UNIVERSITE LOUIS PASTEUR
I.R.E.M. de Strasbourg
10, rue du Général Zimmer
67084 STRASBOURG CEDEX

ANNEE 1981/82

Groupe Informatique

SUPPORT DU COURS D'INITIATION A

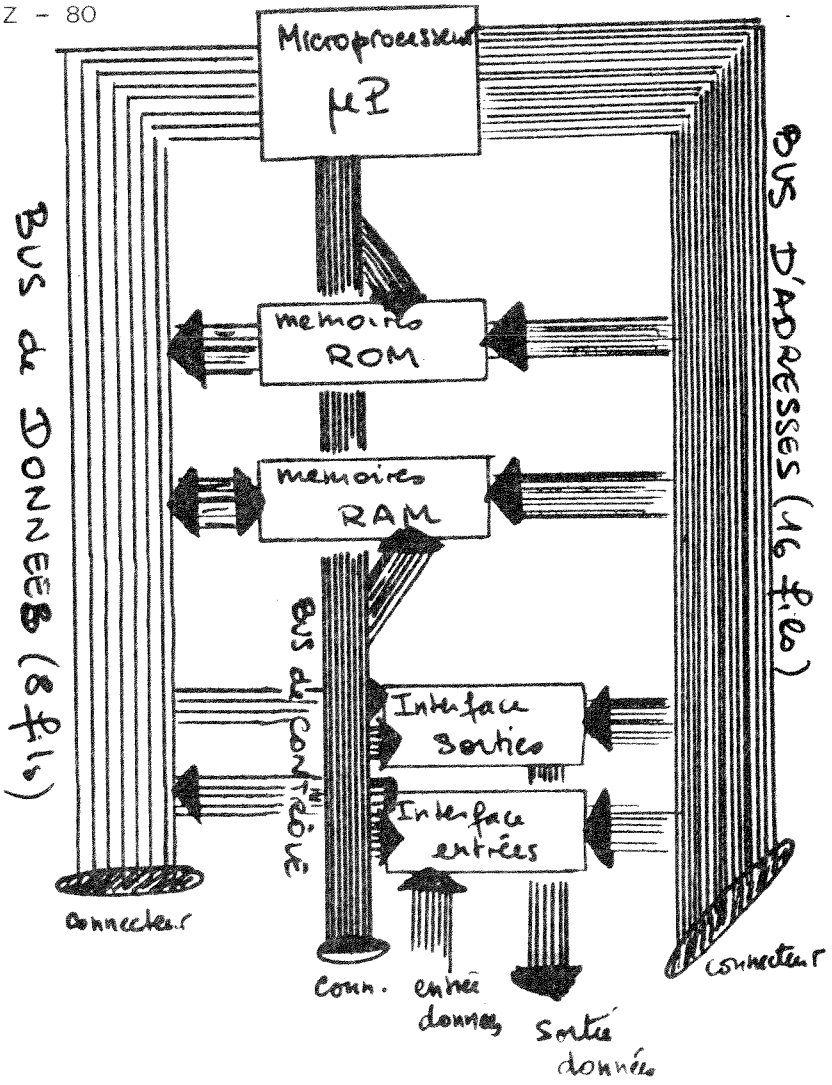
LA PROGRAMMATION

Ch. 1 - L'Ordinateur
Ch. 2 - Le Basic
Ch. 3 - L'Algorithmique

L'ORDINATEUR

Autour du microprocesseur

Exemple : le Z - 80



- Toute donnée qui entre ou sort du microprocesseur
  - . est stockée sous forme binaire sur 1 octet, elle transite dans le bus de données (8 fils)
  - . doit être associée à une adresse permettant de la ranger et de la retrouver. Cette adresse est stockée sur 16 bits ce qui fournit 2^{16} = 65536 adresses (64 K) distinctes possibles : bus d'adresses à 16 fils.
- Les commandes de sortie, d'entrée, d'arrêt, de relance du microprocesseur cons-

tituent elles-même des informations qui transitent par le bus de contrôle (13 fils).

- Les mémoires ROM (read only memory) ne peuvent qu'être lues. On s'en sert pour stocker les programmes de supervision de l'ordinateur, parfois les traducteurs de BASIC ou de LSE, que l'utilisateur risquerait sinon de détériorer par une écriture intempestive.

- Les mémoires RAM (random memory) sont accessibles en lecture et en écriture. Chaque emplacement (octet) a une adresse repérée par 2 octets, par exemple l'adresse $D134_{16}$.

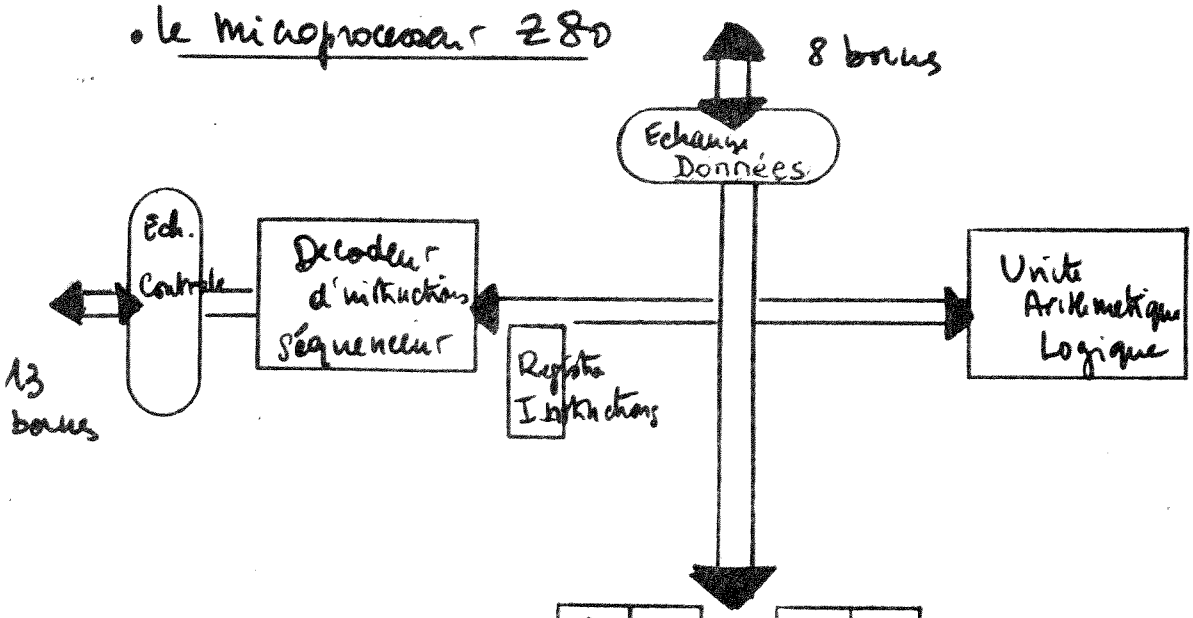
- Les interfaces permettent de gérer les différences de vitesse de fonctionnement d'un périphérique (disque, cassette, imprimante) et du microprocesseur . La donnée issue du microprocesseur est reçue par l'interface (dont l'adresse est connue). Celui-ci la conserve jusqu'à disponibilité du périphérique concerné, puis la transmet. A l'inverse, processus similaire pour une introduction de données.

En résumé :

toute opération effectuée par le microprocesseur comprend trois informations-type :

- un code opératoire précisant les commandes de contrôle à effectuer
- une ou plusieurs adresses
- une ou plusieurs données stockées à ces adresses

• le microprocesseur Z80

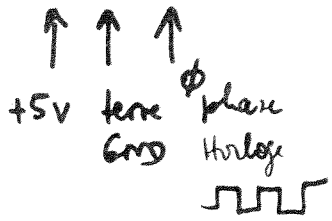


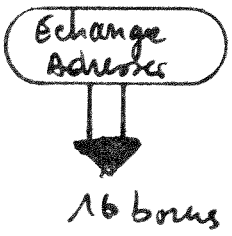
A	F
B	C
D	E
H	L

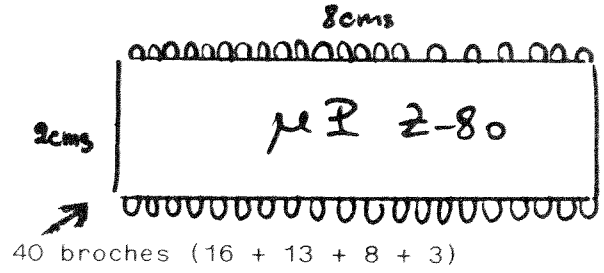
A'	F'
B'	C'
D'	E'
H'	L'

Registres opératoires de calcul

I	R
IX	index
IY	index
SP	stack pointer (pointeur de pile)
PC	progr. counter (compteur ordinal) (adresse d'instruction suivante)







le séquenceur est piloté par une horloge à 2,4 Mega Hertz (0,4 μs)
ou 4 Mega Hertz (0,25 μs)

‖ Synchronisant la succession des ordres élémentaires à exécuter.

- Les ordres élementaires (opérations fondamentales) sont :
  - recherche d'un code instruction et décodage
  - lecture d'un octet en mémoire ou sur un interface
  - écriture " " " " " " " "
  - reception d'une interruption ou relance
  - demande de BUS

  L'exécution d'un tel ordre nécessite 3 à 7 périodes d'horloge.

  Exemple :

  L'instruction permettant le transfert du nombre n dans le registre A (notée LD A, n) s'exécutera en 1,75 μ sec. (4 périodes pour sa recherche et son décodage, 3 périodes pour la lecture en mémoire du nombre n et son transfert dans le registre A.

- Les registres opératoires font partie du microprocesseur, ils peuvent contenir chacun un octet (A, B, C, ...) ou deux octets (AF, BC, DE, IX, PC, SP)

- PC joue un rôle particulier :

A chaque étape il contient l'adresse mémoire où est stockée l'instruction suivante

  au départ, il est à $\emptyset\emptyset\emptyset\emptyset_{16}$. Une instruction du type GOTO 2$\emptyset$ en BASIC se traduit par un chargement d'adresse (définie par 2$\emptyset$) dans le registre PC.

- Les Instructions machines

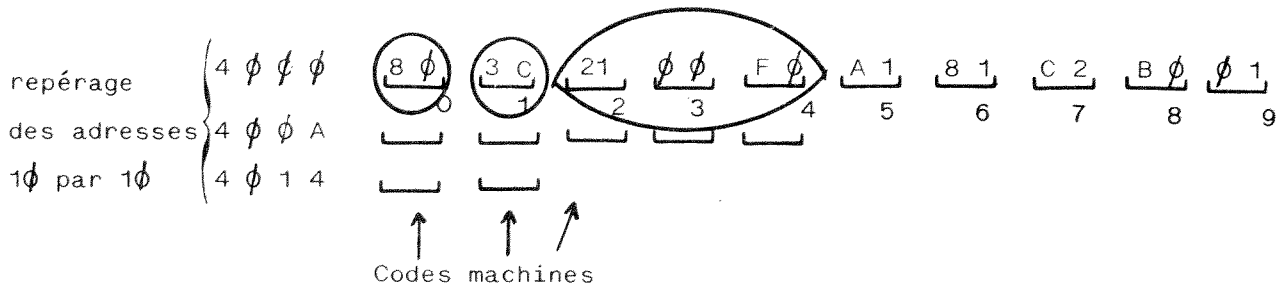
  Ce sont des codes reconnus par le microprocesseur comme succession d'opérations élémentaires à effectuer.
  Ils sont stockés sous forme binaire sur 1, 2 ou 3 octets à une adresse définie par le programmeur.

  Exemple :

  |1|$\emptyset$|$\emptyset$|$\emptyset$|$\emptyset$|$\emptyset$|$\emptyset$|$\emptyset$| ou |8|$\emptyset$|$_{16}$

  est un code machine signifiant l'addition du contenu du registre B au contenu du registre A, le registre A recevant le résultat.

• Un programme écrit en code machine se présente donc de la façon suivante :



4 ϕ ϕ ϕ 8 ϕ ADD A, B

4 ϕ ϕ 1 3 C INC A

4 ϕ ϕ 2 21 ϕϕ Fϕ LD HL, $F\phi\phi\phi_{16}$

4 ϕ ϕ 5

• Pour lancer le programme (RUN) on place l'adresse 4ϕϕϕ dans le registre PC et on relance l'ordinateur.

• Un jeu d'instructions indispensables

→ LD r,n — transfert du nombre n dans le registre r

ϕϕ [r] 11 ϕ [n]

ex : ϕϕ11111ϕϕϕϕ1ϕϕϕ ou 3E ϕ8

LD A,8

r	
A	111
B	ϕϕϕ
C	ϕϕ1
D	ϕ1ϕ
E	ϕ11
H	1ϕϕ
L	1ϕ1

→ ADD A,r — addition

1ϕϕϕ [r]

→ JP nn — saut à l'adresse nn (du registre PC)

11ϕϕϕ11 [n] [n] ou C3 nn

ex : JP $1A\phi\phi_{16}$ C3 ϕϕ 1A $_{16}$

→ CP r — compare le contenu du registre r avec celui du registre accumulateur A

1ϕ111 [r]

ex : CP B 1ϕ111ϕϕϕ ou B8 $_{16}$

→ AND r — intersection bit par bit des contenus de A et de r

1ϕ1ϕϕ [r]

ex : AND B (A ∧ B) s'écrit Aϕ $_{16}$

→ JP NZ,nn | saut conditionnel à l'adresse nn, s'écrit C2 n n /16

- Remarque : Les symboles LD, ADD, CP... représentent de façon plus commode pour le programmeur, les codes machines correspondant.
  La correspondance (jijective) entre symboles et codes machines définit le langage ASSEMBLEUR ou langage symbolique.

- On peut donc programmer en ASSEMBLEUR, un logiciel spécial se chargera de faire la correspondance avec les codes machine . (L'inverse existe parfois aussi : DESASSEMBLEUR)

→ Exemple de programme ASSEMBLEUR (Z-8Ø)

Adresse	Code machine	Assembleur
5EØØ	C5	PUSH BC
5EØ1	D5	PUSH DE
5EØ2	E5	PUSH HL
5EØ3	16ØØ	LD D,ØØ H* ← code caractère n° 1
5EØ5	21ØØDØ	LD HL, DØØØ H ← adresse début d'écran
5EØ8	Ø1E8Ø3	LD BC, Ø3E8 H
5EØB	72	LD (HL), D
5EØC	23	INC HL
5EØD	14	INC D
5EØE	ØB	DEC BC
5EØF	78	LD A,B
5E1Ø	B1	OR C
5E11	C2ØB5E	JP NZ, 5EØBH
5E14	E1	POP H1
5E15	D1	POP DE
5E16	C1	POP BC
5E17	C9	RET

H signifie Hexadécimal

Ce sous-programme a pour fonction d'afficher sur l'écran d'un Z-8Ø tous les caractères disponibles au générateur de caractères de cette machine, à raison de 1ØØØ caractères ($3E8_H$) (Adresse de début d'écran $DØØØ_H$ ou $DØØØ_{16}$)

- On voit l'hermétisme de ce langage et l'investissement que nécessitent des programmes réalisant même des fonctions relativement simples.

- On a donc très vite cherché à construire des langages plus évolués (plus proches de l'utilisateur).
  Ces langages sont ensuite traduits en code machine par un logiciel spécial appelé compilateur ou dans certaines circonstances interpréteur (traduction et exécution ligne par ligne en BASIC par exemple).

  Par exemple, l'écriture en BASIC de l'instruction :

```
20 .....
30 IF TT>0 THEN GOT0 20
```

  générera lors de l'exécution :

  - la recherche du contenu de la zône d'adresse symbolique TT
  - sa comparaison à 0
  - si le résultat est "oui" un saut du compteur ordinal à l'adresse de la ligne 20.

  On peut donc compter de 6 à 8 instructions assembleur (ou machine) pour effectuer l'instruction de la ligne 30 du programme BASIC.

  Un programme écrit dans un langage évolué sera donc

  - plus long qu'un programme-machine (à cause des opérations de traduction), en durée.
  - plus coûteux en mémoire, il faudra conserver le traducteur et des tables de symboles utilisés.

- Les principaux langages évolués utilisés actuellement (et dont le traducteur est fourni avec la machine) sont :

→ le BASIC — langage conçu pour l'initiation à l'informatique et la manipulation de petits ordinateurs

→ le COBOL — Common Business operating Language
langage de gestion de fichiers commode pour les usages de gestion

→ le FORTRAN — Formula Translation
langage commode pour les calculs et les opérations sur matrices - utilisé par les chercheurs et ingénieurs

→ le PL 1 combinaison de FORTRAN et COBOL

→ le PASCAL langage plus récent permettant la structuration de programmes (voir ALGORITHMIQUE)

→ le LSE langage symbolique d'enseignement crée pour l'enseignement dans les lycées.

• POUR CES LANGAGES

→ On dispose d'un logiciel d'édition de texte en langage-source

→ d'un traducteur (compilateur ou interpréteur)

On conserve le programme-source
le programme-objet résultant de la traduction

Nous étudierons ici
- Le BASIC sur APPLE II et LX 410
- Le LSE sur LX410

Table des codes ASCII

Appendix K: ASCII Character Codes

DEC = ASCII decimal code
HEX = ASCII hexadecimal code
CHAR = ASCII character name
n/a = not accessible directly from the APPLE II keyboard

DEC	HEX	CHAR	WHAT TO TYPE	DEC	HEX	CHAR	WHAT TO TYPE	DEC	HEX	CHAR	WHAT TO TYPE
0	00	NULL	ctrl @	32	20	SPACE	space	64	40	@	@
1	01	SOH	ctrl A	33	21	!	!	65	41	A	A
2	02	STX	ctrl B	34	22	"	"	66	42	B	B
3	03	ETX	ctrl C	35	23	#	#	67	43	C	C
4	04	ET	ctrl D	36	24	$	$	68	44	D	D
5	05	ENQ	ctrl E	37	25	%	%	69	45	E	E
6	06	ACK	ctrl F	38	26	&	&	70	46	F	F
7	07	BEL	ctrl G	39	27	'	'	71	47	G	G
8	08	BS	ctrl H or ←	40	28	(	(	72	48	H	H
9	09	HT	ctrl I	41	29	)	)	73	49	I	I
10	0A	LF	ctrl J	42	2A	*	*	74	4A	J	J
11	0B	VT	ctrl K	43	2B	+	+	75	4B	K	K
12	0C	FF	ctrl L	44	2C	,	,	76	4C	L	L
13	0D	CR	ctrl M or RETURN	45	2D	-	-	77	4D	M	M
14	0E	SO	ctrl N	46	2E	.	.	78	4E	N	N
15	0F	SI	ctrl O	47	2F	/	/	79	4F	O	O
16	10	DLE	ctrl P	48	30	0	0	80	50	P	P
17	11	DC1	ctrl Q	49	31	1	1	81	51	Q	Q
18	12	DC2	ctrl R	50	32	2	2	82	52	R	R
19	13	DC3	ctrl S	51	33	3	3	83	53	S	S
20	14	DC4	ctrl T	52	34	4	4	84	54	T	T
21	15	NAK	ctrl U or →	53	35	5	5	85	55	U	U
22	16	SYN	ctrl V	54	36	6	6	86	56	V	V
23	17	ETB	ctrl W	55	37	7	7	87	57	W	W
24	18	CAN	ctrl X	56	38	8	8	88	58	X	X
25	19	EM	ctrl Y	57	39	9	9	89	59	Y	Y
26	1A	SUB	ctrl Z	58	3A	:	:	90	5A	Z	Z
27	1B	ESCAPE	ESC	59	3B	;	;	91	5B	[	n/a
28	1C	FS	n/a	60	3C	<	<	92	5C	\	n/a
29	1D	GS	ctrl shift-M	61	3D	=	=	93	5D	]	] (shift-M)
30	1E	RS	ctrl ^	62	3E	>	>	94	5E	^	^
31	1F	US	n/a	63	3F	?	?	95	5F	_	n/a

ASCII codes in the range 96 through 255 will generate characters on the APPLE which repeat those in the list above (first those in column 2, then the entire series again). Although CHR$(65) returns an A and CHR$(193) also returns an A, APPLESOFT does not recognize the two as the same character when using string logical operators, and a printer connected to your APPLE would print them differently.

BEGINNER'S ALL-PURPOSE SYMBOLIC INSTRUCTION CODE

```
1Ø      REM CALCUL DES MOYENNES
2Ø      S = Ø : SC = Ø
3Ø      INPUT "NOMBRE DE NOTES" ; N
4Ø      FOR I = 1 TO N
5Ø      INPUT "NOTE" ; NT
6Ø      INPUT "COEFFICIENT" ; CF
7Ø      SC = SC + CF : S = S + NT * CF
8Ø      NEXT I
9Ø      S = S/SC
1ØØ     PRINT "MOYENNE PONDEREE" ;
11Ø      END
```

Un programme écrit en BASIC

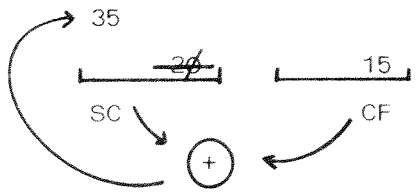
Ce programme a pour objet le calcul de la moyenne pondérée de N notes, introduites successivement avec leur coefficient de pondération. On y trouve, comme dans tout programme :

- des instructions demandant l'introduction des données (au clavier)

  INPUT N, NT, CF
- des instructions déclenchant l'édition de données (à l'écran)

  PRINT S
- des instructions de calcul utilisant les opérateurs arithmétiques
- l'instruction d'affectation notée = à ne pas confondre avec le signe d'égalité mathématique
- des variables, sur lesquelles opèrent ces instructions et qui sont en réalité des zones de la mémoire d'ordinateur repérées par des symboles qui sont les noms donnés à ces variables.

Par exemple : 7Ø SC = SC + CF se comprend ainsi : le contenu de la zone repérée par le symbole SC est additionné à celui de la zone repérée par CF. Le résultat est affecté à la zone repérée par SC (dont l'ancien contenu est alors "écrasé"



- L'ordre dans lequel ces instructions sont exécutées est celui induit pas les numéros de ligne. Il peut y avoir plusieurs instructions par ligne, un séparateur adéquat (: ou \) permet de les délimiter).

- Des commentaires peuvent être insérés
  - → dans le programme (instruction 1Ø)
  - → lors de l'exécution, à l'écran
    ex : PRINT "N =" ; N provoque l'édition du commentaire N = , suivi du contenu actuel de la variable appelée (ici N)

- L'instruction 5 + 8 = ne suffit pas pour obtenir la somme comme cela se passe sur une calculette. Le résultat est en mémoire mais ne s'affichera que par l'instruction PRINT 5 +8 (Ne pas oublier les instructions d'édition dans un programme faute de quoi celui-ci s'éxécutera bien mais l'écran restera vide).

- Remarquons la séquence 5Ø - 7Ø de lecture des données et de cumul, qui est répétée N fois grâce aux instructions de contrôle 4Ø FOR I = 1 TO N
  et 8Ø NEXT I
  l'entier N est lui-même introduit au clavier en début d'exécution du programme et pourra donc changer d'une exécution à une autre.

- Après avoir écrit le programme ligne par ligne :
  - → on l'exécute par la commande RUN
  - → on le liste par la commande LIST
    ou LIST 1Ø - 5Ø
    ou LIST - 7Ø
  - → on le sauvegarde sur un support externe par la commande SAVE suivie d'un nom permettant de l'identifier.

Symboles et opérateurs

- Les noms de VARIABLES sont formés de une ou deux lettres de l'alphabet.

  Ils se terminent par :

  (%) si le contenu de la variable (zone-mémoire) est un nombre entier (valeur absolue inférieure à 32768)

  ($) si le contenu est alphanumérique (maximum 255 caractères). chaine de caractères.

  ( ) si le contenu est réel (valeur absolue inférieure à 10^{38}).

ex : 15.568 → C ; 15 → C% ; TOTO1 → C$; 0.568 E 12 → D

- Les opérateurs arithmétiques +, -, *, /, ↑ (ou ∧)

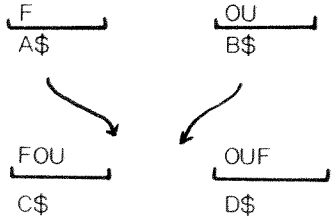
ex : C = A + 3.5 G = A↑B H = (A-B) * (C+D/E)

D = A * B A = A-B

- L'opérateur de concaténation de chaînes de caractères

C$ = A$ + B$

D$ = B$ + A$



- Les opérateurs logiques et de relation <, >, <=, >=, <>

AND, OR, (NOT)

A<B A>C A<=B

E<>G (A>B) AND (C<=D) OR (E<>G)

ex : 10 IF (A<B) AND (C>10) THEN PRINT A

ne pas confondre O et Ø (zéro)

● L'ordre PRINT peut être utilisé de différentes manières :

	Affiche sur l'écran
PRINT 5 + 8	13
? A	le contenu de A (numérique)
PRINT "A"	la lettre A
PRINT A$	le contenu de A$ (alphabétique)
PRINT A, B	les contenus de A et de B sur une même ligne, cadrés par zônes de 16 caractères
PRINT A ; B	les contenus de A et de B sur une même ligne sans espace
PRINT A$; B$	les contenus de A$ et de B$ chaînes de caractères, concaténés
PRINT TAB(2Ø) ; A	le contenu de A cadré à partir de la 20e position dans la ligne
PRINT SPC (5)	5 blancs
PRINT	Saute une ligne
HOME	Vide l'écran

● Exemples :

```
10 PRINT 'BONJOUR ,COMMENT CA VA'
20 PRINT 'CA VA BIEN,MERCI'
30 PRINT 'QUOI DE NEUF'
40 END
RUN

BONJOUR ,COMMENT CA VA?
CA VA BIEN,MERCI
QUOI DE NEUF?
```

```
10 PRINT 'VOICI','CE QUE PROVOQUERAIT',
20 PRINT 'UNE','UTILISATION','DE LA ',
30 PRINT 'VIRGULE.'
40 END

'VOICI          CE QUE PROVOQUERAIT        UNE
UTILISATION     DE LA
VIRGULE.
```

```
 5 PRINT
10 PRINT 'A';'BB';
30 PRINT 'CCC';
40 PRINT 'DDDD';'EEEEE';'FFFFF'
50 END

ABBCCCDDDDEEEEEFFFFF
```

```
 5 PRINT
10 PRINT 'A';'   ';'BB';
20 PRINT '   ';'CCC';'   ';'DDDD';
30 PRINT '   ';'EEEEEE'
40 END
```

```
10 REM TABLE DES CARRES DES ENTIERS DE 1 A 50

20 HOME : PRINT : PRINT : PRINT

30 PRINT TAB(10) ; "TABLE DES CARRES"

40 PRINT TAB(10) ; "****************" : PRINT

50 FOR I = 1 TO 50 : PRINT I*I ; SPC(3) ; : NEXT I

60 END

RUN
```

```
                      TABLE DES CARRES
                      ****************

      1     4     9        16    25     36      49      64...
       \___/ \___/          \___/
         3     3              3
                                                  .../2500
```

- Une édition plus soignée s'obtient en cadrant par zones de 5 caractères en veillant à l'alignement vertical :

```
50 FOR I = 1 TO 50 : PRINT TAB(5*(I-1)+1) ; I*I ;
60 NEXT I
70 END
```

L'écran comprend 40 colonnes et 25 lignes (ou 80 colonnes et 50 lignes).
L'instruction TAB(5*(I-1)+1) ; permet le cadrage de 5 en 5 et le saut à la ligne pour I = 9, 17, 25, 33, 41, 49

- La séquence suivante crée un tableau de 5 colonnes et 10 lignes

```
50 FOR NL = 1 TO 10 : REM NL NUMERO DE LIGNE
60 FOR NC = 1 TO 5  : REM NC NUMERO DE COLONNE
70 K = (NL-1)*5+NC : REM RANG DE L'ELEMENT AFFICHÉ
80 PRINT TAB(5*NC) ; K*K ;
90 NEXT NC
100 PRINT : NEXT NL
110 END
```

- ATTENTION → L'ordre 100 PRINT permet le passage à la ligne suivante (empêché par les ; précédents)
  → La séquence 60 - 100 est répétée 10 fois (sous le contrôle de NL)
  → La séquence 70 - 80 est répétée 5 fois (sous le contrôle de NC) et ce pour chaque valeur de NL, soit 50 fois en tout.

● entrée par affectation directe de variables

exemple :

```
 10 PRINT
 20 PRINT '**CE PROGRAMME CALCULE**'
 30 PRINT 'LA SOMME DE DEUX NOMBRES'
 35 PRINT '************************'
 40 A=15
 50 B=5
 60 S=A+B
 70 PRINT 'LA SOMME=';S
 80 END
RUN

**CE PROGRAMME CALCULE**
LA SOMME DE DEUX NOMBRES
************************
LA SOMME= 20
```

. le nombre 15 est affecté par programme à A

. il faut modifier le programme pour changer les données traitées

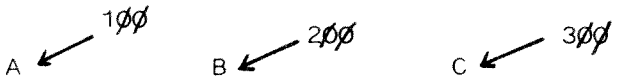
● entrée par lecture d'une table pré introduite

exemple :

```
 10 DATA 100,200,300,450,500
 20 READ A,B,C
 30 S=(A+B+C)
 40 D=(A+B)-C
 45 PRINT
 50 PRINT '**CE PROGRAMME CALCULE**'
 60 PRINT 'LA SOMME ET LA DIFFERENCE'
 70 PRINT 'DE TROIS (3) NOMBRES.'
 80 PRINT '--- ------------------'
100 PRINT 'SOMME=';S;'   ';'DIFFERENCE=';D
110 END
RUN

**CE PROGRAMME CALCULE**
LA SOMME ET LA DIFFERENCE
DE TROIS (3) NOMBRES.
--- ------------------
SOMME= 600    DIFFERENCE= 0
```

. la table DATA 1ØØ, 2ØØ, 3ØØ est consultée à chaque occurence de l'instruction READ



. un pointeur se déplace dans la table à mesure de l'association des données aux variables

. une même table peut être consultée plusieurs fois. Le pointeur évolue à mesure. Si l'on veut se repositionner au début de la table il faut insérer un ordre RESTORE

Ex : 11Ø READ D, E
12Ø RESTORE
13Ø READ U, V, W, X, Y
14Ø D = U + V + W - X - Y
15Ø PRINT D

en 11Ø D et E reçoivent les valeurs 45Ø et 5ØØ puis le pointeur de table est repositionné à 1ØØ, en 15Ø D vaut - 35Ø.

entrée des données (suite) INPUT

- entrée interactive au clavier INPUT

exemple :

```
10 INPUT A,B
20 P=A*B
30 PRINT 'LE PRODUIT EST:';P
40 END
RUN

? 10,20
LE PRODUIT EST: 200
```

. lors de l'exécution de 1Ø un point d'interrogation s'affiche à l'écran

. il faut fournir autant de valeurs que de variables figurant dans l'ordre INPUT, en les séparant par des virgules (ou par des retours de chariot)

. on peut provoquer l'affichage d'un commentaire explicatif lors de l'éxécution de INPUT

Ex : 1Ø INPUT "VALEUR DE N" ; N

RUN

VALEUR DE N ?

(Revoir l'exemple de la fiche initiale BASIC)

● L'existence de numéros de ligne permet de modifier le déroulement normal du programme en effectuant des sauts (ou branchements) conditionnels ou inconditionnels

- Sauts inconditionnels

→ 5ϕ GOTO 1ϕϕ provoque le saut à ligne 1ϕϕ puis continuation en séquence normale. Cette instruction est à utiliser avec prudence. Les professionnels de l'informatique la déconseillent (sauf impossibilité d'y échapper) du fait des difficultés pour un autre programmeur (ou pour soi-même) de retrouver le fil directeur et la logique du programme qui, de saut en saut, risque, comme le serpent, de se mordre la queue - on dit que le programme "boucle" - ; voir la fiche sur la structuration des programmes.

ex : 1ϕ I = ϕ
2ϕ I = I + 1 : PRINT SQR(I) : GOTO 2ϕ
3ϕ END

Ce programme tournera longtemps, s'il n'est pas interrompu de force par un BREAK
Un usage du GOTO est le suivant (jeux)

1ϕ REM DEBUT DU JEU

9ϕ INPUT "VOULEZ-VOUS REJOUER (O/N)" ; A$
1ϕϕ IF A$ = "O" THEN GOTO 1ϕ

le jeu est relancé sans passer par RUN

```
60 GOSUB 1000          1000
70                     15000 RETURN
```

équivaut à l'appel d'un sous-programme (procédure) avec retour en 70 après rencontre de l'instruction 15000.

Procédé intéressant
- pour regrouper une séquence d'instruction à laquelle on fera appel plusieurs fois au cours du même programme
- pour isoler un paragraphe d'instructions particulièrement volumineux (éditions par exemple) de manière à éclaircir la présentation du programme principal.

Exemple :

```
10 REM  JEU DU TURLUTUTU
20 GOSUB 1000
30
100 END

1000 REM EDITION DE LA REGLE DU JEU
1010 PRINT

3000 RETURN
```

Attention: Placer les séquences appelées par un GOSUB après l'instruction END, sinon elles seront exécutées lors du passage en séquence normale et la rencontre d'une instruction RETURN sans qu'il y ait eu d'appel par GOSUB provoque une erreur.

Exemple :

```
1ϕ REM TRI DE TROIS ZONES NUMERIQUES
2ϕ INPUT A, B, C
3ϕ IF A>B THEN  R = A : A = B : B = R
4ϕ IF A>C THEN  R = A : A = C : C = R
5ϕ IF B>C THEN  R = B : B = C : C = R
6ϕ PRINT "NOMBRES TRIES" ; A, B, C
7ϕ END
```

- si la condition est vraie la séquence suivant THEN est exécutée, puis passage à la ligne suivante, sinon passage à la ligne suivante

- dans le cas où il y a des séquences différentes effectives à exécuter selon que la condition est vraie ou fausse, on dispose selon les ordinateurs des deux possibilités suivantes :

```
→ 3ϕ IF A>B THEN  seq 1   ELSE  seq 2
  4ϕ suite
```

```
→ 3ϕ IF A>B THEN  seq 1  : GOTO  5ϕ
  4ϕ seq 2                          attention !
  5ϕ suite
```

que l'on pourra encore écrire

```
  3ϕ IF cond THEN  GOSUB 1ϕϕϕ  ELSE  GOSUB 2ϕϕϕ
  4ϕ suite
```

ou

```
  3ϕ IF cond  THEN  GOTO 1ϕϕϕ
  4ϕ GOSUB 2ϕϕϕ
  5ϕ
```

```
1ϕϕϕ  seq 1                    2ϕϕϕ  seq 2
      ———                            ———
      -
11ϕϕ  GOTO 5ϕ                  21ϕϕ  RETURN
```

- on voit qu'il convient, dans la mesure du possible, de rédiger la condition de telle manière que le programme continue en séquence normale si elle n'est pas vérifiée.

- on peut emboîter plusieurs tests :

```
3ϕ IF  cond 1  THEN  IF cond 2  THEN  seq 1
4ϕ seq 2
```

Instructions de contrôle (suite)

Attention ! • Ici la séq 1 sera exécutée si cond 2 et cond 1 sont vérifiées
à utiliser avec prudence, voir structuration des programmes.
Il est souvent plus commode d'utiliser les opérateurs logiques :

```
3Ø IF (cond 1) AND (cond 2) THEN  seq 1
4Ø seq 2
```

quitte à refaire un test sur l'une des deux conditions :
écrire par exemple un programme permettant d'éditer une liste de noms, 3 par 3, quelle que soit la taille de cette liste.

• La seq 2 sera exécutée dans tous les cas.
Elle sera exécutée seule si cond 2 ou cond 1 non vérifiée.

```
1Ø REM TABLE DE MULTIPLICATION
2Ø FOR X = 1 TO 9
3Ø FOR Y = 1 TO 9
4Ø PRINT X*Y ;" ";
5Ø NEXT Y
6Ø PRINT
7Ø NEXT X
8Ø END
```

ligne X

rupture de ligne →

RUN

1	2	3	4	.....	9
2	4	6	8	.....	18
3	6	9	12	.....	⋮
⋮	⋮	⋮	⋮		⋮
9	18	27	36	.....	81

- Ce programme comprend deux boucles emboîtées qui répètent l'instruction 4Ø neuf fois pour chaque valeur de X qui varie lui-même de 1 à 9. Deux boucles ne peuvent pas se chevaucher.

- L'évolution de l'indice de boucle peut être contrôlée par STEP

par exemple :

```
FOR X = 9 TO 1 STEP -1
ou  FOR X = Ø TO 1Ø STEP 2
    FOR X = Ø TO 1 STEP Ø.Ø1
```

Application au calcul sur tableaux indicés.

ex :

```
1Ø REM CALCUL MATRICIEL (STATISTIQUES)
2Ø INPUT "NOMBRE DE LIGNES" ; M
3Ø INPUT "NOMBRE DE COLONNES" ;  N
4Ø DIM A(M,N), S(M), T(N)
5Ø FOR I = 1 TO M : FOR J = 1 TO N
6Ø INPUT A(I,J)
7Ø S(I) = S(I) + A(I,J) : T(J) = T(J) + A(I,J)
8Ø NEXT J,I
9Ø
   } éditions
```

→ Ce programme lit une suite de nombres et les range dans des zones repérées par un indice de ligne I et de colonne J. En même temps sont calculés et stockés les cumuls de ligne S et les cumuls de colonne T.

Instructions de contrôle (suite et fin)

⟶ Ce programme nécessite la préparation de zones indicées A(I,J), S(I), T(J). Ceci se fait en début de programme par une instruction de dimensionnement.

DIM A(M, N) , S(M) , T(N)

⟶ On peut aussi fixer des dimensions maximales constantes :

DIM A(5Ø, 1ØØ)

⟶ A noter que l'on dispose de l'indice Ø. Par suite DIM A(2Ø) réserve 21 zones A(Ø), A(1), A(2), ..., A(2Ø)

- A noter la possibilité d'introduire une interruption dans un programme en insérant

5Ø STOP

6Ø

L'écran affichera BREAK AT 5Ø et attendra une relance par la frappe de CONT (inue). On peut ainsi recopier des résultats, visionner un graphique etc...

- END indique la fin du programme.

Exemple :

```
10 REM RETOURNEMENT DE VOTRE NOM
20 INPUT "TAPEZ VOTRE NOM" ; N$
30 R$ = ""
40 FOR T = LEN(N$) TO 1 STEP -1
50 R$ = R$ + MID$ (N$, T, 1)
60 NEXT T
70 PRINT : PRINT "VOTRE NOM RETOURNE EST : " ; R$
80 PRINT : PRINT
90 INPUT "AUTRE NOM" ; O$
100 IF O$ = "O" THEN 20
110 END
```

- Ce programme introduit une chaîne de caractères N$. Cette chaîne est balayée de droite à gauche et le Te caractère est concaténé à droite dans R$ préalablement mise à vide.
  - → Rappel: les noms de variables alphabétiques se terminent par un $
  - → Ne pas confondre la chaîne "" vide et " " contenant un blanc.
- LEN (N$) est une fonction du BASIC donnant le nombre de caractères de la chaîne N$.
- MID$(N$, T, 1) est une fonction extrayant la sous-chaîne de longueur 1 à partir du Te caractère. En général MID$ (A$, T, N).
- Il existe LEFT$ (N$, G) et RIGHT$ (N$, D) qui extraient respectivement les G premiers ou les D derniers caractères de N$.
- Les constantes alphabétiques sont fournies entre guillemets (cf. 20, 30,)
- Le signe + (ligne 50) est l'opérateur de concaténation de chaînes.
- Ne pas confondre 11 + 3 et "11" + "3".

Quelques fonctions

- Chaque caractère ou fonction représenté au clavier correspond à un code interne normalisé dit code ASCII de ϕ à 127 (ou 255 selon les machines)
  ainsi le code ASCII de "A" est 65 ou 41_{16} ou ϕ1ϕϕ ϕϕϕ1
  le code " de "Z" est 9ϕ ou $5A_{16}$ ou ϕ1ϕ11ϕ1ϕ
  le retour du chariot CR, est 13 ou ϕD_{16} ou ϕϕϕϕ11ϕ1

- La fonction ASC ("I") ou ASC (B$) donne le code ASCII du caractère appelé.
  Inversement CHR$(N) fournit le caractère dont le code ASCII est N.

- STR$ (K) transforme le nombre K en la chaîne de caractères de ses chiffres
  STR$ (45.88) donne "45.88"

- VAL (X$) réalise l'opération inverse si X$ est une chaîne de caractères numériques

- SQR (N) fournit la racine carrée de N

- SIN (A), COS(A), TAN(A) sont les fonctions trigonométriques des angles exprimés en radians.

- LOG (X), LN(X) fournissent les logarithmes décimaux et népériens

- EXP (T) l'exponentielle

- RND (1) fournit un nombre "au hasard" entre ϕ et 1

- RND (ϕ) fournit toujours le même nombre

- INT (N) fournit la partie entière du réel N

Exemple :

```
1ϕ REM NBRES ENTIERS AU HASARD DE DEUX CHIFFRES
2ϕ FOR I = 1 TO 5ϕ
3ϕ PRINT INT(RND(1)*1ϕϕ) ;"k";
4ϕ NEXT I
5ϕ END
```

nota : on pourra rendre aléatoire l'amorce de la série d'une exécution à une autre en écrivant

RND (RND(1))

Quelques fonctions (suite)

- PEEK (N) fournit le contenu de la mémoire d'adresse N de l'ordinateur

- POKE N,D permet de placer le nombre décimal D à l'adresse N de la mémoire d'ordinateur.

Ces deux fonctions sont utilisées

- soit pour générer à partir du BASIC, un programme en langage machine (cf. fiche sur le fonctionnement d'un ordinateur)

- soit pour gérer directement l'écran ; chaque emplacement de celui-ci correspondant à une adresse-mémoire de la zone dite vidéo. (Voir le partage de la mémoire de chacun des microordinateurs utilisés).

- A noter que l'utilisateur peut créer ses propres fonctions par

DEF FN

ex : 3ϕ DEF FNF(X) = A*X+B

⋮

6ϕ PRINT FNF(ϕ.ϕ1)

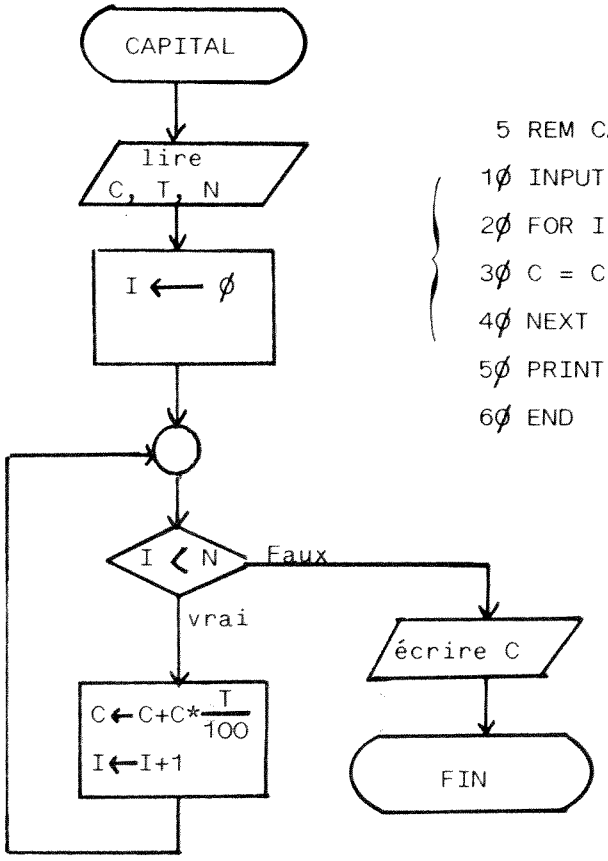
Structuration des Programmes (introduction)

- L'écriture de programmes assurant un grand nombre de fonctions enchaînées selon une logique assez complexe, nécessite une phase de préparation de programme, relevant plus de la logique et de la conception, que de techniques informatiques proprements dites.

- Ces techniques de préparation sont désignées sous le vocable d'algorithmique. Elles doivent aboutir à une structure indépendante du langage de programmation utilisé par la suite.

- Les techniques initialement employées faisaient appel aux organigrammes pour en visualiser le résultat.

Exemple :




```
 5 REM CAPITAL N AN NEES
1∅ INPUT C, T, N
2∅ FOR I = ∅ TO N-1
3∅ C = C + C * T/1∅∅
4∅ NEXT I
5∅ PRINT C
6∅ END
```

Cette technique, agréable pour les problèmes relativement simples, se complique très rapidement pour aboutir à des organigrammes touffus, où les flèches de raccordement s'emmêlent joyeusement. Son caractère spatial oblige ensuite à une gymnastique peu évidente pour aboutir à la structure nécessairement linéaire d'un programme écrit dans un quelconque langage.

Structuration (suite)

Enfin les organigrammes permettent de contourner une réflexion systématique sur l'enchaînement logique à prévoir et autorise tous les laxismes et toutes les contorsions logiques dans lesquelles le programmeur lui-même ne se retrouve plus guère, au bout de quelques semaines.

Les professionnels privilégient actuellement une démarche plus systématique et l'utilisation exclusive de quelques structures très simples, selon des règles précises d'emboîtement : on aboutit à la programmation structurée.

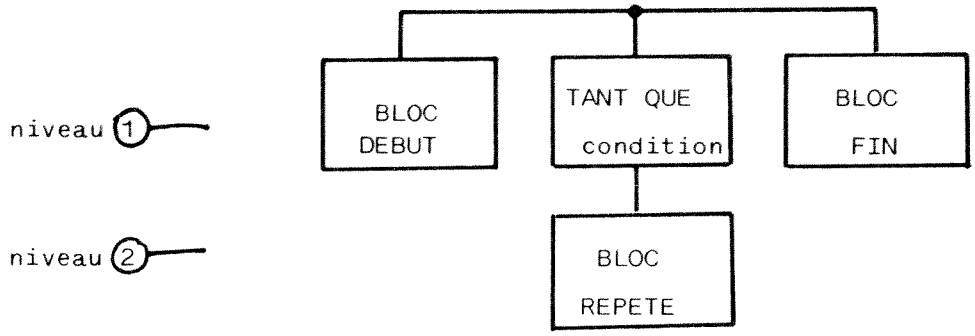
Structuration des programmes (les outils)

- La séquence d'instruction ou BLOC élémentaire

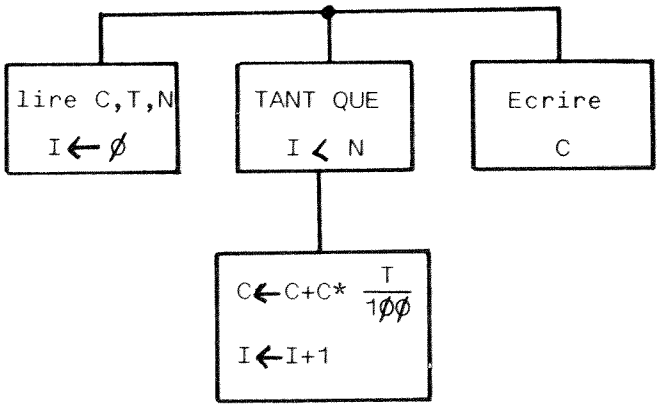
Exemple :

```
D ← D + 1
Q ← N/D
R ← N-D*Q
```

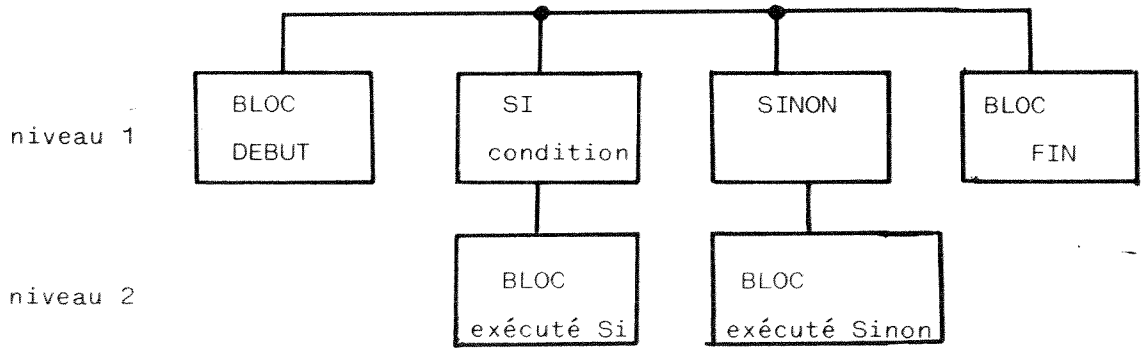
- La structure répétée ou structure TANT QUE



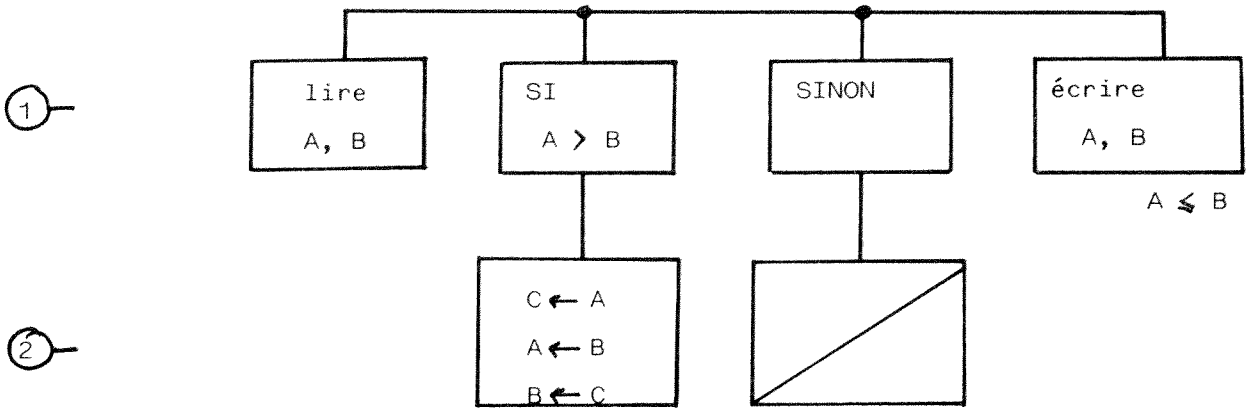
Exemple : (CAPITAL N)



- La structure alternative ou conditionnelle SI SINON

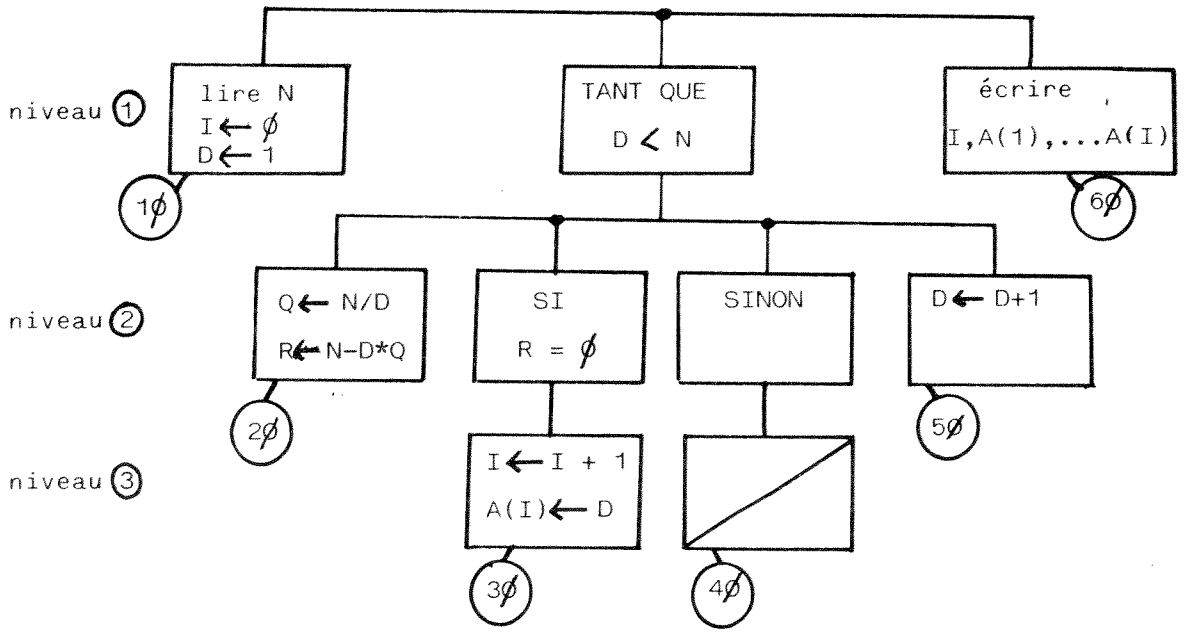


Exemple : rangement de deux nombres



- La combinaison de ces structures.

Exemple : recherche des diviseurs entiers naturels et non nuls/décomposition/d'un entier



Remarque La structure SI SINON est entièrement emboîtée dans la structure TANT QUE, comme l'indiquent les niveaux.

- Le BALAYAGE des feuilles numérotées de l'arborescence ainsi générée permet l'écriture linéaire du programme.

1∅ ...
2∅ ...
3∅ ...
4∅ ...
5∅ ...
6∅ ...

Le travail essentiel consiste à décomposer la tâche à programmer en BLOCS d'opérations selon qu'ils s'exécuteront

- une seule fois
- de façon répétée (contrôlée par quoi ?)
- conditionnellement à la valeur d'une expression logique

On procède par descente progressive du niveau le plus général au plus fin, ce dernier étant atteint lorsque les blocs d'instructions obtenus sont des BLOCS simples, contenus dans une des structures décrites plus haut (et une seule).

Algorithme de la démarche.

1° Décomposer le problème en grands BLOCS successifs s'enchaînant les uns après les autres (si nécessaire).

2° Rechercher dans chaque BLOC, les sous ensembles d'opérations s'effectuant

- une seule fois
- de façon répétée
- conditionnellement

Expliciter les conditions SI SINON et TANT QUE et faire apparaître les structures correspondantes et niveaux.

3° Recommencer le processus 2 au sein de chaque nouveau BLOC mis en évidence.

4° Arrêter lorsque l'ensemble de l'arborescence construite ne comprend plus que des blocs élémentaires insérés dans une des trois structures évoquées plus haut.

Remarque : à un niveau donné, le Bloc DEBUT ou FIN s'exécute une et une seule fois.

5° Numéroter les "feuilles" de l'arborescence de gauche à droite et expliciter leur contenu. (en suivant l'enveloppe convexe de l'arbre)

6° Balayer les feuilles dans l'ordre obtenu et rédiger le programme.

La démarche (suite)

Analyse complète d'un exemple (relativement simple)

Il s'agit de lire un texte, composé de mots séparés par un blanc ou un signe de ponctuation et de générer par programme, un nouveau texte dans lequel les mots (et eux seuls) auront été codés par retournement (BONJOUR donne RUOJNOB), les signes de ponctuation et les espaces restant à leur place entre les mots.

La longueur du message n'est pas donnée à l'avance (maximum 256 caractères) et aucun signe particulier n'indique la fin du texte (ce pourra être un signe de ponctuation ou... des blancs)

Exemple de structuration

- Etape 1

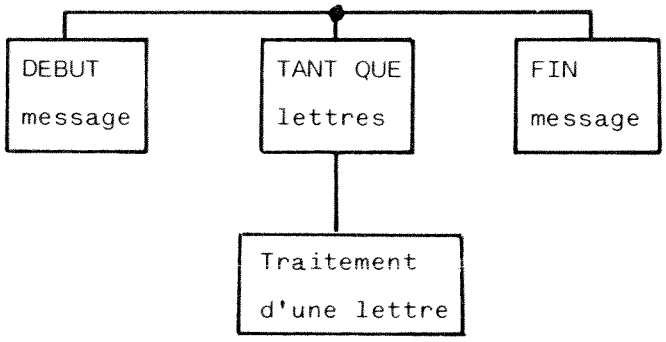
niveau 1

Observer que le retournement se fera nécessairement en balayant successivement chaque lettre ou signe composant le message d'où

- l'Etape 2

niveau 1

niveau 2

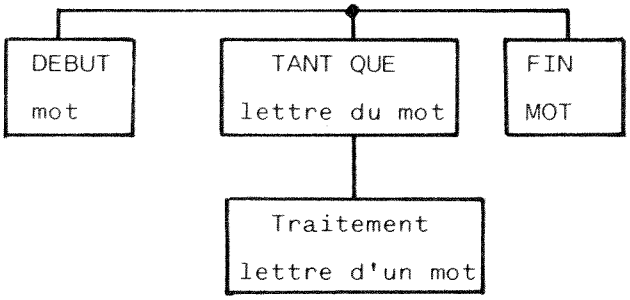


A ce niveau, on observera que les lettres d'un même mot subissent un traitement similaire, conduisant à la création du mot retourné, les autres lettres subissent un traitement différent d'où

- Etape 3 analysant le "bloc traitement d'une lettre" en :

niveau 2

niveau 3

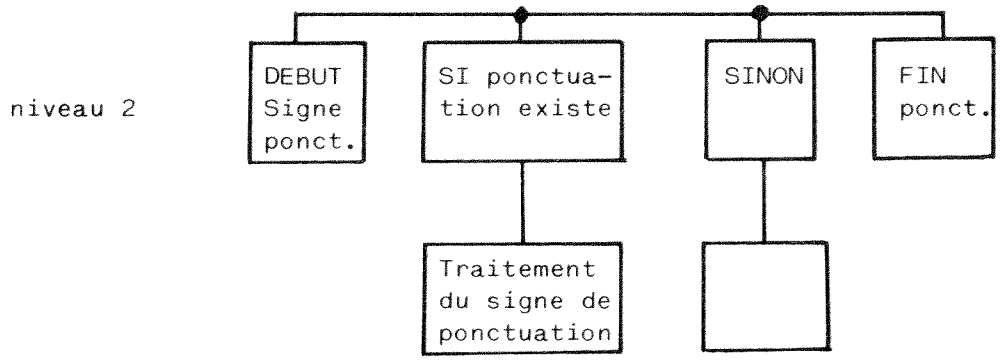


Le bloc du niveau 3 est simple, il comprend les instructions de concaténation du mot retourné : l'analyse descendante est épuisée.

Le bloc FIN d'un mot pose le problème du traitement du signe de ponctuation s'il existe d'où l'analyse de ce bloc.

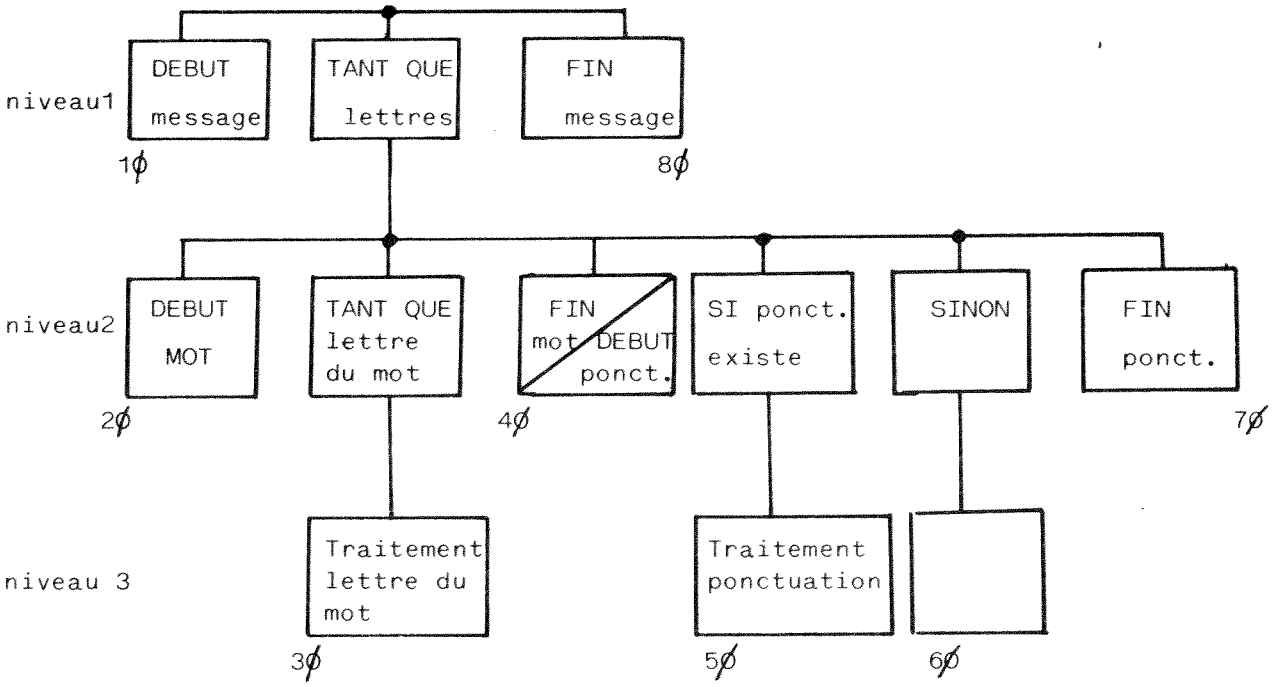
Exemple de structuration (suite)

- Etape 4



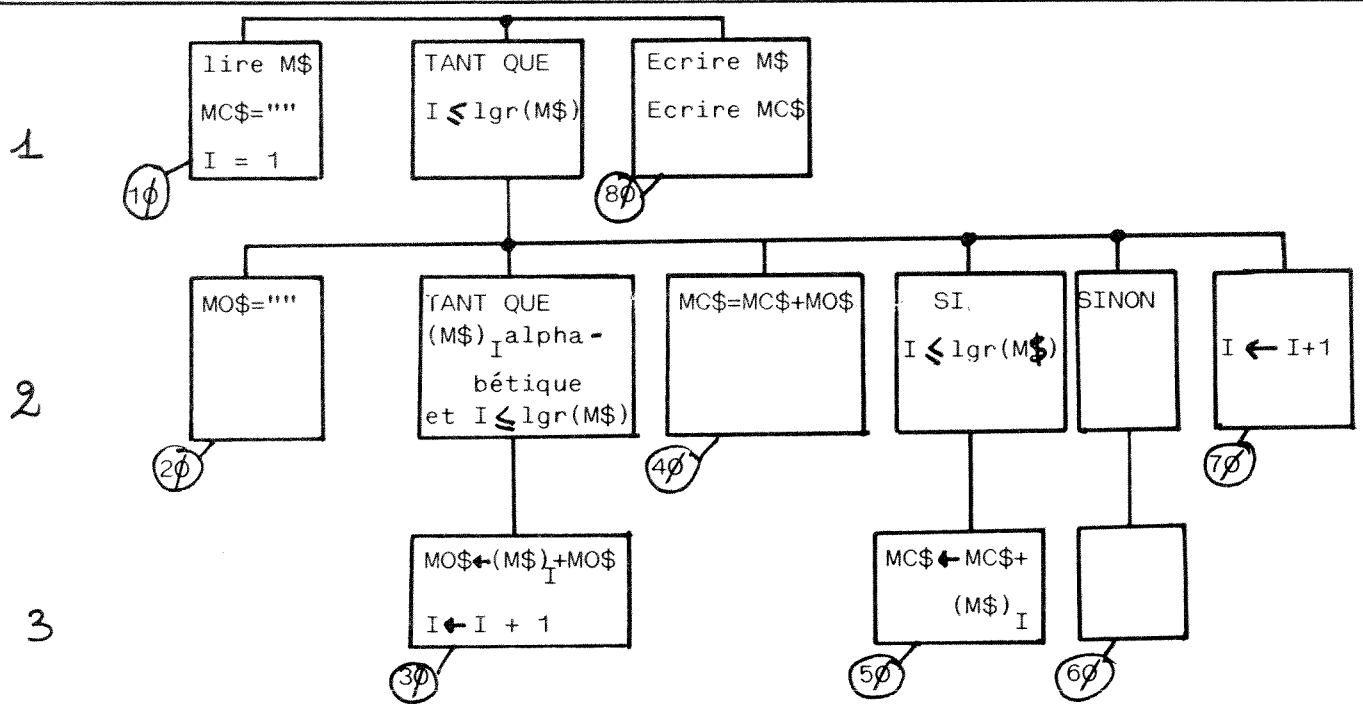
L'analyse en structures est épuisée, il reste à constituer l'arborescence, à numéroter les feuilles et à décrire explicitement chaque bloc.

ARBORESCENCE COMPLETE



- Etape 5 Explication des blocs élémentaires

Exemple de structuration (suite & fin)




```
    Lire M$
MC$ = ""                                          } (1ø)
I = 1
TANT QUE I ≤ LGR(M$)
FAIRE
  ‖ MO$ = ""                                      } (2ø)
  ‖ TANT QUE (M$)I alpha et I ≤ LGR(M$)
  ‖ FAIRE
(2)(3)‖ MO$ ← (M$)I + MO$                         } (3ø)
  ‖ I ← I+1
  ‖   FIN FAIRE
  ‖ MC$ = MC$ + MO$                               } (4ø)
(1)‖ SI I ≤ LGR(M$) ALORS
  ‖   FAIRE
  (3)‖ MC$ ← MC$ + (M$)I                          } (5ø)
  ‖   FIN FAIRE
  ‖ FIN SI                                        } (6ø)
  ‖ I ← I + 1                                     } (7ø)
  FIN FAIRE
écrire M$                                         } (8ø)
écrire MC$
```