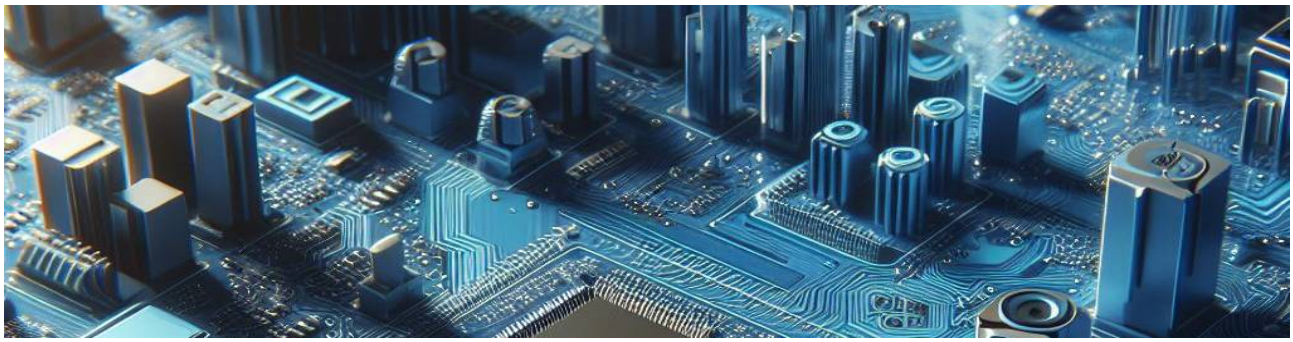
Le grand livre de eFORTH Windows

version 1.0 - 30 novembre 2023



Auteur

- Marc PETREMANN

Table des matières

Commentaires et mise au point

Il n'existe pas d'IDE[1] pour gérer et présenter le code écrit en langage FORTH de manière structurée. Au pire, vous utilisez un éditeur de texte ASCII, au mieux un vrai IDE et des fichiers texte :

- **edit** ou **wordpad** sous Windows
- **PsPad** sous windows
- **Netbeans** sous Windows...

Voici un extrait de code qui pourrait être écrit par un débutant :

```
: inGrid? { n gridPos -- fl } 0 { fl } gridPos getGridAddr for aft
getNumber n = if -1 to fl then then next drop  fl ;
```

Ce code sera parfaitement compilé par eForth Windows. Mais restera-t-il compréhensible dans le futur s'il faut le modifier ou le réutiliser dans une autre application ?

Ecrire un code FORTH lisible

Commençons par le nomage du mot à définir, ici `inGrid?`. Eforth Windows permet d'écrire des noms de mots très longs. La taille des mots définis n'a aucune influence sur les performances de l'application finale. On dispose donc d'une certaine liberté pour écrire ces mots :

- à la manière de la programmation objet en javaScript: `grid.test.number`
- à la manière CamelCoding `gridTestNumber`
- pour programmeur voulant un code très compréhensible `is-number-in-the-grid`
- programmeur qui aime le code concis `gtn?`

Il n'y a pas de règle. L'essentiel est que vous puissez facilement relire votre code FORTH. Cependant, les programmeurs informatique en langage FORTH ont certaines habitudes :

- constantes en caractères majuscules `LOTTO_NUMBERS_IN_GRID`
- mot de définition d'autres mots `lottoNumber:` mot suivi de deux points ;
- mot de transformation d'adresse `>date`, ici le paramètre d'adresse est incrémenté d'une certaine valeur pour pointer sur la donnée adéquate ;
- mot de stockage mémoire `date@` ou `date!`
- Mot d'affichage de donnée `.date`

1 Integrated Development Environment = Environnement de Développement Intégré

Et qu'en est-il du nommage des mots FORTH dans une langue autre qu'en anglais ? Là encore, une seule règle : **liberté totale** ! Attention cependant, eForth Windows n'accepte pas les noms écrits dans des alphabets différents de l'alphabet latin. Vous pouvez cependant utiliser ces alphabets pour les commentaires :

```
: .date     \ Плакат сегодняшней даты
    ….code…  ;
```

ou

```
: .date     \ 海報今天的日期
    ….code…  ;
```

Indentation du code source

Que le code soit sur deux lignes, dix lignes ou plus, ça n'a aucun effet sur les performances du code une fois compilé. Donc, autant indenter son code de manière structurée :

- une ligne par mot de structure de contrôle **`if else then`**, **`begin while repeat…`** Pour le mot if, on peut de faire précéder du test logique qu'il traitera ;
- une ligne par exécution d'un mot prédéfini, précédé le cas échéant des paramètres de ce mot.

Exemple :

```
: inGrid? { n gridPos -- fl }
    0 { fl }
    gridPos getGridAddr
    for
        aft
            getNumber n =
            if
                -1 to fl
            then
        then
    next
    drop
    fl
  ;
```

Si le code traité dans une structure de contrôle est peu fourni, le code FORTH peut être compacté :

```
: inGrid? { n gridPos -- fl }
    0 { fl }   gridPos getGridAddr
    for aft
            getNumber n =
            if -1 to fl  then
        then
```

```
    next
    drop  fl
  ;
```

C'est d'ailleurs souvent le cas avec des structures `case of endof endcase` ;

```
: socketError ( -- )
    errno dup
    case
        2 of      ." No such file "                  endof
        5 of      ." I/O error "                     endof
        9 of      ." Bad file number "               endof
       22 of      ." Invalid argument "              endof
    endcase
    . quit
  ;
```

Les commentaires

Comme tout langage de programmation, le langage FORTH permet le rajout de commentaires dans le code source. Le rajout de commentaires n'a aucune conséquence sur les performances de l'application après compilation du code source.

En langage FORTH, nous disposons de deux mots pour délimiter des commentaires :

- le mot `(` suivi impérativement d'au moins un caractère espace. Ce commentaire est achevé par le caractère `)` ;
- le mot `\` suivi impérativement d'au moins un caractère espace. Ce mot est suivi d'un commentaire de taille quelconque entre ce mot et la fin de la ligne.

Le mot `(` est largement utilisé pour les commentaires de pile. Exemples :

```
dup   ( n – n n )
swap  ( n1 n2 – n2 n1 )
drop  ( n -- )
emit  ( c -- )
```

Les commentaires de pile

Comme nous venons de le voir, ils sont marqués par `(` et `)`. Leur contenu n'a aucune action sur le code FORTH en compilation ou en exécution. On peut donc mettre n'importe quoi entre `(` et `)`. Pour ce qui concerne les commentaires de pile, on restera très concis. Le signe `--` symbolise l'action d'un mot FORTH. Les indications figurant avant `--` correspondent aux données déposées sur la pile de données avant l'exécution du mot. Les indications figurant après `--` correspondent aux données laissées sur la pile de données après exécution du mot. Exemples :

- `words ( -- )` signifie que ce mot ne traite aucune donnée sur la pile de données ;

- `emit ( c -- )` signifie que ce mot traite une donnée en entrée et ne laisse rien sur la pile de données ;
- `bl ( -- 32 )` signifie que ce mot ne traite pas de donnée en entrée et laisse la valeur décimale 32 sur la pile de données ;

Il n'y a aucune limitation sur le nombre de données traitées avant ou après exécution du mot. Pour rappel, les indications entre `(` et `)` sont seulement là pour information.

Signification des paramètres de pile en commentaires

Pour commencer, une petite mise au point très importante s'impose. Il s'agit de la taille des données en pile. Avec eForth Windows, les données de pile occupent 8 octets. Ce sont donc des entiers au format 64 bits. Alors on met quoi sur la pile de données ? Avec eForth Windows, ce seront **TOUJOURS DES DONNEES 64 BITS** ! Un exemple avec le mot `c!` :

```
create myDelemiter
    0 c,
64 myDelimiter c!   ( c addr -- )
```

Ici, le paramètre `c` indique qu'on empile une valeur entière au format 64 bits, mais dont la valeur sera toujours comprise dans l'intervale [0..255].

Le paramètre standard est toujours `n`. S'il y a plusieurs entiers, on les numérotera : `n1 n2 n3`, etc.

On aurait donc pu écrire l'exemple précédent comme ceci :

```
create myDelemiter
    0 c,
64 myDelimiter c!   ( n1 n2 -- )
```

Mais c'est nettement moins explicite que la version précédente. Voici quelques symboles que vous serez amené à voir au fil des codes sources :

- **addr** indique une adresse mémoire littérale ou délivrée par une variable ;
- **c** indique une valeur 8 bits dans l'intervalle [0..255]
- **d** indique une valeur double précision.
  Non utilisé avec eForth Windows qui est déjà au format 32 ou 64 bits ;
- **fl** indique une valeur booléenne, 0 ou non zéro ;
- **n** indique un entier. Entier signé 32 ou 64 bits pour eForth Windows;
- **str** indique une chaîne de caractère. Équivaut à `addr len --`
- **u** indique un entier non signé

Rien n'interdit d'être un peu plus explicite :

```
: SQUARE ( n -- n-exp2 )
```

```
    dup *
  ;
```

Commentaires des mots de définition de mots

Les mots de définition utilisent `create` et `does>`. Pour ces mots, il est conseillé d'écrire les commentaires de pile de cette manière :

```
\ define a command or data stream for SSD1306
: streamCreate: ( comp: <name> | exec: -- addr len )
    create
        here    \ leave current dictionnary pointer on stack
        0 c,    \ initial lenght data is 0
    does>
        dup 1+ swap c@
        \ send a data array to SSD1306 connected via I2C bus
        sendDatasToSSD1306
  ;
```

Ici, le commentaire est partagé en deux parties par le caractère `|` :

- à gauche, la partie action quand le mot de définition est exécuté, préfixé par `comp:`
- à droite la partie action du mot qui sera défini, préfixé par `exec:`

Au risque d'insister, ceci n'est pas un standard. Ce sont seulement des recommandations.

Les commentaires textuels

Ils sont inqués par le mot `\` suivi obligatoirement par au moins un caractère espace et du texte explicatif :

```
\ store at <WORD> addr length of datas compiled beetween
\ <WORD> and here
: ;endStream ( addr-var len ---)
    dup 1+ here
    swap -      \ calculate cdata length
    \ store c in first byte of word defined by streamCreate:
    swap c!
  ;
```

Ces commentaires peuvent être écrits dans n'importe quel alphabet supporté par votre éditeur de code source :

```
\ 儲存在 <WORD> addr 之間編譯的資料長度
\ <WORD> 和這裡
: ;endStream ( addr-var len ---)
    dup 1+ here
    swap -      \ 計算 cdata 長度
    \ 將 c 儲存在由 StreamCreate 定義的字的第一個位元組中：
    swap c!
```

```
;
```

Commentaire en début de code source

Avec une pratique de programmation intensive, on se retrouve rapidement avec des centaines, voire des milliers de fichiers source. Pour éviter des erreurs de choix de fichiers, il est fortement conseillé de marquer le début de chaque fichier source avec un commentaire :

```
\ ******************************************************************
\ key word for UT8 characters
\    Filename:       uekey.fs
\    Date:           29 nov 2023
\    Updated:        29 nov 2023
\    File Version:   1.1
\    MCU:            Linux / Web / Windows
\    Forth:          eForth Windows
\    Copyright:      Marc PETREMANN
\    Author:         Marc PETREMANN
\    GNU General Public License
\ ******************************************************************
```

Toutes ces informations sont à votre libre choix. Elles peuvent devenir très utiles quand on revient des mois ou des années plus tard sur le contenu d'un fichier.

Pour conclure, n'hésitez pas à commenter et indenter vos fichiers sources en langage FORTH.

Outils de diagnostic et mise au point

Le premier outil concerne l'alerte de compilation ou d'interprétation :

```
3 5 25 --> : TEST ( ---)
 ok
3 5 25 -->     [ HEX ] ASCII A DDUP    \ DDUP don't exist
```

Ici, le mot **DDUP** n'existe pas. Toute compilation après cette erreur sera vouée à l'échec.

Le décompilateur

Dans un compilateur conventionnel, le code source est transformé en code exécutable contenant les adresses de référence à une bibliothèque équipant le compilateur. Pour disposer d'un code exécutable , il faut linker le code objet. A aucun moment le programmeur ne peut avoir accès au code exécutable contenu dans ses bibliothèque avec les seules ressources du compilateur.

Avec eForth Windows, le développeur peut décompiler ses définitions. Pour décompiler un mot, il suffit de taper **see** suivi du mot à décompiler :

```
: C>F ( øC --- øF) \ Conversion Celsius in Fahrenheit
```

```
    9 5 */ 32 +
  ;
see c>f
\ display:
: C>F
    9 5 */ 32 +
;
```

Beaucoup de mots du dictionnaire FORTH de eForth Windows peuvent être décompilés.

La décompilation de vos mots permet de détecter d'éventuelles erreurs de compilation.

Dump mémoire

Parfois, il est souhaitable de pouvoir voir les valeurs qui sont en mémoire. Le mot **dump** accepte deux paramètres: l'adresse de départ en mémoire et le nombre d'octets à visualiser :

```
create myDATAS 01 c, 02 c, 03 c, 04 c,
hex
myDATAS 4 dump     \ displays :
3FFEE4EC                                              01 02 03 04
```

Moniteur de pile

Le contenu de la pile de données peut être affiché à tout moment grâce au mot **.s**. Voici la définition du mot **.DEBUG** qui exploite **.s** :

```
variable debugStack

: debugOn ( -- )
    -1 debugStack !
  ;

: debugOff ( -- )
    0 debugStack !
  ;

: .DEBUG
    debugStack @
    if
        cr ." STACK: " .s
        key drop
    then
  ;
```

Pour exploiter **.DEBUG**, il suffit de l'insérer dans un endroit stratégique du mot à mettre au point :

```
\ example of use:
: myTEST
```

```
    128 32 do
        i  .DEBUG
        emit
    loop
;
```

Ici, on va afficher le contenu de la pile de données après exécution du mot i dans notre boucle **do loop**. On active la mise au point et on exécute **myTEST** :

```
debugOn
myTest
\ displays:
\ STACK: <1> 32
\ 2
\ STACK: <1> 33
\ 3
\ STACK: <1> 34
\ 4
\ STACK: <1> 35
\ 5
\ STACK: <1> 36
\ 6
\ STACK: <1> 37
\ 7
\ STACK: <1> 38
```

Quand la mise au point est activée par **debugOn**, chaque affichage du contenu de la pile de données met en pause notre boucle **do loop**. Exécuter **debugOff** pour que le mot **myTEST** s'exécute normalement.

Dictionnaire / Pile / Variables / Constantes

Étendre le dictionnaire

Forth appartient à la classe des langages d'interprétation tissés. Cela signifie qu'il peut interpréter les commandes tapées sur la console, ainsi que compiler de nouveaux sous-programmes et programmes.

Le compilateur Forth fait partie du langage et des mots spéciaux sont utilisés pour créer de nouvelles entrées de dictionnaire (c'est-à-dire des mots). Les plus importants sont `:` (commencer une nouvelle définition) et `;` (termine la définition). Essayons ceci en tapant:

```
: *+ * + ;
```

Ce qui s'est passé? L'action de : est de créer une nouvelle entrée de dictionnaire nommée `*+` et passer du mode interprétation au mode compilation. En mode compilation, l'interpréteur recherche les mots et, plutôt que de les exécuter, installe des pointeurs vers leur code. Si le texte est un nombre, au lieu de le pousser sur la pile, eForth Windows construit le nombre dans le dictionnaire l'espace alloué pour le nouveau mot, suivant le code spécial qui met le numéro stocké sur la pile chaque fois que le mot est exécuté. L'action d'exécution de `*+` est donc d'exécuter séquentiellement les mots définis précédemment `*` et `+`.

Le mot `;` est spécial. C'est un mot immédiat et il est toujours exécuté, même si le le système est en mode compilation. Ce que fait `;` est double. Tout d'abord, il installe le code qui renvoie le contrôle au niveau externe suivant de l'interpréteur et, deuxièmement, il revient du mode compilation au mode interprétation.

Maintenant, essayez votre nouveau mot :

```
decimal 5 6 7 *+ . \ affiche 47 ok<#,ram>
```

Cet exemple illustre deux activités principales de travail dans Forth: ajouter un nouveau mot au dictionnaire, et l'essayer dès qu'il a été défini.

Gestion du dictionnaire

Le mot `forget` suivi du mot à supprimer enlèvera toutes les entrées de dictionnaire que vous avez faites depuis ce mot:

```
: test1 ;
: test2 ;
: test3 ;
forget test2  \ efface test2 et test3 du dictionnaire
```

Piles et notation polonaise inversée

Forth a une pile explicitement visible qui est utilisée pour passer des nombres entre les mots (commandes). Utiliser Forth efficacement vous oblige à penser en termes de pile. Cela peut être difficile au début, mais comme pour tout, cela devient beaucoup plus facile avec la pratique.

En FORTH, La pile est analogue à une pile de cartes avec des nombres écrits dessus. Les nombres sont toujours ajoutés au sommet de la pile et retirés du sommet de la pile. Eforth Windows intègre deux piles: la pile de paramètres et la pile de retour, chacune composée d'un certain nombre de cellules pouvant contenir des nombres de 16 bits.

La ligne d'entrée FORTH:

```
decimal 2 5 73 -16
```

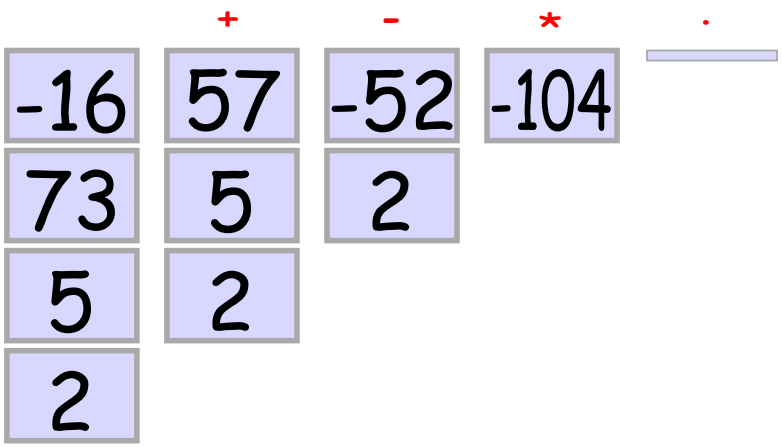
laisse la pile de paramètres dans l'état

Cellule	contenu	commentaire
0	-16	(TOS) Sommet pile
1	73	(NOS) Suivant dans la pile
2	5	
3	2	

Nous utiliserons généralement une numérotation relative à base zéro dans les structures de données Forth telles que piles, tableaux et tables. Notez que, lorsqu'une séquence de nombres est saisie comme celle-ci, le nombre le plus à droite devient *TOS* et le nombre le plus à gauche se trouve au bas de la pile.

Supposons que nous suivions la ligne d'entrée d'origine avec la ligne

```
+ - * .
```



Les opérations produiraient les opérations de pile successives:

Après les deux lignes, la console affiche :

```
decimal 2 5 73 -16  \ affiche: 2 5 73 -16 ok
+ - * .             \ affiche: -104 ok
```

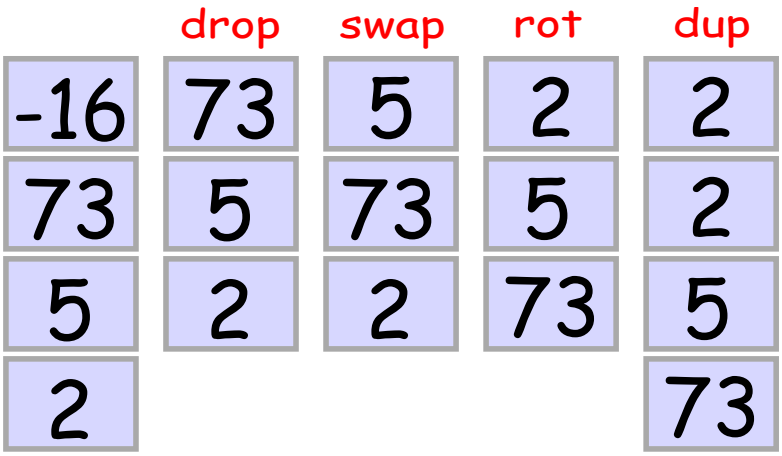
Notez que eForth Windows affiche commodément les éléments de la pile lors de l'interprétation de chaque ligne et que la valeur de -16 est affichée sous la forme d'entier non signé 32 ou 64 bits. En outre, le mot `.` consomme la valeur de données -104, laissant la pile vide. Si nous exécutons . sur la pile maintenant vide, l'interpréteur externe abandonne avec une erreur de pointeur de pile STACK UNDERFLOW ERROR.

La notation de programmation où les opérandes apparaissent en premier, suivis du ou des opérateurs est appelée Notation polonaise inverse (RPN).

Manipulation de la pile de paramètres

Étant un système basé sur la pile, eForth Windows doit fournir des moyens de mettre des nombres sur la pile, pour les supprimer et réorganiser leur ordre. On a déjà vu qu'on peut mettre des nombres sur la pile simplement en les tapant. Nous pouvons également intégrer les nombres dans la définition d'un mot FORTH.

Le mot `drop` supprime un numéro du sommet de la pile mettant ainsi le suivant au sommet. Le mot `swap` échange les 2 premiers numéros. `dup` copie le nombre au sommet, poussant tout les autres numéros vers le bas. `rot` fait pivoter les 3 premiers nombres. Ces



actions sont présentées ci-dessous.

La pile de retour et ses utilisations

Lors de la compilation d'un nouveau mot, eForth Windows établit des liens entre le mot appelant et les mots définis précédemment qui doivent être invoqués par l'exécution du nouveau mot. Ce mécanisme de liaison, lors de l'exécution, utilise la pile de retour (rstack). L'adresse du mot suivant à invoquer est placé sur la pile de retour de sorte que, lorsque le mot courant est terminé en cours d'exécution, le système sait où passer au mot suivant. Comme les mots peuvent être imbriqués, il doit y avoir une pile de ces adresses de retour.

En plus de servir de réservoir d'adresses de retour, l'utilisateur peut également stocker et récupérer à partir de la pile de retour, mais cela doit être fait avec soin car la pile de retour est essentielle à l'exécution du programme. Si vous utilisez la pile de retour pour le

stockage temporaire, vous devez la remettre dans son état d'origine, sinon vous ferez probablement planter le système eForth Windows. Malgré le danger, il y a des moments où l'utilisation de pile de retour comme stockage temporaire peut rendre votre code moins complexe.

Pour stocker dans la pile, utilisez **>r** pour déplacer le sommet de la pile de paramètres vers le haut de la pile de retour. Pour récupérer une valeur, **r>** déplace la valeur supérieure de la pile de retour vers le sommet de la pile de paramètres. Pour supprimer simplement une valeur du haut de la pile, il y a le mot **rdrop**. Le mot **r@** copie le haut de la pile de retour dans la pile de paramètres.

Utilisation de la mémoire

Dans eForth Windows, les nombres 32 ou 64 bits sont extraits de la mémoire vers la pile par le mot **@** (fetch) et stocké du sommet à la mémoire par le mot **!** (store). **@** attend une adresse sur la pile et remplace l'adresse par son contenu. **!** attend un nombre et une adresse pour le stocker. Il place le numéro dans l'emplacement de mémoire référencé par l'adresse, consommant les deux paramètres dans le processus.

Les nombres non signés qui représentent des valeurs de 8 bits (octets) peuvent être placés dans des caractères de la taille d'un caractère. cellules de mémoire en utilisant **c@** et **c!**.

```
create testVar
    cell allot
$f7 testVar c!
testVar c@ .    \ affiche 247
```

Variables

Une variable est un emplacement nommé en mémoire qui peut stocker un nombre, tel que le résultat intermédiaire d'un calcul, hors de la pile. Par exemple:

```
variable x
```

crée un emplacement de stockage nommé, **x**, qui s'exécute en laissant l'adresse de son emplacement de stockage au sommet de la pile:

```
x .    \ affiche l'adresse
```

Nous pouvons alors aller chercher ou stocker à cette adresse :

```
variable x
3 x !
x @ .   \ affiche: 3
```

Constantes

Une constante est un nombre que vous ne voudriez pas changer pendant l'exécution d'un programme. Le résultat de l'exécution du mot associé à un constante est la valeur des données restant sur la pile.

```
\ définit les pins VSPI
19 constant VSPI_MISO
23 constant VSPI_MOSI
18 constant VSPI_SCLK
05 constant VSPI_CS

\ définit la fréquence du port SPI
4000000 constant SPI_FREQ

\ sélectionne le vocabulaire SPI
only FORTH  SPI also

\ initialise le port SPI
: init.VSPI ( -- )
    VSPI_CS OUTPUT pinMode
    VSPI_SCLK VSPI_MISO VSPI_MOSI VSPI_CS SPI.begin
    SPI_FREQ SPI.setFrequency
  ;
```

Valeurs pseudo-constantes

Une valeur définie avec value est un type hybride de variable et constante. Nous définissons et initialisons une valeur et est invoquée comme nous le ferions pour une constante. On peut aussi changer une valeur comme on peut changer une variable.

```
decimal
13 value thirteen
thirteen .      \ display: 13
47 to thirteen
thirteen .      \ display: 47
```

Le mot **to** fonctionne également dans les définitions de mots, en remplaçant la valeur qui le suit par tout ce qui est actuellement au sommet de la pile. Vous devez faire attention à ce que **to** soit suivi d'une valeur définie par **value** et non d'autre chose.

Outils de base pour l'allocation de mémoire

Les mots **create** et **allot** sont les outils de base pour réserver un espace mémoire et y attacher une étiquette. Par exemple, la transcription suivante montre une nouvelle entrée de dictionnaire **graphic-array** :

```
create graphic-array ( --- addr )
    %00000000 c,
    %00000010 c,
```

```
    %00000100 c,
    %00001000 c,
    %00010000 c,
    %00100000 c,
    %01000000 c,
    %10000000 c,
```

Lorsqu'il est exécuté, le mot `graphic-array` poussera l'adresse de la première entrée.

Nous pouvons maintenant accéder à la mémoire allouée à `graphic-array` en utilisant les mots de récupération et de stockage expliqués plus tôt. Pour calculer l'adresse du troisième octet attribué à `graphic-array` on peut écrire `graphic-array 2 +`, en se rappelant que les indices commencent à 0.

```
30 graphic-array  2 + c!
graphic-array  2 + c@ .     \ affiche 30
```

Les nombres réels avec eForth Windows

Si on teste l'opération `1 3 /` en langage FORTH, le résultat sera 0.

Ce n'est pas surprenant. De base, eForth Windows n'utilise que des nombres entiers 32 ou 64 bits via la pile de données. Les nombres entiers offrent certains avantages :

- rapidité de traitement ;
- résultat de calculs sans risque de dérive en cas d'itérations ;
- conviennent à quasiment toutes les situations.

Même en calculs trigonométriques, on peut utiliser une table d'entiers. Il suffit de créer un tableau avec 90 valeurs, où chaque valeur correspond au sinus d'un angle, multiplié par 1000.

Mais les nombres entiers ont aussi des limites :

- résultats impossibles pour des calculs de division simple, comme notre exemple 1/3;
- nécessite des manipulations complexes pour appliquer des formules de physique.

Depuis la version 7.0.6.5, eForth Windows intègre des opérateurs traitant des nombres réels.

Les nombres réels sont aussi dénommés nombres à virgule flottante.

Les réels avec eForth Windows

Afin de distinguer les nombres réels, il faut les terminer avec la lettre "e":

```
3           \ empile 3 sur la pile de données
3e          \ empile 3 sur la pile des réels
5.21e f.    \ affiche 5.210000
```

C'est le mot `f.` qui permet d'afficher un nombre réel situé au sommet de la pile des réels.

Precision des nombres réels avec eForth Windows

Le mot `set-precision` permet d'indiquer le nombre de décimales à afficher après le point décimal. Voyons ceci avec la constante `pi`:

```
pi f.       \ affiche 3.141592
4 set-precision
pi f.       \ affiche 3.1415
```

La précision limite de traitement des nombres réels avec eForth Windows est de six décimales :

```
12 set-precision
1.987654321e f.       \ affiche 1.987654668777
```

Si on réduit la précision d'affichage des nombres réels en dessous de 6, les calculs seront quand même réalisés avec une précision à 6 décimales.

Constantes et variables réelles

Une constante réelle est définie avec le mot **fconstant**:

```
0.693147e fconstant ln2   \ logarithme naturel de 2
```

Une variable réelle est définie avec le mot **fvariable**:

```
fvariable intensity
170e 12e F/ intensity SF!   \ I=P/U   ---  P=170w   U=12V
intensity SF@ f.            \ affiche 14.166669
```

ATTENTION: tous les nombres réels transitent par la **pile des nombres réels**. Dans le cas d'une variable réelle, seule l'adresse pointant sur la valeur réelle transite par la pile de données.

Le mot **SF!** enregistre une valeur réelle à l'adresse ou la variable pointée par son adresse mémoire. L'exécution d'une variable réelle dépose l'adresse mémoire sur la pile données classique.

Le mot **SF@** empile la valeur réelle pointée par son adresse mémoire.

Opérateurs arithmétiques sur les réels

eForth Windows dispose de quatre opérateurs arithmétiques **F+ F- F* F/**:

```
1.23e 4.56e F+ f.   \ affiche 5.790000      1.23-4.56
1.23e 4.56e F- f.   \ affiche -3.330000     1.23-4.56
1.23e 4.56e F* f.   \ affiche 5.608800      1.23*4.56
1.23e 4.56e F/ f.   \ affiche 0.269736      1.23/4.56
```

eForth Windows dispose aussi de ces mots :

- **1/F** calcule l'inverse d'un nombre réel;
- **fsqrt** calcule la racine carrée d'un nombre réel.

```
5e 1/F f.       \ affiche 0.200000      1/5
5e fsqrt f.     \ affiche 2.236068      sqrt(5)
```

Opérateurs mathématiques sur les réels

eForth Windows dispose de plusieurs opérateurs mathématiques :

- **F**** élève un réel r_val à la puissance r_exp

- **FATAN2** calcule l'angle en radian à partir de la tangente.
- **FCOS** (r1 -- r2) Calcule le cosinus d'un angle exprimé en radians.
- **FEXP** (ln-r -- r) calcule le réel correspondant à e EXP r
- **FLN** (r -- ln-r) calcule le logarithme naturel d'un nombre réel.
- **FSIN** (r1 -- r2) calcule le sinus d'un angle exprimé en radians.
- **FSINCOS** (r1 -- rcos rsin) calcule le cosinus et le sinus d'un angle exprimé en radians.

Quelques exemples :

```
  2e 3e f** f.     \ affiche 8.000000
  2e 4e f** f.     \ affiche 16.000000
 10e 1.5e f** f.   \ affiche 31.622776

4.605170e FEXP F.     \ affiche 100.000018

pi 4e f/
FSINCOS f. f.    \ affiche 0.707106 0.707106
pi 2e f/
FSINCOS f. f.    \ affiche 0.000000 1.000000
```

Opérateurs logiques sur les réels

eForth Windows permet aussi d'effectuer des tests logiques sur les réels :

- **F0<** (r -- fl) teste si un nombre réel est inférieur à zéro.
- **F0=** (r -- fl) indique vrai si le réel est nul.
- **f<** (r1 r2 -- fl) fl est vrai si r1 < r2.
- **f<=** (r1 r2 -- fl) fl est vrai si r1 <= r2.
- **f<>** (r1 r2 -- fl) fl est vrai si r1 <> r2.
- **f=** (r1 r2 -- fl) fl est vrai si r1 = r2.
- **f>** (r1 r2 -- fl) fl est vrai si r1 > r2.
- **f>=** (r1 r2 -- fl) fl est vrai si r1 >= r2.

Transformations entiers ↔ réels

eForth Windows dispose de deux mots pour transformer des entiers en réels et inversement :

- **F>S** (r -- n) convertit un réel en entier. Laisse sur la pile de données la partie entière si le réel a des parties décimales.

- **S>F** (n -- r: r) convertit un nombre entier en nombre réel et transfère ce réel sur la pile des réels.

Exemple :

```
35 S>F
F.   \ affiche 35.000000

3.5e F>S .   \ affiche 3
```

Affichage des nombres et chaînes de caractères

Changement de base numérique

FORTH ne traite pas n'importe quels nombres. Ceux que vous avez utilisés en essayant les précédents exemples sont des entiers signés simple précision. Ces nombres peuvent être traités dans n'importe quelle base numérique, toutes les bases numériques situées entre 2 et 36 étant valides :

```
255 HEX . DECIMAL    \ affiche    FF
```

On peut choisir une base numérique encore plus grande, mais les symboles disponibles sortiront de l'ensemble alpha-numérique [0..9,A..Z] et risquent de devenir incohérents.

La base numérique courante est contrôlée par une variable nommée **BASE** et dont le contenu peut être modifié. Ainsi, pour passer en binaire, il suffit de stocker la valeur **2** dans **BASE**. Exemple:

```
2 BASE !
```

et de taper **DECIMAL** pour revenir à la base numérique décimale.

eForth Windows dispose de deux mots pré-définis permettant de sélectionner différentes bases numériques :

- **DECIMAL** pour sélectionner la base numérique décimale. C'est la base numérique prise par défaut au démarrage de eForth Windows;
- **HEX** pour sélectionner la base numérique hexadécimale ;
- **BINARY** pour sélectionner la base numérique binaire.

Dès sélection d'une de ces bases numériques, les nombres littéraux seront interprétés, affichés ou traités dans cette base. Tout nombre entré précédemment dans une base numérique différente de la base numérique courante est automatiquement converti dans la base numérique actuelle. Exemple :

```
DECIMAL     \ base en décimal
255         \ empile 255
HEX         \ sélectionne base hexadécimale
1+          \ incrémente 255 devient 256
.           \ affiche     100
```

On peut définir sa propre base numérique en définissant le mot approprié ou en stockant cette base dans **BASE**. Exemple :

```
: SEXTAL ( ---)         \ sélectionne la base numérique binaire
    6 BASE ! ;
DECIMAL 255 SEXTAL .    \ affiche     1103
```

Le contenu de **BASE** peut être empilé comme le contenu de n'importe quelle autre variable :

```
VARIABLE RANGE_BASE     \ définition de variable RANGE-BASE
BASE @ RANGE_BASE !     \ stockage contenu BASE dans RANGE-BASE
HEX FF 10 + .           \ affiche 10F
RANGE_BASE @ BASE !     \ restaure BASE avec contenu de RANGE-BASE
```

Dans une définition **:** , le contenu de **BASE** peut transiter par la pile de retour:

```
: OPERATION ( ---)
    BASE @ >R       \ stocke BASE sur pile de retour
    HEX FF 10 + .   \ opération du précédent exemple
    R> BASE ! ;     \ restaure valeur initiale de BASE
```

ATTENTION: les mots **>R** et **R>** ne sont pas exploitables en mode interprété. Vous ne pouvez utiliser ces mots que dans une définition qui sera compilée.

Définition de nouveaux formats d'affichage

Forth dispose de primitives permettant d'adapter l'affichage d'un nombre à un format quelconque. Avec eForth Windows, ces primitives traitent les nombres entiers :

- **<#** débute une séquence de définition de format ;
- **#** insère un digit dans une séquence de définition de format ;
- **#S** équivaut à une succession de **#** ;
- **HOLD** insère un caractère dans une définition de format ;
- **#>** achève une définition de format et laisse sur la pile l'adresse et la longueur de la chaîne contenant le nombre à afficher.

Ces mots ne sont utilisables qu'au sein d'une définition. Exemple, soit à afficher un nombre exprimant un montant libellé en euros avec la virgule comme séparateur décimal :

```
: .EUROS ( n ---)
  <# # # [char] , hold #S #>
  type space ." EUR" ;
1245 .euros
```

Exemples d'exécution :

```
35   .EUROS             \ affiche     0,35 EUR
3575 .EUROS             \ affiche     35,75 EUR
1015 3575 + .EUROS      \ affiche     45,90 EUR
```

Dans la définition de **.EUROS**, le mot **<#** débute la séquence de définition de format d'affichage. Les deux mots **#** placent les chiffres des unités et des dizaines dans la chaîne de caractère. Le mot **HOLD** place le caractère **,** (virgule) à la suite des deux chiffres de droite, le mot **#S** complète le format d'affichage avec les chiffres non nuls à la suite de **,** .

Le mot #> ferme la définition de format et dépose sur la pile l'adresse et la longueur de la chaîne contenant les digits du nombre à afficher. Le mot TYPE affiche cette chaîne de caractères.

En exécution, une séquence de format d'affichage traite exclusivement des nombres entiers 32 bits signés ou non signés. La concaténation des différents éléments de la chaîne se fait de droite à gauche, c'est à dire en commençant par les chiffres les moins significatifs.

Le traitement d'un nombre par une séquence de format d'affichage est exécutée en fonction de la base numérique courante. La base numérique peut être modifiée entre deux digits.

Voici un exemple plus complexe démontrant la compacité du FORTH. Il s'agit d'écrire un programme convertissant un nombre quelconque de secondes au format HH:MM:SS:

```
: :00 ( ---)
    DECIMAL #           \ insertion digit unité en décimal
    6 BASE !            \ sélection base 6
    #                   \ insertion digit dizaine
    [char] : HOLD       \ insertion caractère :
    DECIMAL ;           \ retour base décimale
: HMS ( n ---)          \ affiche nombre secondes format HH:MM:SS
    <# :00 :00 #S #> TYPE SPACE ;
```

Exemples d'exécution:

```
59 HMS      \ affiche     0:00:59
60 HMS      \ affiche     0:01:00
4500 HMS    \ affiche     1:15:00
```

Explication : le système d'affichage des secondes et des minutes est appelé système sexagésimal. Les **unités** sont exprimées dans la base numérique décimale, les **dizaines** sont exprimées dans la base six. Le mot :00 gère la conversion des unités et des dizaines dans ces deux bases pour la mise au format des chiffres correspondants aux secondes et aux minutes. Pour les heures, les chiffres sont tous décimaux.

Autre exemple, soit à définir un programme convertissant un nombre entier simple précision décimal en binaire et l'affichant au format bbbb bbbb bbbb bbbb:

```
: FOUR-DIGITS ( ---)
    # # # # 32 HOLD ;
: AFB ( d ---)              \ format 4 digits and a space
    BASE @ >R               \ Current database backup
    2 BASE !                \ Binary digital base selection
    <#
    4 0 DO                  \ Format Loop
        FOUR-DIGITS
    LOOP
    #> TYPE SPACE           \ Binary display
```

```
    R> BASE ! ;             \ Initial digital base restoration
```

Exemple d'exécution :

```
DECIMAL 12 AFB    \ affiche     0000 0000 0000 0110
HEX 3FC5 AFB      \ affiche     0011 1111 1100 0101
```

Encore un exemple, soit à créer un agenda téléphonique où l'on associe à un patronyme un ou plusieurs numéros de téléphone. On définit un mot par patronyme :

```
: .## ( ---)
    # # [char] . HOLD ;
: .TEL ( d ---)
    CR <# .## .## .## .## # # #> TYPE CR ;
: DUGENOU ( ---)
    0618051254 .TEL ;
dugenou   \ display : 06.18.05.12.54
```

Cet agenda, qui peut être compilé depuis un fichier source, est facilement modifiable, et bien que les noms ne soient pas classés, la recherche y est extrêmement rapide.

Affichage des caractères et chaînes de caractères

L'affichage d'un caractère est réalisé par le mot **EMIT**:

```
65 EMIT         \ affiche A
```

Les caractères affichables sont compris dans l'intervalle 32..255. Les codes compris entre 0 et 31 seront également affichés, sous réserve de certains caractères exécutés comme des codes de contrôle. Voici une définition affichant tout le jeu de caractères de la table ASCII :

```
variable #out
: #out+! ( n -- )
    #out +!                 \ incrémente #out
  ;
: (.)  ( n -- a l )
  DUP ABS <# #S ROT SIGN #>
;
: .R  ( n l -- )
  >R (.) R> OVER - SPACES TYPE
;
: JEU-ASCII ( ---)
    cr 0 #out !
    128 32
    DO
        I 3 .R SPACE        \ affiche code du caractère
        4 #out+!
        I EMIT 2 SPACES     \ affiche caractère
        3 #out+!
```

```
        #out @ 77 =
        IF
            CR   0 #out !
        THEN
    LOOP ;
```

L'exécution de **JEU-ASCII** affiche les codes ASCII et les caractères dont le code est compris entre 32 et 127. Pour afficher la table équivalente avec les codes ASCII en hexadécimal, taper **HEX JEU-ASCII** :

```
hex jeu-ascii
 20     21 !   22 "   23 #   24 $   25 %   26 &   27 '   28 (   29 )   2A *
 2B +   2C ,   2D -   2E .   2F /   30 0   31 1   32 2   33 3   34 4   35 5
 36 6   37 7   38 8   39 9   3A :   3B ;   3C <   3D =   3E >   3F ?   40 @
 41 A   42 B   43 C   44 D   45 E   46 F   47 G   48 H   49 I   4A J   4B K
 4C L   4D M   4E N   4F O   50 P   51 Q   52 R   53 S   54 T   55 U   56 V
 57 W   58 X   59 Y   5A Z   5B [   5C \   5D ]   5E ^   5F _   60 `   61 a
 62 b   63 c   64 d   65 e   66 f   67 g   68 h   69 i   6A j   6B k   6C l
 6D m   6E n   6F o   70 p   71 q   72 r   73 s   74 t   75 u   76 v   77 w
 78 x   79 y   7A z   7B {   7C |   7D }   7E ~   7F    ok
```

Les chaînes de caractères sont affichées de diverses manières. La première, utilisable en compilation seulement, affiche une chaîne de caractères délimitée par le caractère " (guillemet) :

```
: TITRE ." MENU GENERAL" ;
    TITRE    \  affiche     MENU GENERAL
```

La chaîne est séparée du mot **."** par au moins un caractère espace.

Une chaîne de caractères peut aussi être compilée par le mot **s"** et délimitée par le caractère **"** (guillemet) :

```
: LIGNE1 ( --- adr len)
    S" E..Enregistrement de données" ;
```

L'exécution de **LIGNE1** dépose sur la pile de données l'adresse et la longueur de la chaîne compilée dans la définition. L'affichage est réalisé par le mot **TYPE :**

```
LIGNE1 TYPE   \ affiche E..Enregistrement de données
```

En fin d'affichage d'une chaîne de caractères, le retour à la ligne doit être provoqué s'il est souhaité :

```
CR TITRE CR CR LIGNE1 TYPE CR
\ affiche
\ MENU GENERAL
\
\ E..Enregistrement de données
```

Un ou plusieurs espaces peuvent être ajoutés en début ou fin d'affichage d'une chaîne alphanumérique :

```
SPACE          \ affiche un caractère espace
10 SPACES      \ affiche 10 caractères espace
```

Variables chaînes de caractères

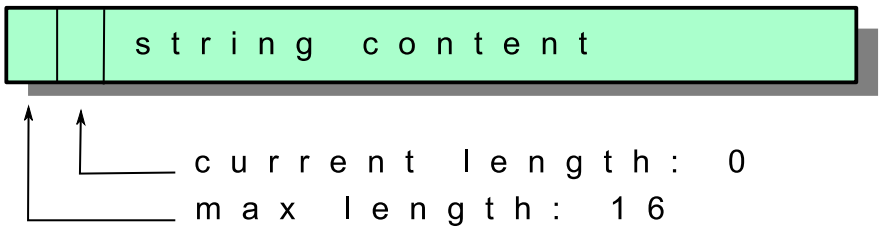
Les variables alpha-numérique texte n'existent pas nativement dans eForth Windows. Voici le premier essai de définition du mot `string` :

```
\ define a strvar
: string  ( comp: n --- names_strvar | exec: --- addr len )
    create
        dup
        c,       \ n is maxlength
        0 c,     \ 0 is real length
        allot
    does>
        2 +
        dup 1 - c@
  ;
```

Une variable chaîne de caractères se définit comme ceci :

```
16 string strState
```

Voici comment est organisé l'espace mémoire réservé pour cette variable texte :



Code des mots de gestion de variables texte

Voici le code source complet permettant la gestion des variables texte :

```
DEFINED? --str [if] forget --str  [then]
create --str


\ compare two strings
: $= ( addr1 len1 addr2 len2 --- fl)
    str=
  ;

\ define a strvar
: string ( n --- names_strvar )
    create
        dup
        ,                   \ n is maxlength
```

```
        0 ,                 \ 0 is real length
        allot
    does>
        cell+ cell+
        dup cell - @
    ;

\ get maxlength of a string
: maxlen$  ( strvar --- strvar maxlen )
    over cell - cell - @
    ;

\ store str into strvar
: $!  ( str strvar --- )
    maxlen$                 \ get maxlength of strvar
    nip rot min             \ keep min length
    2dup swap cell - !      \ store real length
    cmove                   \ copy string
    ;

\ Example:
\ : s1
\     s" this is constant string" ;
\ 200 string test
\ s1 test $!

\ set length of a string to zero
: 0$! ( addr len -- )
    drop 0 swap cell - !
  ;

\ extract n chars right from string
: right$  ( str1 n --- str2 )
    0 max over min >r + r@ - r>
    ;

\ extract n chars left frop string
: left$  ( str1 n --- str2 )
    0 max min
    ;

\ extract n chars from pos in string
: mid$  ( str1 pos len --- str2 )
    >r over swap - right$ r> left$
    ;

\ append char c to string
: c+$!  ( c str1 -- )
```

```
    over >r
    + c!
    r> cell - dup @ 1+ swap !
    ;

\ work only with strings. Don't use with other arrays
: input$ ( addr len -- )
    over swap maxlen$ nip accept
    swap cell - !
  ;
```

La création d'une chaîne de caractères alphanumérique est très simple :

```
64 string myNewString
```

Ici, nous créons une variable alphanumérique `myNewString` pouvant contenir jusqu'à 64 caractères.

Pour afficher le contenu d'une variable alphanumérique, il suffit ensuite d'utiliser `type`. Exemple :

```
s" This is my first example.." myNewString $!
myNewString type   \ display: This is my first example..
```

Si on tente d'enregistrer une chaîne de caractères plus longue que la taille maximale de notre variable alphanumérique, la chaîne sera tronquée :

```
s" This is a very long string, with more than 64 characters. It can't store
complete"
myNewString $!
myNewString type
  \ affiche: This is a very long string, with more than 64 characters. It
can
```

Ajout de caractère à une variable alphanumérique

Certains périphériques, le transmetteur LoRa par exemple, demandent à traiter des lignes de commandes contenant les caractères non alphanumériques Le mot `c+$!` permet cette insertion de code :

```
32 string AT_BAND
s" AT+BAND=868500000" AT_BAND $!  \ set frequency at 865.5 Mhz
$0a AT_BAND c+$!
$0d AT_BAND c+$!    \ add CR LF code at end of command
```

Le dump mémoire du contenu de notre variable alphanumérique `AT_BAND` confirme la présence des deux caractères de contrôle en fin de chaine :

```
--> AT_BAND dump
--addr---  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  ------chars-----
3FFF-8620  8C 84 FF 3F 20 00 00 00 13 00 00 00 41 54 2B 42  ...? .......AT+B
3FFF-8630  41 4E 44 3D 38 36 38 35 30 30 30 30 30 0A 0D BD  AND=868500000...
 ok
```

Voici une manière astucieuse de créer une variable alphanumérique permettant de transmettre un retour chariot, un **CR+LF** compatible avec les fins de commandes pour le transmetteur LoRa:

```
2 string $crlf
$0d $crlf c+$!
$0a $crlf c+$!

: crlf ( -- )       \ same action as cr, but adapted for LoRa
    $crlf type
  ;
```

Les mots de création de mots

FORTH est plus qu'un langage de programmation. C'est un méta-langage. Un méta-langage est un langage utilisé pour décrire, spécifier ou manipuler d'autres langages.

Avec eForth Windows, on peut définir la syntaxe et la sémantique de mots de programmation au-delà du cadre formel des définitions de base.

On a déjà vu les mots définis par **`constant`**, **`variable`**, **`value`**. Ces mots servent à gérer des données numériques.

Dans le chapitre Structures de données pour eForth Windows, on a également utilisé le mot **`create`**. Ce mot crée un en-tête permettant d'accéder à une zone de données mis en mémoire. Exemple :

```
create temperatures
    34 , 37 , 42 , 36 , 25 , 12 ,
```

Ici, chaque valeur est stockée dans la zone des paramètres du mot **`temperatures`** avec le mot **`,`**.

Avec eForth Windows, on va voir comment personnaliser l'exécution des mots définis par **`create`**.

Utilisation de does>

Il y a une combinaison de mots-clés "**`CREATE`**" et "**`DOES>`**", qui est souvent utilisée ensemble pour créer des mots (mots de vocabulaire) personnalisés avec des comportements spécifiques.

Voici comment cela fonctionne en Forth :

- **`CREATE`** **:** ce mot-clé est utilisé pour créer un nouvel espace de données dans le dictionnaire eForth Windows. Il prend en charge un argument, qui est le nom que vous donnez à votre nouveau mot ;
- **`DOES>`** **:** ce mot-clé est utilisé pour définir le comportement du mot que vous venez de créer avec **`CREATE`**. Il est suivi d'un bloc de code qui spécifie ce que le mot devrait faire lorsqu'il est rencontré pendant l'exécution du programme.

Ensemble, cela ressemble à quelque chose comme ceci :

```
forth
CREATE mon-nouveau-mot
    \  code à exécuter lorsqu'on rencontre mon-nouveau-mot
    DOES>
  ;
```

Lorsque le mot **mon-nouveau-mot** est rencontré dans le programme FORTH, le code spécifié dans la partie **does> … ;** sera exécuté.

```
\ define a register, similar as constant
: defREG:
    create ( addr1 -- <name> )
        ,
    does>  ( -- regAddr )
        @
  ;
```

Ici, on définit le mot de définition **defREG:** qui a exactement la même action que **value**. Mais pourquoi créer un mot qui recrée l'action d'un mot qui existe déjà ?

```
$00 value DB2INSTANCE
```

ou

```
$00 defREG: DB2INSTANCE
```

sont semblables. Cependant, en créant nos registres avec **defREG:** on a les avantages suivants :

- un code source eForth Windows plus lisible. On détecte facilement toutes les constantes nommant un registre ;
- on se laisse la possibilité de modifier la partie **does>** de **defREG:** sans avoir ensuite à réécrire les lignes de code qui n'utiliseraient pas **defREG:**

Voici un cas classique, le traitement d'un tableau de données :

```
\ mot de définition pour tableau à une dimension
: array ( comp: -- <name> | exec: index <name> -- addr )
    create
    does>
        swap cell * +
  ;
array temperatures
    21 ,    32 ,    45 ,    44 ,    28 ,    12 ,
0 temperatures @ .    \ display 21
5 temperatures @ .    \ display 12
```

L'exécution de **temperatures** doit être précédé de la position de la valeur à extraire dans ce tableau. Ici nous récupérons seulement l'adresse contenant la valeur à extraire.

Exemple de gestion de couleur

Dans ce premier exemple, on définit le mot **color:** qui va récupérer la couleur à sélectionner et la stocker dans une variable :

```
0 value currentCOLOR
```

```
\ define word as COLOR constant
: color: ( n -- <name> )
    create
        ,
    does>
        @ to currentCOLOR
  ;

$00 color: setBLACK
$ff color: setWHITE
```

L'exécution du mot **setBLACK** ou **setWHITE** simplifie considérablement le code eForth Windows. Sans ce mécanisme, il aurait fallu répéter régulièrement une de ces lignes :

```
$00 currentCOLOR !
```

Ou

```
$00 variable BLACK
BLACK currentCOLOR !
```

Exemple, écrire en pinyin

Le pinyin est couramment utilisé dans le monde entier pour enseigner la prononciation du chinois mandarin, et il est également utilisé dans divers contextes officiels en Chine, comme les panneaux de signalisation, les dictionnaires et les manuels d'apprentissage. Il facilite l'apprentissage du chinois pour les personnes dont la langue maternelle utilise l'alphabet latin.

Pour écrire en chinois sur un clavier QWERTY, les Chinois utilisent généralement un système appelé "pinyin input" ou "saisie pinyin". Pinyin est un système de romanisation du chinois mandarin, qui utilise l'alphabet latin pour représenter les sons du mandarin.

Sur un clavier QWERTY, les utilisateurs tapent les sons du mandarin en utilisant la romanisation pinyin. Par exemple, si quelqu'un veut écrire le caractère "你" ("nǐ" signifiant "tu" ou "toi" en français), il peut taper "ni".

Dans ce code très simplifié, on peut programmer des mots pinyin pour écrire en mandarin. Le code ci-après fonctionne parfaitement dans eForth Windows :

```
\ Work well in eForth Windows
: chinese:
    create ( c1 c2 c3 -- )
        c, c, c,
    does>
        3 type
  ;
```

Pour trouver le code UTF8 d'un caractère chinois, copiez le caractère chinois, depuis Google Translate par exemple. Exemple :

```
Good Morning --> 早安 (Zao an)
```

Copiez 早 et allez dans eForth Windows et tapez :

```
key key key    \ followed by key <enter>
```

collez le caractère 早. Eforth Windows doit afficher les codes suivants :

```
230 151 169
```

Pour chaque caractère chinois, on va exploiter ces trois codes ainsi :

```
169 151 230 chinese: Zao
137 174 229 chinese: An
```

Utilisation :

```
Zao An        \ display 早安
```

Avouez quand même que programmer ainsi c'est autre chose que ce qu'on peut faire en langage C. Non ?

Contenu détaillé des vocabulaires eForth Windows

Eforth Windows met à disposition de nombreux vocabulaires :

- **FORTH** est le principal vocabulaire ;
- certains vocabulaires servent à la mécanique interne pour Eforth Windows, comme **internals**, **asm…**

Vous trouverez ici la liste de tous les mots définis dans ces différents vocabulaires. Certains mots sont présentés avec un lien coloré :

align est un mot FORTH ordinaire ;

CONSTANT est mot de définition ;

begin marque une structure de contrôle ;

key est un mot d'exécution différée ;

LED est un mot défini par **constant**, **variable** ou **value** ;

registers marque un vocabulaire.

Les mots du vocabulaire **FORTH** sont affichés par ordre alphabétique. Pour les autres vocabulaires, les mots sont présentés dans leur ordre d'affichage.

Version v 7.0.7.15

FORTH

-	-rot	,	;	:	:noname	!
?	?do	?dup	.	."	.s	'
(local)	[	[']	[char]	[ELSE]	[IF]	[THEN]
]	{	}transfer	@	*	*/	*/MOD
/	/mod	#	#!	#>	#fs	#s
#tib	+	+!	+loop	+to	+to	<
<#	<=	<>	=	>	>=	>BODY
>flags	>flags&	>in	>link	>link&	>name	>params
>R	>size	0<	0<>	0=	1-	1/F
1+	2!	2@	2*	2/	2drop	2dup
4*	4/	abort	abort"	abs	accept	afliteral
aft	again	ahead	align	aligned	allocate	allot
also	AND	ansi	argc	argv	ARSHIFT	asm
assert	at-xy	base	begin	bg	BIN	binary
bl	blank	block	block-fid	block-id	buffer	bye
c,	C!	C@	CASE	cat	catch	CELL
cell/	cell+	cells	char	CLOSE-FILE	cmove	cmove>
CONSTANT	context	copy	cp	cr	CREATE	CREATE-FILE
current	decimal	default-key	default-key?		default-type	
default-use	defer	DEFINED?	definitions	DELETE-FILE	depth	do

DOES>	DROP	dump	dump-file	DUP	echo	editor
else	emit	empty-buffers		ENDCASE	ENDOF	erase
evaluate	EXECUTE	exit	extract	F-	f.	f.s
F*	F**	F/	F+	F<	F<=	F<>
F=	F>	F>=	F>S	F0<	F0=	FABS
FATAN2	fconstant	FCOS	fdepth	FDROP	FDUP	FEXP
fg	file-exists?		FILE-POSITION		FILE-SIZE	fill
FIND	fliteral	FLN	FLOOR	flush	FLUSH-FILE	FMAX
FMIN	FNEGATE	FNIP	for	forget	FORTH	forth-builtins
FOVER	FP!	FP@	fp0	free	FROT	FSIN
FSINCOS	FSQRT	FSWAP	fvariable	graphics	handler	here
hex	hld	hold	I	if	IMMEDIATE	include
included	included?	internals	invert	is	J	K
key	key?	L!	latestxt	leave	list	literal
load	loop	LSHIFT	max	min	mod	ms
ms-ticks	mv	n.	needs	negate	nest-depth	next
nip	nl	NON-BLOCK	normal	octal	OF	ok
only	open-blocks	OPEN-FILE	OR	order	OVER	pad
page	PARSE	pause	PI	postpone	precision	previous
prompt	quit	r"	R@	R/O	R/W	R>
r|	r~	rdrop	READ-FILE	recurse	refill	remaining
remember	RENAME-FILE	repeat	REPOSITION-FILE		required	reset
resize	RESIZE-FILE	restore	revive	rm	rot	RP!
RP@	rp0	RSHIFT	s"	S>F	s>z	save
save-buffers		scr	sealed	see	set-precision	
set-title	sf,	SF!	SF@	SFLOAT	SFLOAT+	SFLOATS
sign	SL@	SP!	SP@	sp0	space	spaces
start-task	startswith?	startup:	state	str	str=	streams
structures	SW@	SWAP	task	tasks	terminate	then
throw	thru	tib	to	touch	transfer	transfer{
type	u.	U/MOD	UL@	UNLOOP	until	update
use	used	UW@	value	VARIABLE	visual	vlist
vocabulary	W!	W/O	while	windows	words	WRITE-FILE
XOR	z"	z>s				

windows

process-heap HeapReAlloc HeapFree HeapAlloc GetProcessHeap WM_>name WM_PENWINLAST WM_PENEVENT WM_CTLINIT WM_PENMISC WM_PENCTL WM_HEDITCTL WM_SKB WM_PENMISCINFO WM_GLOBALRCCHANGE WM_HOOKRCRESULT WM_RCRESULT WM_PENWINFIRST WM_AFXLAST WM_AFXFIRST WM_HANDHELDLAST WM_HANDHELDFIRST WM_APPCOMMAND WM_PRINTCLIENT WM_PRINT WM_HOTKEY WM_PALETTECHANGED WM_PALETTEISCHANGING WM_QUERYNEWPALETTE WM_HSCROLLCLIPBOARD WM_CHANGECBCHAIN WM_ASKCBFORMATNAME WM_SIZECLIPBOARD WM_VSCROLLCLIPBOARD WM_PAINTCLIPBOARD WM_DRAWCLIPBOARD WM_DESTROYCLIPBOARD WM_RENDERALLFORMATS WM_RENDERFORMAT WM_UNDO WM_CLEAR WM_PASTE WM_COPY WM_CUT WM_MOUSELEAVE WM_NCMOUSELEAVE WM_MOUSEHOVER WM_NCMOUSEHOVER WM_IME_KEYUP WM_IMEKEYUP WM_IME_KEYDOWN WM_IMEKEYDOWN WM_IME_REQUEST WM_IME_CHAR WM_IME_SELECT WM_IME_COMPOSITIONFULL WM_IME_CONTROL WM_IME_NOTIFY WM_IME_SETCONTEXT WM_IME_REPORT WM_MDIREFRESHMENU WM_DROPFILES WM_EXITSIZEMOVE WM_ENTERSIZEMOVE WM_MDISETMENU WM_MDIGETACTIVE WM_MDIICONARRANGE WM_MDICASCADE WM_MDITILE WM_MDIMAXIMIZE WM_MDINEXT WM_MDIRESTORE WM_MDIACTIVATE WM_MDIDESTROY WM_MDICREATE WM_DEVICECHANGE

WM_POWERBROADCAST WM_MOVING WM_CAPTURECHANGED WM_SIZING WM_NEXTMENU WM_EXITMENULOOP
WM_ENTERMENULOOP WM_PARENTNOTIFY WM_MOUSEHWHEEL WM_XBUTTONDBLCLK WM_XBUTTONUP
WM_XBUTTONDOWN WM_MOUSEWHEEL WM_MOUSELAST WM_MBUTTONDBLCLK WM_MBUTTONUP
WM_MBUTTONDOWN WM_RBUTTONDBLCLK WM_RBUTTONUP WM_RBUTTONDOWN WM_LBUTTONDBLCLK
WM_LBUTTONUP WM_LBUTTONDOWN WM_MOUSEMOVE WM_MOUSEFIRST CB_MSGMAX CB_GETCOMBOBOXINFO
CB_MULTIPLEADDSTRING CB_INITSTORAGE CB_SETDROPPEDWIDTH CB_GETDROPPEDWIDTH
CB_SETHORIZONTALEXTENT CB_GETHORIZONTALEXTENT CB_SETTOPINDEX CB_GETTOPINDEX
CB_GETLOCALE CB_SETLOCALE CB_FINDSTRINGEXACT CB_GETDROPPEDSTATE CB_GETEXTENDEDUI
CB_SETEXTENDEDUI CB_GETITEMHEIGHT CB_SETITEMHEIGHT CB_GETDROPPEDCONTROLRECT
CB_SETITEMDATA CB_GETITEMDATA CB_SHOWDROPDOWN CB_SETCURSEL CB_SELECTSTRING
CB_FINDSTRING CB_RESETCONTENT CB_INSERTSTRING CB_GETLBTEXTLEN CB_GETLBTEXT
CB_GETCURSEL CB_GETCOUNT CB_DIR CB_DELETESTRING CB_ADDSTRING CB_SETEDITSEL
CB_LIMITTEXT CB_GETEDITSEL WM_CTLCOLORSTATIC WM_CTLCOLORSCROLLBAR WM_CTLCOLORDLG
WM_CTLCOLORBTN WM_CTLCOLORLISTBOX WM_CTLCOLOREDIT WM_CTLCOLORMSGBOX WM_LBTRACKPOINT
WM_QUERYUISTATE WM_UPDATEUISTATE WM_CHANGEUISTATE WM_MENUCOMMAND WM_UNINITMENUPOPUP
WM_MENUGETOBJECT WM_MENUDRAG WM_MENURBUTTONUP WM_ENTERIDLE WM_MENUCHAR
WM_MENUSELECT WM_SYSTIMER WM_INITMENUPOPUP WM_INITMENU WM_VSCROLL WM_HSCROLL
WM_TIMER WM_SYSCOMMAND WM_COMMAND WM_INITDIALOG WM_IME_KEYLAST WM_IME_COMPOSITION
WM_IME_ENDCOMPOSITION WM_IME_STARTCOMPOSITION WM_INTERIM WM_CONVERTRESULT
WM_CONVERTREQUEST WM_WNT_CONVERTREQUESTEX WM_UNICHAR WM_SYSDEADCHAR WM_SYSCHAR
WM_SYSKEYUP WM_SYSKEYDOWN WM_DEADCHAR WM_CHAR WM_KEYUP WM_KEYDOWN WM_INPUT
BM_SETDONTCLICK BM_SETIMAGE BM_GETIMAGE BM_CLICK BM_SETSTYLE BM_SETSTATE
BM_GETSTATE BM_SETCHECK BM_GETCHECK SBM_GETSCROLLBARINFO SBM_GETSCROLLINFO
SBM_SETSCROLLINFO SBM_SETRANGEREDRAW SBM_ENABLE_ARROWS SBM_GETRANGE SBM_SETRANGE
SBM_GETPOS SBM_SETPOS EM_GETIMESTATUS EM_SETIMESTATUS EM_CHARFROMPOS EM_POSFROMCHAR
EM_GETLIMITTEXT EM_GETMARGINS EM_SETMARGINS EM_GETPASSWORDCHAR EM_GETWORDBREAKPROC
EM_SETWORDBREAKPROC EM_SETREADONLY EM_GETFIRSTVISIBLELINE EM_EMPTYUNDOBUFFER
EM_SETPASSWORDCHAR EM_SETTABSTOPS EM_SETWORDBREAK EM_LINEFROMCHAR EM_FMTLINES
EM_UNDO EM_CANUNDO EM_SETLIMITTEXT EM_LIMITTEXT EM_GETLINE EM_SETFONT EM_REPLACESEL
EM_LINELENGTH EM_GETTHUMB EM_GETHANDLE EM_SETHANDLE EM_LINEINDEX EM_GETLINECOUNT
EM_SETMODIFY EM_GETMODIFY EM_SCROLLCARET EM_LINESCROLL EM_SCROLL EM_SETRECTNP
EM_SETRECT EM_GETRECT EM_SETSEL EM_GETSEL WM_NCXBUTTONDBLCLK WM_NCXBUTTONUP
WM_NCXBUTTONDOWN WM_NCMBUTTONDBLCLK WM_NCMBUTTONUP WM_NCMBUTTONDOWN
WM_NCRBUTTONDBLCLK
WM_NCRBUTTONUP WM_NCRBUTTONDOWN WM_NCLBUTTONDBLCLK WM_NCLBUTTONUP WM_NCLBUTTONDOWN
WM_NCMOUSEMOVE WM_SYNCPAINT WM_GETDLGCODE WM_NCACTIVATE WM_NCPAINT WM_NCHITTEST
WM_NCCALCSIZE WM_NCDESTROY WM_NCCREATE WM_SETICON WM_GETICON WM_DISPLAYCHANGE
WM_STYLECHANGED WM_STYLECHANGING WM_CONTEXTMENU WM_NOTIFYFORMAT WM_USERCHANGED
WM_HELP WM_TCARD WM_INPUTLANGCHANGE WM_INPUTLANGCHANGEREQUEST WM_NOTIFY
WM_CANCELJOURNAL WM_COPYDATA WM_COPYGLOBALDATA WM_POWER WM_WINDOWPOSCHANGED
WM_WINDOWPOSCHANGING WM_COMMNOTIFY WM_COMPACTING WM_GETOBJECT WM_COMPAREITEM
WM_QUERYDRAGICON WM_GETHOTKEY WM_SETHOTKEY WM_GETFONT WM_SETFONT WM_CHARTOITEM
WM_VKEYTOITEM WM_DELETEITEM WM_MEASUREITEM WM_DRAWITEM WM_SPOOLERSTATUS
WM_NEXTDLGCTL WM_ICONERASEBKGND WM_PAINTICON WM_GETMINMAXINFO WM_QUEUESYNC
WM_CHILDACTIVATE WM_MOUSEACTIVATE WM_SETCURSOR WM_CANCELMODE WM_TIMECHANGE
WM_FONTCHANGE WM_ACTIVATEAPP WM_DEVMODECHANGE WM_WININICHANGE WM_CTLCOLOR
WM_SHOWWINDOW WM_ENDSESSION WM_SYSCOLORCHANGE WM_ERASEBKGND WM_QUERYOPEN
WM_QUIT WM_QUERYENDSESSION WM_CLOSE WM_PAINT WM_GETTEXTLENGTH WM_GETTEXT
WM_SETTEXT WM_SETREDRAW WM_ENABLE WM_KILLFOCUS WM_SETFOCUS WM_ACTIVATE
WM_SIZE WM_MOVE WM_DESTROY WM_CREATE WM_NULL SRCCOPY DIB_RGB_COLORS BI_RGB
->bmiColors ->bmiHeader BITMAPINFO ->biClrImportant ->biClrUsed ->biYPelsPerMeter
->biXPelsPerMeter ->biSizeImage ->biCompression ->biBitCount ->biPlanes
->biHeight ->biWidth ->biSize BITMAPINFOHEADER ->rgbReserved ->rgbRed ->rgbGreen

->rgbBlue RGBQUAD StretchDIBits DC_PEN DC_BRUSH DEFAULT_GUI_FONT SYSTEM_FIXED_FONT
DEFAULT_PALETTE DEVICE_DEFAULT_PALETTE SYSTEM_FONT ANSI_VAR_FONT ANSI_FIXED_FONT
OEM_FIXED_FONT BLACK_PEN WHITE_PEN NULL_BRUSH BLACK_BRUSH DKGRAY_BRUSH
GRAY_BRUSH LTGRAY_BRUSH WHITE_BRUSH GetStockObject COLOR_WINDOW RGB
CreateSolidBrush
DeleteObject Gdi32 dpi-aware SetThreadDpiAwarenessContext VK_ALT GET_X_LPARAM
GET_Y_LPARAM IDI_INFORMATION IDI_ERROR IDI_WARNING IDI_SHIELD IDI_WINLOGO
IDI_ASTERISK IDI_EXCLAMATION IDI_QUESTION IDI_HAND IDI_APPLICATION LoadIconA
IDC_HELP IDC_APPSTARTING IDC_HAND IDC_NO IDC_SIZEALL IDC_SIZENS IDC_SIZEWE
IDC_SIZENESW IDC_SIZENWSE IDC_ICON IDC_SIZE IDC_UPARROW IDC_CROSS IDC_WAIT
IDC_IBEAM IDC_ARROW LoadCursorA PostQuitMessage FillRect ->rgbReserved
->fIncUpdate ->fRestore ->rcPaint ->fErase ->hdc PAINTSTRUCT EndPaint BeginPaint
GetDC PM_NOYIELD PM_REMOVE PM_NOREMOVE ->lPrivate ->pt ->time ->lParam
->wParam ->message ->hwnd MSG DispatchMessageA TranslateMessage PeekMessageA
GetMessageA ->bottom ->right ->top ->left RECT ->y ->x POINT CW_USEDEFAULT
IDI_MAIN_ICON DefaultInstance WS_TILEDWINDOW WS_POPUPWINDOW WS_OVERLAPPEDWINDOW
WS_CAPTION WS_TILED WS_ICONIC WS_CHILDWINDOW WS_GROUP WS_TABSTOP WS_POPUP
WS_CHILD WS_MINIMIZE WS_VISIBLE WS_DISABLED WS_CLIPSIBLINGS WS_CLIPCHILDREN
WS_MAXIMIZE WS_BORDER WS_DLGFRAME WS_VSCROLL WS_HSCROLL WS_SYSMENU WS_THICKFRAME
WS_MINIMIZEBOX WS_MAXIMIZEBOX WS_OVERLAPPED CreateWindowExA callback DefWindowProcA
SetForegroundWindow SW_SHOWMAXIMIZED SW_SHOWNORMAL SW_FORCEMINIMIZE SW_SHOWDEFAULT
SW_RESTORE SW_SHOWNA SW_SHWOMINNOACTIVE SW_MINIMIZE SW_SHOW SW_SHOWNOACTIVATE
SW_MAXIMIZED SW_SHOWMINIMIZED SW_NORMAL SW_HIDE ShowWindow ->lpszClassName
->lpszMenuName ->hbrBackground ->hCursor ->hIcon ->hInstance ->cbWndExtra
->cbClsExtra ->lpfnWndProc ->style WINDCLASSA RegisterClassA MB_CANCELTRYCONTINUE
MB_RETRYCANCEL MB_YESNO MB_YESNOCANCEL MB_ABORTRETRYIGNORE MB_OKCANCEL
MB_OK MessageBoxA User32 win-key win-key? raw-key win-type init-console
console-mode stderr stdout stdin console-started FlushConsoleInputBuffer
SetConsoleMode GetConsoleMode GetStdHandle ExitProcess AllocConsole
ENABLE_LVB_GRID_WORLDWIDE
DISABLE_NEWLINE_AUTO_RETURN ENABLE_VIRTUAL_TERMINAL_PROCESSING
ENABLE_WRAP_AT_EOL_OUTPUT
ENABLE_PROCESSED_OUTPUT ENABLE_VIRTUAL_TERMINAL_INPUT ENABLE_QUICK_EDIT_MODE
ENABLE_INSERT_MODE ENABLE_MOUSE_INPUT ENABLE_WINDOW_INPUT ENABLE_ECHO_INPUT
ENABLE_LINE_INPUT ENABLE_PROCESSED_INPUT STD_ERROR_HANDLE STD_OUTPUT_HANDLE
STD_INPUT_HANDLE invalid?ior d0sz wargv
wargc CommandLineToArgvW Shell32 GetModuleHandleA GetCommandLineW GetLastError
WaitForSingleObject GetTickCount Sleep ExitProcess Kernel32 contains? dll
sofunc GetProcAddress LoadLibraryA WindowProcShim SetupCtrlBreakHandler
windows-builtins calls

Ressources

En anglais

- **ESP32forth** page maintenue par Brad NELSON, le créateur de ESP32forth. Vous y trouverez toutes les versions (ESP32, Windows, Web, Linux...)
  https://esp32forth.appspot.com/ESP32forth.html

En chinois

- **Eforth and zen** page maintenue par un groupe de passionnés à Taïwan
  https://eforth.com.tw/academy/

En français

- **eForth** site en deux langues (français, anglais) avec plein d'exemples
  https://eforth.com.tw/academy/library.htm

GitHub

- **Ueforth** ressources maintenues par Brad NELSON. Contient tous les fichiers sources en Forth et en langage C de ESP32forth
  https://github.com/flagxor/ueforth
- **eForth Windows** codes sources et documentation pour eForth Windows. Ressources maintenues par Marc PETREMANN
  https://github.com/MPETREMANN11/eForth-Windows

Index lexical