12

SMAKY6

FORTH

Mai 1981

EPSITEC-system sa

1 MODE D'EMPLOI SIMPLIFIE

Pour démarrer le FORTH, il faut donner successivement les ordres suivants :

1)	FORTH	Le FORTH ne connait pas encore les procédures standards. Pour qu'il puisse les mettre dans son dictionnaire, il faut taper ensuite :
2)	SHOW-DEFINE FORTH	Ce qui insère dans le buffer d'édition le fichier FORTH.FH
3)	PROGRA-C	Démarre la compilation
4)	END	Retourne dans l'éditeur.
5)	CURSOR-KILL-0	Vide le contenu du buffer d'édition.

Le FORTH est maintenant prêt pour une utilisation normale.

Prenons par exemple une procédure qui additionne deux nombres et qui affiche le résultat sur l'écran:

```
: ADD + . ;
```

Pour la compiler, il faut taper : PROGRA-C

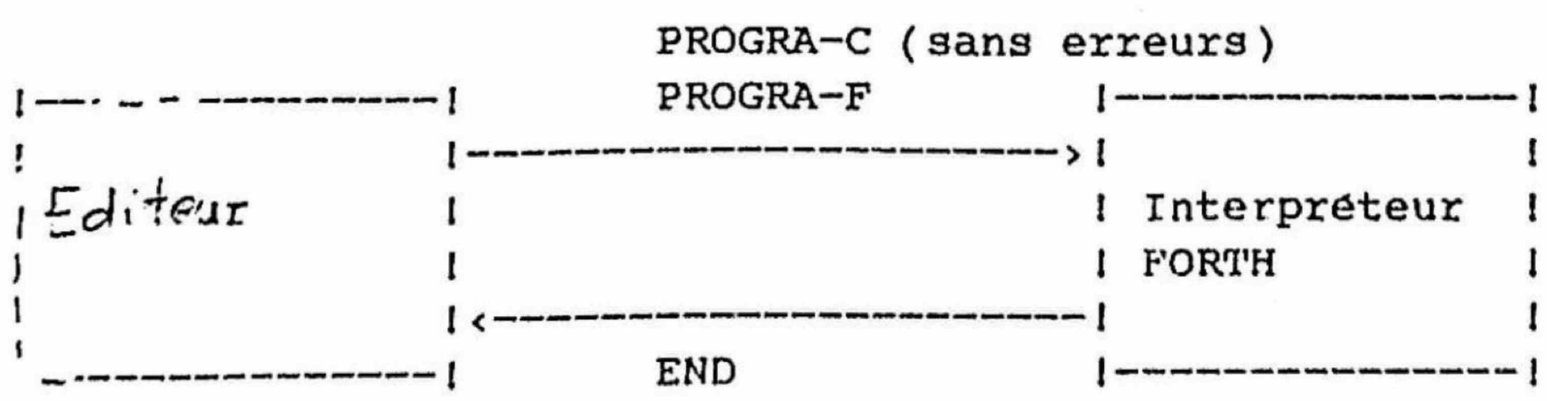
Si la compilation est correcte, nous nous trouvons automatiquement dans l'interpréteur FORTH. Dans le cas contraire, nous restons dans l'éditeur et la ou les erreurs sont signalées par une flèche suivie du message d'erreur.

Depuis l'interpréteur FORTH, nous pouvons essayer la procédure ADD en tapant: 2 3 ADD
L'écran montre alors 5.

Pour retourner dans l'éditeur, il faut appuyer sur END.
Depuis l'éditeur, il est possible d'aller dans l'interpréteur FORTH et tapant: PROGRA-F

L'interpréteur FORTH permet de taper des lignes qui sont interprétées sitôt que le CR a été donné.

Résumé :

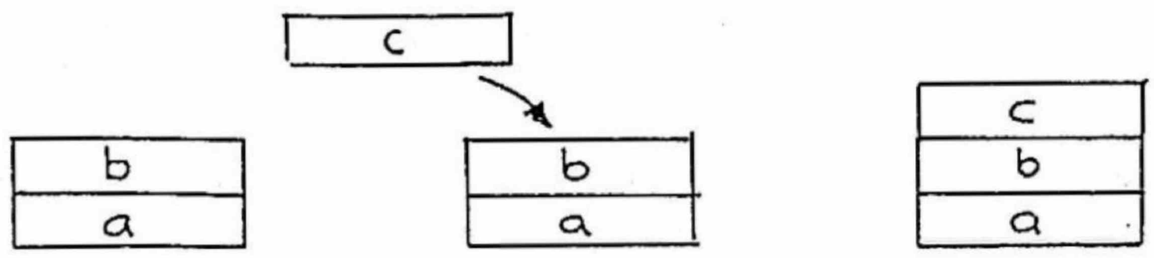


2. LA PILE ARITHMETIQUE

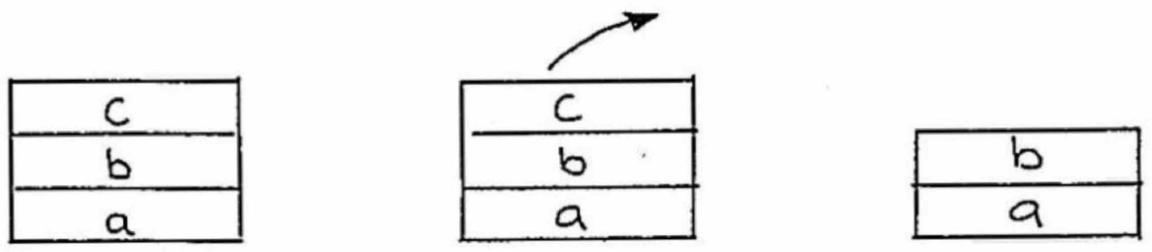
L'arithmétique du FORTH est basée sur l'emploi d'une pile polonaise; c'est là l'une des principales caractéristiques du FORTH.

La pile arithmétique (ou polonaise) est une zone de mémoire dont on se sert en FORTH pour stocker les arguments des diverses fonctions utilisées; lorsqu'une fonction a été exécutée, elle supprime les arguments de la pile et met en place le résultat. L'utilisation d'une pile se fait selon deux principes.

1. Lorsque l'on dépose un nouvel élément sur la pile, il vient se placer au-dessus de tous les autres.



2. De même, on ne peut retirer que l'élément se trouvant au sommet de la pile

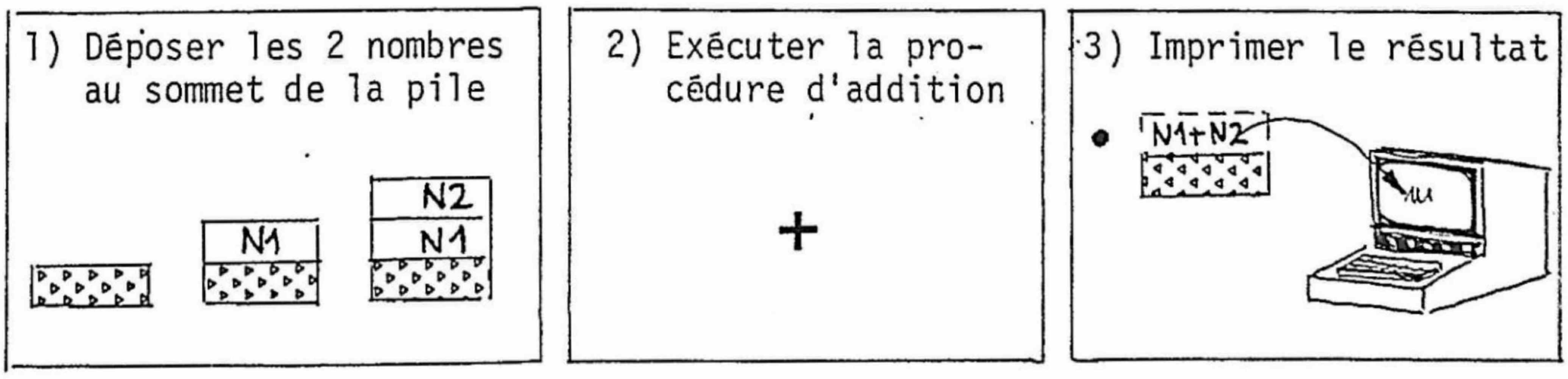


(On agit comme si on se trouvait en présence d'une pile d'assiettes qui a tendance à s'écrouler si on enlève une assiette se trouvant au milieu).

On remarque que le premier élément à entrer dans la pile sera le dernier à en sortir.

En FORTH, les éléments de la pile sont des mots mémoire (16 bits) pouvant représenter un nombre entier (de -32768 à +32767) ou un caractère selon le code ASCII. On peut aussi considérer des éléments formés de deux mots (32 bits) servant à la représentation des nombres réels (de -10^{39} à $+10^{39}$).

Toutes les fonctions du FORTH utilisent comme arguments les éléments se trouvant au sommet de la pile. Prenons par exemple la fonction addition et voyons comment il faut procéder pour faire la somme de deux nombres



Ce qui donne la séquence FORTH suivante: `56 349 + .↵`

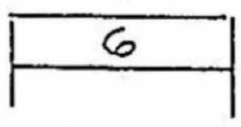
Lorsqu'on imprime un élément (ordre .↵), on l'enlève de la pile !

Pour pouvoir essayer cette séquence et toutes les suivantes, il faut se mettre en mode interprêteur (PROGRAM F).

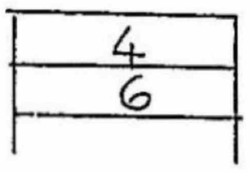
Les ordres ou les nombres doivent être séparés par des espaces ou des tabulateurs.

AUTRE EXEMPLE:

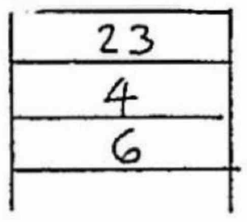
Soit à calculer: (6*(4-23))/8



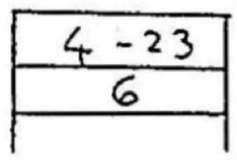
Mettre 6 sur la pile



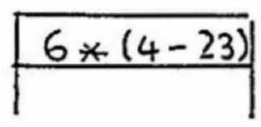
Mettre 4 sur la pile



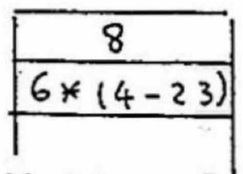
Mettre 23 sur la pile



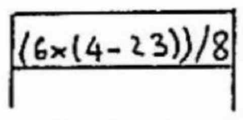
soustraction (-)



multiplication (*)



Mettre 8 sur la pile



Division (/)

En FORTH: `6 4 23 - * 8 /`

L'opération mettre ... sur la pile est sous-entendue à chaque fois qu'on donne un nombre. Elle est donc effectuée automatiquement pour tout nombre donné. Par contre, le résultat final n'est pas imprimé et reste sur la pile jusqu'à ce qu'on appelle la fonction . qui l'enlève de la pile et l'imprime sur le terminal.

ENCORE UN EXEMPLE: Calculons 8+5+(2*5)-(7+6)

En FORTH: `8 5 + 2 5 * + 7 6 + -`

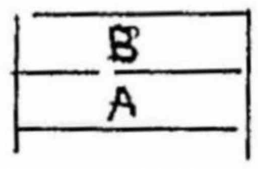
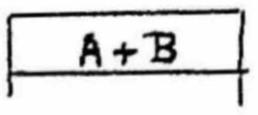
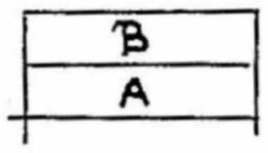
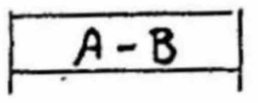
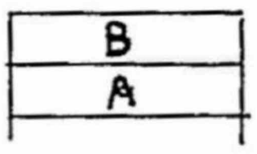
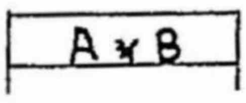
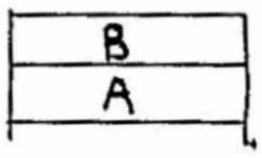
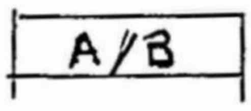
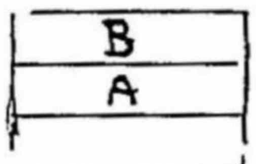
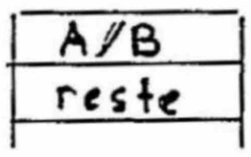
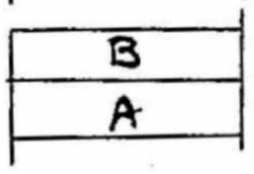
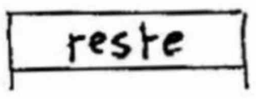
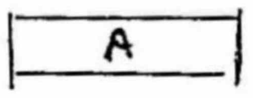
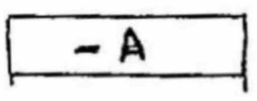
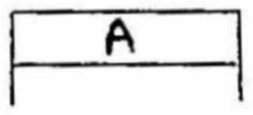
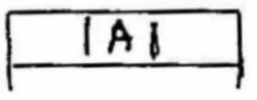
EXERCICE1: Ecrire les séquences FORTH permettant de calculer:

1) 3*(9+2)
2) 5/(4-3-5)
3) 6-((5-3)/4*(2-11))+7)

ATTENTION: En FORTH toutes les fonctions et tous les nombres doivent être séparés par au moins un blanc.

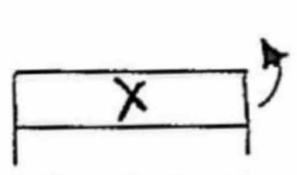
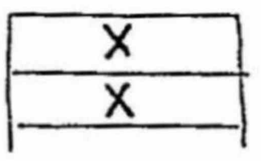
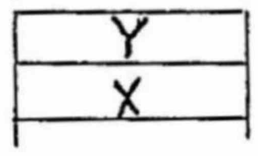
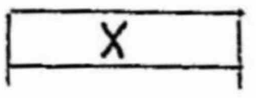
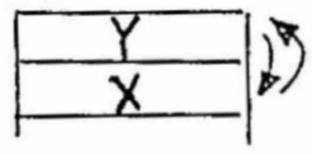
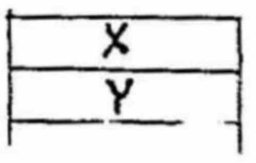
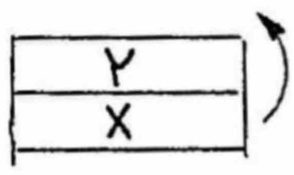
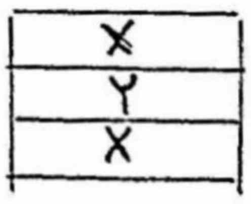
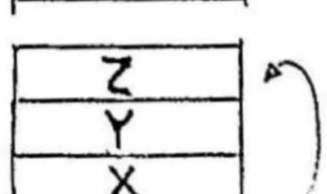
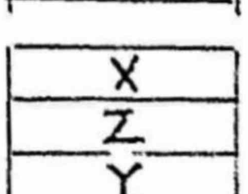
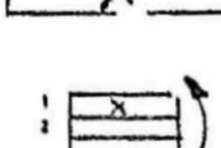
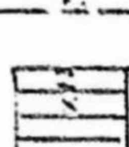
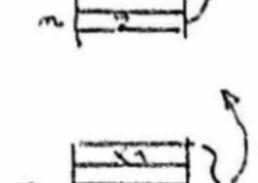
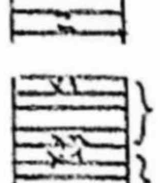
Le système de notations utilisé en FORTH est appelé notation POSTFIXE car on donne d'abord les opérandes et ensuite l'opérateur.

PROCEDURES DE CALCUL ENTIER

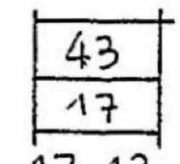
nom	fonction	avant	après
+	addition des 2 nombres au sommet		
-	soustraction		
*	multiplication		
/	division		
/MOD	division avec reste		
MOD	reste de la division (modulo)		
MINUS	opposé du nombre au sommet		
ABS	valeur absolue du nombre au sommet		

PROCEDURES DE MANIPULATION DE LA PILE

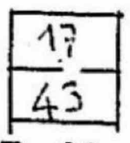
Ces procédures ont pour but de modifier l'ordre dans lequel se trouvent certains éléments de la pile, ce qui permet d'étendre les possibilités de calcul.

DUP	duplifie l'élément se trouvant au sommet		
DROP	enlève l'élément au sommet		
SWAP	change l'ordre des 2 éléments du sommet		
OVER	copie le 2ème élément au-dessus du premier		
ROT	rotation des 3 éléments du sommet		
n NOVER	copie le $n^{ième}$ élément au-dessus du sommet		
n NDUP	duplifie les n éléments supérieurs de la pile		

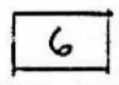
EXEMPLES



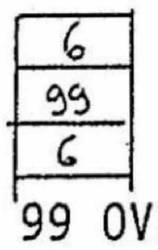
17 43



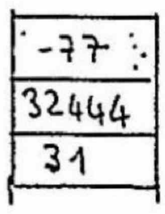
17 43 SWAP



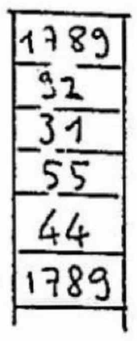
6 99 DROP



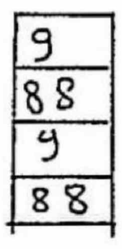
6 99 OVER



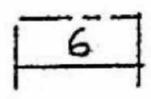
-77 31 32444 ROT



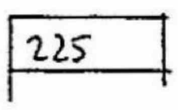
1789 44 55 31 92 5 NOVER



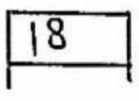
9 88 2 NDUP



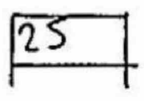
6 15 SWAP DROP



15 DUP *



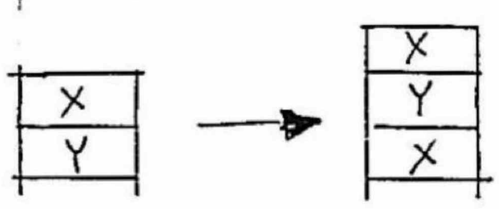
6 DUP DUP + +

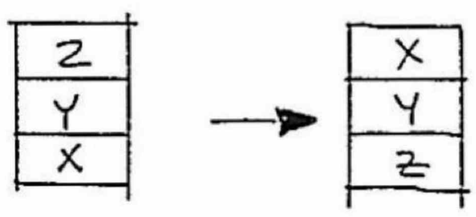


75 100 SWAP -

EXERCICE 2:

a) Écrire les séquences qui permettent de passer de l'état 1 de la pile à l'état 2, puis vérifier les résultats, en utilisant le FORTH.





b) A partir d'un nombre déposé sur la pile, calculer

- le double du nombre (sans utiliser *)
- le triple du nombre (sans utiliser *)
- le quadruple (sans utiliser *) 2 méthodes
- le carré
- le carré du double

c) A partir de deux nombres sur la pile (a et b) calculer

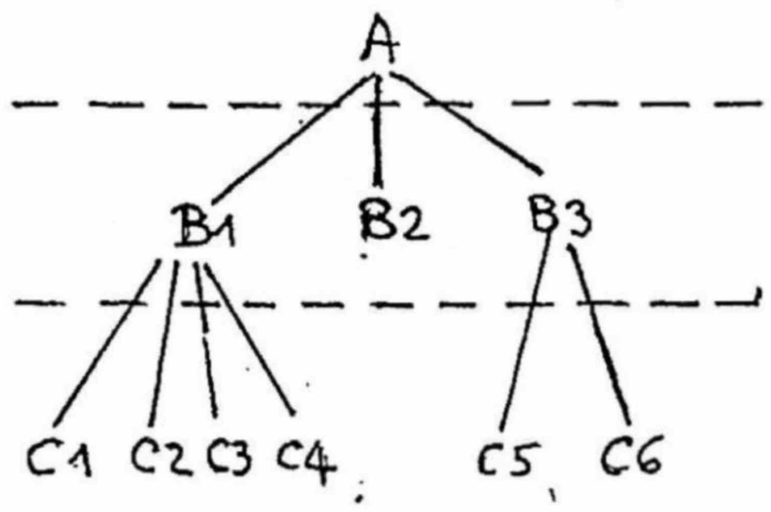
- (a-b)*a
- (a-b)*(a-b)
- a/(b-(a*b))

Il est absolument nécessaire d'avoir bien compris toutes les procédures décrites. De plus, un bon entraînement est recommandé pour apprendre à s'en servir efficacement.

□ .3.2 COMMENT CONSTRUIRE UN PROGRAMME FORTH A L'AIDE DES PROCEDURES

Si on a un problème (P) à résoudre en FORTH, on essaie tout d'abord de le décomposer en plusieurs sous-problèmes (PA, PB, PC, ...) plus simples à résoudre; si parmi ces sous-problèmes certains sont encore trop compliqués, on peut les décomposer à leur tour en sous-sous-problèmes et ainsi de suite jusqu'à obtenir une (grande) quantité de petits problèmes simples que l'on peut résoudre par des procédures FORTH de petite taille (de 2 à 30 instructions). Le fait d'avoir beaucoup de petites procédures augmente la clarté du programme et permet une mise au poini rapide et aisée.

Décomposition du problème



Traduction en FORTH

```
: C1 /instructions/ ;
: C2 /instructions/ ;
: C3 /instructions/ ;
: C4 /instructions/ ;
: C5 /instructions/ ;
: C6 /instructions/ ;

: B1 C1 C2 C3 C4 ;
: B2 /instructions/ ;
: B3 C5 C6 ;

: A B1 B2 B3 ;
```

Pour tester séparément une procédure, on met sur la pile les arguments dont elle a besoin et on l'exécute. Si on retrouve sur la pile le résultat escompté, on peut considérer que la procédure est correcte (deux essais valent mieux qu'un !). Si par contre la procédure donne un résultat faux, il faut commencer par vérifier si les instructions de la procédure correspondent à ce que l'on veut faire; si c'est le cas, il faut alors tester toutes les procédures appelées par celle là et trouver celle(s) qui est (sont) incorrecte(s) et les corriger selon la même méthode.

NB: plus une procédure est longue, plus elle est difficile à mettre au point!

COMMENTAIRES. La procédure (a pour effet de suspendre la compilation du texte tapé jusqu'au premier) rencontré ou, s'il n'y en a pas, jusqu'à la fin de la ligne.

Par exemple: `: SX DUP DUP * + ( CALCULE X*X+X ) ;`

Il doit toujours y avoir au moins un blanc entre (et le début du texte du commentaire.

ECRITURE STANDARD DES NOMBRES ET DES TEXTES

Il existe plusieurs procédures permettant d'imprimer des nombres et des textes en format standard. Nous verrons plus tard des méthodes plus complètes d'entrée/sortie.

DECIMAL	place le FORTH en mode d'entrée sortie décimale (normal)
OCTAL	entrées-sorties en base 8
.	imprime le nombre au sommet de la pile sur le terminal en décimal ou en octal
CR	passe au début de la ligne suivante
"	imprime le texte qui suit jusqu'au prochain " sur le terminal (le premier " doit être séparé du texte par un blanc)

EXEMPLES:

```
: IMPRIME " LE NOMBRE EST: " . ( IMPRESSION ) ; ↵
6 IMPRIME ↵
LE NOMBRE EST: 6

22 OCTAL IMPRIME ↵
LE NOMBRE EST : 26

7 1 + .     ( EXEMPLE DE CALCUL EN BASE 8 ) ↵
10

DECIMAL ↵
: AZX DUP 1 - * " X FOIS X-1 =" . ; ↵
6 AZX ↵
X FOIS X-1 = 30

5 AZX 8 AZX ↵
X FOIS X-1 =  20 X FOIS x-1 = 56

5 AZX CR 8 AZX ↵
X FOIS X-1 = 20
X FOIS X-1 = 56
```

REMARQUE: la procédure " ne peut être appelée que depuis une autre procédure, elle ne peut pas être exécutée directement en mode interpréteur

□ 3.3 DESTRUCTION DE PROCEDURES

FORGET — Cette procédure, suivie d'un nom de procédure enlève du dictionnaire toutes les procédures créées après celle spécifiée y compris.

Cela signifie que si l'on crée une procédure A, puis une autre B, on ne peut supprimer A sans en même temps supprimer B.

Le FORGET est très utile lorsqu'on s'aperçoit que la dernière procédure entrée contient une faute, on peut l'enlever, puis la retaper.

```
: QUARK DUP * / ; ↵
FORGET QUARK ↵
: QUARK SWAP * / . ; ↵
```

ATTENTION: FORGET suivi d'un nom de procédure qui n'existe pas, ou suivi d'aucun nom de procédure, détruit toutes les procédures de calcul avec des nombre réels, ainsi que toutes les procédures graphiques.

LES NOMS DE PROCEDURES

Un nom de procédure peut contenir jusqu'à 55 caractères, cependant le FORTH ne tient compte que des trois premiers et de la longueur totale du nom (afin d'économiser de la place en mémoire). Ainsi si deux noms de procédure ont la même longueur et commençent par les trois mêmes caractères, ils sont synonymes. Dans ce cas, toutes les références faites à l'un ou l'autre nom se rapporteront à celui qui a été défini en dernier. Le fait d'avoir des synonymes dans le dictionnaire FORTH n'est pas considéré comme une erreur, c'est à l'utilisateur de se méfier de ce phénomène.

EXEMPLES:

```
: AAARTX " A+B = " + . ;↵
: AAATTY " A-B = " - . ;↵
6 8 AAARTX↵
```

A-B = -2 (c'est AAATTY qui a été exécutée)

```
FORGET AAATTY↵
6 8 AAARTX↵
```

A+B = 14 (cette fois on a la bonne procédure !)

ERREURS DE COMPILATION

Si une procédure appelée n'existe pas, le compilateur imprime comme message d'erreur le nom en question suivi d'un point d'interrogation.

EXEMPLE:

```
: WW SWAP - DUL . ;↵
```

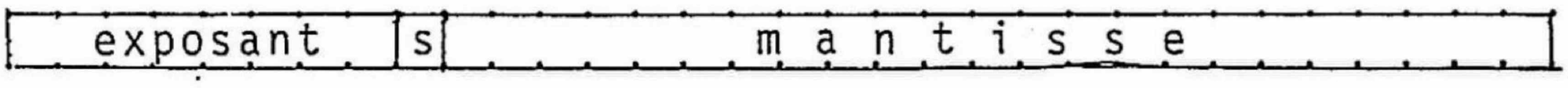
DUL ..? (cette procédure n'existe pas)

FORGET WW : WW SWAP - DUP . ;↵ (on efface tout et on recommence)

OK (maintenant tout est OK)

■ 4.1 LES PROCEDURES DE CALCUL REEL

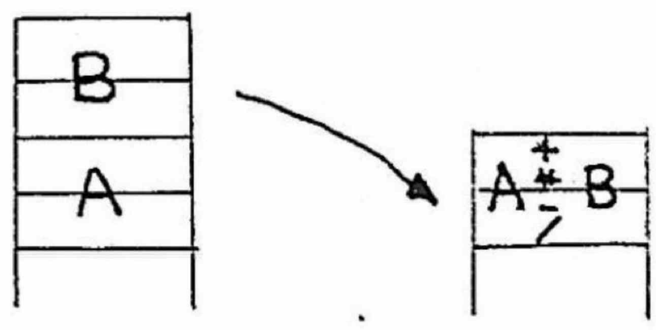
Un nombre réel est formé de deux mots mémoire, donc de deux éléments de la pile. Le format adopté pour les réels est le suivant:



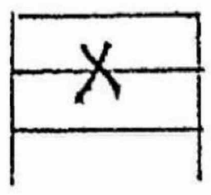
Un nombre réel doit toujours être suivi d'un point. Par exemple:

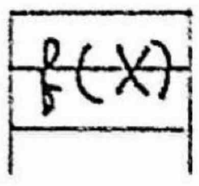
-1. 15.0 6.02

F+	addition des 2 réels sur la pile
F*	multiplication
F-	soustraction
F/	division



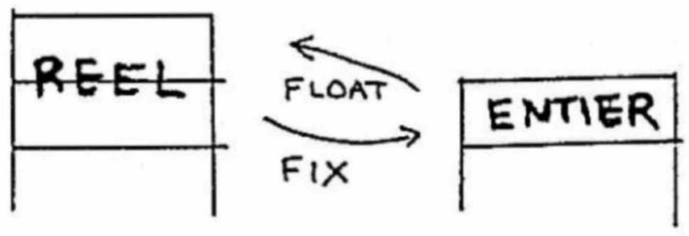
ATAN	arctangente du réel sur la pile
COS	cosinus
EXP	exponentielle
FABS	valeur absolue
FMINUS	opposé
LN	logarithme naturel
LOG	logarithme en base 10
SIN	sinus
SQRT	racine carrée
TAN	tangente





4.2 CONVERSIONS ENTIERS - REELS

FLOAT	transforme l'entier sur la pile (1 mot) en son équivalent réel (2 mots)
FIX	opération inverse de FLOAT, transforme le réel (2 mots) en entier (1 mot). Il faut s'assurer que le réel en question est compris entre -32768.0 et 32767.0



MANIPULATION DE LA PILE AVEC DES ELEMENTS DOUBLES

Toutes les procédures de manipulation de la pile avec des éléments simples ont leur équivalent pour les doubles sauf NOVER et NDUP.

FDUP	duplifie les deux éléments du sommet de la pile c'est-à-dire le réel qui s'y trouve
FSWAP	SWAP avec deux réels
FOVER	OVER avec deux réels
FDROP	DROP avec un réel
FROT	ROT avec trois réels

4.3. IMPRESSION DES REELS

E.	la procédure E. imprime sur la console le réel se trouvant sur la pile selon un format exponentiel

EXEMPLES:

3.2 E. 1615.2 E.↵

.3200000E+01 .1615200E+04

3.3 E.↵

.3300000E+01

5.1 6.1 5.5 FSWAP E.↵

.6100000E+01

F/ E.↵

.9272727E+00

1.2 2.1 F+ E.↵

.3300000E+01

5.676 SIN E. 5.676 COS E.↵

-.5705520E+00 .8212594E+00

1.2 SIN SQRT E.↵

.9654226E+00

```
: FCARRE FDUP F* ;
1.1 SIN FCARRE 1.1 COS FCARRE F+ E.
0.9999990E+00
```

EXERCICE 3.

Ecrire des procédures permettant de calculer:

a) X^Y d'après la formule $x^y = \exp(y*\ln(x))$

b) ARCSIN(X) selon la formule $\arcsin(x) = \arctan((1-x^2)^{\frac{1}{2}}/x)$

c) La distance entre deux points de R^2 de coordonnées (x_1,x_2) et (y_1,y_2) en prenant la formule $d = \mathrm{sqrt}((x_1-y_1)^2+(x_2-y_2)^2)$

d) La surface d'un cercle de rayon donné

5 INSTRUCTIONS DE TEST ET DE REPETITION

5.1 FONCTIONS LOGIQUES

En FORTH les deux valeurs logiques "vrai" et "faux" sont représentées par les variables entières 1 et Ø. On peut effectuer sur ces valeurs les opérations logiques habituelles grâce aux procédures suivantes:

AND	et logique des deux valeurs au sommet de la pile
OR	ou logique
XOR	ou exclusif
NOT	non

EXEMPLES:

```
0 1 OR .↵
1           (faux ou vrai = vrai)
1 0 AND .↵
0           (vrai et faux = faux)
0 0 OR 1 AND .↵
0           ((faux ou faux) et vrai = faux)
```

Il existe aussi des procédures dites de test dont les arguments sont des nombres entiers ou réels et le résultat une valeur logique.

=	prend les deux entiers au sommet, les compare et dépose 1 sur la pile s'ils étaient égaux, 0 sinon.
>	dépose 1 si A>B 0 sinon
<	dépose 1 si A<B 0 sinon

De même pour les réels:

F=	donnent 1 ou 0 selon que le test est vrai ou faux
F>	
F<	

EXEMPLES:

```
6 6 = .↵
1
 ok

6 7 > .↵
0
 ok

3 4 * 5 6 - < .↵
0
 ok

5.676 4.3 SIN F> .↵
1
 ok
```

□ 5.2 IF...THEN

Les instructions IF...THEN permettent d'exécuter une séquence de programme seulement si une certaine fonction logique a donné un résultat vrai (1 sur la pile). Le IF...THEN s'emploie gégéralement de la manière suivante:

```
... /fonctions donnant un résultat logique/ IF /séquence/ THEN ...
```

La séquence d'instructions entre IF et THEN est exécutée seulement si la valeur logique sur la pile avant IF vaut 1.

```
: DIVISE ↵                          (on effectue la division seulement si le
DUP 0 = NOT IF / THEN ; ↵            diviseur est différent de zéro)
6 2 DIVISE .↵
3
 ok                                 (division effectuée)

76 0 DIVISE .↵
0
 ok

.↵                                  (division non effectuée)
76
 ok

: DIV2 ↵                            (cette procédure divise un nombre par 2
DUP 2 MOD Ø = ↵                      seulement s'il est pair)
IF 2 / THEN ;↵
456 DIV2 .↵
228
 ok

9999 DIV2 .↵
9999
 ok
```

REMARQUES: Le IF supprime de la pile la valeur logique qui s'y trouve.
Il faut se souvenir que les différents tests logiques suppriment les nombres testés de la pile; il faut donc duplifier les nombres qui doivent servir après le test. Le THEN doit toujours être présent car il indique la fin de la séquence à exécuter conditionnellement.

EXEMPLE:

```
: LIMIT↵                                          (cette procédure remplace le nombre sur
DUP 100 > OVER 8 < OR ↵                           la pile par un 99 si celui-ci est plus grand
IF DROP 99 THEN " VOICI LE NOMBRE:" . ;↵ grand que 100 ou plus petit que 8)
2 LIMIT↵

VOICI LE NOMBRE: 99

69 LIMIT↵
VOICI LE NOMBRE: 69
```

□ 5.3 IF...ELSE...THEN

Avec ces instructions, on peut exécuter la séquence si la condition avant IF est vraie ou une autre séquence si elle est fausse.

```
... /condition/ IF /séquence1/ ELSE /séquence2/ THEN ...
```

Si la condition (=fonction logique) a donné un résultat vrai, la séquence1 est exécutée, sinon c'est la séquence2

EXEMPLE:

```
: TEST-SGN 0 < IF " NOMBRE NEGATIF" ELSE " NOMBRE POSITIF" THEN ; ↵
4 TEST-SIGN ↵

NOMBRE POSITIF

-89 TEST-SGN ↵

NOMBRE NEGATIF

4 7 -3 + 2 * - 1 / TEST-SGN ↵

NOMBRE POSITIF

: RACINE FDUP 0.0 F>  IF " LA RACINE VAUT: " SQRT E. ELSE " ERREUR NOMBRE NEGATIF"↵
FDROP THEN ;↵
6.2 RACINE↵

LA RACINE VAUT: 0.2489981E+01

-67. RACINE ↵

ERREUR NOMBRE NEGATIF
```

□ 5.4 BEGIN...END

Ces deux instructions servent à faire répéter une séquence jusqu'à ce qu'une condition soit vraie.

```
... BEGIN /séquence/ /condition/ END ...
```

EXEMPLES:

1) Multiplier le nombre sur la pile par 3 jusqu'à ce qu'il soit plus grand que 1000.

```
: EX1 BEGIN 3 * DUP 1000 > END ;↵
25 EX1 .↵

2025
```

2) Imprimer tous les multiples de 7 plus petits que 60.

```
: EX2 7 BEGIN DUP . CR 7 + DUP 60 > END   DROP ;
EX2
```

```
7
14
21
28
35
42
49
56
```

3) Soient trois nombres sur la pile, on veut diviser le plus grand des deux par 2 et répéter l'opération jusqu'à ce que l'on obtienne deux nombres égaux.

```
: ZZ BEGIN
2 NDUP > IF SWAP THEN          (inverse l'ordre si le 2e est plus grand)
2 /                            (divise le plus grand par 2)
2 NDUP =                       (teste s'ils sont égaux)
END . . ;                      (si oui: imprime les nombres, sinon continue)
6 4 ZZ
```

```
1 1
```

```
12 14 ZZ
```

```
3 3
```

4) Tabule la fonction ln sin.

```
: FONXION LN SIN ;
: TABULE 0.5                                                    (valeur de départ)
" TABULATION DE LA FONCTION SIN(LN(X)) DE 0.5 A 10.0" CR CR
BEGIN
FDUP E.                                                         (imprime X)
FDUP FONXION E.                                                 (imprime F(X))
CR 0.3 F+ ( INCREMENT ) FDUP 10.0 F>                            (test de fin)
END ;
TABULE
```

```
TABULATION DE LA FONCTION SIN(LN(X)) DE 0.5 A 10.0

.500000E+00    -.638592E+00
.800000E+00
```

etc.

□ 5.5 BEGIN...IF...WHILE

Avec ces instructions de répétition, on fait la séquence tant qu'une condition est vraie.

...BEGIN /condition/ IF /séquence/ WHILE...

EXEMPLES:

1. Calculer la somme des entiers de 1 à n (n sur la pile)

```
: SOMME  0 BEGIN  OVER 0 >↵   (commence avec une somme nulle, teste si n > 0)
IF OVER + SWAP 1 - SWAP ↵     (si oui additionne n à la somme et ôte 1 ã n)
WHILE . DROP ;↵               (recommence...imprime la somme et supprime n
6 SOMME ↵                      (qui vaut maintenent 0)
```

21

2) Soustrait 5 au nombre sur la pile et imprime le nombre restant tant que celui-ci est plus grand que 9.

```
: S5 BEGIN DUP 9 ↵
IF DUP " RESTE: " . CR↵
5 - WHILE DROP ;↵
12 S5 ↵
```

RESTE: 12

```
6 S5↵
```

(rien n'est imprimé vu que le nombre de départ est inférieur à 9).

REMARQUE: dans un BEGIN-END la séquence est toujours exécutée au moins une fois vu que le test se fait à la fin; tandis qu'avec BEGIN-IF-WHILE la séquence peut ne pas être exécutée une seule fois si la condition est fausse dès le départ comme dans le dernier cas de l'exemple 2.

□ 5.6 DO...LOOP et DO... +LOOP

Ces instructions font exécuter une séquence tant que l'indice spécifié est plus petit que se valeur maximale. Chaque exécution de la séquence additionne 1 à l'indice ou un nombre quelconque si on utilise +LOOP. L'indice de départ et sa limite sont initalisés par les deux entiers se trouvant sur la pile au moment de l'exécution du DO.

imax imin DO /séquence/ LOOP	ou

imax imin DO /séquence/ incr +LOOP

La procédure I dépose sur la pile la valeur actuelle de l'indice.

EXEMPLES:

1) Ecrire 3 fois "ça tourne" sur l'écran.

```
: ECRIT 3 1 DO ↵              (l'indice ira de 1 à 3)
" CA TOURNE" CR↵              (impression)
LOOP ;↵
ECRIT↵
```

CA TOURNE
CA TOURNE
CA TOURNE

2. Imprimer les nombres entiers de 8 è 15

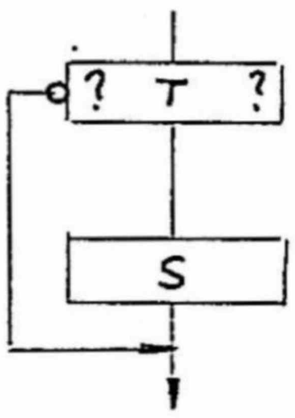
```
: ENTIERS 15 8 DO I . LOOP ;↵
ENTIERS↵
8 9 10 11 12 13 14 15
```

3) Imprimer une * sur l'écran en n^{e} position, où n est le nombre sur la pile.

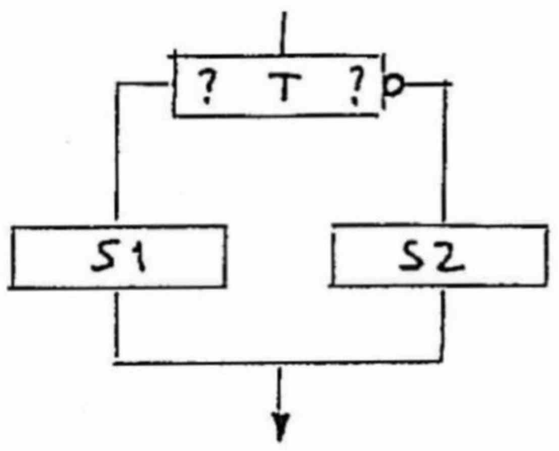
```
: XXX 1 DO "  " LOOP " *" ;
4 XXX↵
   *
   —
20 XXX↵
                    *
                    —
: CCC 10 2 DO I DUP * 2 / XXX CR LOOP ; CCC↵
  *
    *
      *
        *
          *
            *
              *
```

□ 5.7 RESUME DES INSTRUCTIONS CONDITIONNELLES ET REPETITIVES

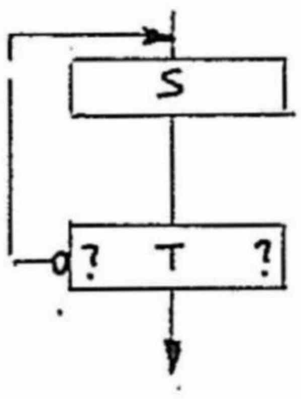
IF...THEN



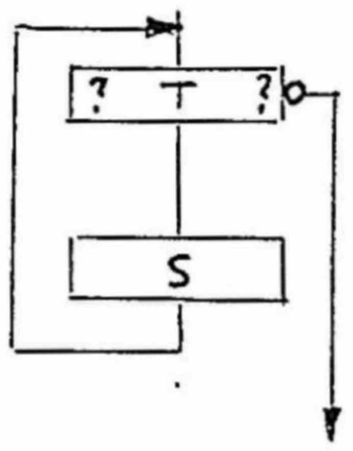
IF...ELSE...THEN



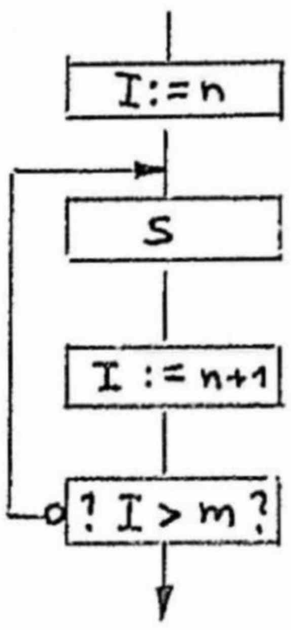
BEGIN...END



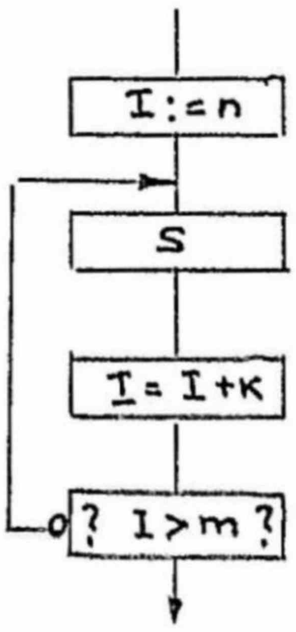
BEGIN...IF...WHILE



m n DO...LOOP



m n DO... k +LOOP



Toutes ces instructions peuvent être combinées pour obtenir des structures plus complexes. Par exemple, introduire un DO...LOOP à l'intérieur de la séquence contrôlée par un BEGIN...END.

On remarquera que les organigrammes précédents ont tous un seul point d'entrée et un seul point de sortie ce qui permet un bon contrôle de la logique du programme. Il est bien entendu que ces différentes structures ne peuvent pas se chevaucher.

EXEMPLE: ... IF ... BEGIN ...THEN ... END ... est totalement faux.

Par contre, on peut toujours les imbriquer les uns dans les autres.

EXEMPLE:

```
IF ... BEGIN ... END
ELSE ... IF ... DO ... LOOP
         ELSE...
         THEN
         ...
THEN
...
```

■ 6.1 LES VARIABLES

Les variables sont des zones de mémoire de taille et de structure diverses qui sont repérées par un nom. Dans une variable on peut stocker: un entier, un réel, un caractère, etc. Il existe un type de variable pour chaque type de donnée traitée par le FORTH.

VARIABLES SIMPLES

n INTEGER nom	Définit une variable pouvant contenir un entier, c'est à dire occupant 1 mot mémoire. La variable porte le nom indiqué après INTEGER et contient comme valeur initiale le nombre n.
r FLOATING nom	Définit une variable de type réel (2 mots) dont la valeur initiale est r.

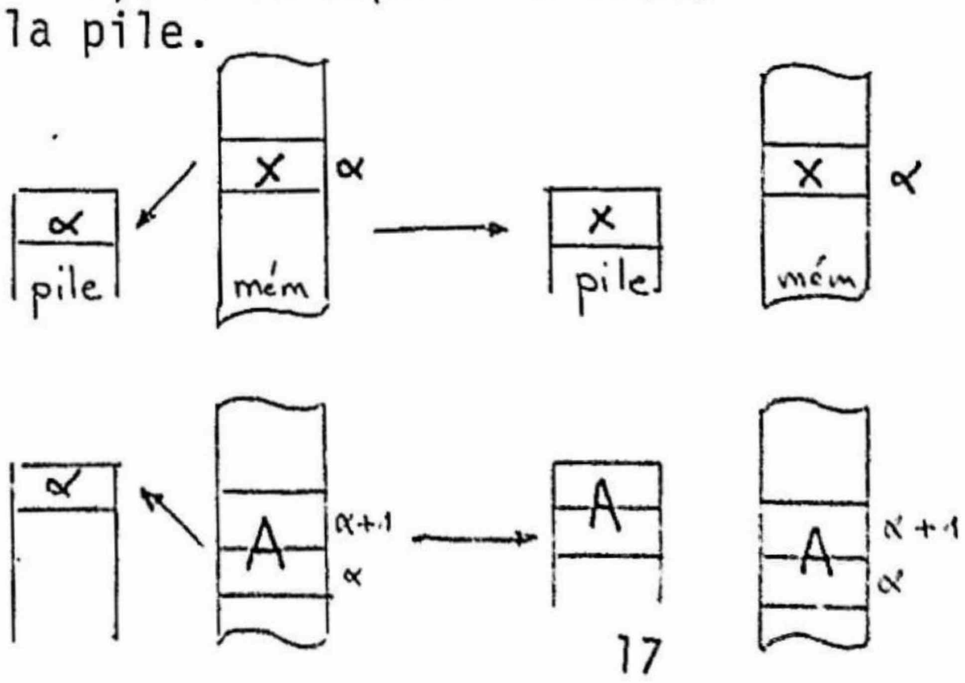
□ 6.2 UTILISATION DES VARIABLES

Les deux opérations de base portant sur les variables sont:

. transférer le contenu du sommet de la pile dans une variable
. prendre le contenu d'une variable et le déposer sur la pile.

L'appel d'une variable se fait comme pour les procédures simplement en donnant un nom. L'appel d'une variable a pour effet de déposer sur la pile l'adresse mémoire de celle-ci; on peut ensuite en utilisant la procédure adéquate soit amener sur la pile la valeur se trouvant à cette adresse (c'est-à-dire dans cette variable), soit déposer à cette adresse ce qui se trouve sous le sommet de la pile.

&	Remplace l'adresse au sommet de la pile par le mot (16 bits) se trouvant à cette adresse
F&	Remplace l'adresse au sommet de la pile par les deux mots se trouvant à l'adresse et à l'adresse+1.



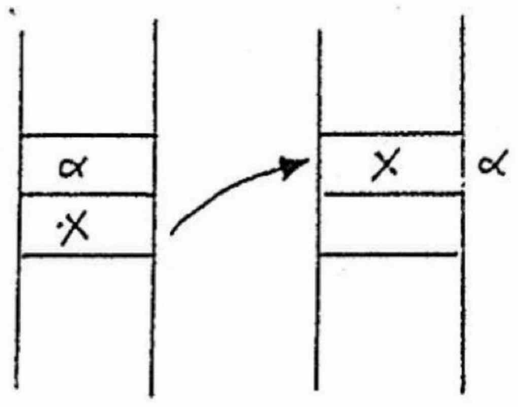
!
F!
B&
B!

Dépose la valeur sous le sommet de la pile à l'adresse indiquée par le sommet

Comme ! mais avec les deux mots sous le sommet.

comme & , mais sur 1 byte

comme ! , mais sur 1 byte



EXEMPLES:

```
569 INTEGER AA↵          (définit une variable entière contenant 569)
AA↵                      (dépose l'adresse AA sur la pile)
&↵                       (met sur la pile le contenu de AA)
.↵                       (et l'imprime)
569

33 INTEGER BB↵
777↵                     (777 sur la pile)
BB↵                      (adresse BB par dessus)
!↵                       (met le 777 dans BB)
BB & .↵                  (on vérifie que BB contient bien 777 maintenant)
777

AA & 31 +↵               (contenu de AA + 31)
BB !↵                    (dans BB)
BB & .↵
600                      (nouvelle valeur de BB)
```

Les variables peuvent être utiles lorsqu'il devient difficile de manipuler la pile parce qu'elle contient trop de données à utiliser en même temps.

+!

Ajoute le nombre sous la pile au contenu du mot mémoire dont l'adresse se trouve au sommet de la pile

```
AA & .↵
569

11 AA +!↵                (additionne 11 à AA)
AA & .↵
580


0.0 FLOATING MAXI 0.0 FLOATING PAS
: TAB↵                   (tabulation d'une fonction sinus avec possibilité
                          d'indiquer les limites et le pas)
PAS F! MAXI F!↵          (les deux nombres sur la pile sont le pas et le max.)
BEGIN FDUP MAXI F& F<↵   (teste si la valeur est plus grande que max.)
IF FDUP E. FDUP SIN E. CR↵ (si non calcule et imprime)
PAS F& F+ WHILE FDROP ;↵ (incrémente le nombre sur la pile et continue )
0.0 6.3 0.5 TAB↵

0.00000E 00  0.0000E 00
0.50000E 01    ...
```

TABLEAUX

N ARRAY nom	définit un tableau d'entiers de N éléments. Les éléments du tableau ne sont pas initialisés.
N FARRAY nom	définit un tableau de réals de N éléments.

◻ 6.3 UTILISATION DES TABLEAUX

Pour obtenir l'adresse d'un élément d'un tableau, il faut déposer sur le pile le numéro de l'élément désiré, puis donner le nom du tableau; on peut ensuite utiliser les procédures ! et & pour les ARRAYs et F! et F& pour les FARRAYs.

EXEMPLES:

```
5 ARRAY TB↵          (définit un tableau de 5 éléments entiers)
333 1 TB !↵          (met 333 dans le premier élément du tableau)
299 4 TB↵            (met 299 dans le quatrième élément du tableau)
1 TB & .↵            (imprime le contenu du premier élément)
```

333

```
: TBIMPRIME↵         (procédure pour imprimer tous les éléments de TB)
  5 1 DO↵            (boucle de 1 à 5, car il y a 5 éléments)
      I TB &↵        (prend le contenu du Ieme élément. I vaut 1, puis 2, 3...)
      . LOOP ;↵      (imprime et continue la boucle)
77 2 TB ! -56 3 TB ! 31888 5 TB !↵     (place des nombres dans TB)
TBIMPRIME↵
```

333 77 -56 299 31888

```
: TB/2 ↵             (procédure pour diviser par 2 tous les éléments de TB)
  5 1 DO ↵
      I TB & 2 / I TB !↵   (divise l'élément par 2 et remplace par le résultat)
      LOOP ;↵
TB/2 TBIMPRIME↵
```

166 38 -28 149 15944

```
: IMP100↵            (imprime les éléments du tableau qui sont supérieurs à 100)
  5 1 DO I TB & 100 >↵
          IF I TB & . THEN↵
      LOOP ;↵
IMP100↵
```

166 149 15944

REMARQUE:

. Si à l'intérieur d'une boucle DO ... LOOP on appelle une procédure, il n'est pas possible d'utiliser I dans cette procédure. Si on a besoin de l'indice de la boucle dans la procédure en question, il faut soit le déposer dans la pile avant d'appeler cette procédure, soit le stocker dans une variable de façon à ce que la procédure puisse l'atteindre.

EXERCICES:

1) Ecrire une procédure qui calcule le carré d'un nombre, mais seulement si le nombre est plus petit que 1000 en valeur absolue.

2) Même problême qu'avant, mais cette fois on imprime un message indiquant que le nombre n'a pas été mis au carré.

3) A partir d'un nombre sur la pile, imprimer le double dudit nombre, puis le double du double, et ainsi de suite jusqu'à ce que le dernier nombre imprimé soi plus grand que 2000.

4) Ecrire une procédure qui trouve tous les diviseurs d'un nombre donné et les imprime (A est diviseur de B, si B modulo A = 0).

5) Ecrire une procédure qui cherche le plus petit élément d'un AARAY.

6) Ecrire une procédure qui donne l'indice du premier élément d'un ARRAY qui est plus petit que 10 ou imprime un message s'il n'y en a pas.

7) Ecrire une procédure qui classe en ordre croissant les éléments d'un ARRAY (tri par échange ou tri pas bulles).

8) Ecrire une procédure qui dessine une fonction sur le terminal à l'aide de * .

9) Ecrire des procédures de calcul matriciel: addition de matrices, multiplication par un scalaire et multiplication matricielle.

(EXEMPLE: créer une matrice 7,4:

```
28 FLOATING M1↵
: MT1 1 - SWAP 1 - 4 * + M1 ;↵
6 2 MT1 F& ... 7.283 3 3 MT1 F!↵
```

7 TRAITEMENT DES CARACTERES

7.1 CODE ASCII DES CARACTERES

A chaque caractère alphanumérique existant correspond un nombre qui est utilisé pour représenter ce caractère dans la mémoire totalement numérique de l'ordinateur. Ce nombre est le code ASCII du caractère

CODES ASCII (octal)							
	40	Ø	60	@	100	P	120
!	41	1	61	A	101	Q	121
"	42	2	62	B	102	R	122
#	43	3	63	C	103	S	123
$	44	4	64	D	104	T	124
%	45	5	65	E	105	U	125
&	46	6	66	F	106	V	126
'	47	7	67	G	107	W	127
(	50	8	70	H	110	X	130
)	51	9	71	I	111	Y	131
*	52	:	72	J	112	Z	132
+	53	;	73	K	113	[	133
,	54	<	74	L	114	\	134
-	55	=	75	M	115	]	135
.	56	>	76	N	116	↑	136
/	57	?	77	O	117	←	137

Ainsi, si on tape sur la touche C d'un clavier, la machine recevra le nombre 103_8. Incersément, si l'ordinateur envoie le nombre 107_8, l'écran affichera un G .

On remarque que les codes ASCII sont dans un ordre relativement logique, c'est-à-dire qui correspond à l'ordre alphanumérique habituel.
De ce fait, si l'on veut trier dans l'ordre alphabétique une série de caractères, il suffit de trier leurs codes ASCII dans l'ordre numérique.

WCH	imprime le caractère qui est sur la pile

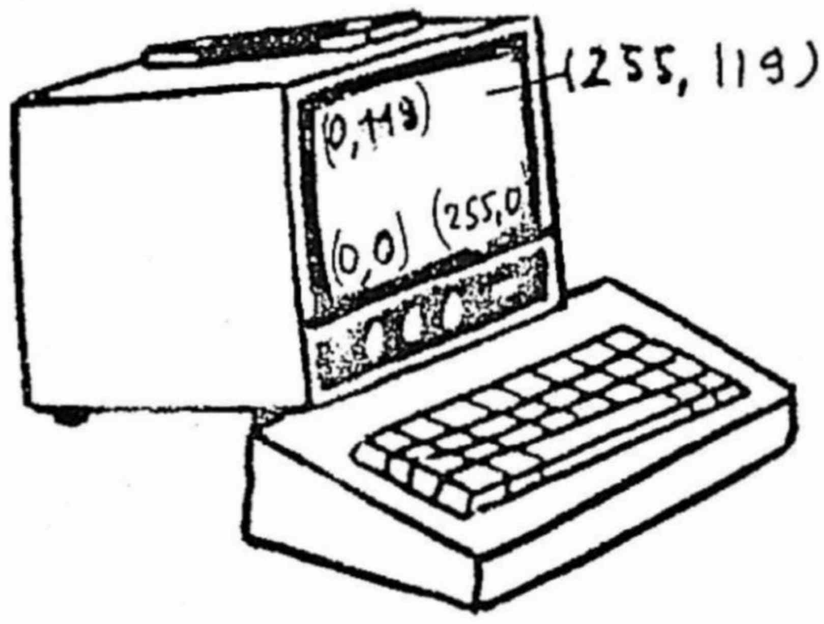
■ 8.1 GRAPHISME SUR L'ECRAN SMAKY6

Le software graphique du FORTH est un ensemble de procédures permettant de dessiner sur l'écran.

L'écran peut travailler sous deux modes:

- mode alphanumérique: les coordonnées transmises par l'ordinateur sont affichées sous forme de lettres et de chiffres
- mode graphique: les données reçues sont considérées comme les coordonnées d'un point de l'écran.

Les coordonéées de l'écran graphique du SMAKY6 s'étendent de 0 à 255 en X et de 0 à 119 en Y.



PROCEDURES GRAPHIQUES

I CMOD	initialise le mode graphique I (I = 0 effacement I = 4 — — — I = 1 trait normal I = 5 — · — : — I = 2 — I = 6 + + + + I = 3 · · · ·)
AG-DISP	allume l'écran graphique et l'écran alphanumérique
A-DISP	allume l'écran alphanumérique
G-DISP	allume l'écran graphique
NEW	initialise un nouveau dessin (le prochain point tracé ne sera pas relié au précédent)

□ 8.2 DEFINITION DE L'ESPACE DE TRAVAIL SUR LE TERMINAL

xmin xmax ymin ymax VIEWPORT nombres entiers	détermine la portion de l'écran qui va être utilisée pour le dessin

EXEMPLE:

Si on ne veut dessiner que dans la moitié gauche, on prendra

0 127 0 119 VIEWPORT

Tout segment dont une extrémité se trouve en dehors des limites ne sera pas dessiné.

Après avoir fini un dessin, on peut redéfinir un nouveau VIEWPORT ailleurs sur l'écran.

8.3 DEFINITION DES COORDONNEES DE L'UTILISATEUR

`rxmin rxmax rymin rymax WINDOW` — définit le système de coordonnées propres à l'utilisateur.

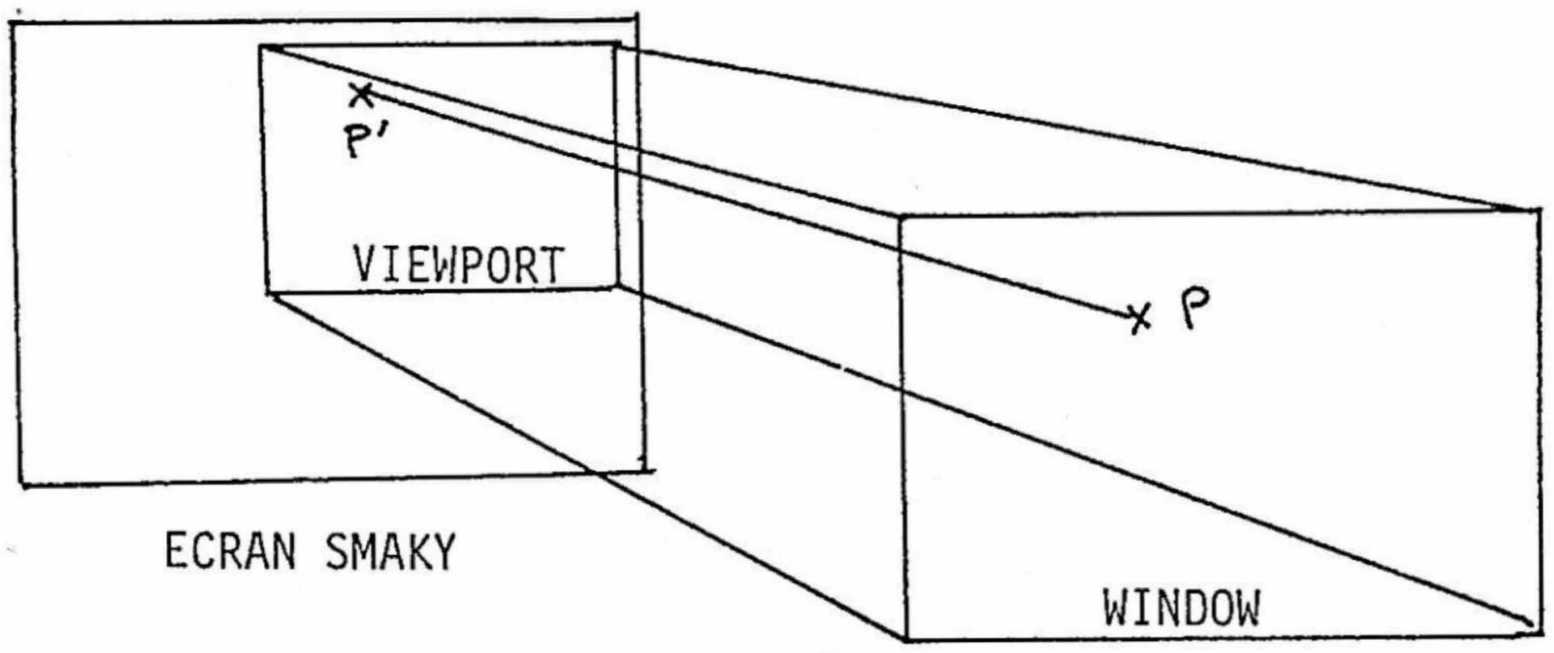
L'utilisateur peut choisir le système qui convient le mieux au problème qu'il a à traiter.

S'il veut, par exemple, traiter la fonction $f: x \longrightarrow x^3$ pour x allant de -3 à +2, un bon choix serait de prendre

```
-3. 2.-30. 10. WINDOW
```

Si on ne sait pas à priori quel est l'expace nécessaire pour contenir le dessin, on choisir un WINDOW assez grand, quitte à le réduire par la suite.

SCHEMA DE LA RELATION ENTRE VIEWPORT ET WINDOW



A tout point exprimé en coordonnées de l'utilisateur correspond un point P' sur l'écran dont les coordonnées sont calculées par les différentes procédures graphiques. L'utilisateur travaille toujours dans ses propres coordonnées et c'est le software graphique qui se charge de les convertir en coordonnées sur l'écran graphique.

DESSIN:

`x y DRAW` — relie par un segment le dernier point tracé avec le point de coordonnées (x,y).
x et y sont des nombres réels représentant les coordonnées de l'utilisateur du point.

```
n FARRAY AX a FARRAY AY
1 AX a AY n CURVE
```

Relie les points dont les coordonnées sont contenues dans AX pour les x et AY pour les y, n indiquant le nombre de points à relier (n ne doit évidemment pas excéder la dimension des FARRAYs).

AUTRES PROCEDURES

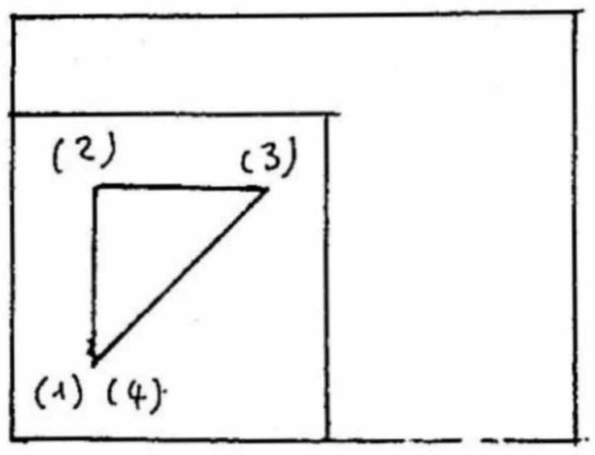
`PAGE` — efface l'écran

`x y DRAW` — comme DRAW avec les coordonnées physiques (entières)

EXEMPLES:

1) Dessiner un triangle de base 3 et dont l'angle inférieur gauche est situé à l'orgine des coordonnées

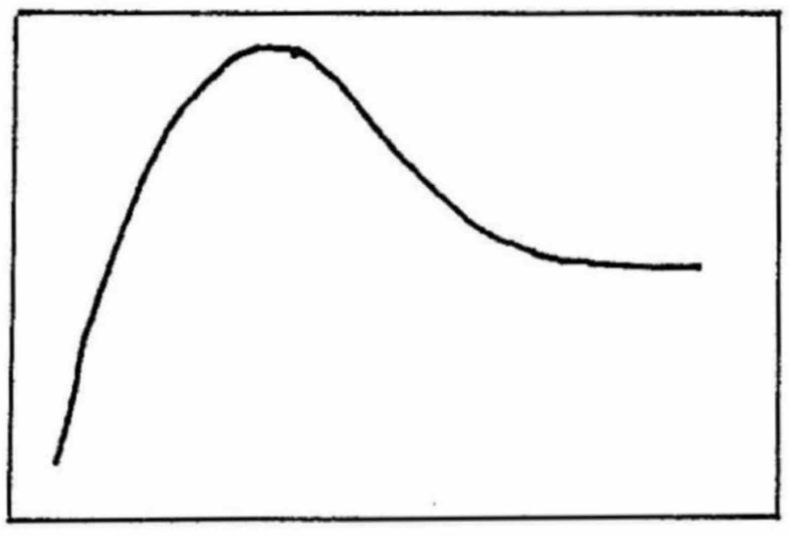
```
PAGE 1 CMOD ↙
AG-DISP↙
0 100 0 50 VIEWPORT↙          (on se donne un espace de dessin carré)
-1. 4. -1. 4. WINDOW↙         (espace utilisateur)
 NEW↙
 0. 0. DRAW 0. 3. DRAW 3. 3. DRAW 0. 0. DRAW G-DISP↙
```



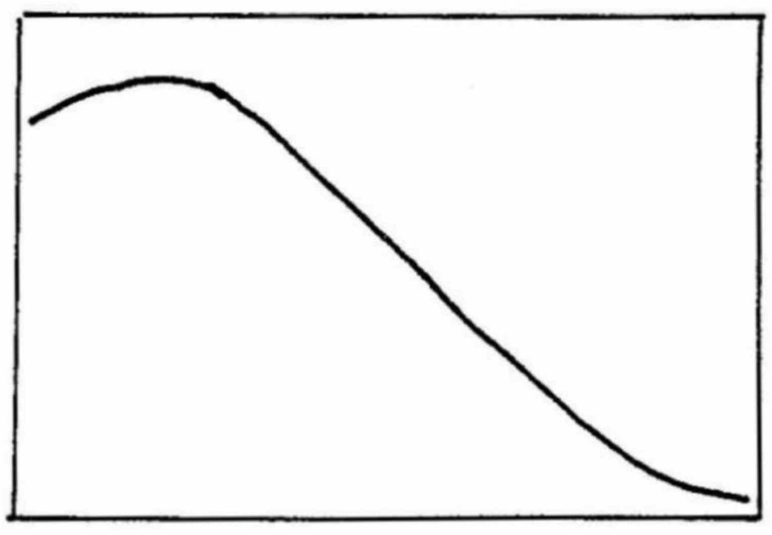
2) Trace la fonction f(x): x exp(-x)sin(x) de -0.5 ã 10.

```
: FONCTION FDUP FMINUS EXP FSWAP SIN F* ;↙
: DESSIN PAGE 1 CMOD G-DISP↙
  0 255 0 119 VIEWPORT↙                    (prend tout l'écran)
  -1. 5.5 -0.9 0.4 WINDOW NEW 0.5↙
 NEW -0.5↙
 BEGIN↙
    FDUP FDUP FONCTION DRAW ↙
    0.1 F+ FDUP 5. F>↙
END AG-DISP ;↙

DESSIN ↙
```



-1. 5.5 -0.9 0.4 WINDOW



0.5 3. 0. 0.4 WINDOW

EXERCICES:

1) Faire un programme qui dessine une étoile

2) Dessiner la fonction x.sin(1/x) de -10. à 10.

3) Ecrire un programme qui dessine un cercle ou une ellipse centrée à l'origine.

■ 9 LA RECURSIVITE ET SON EMPLOI EN FORTH

On dit qu'une fonction est récursive si elle est définie à partir d'elle-même.

Par exemple, la fonction factorielle,

$$N! = N*(N-1)*(N-2)*...*3*2*1$$

est telle que $N! = N*(N-1)$ si $N > 0$.

Autrement dit, on a une fonction F définie ainsi

$$F(n) = \begin{cases} 1 & \text{si } n=0 \\ n*F(n-1) & \text{sinon} \end{cases}$$

□ . 9.1 LA SUITE DE FIBONACCI

Cette suite est définie de la manière suivante:

Chaque nombre de la suite vaut la somme des deux précédents, ce qui donne 1,1,2,3,5,8,13,21,34,55,...

La fonction F(n) qui donne le $n^{ième}$ terme de la suite est définie par:

$$F(n) = \begin{cases} 1 & \text{si n est inférieur à 3} \\ F(n-1)+F(n-2) & \text{sinon} \end{cases}$$

Nous sommes donc en présence de deux fonctions récursives.
Voyons comment les programmer en FORTH.

```
: FACTORIELLE ①
  DUP 0 = IF ↙                          (test si N=0)
            DROP 1 ↙                    (dans ce cas, le résultat est 1)
          ELSE ↙                        (sinon)
            DUP ↙
            1 - FACTORIELLE ↙           (calcule (N-1)!
          ② *                           (puis N*(N-1)!
          THEN ; ↙
```

Voyons maintenant ce qui se passe lorsqu'on calcule 4!

PROGRAMME		PILE
FACTORIELLE		
(1) 4≠0	FACTORIELLE	4
(1) 3≠0	FACTORIELLE	4 3
(1) 2≠0	FACTORIELLE	4 3 2
(1) 1≠0	FACTORIELLE	4 3 2 1
(1) 0=0	1 sur pile et ;	4 3 2 1 0
(2) * ;		4 3 2 1 1
(2) * ;		4 3 2 1
(2) * ;		4 3 2
(2) * ;		4 6
FIN		24

Calcul du $n^{ième}$ terme de la suite:

```
: FIBANACCI DUP 3   IF DROP 1↵
                    ELSE DUP L - FIBANACCI ↵
                         SWAP 2 -F FIBONACCI ↵
                         + ↵
                    THEN ; ↵
```

REMARQUE VITALE:

Une fonction récursive doit toujours présenter au moins un cas trivial c'est-à-dire qu'il doit exister une valeur pour laquelle le calcul de la fonction puisse être fait sans appler une nouvelle fois la procédure.

Par exemple N = ¼ donne immédiatement le résultat 1 pour N! ..
Sinon le processus de calcul serait infini.

□ 9.2 PROBLEME DE LA TOUR DE HANOI

Soit trois piquets et une série de disques de rayon décroissant et percés d'un trou en leur centre empilés sur le 1er piquet.

Le jeu consiste à transférer tous les disques du piquet A sur le piquet B tout en respectant les deux règles suivantes:

- on ne peut déplacer qu'un disque à la fois
- on doit obligatoirement déposer ce disque sur un disque ayant un rayon supérieur au sien.

Après une intense réflexion, on se dit qu'il va falloir par un moyen quelconque mettre tous les disques de A sur C sauf le dernier qui ira sur B, puis ramener les disques de C sur B. Reste à savoir comment on va déplacer les N-1 disques de A vers C. Il suffit en fait d'utiliser le même raisonnement que pour A vers B, mais avec un disque de moins. De même pour C vers B.
On peut alors généraliser cette méthode en considérant une fonction D(d,a,t,n) où d est le piquet de départ, a le piquet d'arrivée, t le piquet de transition et n le nombre de disques à déplacer.

D peut alors se définir ainsi:

si n≠1 D(d,a,t,n) = D(d,t,a,n-1) ①
= D(d,a,t,1) ②
= D(t,a,d,n-1) ③

Le cas trivial se produit pour n=1 (prendre le disque au sommet de d et le poser sur a).

□ 9.3 UTILISATION DES VARIABLES POUR LES PROCEDURES RECURSIVES

Si une procédure récursive utilise des variables, il faut se souvenir que celles-ci seront modifiées à chaque passage dans la procédure.

Supposons qu'une procédure P soit construite de la manière suivante:

: P ... A ! ... P ... A & ... ;

il y a de fortes chances qu'au moment où l'on fait A & on ne retrouve pas ce qu'on a mis dans A avant d'appeler P car cet appel à P a eu bien entendu pour effet de faire exécuter P donc la séqeunce A !
ce qui aura mis quelque chose dans A, détruisant ce qui y était précédemment. Pour remédier à ce contretemps, il faut avant d'appeler P "sauver", c'est-à-dire déposer le contenu de A sur la pile (en dessous des éléments utilisés par P), puis au retour reprendre cette valeur de la pile et la remettre dans A.

10 EDITEUR ET PROCEDURES FORTH NON STANDARD

Les commandes de l'éditeur sont les mêmes que celles du programme SMILE, sauf pour les commandes relatives à la touche (PROGRA).

Pour compiler le programme situé dans le buffer courant, faire (PROGRA) (C)

* S'il y a des fautes de compilation, toutes les procédures qui viennent d'être compilées sont enlevées du dictionnaire, et les erreurs sont indiquées dans le programme par des " ↑ ERROR".

EXEMPLE:

```
: ADD + PRINT ;        (la procédure PRINT n'existe pas)
              ↑ ERROR
```

Pour chercher une erreur, faire (SEARCH) (^)

Lorsqu'une faute est corrigée et que le programme est à nouveau compilé, les messages d'erreur sont supprimés.

** Si la compilation est correcte, on passe automatiquement en mode interpréteur, ce qui permet d'essayer le programme.

Avant de passer au mode interpréteur, la pile est vidée.

PROCEDURES FORTH NON STANDARD

LOAD	compile le texte se trouvant dans l'éditeur. Les erreurs sont indiquées au fur et à mesure par ...? et un coup de buzzer. Lorsque la compilation est terminée, la pile n'est pas vidée. Les procédures compilées sont de toutes façons ajoutées au dictionnaire, même s'il y a des fautes.
EDIT	Appel de l'éditeur
. E. WCH	impression d'un entier impression d'un flottant impression d'un caractère
% E%	lecture d'un entier lecture d'un flottant
NOOK	variable d'un byte contrôlant l'impression du message OK. 1 NOOK B! pas de OK 0 NOOK B! ok
WORD	lecture d'un mot. La longueur est stockée dans ID (byte) et les caractères suivants dans WD,WD+1,...,WD+N
X Y **	X puissance Y (flottants) X > 0