INITIATION AU LANGAGE FORTH - VOLET 1

Olivier SINGLA

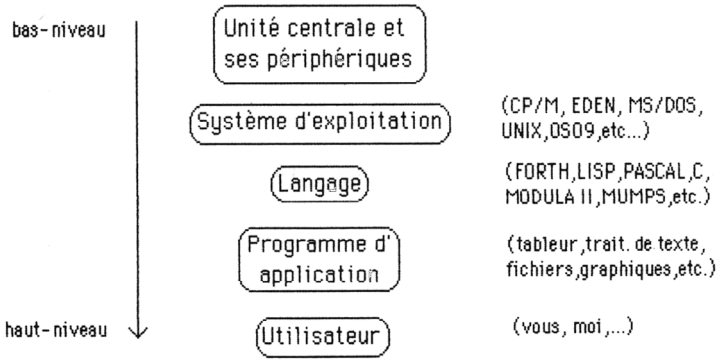
I- PRÉLIMINAIRE

L'objet de cette série d'articles consacrés à l'initiation au FORTH n'a pour objectif que de vous aider dans l'appropriation de ce fabuleux langage, et principalement au travers de la pratique de la version de FORTH disponible sur les machines 8 bits z80 agrées par l'Éducation nationale. Nous renvoyons les lecteurs au numéro de l'EPI de décembre 84 pour les généralités sur le langage FORTH, et une description rapide de *zFORTH* (qui devrait donc être distribué par le CNDP quand cet article paraîtra). Cette série d'articles s'adresse à tout le monde, mais nous supposerons néanmoins que les lecteurs potentiels ont une expérience de la programmation (quel que soit le langage), et qu'ils connaissent la signification de termes tels que : bits, octets, adresse-mémoire, etc. En outre, pour ceux qui disposeraient de *zFORTH* et de son manuel de référence, nous les invitons fermement à lire et à pratiquer le chapitre IV ("Notes pour travailler un programme sous *zFORTH").* Ceci étant dit, lançons-nous dans l'aventure sans plus attendre I

2- LES DIFFÉRENTES COUCHES DE LOGICIELS

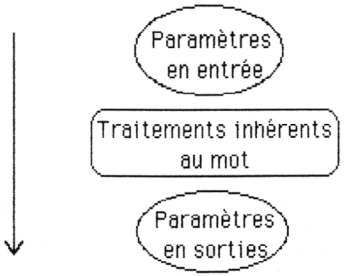
Il est très important de situer le langage par rapport aux autres logiciels pouvant cohabiter dans la machine. Nous parlerons donc en terme de **Couches- de Iogiciels ,** représentées par le shéma suivant :

Bien sûr, dans chaque élément, nous pourrons retrouver plusieurs couches de logiciels. C'est le cas par exemple pour un système d'exploitation tel que CP/M avec 4 couches de logiciels (TPA, CCP, BIOS et BDOS), pour un langage ou un programme d'application.



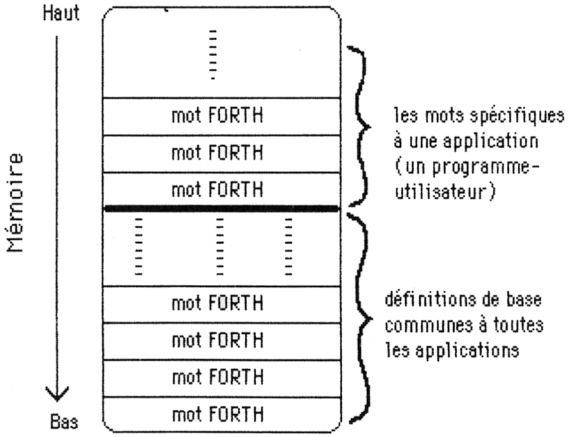
3- LES COMPOSANTES DE BASE DE FORTH

Le premier concept à définir est celui de MOT (encore appelé DEFINITION). Un mot FORTH correspond à une parcelle de programme. Nous pourrons le définir par le schéma suivant :



Un "programme" FORTH sera donc constitué par un ensemble cohérent et ordonné de plusieurs définitions ou mots. Les définitions FORTH sont regroupées dans ce qui est appelé le DICTIONNAIRE . On peut représenter le dictionnaire FORTH de la manière suivante :

On notera que les nouvelles définitions viendront s'ajouter en haut du dictionnaire : celui-ci croît donc vers le haut de la mémoire. Nous pouvons déjà faire la distinction entre 2 couches de mots : la couche des mots disponibles après l chargement de votre version du langage FORTH, et la couche des mots que vous pourrez définir et qui constitueront votre ou vos programmes. La première couche contiendra des mots standards que l'on retrouvera quelle que soit la version de FORTH.

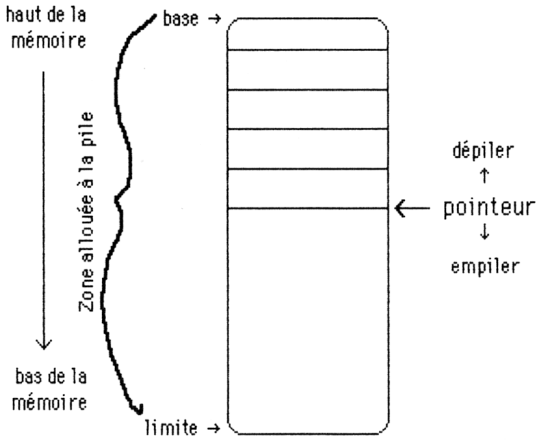


On peut remarquer une analogie certaine avec des langages procéduraux tels que PASCAL, LSE ou LOGO (un programme = un ensemble de procédures).

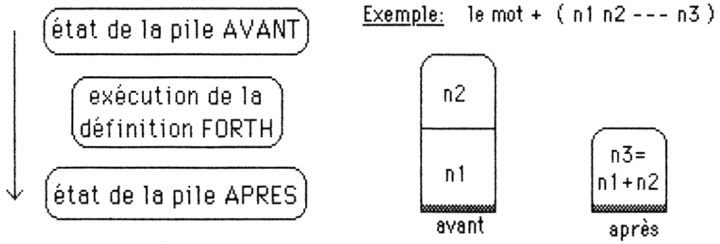
La première différence vient dans la définition elle-même d'un mot FORTH : au lieu d'utiliser un jeu d'instructions figé, avec une syntaxe spécifique pour chaque instruction, on fera simplement référence à d'autres mots FORTH, déjà existants... Ainsi, nous pouvons dire, shématiquement, qu'une définition FORTH est constituée de références à un ensemble d'autres définitions FORTH. Première conséquence : on peut dire que FORTH est un langage globalement sans syntaxe, dans la mesure où les mots référencés sont tapés librement, avec au moins un espace séparateur (pour que le système FORTH puisse les isoler). Si syntaxe il y a à respecter, cela ne dépend que du mot alors invoqué... Deuxième conséquence : une définition FORTH ne pourra faire référence qu'à des mots FORTH existants (déjà définis) lors de la création de cette définition.

La deuxième différence vient dans la manière de passer les paramètres entre les différents mots. Est privilégiée l'emploi d'une pile, appelée PILE DES PARAMÈTRES . Comment définir une pile ? Disons que c'est une structure de donnée à partir de laquelle deux types d'accès sont possibles **empiler** un objet ou **dépiler** un objet. La pile est utilisé par tous les langages de programmation (exemple : en BASIC, chaque fois qu'un GOSUB est rencontré, est empilée sur la pile de retour l'adresse de retour au programme appelant, et chaque fois que RETURN est rencontré, est alors dépilée l'adresse de retour). Le FORTH sur lequel nous allons travailler étant implanté sur un micro-ordinateur architecturé autour d'un microprocesseur 8 bits avec une capacité d'adressage de

64 Koctets (compteur ordinal sur 16-bits), les objets manipulés sur la pile seront, de manière générale, des MOTS de 16-bits. Décrivons la structure et le fonctionnement de la pile des paramètres en FORTH grâce au shéma suivant :



La plupart des mots FORTH seront ainsi définis par rapport à la pile. Cela peut être illustré par le schéma suivant



Nous pouvons remarquer comment est présentée, de manière symbolique, l'action d'un mot FORTH sur la pile des paramètres : (état avant --- état après). Par convention, n désignera un entier codé sur 16-bits (soit un entier compris entre -32768 et +32767, devinez pourquoi...)

4.- OPÉRATIONS ÉLÉMENTAIRES

Nous avons défini dans le chapitre précédent le mot + : celui-ci dépile 2 entiers 16-bits, effectue leur somme, et empile alors le résultat. Le mot. (point, mais prononcez "dot") affichera alors ce nombre sur le terminal.

123 -10 + . 113 ok

Expliquons brièvement les mécanismes mis en jeu lorsque nous validons après avoir tapé ceci : chaque mot est isolé, l'espace étant considéré comme le séparateur, puis est recherché dans le dictionnaire. Si le mot est trouvé, son exécution est alors invoquée, mais si la recherche a échoué, le système tente une conversion ASCII->binaire, par rapport à la base courante (pour le moment la base 10). Si cette conversion s'avère impossible, une erreur est générée (« mot non trouvé et nombre Incorrect »), sinon la valeur trouvée est empilée sur la pile des paramètres. Le schéma suivant illustre l'algorithme utilisé par l'interprète en état exécution (pour chaque mot isolé) :

```
si trouvé alors
  |  l'exécuter
sinon
  |  tenter la conversion ASCII→binaire par rapport à la base courante
  |  si échec alors
  |  |  générer 'erreur système #0
  |  sinon
  |  empiler la valeur
```

Nous invitons le lecteur à se référer au manuel de référence de *zFORTH* ("Glossaire rapide", en annexe 0 et à expérimenter, plus particulièrement, les mots suivants :

- arithmétique en simple-précislon
- entrées-sorties
- opérations logiques
- manipulations de piles

5- VARIABLES ET CONSTANTES NUMÉRIQUES

FORTH offre, comme la plupart des langages de haut-niveau, la possibilité de manipuler des variables et des constantes par un nom symbolique, choisi par le programmeur. Par exemple, pour créer une constante numérique de nom LONG et de valeur 365, nous taperions ceci :

```
365 CONSTANT LONG
```

Ceci a pour conséquence de créer une définition de nom `LONG` dans le dictionnaire. On peut d'ailleurs vérifier par `WORDS` qui va afficher les mots contenus dans le dictionnaire (en commençant par

les derniers définis). Le fait d'invoquer LONG amènera alors la valeur 365 sur la pile.

Pour créer une variable de nom AGE : VARIABLE AGE

Une définition de nom AGE est alors crée dans le dictionnaire, et son invocation amènera sur la pile, non la valeur de la variable, mais l'adresse de cette valeur. Deux mots vont nous permettre de déposer une valeur à cette adresse, ou à l'inverse de la lire. Ce sont :

@ (fetch) (adr --- n) et ! (store) (n adr ---)

Par exemple :

```
24 AGE !        ( initialise la variable AGE à la valeur 24 )
AGE @ .         ( lit la valeur de AGE et l'affiche )
```

De ceci, nous pouvons tirer 2 conclusions importantes...

La première est que tout en manipulant sur la pile des valeurs entières codées sur 16-bits, l'utilisation qui en faite par différents mots permet de les considérer comme des valeurs, des adresses mémoire, etc. Remarquons au passage que bien que manipulant l'adresse de la variable, nous pouvons parfaitement ignorer où elle se situe physiquement en mémoire : les mots @ et ! rendent transparents les accès à une variable.

La deuxième est que, à partir d'un mot FORTH (ici VARIABLE et CONSTANT), nous avons crée d'autres mots dans le dictionnaire. Les mots VARIABLE et CONSTANT seront appelés des MOTS GÉNÉRATEURS. FORTH offre ainsi la possibilité remarquable de se définir ses propres mots générateurs. Pour cela, il faut donner une réponse aux 2 questions suivantes :

- Que faire lors de la création du mot dans le dictionnaire ?
- Que faire lors de l'invocation des mots générés ?

Ceci nous permet de dire que le contexte FORTH nous permet d'écrire un compilateur (ou plutôt des compilateurs...) de manière très simple, en éliminant tous les problèmes syntaxiques qui peuvent se poser dans un langage compilé traditionnel.

CRÉONS D'AUTRES MOTS FORTH

Supposons que nous désirions créer un mot FORTH qui calcule le complément à 100 d'une valeur donnée sur la pile. La définition en serait la suivante :

```
: COMPLEMENT    ( n 1 --- n2 )
  100 SWAP - ;
```

Nous vous conseillons de regarder dans le manuel de référence de *zFORTH* le fonctionnement des mots suivants : `( SWAP - ;`

Si vous ne comprenez toujours pas, laissez tomber le FORTH et revenez à votre LSE chéri...

Le rôle du : est de créer l'entête du mot COMPLEMENT dans le dictionnaire (avec le chaînage aux mots précédents), puis de faire basculer le système en état COMPILATION. Résumons le fonctionnement de l'interprète FORTH en état compilation (pour chaque mot qui a été isolé) :

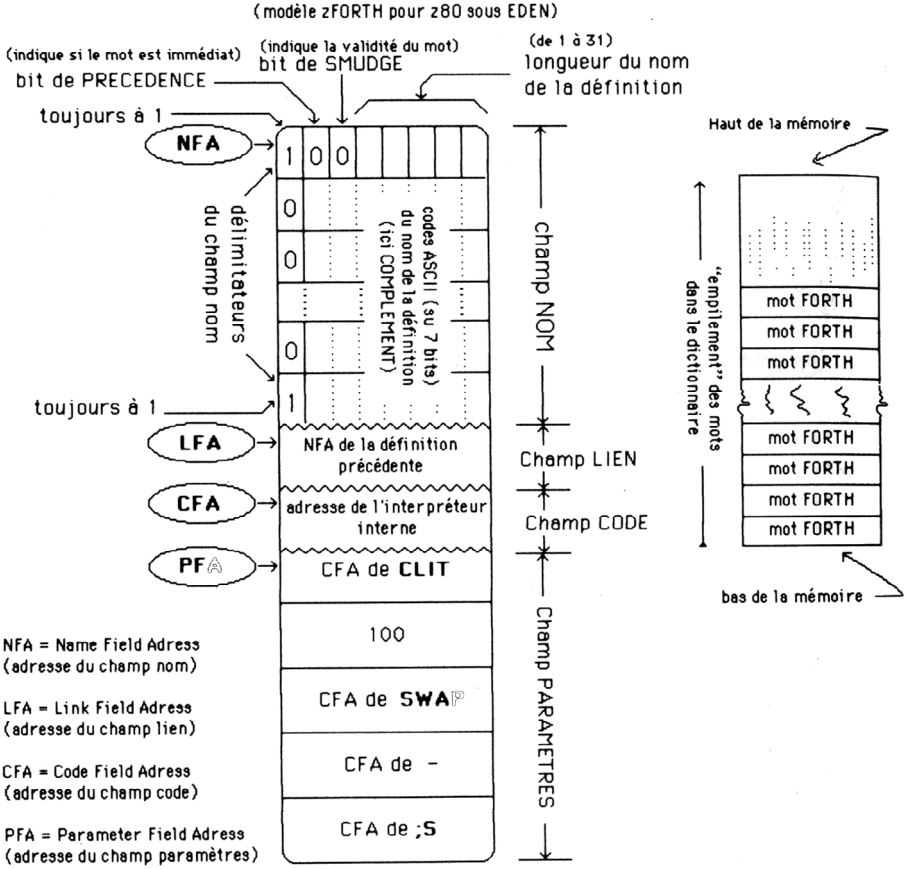
```
rechercher le mot dans le dictionnaire
si trouvé alors
  |  si mot immédiat alors
  |  |  l'exécuter
  |   sinon
  |    "compiler" son adresse au sein de la définition encours de création
sinon tenter la conversion en binaire (ASCII->binaire)
si succès alors
  |  compiler" CLIT, LIT ou DLIT et la valeur sur 8, 16 ou 32 bits
sinon
  |  générer l'erreur-système #0 ("mot non trouvé et nombre invalide")
```

Analysons, à l'aide du schéma suivant, la structure des mots dans le dictionnaire, et le code généré par le compilateur FORTH :

Nous ne saurions trop vous engager de disséquer soigneusement ce petit programme sans prétention, manuel de référence de zFORTH et "glossaire rapide" en mains...

Voir pages suivantes →

Structures des mots dans le dictionnaire



Définition... : COMPLEMENT
100 SWAP - ;

(Dans cet exemple, l'interpréteur interne est l'interpréteur de mots FORTH de haut-niveau, soit la procédure d'exécution compilée par :)

7. Un exemple de programme en FORTH

Nous allons donner un exemple de programme écrit en zFORTH. Il s'agit du fameux jeux des allumettes. On fixe le nombre initial d'allumettes, l'option gagnante, le nombre minimum et maximum d'allumettes à retirer.

écran #1

```
 0 :  VARIABLE #nombre      \ nbre d'allumettes initial
 1 :  VARIABLE #pile        \ nbre d'allumettes restant
 2 :  VARIABLE #option      \ option gagnante (0 ou 1)
 3 :  VARIABLE #mini        \ nbre minimum d'allumettes à retirer
 4 :  VARIABLE #maxi        \ nbre maximum d'allumettes à retirer
 5 :  VARIABLE #begin       \ à 1 si le joueur commence, sinon à 0
 6 :  VARIABLE #joueur      \ no. du joueur en cours (0=humain, sinon 1)
 7 :  VARIABLE #prise       \ nbre d'allumettes retirées par chacun des 2 joueurs
 8 :  VARIABLE same         \ flag mis à 1 si on demande de fixer les règles
 9 :
10 :
11 :
12 :
13 :
14 :
15 :                                  -->
```

écran #2

```
 0 :  : titre
 1 :    " *** Jeu des Allumettes ***"  CLS
 2 :    80 OVER - 2/ SPACES  DDUP TYPE  UNDER CR
 3 :    80 OVER - 2/ SPACES "- CHARS  CR CR  ;
 4 :  : option?
 5 :    ." Gagne-t-on en évitant la dernière allumette ? "
 6 :    O/N? #option !  ;
 7 :  : nombre?  CR
 8 :    ." Avec combien d'allumettes part-on (de 15 à 60) ? "
 9 :    BELL GET/NBRE DUP
10 :    IFNOT  OVER 15 60 [WITHIN] 0= OR DUP IF UNDER ENDIF  ENDIF
11 :    IF  CR 11 EMIT 79 SPACES 11 EMIT MYSELF  ENDIF
12 :    #nombre !  ;
13 :  : begin?  CR
14 :    ." Désirez-vous jouer en premier ? " O/N? #begin !  ;
15 :                          -->
```

écran #3

```
 0 :  : mini?  CR
 1 :    ." Quel est le minimum à retirer (de 1 à 7) ? "
 2 :    BELL  GET/NBRE DUP
 3 :    IFNOT  OVER 1  7 [WITHIN] 0= OR DUP IF UNDER ENDIF  ENDIF
 4 :    IF CR 11 EMIT 79 SPACES 11 EMIT MYSELF  ENDIF
 5 :    #mini !  ;
 6 :  : maxi?  CR
 7 :    ." Quel est le maximum à retirer (de " #mini @  1+ .
 8 :    ." à 9) ? "  BELL  GET/NBRE DUP  IFNOT
 9 :      OVER #mini @ 1+  9 [WITHIN] 0= OR DUP IF UNDER ENDIF
10 :    ENDIF   IF CR 11 EMIT 79  SPACES 11 EMIT MYSELF ENDIF
11 :    #maxi !  ;
12 :  : paramètres?  same @0<>  IF
13 :      CR ." Fixons les règles du jeu, voulez-vous..." CR CR
14 :      option?  nombre?  mini?  maxi?
15 :    ENDIF   begin?   ;              -->
```

écran #4

```
 0 :  : mise-à-jour
 1 :    @CURS  #prise @ #pile -1
 2 :    7 0 !CURS  80 SPACES
 3 :    7 80 #pile @ - 2/ !CURS  #pile @ "I CHARS
 4 :    6 30 !CURS  #pile ?  !CURS   ;
 5 :  : page
 6 :    titre  ." On gagne en "
 7 :    #option @0<>  IF ." évitant" ELSE ." prenant" ENDIF
 8 :    ."  la dernière allumette..."  CR
 9 :    ." Minimum à retirer= " #mini ?  CR
10 :    ."  --- Maximum à retirer= " #maxi ?  CR CR
11 :    ." Nombre d'allumettes restant = "  #pile ?  CR
12 :    mise-àjour    ;
13 :
14 :
15 :                          -->
```

écran #5

```
 0 :  : (Xjeu)
 1 :    ." Combien retirez-vous d'allumettes ? "  BELL  GET/NBRE
 2 :    IF #prise 0! ELSE #prise ! ENDIF
 3 :    #prise @   1 #mini @ [WITHIN]  #prise #pile @= AND  IFNO
 4 :      #prise @  #mini @  #maxi @  [WITHIN]  0=  #prise @  #p
 5 :      OR  IF  CR 11 EMIT 79 SPACES 11 EMIT MYSELF  ENDIF
 6 :    ENDIF    ;
```

```
 7  :   : Xjeu
 8  :     10 0 !CURS ." A vous de jouer..."  25 SPACES (Xjeu)
 9  :     CR 2 11 CHARS  160 SPACES  2 11 CHARS
10  :     ." Vous venez donc de retirer "  #prise ?
11  :     ." allumettes..."  mise-à-jour   ;
12  :
13  :
14  :
15  :                          -->
```

écran #6

```
 0  :   : (Ojeu)
 1  :     3000 0  DO LOOP   \ on fait semblant de bosser dur...
 2  :     #pile @  #mini @ <=  IF #pile #prise @! EXIT ENDIF   #prise 0!
 3  :     #maxi @ 1+  #mini @  DO
 4  :       #pile @ 1 -  #mini #maxi @+  MOD
 5  :       #option @0=  IF  0=  ELSE  1 #mini  [WITHIN]  ENDIF
 6  :       IF  1 #prise !  ENDIF   \ hé hé, c'est une situation gagnante
 7  :     LOOP
 8  :     #prise @0=  IF 9 RND #prise !  ENDIF
 9  :     #prise @  #mini @  #maxi @  [WITHIN]  IFNOT MYSELF ENDIF
10  :     #prise @  #pile @  >  IF MYSELF ENDIF   ;
11  :   : Ojeu
12  :     14 0  !CURS  ." A moi donc de jouer..."  (Ojeu)   25 SPACES
13  :     CR  2 11 CHARS  160 SPACES  2 11 CHARS
14  :     ." Je viens donc de retirer "  #prise ?
15  :     ." allumettes..."   mise-à-jour      ;      -->
```

écran #7

```
 0  :   : diagnostic
 1  :     19 0 !CURS
 2  :     #joueur #option @=  IF
 3  :       ." Bravo, vous avez gagné !!"
 4  :     ELSE
 5  :       ." ha ha, je vous ai bien eu, avouez-le !!"
 6  :     ENDIF    ;
 7  :   : again?
 8  :     21 0 !CURS  ." Une autre partie ? "  0/N?  ;
 9  :   : same?
10  :     22 0 !CURS  ." On garde les mêmes règles ? "  0/N? 0= ;
11  :
12  :
13  :
14  :                          -->
15  :
```

écran #8

```
 0  :  : (jouer)
 1  :    titre paramètres?  #nombre #pile @!  #prise 0!  page
 2  :    #begin #joueur @!
 3  :    BEGIN
 4  :      #joueur #0<> IF Xjeu ELSE Ojeu ENDIF
 5  :      #joueur @0= #joueur ! \ un tour chacun, quoi...
 6  :    #pile @0= UNTIL
 7  :    diagnostic again?
 8  :    IF  same? same ! MYSELF ENDIF  ;
 9  :  : JOUER
10  :    same 1! (jouer)  ;
11  :
12  :  :S  \ That's all folks !!
13  :
14  :
15  :
```

Dans les prochains numéros de ce même bulletin, nous aborderons d'autres aspects du langage FORTH : machine-virtuelle FORTH, structures de contrôles, structures de données, structures des vocabulaires, gestion de la mémoire virtuelle, etc. Lancez-vous dans la programmation en FORTH, et vous redécouvrirez votre bon vieux Micral (ou Logabax ou Sil'z), et en outre, je pense que, comme moi-même, vous vous apercevrez que vous êtes plus efficace, à tous les points de vue (qualitatif et quantitatif).

Pour terminer, voici une bonne nouvelle pour tous les passionnés du langage FORTH dans l'Éducation nationale : je vais réaliser une implantation de FORTH sur MO5, dans le cadre du para-réseau. Cette version sera *opérationnelle dès la rentrée scolaire 85*, Elle tiendra compte de la mémoire auxillaire de 64K (organisée en 4 banques de 16K, commutables, et situées à la même adresse que la ROM BASIC). Seront intégrés les outils suivants : éditeur d'écrans-FORTH et éditeur de texte, macro-assembleur 6809, débugger, décompileteur FORTH, gestion d'overlays relogeables, virgule flottante. Les entêtes des mots FORTH seront séparés, ce qui permet de supprimer les entêtes des mots d'un programme terminé (pour ne garder que le code exécutable). En outre, ce FORTH sera en code direct chaîné, c'est-à-dire que le CFA (champ code) contiendra du code machine au lieu d'une adresse. Le fait d'éliminer ainsi une indirection fait gagner un temps appréciable à l'exécution. Je désire que ce FORTH soit un système de développement complet, relativement indépendant du matériel sur lequel il sera implanté. C'est d'ailleurs à

partir de ce FORTH que sera écrit le logiciel d'EAO OEOMETRIX, sur M05. Tout courrier (suggestions, etc...) sera le bienvenu !

Olivier SINGLA
Ecole Normale d'instituteurs
Bd. Montauriol
82000 MONTAUBAN