propose

PASCAL BASE

version 1

Introduction :

La société FREE GAME BLOT est heureuse de proposer à tous les possesseurs d'ordinateurs Thomson T07 - TO7/70 - MO5, le langage **Pascal Base,**.

Le Pascal est un véritable langage informatique moderne qui reprend la totalité des notions actuelles et des règles de l'art réellement utilisées dans l'industrie informatique mondiale.

Avec **Pascal base** vous découvrirez et vous maîtriserez bientôt la structuration, la modularité, la récursivité.

En outre, vous approcherez de bien plus près qu'avec le basic, des performances étonnantes de votre micro-ordinateur.

Pascal Base vous apporte un outil très efficace qui rapidement vous mènera vers de nouveaux horizons en vous initiant à l'informatique de demain.

Pour bien démontrer les qualités du **Pascal Base** la société d'édition FREE GAME BLOT propose une dotation suivant les modalités diffusées dans la presse, qui récompensera les 20 meilleurs logiciels (jeux utilitaires, éducations, etc...) écrits en **Pascal Base.**

Pour tous les renseignements sur cette dotation, écrire à *FREE GAME BLOT, Cedex 205, 38190 Crolles.*

PASCAL BASE

MANUEL D'UTILISATION

Implantation sur MO5 - T07 ext 16K - T07/70 :

Destiné avant tout à l'apprentissage du langage Pascal, **le Pascal Base** représente en fait un sous-ensemble du langage développé par Wirth, ceci principalement pour une vocation d'initiation au langage et une capacité mémoire de moyenne importance.

Si on considère les caractéristiques suivantes comme caractéristiques de toutes les possibilités du Pascal, à savoir :

— structuration des données
— structuration des programmes
— modularité des sous-programmes et fonctions
— récursivité totale.

Le Pascal Base reprend toutes ces caractéristiques à l'exception de la première. Le **Pascal Base** connaît un seul type de données, à savoir le type entier INTEGER, et tableaux d'entiers. L'utilisateur peut néanmoins, grâce aux tableaux, amener une structuration de ses données.

En revanche, les structures de programmes sont implémentées en totalité en **Pascal Base** (à l'exception de la structure WITH qui suppose une structuration des données), à savoir les structures IF, FOR, REPEAT, WHILE et CASE.

Les procédures et fonctions du **Pascal Base** sont conformes à la norme Pascal, avec passage des paramètres par valeur et par adresse.

Enfin, le **Pascal Base** utilise une pile d'exécution, rendant possibles tous les cas de récursivité (appel d'une procédure par elle-même). L'implantation sur la gamme Thomson est en outre caractérisée par :

1. la présence d'un éditeur pleine page permettant la création des programmes source
2. une compilation en une seule passe, mais nécessitant une étape de génération de code supplémentaire (via le "LOADER")
3. deux extensions au compilateur :

— la possibilité de définir une suite d'instructions 6809 au niveau du source, permettant d'augmenter l'accès au niveau machine,
— la possibilité de chaîner plusieurs programmes source au niveau de la compilation.

SOMMAIRE

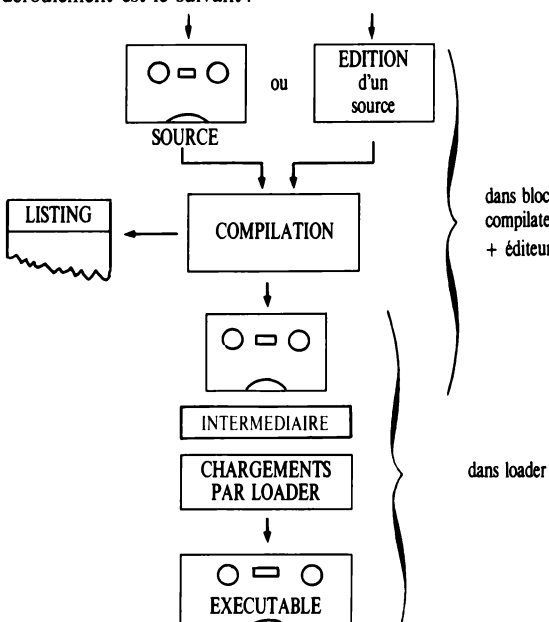
I. Description du système Pascal :

Le système Pascal comprend deux entités :

— un bloc "compilateur + éditeur", qui permet de charger et/ou modifier et/ou sauvegarder un programme écrit en Pascal (programme source), puis de le compiler afin d'obtenir un programme intermédiaire.

— un "loader" (chargeur) qui permet de reprendre un programme intermédiaire généré lors d'une précédente compilation et de créer un programme exécutable par la machine.

Le déroulement est le suivant :



Le bloc "compilateur + éditeur" est activé par un RUN "PASCAL" sur la cassette système.
Le "loader" est activé par un RUN "LOADER" sur la cassette système.

II. Utilisation du compilateur-éditeur

Après lancement par un RUN "PASCAL" du bloc compilateur + éditeur, le système affiche la ligne suivante qui représente le menu du système : "L)ire, E)diter, S)auvegarder, C)ompiler".
On choisit alors :

L : lecture d'un programme source sur cassette
E : édition d'un texte lu précédemment ou nouveau
S : sauvegarde d'un programme source sur cassette
C : compilation du texte source en mémoire.

II.1 Lecture d'un programme source

Le système demande : "Fichier :" et on donne le nom du fichier qui doit être chargé en mémoire. Par défaut, l'extension de fichier est. PAS.

II.2 Ecriture d'un programme source

Le système demande : "Nom du fichier :" et on donne le nom que prendra sur cassette le source qui est en mémoire. L'extension sera par défaut. PAS, sauf si elle est donnée explicitement. Ensuite, est affiché : "(ENTREE) quand cassette prête". On tape alors sur la touche ENTREE quand la cassette est prête en enregistrement.

II.3 Utilisation de l'éditeur

L'éditeur proposé dans le système Pascal Base est un éditeur pleine page, sur 24 lignes, la ligne du haut étant réservée à l'éditeur.

Si on avait auparavant lu un texte depuis la cassette, l'éditeur affiche alors la première page de ce texte, sinon un écran vierge est affiché. Le texte est organisé en lignes logiques, pouvant contenir chacune 120 caractères, soit 3 lignes physiques sur l'écran. Le compilateur lira en fait ligne logique par ligne logique, et un identifieur, un nombre, une chaîne de caractères pourra chevaucher une ligne **physique,** à condition de ne pas chevaucher une ligne logique.

On peut alors modifier ce texte en utilisant les touches suivantes :

↑ : permet de remonter d'une ligne sur l'écran. Si on était sur la première ligne du texte, une ligne blanche est insérée

↓ : permet de descendre d'une ligne sur l'écran. Si on était sur la dernière ligne du texte, une ligne blanche est créée

→ : permet de se déplacer d'une position à droite. En fin de ligne logique, on passe sur le premier caractère de la ligne suivante

← : permet de se déplacer d'une position à gauche. En début de ligne logique, on passe sur le dernier caractère de la ligne précédente.

ENTREE : insère une ligne blanche à la suite de la ligne logique courante, et se positionne sur cette ligne.

INS : insère un caractère espace à la position courante du curseur. Si la ligne logique comptait 120 caractères, le 120e caractère est perdu.

EFF : efface le caractère à la position du curseur.

RAZ : permet d'afficher la page suivante du texte. Sans effet si la fin du texte était déjà atteinte.

↰ : permet d'afficher la page précédente du texte. Sans effet si le début du texte était déjà atteint.

CNT-X : efface la fin de la ligne logique à partir de la position du curseur.

CNT-D : positionne le curseur en début de ligne logique courante.

CNT-E : positionne le curseur en fin de ligne logique courante.

CNT-A : affiche le caractère [

CNT-Z : affiche le caractère]

CNT-T : positionnement en début de texte

CNT-U : positionnement en fin de texte

CNT-S : positionnement au début de la ligne logique suivante

CNT-P : impression du texte sur imprimante

CNT-F : recherche d'une chaîne : la ligne de contrôle affiche alors > et on tape la chaîne à retrouver, terminée par ENTREE. Le curseur se positionne alors sur la première occurrence de la chaîne comprise entre la position curseur actuelle et la fin du texte.

CNT-N : effacement du texte complet, après confirmation

CNT-Q : fin d'édition et retour au menu général du système Pascal.

CNT-W : affiche le caractère _ (souligné).

Note : La mémoire de texte étant limitée, le message "Buffer plein" peut survenir ; dans ce cas on ne peut que faire des déplacements ou suppressions.

II.4 Utilisation du compilateur

Lorsque l'on a choisi l'option "C" du système **Pascal Base**, le processus de compilation commence si un texte est présent dans la mémoire de l'éditeur (sinon, impression du message "Source non trouvé" et retour au menu du système). Le compilateur demande alors "LISTING (O/N/P) :" et l'on répond :

O — si on désire le listing de compilation à l'écran,
N — si on ne veut pas de listing de compilation,
P — si on désire le listing de compilation sur imprimante.

"OBJET (O/N) :" et l'on répond :

O — si on désire que le fichier intermédiaire de compilation soit écrit sur cassette,
N — si on désire faire uniquement une vérification syntaxique du programme et ne pas générer de code intermédiaire.

Si on a répondu "O", le compilateur demande : (ENTREE) quand cassette prête :" et on tape sur la touche ENTREE quand la cassette destinée à recevoir le fichier intermédiaire est prête en enregistrement.

Le processus de compilation commence alors, et s'arrêtera soit sur la fin normale du programme, soit sur une erreur détectée. Dans ce dernier cas, le numéro de l'erreur est imprimé avec une flèche indiquant à quel endroit dans la ligne, l'erreur a été détectée. Il faut alors ré-éditer le texte, puis le recompiler.

Dans le cas d'une fin normale du programme, il faudra ensuite appeler le "Loader" qui reprendra le code intermédiaire afin de créer un code exécutable. Le nom du fichier intermédiaire sera le nom donné après la directive PROGRAM, complété de l'extension INT. Ainsi, pour un programme comprenant la ligne PROGRAM ESSAI 1 ; le fichier intermédiaire se nommera ESSAI 1.INT.

II.5 Utilisation du Loader

Le "Loader" est destiné à recevoir du code intermédiaire issu du compilateur et à le transformer en code exécutable 6809. Il est activé, en utilisant la cassette système, en tapant LOAD "LOADER".

Une fois le chargement effectué, il demande "Fichier intermédiaire :"; on tape alors le nom du fichier intermédiaire à transformer (c'est le nom donné après le PROGRAM dans le fichier source), l'extension. INT est prise par défaut.

A partir de ce moment, le loader recherche sur cassette le fichier intermédiaire. Le processus de transformation est alors effectué et le loader

donne alors les adresses début et fin du code exécutable, puis affiche
"<ENTREE> quand cassette prête"
On tape alors sur la touche "ENTREE" quand la cassette destinée à recevoir le fichier binaire exécutable est prête en enregistrement.

Le fichier sauvegardé aura le même nom que le fichier intermédiaire, mais avec l'extension. BIN.

Le programme peut être lancé par un EXEC adresse, l'adresse étant celle donnée par le loader comme adresse de début. Il peut néanmoins être déplacé en mémoire, puisque entièrement relogeable.

III. Compléments au manuel de programmation

L'implantation sur MO5 et TO7, TO7/7O du compilateur **Pascal Base** apporte quelques modifications au manuel de programmation.

III.1 Nouvelle procédure prédéfinie ENCODE

Elle permet d'entrer une séquence d'instructions 6809 qui devront être exécutées au sein du programme Pascal. Cette séquence devra être codée en héxadécimal. Ainsi, si on veut afficher A, il faut faire en assembleur

```
LDB   #'A soit C6 41
SWI O2H soit 3F O2
```

il suffira de mettre dans le source Pascal

```
ENCODE ('C6413FO2') ;
```

Le code généré devra obligatoirement être relogeable. On peut également dans un ENCODE, adresser directement un paramètre ou une variable Pascal, en retenant que:

— toutes variables INTEGER occupent 2 octets
— tout tableau occupe (borne sup - borne inf + 1) * 2 octets pour chaque dimension
— paramètres et variables sont rangés en ordre décroissant dans la mémoire, paramètres d'abord.
— le 1er paramètre (ou 1ere variable si pas de paramètres) est rangé en -8 par rapport au registre U
— dans le cas d'un paramètre passé par adresse, le paramètre contient une adresse de variable.

Ainsi, si T= ARRAY [1..3] OF INTEGER et que l'on a : PROCEDURE TOTO, (VAR I : INTEGER ; J : INTEGER) ; VAR A, B : INTEGER ; TAB : T ;
Les variables sont rangées ainsi :
— adresse de I en -8,U
— valeur de J en -10,U
— valeur de A en -12,U

— valeur de B en -14,U
— valeur de TAB [1] en -16,U
— valeur de TAB [2] en -18,U
— valeur de TAB [3] en -20,U

De plus, la valeur retournée par une fonction se trouve à l'adresse contenue dans U.
Ainsi, pour définir une fonction GET qui retourne la valeur ASCII d'une touche tapée au clavier : (MO5 uniquement)

```
FUNCTION GET: INTEGER; BEGIN ENCODE ('3F0AC10027FA4FEDC4'); END;
```

En faisant dans le programme, A = GET ; on obtiendra dans A, la valeur ASCII du caractère frappé au clavier.
Ici, le code assembleur est :

```
3FOA AO:  SWI    OA#     lecture clavier
C1OO      CMPB   #O      si B à O, pas de touche enfoncée
27FA      BEQ    AO      alors, on recommence
4F        CLRA           O→A
EDC4      STD    ,U      (A,B) : résultat de la fonction.
```

Attention : les registres D et Y peuvent être modifiés dans un ENCODE, mais X, S, U doivent rester inchangés (ou modifiés, mais restaurés).

III.2 Inclusion de fichier source à la compilation

Dans la version MO5/TO7, on peut lors de la compilation indiquer que l'on veut charger un texte qui devra être compiler à la suite du texte en cours. Ceci servant principalement dans le cas où le programme à compiler est trop important pour tenir en entier dans l'éditeur. L'inclusion se fait en mettant un commentaire spécial : (* $ nom de fichier *) dans le programme.

Le compilateur demande alors d'enlever la cassette du code intermédiaire (attention à ne pas déplacer la position de la bande), de mettre la cassette du fichier à inclure, puis taper ENTREE

Il remplace alors le texte en mémoire avec le nouveau texte, demande de remettre la cassette du code intermédiaire et reprend la compilation.
Notes : Le texte originel est **effacé** de la mémoire. Il y a donc intérêt à l'avoir sauvegardé **avant** la compilation.
— tout ce qui suit un commentaire spécial d'inclusion dans un texte n'est jamais compilé. Le commentaire d'inclusion doit donc normalement n'être suivi d'aucune autre ligne.
— après compilation, si on retourne dans l'éditeur, c'est le fichier "inclus" que l'on y retrouve.

Exemple d'utilisation :
Fichier DEMO :

```
PROCEDURE P1 ;
BEGIN
   .
   .
   .
corps de P1
   .
   .
   .
END ;
BEGIN
   .
   .
   .
programme principal
   .
   .
   .
END.
```

Fichier MAINDEMO :

```
PROGRAM MAINDEMO ;
CONST
   .
   .
   .
TYPE
   .
   .
   .
VAR
   .
   .
   .
(* $ DEMO*)
```

A la compilation de MAINDEMO, le compilateur, après avoir compilé les constantes, types et variables, ira chercher la suite du programme dans DEMO.

III.3 Codes d'erreurs supplémentaires :

184 : erreur dans instructions ENCODE (caractère non hexadécimal),
185 : longueur impaire sur chaîne ENCODE,
186 : chaîne attendue dans instruction ENCODE,
263 : erreur - fin de fichier trouvée avant la fin du programme (END.)

PASCAL BASE

MANUEL DE PROGRAMMATION

Le présent manuel a pour fonction de présenter le langage **Pascal Base** dont les rôles et objectifs sont exposés dans le manuel d'utilisation.

Une grande importance sera donnée à la syntaxe du Pascal, représentée sous forme de diagrammes, et qui devra être respectée sous peine d'erreurs à la compilation.

On y abordera successivement la structure du programme Pascal, les objets Pascal - identifieurs, nombres - les expressions, la notion de bloc.

A partir de cette notion de bloc, fondamentale en Pascal, on développera l'écriture des parties déclarations et instructions.

Enfin, un exemple complet de programme en Pascal sera donné.

SOMMAIRE

I. Le programme Pascal

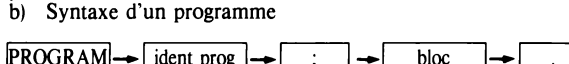
a) Un programme Pascal est constitué comme une phrase, c'est-à-dire comme un ensemble de mots (et éventuellement de nombres et de caractères spéciaux) terminée par un point.

Il est organisé en lignes (non numérotées comme en Basic), plus pour des raisons de mise en page que de séparation d'instructions. Néanmoins, il faut avoir à l'esprit qu'un identifieur, ou un nombre, ou une chaîne de caractères entre quotes ne peut être "à cheval" sur 2 lignes.

Les identifieurs et nombres doivent être séparés entre eux soit par des caractères spéciaux, soit par des espaces.

Ainsi dans FOR I := 1 TO 10 des espaces sont nécessaires entre FOR et I, entre TO et 10, mais non nécessaire des deux côtés du symbole :=, attention aussi à ne pas avoir d'espaces entre : et = dans le symbole Pascal := (ainsi que pour.., >= etc...). Les commentaires en Pascal s'écrivent de la façon suivante : (*... commentaire....*) et peuvent comprendre plusieurs lignes. Des commentaires spéciaux, dominant des instructions au compilateur, s'écrivant : (* $... commentaire spécial...*) peuvent exister selon l'implantation ; se reporter pour cela au manuel d'utilisation.

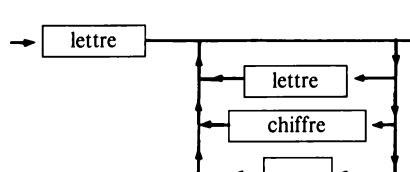
b) Syntaxe d'un programme



L'identificateur de programme peut avoir ou ne pas avoir d'utilité, selon les implantations (référence d'un fichier intermédiaire de compilation, etc..). Voir pour cela le manuel d'utilisation.

I.1 Les identifieurs

a) Syntaxe :



La syntaxe diffère peu de celle du Basic, à savoir une lettre suivie de caractères pouvant être des lettres, des chiffres ou des _ (souligné).

Le Pascal accepte des identifieurs de 1 à 8 caractères, les caractères au-delà du huitième étant ignorés. Le caractère _ sert à rendre plus lisible l'identifieur, mais sera ignoré (ID _ 1 _ 2 sera identique à ID12).

Contrairement au Basic, où les identifieurs ne servent que pour les variables, en Pascal, les identifieurs peuvent désigner un nom de programme, de constante, de type, de variable, de procédure ou de fonction.

b) **Exemples :**

I
I3
ID_PROG
F1A2
} corrects

2B erroné commence par un chiffre

A+1 erroné en tant qu'identifieur à cause du +

IDENTIFIC2
IDENTIFIEUR
} corrects, mais représentent le même objet

c) **Notes :**

— tout identifieur, en Pascal, doit être déclaré avant toute utilisation; ex : l'instruction A:=1; ne sera valable sémantiquement que si A a été déclaré précédemment (par VAR A : INTEGER;)

— un identifieur en Pascal a une "portée" et peut-être "global" ou "local'. Ces notions seront vues en détail avec la notion de bloc de programme. Pour l'instant, on retiendra qu'un même identifieur peut être déclaré plusieurs fois avec des portées différentes. Cela n'est valable que dans le cas des identifieurs pré-définis (comme ABS, READ, MAXINT..) et les identifieurs définis par l'utilisateur, mais pas dans le cas des identifieurs réservés (comme VAR, REPEAT...).

I.2 Les nombres

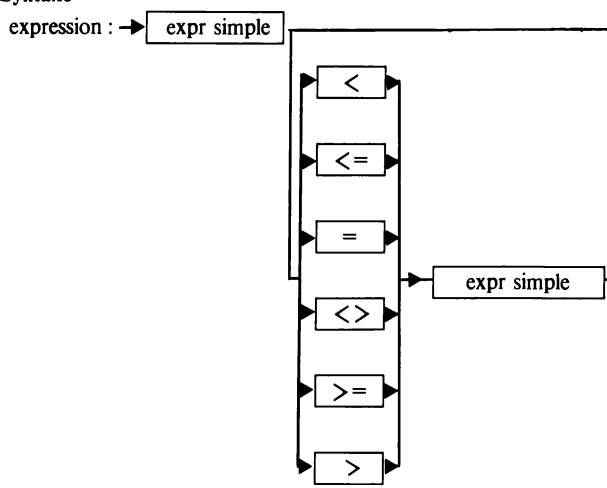
Dans un Pascal "entier", seuls des nombres entiers sur 16 bits sont utilisables, c'est-à-dire des nombres compris entre -32767 et + 32767. Ils peuvent être écrits en décimal, ou en hexadécimal (non signé) si précédés du symbole #

Exemples : 100
—1
AB (171)
FFFF (—1).

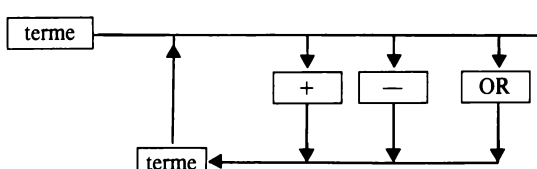
1.3 Les expresssions

Elles sont exclusivement numérique dans un PASCAL n'acceptant que le type INTEGRER :

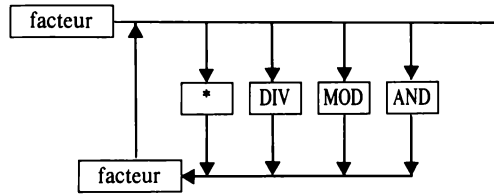
a) Syntaxe



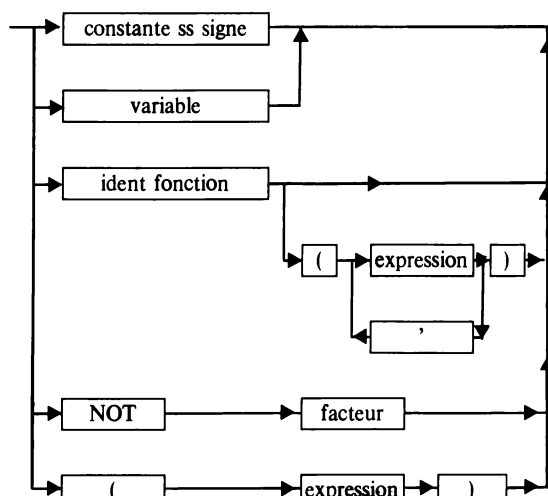
expression simple :

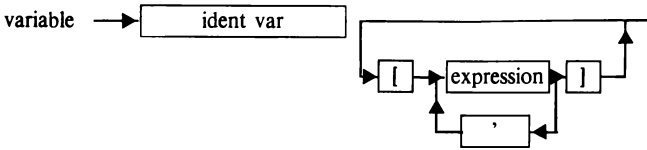


terme :



facteur :





b) Composition des expressions

Elle découle directement de la syntaxe ci-dessus et on peut établir la hiérarchie des opérations (en partant des plus prioritaires) :

1. NOT, expressions entre parenthèses
2. *, DIV, MOD, AND
3. +, —, OR, + et — unaires
4. <, <=, =, <>, >, >=

Attention à cette hiérarchie qui est différente de celle du Basic ainsi X = Y OR X = Z est bon en Basic; (= plus prioritaire que OR alors qu'en Pascal, on a une erreur de compilation, car OR est évalué en premier (il faut écrire (X=Y) OR (X=Z)).

c) **Opérateurs Pascal**

Outre les classiques +, -, *, (...), <, <=, =, <>, >, >=, on dispose en Pascal des opérateurs suivants :

— DIV : donne le quotient de la division; attention : / représente en Pascal une division réelle, et est donc absent dans ce Pascal.

— MOD : donne le reste de la division.

Exemple :

13 DIV 3 → 4	et	13 MOD 3 → 1
—13 DIV 3 → —4	et	—13 MOD 3 → —1
13 DIV (—3) → —4	et	13 MOD (—3) → +1
13 DIV (—3) → 4	et	—13 MOD (—3) → —1

— AND : intersection logique des deux entiers.

— OR : réunion logique des deux entiers.

— NOT : (unaire) : complément à un de l'entier; ainsi

13 AND 6 → 4

13 OR 6 → 15

NOT 1 → -2

Note : Les opérateurs de relation <, <=, =, <>, >, >= fournissent comme résultat - 0 (0000 hexa) si la relation est fausse, - —1 (FFFF hexa) si la relation est vraie.

d) **Elements de tableau**

On y accède en écrivant le nom du tableau, suivi des indices séparés par des virgules, entre crochets. Ex : TABLEAU [1, A + 2, K [4]]

e) **Utilisation des fonctions**

On peut utiliser des appels de fonctions dans des expressions, l'appel se fait alors comme pour des procédures. Ainsi, si on a défini la fonction :
FUNCTION FONC (A,B : INTEGER) : INTEGER;
on peut l'utiliser ainsi dans une expression :
A + 4 - FONC (X, TABLEAU [1]) DIV 2.

Une fonction prédéfinie ABS existe qui retourne la valeur absolue d'un nombre
ABS (3) → 3 et ABS (-3) → 3.

II. Le bloc Pascal

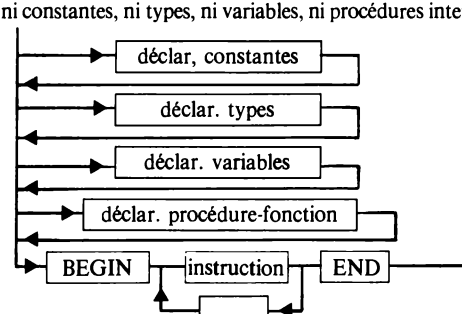
C'est une entité fermée comprenant une suite d'instructions avec les éléments qu'elle manipule. Cette entité peut donc être le programme lui-même, ou une procédure ou fonction.

Du fait que tout objet Pascal doit être déclaré avant d'être utilisé, le bloc se compose dans l'ordre :

— des déclarations de constantes
— des déclarations de types
— des déclarations de variables
— des procédures et fonctions internes au bloc
— du corps du bloc, c'est-à-dire une suite d'instructions délimitées par BEGIN... END.

Seul le corps du bloc est obligatoire, un programme ou une procédure pouvant n'avoir ni constantes, ni types, ni variables, ni procédures internes.

a) **Syntaxe :**



b) **Exemples :**

```
Procédure TITI ;
CONST N = 10 ;
TYPE TAB = ARRAY [1..N] of INTEGER ;
VAR T : TAB ;
BEGIN, WRITE (T [5]) ;
END ;
```

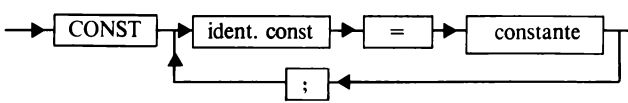
programme minimum (qui ne fait rien).

```
PROGRAM TOTO ;
BEGIN
END.
```

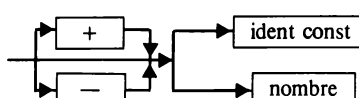
(l'instruction entre BEGIN et END est une "instruction vide".

II.1 La déclaration de constantes

a) Syntaxe :



avec constante :



b) **Fontionnement :**

Permet de déclarer les constantes numériques qu'utilisera le bloc dans lequel est faite la déclaration.

Le fait de déclarer des constantes permet d'éviter, si une valeur utilisée souvent est changée, de la modifier à plusieurs endroits dans le bloc. Il suffira de modifier la valeur de la constante.

c) **Equivalent Basic :**

On peut aussi déclarer des constantes en Basic, en écrivant par exemple C1 = 3 mais deux différences doivent être signalées :

— en Basic, on a écrit une instruction alors qu'en Pascal, CONST C1 = 3; est une déclaration et ne genère pas de code.

— en Basic, rien n'empêche d'écrire plus loin C1 = 6, alors qu'en Pascal, on ne peut affecter une constante : C1:=6 ; générerait une erreur.

d) **Exemples :**

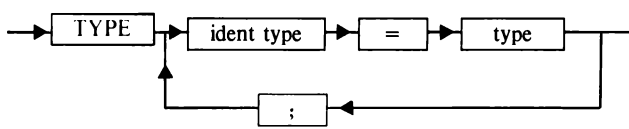
```
CONST
CONSTANT = 7;
C1 = —3;
C2 = —C1;
```

e) **Notes :**

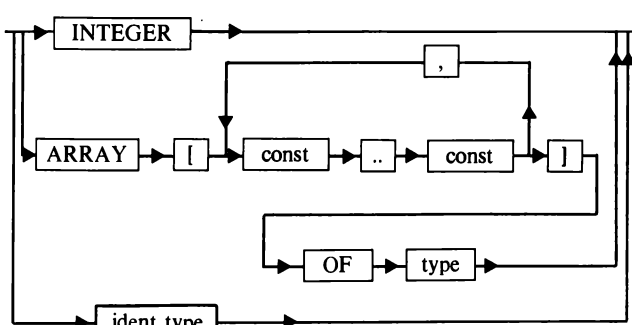
Il existe une constante prédéfinie en Pascal, MAXINT, et dont la valeur est le plus grand entier représentable, soit 32767, et qui peut être utilisée dans un programme.

II.2 La déclaration de types

a) Syntaxe :



avec type



b) **Fonctionnement**

Permet de déclarer les types qu'utiliseront les variables du bloc dans lequel est faite la déclaration. Les déclarations de type permettent surtout de donner un nom symbolique pour des types utilisés souvent. MAT 1010 = ARRAY [1..10,1..10] OF INTEGER, permettra de déclarer les tableaux A et B comme : VAR A,B : MAT 1010 ;

c) **Equivalent Basic :**

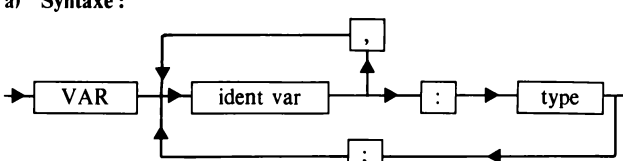
Pas d'équivalent.

d) **Exemples :**

```
TYPE
     INT          = INTEGER ;
     ARR          = ARRAY [-3..2] OF INTEGER ;
     DIM1         = ARRAY [1..10] OF INTEGER ;
     DIM2         = ARRAY [1..10] OF DIM1 ;
     (équivalent à
     DIM2         = ARRAY [1..10,1..10] OF INTEGER ;
     DIM2         = ARRAY [1..10] OF ARRAY [1..10] OF INTEGER ;
avec CONST
     BORNE 1      = —4 ; BORNE 2 = 3 ;
TYPE LIGNE        = ARRAY [BORNE 1..BORNE 2] OF INTEGER
```

II.3 La déclaration de variables

a) **Syntaxe :**



b) **Fonctionnement :**

Permet de déclarer toutes les variables qui seront utilisées dans le bloc auquel appartient la déclaration. Rappel : tout objet Pascal, principalement les variables, doivent être déclarées avant utilisation.

c) **Equivalent basic :**

Pas d'équivalent (déclaration implicite).

d) **Exemple :**

```
VAR
  A,B : INTEGER ;
  T1,T2 : ARRAY [1..10] OF INTEGER ;
avec type :
  MAT 1010 = ARRAY [1..10,1..10] OF INTEGER ;
VAR
  U : MAT 1010 ;
  V : ARRAY [1..10] OF MAT 1010 ;
```

III. Procédures et fonctions

Elles diffèrent fondamentalement de ce qui existe en Basic.
Pour plusieurs raisons :

— elles ont un nom en Pascal,
— on peut leur passer des paramètres, avec deux modes de passage,
— une procédure ou fonction Pascal constitue un **bloc** qui peut avoir des constantes, types, variables qui lui sont locales, et sont inconnus à l'extérieur du bloc.

La structure Basic s'en rapprochant le plus est la structure :
— DEF FNx (déclaration),
— ...FNx (...) (utilisation),
où l'on retrouve les deux premières caractéristiques citées plus haut (avec quelques restrictions).

a)- **Nom des procédures et fonctions :**

C'est un identifieur ayant les mêmes caractéristiques que tout identifieur Pascal. Il servira à appeler la procédure ou la fonction.

b)- **Paramètres :**

Ils permettent de passer des valeurs entre deux procédures, sans se soucier au niveau de la procédure appelée du nom des variables que lui passe la procédure appelante. Ainsi la procédure

```
PROCEDURE PROC2 (X : INTEGER) ;
```

qui possède un paramètre X, peut être appelée par :

```
PROC2 (A)
```

ou

```
PROC2 (2)
```

etc.

Au moment de l'appel, la valeur de A ou la valeur 2 sera copiée dans le paramètre X, et la procédure PROC2, en utilisant X, utilisera en fait soit la valeur de A, soit 2.

On distingue en fait, en Pascal, deux modes de passage des paramètres :

- passage par valeur
- passage par adresse

c)- **Passage par valeur :**

Lors d'un passage par valeur, le mécanisme est le suivant : une copie du paramètre est effectuée, et la procédure ou fonction travaille sur cette copie, n'altérant en aucun cas la valeur du paramètre chez l'appelant. On a alors comme *caractéristiques :*

1. Au moment de l'appel, le paramètre peut être une expression au sens général du terme.
2. C'est la valeur qui est copiée lors de l'appel et la procédure appelée travaille sur cette copie.
3. Dans la procédure appelée, on ne peut modifier la valeur du paramètre (puisqu'il s'agit d'une copie).
4. Ce mode de passage est généralement utilisé pour des paramètres en entrée de la procédure appelée.
5. Dans le cas où le paramètre est un tableau, ce mode de passage est gourmand puisqu'il effectue une copie complète du tableau.

Déclaration du paramètre :

Un paramètre est passé par valeur lorsqu'il n'est pas précédé du mot-clé VAR.

Exemple :

```
Déclaration : PROCEDURE P1 (X : INTEGER) ;
              BEGIN
                .
                .
                .
                U : = X ;
                .
                .
              END ;
Utilisation : P1 (A);       si A : INTEGER
              P1 (A + 7) ;
```

A ou A + 7 est copié, et X représente alors cette copie dans P1. A noter que dans P1, l'écriture X : = U serait interdite.

Paramètre tableau :

```
Si TYPE ARR = ARRAY [ 1.. 10 ] OF INTEGER
        PROCEDURE P1 (X : ARR)
        VAR U : INTEGER
        BEGIN
           .
           .
           .
           U : = X [ 5 ];
           .
           .
           .
        END ;
```

Utilisation : P1 (A) si A est de type ARR

(dans ce cas, une expression ne peut être passée à une procédure : pas d'expressions de tableaux).

d) Passage par adresse :

Lors d'un passage par adresse, le mécanisme est le suivant : c'est l'adresse du paramètre qui est fournie à la procédure (ou fonction), et elle travaille donc directement sur la variable de l'appelant (et peut donc la modifier), on a alors comme **caractéristiques :**

1. au moment de l'appel, le paramètre doit être une variable, et en aucun cas une expression (car une expression n'a pas d'adresse statique).
2. c'est l'adresse qui est fournie à l'appel, et la procédure appelée travaille directement sur la variable de l'appelant.
3. la procédure appelée peut modifier la valeur du paramètre (puisqu'elle travaille directement sur la variable).
4. ce mode de passage est généralement utilisé pour des paramètres en sortie de la procédure appelée.
5. dans le cas où le paramètre est un tableau, ce mode de passage est économique, puisqu'il n'effectue pas de copie du tableau. Conseillé même pour un tableau en paramètre d'entrée.

Déclaration du paramètre

Un paramètre est passé par adresse lorsqu'il est précédé du mot-clé VAR.

Exemple :

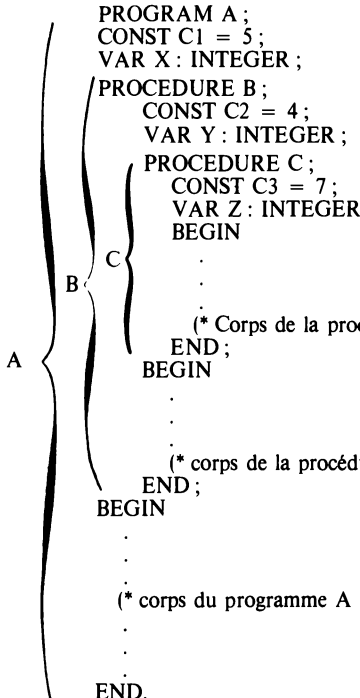
Déclaration : PROCEDURE P1 (VAR X : INTEGER) ;
BEGIN
.
.
.
X : = U (permis, car par adresse)
.
.
END ;

Utilisation : P1 (A)
(ici, P1 (A + 4) interdit).
L'adresse de A est passée à P1, et en travaillant sur X, la procédure P1 travaille en fait sur A.

e) Procédure ou fonction = Bloc

En Pascal, une procédure ou une fonction représente en fait un bloc fermé, où seuls sont connus de l'extérieur le nom de la procédure, ainsi que la nature de ses paramètres. Les autres identifieurs du bloc (constantes, variables, etc.) sont locaux et inconnus à l'extérieur.
La procédure ou la fonction a donc une structure semblable à celle du programme principal, sauf l'en-tête (PROCEDURE (ou FONCTION) au lieu de PROGRAM)
Une procédure peut donc avoir ses propres constantes, types variables et mêmes procédure internes qui seront inconnues de l'extérieur.

Exemple : (voir P16)



le programme A connaît : — la constante C1
— la variable X
— la procédure B
(et le programme A)

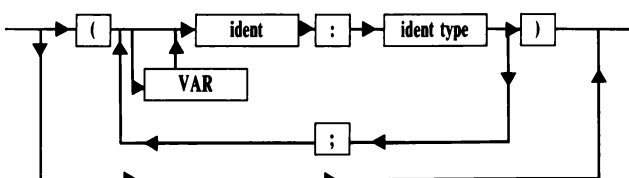
la procédure B connaît : — les constantes C1, C2
— les variables X, Y
— la procédure C (et A et B récursivement).

la procédure C connaît : **tout**

f) Syntaxe de la déclaration de procédure ou fonction.
procédure :

fonction :

liste parms :



g) Exemples de déclarations de procédures et fonctions

sans paramètres

— PROCEDURE P1 ;

ou

— FUNCTION P2 : INTEGER ;

avec paramètres

Si TAB = ARRAY [1.. 20] OF INTEGER ; (type)

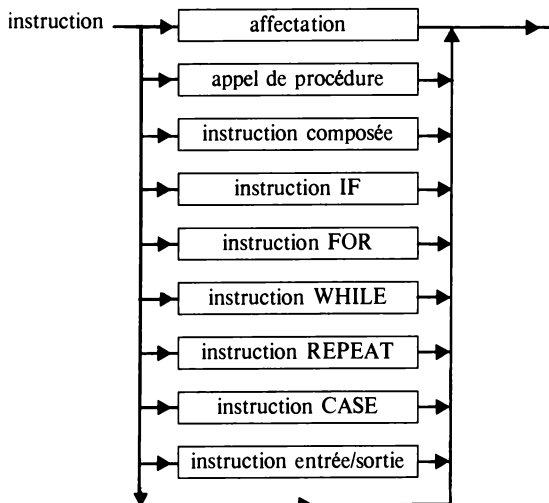
— PROCEDURE P1 (T : TAB ; VAR A, B : INTEGER) ;
ici T passé par valeur, A et B par adresse.

— FUNCTION P2 (VAR T, U : TAB ; A : INTEGER)
: INTEGER ;
ici T et U passés par adresses, A par valeur.

NOTE : attention au **U** dans FUNCTION

IV - INSTRUCTIONS PASCAL

Syntaxe

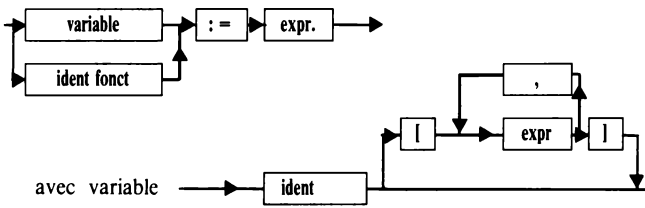


Une instruction est une séquence de code exécutable, conforme à une certaine syntaxe.
A noter :

- qu'un ensemble BEGIN... suite d'instructions... END est en lui-même une instruction au sens PASCAL.
- qu'une instruction peut être « vide » (intéressant dans certains cas).

IV.1 — L'INSTRUCTION D'AFFECTATION

a) Syntaxe



b) Fonctionnement

On affecte la valeur de l'expression à la variable figurant à gauche du ":="

Cette variable peut être :

— un identifieur déclaré en VAR ou comme paramètre d'une procédure ou fonction si la mode de passage est par adresse. Il peut être de type simple (INTEGER) ou élément de tableau ([...]) ou tableau (dans ce dernier cas, l'expression à droite doit se réduire à un identifieur de tableau de même type, ceci permettant de copier un tableau dans un autre).

— un nom de fonction : l'instruction permet alors d'affecter une valeur qui sera retournée par la fonction quand elle sera exécutée.

c) Equivalent BASIC

C'est l'instruction LET variable = expression, avec LET facultatif.
Attention : en Pascal, on utilise le symbole := au lieu de =

d) Exemples

— soient les variables A, B, C : INTEGER ;
X, Y : ARRAY [1.. 10] OF INTEGER
U : ARRAY [1.. 5, 1.. 2] OF INTEGER

les affectations suivantes sont valides :

```
A := B ;
X [ 5 ] := C ;
U [ 2, A + 1 ] := X [ 1 ] + C ;
X := Y ; (affectation de tableaux de même type)
```

celles-ci sont erronées :

```
X : = Y [ 1 ] ; (types non identiques : tableau et élément)
X : = U ; (tableaux de types différents, même si leur taille est
           identique)
P : = A ; (si P est un paramètre par valeur ; l'écriture serait
           correcte si P est un paramètre par adresse)
```

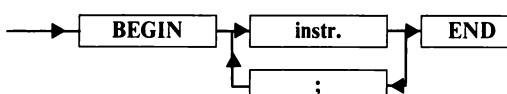
- cas d'une fonction :

```
FUNCTION F1 : INTEGER ;
BEGIN
   F1 : = 1 ; (affectation de 1 à la valeur de la fonction)
END ;
```

Si on écrit A : = F1 ; on récupérera la valeur 1 dans A

IV.2 — L'INSTRUCTION COMPOSEE BEGIN... END

a) Syntaxe



b) Fonctionnement

La structure BEGIN... END n'est que fonctionnelle, et ne produit aucune génération de code en elle-même (mais bien sûr, les instructions à l'intérieur du bloc en génèrent).
Son seul rôle est lexical, et permet de regrouper plusieurs instructions en une seule, afin de permettre au compilateur de savoir à quel ensemble appartient telle ou telle instruction.

Exemple :

— IF A = 1 THEN B : = 1 ; C : = 1 ;
(l'instruction C : = 1 ne **fait pas partie** du IF, et sera toujours exécutée (même si A<>1)).

— IF A = 1 THEN BEGIN B : = 1 ; C : = 1 END ;
(là, le compilateur considère BEGIN... END comme 1 instruction C : = 1 ne sera pas exécuté que si A = 1).

c) Equivalent BASIC

Notion inexistante en BASIC, l'équivalent est une suite quelconque d'instructions

d) Notes

— Le ";" étant un séparateur d'instrutions, il n'a pas normalement à figurer avant le END.
Toutefois, on peut en mettre un, et cela est même conseillé :
BEGIN A : = 1 ; B : = 2 ; END ;
La construction est valide (on a une instruction vide entre ; et END), et de plus si on ajoute une instruction C : = 3 avant le END, on ne risque pas d'"oublier" le ";" entre les deux instructions B : = 2 et C : = 3

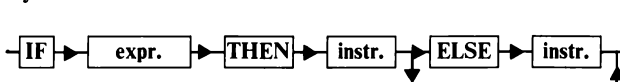
e) Exemples

— BEGIN END ;
(instruction nulle - intéressant dans certains cas)
— BEGIN A : = 1 ; B : = 2 ; END ;
— Imbrication

```
BEGIN
  A : = 1 ;
  BEGIN B : = 2 ; END ;
END ;
```

IV.3 — L'INSTRUCTION IF

a) Syntaxe



b) Equivalent BASIC

10 IF expr THEN suite instr ELSE suite instr.

c) notes.

— La partie "ELSE instr." est (comme en Basic) facultative.
Afin que le compilateur puisse reconnaître laquelle des deux formes est utilisée (avec ou sans ELSE), le mot-clé "ELSE" ne peut être précédé d'un ";"
La construction IF A = 1 THEN B : = A ; ELSE B : = 0 ; est erronée
— Dans le cas d'instructions IF imbriquées, un "ELSE" se rapporte au IF le plus proche
Ainsi : IF N >= 0 THEN
IF I = 1 THEN
R : = 1 ELSE R : = 2

R ne deviendra 1 que si N > = 0 **et** I = 1 (dernier IF)
R ne deviendra 2 que si N > = 0 **et** I <> 1 (dernier IF)
Pour avoir R : = 2 quand N < 0, on pourrait écrire

```
IF N > = 0 THEN
IF I = 1 THEN R : = 1
                ELSE BEGIN END (instr "vide")
ELSE R : = 2 ;
```

d) Exemples

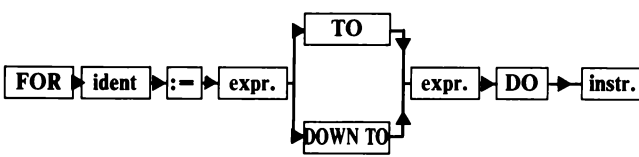
— IF N < 0 THEN N : = — N ; équivalent à
IF N > = 0 THEN ELSE N : = — N ;
(instruction "vide" sur le THEN)

— avec des instructions composées :

```
IF N = 0 THEN BEGIN S : = 1 ; E : = 2 END
          ELSE BEGIN S : = 2 ; E : = 1 END ;
```

IV.4 — L'INSTRUCTION FOR

a) Syntaxe



b) Equivalent BASIC

```
10 FOR ident = expr 1 TO expr 2 STEP      1     (TO)
                                         —1     (DOWN TO)

20 IF expr 1 > expr 2 (TO)                      THEN 110
      expr 2 > expr 1 (DOWNTO)

100 NEXT ident
110 —
```

c) Différences

— En PASCAL, deux "pas" seulement sont possibles : 1 ou —1
— En PASCAL, l'indice de boucle doit être
- de type INTEGER impérativement
- accessible directement (paramètre non utilisables)

— En PASCAL, la comparaison valeur initiale avec valeur finale est effectuée **avant** l'instruction ; l'effet possible est que la boucle ne soit jamais exécutée.

d) Exemples

— FOR I : = 1 TO 10 DO ;
(intruction "vide" réalisation d'une "temporisation
Basic :
FOR I = 1 TO 10 : NEXT I)

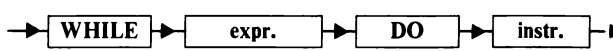
— FOR I : = 1 TO 10 DO A [I] : = 0 ;
(ici instruction simple A [I] : = 0)

— FOR I : = 1 TO 10 DO BEGIN A [I] : = 0 ; B [I] : = 1 END ;
(ici instruction composée BEGIN... END)

— FOR I : = 10 DOWNTO 1 DO
FOR J : = 1 TO 5 DO A [I,J] : = 0 ;
(boucles imbriquées - FOR étant une instruction).

IV.5 — L'INSTRUCTION WHILE

a) Syntaxe



b) Fonctionnement

Tant que l'expression est vraie, l'instruction (simple ou composée) est exécutée.
La boucle peut ne jamais être exécutée (cas où l'expression est fausse d'entrée).

c) Equivalent BASIC

```
10 IF NOT (expr) THEN 110
 .
 .
 .
100 GOTO 10
110
```

d) Exemples

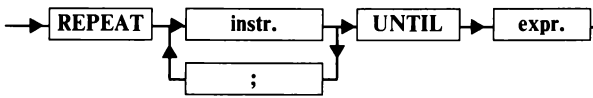
— WHILE 1 = 1 DO ;
(instruction vide et de plus, l'expression a peu de chance d'être fausse ! boucle jusqu'à la fin des temps)

— WHILE S < 12 DO S : = S + N

— WHILE S < 12 DO BEGIN S : = S + N ; i : = i + 1 END ;

IV.6 — L'INSTRUCTION REPEAT

a) Syntaxe



b) Fonctionnement

L'instruction (où la suite d'instructions) est effectuée jusqu'à ce que l'expression soit vraie.

La suite d'instructions est toujours exécutée au moins une fois.

c) Equivalent BASIC

```
10
.
.
.
100 IF expr THEN 10
```

d) Exemples

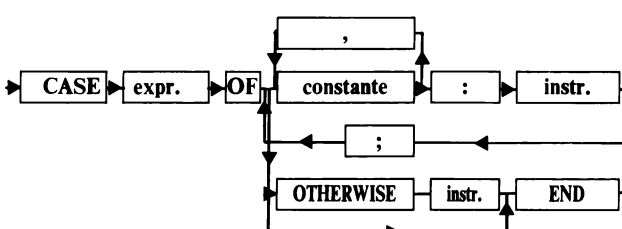
— REPEAT UNTIL 1 = 2 ;
(instruction vide et de plus, l'expression a peu de chances d'être vraie ! boucle jusqu'à la fin des temps).

— REPEAT S : = S + N UNTIL S > 12 ;

— REPEAT S : = S + N ; i : = i + 1 UNTIL S > 12

IV.7 — L'INSTRUCTION CASE

a) Syntaxe



b) Fonctionnement

L'instruction CASE permet de traiter plusieurs cas d'une même condition, en associant une instruction à une ou plusieurs valeurs d'une expression donnée, et ceci autant de fois que souhaité. Pour les valeurs de l'expression dont le cas n'a pas fait l'objet d'une instruction, deux cas peuvent se produire :

— l'instruction comporte une extension "OTHERWISE" l'instruction suivant la clause OTHERWISE sera exécutée.
— l'instruction ne comporte pas d'extension "OTHERWISE" sur ces valeurs, une erreur aura lieu à l'exécution.

c) Equivalent BASIC

Les instructions BASIC se rapportant le plus au CASE sont ON.. GOTO et ON.. GOSUB, avec les différences :

— en Basic, les valeurs doivent se suivrent de 1 à 1 car elles ne sont pas données explicitement.
— un "ajustement" doit être fait en Basic pour les valeurs l'

d) Exemples

```
CASE A OF                    10 ON A GOTO 20, 30, 40, 40
  1 : instr 1 ;                    30, 30, 30, 30, 30, 50
  3,4 : instr 2 ;            20 instr 1 : GOTO 60
 10 : instr 3 ;              30 instr 4 : GOTO 60 : REM Otherwise
 40 : instr 2 ;              GOTO 60
OTHERWISE instr 4 ;          50 inst 3
END ;                        60 ...

CASE A OF                    10 ON A + 4 GOTO 20, 30, 30
—3 : instr 1 ;                       30, 40
  2 : instr 2 ;              20 instr 1 : GOTO 50
END ;                        30 STOP : REM pas d'otherwise
                             40 instr 2
                             50 ...
```

IV.8 — L'APPEL DE PROCEDURES OU DE FONCTIONS

a) Syntaxe

b) Fonctionnement

L'appel de procédure et de fonction se fait en donnant le nom suivi d'une liste de paramètres entre parenthèses (si elle possède des paramètres).

Un appel de procédure se fait au niveau d'une instruction alors qu'un appel de fonction se fait à l'intérieur d'une expression (puisqu'elle retourne une valeur).

La liste de paramètres doit correspondre à la liste donnée lors de la déclaration de la procédure ou de la fonction.

1) en nombre : pas de paramètres en plus ou en moins.
2) en type : les paramètres lors de l'appel doivent être de même type que lors de la déclaration.
3) en mode de passage :
   — à un paramètre passé par valeur et de type simple (INTEGER) peut correspondre une expression.
   — à un paramètre passé par valeur mais de type tableau, ou à un paramètre passé par adresse doit correspondre un identifieur de même type.

c) Equivalent BASIC

— Procédures : équivalent au GOSUB, mais pas de paramétrage possible.
— Functions : équivalent au FN de certains BASICS mais avec des limitations en Basic :
   — nombre souvent limité de paramètres
   — passage de paramètres uniquement par valeurs.

d) Exemples

- **déclaration :** (si on a un type T10 : ARRAY [1.. 10] OF INTEGER

```
PROCEDURE P1 (A, B : INTEGER ; C : T10 ;
              VAR D : INTEGER ; VAR E : T10) ;
```

- **utilisation**

(si on a les variables X, Y, Z : INTEGER
T, U, V : T10)

l'appel peut-être

```
IF   X = 1 THEN
   P1 (X + Y — 2, V [ Z ] — X, T, Y, U) ;
```

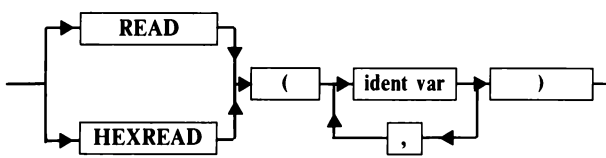
Les paramètres pour A et B pouvant être des expressions
Les paramètres pour C, D, E devant être des identifications de même type.

- Si P1 était une fonction, on aurait par exemple :
  IF P1 (X + Y — 2, V [Z] — X, T, Y, U)<>10 THEN...
- Si P1 n'avait pas de paramètres, donc déclaré :
  FUNCTION P1 : INTEGER ;
  On aurait :
  IF P1<>10 THEN...

IV.9 — PROCEDURES D'ENTREE/SORTIE

a) Procédures de lecture READ et HEXREAD

syntaxe :

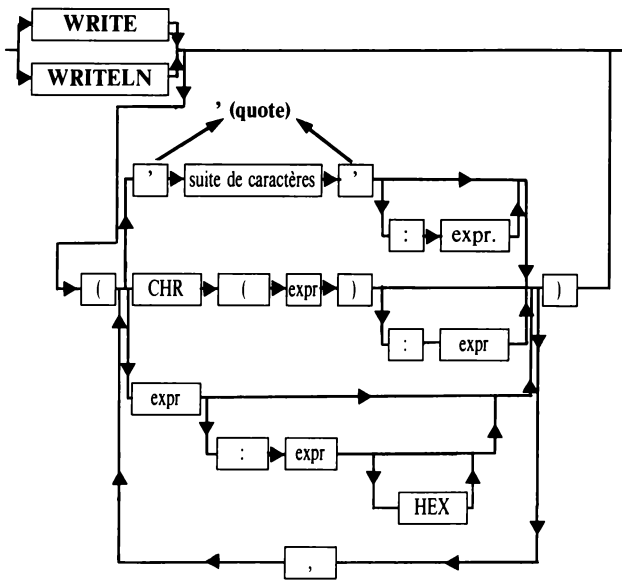


Ces procédures permettent de lire des valeurs et de les affecter aux variables de la liste.
READ lit des valeurs en décimal précédées ou non d'un signe + ou —
HEXREAD lit des valeurs en hexadécimal non signées
Au moment de l'exécution, on peut taper autant d'espaces et de retour chariot souhaités, puis la valeur, et la saisie s'arrête sur le premier espace ou retour-chariot tapé après la valeur.
Toute autre touche produit une erreur, à l'exclusion du caractère de correction. (touche ←—)

Exemples

avec A : INTEGER, T : ARRAY [1.. 10] OF INTEGER
READ (A, T [1], T [2]) ; pour lire en décimal ou HEXREAD (A) ; pour lire en hexadécimal.

b) Procédures d'écriture WRITE et WRITELN
Syntaxe



Ces procédures permettent d'écrire des informations sur l'écran. Elles sont similaires, sauf que WRITELN effectue un saut en début de ligne suivante après l'impression.
On peut écrire :
— des chaînes de caractères entre quotes :
WRITELN ('BONJOUR') ;
pour utiliser une quote dans la chaîne, il faut la doubler.
WRITELN ('L''OISEAU') ; écrira L'OISEAU.
On peut spécifier la longueur en caractères sur laquelle on veut écrire la chaîne par : nombre de caractères

La chaîne sera complétée par des espaces,
WRITE ('BONJOUR' : 10) écrira BONJOUR suivi de 3 espaces

— des caractères spéciaux par CHR (expression)
où l'expression est le code ASCII du caractère à imprimer
WRITE (CHR (12)) fera un effacement de page (code OCH), on peut comme pour les chaînes spécifier une longueur en caractères pour effectuer l'écriture.
ex : WRITE (CHR (65) : 10) écrira A suivi de 9 espaces.

— des valeurs numériques d'expression, sur un nombre de caractères spécifié, en décimal ou en hexadécimal. Si on ne précise pas le nombre de caractères pour écrire la valeur, par défaut 6 caractères seront pris en décimal.

En mode hexadécimal, le nombre de caractères est obligatoire et doit être suivi du mot-clé HEX.

Exemples :
WRITE (100) écrira 100 précédé de 3 espaces
WRITE (100 : 3) écrira 100 seul
WRITE (100 : 2 HEX) écrira 64
WRITE (100 : 4 HEX) écrira 0064
(en hexadécimal, les nombres sont précédés à gauche de 0 jusqu'à occurrence de 4 caractères, d'espaces ensuite)
WRITE (A) écrira la valeur de A sur 6 caractères.
Note : dans tous les cas où un nombre de caractères est précisé, ce nombre est pris module 256.
ainsi : 257 équivaut à : 1
: —1 équivaut à : 255

ANNEXE A
SYMBOLES PASCAL

' + — * [ ] ( ) # , ; : := . .. < <= <> > >= =

MOTS-CLES RESERVES

DO IF OF OR TO AND DIV END FOR MOD NOT VAR HEX CASE ELSE THEN TYPE ARRAY BEGIN CONST UNTIL WHILE DOWNTO REPEAT INTEGER PROGRAM FUNCTION OTHERWISE PROCEDURE.

MOTS PREDEFINIS

— Constante MAXINT
— Fonctions ABS CHR
— Procédures READ HEXREAD WRITE WRITELN.

Suivant les implémentations, d'autres mots prédéfinis peuvent exister. Voir alors le manuel d'utilisation.

ANNEXE B
MESSAGES D'ERREUR

1 - erreur dans type simple (.. attendu)
2 - identifieur attendu
3 - PROGRAM attendu
4 -) attendu
5 - : attendu
6 - symbole illégal
8 - OF attendu
9 - (attendu
11 - [attendu
12 -] attendu
13 - END attendu
14 - ; attendu
16 - = attendu
17 - BEGIN attendu
20 - , attendue
22 - . attendu
50 - erreur dans une constante.
51 - := attendu
52 - THEN attendu
53 - UNTIL attendu
54 - DO attendu
55 - TO ou DOWNTO attendu
101 - identifieur déclaré 2 fois
102 - borne inférieure supérieure à la borne supérieure.
103 - mauvaise classe d'identifieur.
104 - identifieur non déclaré.
105 - signe non permis.
120 - type de fonction erroné.
127 - substitution illégale de paramètre.
129 - conflit de types entre opérandes.

151 - assignement non autorisé.
182 - HEX non autorisé pour caractères ou chaînes.
183 - CHR interdit dans une expression.
203 - entier trop grand.
205 - chaîne nulle interdite.
207- hexadécimal trop grand.
208 - fin de chaîne après fin de ligne.
261 - erreur interne : table pleine.
262 - erreur interne : pile pleine.
Tout autre numéro d'erreur, surtout — 1, représente une erreur système.
Selon les implantations, d'autres cas d'erreurs peuvent être trouvés dans le manuel d'utilisation.

ANNEXE C
EXEMPLE DE PROGRAMME

Un exemple de programme écrit en Pascal **BASE** est donné ci-dessous. Ce programme ne se veut ni original, ni un modèle de technique de programmation, mais est destiné à montrer l'utilisation de pratiquement toutes les possibilités du Pascal.

```
PROGRAM C JOUR :
    (* programme de recherche du jour de la semaine correspondant
    à une date donnée *)
CONST
    AN1 = 1900 ; (* limite inférieure année de calcul *)
    AN2 = 2000 ; (* limite supérieure année de calcul *)
TYPE
    TTABLE = ARRAY [ 1.. 12 ] OF INTEGER : (* nombre de jours
    par mois *)
VAR
    JOUR, MOIS, ANNEE, CENTAN : INTEGER ; (* décomposi-
    tion date *)
    REP : INTEGER ; (* réponse *)
    TABJOUR : TTABLE ; (* table du nb jours/mois *)
PROCEDURE INIT (VAR T : TTABLE);
    (* initialisation de la table donnant le nombre de jours/mois *)
VAR I : INTEGER ;
```

```
BEGIN
   FOR I : = 1 TO 12 DO T [I] : = 31 ;
   T [4] : = 30 ; T [6] : = 30 ; T [9] : = 30 ; T [11] := 30 ;
END ;
PROCEDURE ERREUR ;
   (* affichage de message d'erreur *)
BEGIN WRITELN (*** ERREUR ***), WRITELN ; END ;
PROCEDURE GETDATE (VAR CENT, AN, M, J : INTEGER) ;
   (* obtention de la date *)
FUNCTION BISSEXTILE (C, A : INTEGER) : INTEGER ;
   (* recherche si année bissextile *)
BEGIN
   IF ((A < > 0) AND ((A MOD 4) = 0)) OR ((A = 0) AND ((C
   MOD 4) = 0))
      THEN BISSEXTILE : = —1 (* vrai *)
      ELSE BISSEXTILE : = 0 ; (* faux *)
END ;
PROCEDURE GETANNEE (VAR CENT, AN : INTEGER) ;
   (* obtention de l'année *)
VAR A : INTEGER ;
BEGIN
   WRITE ('année : '); READ (A) ;
   WHILE (A<AN1) OR (A>AN2) DO BEGIN ERREUR ;
   READ (A) ; END ;
   CENT : = A DIV 100 ; AN : = A MOD 100 ;
END ;
PROCEDURE GETMOIS (VAR M : INTEGER) ;
   (* obtention du mois *)
BEGIN
   WRITE ('mois (1 à 12) : ') ; READ (M) ;
   WHILE (M<1) OR (M>12) DO BEGIN ERREUR ; READ
   (M) ; END ;
END ;
PROCEDURE GETJOUR (VAR J : INTEGER ; M : INTEGER) ;
   (* obtention du jour *)
BEGIN
   WRITE ('jour : ') ; READ (J) ;
   WHILE (J<1) OR (J>TABJOUR [ M ]) DO BEGIN ERREUR ;
```

```
    READ (J) ; END ;
END ;
BEGIN (* de GETDATE *)
    GETANNEE (CENT, AN) ;
    IF BISSEXTILE (CENT, AN) THEN TABJOUR [2] : = 29
                             ELSE TABJOUR [2] : = 28 ;
                             (* jours de février *)
    GETMOIS (M) ;
    GETJOUR (J,M) ;
END ;
FUNCTION CALCUL J (C, A, M, J : INTEGER) : INTEGER ;
    (* calcul du jour dans la semaine *)
VAR MC, AC, CC : INTEGER ;
BEGIN
    (* pour le calcul, mars est le mois 1, janvier 11 et février 12 *)
    IF M < = 2 THEN BEGIN MC : = M + 10 ;
                          AC : = A — 1 END
               ELSE BEGIN MC : = M — 2 ; AC : = A END ;
    IF AC < 0 THEN BEGIN AC : = 99 ; CC : = C — 1 END
              ELSE CC : = C ;
    CALCULJ : = ((26*MC—2) DIV 10 + J + AC + AC DIV 4
                 + CC DIV 4 — 2*CC) MOD 7 ;
END ;
PROCEDURE AFJOUR (J : INTEGER) ;
    (* affichage du jour dans la semaine correspondant *)
BEGIN
    CASE J OF
      0 : WRITE ('DIMANCHE') ;
      1 : WRITE ('LUNDI') ;
      2 : WRITE ('MARDI') ;
      3 : WRITE ('MERCREDI') ;
      4 : WRITE ('JEUDI') ;
      5 : WRITE ('VENDREDI') ;
      6 : WRITE ('SAMEDI') ;
    END ;
END ;
BEGIN (* programme principal *)
    WRITELN ('... RECHERCHE D"UN JOUR DE SEMAINE...') ;
    WRITELN ('... A PARTIR D"UNE DATE DONNEE...') ;
```

```
    WRITELN ;
    INIT (TABJOUR) ;
    REPEAT
       GETDATE (CENTAN, ANNEE, MOIS, JOUR) ;
       WRITE ('LE ', JOUR : 1, '/', MOIS : 1, '/', CENTAN : 1,
       ANNEE :1, 'EST UN') ;
       AFJOUR (CALCUL J (CENTAN, ANNEE, MOIS, JOUR)) ;
       WRITELN ; WRITELN ;
       WRITE ('un autre calcul (0 = NON, 1 = OUI) : ') ;
       READ (REP) ;
       WHILE (REP <> 0) AND (REP<>1) DO BEGIN
       ERREUR ; READ (REP) ; END ;
    UNTIL REP = 0 ;
END.
```

Nota : Ce programme est intégré sur votre cassette système avec le nom de fichier : CJOUR

INDEX

Cet index regroupe tous les termes utilisés à la fois dans le manuel de programmation et le manuel d'utilisation. Le numéro de page est précédé de :

P→ pour le manuel de programmation
U→ pour le manuel d'utilisation

Symboles

ANNEXE

```
      (*******************************************)
      (***   ROUTINES SUPPLEMENTAIRES  --  MO5   ***)
      (*******************************************)

(* ensembles de routines a declarer en totalite ou en partie dans
   un programme pour acceder aux fonctions du moniteur *)

PROCEDURE COUT (C:INTEGER);
   (* envoi du caractere de code ASCII C sur l'ecran *)
   (* identique a un WRITE(CHR(C)) *)
BEGIN   ENCODE('EC583F02');   END;

FUNCTION KEYPRESS:INTEGER;
   (* test touche enfoncee   retourne le code ASCII de
        la touche ou 0 si pas de touche enfoncee *)
BEGIN   ENCODE('3F0C27034F20024F5FEDC4');   END;

FUNCTION CHIN INTEGER;
   (* attend la frappe d'un caractere et en retourne le code *)
BEGIN   ENCODE('3F0AC10027FA4FEDC4');   END;

PROCEDURE PSET (X,Y,COLR:INTEGER);
   (* allumage d'un point en X,Y de couleur COLR *)
   (* identique a l'instruction PSET graphique du BASIC *)
BEGIN   ENCODE('7F20363410AE5810AE56EC54F720293F103510');   END;

PROCEDURE LINE (X1,Y1,X2,Y2,COLR INTEGER);
   (* trace d'un segment de droite entre X1,Y1 et X2,Y2 *)
   (* si X1 ou Y1 est negatif, on repart du dernier point allume *)
BEGIN   ENCODE('7F20363410AE582B0C10AE562B07BF2032');
        ENCODE('108F2034AE5410AE52EC50F720293F0E3510');   END;

FUNCTION POINT (X,Y INTEGER):INTEGER;
   (* retourne la couleur du point en X,Y (idem BASIC) *)
BEGIN   ENCODE('3410AE5810AE563F141DEDC43510');   END;

FUNCTION FSCREEN(C,L INTEGER):INTEGER;
   (* retourne le code ASCII du caractere en colonne C, ligne L *)
BEGIN   ENCODE('3410AE5BEC561F9B3F1A4FEDC43510');   END;

PROCEDURE PLAY (NOTE,OCTAVE,DUREE,TEMPO,TIMBRE INTEGER);
   (* joue la note donnee : valeurs pour NOTE: 0=silence,1=DO,  *)
   (* 2=DO#,3=RE,4=RE#,5=MI,6=FA,7=FA#,8=SOL,9=SOL#,10=LA,  *)
   (* 11=LA#,12=SI   ---   autres valeurs idem BASIC *)
BEGIN   ENCODE('EC586620445A26FC87203FEC54FD203B');
        ENCODE('EC52FD2039EC50F7203DEC583F1E');   END;

PROCEDURE CONSOLE (HAUT,BAS INTEGER);
   (* definition de fenetre : limites haut et bas *)
   (* si un parametre est negatif, il reste inchange *)
BEGIN
  IF HAUT>=0 THEN WRITE (CHR(31),CHR(32+HAUT DIV 10),
                                 CHR(32+HAUT MOD 10));
  IF BAS>=0  THEN WRITE (CHR(31),CHR(16+BAS DIV 10),
                                 CHR(16+BAS MOD 10));
END;

PROCEDURE CLS;                    (* effacement ecran *)
BEGIN   WRITE (CHR(12));   END;

PROCEDURE CURSON;              (* affichage du curseur *)
```

```
BEGIN   WRITE (CHR(17));   END;

PROCEDURE CURSOFF;          (* extinction du curseur *)
BEGIN   WRITE (CHR(20));   END;

PROCEDURE LOCATE (C,L:INTEGER);
   (* positionnement curseur colonne C ligne L *)
BEGIN   WRITE (CHR(31),CHR(64+L),CHR(64+C));   END;

PROCEDURE POKE (ADR,VAL:INTEGER);
   (* ecriture de la valeur de VAL a l'adresse absolue ADR *)
   (* l'octet de poids fort de VAL est ignore *)
BEGIN   ENCODE('10AE58EC56E7A4');   END;

FUNCTION PEEK (ADR:INTEGER):INTEGER;
   (* lecture de l'octet se trouvant a l'adresse absolue ADR *)
   (* la valeur retournee est comprise entre 0 et 255 *)
BEGIN   ENCODE('10AE58E6A44FEDC4');   END;

PROCEDURE COLOR (FORME,FOND:INTEGER);
   (* definition des couleurs de forme et de fond pour les caracteres
      a venir - codage des couleurs identiques au BASIC *)
   (* si un parametre est negatif, il reste inchange *)
BEGIN     IF FORME>=0 THEN WRITE (CHR(27),CHR(64+FORME));
          IF FOND>=0  THEN WRITE (CHR(27),CHR(80+FOND));   END;

PROCEDURE SCREEN (FORME,FOND,TOUR:INTEGER);
   (* definition des couleurs de forme, fond et tour de l'ecran
      - codage des couleurs identiques au BASIC *)
   (* si un parametre est negatif, il reste inchange *)
BEGIN     IF FORME>=0 THEN WRITE (CHR(27),CHR(32),CHR(64+FORME));
          IF FOND>=0  THEN WRITE (CHR(27),CHR(32),CHR(80+FOND));
          IF TOUR>=0  THEN WRITE (CHR(27),CHR(96+TOUR));    END;

PROCEDURE POSCURS (VAR C,L:INTEGER);
   (* retourne les positions colonne et ligne du curseur dans les
      variables (passees par ADRESSE) C et L *)
BEGIN    ENCODE ('B6201BF6201C4CC0282EFB4ACB2834024F');
         ENCODE ('EDD8F83504EDD8F6'); END;

PROCEDURE ATTRB (HAUT,LARG:INTEGER);
   (* definition des attributs hauteur et largeur des caracteres
      0 si simple, 1 si double *)
BEGIN     IF ((HAUT=0) OR (HAUT=1)) AND ((LARG=0) OR (LARG=1)) THEN
            WRITE(CHR(27),CHR(112+2*LARG+HAUT));    END;

PROCEDURE SWIMPR;
   (* permet la redirection vers imprimante de tous les ordres
      WRITE et WRITELN qui suivront *)
BEGIN ENCODE ('6C88678604B720423F24B62043810426F2'); END;

PROCEDURE SWVIDEO;
   (* permet la redirection vers ecran de tous les ordres
      WRITE et WRITELN qui suivront *)
BEGIN ENCODE ('6F88678610B720423F24'); END;

FUNCTION RND : INTEGER;
   (* retourne un nombre aleatoire entre -32768 et 32767 *)
BEGIN ENCODE ('A688691F898401545454548900E688685889005858');
      ENCODE ('890044EC88685949ED8868EDC4'); END;
```

```
    (*************************************************************)
    (***   ROUTINES SUPPLEMENTAIRES  --  TO7 ext 16K   TO7-70   ***)
    (*************************************************************)

(* ensembles de routines a declarer en totalite ou en partie dans
   un programme pour acceder aux fonctions du moniteur *)

PROCEDURE COUT (C:INTEGER);
   (* envoi du caractere de code ASCII C sur l'ecran *)
   (* identique a un WRITE(CHR(C)) *)
BEGIN   ENCODE('EC58BDE803');   END;

FUNCTION KEYPRESS:INTEGER;
   (* test touche enfoncee   retourne le code ASCII de
      la touche ou 0 si pas de touche enfoncee *)
BEGIN   ENCODE('BDE80924034F20024F5FEDC4');   END;

FUNCTION CHIN:INTEGER;
   (* attend la frappe d'un caractere et en retourne le code *)
BEGIN   ENCODE('BDE806C10027F94FEDC4');   END;

PROCEDURE PSET (X,Y,COLR:INTEGER);
   (* allumage d'un point en X,Y de couleur COLR *)
   (* identique a l'instruction PSET graphique du BASIC *)
BEGIN   ENCODE('7F6041341 0AE5810AE56EC54F76038BDE80F3510');   END;

PROCEDURE LINE (X1,Y1,X2,Y2,COLR:INTEGER);
   (* trace d'un segment de droite entre X1,Y1 et X2,Y2 *)
   (* si X1 ou Y1 est negatif, on repart du dernier point allume *)
BEGIN   ENCODE('7F6041341 0AE582B0C10AE562B07BF603D');
        ENCODE('10BF603FAE5410AE92EC50F76038BDE80C3510');   END;

FUNCTION POINT (X,Y:INTEGER):INTEGER;
   (* retourne la couleur du point en X,Y (idem BASIC) *)
BEGIN   ENCODE('3410AE5810AE58BDE8211DEDC43510');   END;

FUNCTION FSCREEN(C,L:INTEGER):INTEGER;
   (* retourne le code ASCII du caractere en colonne C, ligne L *)
BEGIN   ENCODE('3410AE58EC561F9BBDE8244FEDC43510');   END;

PROCEDURE PLAY (NOTE,OCTAVE,DUREE,TEMPO,TIMBRE:INTEGER);
   (* joue la note donnee   valeurs pour NOTE  0=silence,1=DO,  *)
   (* 2=DO#,3=RE,4=RE#,5=MI,6=FA,7=FA#,8=SOL,9=SOL#,10=LA,  *)
   (* 11=LA#,12=SI   ---   autres valeurs idem BASIC *)
BEGIN   ENCODE('EC568620415A26FCB76037EC54FD6033');
        ENCODE('EC52FD6031EC50F76035EC58BDE81E');   END;

PROCEDURE CONSOLE (HAUT,BAS:INTEGER);
   (* definition de fenetre   limites haut et bas *)
   (* si un parametre est negatif, il reste inchange *)
BEGIN
  IF HAUT>=0 THEN WRITE (CHR(31),CHR(32+HAUT DIV 10),
                                 CHR(32+HAUT MOD 10));
  IF BAS>=0  THEN WRITE (CHR(31),CHR(16+BAS DIV 10),
                                 CHR(16+BAS MOD 10));
END;

PROCEDURE CLS;                    (* effacement ecran *)
BEGIN   WRITE (CHR(12));   END;

PROCEDURE CURSON;                 (* affichage du curseur *)
```

```
BEGIN   WRITE (CHR(17));   END;

PROCEDURE CURSOFF;          (* extinction du curseur *)
BEGIN   WRITE (CHR(20));   END;

PROCEDURE LOCATE (C,L:INTEGER);
   (* positionnement curseur colonne C ligne L *)
BEGIN   WRITE (CHR(31),CHR(64+L),CHR(64+C));   END;

PROCEDURE POKE (ADR,VAL:INTEGER);
   (* ecriture de la valeur de VAL a l'adresse absolue ADR *)
   (* l'octet de poids fort de VAL est ignore *)
BEGIN   ENCODE('10AE58EC56E7A4');   END;

FUNCTION PEEK (ADR:INTEGER):INTEGER;
   (* lecture de l'octet se trouvant a l'adresse absolue ADR *)
   (* la valeur retournee est comprise entre 0 et 255 *)
BEGIN   ENCODE('10AE58E6A44FEDC4');   END;

PROCEDURE COLOR (FORME,FOND:INTEGER);
   (* definition des couleurs de forme et de fond pour les caracteres
      a venir - codage des couleurs identiques au BASIC *)
   (* si un parametre est negatif, il reste inchange *)
BEGIN    IF FORME>=0 THEN WRITE (CHR(27),CHR(64+FORME));
         IF FOND>=0  THEN WRITE (CHR(27),CHR(80+FOND));   END;

PROCEDURE SCREEN (FORME,FOND,TOUR:INTEGER);
   (* definition des couleurs de forme, fond et tour de l'ecran
      - codage des couleurs identiques au BASIC *)
   (* si un parametre est negatif, il reste inchange *)
BEGIN    IF FORME>=0 THEN WRITE (CHR(27),'@ ',CHR(64+FORME));
         IF FOND>=0  THEN WRITE (CHR(27),'A ',CHR(80+FOND));
         IF TOUR>=0  THEN WRITE (CHR(27),CHR(96+TOUR));    END;

PROCEDURE POSCURS (VAR C,L:INTEGER);
   (* retourne les positions colonne et ligne du curseur dans les
      variables (passees par ADRESSE) C et L *)
BEGIN   ENCODE ('B6601BF6601C4CC0282EFB4ACB2834024F');
        ENCODE ('EDD8F83504EDD8F6'); END;

PROCEDURE ATTRB (HAUT,LARG:INTEGER);
   (* definition des attributs hauteur et largeur des caracteres
      0 si simple, 1 si double *)
BEGIN    IF ((HAUT=0) OR (HAUT=1)) AND ((LARG=0) OR (LARG=1)) THEN
           WRITE(CHR(27),CHR(76+2*LARG+HAUT));    END;

PROCEDURE SWIMPR;
   (* permet la redirection vers imprimante de tous les ordres
      WRITE et WRITELN qui suivront *)
BEGIN ENCODE ('6C8867B640B7602BBDE812B6602C814026F1'); END;

PROCEDURE SWVIDEO;
   (* permet la redirection vers ecran de tous les ordres
      WRITE et WRITELN qui suivront *)
BEGIN ENCODE ('6F8867B610B7602BBDEB12'), END;

FUNCTION RND   INTEGER;
   (* retourne un nombre aleatoire entre -32768 et 32767 *)
BEGIN ENCODE ('A688691FB9B401545454548900E688685889005858');
      ENCODE ('890044EC8868594ED8B68EDC4'); END;
```