomalies magnétiques (MAD), capteurs optiques, etc.

- **Systèmes de communication**, répartis en phonique (radio), transmission de données (data link) et transmission par téléscripteurs (RATT).

- **Systèmes de contre-mesure électroniques** (ECM), composés d'appareils de brouillage et de bruit, électromagnétique et à infra-rouges...

- **Systèmes d'armement** composés de torpilles, de canons, de missiles et antimissiles, systèmes missile/torpille, charges en profondeur, mines.
Les programmes tiennent compte, bien entendu, des modalités d'emploi de ces systèmes et de leurs limites.

- **Ressources humaines** : toutes les activités effectuées par l'homme à bord des unités ou à terre sans toutefois être l'objet d'un exercice spécial des équipes de commandement, même si elles font partie intégrante d'une manœuvre.
Ainsi des activités liées au système radar à bord d'un navire qui se trouvent placées sous le contrôle du commandement pour ce qui concerne la mise en route, le mode de fonctionnement, l'utilisation des appareils, alors même que ce sont les opérateurs radar qui contrôlent les appareils et recueillent l'information.
Ces données — en général position, cap et vitesse d'une unité donnée — après avoir été éventuellement reliées à d'autres, sont ensuite présentées à la salle de commande sur écrans graphiques.
A l'ASTT, dont le but n'est pas de former des opérateurs radar (c'est le rôle des simulateurs spéciaux), toutes les tâches accom-



B. Liotta/Il Dagherrotipo

Les photos représentent les cinq consoles-instructeur et l'écran géant (en haut) ainsi que les consoles cabine (en bas).

plies par les opérateurs sont simulées par des programmes spécifiques. Ceux-ci fournissent ensuite aux équipes de commandement les mêmes informations que celles qu'ils auraient reçue directement des radars.

Les trois activités principales des ASTT concernent les activités suivantes :

— préparation,

— exécution,

— revisualisation des manœuvres,

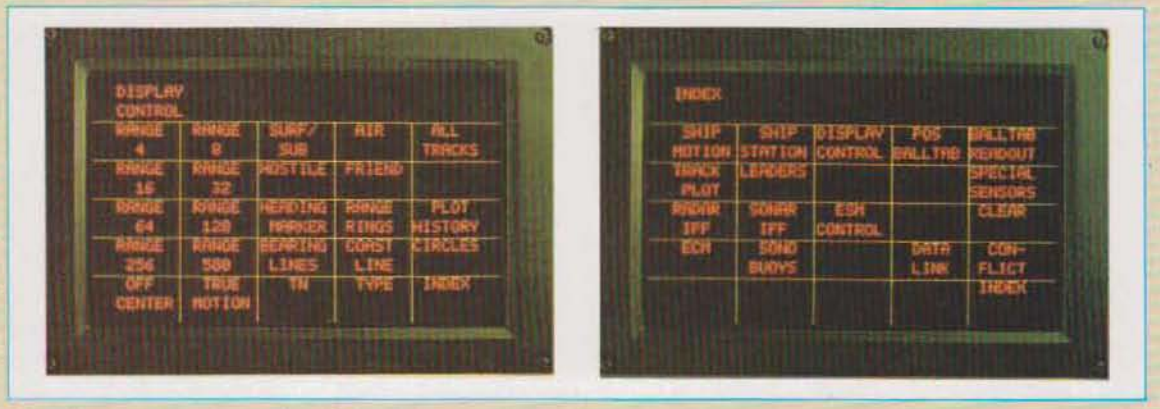
— accessoirement, archivage et la mise à jour des données relatives aux unités aéronavales, aux scénarios opérationnels, etc.

Un groupe d'officiers instructeurs programme les manœuvres en fonction des exigences de l'entraînement et des groupes de commandement à former (l'ASTT est en mesure de couvrir jusqu'à dix manœuvres différentes et simultanées).

On choisit un théâtre d'opération (jusqu'à 160) dont les dimensions maximales atteignent 1 000 × 1 000 milles, de 0 à 900 mètres de profondeur, et de 0 à 90 000 pieds d'altitude, et on détermine les unités qui doivent y participer, ainsi que leurs positions initiales.

Les opérations se déroulent en partie dans des bureaux grâce notamment au dialogue avec l'ordinateur qui demande à l'utilisateur toutes les informations nécessaires, comme le nom de la manœuvre, le théâtre des opérations, l'état de la mer, la visibilité, les conditions de propagation radio, le plan des fréquences de transmission à employer, les unités en jeu et leur position initiale.

B Liotta/Il Dagherrotipo



Visualisation des consoles du système ASTT.

Toutes ces informations sont mises en mémoire sur un disque.

L'exécution d'une manœuvre est précédée par une phase d'explication au cours de laquelle sont précisées, aux personnes en formation, les ***tâches de chacun « à bord »*** *des cabines ainsi que toutes les informations nécessaires au déroulement correct du « jeu ».*

Une fois à bord de leur cabine, les officiers deviennent les commandants de leurs unités. A partir de ce moment, leurs décisions détermineront le déroulement des manœuvres.

De leur cabine, les officiers voient et perçoivent le monde extérieur (théâtre opérationnel) comme s'ils se trouvaient réellement en ***salle de commandement de l'unité simulée.***

Cela signifie que, si une unité se trouve en mer avec capteurs et systèmes de communication éteints, et qu'aucune autre unité n'est détectée dans les limites de sa visibilité, l'écran tactique de la cabine ne reproduit que le symbole de l'unité en question. En revanche, si dans la cabine le radar est allumé et si toutes les conditions nécessaires à la détection sont satisfaites (portée du radar, fréquence de transmission non brouillée par des contre-mesures électroniques, réflexivité radar de la cible, conditions de propagation), les symboles des unités découvertes par le radar apparaîtront normalement à l'écran, ainsi que leur position.

Chez les instructeurs, il en va autrement. L'écran tactique fera apparaître, indépendamment des indications des divers capteurs à bord de chaque unité, toutes les unités en jeu dans leur position réelle, comme si tout le théâtre opérationnel était vu en surplomb sans limitation. Pendant le déroulement du jeu, les instructeurs suivent les événements grâce aux consoles, à l'écran géant, aux télescripteurs :

— Ils commandent les « fightback targets » ou cibles de contre-attaque (unités sans salle de commande proprement dite, pilotées par des équipes à partir des consoles).

— Ils peuvent intervenir en ***modifiant le film des événements.*** *Par exemple, après avoir engagé un combat contre une unité ennemie, une unité donnée subit des dommages tels que radars et systèmes d'antennes de communications hors d'usage.*

— Par contre, les instructeurs pourraient décider de modifier cette situation pour permettre à l'unité de continuer les manœuvres sans limitations, et changer la nature des dégâts enregistrés.

— Les instructeurs peuvent aussi rétablir ou simuler d'autres avaries.

— Lorsque la manœuvre a atteint les objectifs fixés, les instructeurs arrêtent l'action pour passer à une phase d'analyse de ce qui s'est produit. Ce travail est largement simplifié par les possibilités offertes par l'ASTT qui élimine rapidement les éventuelles erreurs.

La possibilité de ***revoir sur écran géant le déroulement de la manœuvre*** *dans son intégralité, de réécouter les communications radio, d'accélérer le playback et de se* ***repositionner*** *à tout moment du « wargame » fait de l'analyse a posteriori la partie la plus intéressante de l'entraînement tactique.*

Bruno Liotta
Datamat Ingegneria dei Sistemi (Italie)

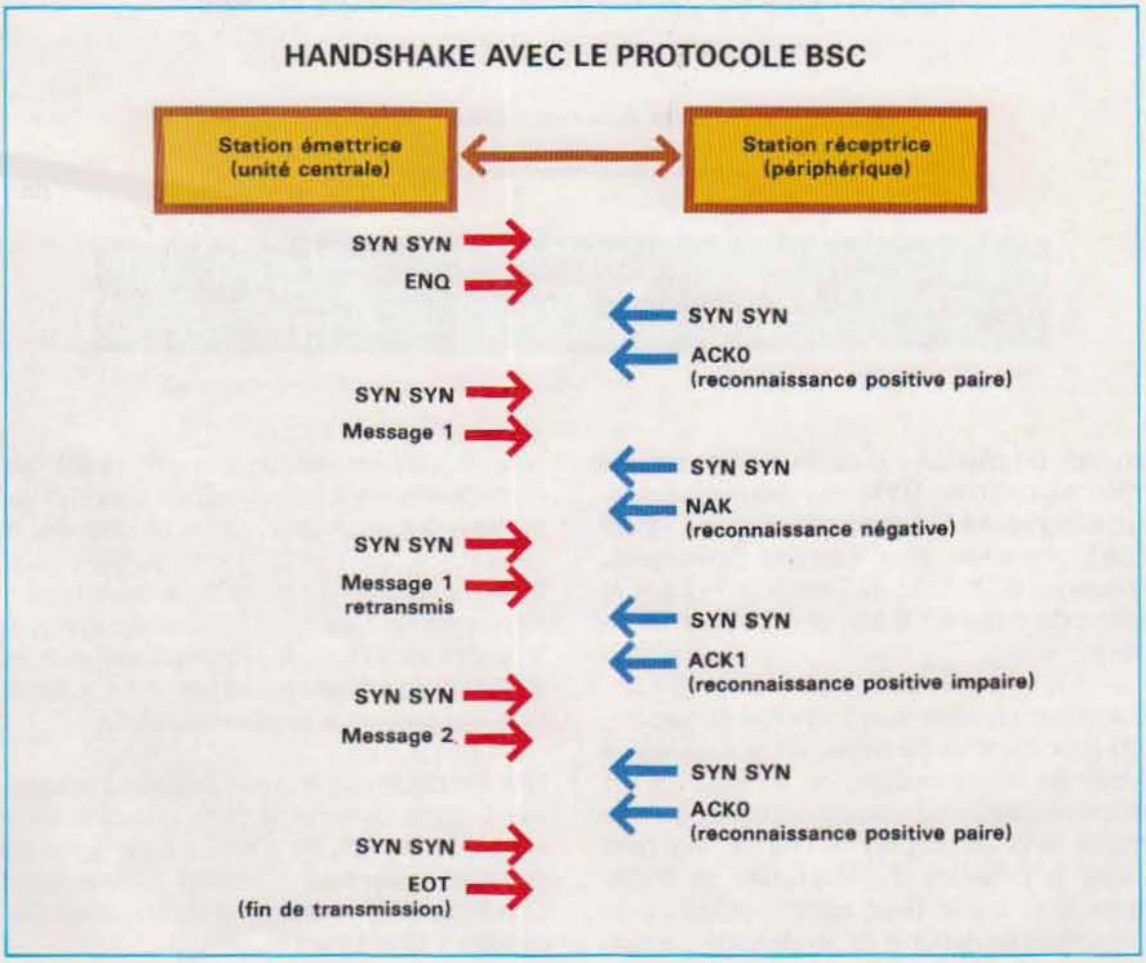
Le protocole BSC

La procédure BSC (Binary Synchronous Communication) a été introduite pour la première fois par IBM en 1966 ; elle a permis de stabiliser la situation confuse qui prévalait alors en matière de communication. Sa très large diffusion en a fait un standard de facto, même si l'initiative ne venait pas d'un organisme officiel de normalisation.

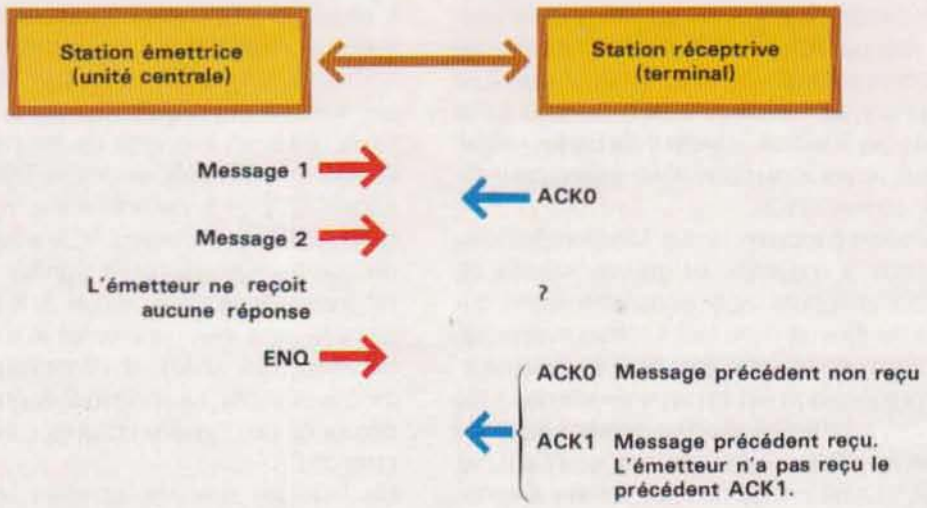
Le protocole concerne tout le faisceau de communication à moyenne et grande vitesse du type bidirectionnel non simultané (semi-duplex). Il ne prévoit donc pas l'échange simultané d'informations entre émetteur et récepteur. La communication est de type synchrone : les messages sont précédés d'un nombre défini de caractères SYN (tableau codes ASCII et EBCDIC). La séquence de « handshake » adoptée (schéma ci-dessous) est composée d'une version modifiée de l'ENQ/ACK. On a prévu deux caractères de reconnaissance positive (ACK0 et ACK1) qui sont envoyés alternativement par le récepteur pour signifier que le texte n'est pas entaché d'erreur.

L'emploi des accusés de réception pairs et impairs garantit la réception des reconnaissances et des messages tout entiers. En effet, si l'unité qui transmet ne reçoit rien après l'envoi d'un texte, dans un intervalle de temps donné, elle envoie à nouveau le caractère ENQ (tableau 1 page 1350). Si le destinataire a reçu le message précédent, il envoie l'ACK alternatif (ACK1 dans le tableau cité) pour signifier que seule la reconnaissance a été perdue. Si le message n'a pas été reçu, elle retransmet le même accusé de réception ACK0 et l'émetteur renvoie le message perdu. Le tableau 2 donne le contenu des blocs de données échangés avec le protocole BSC.

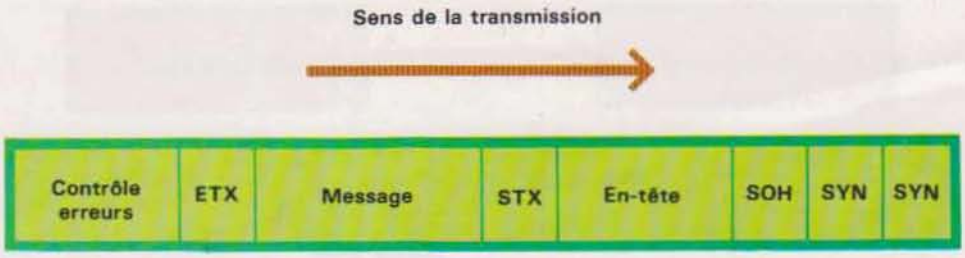
Un bloc de données contient en général 3 éléments : un en-tête optionnel (informations extérieures au message), le texte proprement dit et un dernier élément pour le contrôle des



SIGNIFICATION DES ACCUSES DE RECEPTION ACK ALTERNATIFS EN BSC



FORMAT DES INFORMATIONS TRANSMISES EN BSC



erreurs. De plus, on trouve certains caractères spéciaux comme : **SYN** pour assurer la synchronisation entre émetteur et récepteur, **SOH** Start of Header pour identifier l'en-tête du message, **STX** Start of Text pour indiquer le début du texte, **ETX** End of Text pour fin de texte.

L'en-tête comporte un nombre fixe de caractères pour identifier l'émetteur et le destinataire (code de reconnaissance ou adresse) ; il est employé s'il existe plusieurs dispositifs sur une même ligne. La longueur fixe de l'en-tête rend inutile la présence d'un caractère de fin de texte alors que le texte exige la présence de caractères de début et de fin de texte. Le bloc pour le contrôle des erreurs est généré par l'émetteur grâce à des algorithmes appropriés. Au fur et à mesure qu'il prélève les données, le récepteur reconstruit ce bloc de contrôle (avec le même algorithme) et en fin de transmission il compare les deux blocs reconstruits et reçus. S'ils sont identiques, le récepteur envoie le signal ACK (pair ou impair) ; sinon, c'est le signal NAK qui permettra la retransmission.

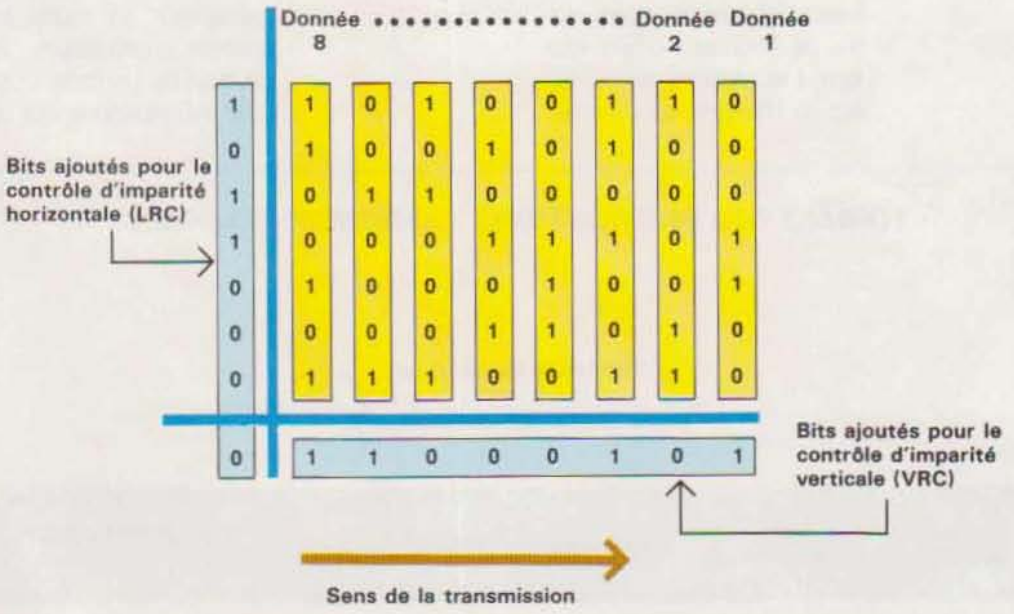
Une fois déterminé le mode de parité à adopter (dans le cas du schéma page ci-contre, on a adopté l'imparité), on effectue deux types de contrôle : **vertical** (Vertical Redundancy Check ou VRC), **horizontal** (Longitudinal Redundancy Check ou LRC).

A chaque caractère envoyé (représenté verticalement dans le tableau) l'émetteur ajoute un 1 si le nombre de bits 1 dans le caractère est pair, et un 0 s'il est impair. Le nombre global de bits 1 de chaque caractère doit donc être impair. Si le récepteur découvre que le nombre de 1 d'un caractère est pair, il a détecté une erreur, auquel cas il demande la retransmission. Cette méthode (VRC) n'est plus valable lorsque le récepteur comptabilise un nombre impair de 1, mais le caractère reçu ne correspond cependant pas à celui qui a été envoyé (à cause d'une double inversion par exemple). C'est pour cela qu'on ajoute le contrôle horizontal (LRC) qui analyse tous les bits de manière longitudinale et ajoute un 1 si le nombre de 1 est impair. Le contrôle se fait pour tous les caractères qui constituent soit le texte, soit l'en-tête (voir tableau page 1351). La colonne de bits, complétée par le bit d'imparité, est enregistrée dans la phase finale de contrôle des erreurs. Si la transmission est correcte, la séquence de bits enregistrée par l'émetteur est celle qui a été recomposée par le récepteur.
Le protocole BSC, encore aujourd'hui très répandu dans les communications à courte et longue distance, de type semi-duplex, ne permet donc pas l'échange simultané d'informations dans les deux sens. Autrement dit, en trafic dense, les attentes sont longues et ne sont réduites qu'avec un dialogue de type bidirectionnel simultané.

Le protocole HDLC

Mis au point par l'International standard organisation (ISO), ce protocole (High level Data Link Control) a une structure lui permettant d'accueillir des communications de type semi-duplex et bidirectionnel simultané. A la différence du BSC à base de caractères — il prévoit de transmettre des caractères (ASCII ou EBCDIC) sur les lignes — le protocole HDLC s'appuie sur les bits, car il permet la transmission d'une configuration quelconque de bits.
Du point de vue du contrôle du trafic, le protocole ne fait pas référence à des unités émettrices et réceptrices mais à des stations primaires et secondaires : la primaire entame le dialogue et les secondaires y répondent.

CONTROLE D'IMPARITE HORIZONTALE (LRC) ET VERTICALE (VRC)

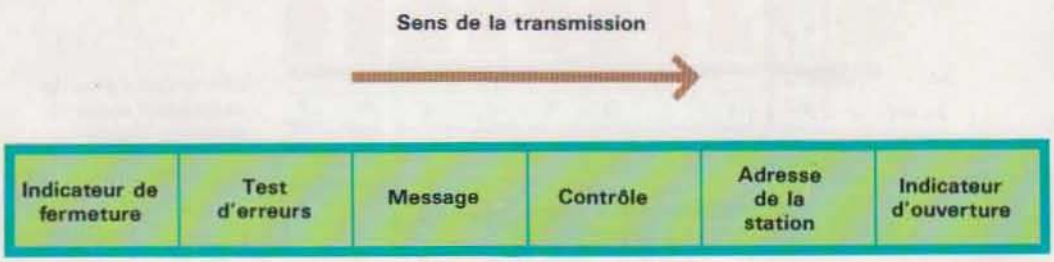


Cette schématisation est liée au type de configuration adopté (point-à-point, multipoint,...) ; cela ne signifie pas nécessairement que la station secondaire ne peut parler que si elle est interrogée.
En général, le dialogue entre stations peut être asynchrone, car chacune d'elles communique quand elle le désire. Le format des informations transmises en HDLC est donné ci-dessous. Le bloc tout entier est défini une **trame** décomposée en message et en informations de contrôle. Toutefois, le message peut ne pas se trouver dans la trame : les éléments de contrôle servent alors à adresser ou à fournir des informations aux stations primaires et secondaires.
Toutes les trames sont numérotées en séquence, chacune comptabilisant les trames qu'une station a précédemment envoyées ainsi que le numéro de la trame que l'émetteur s'attend à recevoir en retour. La part de handshake est donc interne au protocole même.
Voici énumérés les principaux champs composant une trame :

Indicateur C'est une séquence de bits, unique dans toute la trame et d'une longueur fixe (par exemple la séquence 01111110). L'indicateur informe le récepteur qu'une trame va être envoyée, suivie de l'adresse d'une station. Les stations secondaires se mettent en attente.

Adresse de la station contrôle Elle identifie le destinataire du message, éliminant ainsi les trames dupliquées, omises ou erronées. Elle contient notamment le nombre de trames transmises (NS) et de trames reçues (NR). Les valeurs de NS et de NR sont différentes pour chaque station. Dans un dialogue entre deux stations, les trames signifient : « le numéro de la trame que j'envoie à la station est NS ; la prochaine trame à recevoir de cette station est NR ». La station compare son NR et la valeur de NS contenue dans la trame ; s'ils sont égaux, et s'il n'y a pas d'erreur (voir plus loin), la trame est acceptée et les valeurs de NR et NS mises à jour. En plus de NR et NS, le bloc de contrôle contient des informations supplémentaires qui permettent de transmettre beaucoup de trames avant même que le récepteur ait fourni la réponse protocolaire. Enfin, le bloc de contrôle contient des informations sur la na-

FORMAT DES INSTRUCTIONS TRANSMISES EN HDLC



ture et la longueur du message qui suit.

Message La longueur est illimitée et son code quelconque (EBCDIC, ASCII, binaire,...), grâce à la combinaison d'informations contenues dans l'adresse de la station et dans le bloc de contrôle. Si une station ne peut dialoguer qu'en ASCII, on associe ce code à son adresse. On précise alors dans le bloc de contrôle que le message qui suit est en ASCII.

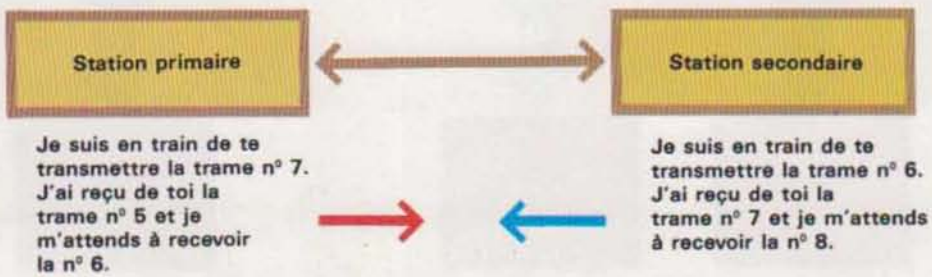
Test erreur Il est réalisé selon la méthode décrite BSC, ou en employant des techniques proches. L'émetteur envoie son propre bloc pour le test des erreurs et le récepteur compare celui qu'il a calculé et celui contenu dans la trame. Si les deux blocs ne sont pas identiques, la trame est rejetée. Le récepteur met à jour son compteur NR, et il en demandera une nouvelle transmission. Il n'est nullement nécessaire d'interrompre la communication pour retransmettre des messages ; cette demande (handshake) est, en outre, interne au protocole. (Pour cette raison on peut qualifier le protocole HDLC de bidirectionnel simultané.) (Voir ci-dessous.)

Indicateur de fermeture De même format que l'indicateur d'ouverture, il identifie la fin de la trame transmise.

Le protocole HDLC constitue aujourd'hui la base pour réaliser des réseaux d'ordinateurs complexes. Grâce à sa souplesse, on a pu créer des extensions de protocole, du type SDLC (Synchronous Data Link Control) typique d'IBM, et actuellement le plus répandu en communications « parallèle ». Adopté en 1978 par l'institut de normalisation IEEE (Institute of Electric and Electronic Engineering), il est employé pour gérer des périphériques rapides (unités disque et bande, instruments de mesure électroniques) à courte distance (quelques mètres). Le protocole est du type « octet sériel, bit parallèle » et permet jusqu'à 15 machines de converser à grande vitesse (jusqu'à un million de caractères par seconde).

Le protocole IEEE-488 fixe, en outre, un standard au niveau des interfaces, en établissant des règles précises concernant les niveaux de tension, les appareils de connexion et les câbles de liaison. Un câble IEEE-488 est composé de 16 fils, dont 8 utilisés pour la transmission des données et les autres pour le contrôle des données transmises. C'est la raison pour laquelle le protocole est dit parallèle : un caractère se présente au récepteur simultanément,

HANDSHAKE INTERNE A LA TRAME DANS LE PROTOCOLE HDLC



sur les 8 fils destinés à la transmission des données. La liaison des divers dispositifs est illustrée dans le schéma ci-dessous (définie comme bus). Chaque dispositif comporte une adresse (variant entre 1 et 15). Le protocole établit, qu'à un instant donné, il y a sur le bus un émetteur unique et un ou plusieurs récepteurs. Le bloc de données, que l'émetteur envoie, est reçu par le ou les récepteurs dont l'adresse est incluse dans le bloc transmis. A l'instant suivant, l'émetteur peut changer, comme d'ailleurs les récepteurs.

Les **lignes données** sont composées de 8 fils où normalement transitent des caractères ASCII en parallèle ; 7 fils sont employés pour l'information et le 8e est dédié au contrôle de parité. Les lignes transmettent également les instructions que les dispositifs veulent échanger ainsi que l'adresse des dispositifs choisis pour l'écoute.

Les **lignes de handshake** sont au nombre de 3 et sont utilisées pour coordonner le transfert d'information sur le bus. La communication est asynchrone et la vitesse de transfert entre le dispositif émetteur et un ou plusieurs récepteurs s'adapte automatiquement à la vitesse du dispositif le plus lent. Les diverses lignes de handshake ont la signification suivante :

DAV Data Available. Est pilotée par l'émetteur pour informer le récepteur qu'il y a des données sur le bus.

NRFD Not Ready For Data. Est contrôlé par le récepteur pour informer l'émetteur qu'un des récepteurs n'est pas encore prêt.

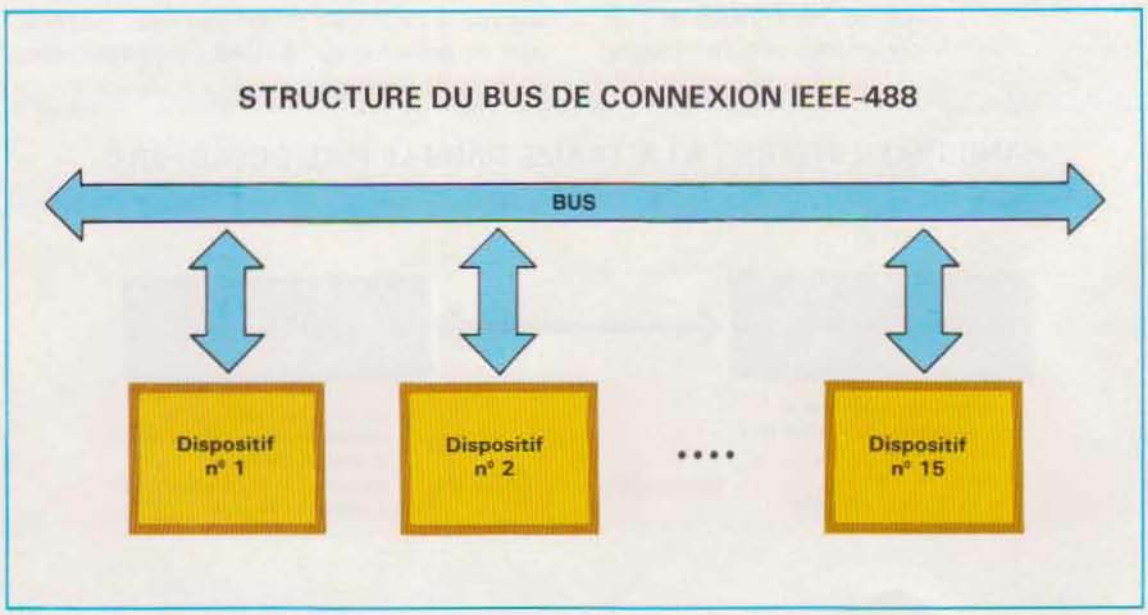
NDAC Not Data Accepted. Est toujours géré par le récepteur : les données ont bien été reçues mais on ne peut en accepter d'autres. Cette ligne est prise par l'auditeur le plus lent à la fin d'une transmission.

Les **lignes de contrôle de données** sont employées pour régler le flot de données sur le bus. Chaque ligne prend la signification suivante :

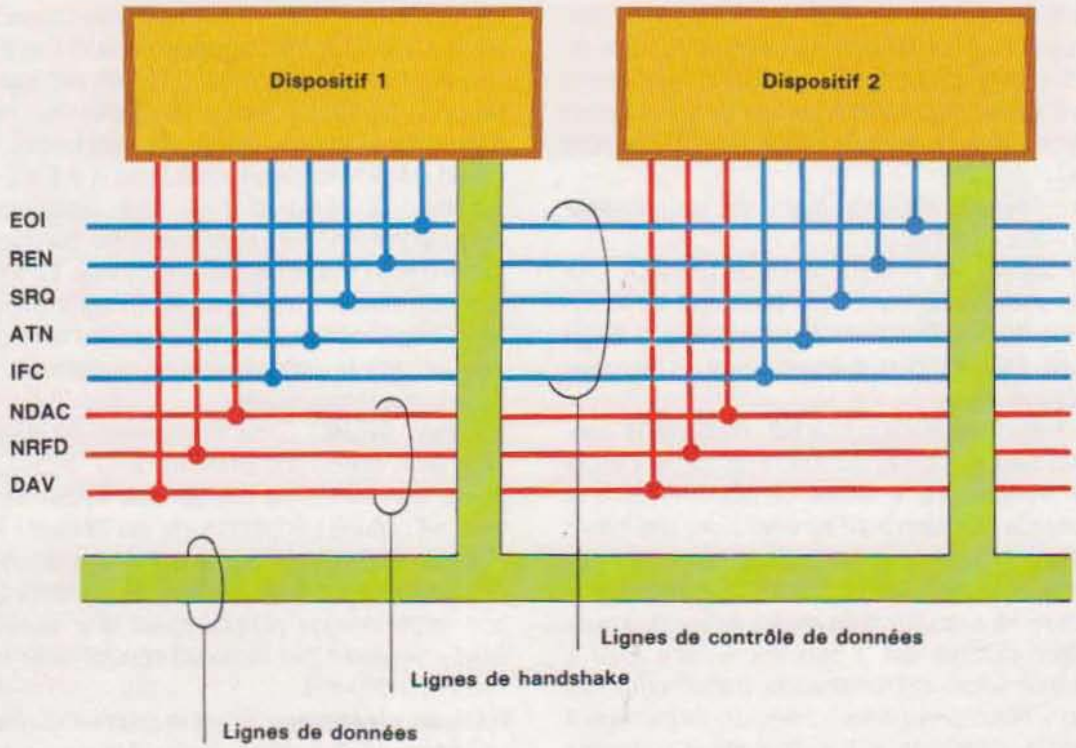
IFC Interface Clear. Est utilisée pour faire un reset (réinitialisation) du bus, en le positionnant donc sur « départ ».

ATN Attention. Oblige tous les dispositifs à interpréter le contenu du bus comme « ordre » (ligne haute) ou comme « donnée » (ligne basse).
Grâce aux ordres, on décrit l'émetteur et le récepteur ; leur adresse est enregistrée dans les lignes données. Sur le bus données, les dispositifs échangent éventuellement leurs informations.

SRQ Service Request. Est la ligne qui permet le dialogue entre des dispositifs sous interruption (interrupt). Quand



SCHEMA DETAILLE DU PROTOCOLE IEEE-488



cette ligne est employée, un des quinze équipements connectés au bus devient contrôleur et établit comment le dialogue doit être poursuivi. On peut imaginer la situation suivante : un dialogue a lieu entre émetteur et récepteurs ; la ligne SRQ devient haute (signifiant qu'un autre appareil demande un droit d'émission). Le contrôle interroge les divers dispositifs (polling) et dès qu'il découvre celui qui a demandé la ligne, lui donne la priorité.

REN Remote Enable. Est gérée par ses équipements qui peuvent se trouver dans deux états différents : local et remote (sur place et à distance). En local, ils peuvent accepter des ordres d'un opérateur, par exemple à partir d'un clavier ; sur « remote », ils peuvent accepter des ordres et des données du bus.

EOI End Or Identify. Indique la fin d'un groupe d'informations transmises.

Réseaux pour la communication de données

Tous les éléments employés pour la télétransmission de données-interfaces, lignes, modes de transmission, protocoles... se retrouvent dans les réseaux de communication.
Un réseau est, en général, un ensemble matériel (les machines) et logiciel organisant et gérant la communication.
Grâce au réseau, les ressources communes sont partagées. Par « ressource », on entend tout ce qui est mis à disposition de l'utilisateur : imprimante, mémoire de masse, programme, banque de données offertes localement ou à distance. Un réseau peut être composé par des gros ordinateurs appelés « mainframes » (unités centrales) ou ordi-

nateurs-hôtes, par de petits ordinateurs (mini et micro), des terminaux et des périphériques.
Un réseau peut être de type **centralisé** ou **distribué.** Dans l'architecture centralisée, c'est l'unité centrale (UC) qui effectue la majeure partie des traitements demandés aux divers points d'entrée du réseau. A ces points sont reliés des terminaux ou des dispositifs d'entrée/sortie dont la tâche est d'interroger l'UC.
Un tel réseau est donc fondé sur la centralisation des informations.
A l'opposé, le réseau distribué possède plusieurs unités de traitement **(nœuds)** au moins dotés de la même puissance, selon qu'il s'agit d'une UC ou, plus fréquemment, d'un mini-ordinateur.
Un nœud regroupe un certain nombre de périphériques au niveau local ou à distance. De par son architecture, le réseau distribué renforce et facilite la diffusion d'informations sur des zones géographiques éloignées. Ce dernier type de réseau est aujourd'hui le plus répandu, car il élimine le « goulot d'étranglement » de l'ordinateur central, qui a souvent à faire face à d'importantes demandes de traitement. Plusieurs ordinateurs interconnectés se partagent ainsi la tâche, et si l'un d'entre eux devient inopérant (panne, par exemple), les autres sont en mesure de prendre en charge ses travaux.
La création de réseaux d'ordinateurs permet, aujourd'hui, d'accéder à d'importantes banques de données, élaborées et distribuées par eux. Cet accès est également possible au niveau domestique : il suffit de doter la télévision d'un appareil spécial pour établir une connexion avec le nœud le plus proche. Le réseau fournit les informations les plus variées : horaires de trains et d'avions, pharmacies de garde, informations communales... Dans le cas du vidéotex, il est possible de relier un clavier au réseau téléphonique pour obtenir directement une réservation de billet d'avion, envoyer un message à d'autres usagers, etc. Dans certains pays, on reçoit déjà à domicile son journal, par simple connexion d'une imprimante à la télévision.

Topologie et caractéristiques des réseaux

Les nœuds d'un réseau s'articulent de différentes manières, tout en respectant les architectures décrites ci-dessous.

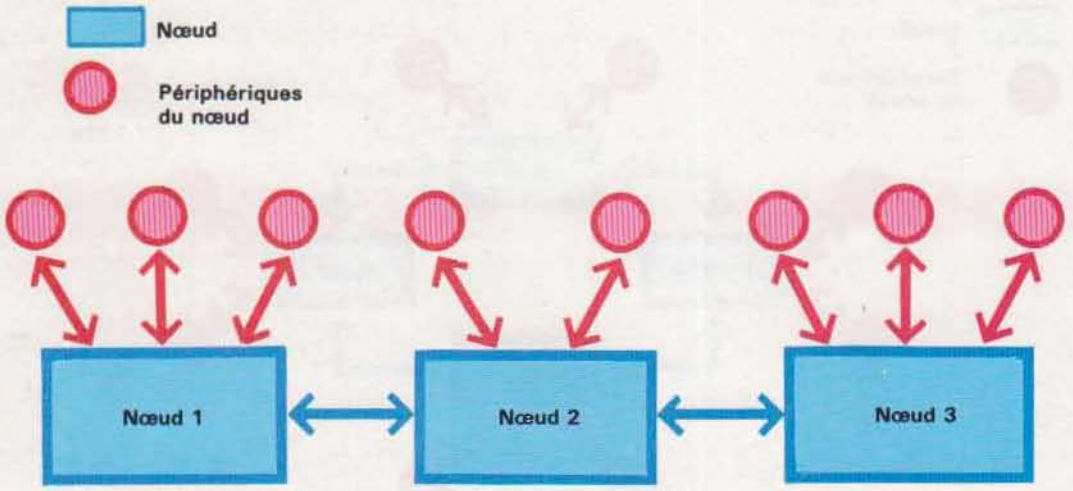
Réseau en chaîne. C'est la généralisation de la connexion point-à-point (schéma 1, p. 1357). Les nœuds sont reliés en cascade ; les communications partant du nœud 1 et destinées au nœud 3 passent forcément par le nœud 2. Le réseau est simple à installer, mais il présente l'inconvénient de dépendre du nœud intermédiaire pour les liens entre les extrémités. Si le nœud 2 ou tout autre nœud intermédiaire ne fonctionne pas, les communications sont totalement interrompues. La topologie en chaîne est employée lorsque chacun des nœuds doit accomplir une tâche bien précise et que la communication entre tous les nœuds n'est pas vitale.

Réseau étoilé. C'est l'extension du réseau centralisé concentré (schéma 2, p. 1357). Le nœud central trie la masse des informations vers les nœuds périphériques, qui mettent à la disposition de l'extérieur la partie des données à laquelle ils ont droit d'accès. La communication entre nœuds périphériques doit transiter nécessairement par le nœud central dont tout le réseau dépend.

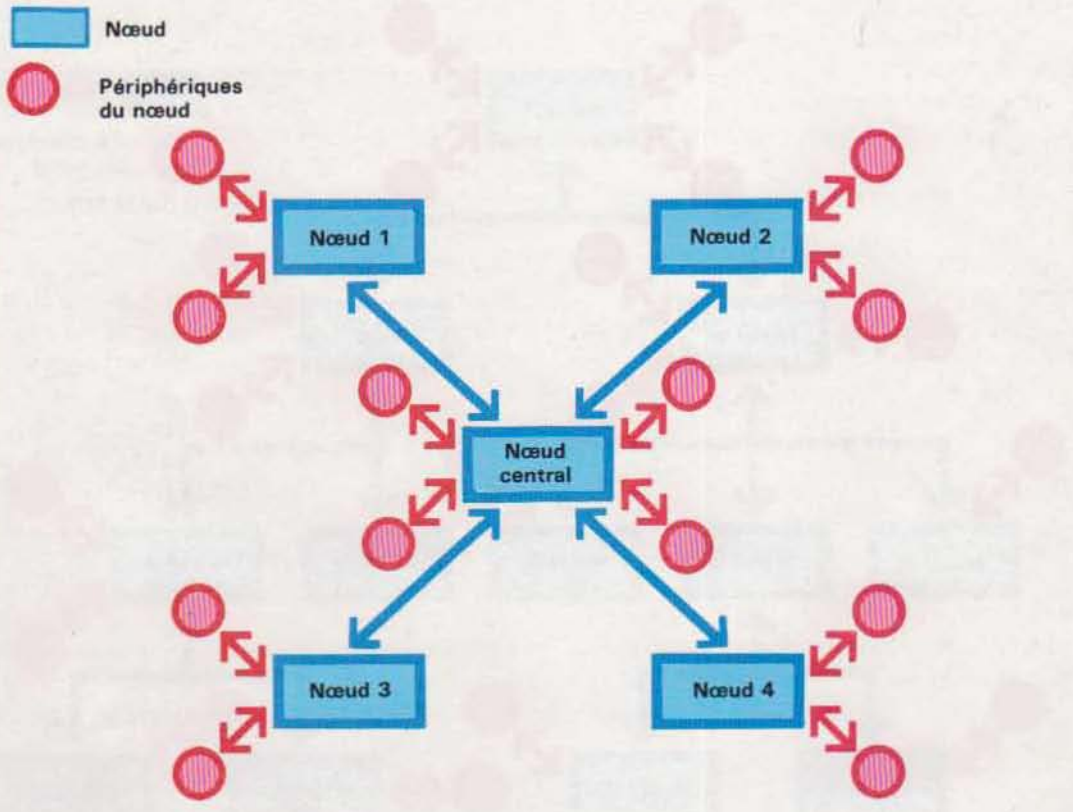
Réseau en anneau. C'est le premier exemple de réseau non asservi à un nœud (schéma 3, p. 1358). Les informations partant du nœud 1 vers le 3 transitent soit par le nœud 2, soit par le nœud 4. En général, on convient d'un chemin (par exemple en passant par le nœud 2) et, en cas de difficulté, on adopte une voie alternative (le nœud 4). Le réseau en anneau présente donc l'avantage d'établir une communication même quand surgit un problème au niveau d'un nœud. Seul inconvénient : son coût, chaque nœud étant doté de deux lignes de communication. Dans le réseau en anneau, aucun ordinateur (ou nœud) n'est privilégié par rapport aux autres, et ils ont la même puissance de calcul.

Réseau arborescent ou hiérarchisé (schéma 4, p. 1358). Dans cette architecture, chaque nœud a des fonctions bien particulières avec un rapport de dépendance à l'égard des autres nœuds selon des niveaux hiérarchiques. Le nœud 2 (p. 1359), par exemple, transfère des informations aux nœuds 4, 5 et 6 tandis que le 5 transmet vers les nœuds 9 et 10.
Pour communiquer depuis le nœud 10 avec le nœud 6, on doit passer la demande au nœud

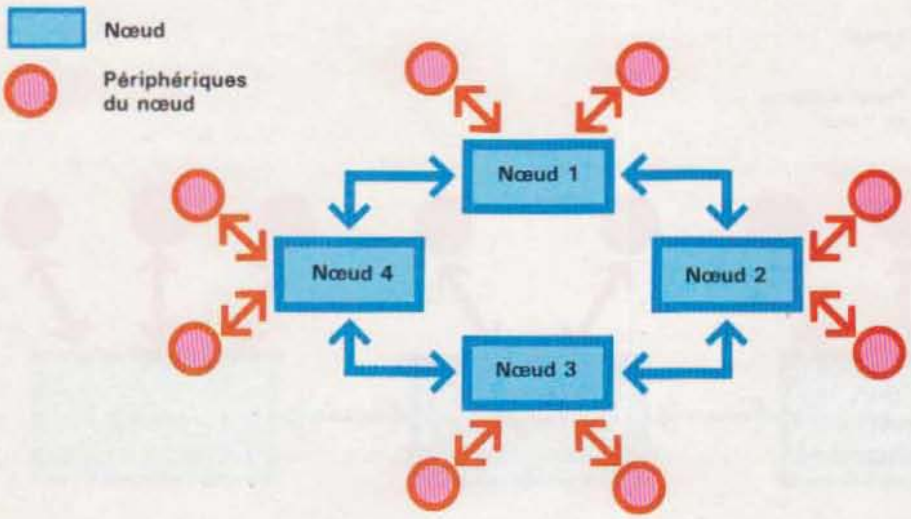
1 / RESEAU EN CHAINE



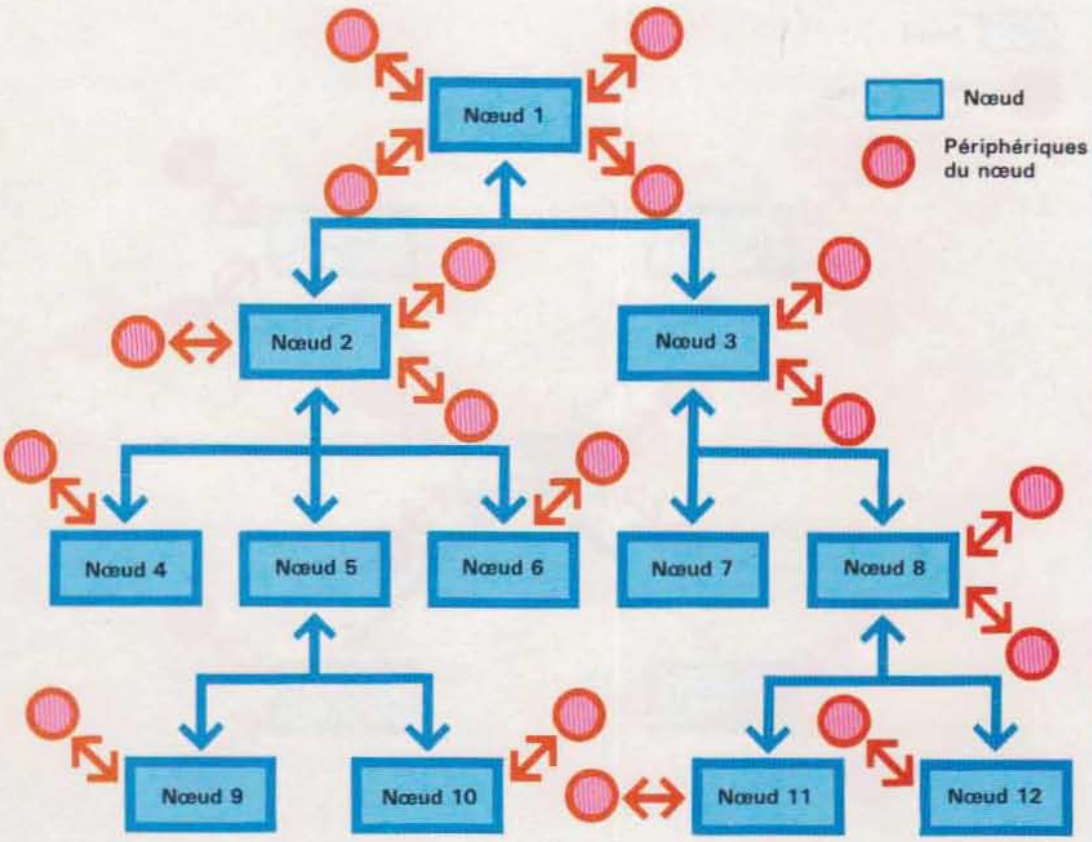
2 / RESEAU EN ETOILE



3 / RESEAU EN ANNEAU



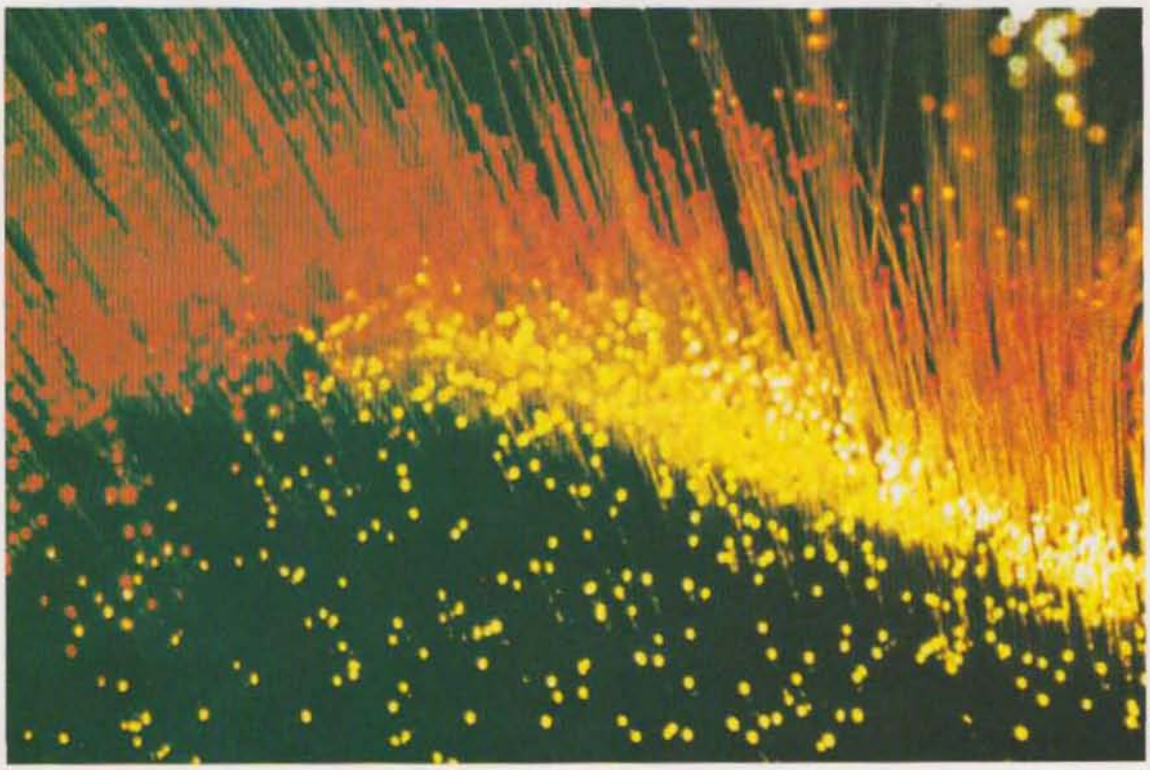
4 / RESEAU ARBORESCENT OU HIERARCHISE



supérieur (le 5), ce dernier « reportant » ensuite au 2 pour enfin aboutir au nœud d'arrivée 6. Si une demande de traitement a été adressée au nœud 9 alors que ce n'est pas de son ressort, elle sera « renvoyée » au nœud supérieur et ainsi de suite. La topologie hiérarchisée est surtout employée dans les réseaux où la sécurité d'accès aux informations est vitale (réseaux privés reliant des agences bancaires, etc.). En réalité, les architectures sont souvent imbriquées pour obtenir des réseaux certes complexes, mais bénéficiant des avantages de chacune d'entre elles.

Dans ces réseaux, deux propriétés revêtent une importance toute particulière : **stockage** et **retransmission** (store and forward) d'une part, et **communication par paquets** (packet switching) d'autre part. La première est une technique de transmission de messages entre deux nœuds non directement reliés. Dans un réseau, même complexe, un message a un nœud pour origine et un autre pour destination. Dans les protocoles modernes de communication (voir HDLC), le nom de l'émetteur et l'adresse du destinataire sont écrits dans le message même. Celui-ci devra donc passer par des nœuds intermédiaires qui, sans être les vrais destinataires, ont une fonction importante dans le réseau : ils stockent en mémoire le message si la ligne est occupée et le font « suivre » (forward) dès qu'elle se libère. C'est le cas dans l'architecture 1 par exemple (p. 1357) de la fonction du nœud 2 quand les deux autres (1 et 3) désirent communiquer. Ainsi, le nœud d'un réseau de type stockage et retransmission doit être suffisamment « intelligent » pour savoir si le message reçu lui est destiné ou ne fait que transiter vers le haut. La commutation par paquets est une autre technique, plus rapide, de transit d'un message à travers des nœuds qui n'en sont pas les destinataires. Avec cette méthode, les informations voyagent entre deux nœuds en suivant un chemin fixé à l'avance. En d'autres termes, s'il existe différentes voies reliant deux nœuds, la commutation par paquet est capable de déterminer quelle est la voie libre à un instant donné. Les informations sont empilées pour être ensuite transmises, toujours avec une longueur fixe, ce qui accélère la communication.
Imaginons, par exemple, un autobus avec 50

Conduction de lumière polychrome dans un faisceau de fibres optiques.



Schott

personnes à bord et dont l'itinéraire est de traverser la ville. Le chauffeur changera son trajet s'il se rend compte que celui-ci est embouteillé. Les 50 personnes sont synonymes d'autobus à pleine charge, pour que le voyage ne soit pas économiquement une perte. Il se produit la même chose avec des informations empilées et d'une longueur fixe.

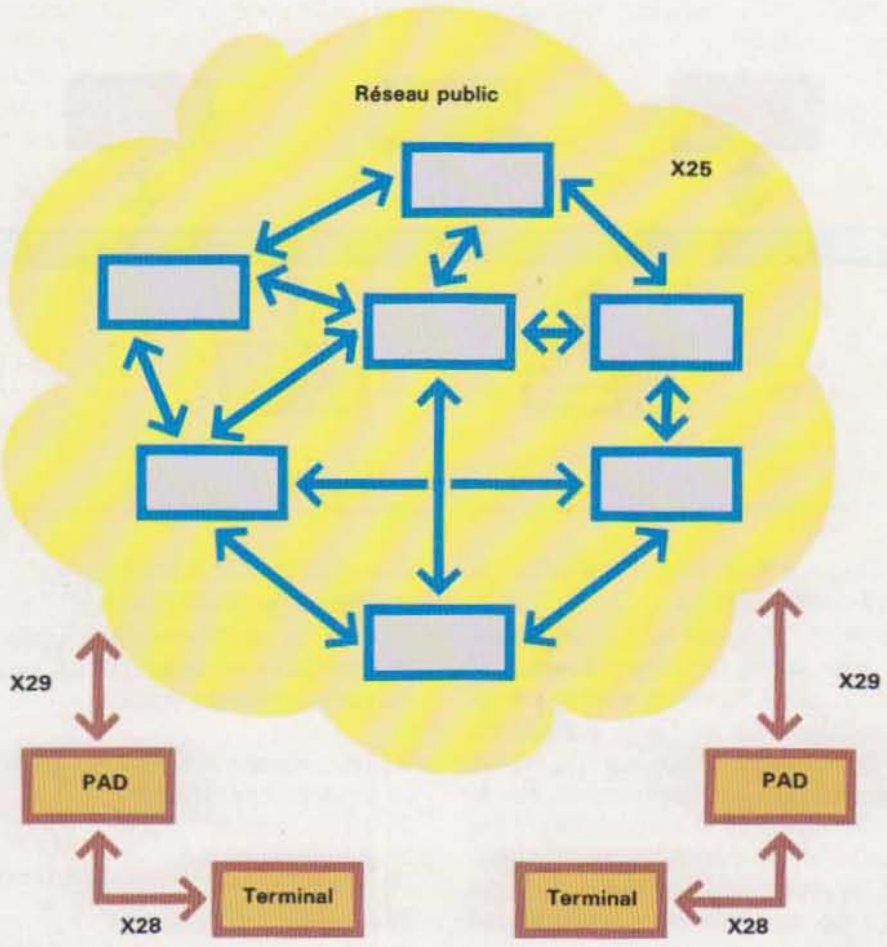
Réseaux publics

La diffusion des mini et des micro-ordinateurs a amené les PTT à créer des systèmes d'échange et de transfert d'informations.
Le concept nouveau, par rapport aux télécommunications traditionnelles, présente deux aspects : l'interconnexion des ordinateurs disponibles affectés à la gestion du trafic (échange d'informations dans le réseau), la comptabilisation de la durée de la communication sur d'autres bases que le temps.
Avant l'avènement des réseaux dits publics, pour que deux utilisateurs communiquent à distance, ils devaient demander une ou plusieurs lignes téléphoniques. La ligne n'était alors disponible qu'après avoir composé le numéro avec un débit faible, de l'ordre de 300 bps. La durée était comptée même s'il y avait des silences dans les communications après ouverture de la ligne.
Dans les réseaux publics, au contraire, seul est comptabilisé le volume d'informations transmises (les silences ne pèsent donc pas sur les coûts). Ce résultat est lié à l'emploi de la commutation par paquets et aux protocoles modernes grâce auxquels l'itinéraire emprunté par un message dans le réseau n'est pas toujours le même. Celui qui ne transmet pas est ignoré du réseau, alors qu'un émetteur aura la garantie que son message arrivera à destination, quelle que soit la voie prise.
Ce réseau public appelé Transpac (transmission par paquets) a été mis en service en France vers 1979. La plupart des réseaux publics nationaux obéissent à une standardisation des modes d'accès. Les normes font référence soit aux interfaces entre réseau et ordinateur (ou terminal) soit aux protocoles de communication.
Le Comité Consultatif International de Télégraphe et de Téléphone (CCITT) dont le siège est à Genève, a établi la recommandation qui, de nos jours, est la plus suivie pour les réseaux publics, et qui est devenue une norme de facto : X25. Elle fixe les règles que doivent suivre les nœuds par des dispositifs matériel et logiciel appelés PAD (pour Packet Assembler-Disassembler ou assemblage-désassemblage de paquets) et par des règles provenant de la recommandation X3 (toujours du CCITT). La liaison d'un terminal au PAD est gérée par la recommandation X28, celle du PAD au réseau par X29 (voir page ci-contre). Grâce aux recommandations du CCITT, la transmission de données a réalisé un progrès décisif, notamment en permettant à des ordinateurs d'origine différente de dialoguer entre eux.
Imaginons une société qui dispose d'un ordinateur de marque A à Lyon et d'un ordinateur de marque B à Paris et enfin d'un ordinateur de marque C à Brest. L'existence d'un réseau public respectant le protocole X25 oblige les constructeurs A, B et C à doter leur machine d'interfaces adéquates pour autoriser leur fonctionnement selon les modalités précises d'accès au réseau. Notre société pourra ainsi faire dialoguer ses 3 ordinateurs sans être obligée de construire un réseau privé.

Réseaux locaux

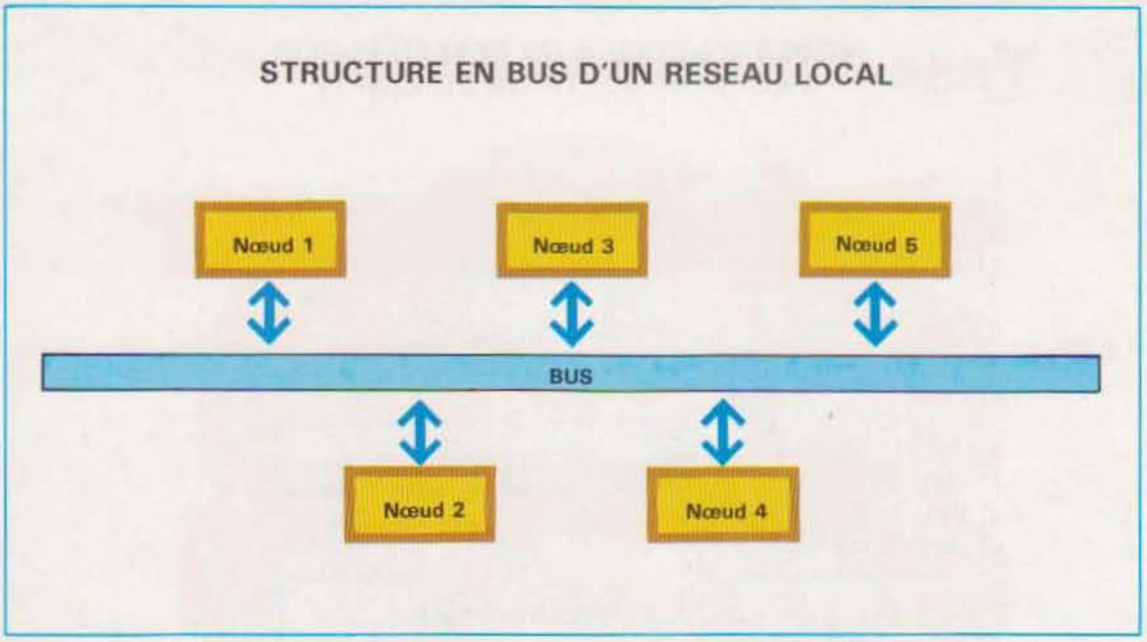
Les réseaux conçus sont généralement connus pour couvrir des aires géographiques étendues. Mais on peut aussi avoir besoin de distribuer des informations dans un espace plus restreint, de l'ordre de quelques kilomètres voire de quelques centaines de mètres. Ce sera le cas d'une société ayant installé ses ordinateurs dans un même bâtiment, ou dans plusieurs bâtiments voisins. Dans ce cas, il vaut mieux concevoir un « réseau local » dans lequel on ne recourt pas aux lignes téléphoniques.
Le plus gros avantage des réseaux locaux, par rapport aux réseaux publics, sera la vitesse de communication entre les nœuds. On peut, en effet, atteindre jusqu'à 100 millions de bits par seconde (alors que dans les réseaux étendus, on ne dépasse pas les 50 à 60 000 bps).
Les lignes de communication sont des câbles (en tresse ou coaxiaux) aboutissant à des dérivations sur lesquelles sont connectés les nœuds. Dans un bâtiment, le réseau local est installé en passant un câble dans les locaux où se trouveront les nœuds, en prévoyant autant

MODES D'ACCES A UN RESEAU PUBLIC (RECOMMANDATIONS DU CCITT)



de prises qu'il y aura d'appareils à connecter. La fibre optique commence à se subtituer au câble comme support physique. Cette solution offre la plus grande garantie pour la transmission. Dans la fibre optique, les signaux se propagent à la vitesse de la lumière sous forme d'impulsions, non électriques, et donc la communication n'est pas troublée par l'extérieur. Les topologies du réseau local (souvent défini par le sigle LAN, Local Area Network) sont les mêmes que pour les réseaux en général (en étoile ou en anneaux) avec, en plus, une nouvelle configuration, appelée **bus** (voir page 1362). Dans cette dernière structure, il y a une totale indépendance entre nœuds donc remplacement d'un nœud indisponible.

Chaque nœud envoie un message le long du bus dans les deux directions, et le nœud destinataire le saisit s'il lui est adressé. Le protocole pour les communications dans les réseaux locaux a été mis au point par l'Institute of Electrical and Electronics Engineers (IEEE). La recommandation IEEE 802 établit les distances maximales, les nœuds, les vitesses et les modalités de transmission de données pour les réseaux locaux. Une solution intéressante consiste en l'adoption, comme moyen de communication, des lignes téléphoniques internes à la société. Elles sont gérées par un système appelé PBX (Private Branch X-change) qui permet de faire dialoguer deux bureaux ou d'orienter vers les différents bureaux les appels venant de



l'extérieur. On peut aussi relier l'ordinateur au réseau téléphonique intérieur.
Les équipements connectables sur un réseau local sont très variés : micro-ordinateur et mini-ordinateur, unité bande ou disque, terminaux, imprimantes... La tendance actuelle est de connecter au réseau beaucoup d'appareils de traitement et peu de périphériques, car le coût des unités centrales va en décroissant (ce qui n'est pas le cas des périphériques). L'objectif est donc de partager, entre plusieurs unités centrales, les mêmes unités disque ou bande et la même imprimante.
La phase de croissance suivante, pour un réseau local, est de se connecter aux réseaux externes, donc la possibilité pour tous les dispositifs d'un LAN de s'interfacer à un réseau public, pour accéder à des banques de données et à des puissances de traitement géographiquement éloignées.
De nombreuses études sur ce sujet sont aujourd'hui en cours et beaucoup de solutions s'inspirent de l'emploi des satellites pour amplifier le volume de données transmises dans un même laps de temps.

Vers des architectures standard

Tous les instruments utilisés dans la communication de données, comme les lignes, les protocoles, les réseaux... constituent les briques d'un édifice dont la construction n'est pas encore achevée. C'est l'architecture générale des réseaux de communication. Deux architectures semblent émerger du lot pour unifier la communication de données :

1 / Le réseau SNA (System Network Architecture) d'IBM,
2 / Le modèle OSI (Open System Interconnection).

Construites par couches, ces architectures obéissent à des conceptions différentes. La première couche décrit la manière de réaliser les connexions électriques, donne les valeurs des signaux, etc. La deuxième couche établit les modalités et les protocoles de connexion et ainsi de suite, jusqu'à définir le projet au niveau le plus élevé.
Les deux architectures SNA et OSI comportent sept couches (qui ne sont pas identiques dans les deux solutions). SNA, mise au point par IBM, constitue le modèle dont s'inspirent les réseaux établis par cette compagnie. L'architecture OSI a été mise au point par l'Organisation internationale de normalisation (ISO). L'objectif du modèle est de définir un modèle d'architecture de réseau souple dont pourraient s'inspirer tous les autres constructeurs d'ordinateurs. Déjà beaucoup y font référence tant pour le matériel que pour le logiciel.

Test 23

1 / La fonction de handshake ("main tendue") sert à :

a) relier deux ordinateurs
b) synchroniser celui qui émet et celui qui reçoit
c) construire un protocole
d) faire communiquer à distance deux dispositifs

---

2 / Les caractères SYN sont employés :

a) dans le protocole HDLC
b) dans le protocole BSC
c) dans les protocoles asynchrones
d) dans le contrôle des erreurs

---

3 / Dans le protocole HDLC, le texte transmis peut être :

a) composé de caractères seulement
b) composé de nombres seulement
c) quelconque
d) semi-duplex seulement

---

4 / Le protocole IEEE-488 est :

a) sériel
b) parallèle
c) semi-duplex
d) pour les longues distances

---

5 / La ligne ATN (attention) est employée dans :

a) le protocole BSC
b) le protocole HDLC
c) le protocole IEEE-488
d) aucun d'entre eux

---

6 / La commutation par paquets (packet switching) sert à :

a) commuter deux signaux sur un réseau
b) envoyer à destination un message en suivant des voies différentes
c) échanger un message
d) exclure un message

---

7 / Un réseau public permet :

a) uniquement l'usage du protocole IEEE-488
b) uniquement la transmission de données dans une zone limitée
c) la liaison de dispositifs géographiquement distants
d) l'emploi de cables coaxiaux

8 / L'architecture en bus est employée dans :

a) les réseaux publics
b) les réseaux locaux
c) le protocole X3
d) les connexions sérielles

Les solutions du test se trouvent en page 1367

Instructions du Basic pour la communication de données

Même en Basic — le langage le plus répandu en micro-informatique — on peut gérer les communications de données. Les intructions disponibles sont toutefois limitées en nombre et toujours étroitement liées au type de matériel. Autrement dit, il n'existe pas d'instructions standard et, par la suite, nous présenterons, à titre d'exemple, certaines instructions capables de résoudre des problèmes de communication asynchrone sur RS232.

La mise en route d'une communication traverse les principales phases suivantes :

— ouverture du canal avec une instruction du type OPEN,
— lecture ou écriture de caractères sur le canal précédemment ouvert.

La logique suivie consiste à assimiler le canal de communication à un fichier et donc à réaliser les opérations d'E/S comme s'il s'agissait d'un accès à un fichier.

L'instruction OPEN réserve un tampon d'E/S considéré comme un fichier. En général, les paramètres à fournir sont :

N Numéro du canal. Il correspond parfois au n° d'une porte d'E/S ou d'un adaptateur.

V Vitesse de transmission (bits par seconde : bps). On prévoit en général les vitesses de 75, 110, 150, 300, 600, 1 200, 1 800, 2 400, 4 800, 9 600 bps, la plus courante étant 1 200 bps.

P Parité. Elle indique la méthode employée pour contrôler la parité du caractère.

Il existe plusieurs méthodes :
— absence de contrôle ; le bit de parité n'est pas transmis
— absence de contrôle ; bit de parité toujours transmis avec la valeur 0
— absence de contrôle ; bit de parité toujours transmis avec la valeur 1
— contrôle de parité paire
— contrôle de parité impaire.

Ce choix peut sembler trop large car il inclut des cas que l'on rencontre rarement dans la réalité. En fait, il faut dire que bien souvent on doit « interfacer » deux ou plusieurs appareils de différents constructeurs gérés par des systèmes et programmés avec des langages différents. Dans ce cas, la variété des interfaces est une condition essentielle pour établir la connexion.

D Nombre de bits constituant un caractère (bit de parité exclu). Les valeurs admises sont 4, 5, 6, 7, 8 ; le plus courant est 7.

S Nombre de bits de stop ; les valeurs les plus employées sont 1 et 2.

F Numéro du fichier logique associé au canal, à utiliser dans les opérations d'E/S.

C Nombre maximum d'octets pouvant être contenus dans le tampon de communication.

Par exemple l'instruction :

OPEN 'COM 3 : 1200, S,7,1' AS 1,128

ouvre un fichier de communication avec les paramètres suivants :

— numéro du canal : 3
— vitesse de transmission : 1200 bps
— bit de parité toujours à 1 (on suppose qu'il est possible d'indiquer cette option avec la lettre S ; en réalité il faut se référer au manuel d'utilisation de l'ordinateur)
— nombre de bits composant un caractère : 7
— nombre de bits de stop : 1

Le mot réservé AS suivi des paramètres 1,128 assigne ce canal au fichier 1 avec un tampon de 128 caractères.

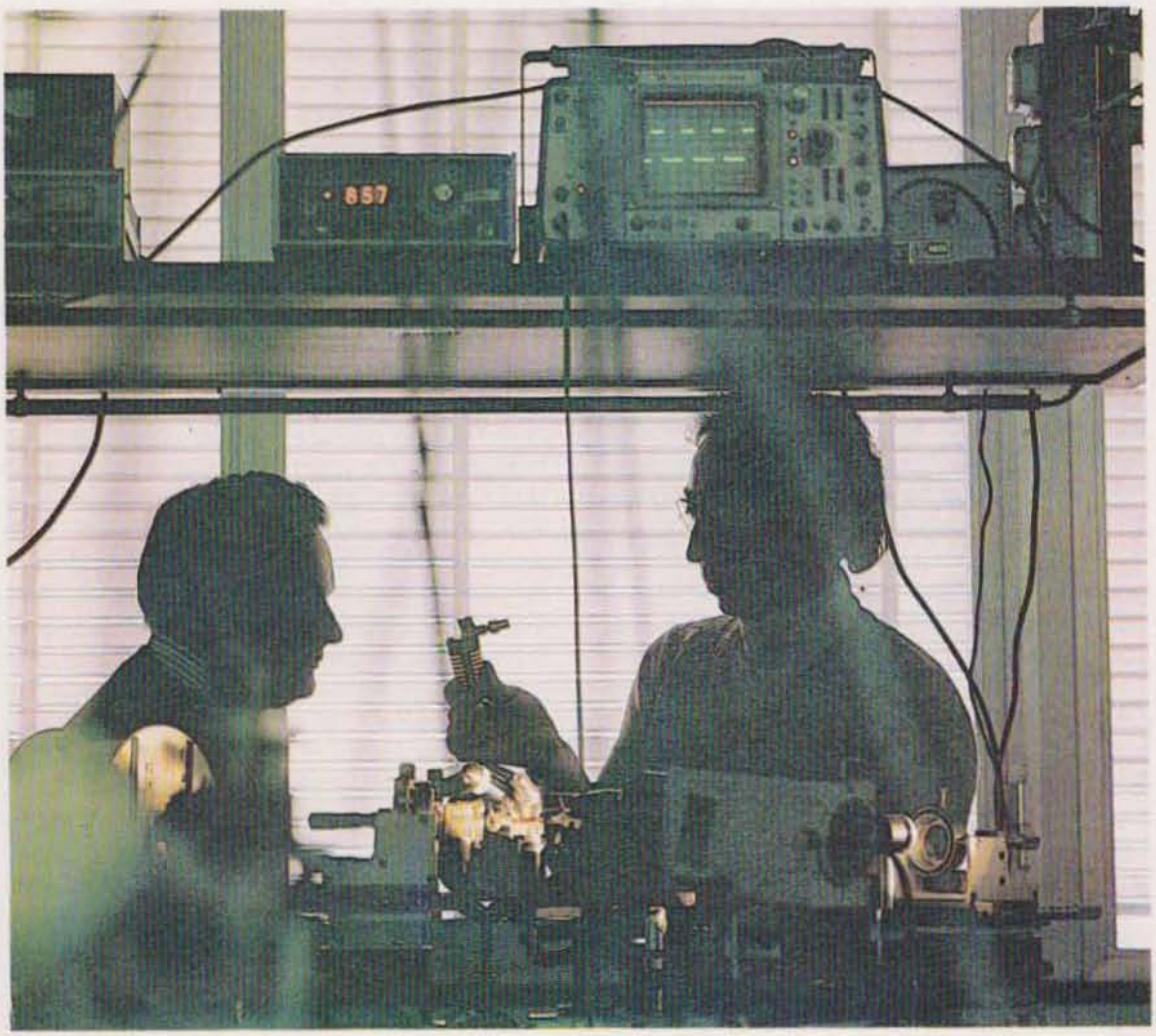
En plus de ces paramètres principaux, il faut aussi, dans certains cas, gérer l'état des signaux. En particulier, si on envisage de placer un modem dans la ligne, il sera nécessaire de disposer d'indicateurs spécifiant l'état des lignes (actif/non actif).

Les principales lignes employées dans ce type de communication sont :

— Request to Send
— Clear to Send
— Data Set Ready
— Carrier Detector.

Leur emploi dépend du type de matériel et du logiciel. Dans certaines applications, on peut, par exemple, employer ces lignes pour générer des interruptions et lancer un sous-programme de saisie d'un caractère. Une fois les paramètres définis, le logiciel déclenche une boucle d'attente, au cours de laquelle le processeur examine continuellement l'état d'un signal indiquant une liaison active (par exemple le Carrier Detector).

Conduction de lumière laser dans un faisceau de fibres optiques.



Italtel

Cette boucle n'est abandonnée que dans deux occasions :

— après une interruption
— après arrêt du signal.

L'interruption indique l'arrivée d'un caractère. La boucle est provisoirement abandonnée et la routine de saisie est mise en route. A la fin, on reprend le cours du programme au point où il a été interrompu (la boucle d'attente est donc relancée) jusqu'à l'arrivée du caractère suivant ou à la fin du signal de contrôle. Dans ce dernier cas (fin de la liaison), le programme s'arrête.
Ce type de gestion n'est pas aussi simple qu'il paraît. Tout d'abord, il ne fonctionne que sur certaines machines et dans certaines configurations ; de plus, au niveau du logiciel il est rendu encore plus complexe par toute une série de contrôles et de vérifications. Les principaux contrôles à effectuer sont les suivants :

— time out (temps de débouclement)
— remplissage du tampon
— parité
— priorité des interruptions
— fin de la ligne ou autres interruptions accidentelles.

On parle de **time out** lorsque, entre eux deux caractères successifs, l'intervalle de temps écoulé est supérieur au maximum prévu : le logiciel doit alors être en mesure de répéter cette situation anormale, et d'y remédier. Normalement, on envoie un message de diagnostic et on interrompt la liaison : il peut arriver que le dépassement ne soit pas dû à une situation anormale, mais simplement à une vitesse de transmission mal adaptée. Dans ce cas, c'est le logiciel qui résout ce défaut.

Le **remplissage du tampon,** surtout en Basic interprété, se produit quand le programme doit faire trop de traitements dans le sous-programme de saisie. Dans cette routine, il faut désamorcer l'interruption, sinon, à l'arrivée d'un caractère, la routine s'auto-appellerait, éventuellement plusieurs fois. La routine ne peut alors être interrompue et, si son déroulement prend trop de temps, les caractères qui continuent d'arriver font déborder le tampon. D'où génération d'une erreur ou, pire, un programme faussé.
Une manière d'éviter cet inconvénient consiste à réduire au minimum les traitements de la routine de saisie, mais même cette solution peut présenter des risques ; ainsi, en éliminant les contrôles, on peut accepter comme donnée valable une valeur erronée. Il existe toutefois des interfaces capables d'effectuer le contrôle sur les caractères reçus. En cas d'erreur, ces interfaces envoyent à l'ordinateur un code spécial à interpréter dans la routine de saisie (dans laquelle il faut toujours insérer les principaux contrôles).

Le **contrôle de parité** constitue un autre point critique du programme de gestion des communications. Normalement, le contrôle est réalisé par le matériel de manière totalement transparente. On informe le programme de l'existence d'une erreur avec un code particulier et l'activation d'un signal.
Dans ce dernier cas, en plus de la ligne pour l'arrivée des données, il faut activer une autre ligne d'interruption avec les complications qui peuvent en découler par rapport à la gestion des priorités.
Au moment de la génération, les **interruptions provoquent une réservation de service.** Il se crée donc une file de demandes qui s'amenuise au fur et à mesure que le système répond à celles qui ont la plus forte priorité. Le déroulement, complètement asynchrone, n'est nullement lié aux attentes de l'ordinateur. La mise en file des diverses interruptions n'a donc pas d'influence sur le flux des données en arrivée, qui s'accumulent dans le tampon. Si le système est dédié à des tâches prioritaires, il ne peut prélever les données du tampon (avec la routine de saisie) ; il se produira un phénomène analogue à celui de la saturation du tampon à cause de la lenteur excessive du traitement. Ce défaut est encore plus difficile à surmonter, car il s'agit d'une cause externe au flux normal du programme. Il faut alors assigner, à l'interface de communication, la priorité maximale.
Il faut enfin examiner l'éventualité d'une **panne** bloquant le signal au niveau actif, même quand la transmission est terminée. Dans ce cas, la boucle de saisie, utilisant ce signal comme indicateur d'état, ne s'interromprait jamais

Solutions du test 23

1 / b : la fonction de handshake sert à synchroniser l'ordinateur émetteur et l'ordinateur récepteur.

2 / b : les caractères SYN sont employés dans le protocole BSC

3 / c : dans le protocole HDLC le texte transmis peut être quelconque

4 / b : le protocole IEEE-488 est parallèle

5 / c : la ligne ATN (attention) est employée dans le protocole IEEE-488

6 / b : la commutation par paquets (packet switching) sert à envoyer à destination un message en suivant des voies différentes

7 / c : un réseau public permet la liaison de dispositifs géographiquement distants

8 / b : l'architecture en bus est employée dans les réseaux locaux

et l'ordinateur resterait en attente sans fournir aucun message de diagnostic.
Cette éventualité est à introduire dans la gestion du time out, en prévoyant un laps de temps opportun, bien supérieur à celui nécessaire à la réception d'un caractère ; le programme s'arrêtera alors ensuite, quoi qu'il arrive (par time out).
Pour gérer correctement une interface de communication, il faut donc disposer d'un temporisateur, une horloge capable de signaler le temps passé. Les instructions liées à la question de cette minuterie ont une syntaxe dépendant du matériel.
Une formule très employée est :

TIME$-A$ pour définir le temps
B$-TIME$ pour définir la lecture du temps.

En général, la valeur temporelle est exprimée comme une chaîne de 6 caractères, avec heure, minutes et secondes, dans le format HH MM SS. La première instruction définit la valeur indiquée en A$, la seconde lit la valeur courante comme différence entre la valeur définie et la valeur lue.
La gestion du time out s'obtient en contrôlant le temps passé dans la routine de saisie ; s'il est supérieur à une valeur maximum fixée, on met en action un indicateur d'erreur et on arrête le programme ; sinon on met à zéro le temporisateur pour qu'il soit prêt au contrôle suivant.
On mesure ainsi l'intervalle de temps entre l'arrivée d'un caractère et le suivant ; mais ce contrôle ne couvre pas toutes les possibilités de time out. Le programmeur devra donc déterminer quels sont les autres points critiques du programme et prévoir des contrôles.
Malgré ses limites, le Basic est encore le seul langage (en dehors de l'Assembleur) à permettre ce type d'applications dans la catégorie des micro-ordinateurs.
Les autres langages ne prévoient en général aucune instruction pour la communication de données.

Systèmes d'exploitation

Le fonctionnement d'une application développée sur micro est en général guidée par l'interpréteur Basic. Les calculs, les entrées et les sorties de données sont préparées dans le programme qui offre à l'utilisateur un moyen d'exploiter de manière efficace la machine.
En généralisant ce concept, on arrive à la définition du **système d'exploitation :** un ensemble de programmes et de procédures qui permet d'exploiter l'ensemble des ressources de l'ordinateur.
Même quand il s'agit d'un ordinateur personnel, le Basic ne suffira pas toujours, à lui seul, pour en coordonner toute l'activité. Il faut donc utiliser des systèmes d'exploitation comme le CP/M, le MPM, etc.
En général, plusieurs systèmes d'exploitation peuvent résider en mémoire d'ordinateur et leur intervention est liée aux nécessités de l'application déroulée.
L'utilisateur qui crée des programmes faisant appel à beaucoup de fichiers choisira, par exemple, un système d'exploitation orienté multiprogrammation.
Si, en théorie, il est toujours possible de mettre un quelconque système d'exploitation sur un matériel, dans la pratique l'ordinateur est, à l'origine, doté d'un système d'exploitation mis au point par le constructeur. C'est ce système qui épouse d'ailleurs le mieux la partie matériel et qui en optimise l'emploi.
Dans la définition d'un système d'exploitation, on a évoqué les **ressources.** Ce terme fait référence à n'importe quelle caractéristique d'un système de traitement tel que la mémoire centrale, les mémoires de masse, le temps machine...
Le système d'exploitation agit comme coordinateur de ces ressources.
Pour expliquer son fonctionnement, on fera constamment référence au système Unix qui a été mis au point par les laboratoires Bell. Unix s'est très vite fait une place dans les universités américaines par sa facilité d'emploi et par son caractère convivial. A l'heure actuelle, tous les fabricants d'ordinateurs ont tendance à doter leur machine de ce système.

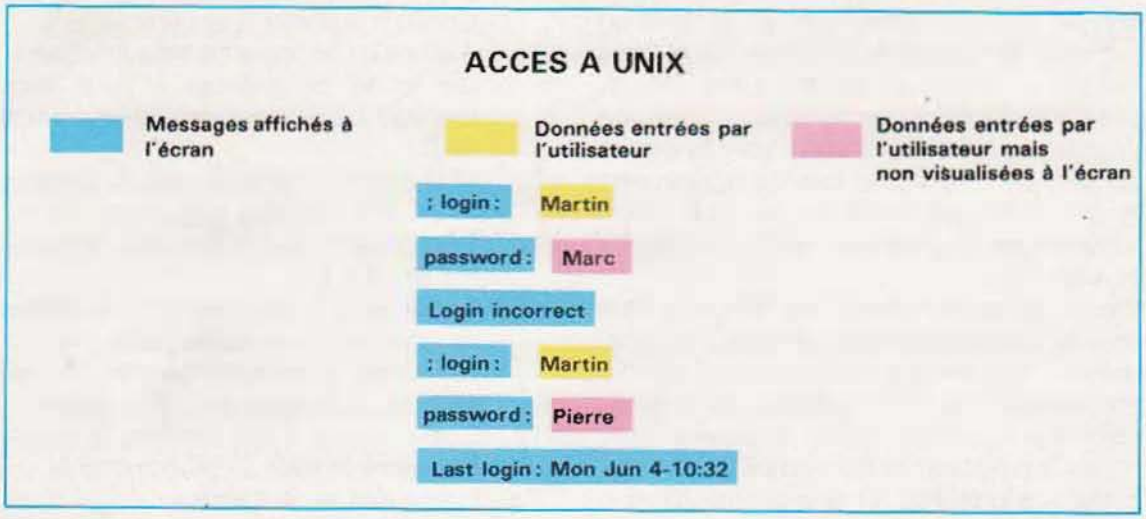
Mono-utilisateur et multi-utilisateur

Ces caractéristiques indiquent combien de personnes peuvent se servir simultanément d'un ordinateur. Aux points d'accès correspondent les écrans et les claviers :

— En **mono-utilisateur,** on dispose d'un seul terminal, qui correspond à la machine elle-même.

— Le **multi-utilisateur,** constitue, en revanche, une caractéristique typique des mini-ordinateurs et des gros ordinateurs.

Les systèmes d'exploitation coordonnent l'accès à l'ordinateur des divers utilisateurs, en répartissant entre eux le temps-machine, la mémoire, les fichiers et en n'acceptant que les personnes dûment autorisées.
Le nombre d'usagers en titre est défini, et l'accès est lié à l'introduction d'un mot de passe confidentiel.
L'accès **(login)** à un ordinateur géré par Unix est représenté schématiquement dans la figure ci-contre.
L'utilisateur frappe sur une touche quelconque et le système répond avec la chaîne « ;login : ». Il signe en introduisant son nom et le système demande son mot de passe.
Dans l'ordinateur, on a déjà défini le mot de passe pour « Martin » qui correspond au mot « Pierre » ; si, comme mot de passe, on introduit « Marc », Unix refuse l'accès en visualisant le message « Login incorrect ».
En retapant une touche quelconque, « ;login : » apparaîtra à nouveau comme une invitation à s'identifier.
Si le mot de passe est correct, il a accès au système ; Unix visualisera la date à laquelle « Martin » a fait la dernière connexion et le caractère qui signifie « je suis prêt à accepter les instructions de Martin ».
Les codes confidentiels (mots de passe) introduits ne sont pas visualisés, pour éviter que des usagers non autorisés n'en prennent connaissance.



Traitement par lots et interactivité

Les systèmes d'exploitation réalisant du **traitement par lots** ont, pendant longtemps, été les seuls outils à la disposition des utilisateurs. Exécutant une seule tâche à la fois, ils sont désormais désuets. Les dispositifs d'entrée étaient constitués de lecteurs de cartes perforées grâce auxquelles programmes et données à traiter étaient chargés. Les divers travaux à exécuter par la machine étaient organisés en file d'attente. Le système d'exploitation les traitait l'un après l'autre, selon la technique FIFO First In/First Out (premier entré/premier sorti) : la priorité était accordée au premier travail chargé en mémoire centrale.

A l'inverse, les systèmes d'exploitation dits « interactifs » travaillent en temps réel. Les terminaux sont donc des dispositifs interactifs d'entrée-sortie. Un caractère (ou une chaîne de caractères) indique à l'écran que la ligne utilisateur est ouverte. Avec le système interactif Unix, ce caractère est le symbole dollar ($). Les ordinateurs personnels sont, eux aussi, commandés par des processus interactifs tels que Basic ou CP/M. Dans un ordinateur, c'est le système d'exploitation (à traitement par lots ou interactif) qui a le contrôle des opérations.

Imaginons, à titre d'exemple, qu'un programme d'application exécute une opération interdite : le système d'exploitation doit la détecter, la signaler et reprendre le contrôle. Dans un système de traitement par lots, cela signifie abandonner le travail en cours et passer au suivant ; dans le cas d'un système interactif, c'est afficher le caractère guide-opérateur.

Multiprogrammation

La multiprogrammation n'existe pas dans toutes les machines, mais c'est une caractéristique commune à tous les systèmes d'exploitation dits « multi-utilisateurs », ce qui permet de faire exécuter simultanément différents programmes.

La **multiprogrammation** a été conçue, à l'origine, pour exploiter les temps morts de la machine. Supposons qu'un opérateur soit en train d'entrer des données à partir d'un terminal. L'exécution du programme n'interviendra qu'en fin de saisie. Or le temps de saisie (quelques secondes) est en fait extrêmement long en comparaison des temps de traitement des ordinateurs. Si le système d'exploitation le permet, la machine peut alors exécuter un autre programme pendant que l'opérateur entre ses données.

La multiprogrammation est donc la capacité d'un système d'exploitation à gérer simultanément plusieurs programmes résidents en mémoire d'ordinateur. Mais gérer ne veut pas dire exécuter simultanément. En effet, si l'ordinateur n'a qu'une seule unité centrale (UC), celle-ci ne peut, à un instant donné, s'occuper que d'une seule tâche. Il faut comprendre la gestion simultanée comme la capacité d'exploiter les temps morts en suspendant temporairement l'exécution d'un programme pour donner la possibilité à un autre d'être exécuté.

Plusieurs programmes en machine demandent généralement à être exécutés en même temps. C'est pourquoi les systèmes d'exploitation

orientés multiprogrammation doivent gérer un ordre de priorité des programmes. Supposons que cet ordre soit un nombre entier compris entre 1 et 99 et que le niveau de priorité soit décroissant : les programmes ayant la priorité 20 sont plus importants que les programmes de priorité 40, qui sont à leur tour plus importants que les programmes ayant la priorité 60, et ainsi de suite.

Ces programmes forment une queue souvent appelée **liste d'ordonnancement**. Pour chaque cas, le système d'exploitation doit décider quel programme il doit exécuter en premier, comme le montre le schéma ci-dessous : parmi les 3 programmes demandant à être exécutés, A a la priorité 30, B la priorité 20, et C la priorité 10.

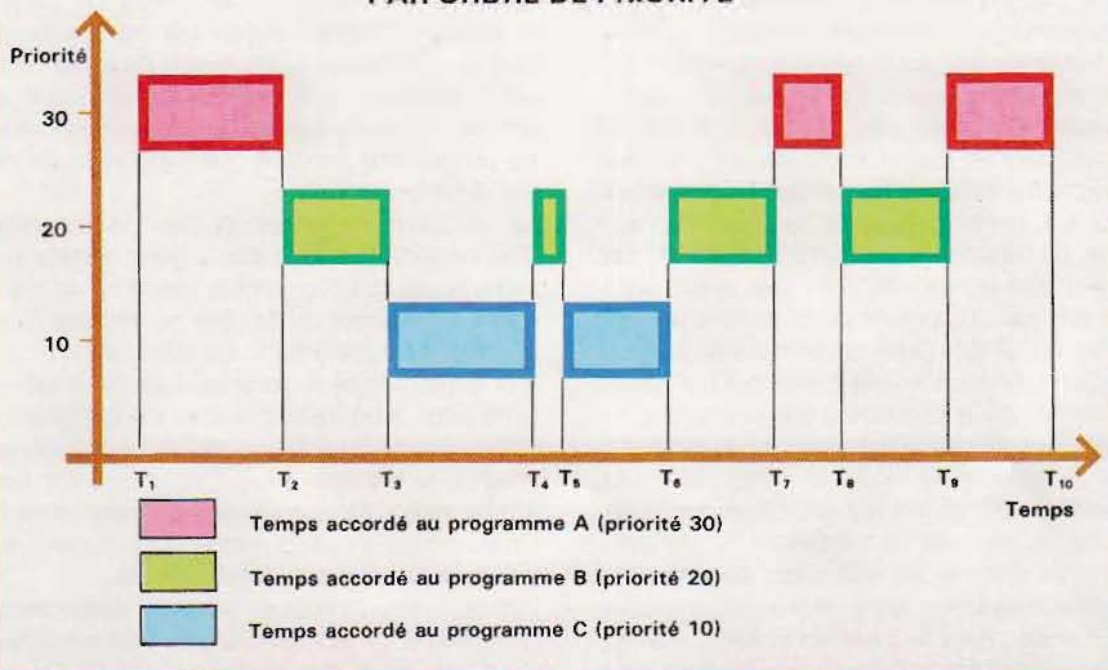
Voici la description de ce qui se passe entre T_1 et T_{10} :

- T_1 La machine exécute d'abord A, le seul programme résident à cet instant-là.
- T_2 Le programme B (plus prioritaire que A) demande à être exécuté ; le système d'exploitation suspend A et lance B.
- T_3 Le programme C (plus important que B) demande à être traité ; le système d'exploitation suspend B et commence C.
- T_4 C effectue une opération lente (par exemple entrée de données à partir d'un terminal) ; le système reprend l'exécution de B.
- T_5 La phase lente de l'exécution de C prend fin ; B est suspendu et C repris.
- T_6 L'exécution de C est terminée ; le système passe à B.
- T_7 B exécute une opération d'entrée/sortie de données ; le système passe à A.
- T_8 La phase d'entrée/sortie de B se termine ; A est suspendu et B reprend.
- T_9 L'exécution de B est terminée, le système donne la main au programme A.
- T_{10} L'exécution de A s'achève.

Ainsi, la multiprogrammation par ordre de priorité ne fait pas qu'exploiter les temps morts (l'activité de l'UC ne cesse à aucun moment) ; elle permet également d'établir une hiérarchie des programmes d'après leur importance. L'exemple donné pourrait correspondre à un ordinateur relié à trois terminaux exécutant chacun un programme A, B ou C.

Que se passe-t-il quand deux programmes ont la même priorité, 50 par exemple, l'un (X) étant

MULTIPROGRAMMATION : EXECUTION DE TROIS PROGRAMMES PAR ORDRE DE PRIORITE



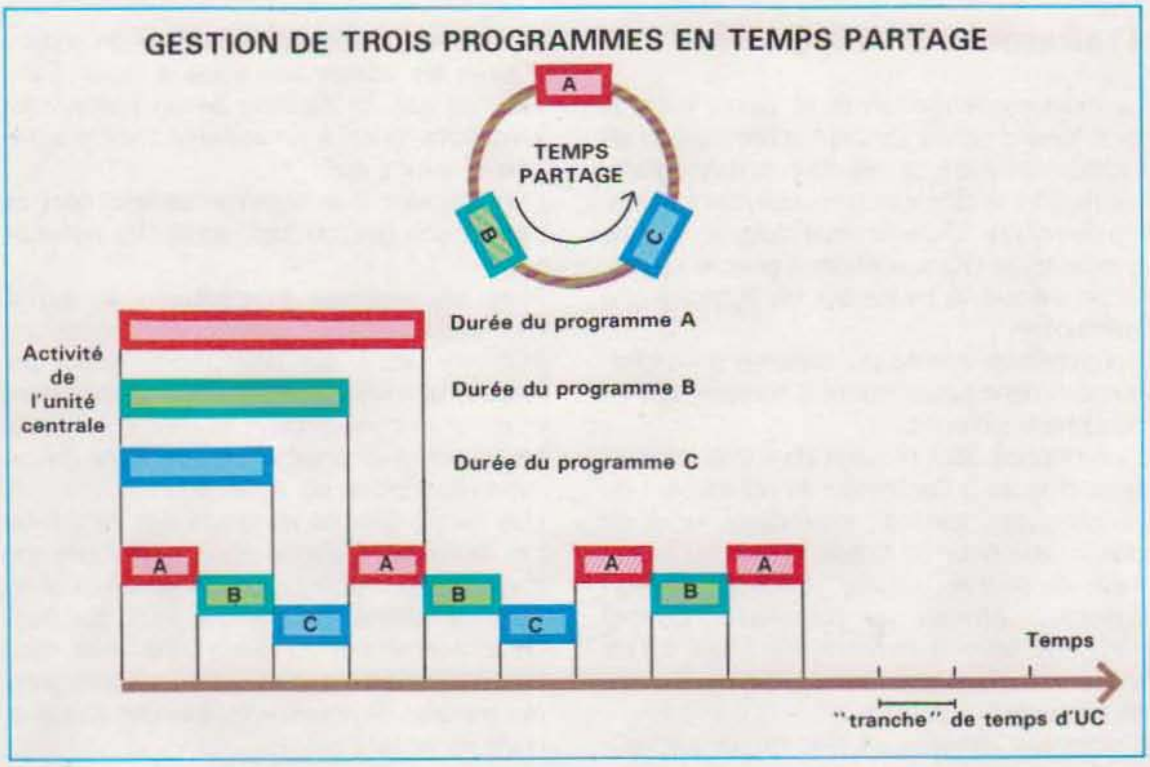
en cours d'exécution, l'autre en (Y) en attente ? Deux solutions sont envisageables :

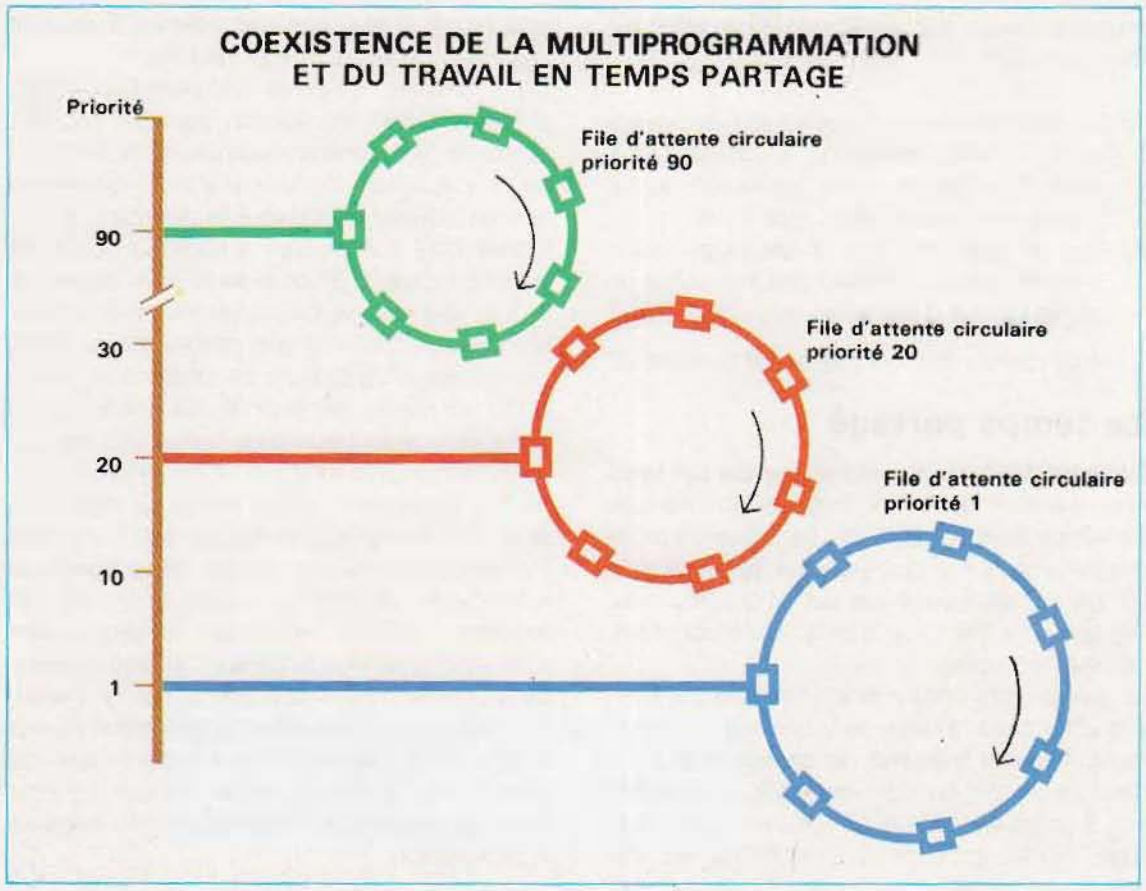
1/ Le programme X doit être terminé avant qu'X ne soit commencé, à moins que X n'ait à effectuer une opération lente, auquel cas Y peut alors être traité.
2/ Les programmes X et Y seront exécutés alternativement pendant des intervalles de durée définie à l'avance.

C'est ce dernier cas qui est décrit ci-dessous.

Le temps partagé

Exécuter des programmes en **temps partagé** revient à affecter, à chacun d'eux, un intervalle de temps bien précis. L'UC ne travaille sur un programme donné que pendant le temps prévu. Elle passe ensuite aux autres programmes, l'un après l'autre, jusqu'à ce qu'elle recommence alors le cycle.
La gestion des programmes est effectuée cycliquement sur la base de la période de traitement, appelée **tranche de temps d'UC.**
Dans l'exemple illustré ci-dessous, nous avons pris 3 programmes : A, B et C, en temps partagé. Aucun concept de priorité ne leur est attaché, A, B et C ont « les mêmes droits » et l'UC s'en occupera à tour de rôle.
Dans certains systèmes d'exploitation, multiprogrammation et temps partagé peuvent coexister (voir schéma page suivante). En théorie, il y a autant de files d'attente circulaires que de valeurs affectées à la priorité.
L'ordinateur servira tout d'abord la queue de priorité 1, puis la 20, et enfin la 90. L'utilisateur a, à sa disposition, les moyens de hiérarchiser les niveaux d'intérêt des programmes. Ainsi, un groupe d'utilisateurs se divisera le temps d'UC au niveau de priorité 50, tandis qu'un autre groupe se placera au niveau 20, etc.
Le système d'exploitation UNIX fonctionne en multiprogrammation et en temps partagé, mais la priorité d'un programme n'est pas établie par l'utilisateur. Au début de leur exécution, tous les programmes sont au niveau de priorité, par exemple 10. Plus l'exécution du programme s'avère longue, plus UNIX augmentera la valeur de la priorité (en « déclassant » le programme). Ce mécanisme pénalise les programmes qui durent trop longtemps et favorise ceux qui sont courts. L'objectif recherché est d'encourager au maximum l'interactivité du système d'exploitation.





Traitement en temps réel

La multiprogrammation et le temps partagé sont fondés sur le concept d'interruption de l'activité en cours. Si, pendant qu'un programme de priorité 50 s'exécute, un autre programme de priorité 40 demande la main, le système d'exploitation doit suspendre le premier au profit du second. A ce dernier est associée une interruption.

L'architecture interne du système d'exploitation doit donc lui permettre d'accepter des interruptions externes.

L'interruption peut provenir de n'importe quel point d'accès à l'ordinateur et notamment du périphérique : terminal, imprimante, unité de disque, dérouleur de bande mais aussi instrument de mesure, capteur. Dans le cas d'un ordinateur pilotant un processus industriel (raffinerie, usine automobile, etc.), des câbles transportent les signaux électriques provenant de capteurs.

L'ordinateur décidera s'il doit fermer une vanne, tourner un moteur, ou activer un relais... d'après les valeurs des signaux reçus. Dans tous les cas, la décision de l'ordinateur est immédiate, grâce à un système d'exploitation orienté temps réel.

L'architecture d'un système **temps réel** lui permet une gestion très rapide des interruptions.

Avec les systèmes d'exploitation en temps réel, une interruption externe peut également être associée à des programmes écrits par l'utilisateur ; un cas typique est la fermeture d'un contact électrique à laquelle correspond l'exécution d'un programme de gestion de cet événement prévu par l'utilisateur.

Unix ne travaille pas en temps réel, ne prévoit pas de connexions avec des périphériques non classiques et n'offre pas la possibilité de gérer des interruptions. En revanche, il est idéal dans un environnement où divers utilisateurs désirent simultanément effectuer des calculs, gérer des banques de données ou travailler en traitement de texte.

Système de gestion des fichiers

Un gestionnaire de fichiers s'occupe de la gestion des mémoires de masse, donc plus particulièrement des disques par le système d'exploitation.

Tous les ordinateurs possèdent des mémoires non volatiles, car ce sont elles qui conservent les informations utiles. Ces mémoires contiennent également le système d'exploitation, qui est automatiquement chargé dans la mémoire centrale à la mise sous tension. Dans certains ordinateurs personnels, le système d'exploitation est résident sur disques durs et il en est extrait pour être chargé en mémoire selon des procédures particulières (dites de mise en route). Ces mêmes disques contiennent aussi des informations enregistrées par les utilisateurs ; c'est le système de gestion des fichiers qui a la charge de les organiser efficacement.

Les données enregistrées dans les mémoires de masse sont toujours regroupées en fichiers, et eux-mêmes subdivisés en enregistrements.

- **Classification d'après le contenu.** Les enregistrements contiennent des données en code ASCII (ou EBCDIC) ou binaire ; dans le premier cas, ce sont des programmes sources ou des textes, alors que dans le second, ce sont des résultats de compilations (fichiers tranférables), des programmes sous forme exécutable ou des fichiers de données résultats.
- **Classification d'après les modes d'accès.** Il s'agit des fichiers à accès séquentiel et des fichiers à accès direct. Dans le séquentiel, l'accès à un enregistrement passe par un « parcours » de tous les enregistrements précédents. En accès direct, l'adressage d'un enregistrement particulier se fait directement.
- **Classification d'après le type d'enregistrement.** Certains fichiers ont des enregistrements de longueur fixe, d'autres variable. La longueur fixe est caractéristique : par exemple, des programmes sources.

Système informatique composé d'un ordinateur personnel connecté à un ordinateur central par modem et ligne téléphonique.

Marka/Photri

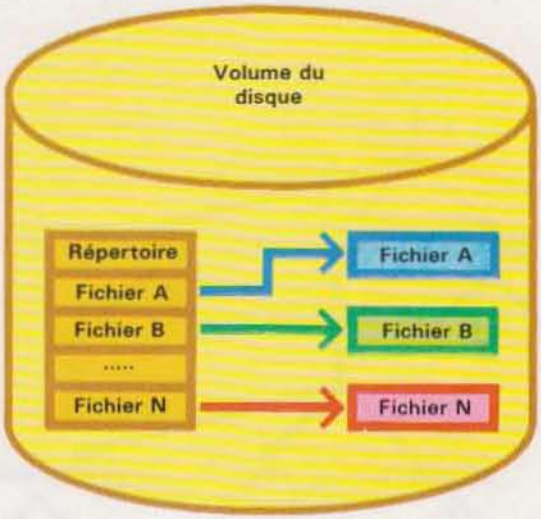
La mission du système d'exploitation est de gérer tous les types de fichiers et, dans un environnement multi-utilisateur, de garantir à X que ses fichiers ne seront pas modifiés ou effacés par Y.
Supposons qu'on se trouve en présence de 2 utilisateurs, l'un (X) écrivant un enregistrement, l'autre (Y) demandant à le lire. Les deux opérations devront être séparées de telle manière, que tant que X écrira, Y ne pourra pas lire.
L'association du concept de ressource à la quantité totale de mémoire des disques fait apparaître un autre aspect intéressant du gestionnaire de fichiers : la partition de cette mémoire entre utilisateurs.
Les mini-ordinateurs acceptent des disques d'une capacité globale de plusieurs milliers de méga-octets. Une fois définie la mémoire disponible, il faut décider de sa partition entre l'utilisateur. Une méthode consisterait à découper également l'espace mémoire. Mais elle ne satisfait pas à toutes les exigences, car certains utilisateurs auront besoin de plus d'espace que d'autres.
Les systèmes d'exploitation modernes allouent de l'espace sur disque au fur et à mesure qu'il devient nécessaire, en évitant toutefois qu'un seul utilisateur utilise tout. Aussi faudra-t-il imposer des limites précises à ne pas dépasser.
Une fois l'espace sur disque alloué, il reste à donner, à l'utilisateur, la possibilité de retrouver ses fichiers de façon rapide et transparente. C'est là qu'intervient le concept de répertoire (directory) des volumes composant le disque.
Un volume est une division logique (et non pas physique) du disque, couvrant plusieurs applications ou commun à plusieurs utilisateurs. Autrement dit, on préfère « découper » le disque en plusieurs parties pour permettre à chaque utilisateur d'être indépendant et de ne subir aucune interférence de la part des autres utilisateurs. Cette division en volumes peut également être plus élaborée.
Pour le moment, considérons un volume comme une partie du disque affecté à un utilisateur et possédant au moins un répertoire. Le répertoire est une zone du disque dans laquelle sont reportés tous les noms des fichiers enregistrés, des informations concernant la structure des fichiers et des enregistrements, et surtout l'adresse de chaque fichier sur le disque (illustration ci-contre).

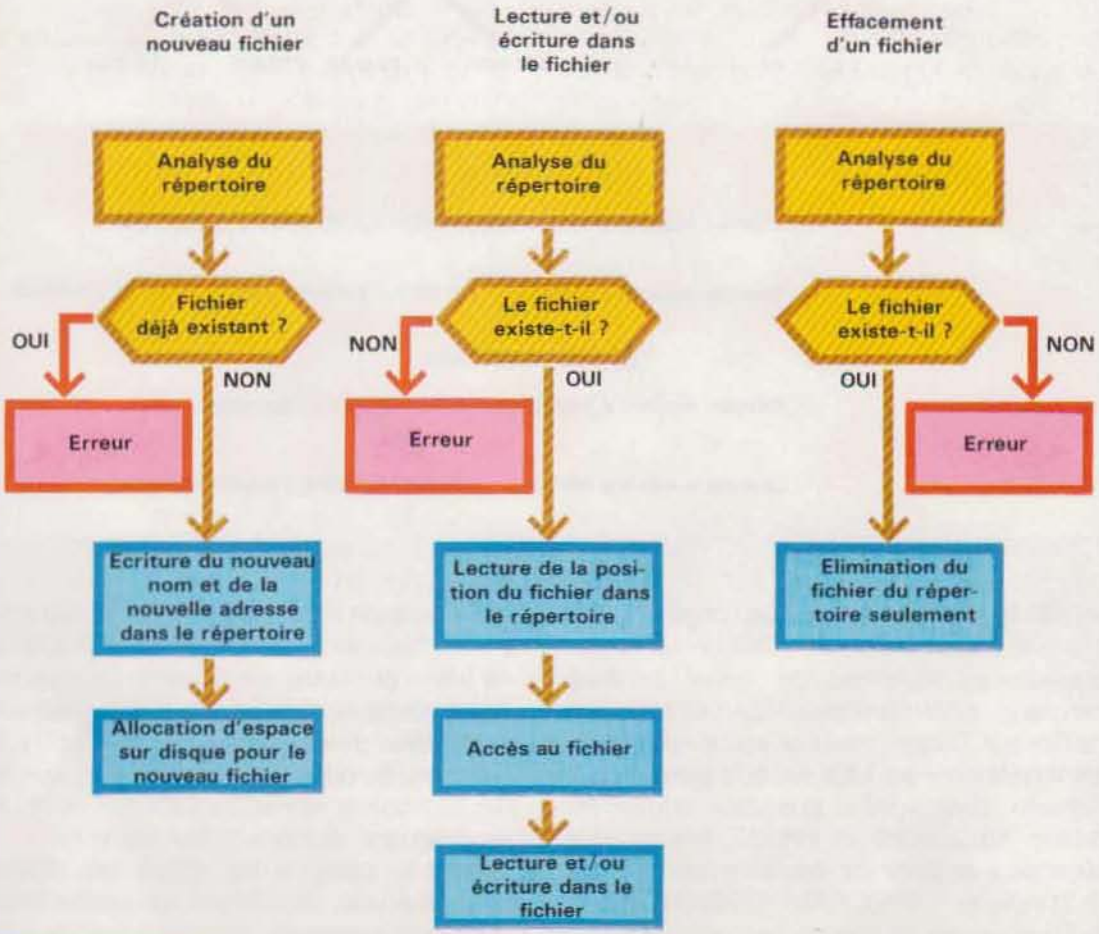
Chaque fois qu'un nouveau fichier est créé, il est enregistré dans le répertoire puis dans la zone du disque qui le contient physiquement.
Si d'autres informations sont à introduire dans le nouveau fichier, le système d'exploitation consulte le répertoire puis, une fois pointé le fichier en question, il détermine sa position sur le disque et y accède en mode direct. Même procédure pour la lecture d'un fichier.
Enfin, si l'on désire savoir quels fichiers sont enregistrés sur le disque, il suffit de lire le répertoire et de l'imprimer.

Structure du gestionnaire fichiers Unix

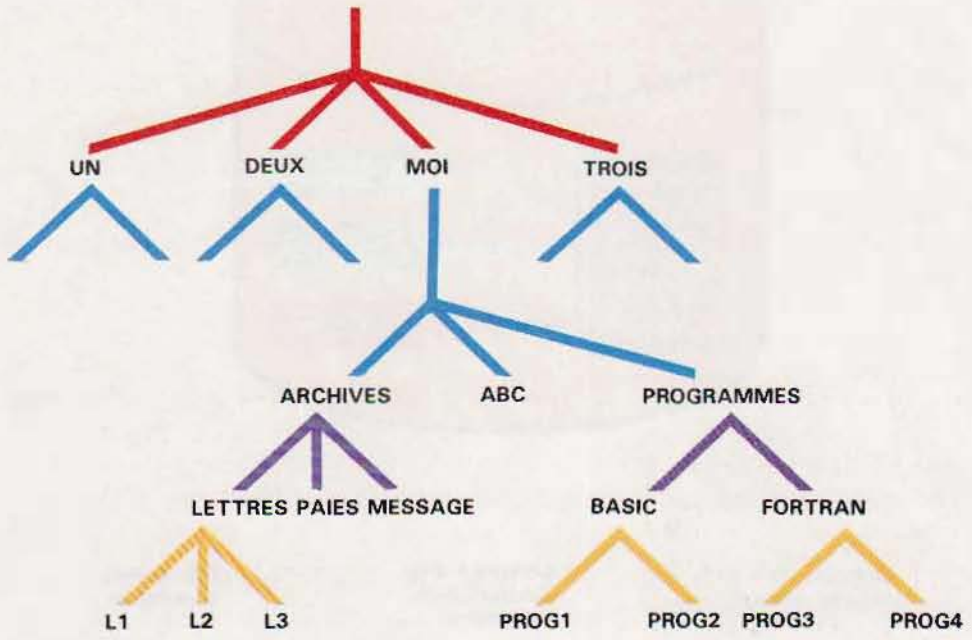
Dans le système d'exploitation Unix, le concept de répertoire est étendu : en effet, le disque contient des fichiers et des répertoires reliés entre eux. Un répertoire est considéré par Unix comme un fichier qui peut renvoyer à d'autres fichiers ou répertoires.
Le répertoire d'arrivée, le **sous-répertoire**, peut à son tour aiguiller vers d'autres sous-répertoires ou d'autres fichiers, et ainsi de suite. La structure du système de gestion de fichiers est donc arborescente, comme le montre le schéma de la page suivante.
Le symbole (barre oblique) désigne, dans Unix, le répertoire racine d'où partent toutes les branches ; de celui-ci partent, par exemple, les répertoires UN, DEUX, MOI, TROIS. Un répertoire se reconnaît au fait qu'il donne naissance à au moins une ramification (structure p. 1376). Le répertoire MOI mène à deux sous-répertoires (ARCHIVES et PROGRAMMES) et à un fichier (ABC) ; ARCHIVES conduit à un autre sous-répertoire (LETTRES) et à deux fichiers (PAIE et MESSAGE). LETTRES mène à 3 fichiers (L1, L2 et L3). Le même type de structure se retrouve à partir de PROGRAMMES.
Dans Unix, un fichier est une extrémité, ou plutôt un module, d'où rien ne part (une feuille de l'arbre), tandis qu'un répertoire génère une structure permettant de regrouper d'un point de vue logique d'autres éléments (ARCHIVES contiendra des fichiers de données, PROGRAMMES contiendra des programmes).
Pour accéder à un fichier, il faut indiquer à Unix un parcours, et un seul, à suivre depuis le répertoire-racine jusqu'au fichier final. Dans la construction de ce parcours univoque, la barre oblique sert à séparer les répertoires, à l'excep-

ACCES AU DISQUE PAR L'INTERMEDIAIRE DU REPERTOIRE





STRUCTURE DU SYSTEME DE GESTION DES FICHIERS ET VOIES D'ACCES AUX FICHIERS UNIX



En partant de /

Chemin menant à MESSAGE : / MOI / ARCHIVES / MESSAGE

Chemin menant à PROG2 : / MOI / PROGRAMMES / BASIC / PROG2

En partant de / MOI

Chemin menant à MESSAGE : ARCHIVES / MESSAGE

Chemin menant à PROG2 : PROGRAMMES / BASIC / PROG2

tion de la première qui indique l'origine, c'est-à-dire le répertoire-racine. Chaque utilisateur possède un répertoire de travail (working directory). Si le répertoire MOI est affecté à l'utilisateur Dupont, celui-ci est automatiquement positionné sur MOI dès qu'il entre dans le système. Cela signifie, que pour trouver les fichiers MESSAGES et PROG2, il n'est plus nécessaire de partir du répertoire racine. Il suffit d'indiquer : PROGRAMMES/BASIC/PROG 2. Dans ce cas, le chemin ne commence pas par / puisque on n'est pas parti du répertoire racine, mais du sous-répertoire MOI (répertoire de travail de l'utilisateur Dupont). Un répertoire de travail se modifie à l'aide d'une commande du système d'exploitation ; Dupont pourra, par exemple, se positionner directement sur BASIC et accéder aux fichiers PROG1 et PROG2 en indiquant uniquement leur nom.

Ci-contre, nous avons figuré un dialogue utilisateur/Unix dans lequel apparaissent quelques commandes en interaction avec le systè-

me de gestion de fichiers. Pour savoir dans quel répertoire il se situe, l'utilisateur introduit la commande **pwd** (print working directory). Le système répond en décrivant le chemin qui, partant du répertoire-racine, mène au répertoire de travail.

La commande **is** visualise les noms des fichiers du répertoire de travail (pour /MOI, il n'y a qu'ABC, puisque ARCHIVES et PROGRAMMES sont également des répertoires). Pour connaître tous les noms de fichiers et de répertoires auxquels mène/MOI, il suffit d'ajouter l'option **a** (all) derrière la commande **ls** (les options doivent être précédées du symbole -). La réponse du système d'exploitation contient tous les noms de fichiers auxquels mène /MOI. Avec la commande file, on peut savoir si le nom donné correspond à un répertoire ou à un fichier.

La commande **ed** (change directory) déplace le point de travail et, dans l'exemple ci-dessous, permet de passer directement au répertoire FORTRAN. Une même ligne accepte plusieurs commandes, séparées par ; (point-virgule). Ici, **ls-a** a été ajouté pour obtenir, à l'aide d'une seule commande, le changement de répertoire et la liste des fichiers et des répertoires contenus dans le nouveau répertoire.

La commande **cat** (catalog) permet d'accéder au contenu d'un fichier et, dans notre exemple, d'avoir la liste des instructions composant le programme PROG3. La commande **mkdir** (make directory) crée un nouveau répertoire, atteint en passant par FORTRAN (dans lequel on trouvera, par exemple, tous les utilitaires des programmes Fortran).

Après **mkdir**, la liste des noms auxquels mène FORTRAN contient désormais **SUBR** (sous-programmes), et la commande **file** indique qu'il s'agit d'un répertoire.

QUELQUES COMMANDES DU SYSTEME DE GESTION DES FICHIERS UNIX

Messages-écran

Données entrées par l'utilisateur

```
$ pwd
/MOI
$ ls
ABC
$ ls -a
SAUVEGARDES ABC PROGRAMMES
$ file ARCHIVES ABC PROGRAMMES
ARCHIVES:   directory
ABC:        ascii text
PROGRAMMES: directory
$ cd PROGRAMMES/FORTRAN; ls-a
PROG3 PROG4
$ cat PROG3
        (liste du programme PROG3)
$ mkdir SUBR
$ ls-a
PROG3 PROG4 SUBR
$ file SUBR
SUBR :      directory
$
```

Marka/Photri

Système de traitement de texte avec imprimante intégrée.

L'utilisateur peut protéger ses fichiers contre les effacements ou les modifications par d'autres utilisateurs ; pour cela, il dispose de commandes spéciales offrant des protections de deux types : en écriture ou en lecture. Si le fichier est protégé en écriture, seul son créateur peut l'écraser ou le modifier. Un autre utilisateur, positionné dans le répertoire en question à l'aide de la commande **cd**, pourra seulement lire son contenu.
Si le fichier est protégé en lecture, il ne pourra être lu par d'autres utilisateurs que s'ils possèdent la clé de lecture.
Unix accepte deux catégories de « lecteurs » : les **utilisateurs ordinaires** (users) et les **super-utilisateurs** (superusers). Les premiers se déplacent à l'intérieur des répertoires qui leur sont associés et, s'ils sautent à d'autres répertoires, ils ne sont en mesure de réaliser que des opérations autorisées par leurs créateurs (par exemple lire un fichier s'il est protégé en écriture) ; les seconds peuvent tout faire, en ce sens qu'ils ont la possibilité de modifier les mécanismes de protection des fichiers et des répertoires. Le super-utilisateur est généralement une personne qui reconfigure le système d'exploitation pour obtenir un bon environnement de travail multi-utilisateur, avec un minimum de coordination.

Processus simultanés

La question de la coordination des opérations lancées simultanément par plusieurs programmes se pose, en effet, dans tout système d'exploitation multi-utilisateur comme Unix.
En multiprogrammation et en temps partagé, il existe, nous l'avons vu, des méthodes pour déterminer quel programme doit disposer de l'UC et quels autres doivent attendre.
Il nous reste à expliquer les techniques qui vont permettre à ces programmes soit d'échanger des informations entre eux, soit d'interrompre leur déroulement quand une ressource n'est pas disponible (car déjà employée par un autre programme).
C'est là qu'intervient le concept de **processus,** vu par le système d'exploitation comme une séquence d'opérations à exécuter en une seule fois. Il peut s'agir d'une procédure, d'un sous-programme, ou même d'une instruction unique.

Les concepteurs de systèmes d'exploitation ont tendance à décomposer les opérations en opérations plus simples jusqu'à atteindre un niveau plancher, lequel correspond à une instruction-machine. De cette manière, on obtient un système d'exploitation composé de N modules reliés entre eux, chacun ayant la charge de piloter les opérations s'y rapportant. Le lien entre le processus et les opérations à effectuer dépend donc du système d'exploitation.

En multiprogrammation, un programme peut connaître un début d'exécution, il est ensuite suspendu par un autre programme de priorité plus élevée, puis repris, etc. Cet exemple fait alors apparaître deux états principaux d'un processus : il est **bloqué** ou **actif.** Quand un processus est bloqué, il attend qu'une ressource devienne disponible pour poursuivre ; quand il est actif, il utilise l'unité centrale.

Il existe plusieurs états de blocage dans les systèmes d'exploitation : un processus peut attendre qu'une opération d'entrée/sortie soit terminée, ou que la ressource unité centrale soit libérée par un autre processus. L'état actif, lui, n'a pas de subdivision.

C'est pourquoi la principale tâche d'un système d'exploitation consiste à coordonner les attentes des processus en définissant les transitions possibles entre un type de blocage et un autre, ou entre un état de blocage et un état actif. Les changements d'état sont pilotés par le système d'exploitation en fonction des types des processus mis en œuvre.

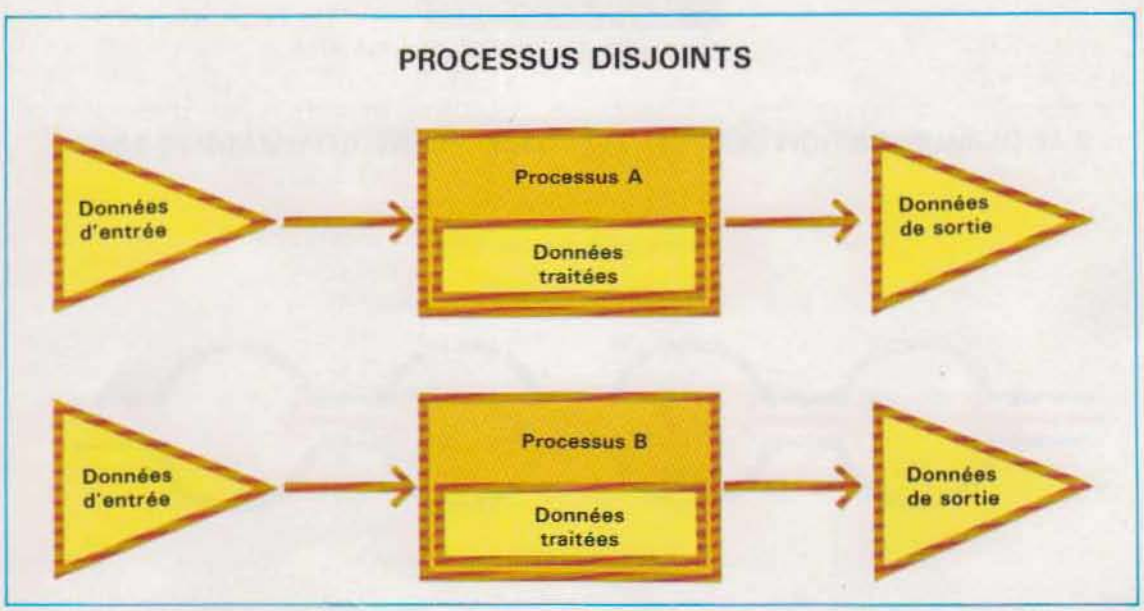
Les processus **simultanés** sont de deux ordres : **disjoints** et en **interaction.** Deux processus sont dits disjoints quand ils n'agissent pas sur la même base de données ; en interaction quand ils possèdent des variables communes.

Deux processus disjoints ont un comportement indépendant dans le temps : les résultats produits par le premier ne doivent pas être utilisés par le second ; le système d'exploitation ne fournit de service qu'à l'un ou à l'autre (schéma ci-dessous).

Deux processus en interaction fonctionnent différemment : les modalités d'accès aux données communes doivent être préalablement établies. Comme le montre la figure 1 (page suivante), ces données peuvent également être traitées simultanément au déroulement des processus, nécessitant, de ce fait, des mécanismes de synchronisation des processus.

Communication entre les processus

La forme la plus simple de communication entre processus est de type « historique », qui permet aux données de sortie d'un processus d'être utilisées en entrée par le processus suivant. Les commandes Unix autorisent ce mode de synchronisation, appelé **pipe-line,** en définissant les données de sortie d'une commande comme étant les données d'entrée d'une autre commande. Les deux commandes sont alors

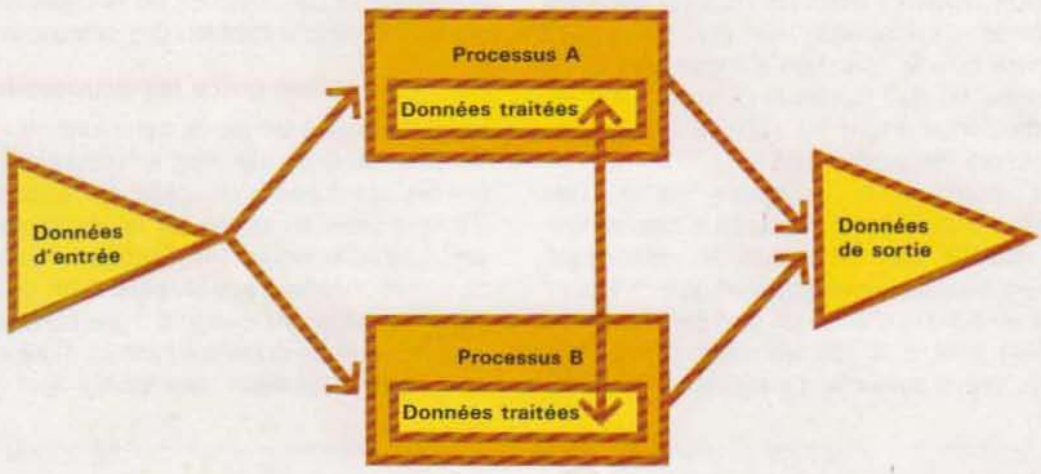


introduites de façon séquentielle, séparées par une barre verticale signifiant que les données doivent être enchaînées ou échangées en pipeline.
Ce concept reprend l'image d'un oléoduc auquel sont reliés des producteurs et des consommateurs de pétrole. Ici, le pétrole représente les données, alors que les producteurs et les consommateurs sont les commandes Unix.
Dans l'exemple illustré par le schéma 2, la 1re commande (ls-a) génère la liste des fichiers et des répertoires contenus dans le répertoire de travail, la 2e (grep rime) extrait toutes les lignes contenant le mot rime, la 3e (sort) trie ces lignes et les classe par ordre alphabétique, tandis que la 4e lance leur impression.
Dans le cas de la communication historique, le comportement dans le temps du processus est défini à l'avance. Quand cette programmation n'est pas réalisable, il faut alors résoudre deux problèmes :

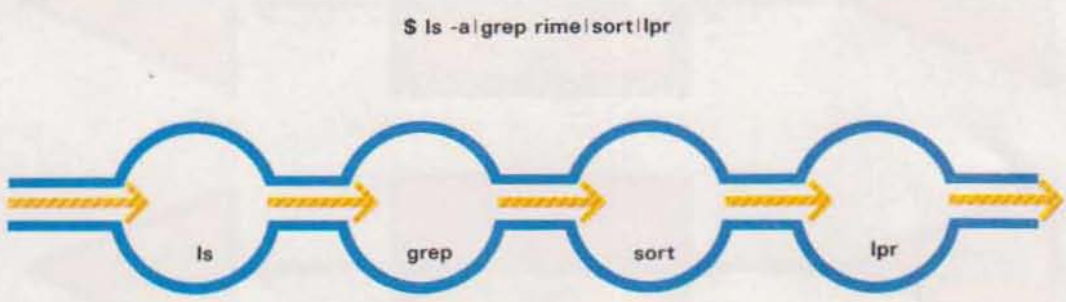
- l'accès aux données partagées
- l'échange des données entre processus

L'accès aux données partagées par un processus implique la possibilité de modification de ces données par le même processus. Dans ce cas, l'accès est interdit à tout autre processus

1 / PROCESSUS EN INTERACTION

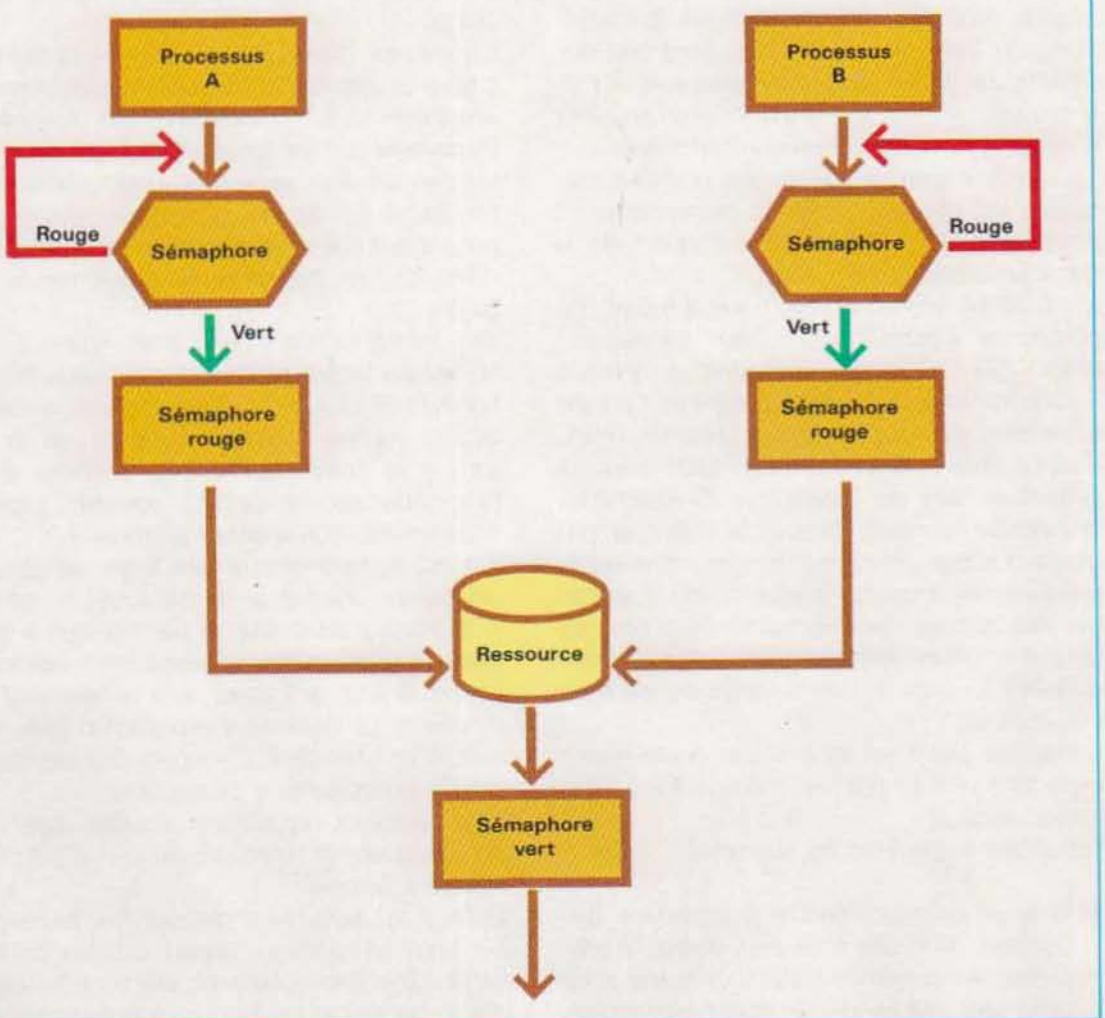


2 / COMMUNICATION DU TYPE PIPE-LINE ENTRE COMMANDES UNIX



avant la fin des opérations sur les données. La synchronisation est mise en œuvre par le jeu de « sémaphores » fournis par le système. Le processus désirant accéder aux données demande au système si le feu est vert, c'est-à-dire si les opérations précédentes sont terminées (voir schéma ci-dessous). Si le feu est rouge ce processus est provisoirement « endormi ».
L'emploi de sémaphores s'applique également aux autres ressources. Ils éviteront, par exemple, le mélange de sorties sur imprimante de processus simultanés. Ainsi, les processus désirant utiliser la ressource imprimante devront tout d'abord en demander l'accès en interrogeant les sémaphores. Il peut toutefois se produire des cas d'« étreinte fatale » (blocage complet du système), lorsque deux processus, ou plus, attendent indéfiniment un événement. Cette situation sera illustrée par l'exemple suivant : un processus A accède à un fichier sur disque ; pour le protéger, le système fait passer au rouge le feu relié à ce disque. Le même processus va ensuite demander l'intermédiaire d'accès au dérouleur de bande magnétique sur lequel il voudrait passer. Mais pendant que le processus A lisait le fichier sur disque, un processus B utilisait la bande magnétique et essayait de faire passer au rouge le feu relié au fichier sur disque pour y accéder. Le processus A attend que B débloque la bande, tandis que B attend que A libère le fichier sur disque. C'est cela l'étreinte fatale.

ACCES A UNE RESSOURCE (FICHIER-DISQUE) AVEC RECOURS AUX SEMAPHORES



Pour l'éviter, des régions dites critiques (à l'intérieur desquelles sont gérés les processus simultanés) ont été définies dans les systèmes d'exploitation. La gestion des processus dans la région critique est effectuée sur la base de trois postulats, formulés en 1965 par Dijkstra :

1/ Quand un processus désire entrer dans une région critique, sa demande n'est maintenue que pendant un temps fini.
2/ Un seul processus à la fois peut occuper une région critique.
3/ Un processus se trouvant dans une région critique ne peut y rester que pendant un temps fini.

Second volet de la gestion des processus simultanés : **l'échange des données.** Qu'il s'agisse de données d'entrée, de sortie ou de données traitées, l'échange se fera toujours en utilisant les zones communes aux divers processus, dans lesquelles les données sont introduites et prélevées. Ces zones sont soit des fichiers sur disque, soit des espaces de la mémoire centrale mis à la disposition du système d'exploitation à l'intention des processus.
Le synchronisme de l'accès aux données partagées est résolu à l'aide de sémaphores ; il nous reste donc à considérer l'aspect de la communication.
Etant donné une zone commune, il existe des processus « producteurs », qui introduisent alors des données, et des processus « consommateurs », qui les prélèvent. La zone commune est toujours de dimensions finies, d'où certains conflits de vitesse entre celui qui produit et celui qui consomme. Si le producteur est le plus rapide, la zone se sature, empêchant d'autres données d'entrée ; si c'est le consommateur qui est le plus rapide, il se forme des queues dans lesquelles N processus consommateurs attendent de recevoir des données. L'usage de l'unité centrale n'est alors pas optimal.
Ce dernier point est sans aucun doute moins important que le premier ; de ce fait, il est parfois négligé.
Les règles fixées sont les suivantes :

1/ Si le producteur cherche à introduire des données dans une zone déjà pleine, le processus est suspendu jusqu'à ce que la zone commune soit libérée de quelques données par un consommateur.
2/ Si le consommateur cherche à prélever des données dans une zone commune vide, le processus est suspendu jusqu'à ce qu'un producteur y introduise des données.

Les données se trouvant dans la zone commune sont gérées selon le système FIFO (Premier entré, premier sorti) ou LIFO (Dernier entré, premier sorti). Dans le premier cas, les données entrées en premier sont également prélevées en premier, alors que dans le second, ce sont les dernières données entrées qui sont prélevées en premier. Le système utilisé dépend de l'application.
Quand les processus désirant communiquer sont plus de deux, la zone commune doit pouvoir distinguer, par exemple, les données introduites par le processus A mais destinées aux processus B et C, à l'exclusion d'autres processus.
Le système d'exploitation utilise alors des mécanismes internes lui permettant d'identifier les émetteurs et les destinataires des données et de garantir que les processus ne recevront que les données les concernant. Pour cela, la zone commune est divisée en plusieurs zones tampons ayant chacune un code qui lui permet de n'être lue que par les processus le reconnaissant.
Ces zones tampons sont assimilables à des boîtes aux lettres fermées à clé ; seuls les détenteurs de clés peuvent prendre connaissance de leur courrier. Ce n'est d'ailleurs pas un hasard si la communication de données entre processus simultanés est souvent appelée « communication à boîtes à lettres ».
Elle est également associée à des utilisateurs simultanés : l'échange de messages donne lieu à un service de « courrier électronique » dont peuvent se servir tous ceux qui ont un terminal connecté à un ordinateur ou à un réseau d'ordinateurs. Le système d'exploitation Unix permet, à un utilisateur, d'envoyer des messages à tous les utilisateurs du système.
Les messages reçus sont archivés dans des fichiers privés et regroupés au sein d'une base logique (contenu).
Enfin, il est possible d'envoyer des messages sur base temporelle : départ à telles date et heure. Si le destinataire est identique à l'expéditeur, on a alors créé un agenda électronique.

L'informatique graphique en Basic

L'informatique graphique est l'application la plus spectaculaire de l'informatique. Née seulement sur de gros systèmes, elle s'est depuis largement développée notamment sur micro-ordinateurs. Cette diffusion a été rendue possible grâce aux progrès réalisés au niveau du matériel. Les premiers micros étaient, pour des raisons économiques, des machines limitées à des applications simples. La majorité d'entre eux n'avait pas de réelles capacités graphiques, ni la haute résolution.

L'évolution du matériel et la baisse des coûts ont ensuite permis de fabriquer des systèmes rentrant dans la catégorie « ordinateur personnel » ayant des capacités graphiques étendues. Les producteurs de logiciels spécialisés se sont alors tournés vers cette catégorie de matériels et ont écrit des programmes graphiques de plus en plus sophistiqués.

Un ordinateur destiné à des emplois graphiques est très différent des ordinateurs classiques.

La première particularité concerne la diversité des périphériques qu'il doit gérer.

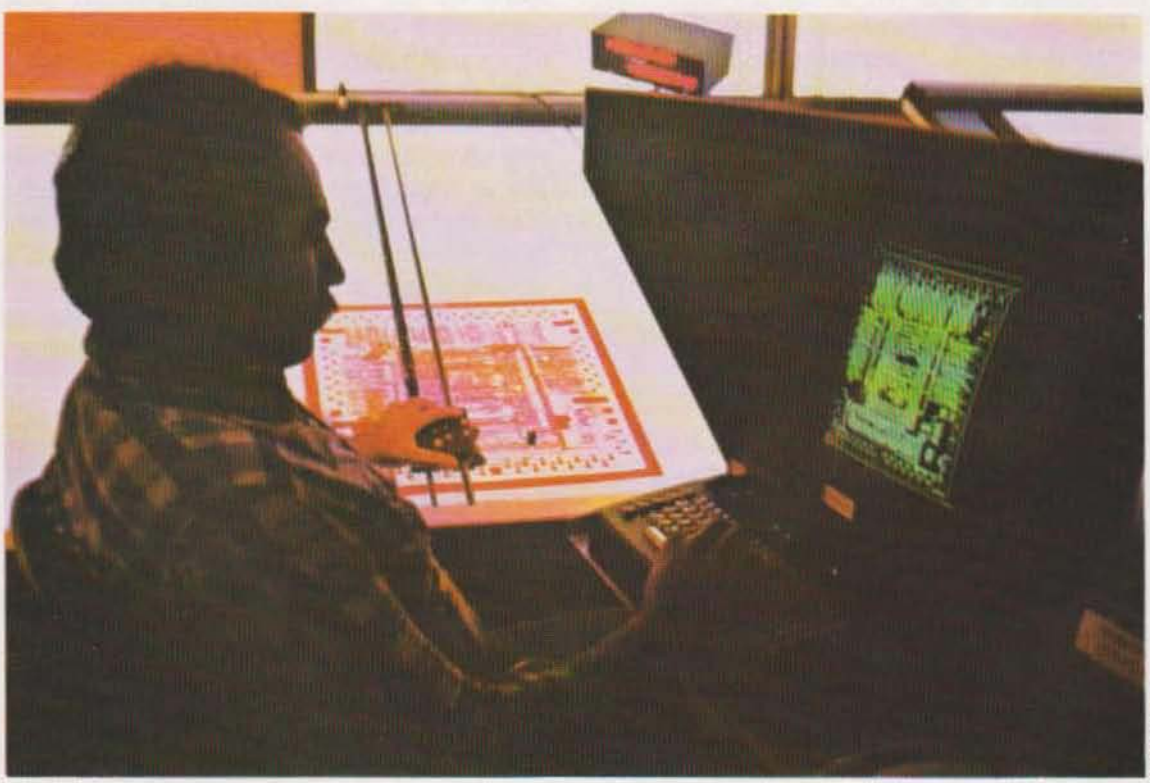
Avec une machine classique de la catégorie micro, les périphériques appartiennent à 4 classes : écran, clavier, disque (ou bande magnétique) et imprimante. Pour pouvoir être utilisée à des fins graphiques, la même machine doit être complétée de nombreux autres périphériques du type traceur de courbes, tablette graphique, et surtout des écrans graphiques.

S'ils utilisent des ports d'E/S et des protocoles de communication semblables aux autres, ces nouveaux équipements demandent néanmoins des logiciels d'application spécialisés.

La deuxième différence principale se situe au niveau de la structure matérielle interne.

L'informatique graphique (ou infographie) de-

Système informatique pour la Conception assistée par ordinateur (CAO).



Marka/Jarner

mande de grandes capacités de mémoire, ce qui oriente l'utilisateur vers des processeurs 16 ou 32 bits ou, à défaut, vers l'emploi de cartes d'extension particulières. Sauf quelques rares exceptions, la première solution oblige à adopter des machines de la catégorie « mini », alors que la seconde est la plus courante avec les « micro » (ces machines ont, pour la plupart, des microprocesseurs 8 bits, ce qui les rend inadaptées à la gestion de mémoires supérieures à 64 K). Toutefois, l'emploi de dispositifs d'extension tend à disparaître avec l'apparition des micros 16 bits.

Applications de l'informatique graphique

En pratique, il n'existe pas d'application dans laquelle le graphique ne puisse pas constituer une aide rapidement indispensable, ou tout au moins un complément utile.
Pour être interprété et assimilé, le résultat d'un traitement présenté sous forme numérique demande un grand effort de concentration et d'analyse. Ce même résultat, représenté graphiquement, sera d'une compréhension plus immédiate.
Ces considérations ont abouti à l'emploi de l'informatique graphique dans des domaines moins techniques, comme les études démographiques, l'analyse statistique et les applications économiques en général.
L'information graphique connaît actuellement 4 types d'application :

- le graphique de décision
- la conception assistée par ordinateur (CAO)
- le traitement des images
- l'animation

Le graphique de décision

Cette expression désigne les applications orientées vers la résolution de problèmes à caractère économique ou, plus généralement, relatifs à la prise d'une décision au niveau de l'entreprise (aide à la décision).
L'automatisation du travail de bureau rend souvent nécessaire la représentation synthétique de certains résultats importants. L'emploi de graphiques, pour mettre en évidence l'évolution de certains phénomènes économiques, a pris une telle importance qu'il a engendré un véritable secteur de l'informatique graphique : le graphique de décision. Du point de vue de la programmation, ce sont là des applications simples, qui ne demandent généralement pas de traitements particuliers ni d'algorithmes complexes.

La conception assistée par ordinateur (CAO)

La CAO a été l'un des premiers domaines d'application de l'informatique graphique. Les potentialités de l'ordinateur peuvent, en effet, y être pleinement exploitées.
De grandes quantités de données relatives au travail de conception sont mémorisées et rappelées par un ou plusieurs menus. Il est ainsi possible de créer des banques de données contenant des modules ou les détails déjà prêts d'un dessin. Pour obtenir un dessin fini, le concepteur n'a qu'à relier les différents modules, tandis que l'ordinateur affine le dessin à l'aide des éléments de détail.
Dans les cas d'applications plus complexes, l'ordinateur permet d'obtenir différentes vues de l'objet, afin d'en tirer des plans d'exécution ; de plus, il effectue également les calculs nécessaires à la détermination des dimensions ou des caractéristiques les plus adaptées à une application particulière. Nous allons maintenant examiner quatre domaines d'emploi.
La conception mécanique. C'est l'aspect le plus connu de la CAO. Elle est rendue possible par un logiciel produisant des plans mécaniques par composition d'éléments géométriques élémentaires ou utilisant des symboles prédéfinis (éléments de figure).
Les éléments géométriques entrant dans la réalisation d'une figure sont précisémment :

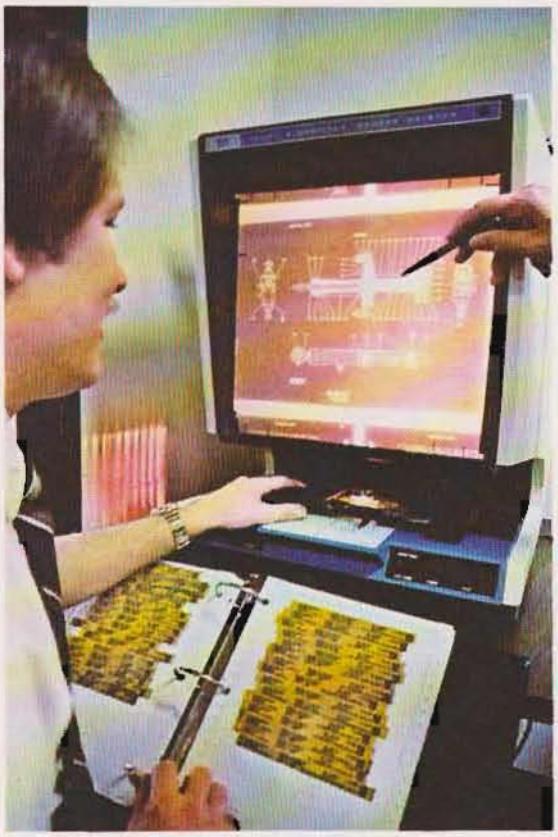
— les points
— les segments de droite
— les cercles
— les lignes courbes
— les tangentes
— les arcs et les raccords

En employant un jeu de commandes de sélection et de positionnement des différents éléments, on aboutit alors à des dessins très complexes.

Les systèmes de CAO prévoient généralement

la possibilité de produire automatiquement des parties symétriques, ou d'agrandir un détail. Les figures composées sont mémorisées sur disque et reproduites sur papier. Le gain de temps qui en découle par rapport à la méthode de dessin manuel classique est considérable.
Autre avantage : la facilité avec laquelle le concepteur apporte des modifications à un dessin ; un élément particulier peut être rappelé et modifié ou redessiné en quelques minutes, alors que des journées entières de travail manuel étaient nécessaires auparavant.
Deuxième avantage : la production automatique d'instructions d'asservissement des machines-outils à commande numérique.
Dans la production en série, on cherche à réduire l'intervention humaine, notamment au cours des phases de fabrication des pièces mécaniques. Il existe des machines entièrement automatiques (tours, fraiseuses, fileteuses...), dites « à commande numérique », et pilotées par ordinateur.
Les instructions nécessaires à la commande des machines sont automatiquement générées lors de la conception de l'élément mécanique par le système de CAO : c'est alors de la fabrication assistée par ordinateur (FAO). Le processus global de conception d'une pièce et de production des codes de commande s'appelle CFAO (Conception et fabrication assistées par ordinateur).

La conception électronique. Cette application de l'informatique graphique est très proche de la précédente. Introduite plus récemment, elle doit connaître un très grand développement dans un proche avenir. Ses champs d'application vont de la conception des circuits intégrés à celle de cartes électroniques tout entières, avec production automatique des schémas de fabrication et liste des composants nécessaires.
Là aussi, les avantages dépassent le simple gain de temps dans l'exécution de la partie graphique. Au fur et à mesure que le projet progresse, l'ordinateur vérifie les stocks de composants, contrôle les choix du concepteur du point de vue de la compatibilité et de l'interconnexion des composants employés et, même, effectue une sélection entre différents composants concurrents, en fonction des caractéristiques physiques requises.


Marka/L. de Wys

Exemple d'application graphique.

La conception architecturale. L'informatique graphique s'est avérée extrêmement utile en architecture, depuis l'étude de l'habitabilité des locaux, de leur aménagement, jusqu'à l'analyse structurelle et, dans les programmes les plus complexes, au dessin de vues en perspective.
Cette dernière application, qui exige des logiciels et des matériels de pointe, connaît actuellement un vif succès tant dans les petites que dans les grandes entreprises.
La plupart des difficultés concernant les algorithmes d'effacement des lignes non visibles ont été résolues. La construction d'une vue en perspective ou d'une axonométrie demande, en effet, la mise au point d'une logique complexe pour déterminer si une ligne donnée est visible ou cachée par d'autres éléments de la figure. Les applications informatiques utilisent presque exclusivement la technique axonométrique, plus simple à gérer. Ce n'est que dans des cas particuliers que le système prévoit l'effacement automatique des lignes cachées.

Dans ce secteur d'application, une autre technique est très importante : l'ombrage. Dans un dessin fait à la main, c'est l'artiste qui, d'après sa sensibilité, décide où et comment il doit ajouter des ombres pour que le résultat donne l'impression d'être tridimensionnel.
Avec l'ordinateur, le processus est beaucoup plus complexe ; en effet, il faut décrire à la machine tous les algorithmes appliqués ainsi que les décisions qui en découlent. Ces algorithmes demandent des machines très puissantes, même plutôt simples.

Le design industriel. Les designers n'auront pas été les derniers à recourir à la technique graphique pour concevoir leurs produits.
Ce domaine d'application, très vaste, couvre tous les secteurs dans lesquels il faut associer, à la conception technique d'un produit, une étude esthétique approfondie du résultat à obtenir. L'ordinateur simplifie considérablement le travail de mise en forme d'une idée artistique, qui peut aussi bien concerner la création d'un objet que la reproduction d'un motif sur tissu. Les stylistes et les créateurs de mode sont donc d'autres utilisateurs attitrés.

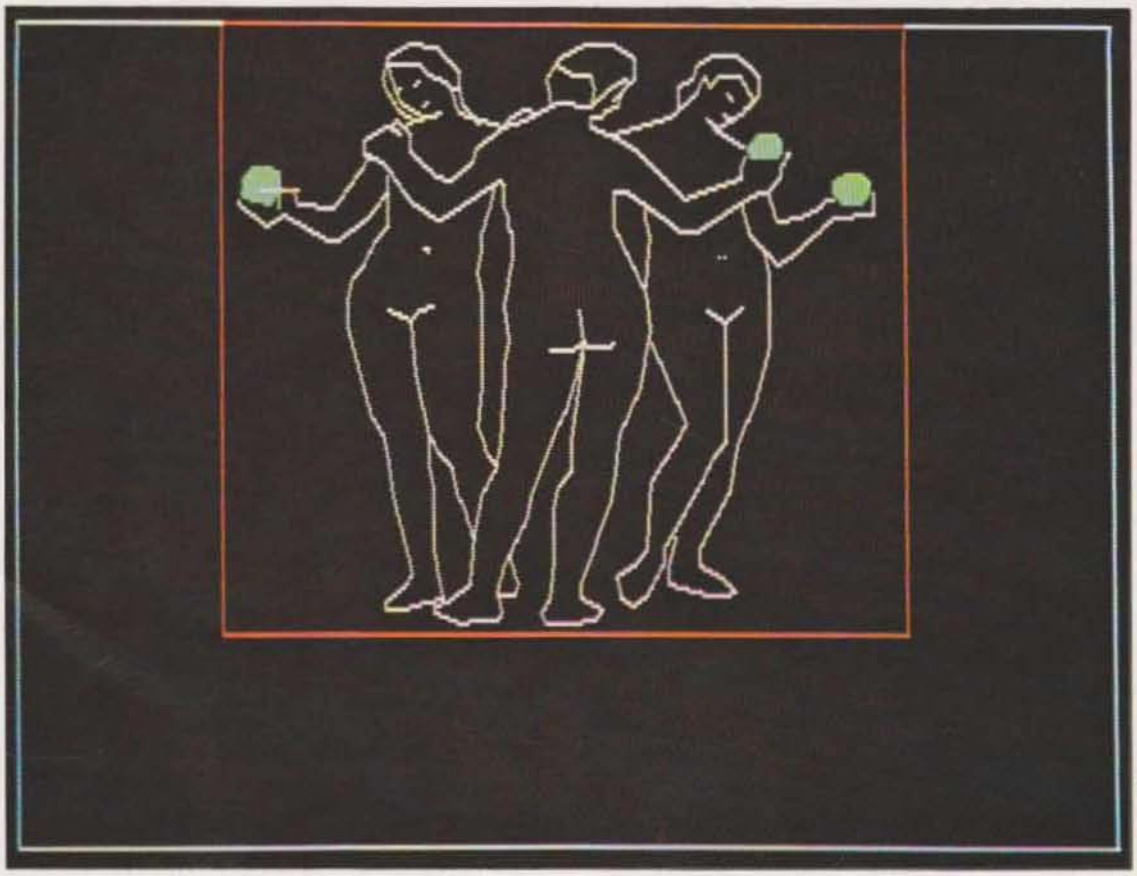
Le traitement des images

Dans l'informatique graphique, le traitement des images occupe une place particulière, parce qu'il demande l'emploi de techniques logicielles et d'équipements spécialisés, ayant trait à l'acquisition et au traitement des images, ainsi qu'à leur reconnaissance.
Une image se présente sous différentes formes (photographie, dessin) et a toujours un caractère « continu ». Pour être entrée dans l'ordinateur, elle doit être « numérisée », c'est-à-dire traduite point par point en code binaire. Cette conversion est effectuée à l'aide de dispositifs de précision allant de la simple tablette graphique à des systèmes de lecture ou à des caméras complexes.
Le logiciel nécessaire au fonctionnement de ces appareils pose également de gros problèmes, dûs en grande partie à la logique binaire de l'ordinateur. Celle-ci le met en situation de n'identifier que deux états (1 et 0) tandis que l'acquisition d'une image requiert la capacité de reconnaître la tonalité d'une couleur ou les tons de gris dans le cas d'une image monochromatique. Dans la réalité, les nuances de couleurs sont infinies ; or l'ordinateur, aussi puissant soit-il, ne peut prendre en compte ou n'en simuler qu'un nombre limité. C'est donc aux programmes d'application d'optimiser les ressources afin d'obtenir une représentation aussi proche que possible de l'original.
La technique utilisée consiste, grosso modo, à reproduire les nuances en épaississant plus ou moins les points constituant la figure. Si les points sont très proches, on a un ombrage très

Exemples d'images graphiques obtenues par ordinateur : le visage d'Einstein (à gauche), le pont de Brooklyn (à droite).

Marka/L. de Wys

Marka/R. Vila



Centre THC/Marka

Application à l'art du traitement des images sur écran polychrome.

marqué, alors que l'on obtient des effets d'estompage en les écartant. Les systèmes les plus évolués emploient des moniteurs couleur qui, grâce à des combinaisons spéciales, fournissent des représentations très précises.

Conséquence directe du développement des techniques d'acquisition et de traitement des images, les méthodes de reconnaissance des images avancent à grands pas, même si elles ne sont pas facilement accessibles pour le moment. Ce champ d'application, appelé « reconnaissance des formes » (pattern recognition) constitue aujourd'hui une discipline à part entière. Ses implications dans d'autres secteurs sont très importantes, comme la réalisation de têtes de « lecture » ou de classification d'objets par l'ordinateur à partir de leur image.

L'animation

Il est possible d'exploiter la très grande rapidité de réponse des ordinateurs pour préparer des programmes capables d'animer des dessins avec une résolution telle que l'on obtient une impression de mouvement continu.

Les champs d'application de l'animation sont divers, allant de la préparation des jeux vidéo à l'analyse cinématique d'un phénomène. Certains programmes simulent les déplacements du corps d'un conducteur lors d'un accident automobile et permettent de décider du meilleur agencement interne d'une voiture.

Les terminaux graphiques

Les dispositifs d'E/S orientés graphiques constituent une vaste gamme de produits. Nous allons examiner, de façon plus approfondie, les périphériques connectables aux micro-ordinateurs et aux ordinateurs personnels ; nous nous contenterons d'évoquer rapidement ceux utilisés dans les grands systèmes informatiques.

Dans la catégorie micros et ordinateurs personnels, les programmes d'application graphiques se répondent peu à peu avec des prix qui les mettent à la portée du plus grand nombre.

Dispositifs d'entrée

Les données d'entrée à fournir à un logiciel graphique se composent essentiellement d'une série de coordonnées représentant le dessin ou la figure à mettre en mémoire. Les dispositifs d'entrée peuvent donc être vus comme des transducteurs de position.

La figure ci-contre reproduit le schéma de principe d'un dispositif général d'acquisition d'entrées graphiques. Le graphique à saisir est dessiné sur le plan de travail, par exemple sur une feuille de papier millimétrée. Chacun des points du graphique est désigné par rapport à un système de coordonnées (X, Y) correspondant aux bords de la feuille.

Entrer les graphiques en machine, c'est simplement lui transmettre les coordonnées de chacun des points du dessin.

L'opérateur déplace, sur le dessin, un transducteur de position qui produit deux signaux : le premier proportionnel au déplacement le long de l'axe X et l'autre proportionnel au déplacement le long de l'axe Y. Au fur et à mesure que le transducteur est déplacé sur le contour de la figure, l'ordinateur enregistre les coordonnées relatives à chaque point, qui ne peut prendre que deux valeurs, ON=visible, ou OFF=non visible.

Si le transducteur est positionné, par exemple, au niveau du point P, l'ordinateur collecte ses coordonnées (X, Y) ; si, de plus, l'état ON du point P est activé, la machine mémorise ce point et, éventuellement, l'affiche à l'écran.

Le principe sur lequel sont fondés les transducteurs de position est généralement de type analogique ; le terminal doit donc être assorti d'un convertisseur analogique/numérique. Il faut également prévoir un logiciel capable de ramener le signal à des valeurs compatibles avec les dimensions de l'écran.

La figure de la page 1390 représente le schéma de principe d'un transducteur potentiométrique qui mesure indirectement une distance par l'intermédiaire d'une mesure de résistance. Sa structure physique ressemble à celle d'un appareil à dessiner universel.

Deux curseurs métalliques se déplacent le long de deux résistances perpendiculaires l'une par rapport à l'autre. Chacun des points du plan est repéré d'une seule façon : en positionnant correctement les deux curseurs le long des deux résistances (guides). Une fois le point P atteint, les valeurs de résistance* Ry et Rx, qui se mesurent entre les points P,B et B,A, définissent de façon univoque les coordonnées du point.

Les valeurs de résistance sont converties en valeurs numériques et envoyées à l'ordinateur. Ce processus est résumé par les 3 phases suivantes :

1/ lecture de la position
2/ transformation de la donnée en valeur numérique
3/ conversion à l'aide de facteurs d'échelle

Chacune d'elles introduit des inexactitudes ou des erreurs dont l'addition détermine la précision et la répétabilité du système. Les causes d'imprécision dépendent essentiellement de la construction mécanique des transducteurs et aussi de la qualité du CAN (convertisseur analogique/numérique).

Pour la précision mécanique, il n'est pas possible de faire des estimations a priori ; la seule pratique reste l'essai de l'appareil. A l'inverse, la précision du CAN peut généralement être évaluée à l'avance.

L'élément déterminant (et non l'unique) est le nombre de bits sur lequel travaille le convertisseur. Un convertisseur à 4 bits, par exemple, fournit en sortie un nombre binaire à 4 bits, c'est-à-dire une valeur comprise entre 0 et 15 (en décimal).

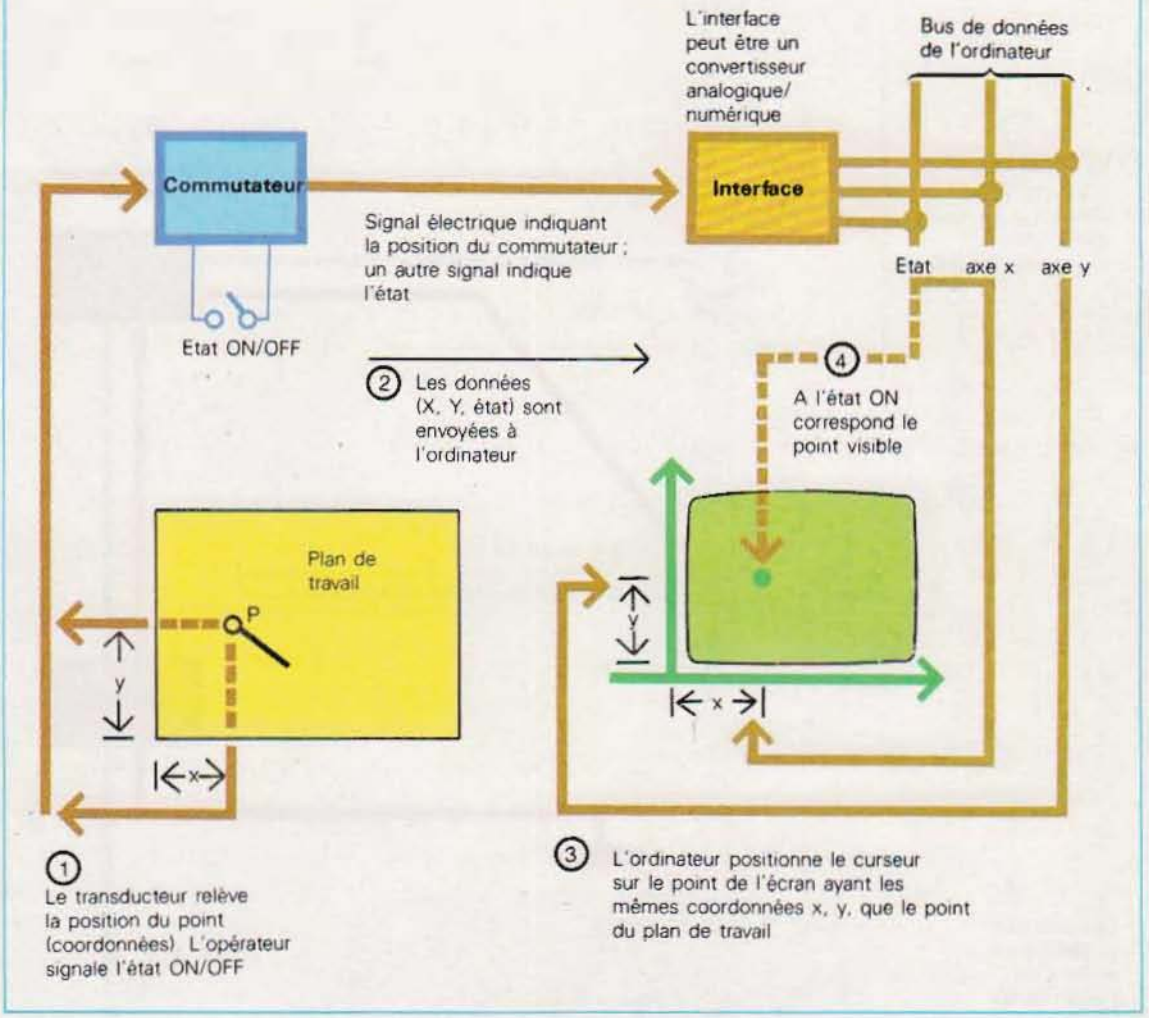
La plus petite quantité fournie est 1, ce qui signifie que sa précision, exprimée sous forme d'un ordre de grandeur, est de 1/15, soit à peu près 6 %. Il convient d'ailleurs de noter que la précision s'améliore très nettement au fur et à

*La valeur d'une résistance se mesure en ohms (symbole Ω). Cette valeur dépend de la longueur du fil constituant la résistance (toutes autres caractéristiques étant égales). Cette dépendance s'exprime à l'aide d'une fonction de proportionnalité. Si une résistance de 100 Ω est longue de 2 cm, la valeur mesurée en son milieu est de 50 Ω, alors qu'à un quart de sa longueur, elle est de 25, etc. D'une façon générale :

Déplacement = K R

K étant un facteur de tarage et R la valeur de résistance mesurée (dans l'exemple K = 2 cm/100 Ω = 1/50 [cm Ω]).

SCHEMA DE PRINCIPE D'UN PERIPHERIQUE D'ENTREE POUR APPLICATIONS GRAPHIQUES

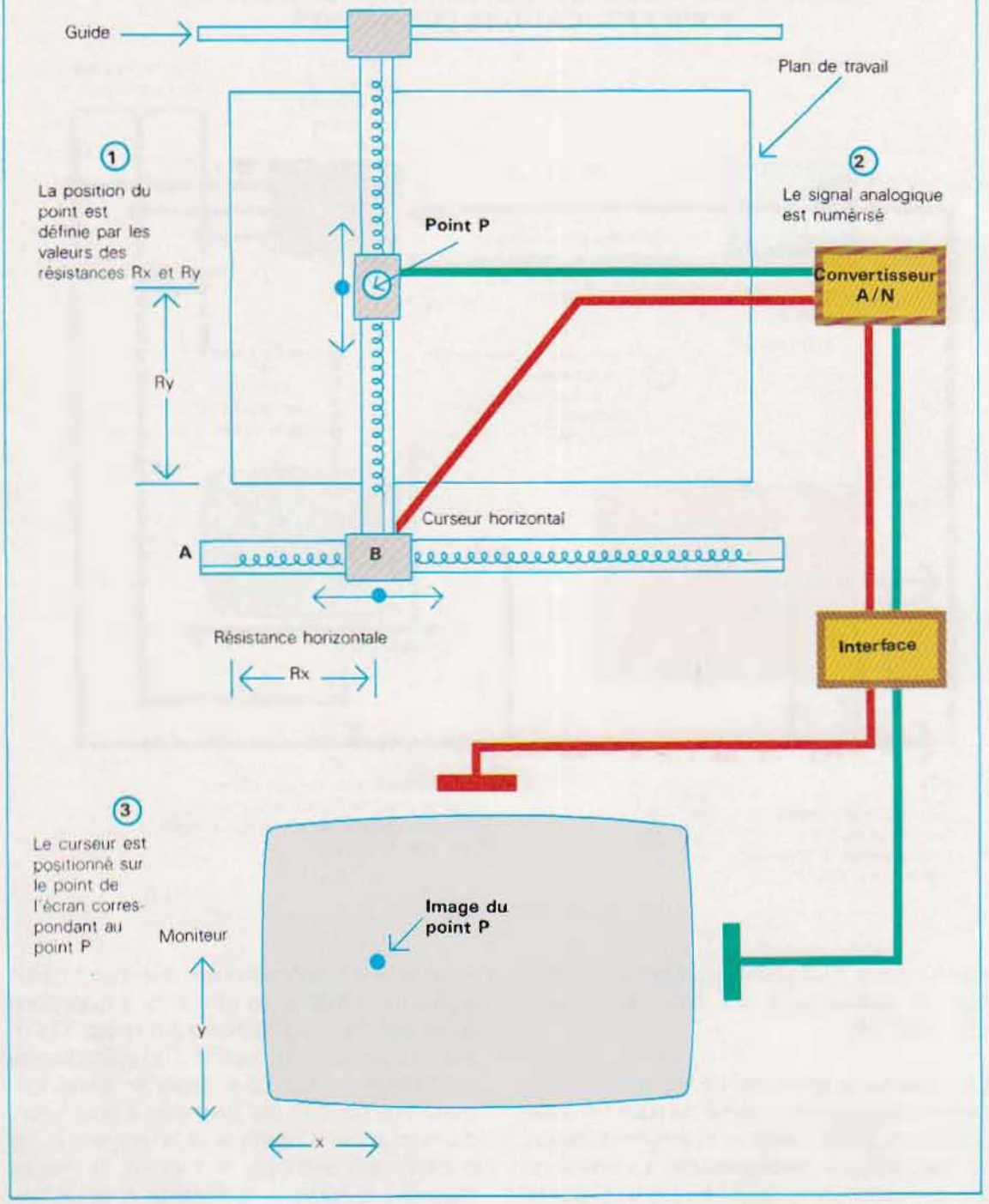


mesure que le nombre de bits augmente. Ainsi, avec un convertisseur à 5 bits, elle passe à 1/31, soit 3 %.

La tablette graphique. Le transducteur potentiométrique est très utilisé dans les tablettes graphiques, mais avec une légère modification par rapport au schéma présenté. La construction des potentiomètres de forme rectiligne et de longueur égale à celle d'une feuille à dessin est très coûteuse et demande l'emploi d'un système de guides.
Le potentiomètre rotatif (voir figure 2 en page 1391) est beaucoup plus économique mais il mesure un angle et non une distance.

Pour obtenir les coordonnées d'un point, il faut utiliser un système de référence à **coordonnées polaires** (voir schéma 3 en page 1391). Avec ce système, la position d'un point du plan est définie à l'aide d'un angle et d'une longueur. Le point P, par exemple, a pour coordonnées polaires l'angle α et le segment A ; le point P1 est défini par α 1 et A1. Si l'on se reporte à la figure 2, la distance A est la longueur du bras, et l'angle sa rotation (dont dépend la valeur de résistance). Lorsque le bras tourne, la valeur de résistance (R) est proportionnelle à l'angle, et il est donc possible de repérer tous les points se trouvant sur la circonférence de même centre que le potentio-

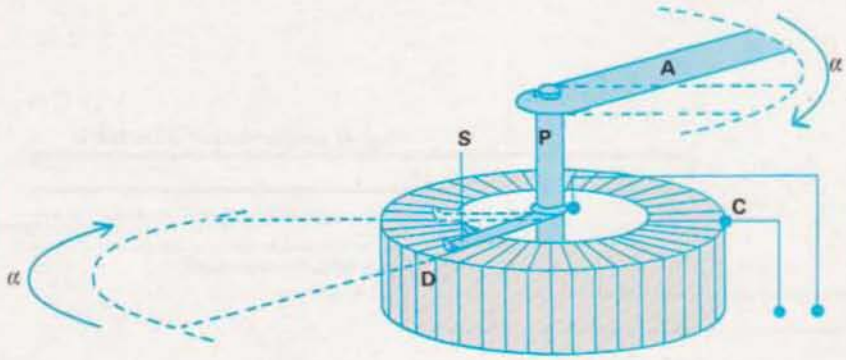
1 / SCHEMA D'UN TRANSDUCTEUR POTENTIOMETRIQUE DE POSITION



mètre et de rayon A. Avec ce système, on n'obtient pas la position de tous les points du plan, mais seulement ceux qui sont circonscrits par le bras dans sa rotation. Pour obtenir tous les points, il faut monter un deuxième potentiomètre à l'extrémité du bras A, et un deuxième bras B. Toutes les positions du point de travail sont alors atteintes.

Le schéma détaillé de la tablette graphique est représenté p. 1392.

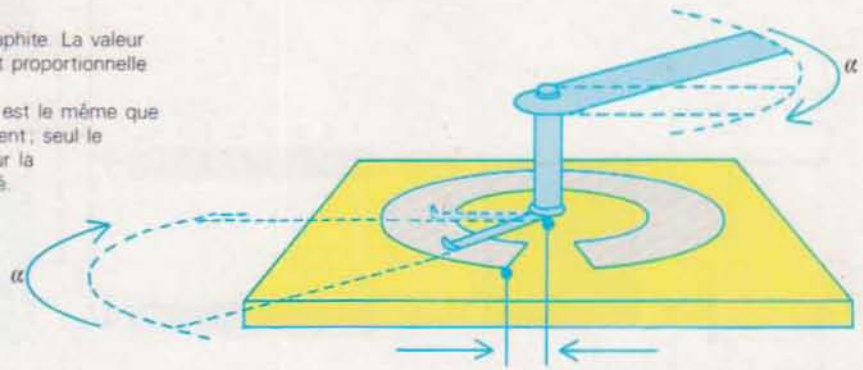
2 / POTENTIOMETRES UTILISES DANS LES TABLETTES GRAPHIQUES



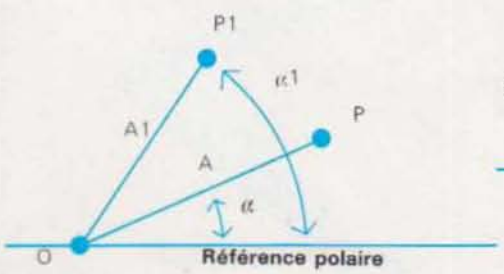
Résistance fonction de l'angle de rotation

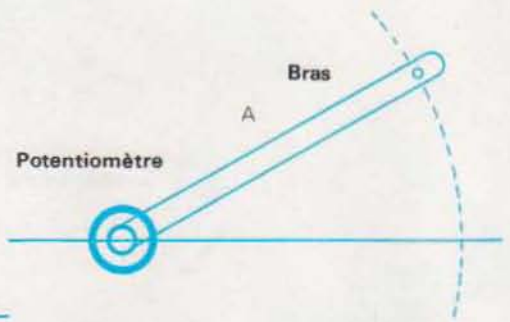
Schéma de fonotionnement d'un potentiomètre. La rotation de la tige A provoque celle de l'axe P qui, à son tour, entraîne le déplacement du curseur à frottement S. Quand la position de S varie, la quantité de fil électrique comprise entre les points C et D (respectivement l'extrémité de l'enroulement et le contact S), varie également. Pour des puissances limitées, comme dans ce type d'application, l'enroulement de fil est remplacé par une fine couche d'un matériau spécial (graphite, par exemple).

Potentiomètre à graphite. La valeur de la résistance est proportionnelle à celle de l'angle
Le fonctionnement est le même que dans le cas précédent ; seul le matériau utilisé pour la résistance a changé.



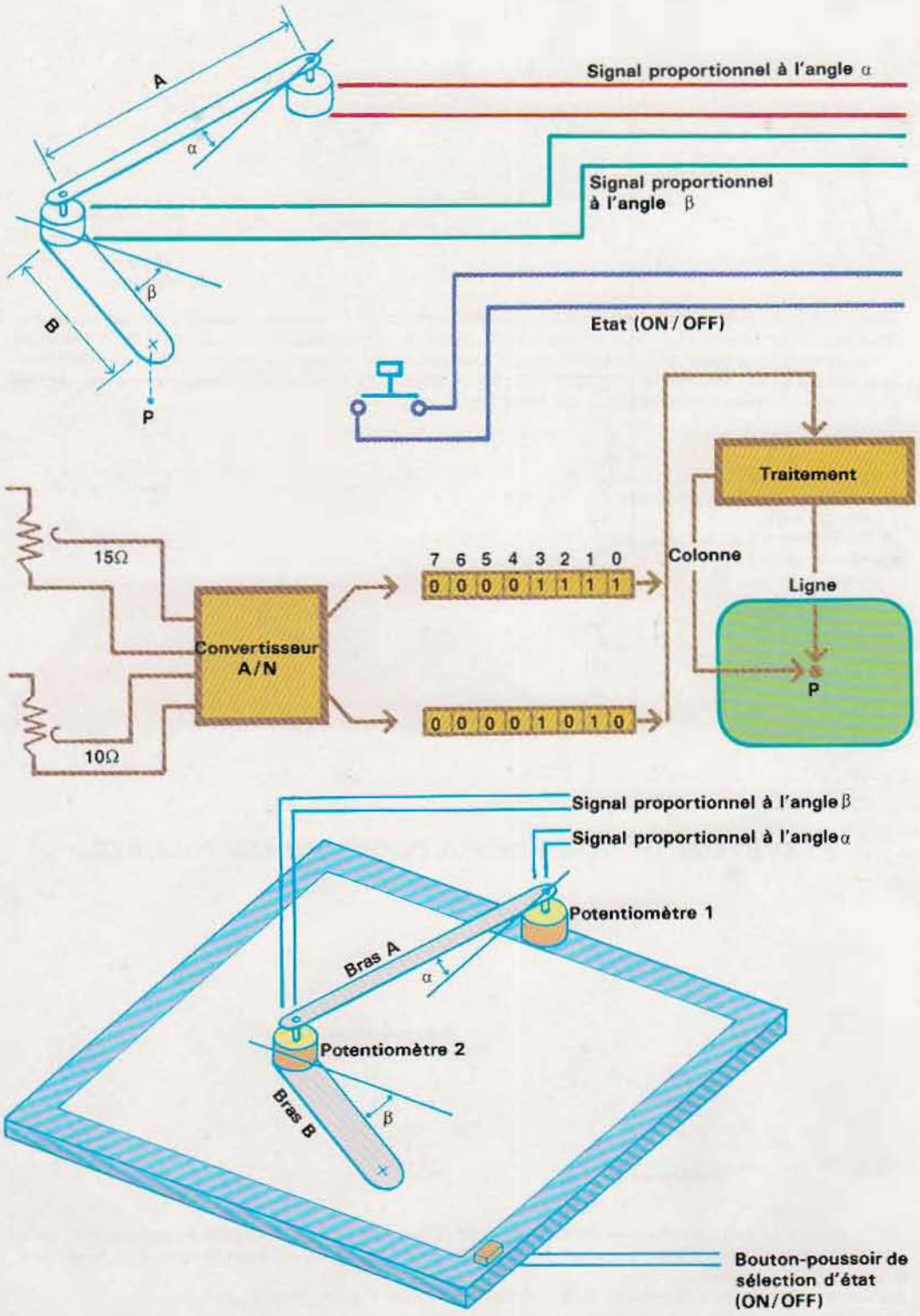
3 / SYSTEME DE REFERENCE A COORDONNEES POLAIRES





Coordonnées polaires. La position d'un point est repérable à partir des valeurs d'un angle et d'une distance. Celle-ci mesure le segment entre le point considéré et l'origine O ; l'angle pris en compte est compris entre le segment et une droite de référence.
Dans notre exemple, les coordonnées (A, α) définissent le point P et (A_1, α_1) le point P1.

4 / SCHEMA D'UNE TABLETTE GRAPHIQUE



Les coordonnées fournies par ce système sont polaires : il faut donc les transformer en coordonnées cartésiennes (X, Y). La méthode la plus simple est illustrée ci-dessous.
La formule de résolution est :

$X = A*COS(\alpha) + B*COS(\beta)$
$Y = A*SIN(\alpha) + B*SIN(\beta)$

avec comme données :

X,Y = coordonnées cartésiennes du point

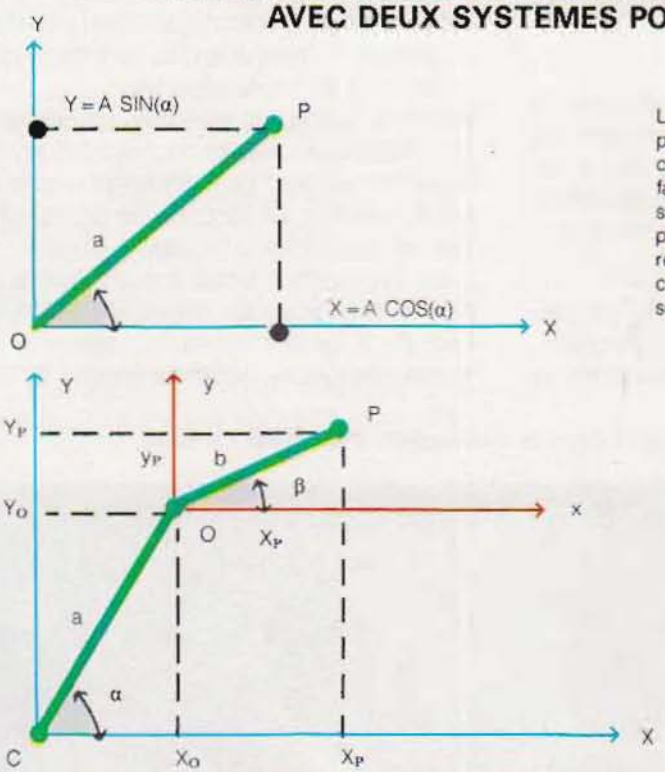
A,B = longueurs des deux bras
α,β = angles formés par les deux bras par rapport à une droite de référence

Normalement les longueurs des deux bras sont égales (A=B), et donc :

$X = A*(COS(\alpha) + COS(\beta)$
$Y = A*(SIN(\alpha) + SIN(\beta)$

Les valeurs X et B sont fournies par deux convertisseurs A/N (CAN) (un pour chaque potentiomètre) sous forme de valeurs numé-

DETERMINATION DES COORDONNEES D'UN POINT AVEC DEUX SYSTEMES POLAIRES



La transformation des coordonnées polaires (α ,A) en coordonnées cartésiennes (X, Y), est immédiate en faisant coincider les origines des deux systèmes de référence (point O) et en prenant l'axe X comme droite de référence des angles. Les coordonnées cartésiennes sont les projections du segment A sur les axes.

Les coordonnées du point P par rapport aux axes x, y sont :
$x_P = b\ COS(\beta)$
$y_P = b\ SIN(\beta)$

Celles du point O par rapport aux axes X, Y sont :
$X_O = a\ COS(\alpha)$
$Y_O = a\ SIN(\alpha)$

Les coordonnées de P par rapport au système X, Y s'obtiennent en additionnant les coordonnées de P par rapport à x, y à celles de O par rapport à X, Y :
$X_P = X_O + x_P = a\ COS(\alpha) + b\ COS(\beta)$
$Y_P = Y_O + y_P = a\ SIN(\alpha) + b\ SIN(\beta)$

a et b étant constants (longueurs des bras de la tablette), la position du point P est déterminée par les valeurs des angles de rotation nécessaires au positionnement de l'élément de contact en P.

riques : le logiciel se limitera au calcul des deux expressions mentionnées. Les transducteurs potentiométriques ne sont pas les seuls dispositifs employés. Dans les systèmes de grande taille, ils ne sont pas du tout utilisés. Leur inconvénient provient de l'encombrement des bras sur le plan de travail et aux tolérances mécaniques nécessaires. Les dispositifs dotés de photostyles fournissent de meilleurs résultats. Le transducteur comprend, comme élément sensible, un photostyle dont la pointe, composée d'un interrupteur, se ferme lorsqu'il y a pression sur le papier et qui signale ainsi l'état ON/OFF du point. Un tel transducteur nécessite des systèmes de relevé de cordonnées bien plus complexes que le précédent.
Parmi les plus usités, on notera :

- Les champs électriques ou magnétiques. La tablette génère un champ électrique ou magnétique ; le photostyle comporte un capteur qui fournit une tension proportionnelle à l'intensité du champ qui l'atteint, elle-même fonction de la position.
- Les ultrasons. Le photostyle émet, en permanence, des ultrasons. Des détecteurs, placés au bord de la feuille, mesurent le temps écoulé entre l'émission et la réception du son, proportionnel à la distance. On peut ainsi repérer la position du style par rapport aux détecteurs.

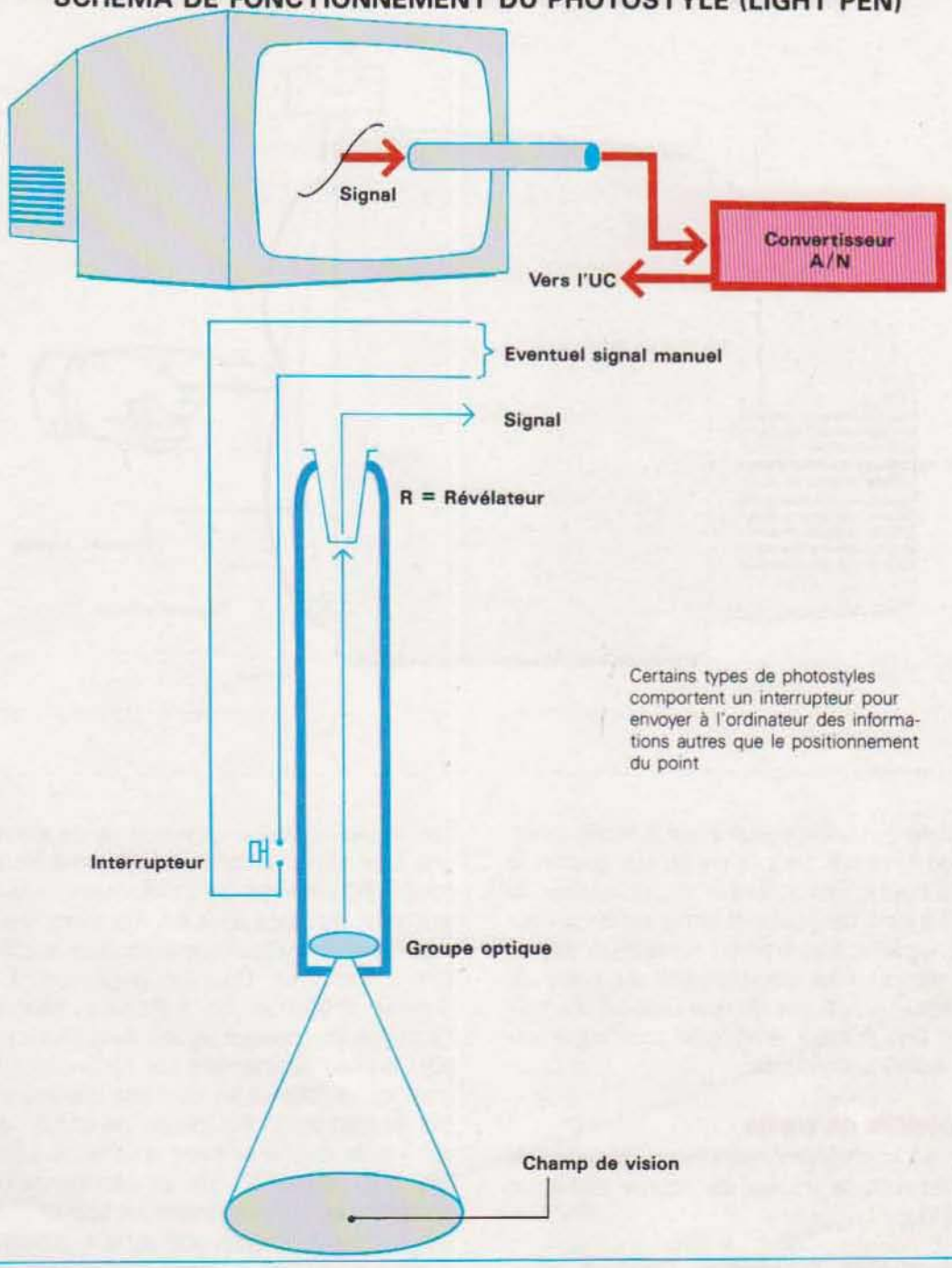
Ces méthodes nécessitent des matériels très complexes, qui ne sont pas à la portée des micros ou des ordinateurs personnels. Il existe quelques moyens — mécaniques le plus souvent — pour prendre les mesures d'objets tridimensionnels. Dans ces applications, outre le coût élevé du matériel, les logiciels manquent de puissance pour mémoriser et gérer des figures tridimensionnelles.

Dispositifs spéciaux. Il existe d'autres moyens pour communiquer des informations graphiques à l'ordinateur. Ils sont toutefois plus complexes et moins répandus.
Les deux principaux systèmes qui ont été portés, depuis peu, sur les micro-ordinateurs sont le photostyle (light pen) et l'écran tactile, sensible au toucher. Le fonctionnement du photostyle est illustré dans la page ci-contre.
C'est un capteur sensible à la lumière émise par le moniteur. En déplaçant le stylo sur l'écran, la lumière reçue est convertie en signaux électriques, envoyés ensuite à l'ordina-

Utilisation de programmes graphiques dans la conception d'un motif de tissu.

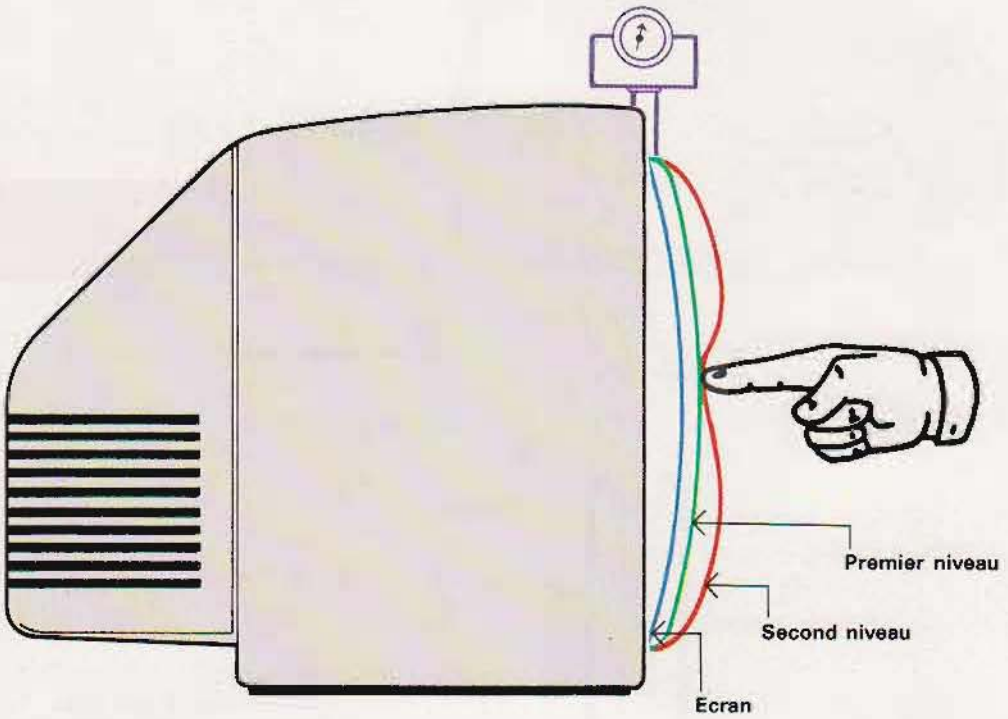
Marka

SCHEMA DE FONCTIONNEMENT DU PHOTOSTYLE (LIGHT PEN)



teur (après conversion analogique/numérique). Ces signaux, traités par une unité de contrôle reliée à l'écran, permettent de repérer les coordonnées du photostyle. On peut ainsi « lire » un dessin sur l'écran ou le créer comme s'il s'agissait d'un quelconque crayon à papier. Le fonctionnement d'un écran tactile est explicité par le schéma ci-dessus. La surface externe de l'écran est composée de deux feuilles de verre séparées par un espace très étroit. En

PRINCIPE DE FONCTIONNEMENT DE L'ECRAN TACTILE



exerçant une faible pression sur la feuille externe, on l'incurve jusqu'à ce qu'elle touche la feuille interne ; on obtient ainsi une variation de la différence de potentiel entre les deux couches laquelle, convertie en numérique, fournit une mesure des coordonnées du point de contact. Il s'agit, par comparaison, d'une méthode peu précise, employée pour accomplir des tâches secondaires.

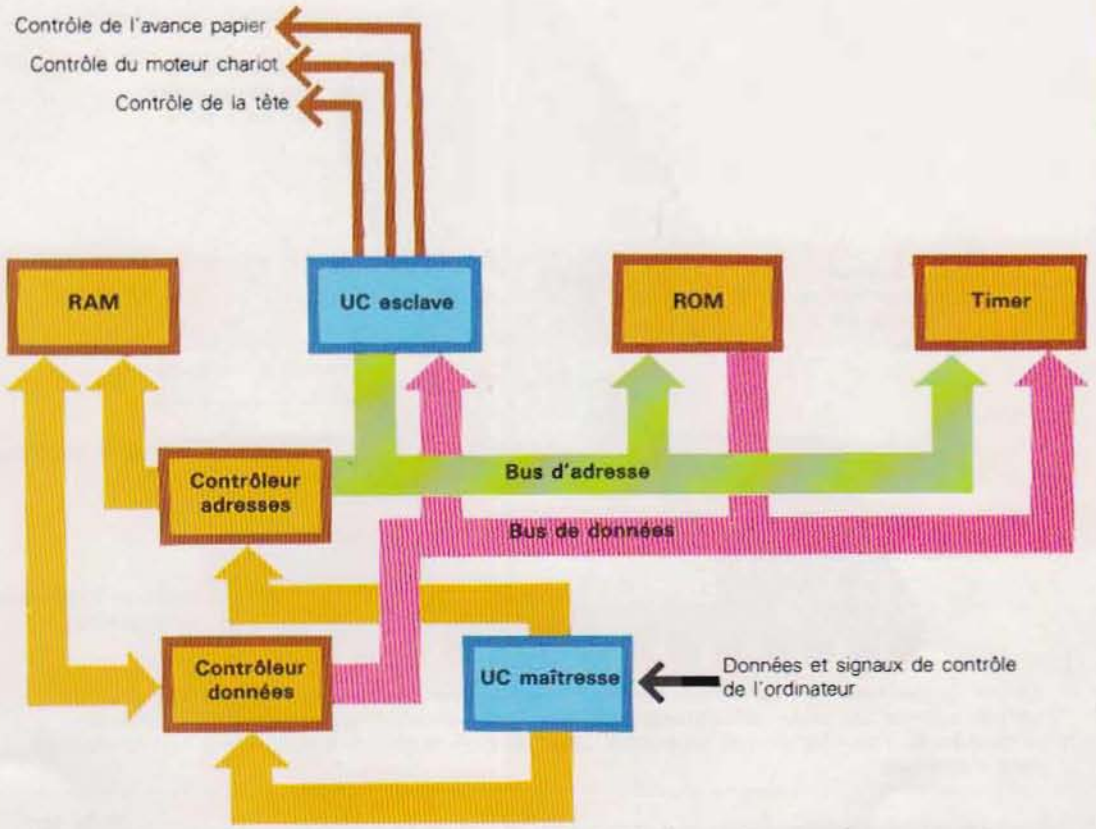
Dispositifs de sortie

Pour les applications graphiques, on retrouve l'imprimante, le traceur de courbe et l'écran graphique.

L'imprimante graphique. Pour les applications graphiques n'exigeant pas une précision très élevée, on peut utiliser les imprimantes dites semi-graphiques ou graphiques. Il s'agit d'imprimantes à aiguilles ordinaires (schéma ci-contre) permettant cependant d'adresser chaque élément de la matrice. Dans son emploi normal (impression de caractères), une série d'instructions correspond à chaque code ASCII envoyé par l'ordinateur ; ces instructions, stockées en ROM, n'activent que les aiguilles de la matrice correspondant au caractère à imprimer. Pour un graphique, il est possible d'éliminer ce traitement interne à l'imprimante ; chaque aiguille de la matrice est alors pilotée directement par l'ordinateur. Les graphiques obtenus ne sont pas très précis et leur gestion est plutôt lourde, surtout à cause des limites de mouvement du chariot. La tête d'écriture ne peut en effet se déplacer qu'horizontalement. Le mouvement vertical se réalise par avance du papier, une ligne à la fois. Le graphique obtenu a donc, sur un des deux axes, une résolution qui, dans le meilleur des cas, est celle de l'interlignage de l'imprimante. On peut légèrement améliorer sa qualité en sélectionnant soigneusement les aiguilles ; du point de vue graphique cependant, la présentation reste de mauvaise qualité.

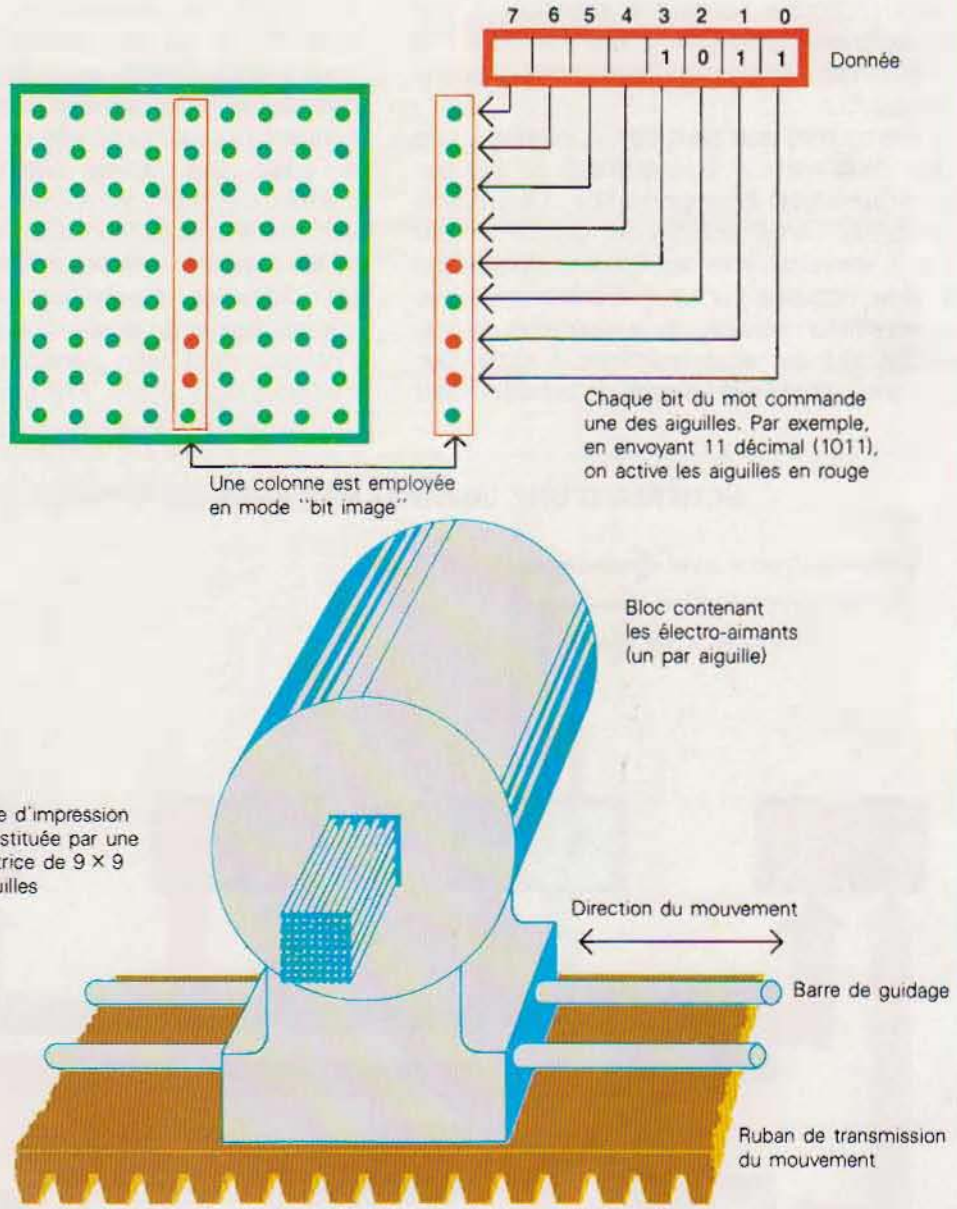
La méthode « column scan bit » (voir p. 1398) permet d'améliorer la qualité du graphique. Cette technique consiste à sélectionner une des colonnes composant la tête d'écriture et à commander de manière autonome chacune des aiguilles.
La même méthode peut être appliquée à une ligne de la matrice (bien que cela ne soit pas prévu sur toutes les imprimantes). Ce système adresse un certain nombre de points (7 × 9 ou 9 × 9, selon les dimensions de la matrice) dans la zone occupée par un caractère, avec une augmentation sensible de la résolution du graphique. Sur certaines machines, il existe des utilitaires, lancés par un code de contrôle, dont la fonctionnalité est de reproduire sur l'imprimante le contenu intégral de l'écran vidéo. En réalité, le déroulement de ces sous-programmes est plus complexe. Dans beaucoup d'équipements, on a prévu une zone de mémoire spéciale, appelée page graphique, qui contient l'image numérisée de ce qui apparaît à l'écran. Les routines décrites prélèvent le contenu de cette zone mémoire et, en fonction des codes binaires rencontrés, mettent en route les aiguilles correspondantes. Dans ce cas, les difficultés proviennent du mécanisme d'adressage et de la sélection de la mémoire à l'intérieur de la page graphique. De plus, dans certaines applications, il faut, avant le transfert

SCHEMA D'UNE IMPRIMANTE SEMI-GRAPHIQUE



L'UC maîtresse reçoit les données et les signaux de contrôle de l'ordinateur (ou mieux des circuits et des interfaces) et les transfère dans la RAM ou met en action une des ROM selon le mode d'impression choisi.
En écriture normale, la ROM qui contient le jeu de caractères sélectionnés est connectée au bus de données.
L'UC esclave enregistre ces signaux et les envoie aux organes contrôlant les dispositifs d'impression.
En mode graphique les données reçues par l'UC maîtresse sont transférées dans la mémoire RAM et ensuite aux organes mécaniques via l'UC esclave.
Les imprimantes plus sophistiquées comportent plusieurs ROM, en fonction du type de caractère que l'on veut obtenir.

METHODE D'IMPRESSION "BIT IMAGE"



Ce type de fonctionnement est normalement mis en route en deux temps. On envoie d'abord un code de contrôle suivi par une valeur numérique qui exprime le nombre de bits à représenter (ils seront envoyés ultérieurement). Puis il faut envoyer les données, groupées en mots de 7 ou 8 bits, selon la logique présentée dans le graphique.

sur papier, inverser l'écran vidéo, car le dessin est tracé en blanc sur noir alors que l'imprimante ne peut écrire qu'en noir sur blanc. Si le graphique est composé d'un dessin où le sens tridimensionnel est obtenu avec des zones hachurées, l'absence d'inversion provoque des résultats catastrophiques.

La table traçante. C'est l'unité de sortie la plus usitée dans les applications de caractère technique. Elle fournit des graphiques d'une qualité comparable voire meilleure que ceux réalisés à la main.

Les traceurs se subdivisent en deux catégories, selon le moyen employé pour tracer le dessin :