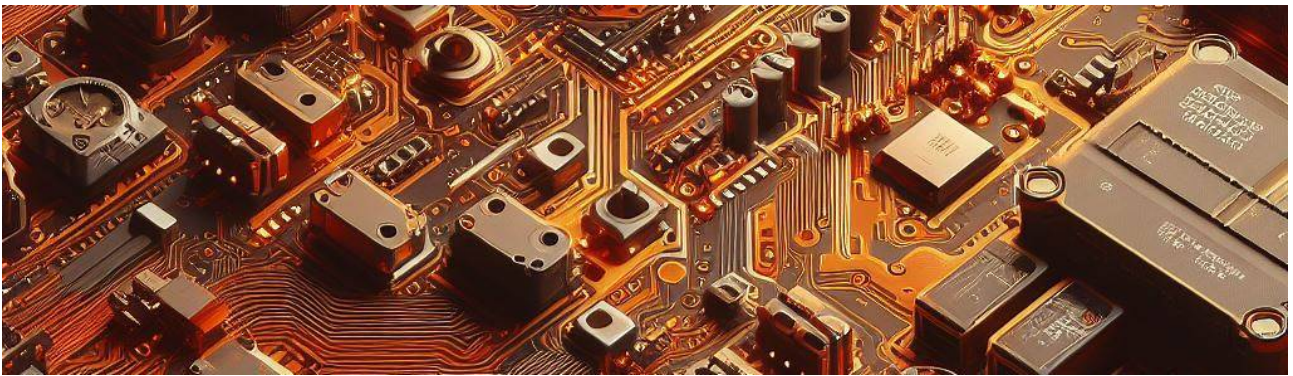


Le grand livre de eFORTH Linux

version 1.0 - 21 novembre 2023



Auteur

- Marc PETREMANN

Collaborateur(s)

- XXX

Table des matières

Installer eForth sous Linux.....	3
Prérequis.....	3
Installer eForth Linux sous Linux.....	4
Lancer eForth Linux.....	6
Un vrai FORTH 64 bits avec eForth Linux.....	7
Les valeurs sur la pile de données.....	7
Les valeurs en mémoire.....	7
Traitement par mots selon taille ou type des données.....	8
Conclusion.....	9
Dictionnaire / Pile / Variables / Constantes.....	11
Étendre le dictionnaire.....	11
Gestion du dictionnaire.....	11
Piles et notation polonaise inversée.....	12
Manipulation de la pile de paramètres.....	13
La pile de retour et ses utilisations.....	13
Utilisation de la mémoire.....	14
Variables.....	14
Constantes.....	15
Valeurs pseudo-constantes.....	15
Outils de base pour l'allocation de mémoire.....	15

Installer eForth sous Linux

eForth Linux est une version très puissante destinée au système Linux. eForth Linux fonctionne sur toutes les versions récentes de Linux, y compris dans un environnement virtuel Linux.

Prérequis

Vous devez disposer d'un système Linux opérationnel :

- installé sur un ordinateur utilisant Linux comme seul système d'exploitation ;
- installé dans un environnement virtuel.

Si vous disposez seulement d'un ordinateur sous Windows 10 ou 11, vous pouvez installer Linux dans le sous-système **WSL**¹.

Le Sous-système Windows pour Linux permet aux développeurs d'exécuter un environnement GNU/Linux (et notamment la plupart des utilitaires, applications et outils en ligne de commande) directement sur Windows, sans modification et tout en évitant la surcharge d'une machine virtuelle traditionnelle ou d'une configuration à double démarrage.

L'intérêt d'une installation d'une distribution Linux dans **WSL** permet d'avoir à disposition une version Linux en mode commande en quelques secondes. Ici, **Ubuntu** est accessible depuis le système de fichiers Windows et se lance en un seul clic :

1 WSL = Windows Subsystem Linux

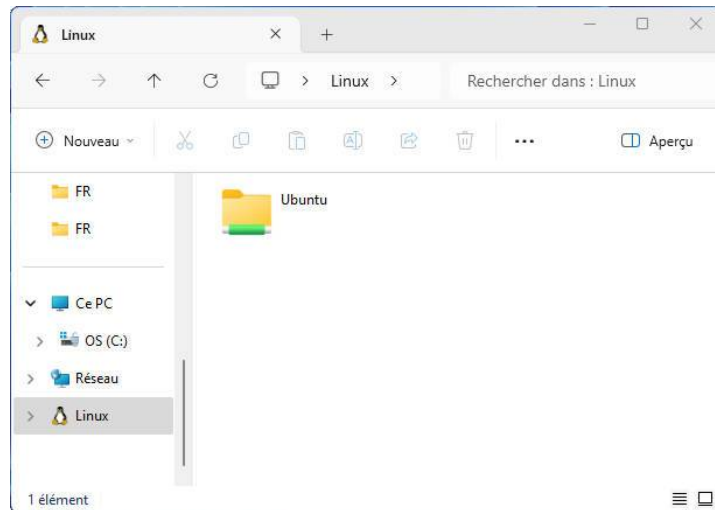


Figure 1: Ubuntu accessible en un clic depuis WSL sous Windows

Toutes les instructions pour installer **WSL2** puis la distribution Linux de votre choix sont disponibles ici :

<https://learn.microsoft.com/fr-fr/windows/wsl/install>

Par défaut, WSL2 propose d'installer la distribution **Linux Ubuntu**.

Installer eForth Linux sous Linux

Si vous lancez Ubuntu (ou toute autre version de Linux), vous vous retrouverez par défaut dans votre répertoire utilisateur. On commence par créer un dossier **ueforth** :

```
mkdir ueforth
```

Puis sélectionnez ce dossier :

```
cd ueforth
```

On va maintenant télécharger la version du fichier binaire de ueForth Linux :

- soit depuis la page d'accueil du site ESP32forth de Brad NELSON :
<https://esp32forth.appspot.com/ESP32forth.html>
- soit depuis le dépôt de stockage eforth Google :
<https://eforth.storage.googleapis.com/releases/archive.html>

Dans la liste des fichiers proposés, copiez le lien web mentionnant linux :

```
https://eforth.storage.googleapis.com/releases/ueforth-7.0.7.15.linux
```

Sous Linux, tapez la commande **wget** :

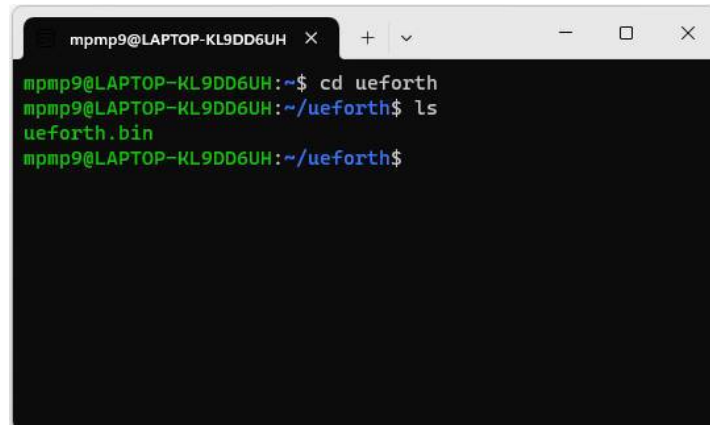
```
wget https://eforth.storage.googleapis.com/releases/ueforth-7.0.7.15.linux
```

Le téléchargement va déposer automatiquement le fichier dans le dossier ueforth précédemment sélectionné. Si vous avez repris le lien ci-avant, vous vous retrouvez avec un fichier nommé **ueforth-7.0.7.15.linux** dans ce dossier.

On renomme ce fichier avec al commande **mv** :ueforth-7.0.7.15.linux

```
mv ueforth-7.0.7.15.linux ueforth.bin
```

On vérifie que tout s'est bien déroulé avec une simple commande **ls** :



```
mpmp9@LAPTOP-KL9DD6UH:~$ cd ueforth
mpmp9@LAPTOP-KL9DD6UH:~/ueforth$ ls
ueforth.bin
mpmp9@LAPTOP-KL9DD6UH:~/ueforth$
```

Figure 2: le fichier ueforth.bin se trouve bien dans notre dossier ueforth

Il nous reste une dernière manipulation à effectuer, rendre ce fichier exécutable par le système Linux. Si vous avez l'explorateur de fichier **Nautilus** installé, ce qui est généralement le cas, tapez :

```
nautilus
```

Dans Nautilus, sélectionnez le fichier ueforth.bin, puis :

- clic droit → sélectionner *Properties*
- dans le pop-up, sélectionnez *Permissions*
- dans l'onglet *Permissions*, cochez *Execute*

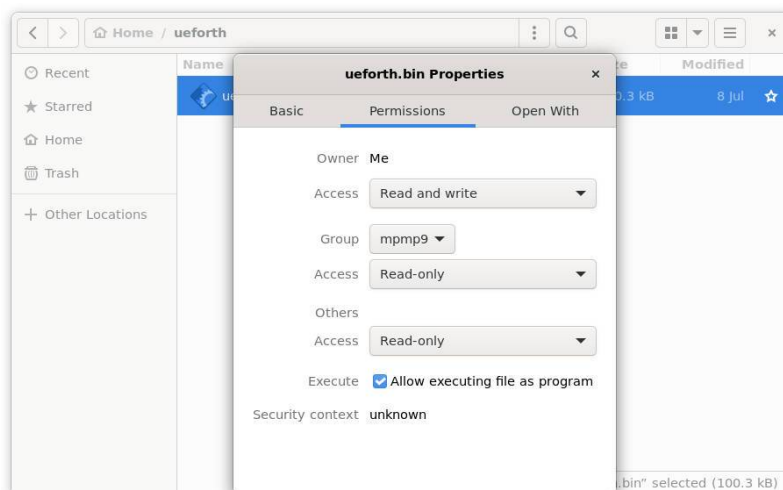


Figure 3: case Execute cochée rend le fichier exécutable comme programme sous Linux

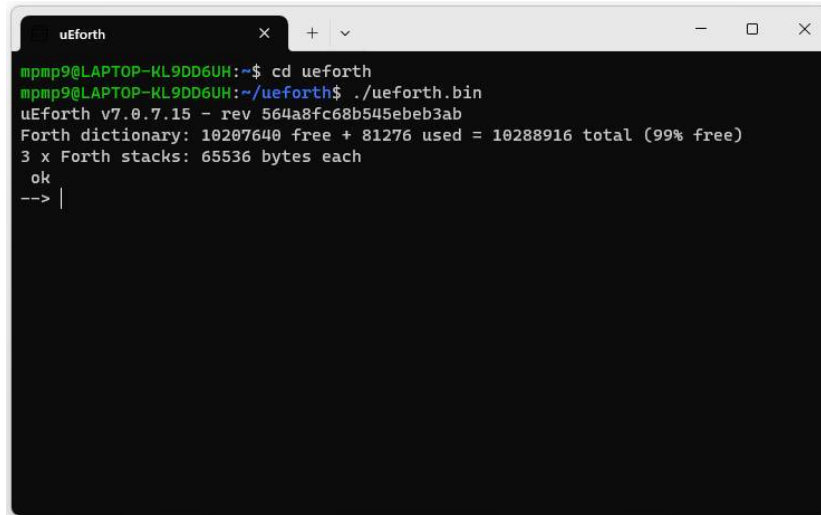
Et c'est fini !

Lancer eForth Linux

Pour lancer **eForth** au démarrage de **Linux** :

```
cd ueforth
./ueforth.bin
```

eForth Linux démarre aussitôt :



```
uEforth
mpmp9@LAPTOP-KL9DD6UH:~$ cd ueforth
mpmp9@LAPTOP-KL9DD6UH:~/ueforth$ ./ueforth.bin
uEforth v7.0.7.15 - rev 564a8fc68b545ebeb3ab
Forth dictionary: 10207640 free + 81276 used = 10288916 total (99% free)
3 x Forth stacks: 65536 bytes each
ok
--> |
```

Figure 4: eForth Linux est actif

Vous pouvez maintenant tester eForth et programmer vos premières applications en langage FORTH.

ATTENTION : cette version eForth gère les entiers au format 64 bits. C'est facile à vérifier :

```
cell . \ display : 8
```

Soit une dimension de 8 octets pour les entiers. Cet avertissement est indispensable si vous reprenez du code FORTH écrit pour des versions 16 ou 32 bits.

Bonne programmation.

Un vrai FORTH 64 bits avec eForth Linux

eForth Linux est un vrai FORTH 64 bits. Qu'est-ce que ça signifie ?

Le langage FORTH privilégie la manipulation de valeurs entières. Ces valeurs peuvent être des valeurs littérales, des adresses mémoires, des contenus de registres...

Les valeurs sur la pile de données

Au démarrage de eForth Linux, l'interpréteur FORTH est disponible. Si vous entrez n'importe quel nombre, il sera déposé sur la pile sous sa forme d'entier 64 bits :

```
35
```

Si on empile une autre valeur, elle sera également empilée. La valeur précédente sera repoussée vers le bas d'une position :

```
45
```

Pour faire la somme de ces deux valeurs, on utilise un mot, ici **+** :

```
+
```

Nos deux valeurs entières 64 bits sont additionnées et le résultat est déposé sur la pile. Pour afficher ce résultat, on utilisera le mot **.** :

```
. \ affiche 80
```

En langage FORTH, on peut concentrer toutes ces opérations en une seule ligne:

```
35 45 + . \ display 80
```

Contrairement au langage C, on ne définit pas de type **int8** ou **int16** ou **int32** ou **int64**.

Avec eForth Linux, un caractère ASCII sera désigné par un entier 64 bits, mais dont la valeur sera bornée [32..256[. Exemple :

```
67 emit \ display C
```

Les valeurs en mémoire

eForth Linux permet de définir des constantes, des variables. Leur contenu sera toujours au format 64 bits. Mais il est des situations où ça ne nous arrange pas forcément. Prenons un exemple simple, définir un alphabet morse. Nous n'avons besoin que de quelques octets :

- un pour définir le nombre de signes du code morse
- un ou plusieurs octets pour chaque lettre du code morse

```
create mA ( -- addr )
  2 c,
  char . c,  char - c,
```

```

create mB ( -- addr )
  4 c,
  char - c,  char . c,  char . c,  char . c,

create mC ( -- addr )
  4 c,
  char - c,  char . c,  char - c,  char . c,

```

Ici, nous définissons seulement 3 mots, **mA**, **mB** et **mC**. Dans chaque mot, on stocke plusieurs octets. La question est: comment va-t-on récupérer les informations dans ces mots?

L'exécution d'un de ces mots dépose une valeur 64 bits, valeur qui correspond à l'adresse mémoire où on a stocké nos informations morse. C'est le mot **c@** qui va nous servir à extraire le code morse de chaque lettre :

```

mA c@ . \ affiche 2
mB c@ . \ affiche 4

```

Le premier octet extrait ainsi va nous servir à gérer une boucle pour afficher le code morse d'une lettre :

```

: .morse ( addr -- )
  dup 1+ swap c@ 0 do
    dup i + c@ emit
  loop
  drop
;

mA .morse \ affiche .-
mB .morse \ affiche ...
mC .morse \ affiche -.-

```

Il existe plein d'exemples certainement plus élégants. Ici, c'est pour montrer une manière de manipuler des valeurs 8 bits, nos octets, alors qu'on exploite ces octets sur une pile 64 bits.

Traitement par mots selon taille ou type des données

Dans tous les autres langages, on a un mot générique, genre **echo** (en PHP) qui affiche n'importe quel type de donnée. Que ce soit entier, réel, chaîne de caractères, on utilise toujours le même mot. Exemple en langage PHP :

```

$bread = "Pain cuit";
$price = 2.30;
echo $bread . " : " . $price;
// affiche  Pain cuit: 2.30

```

Pour tous les programmeurs, cette manière de faire est LA NORME! Alors comment ferait FORTH pour cet exemple en PHP?


```

: pain s" Pain cuit" ;
: prix s" 2.30" ;
pain type    s" : " type    prix type
\ affiche    Pain cuit: 2.30

```

Ici, le mot **type** nous indique qu'on vient de traiter une chaîne de caractères.

Là où PHP (ou n'importe quel autre langage) a une fonction générique et un analyseur syntaxique, FORTH compense avec un type de donnée unique, mais des méthodes de traitement adaptées qui nous informent sur la nature des données traitées.

Voici un cas absolument trivial pour FORTH, afficher un nombre de secondes au format HH:MM:SS:

```

: :##
  # 6 base !
  # decimal
  [char] : hold
;
: .hms ( n -- )
  <# :## :## # # #> type
;
4225 .hms \ display: 01:10:25

```

J'adore cet exemple, car, à ce jour, **AUCUN AUTRE LANGAGE DE PROGRAMMATION** n'est capable de réaliser cette conversion HH:MM:SS de manière aussi élégante et concise.

Vous l'avez compris, le secret de FORTH est dans son vocabulaire.

Conclusion

FORTH n'a pas de typage de données. Toutes les données transitent par une pile de données. Chaque position dans la pile est TOUJOURS un entier 64 bits !

C'est tout ce qu'il y a à savoir.

Les puristes de langages hyper structurés et verbeux, tels C ou Java, crieront certainement à l'hérésie. Et là, je me permettrai de leur répondre : pourquoi avez-vous besoin de typer vos données ?

Car, c'est dans cette simplicité que réside la puissance de FORTH: une seule pile de données avec un format non typé et des opérations très simples.

Et je vais vous montrer ce que bien d'autres langages de programmation ne savent pas faire, définir de nouveaux mots de définition :

```

: morse: ( comp: c -- | exec -- )
  create
    c,
  does>
    dup 1+ swap c@ 0 do

```

```

        dup i + c@ emit
      loop
    drop space
  ;
2 morse: mA      char . c,   char - c,
4 morse: mB      char - c,   char . c,   char . c,   char . c,
4 morse: mC      char - c,   char . c,   char - c,   char . c,
mA mB mC        \ display   .- -... -.-.

```

Ici, le mot **morse:** est devenu un mot de définition, au même titre que **constant** ou **variable...**

Car FORTH est plus qu'un langage de programmation. C'est un méta-langage, c'est à dire un langage pour construire votre propre langage de programmation....

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, ESP32forth 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 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. ESP32forth 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
```

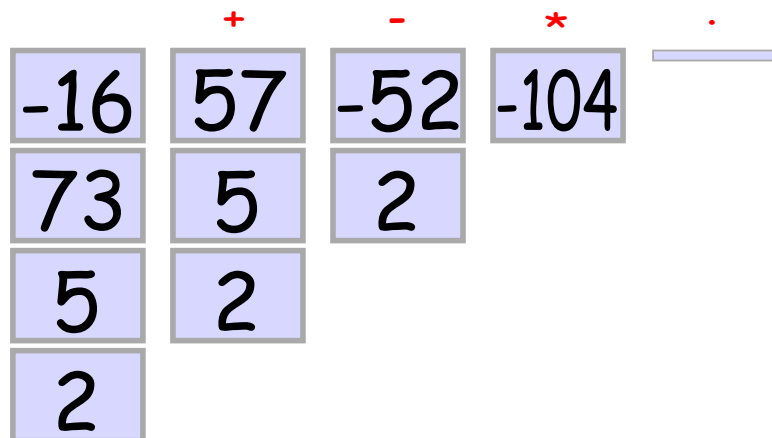
laisse la pile de paramètres dans l'état

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
```

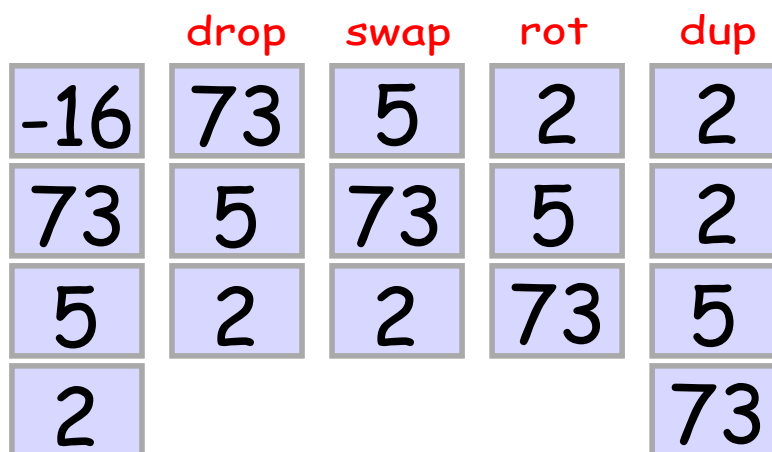
Notez que ESP32forth affiche commodément les éléments de la pile lors de l'interprétation de chaque ligne et que la valeur de -16 est affichée sous la forme d'entier non signé 32 bits. En outre, le mot `.` consomme la valeur de données -104, laissant la pile vide. Si nous exécutons `.` sur la pile maintenant 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, ESP32forth 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, ESP32forth 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 placée 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 ESP32forth. 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 ESP32forth, les nombres 32 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 une 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
```


Index lexical

c!.....	14	forget.....	11	11
c@.....	14	mémoire.....	14	11
constant.....	15	pile de retour.....	13	@.....	14
drop.....	13	value.....	15		
dup.....	13	variable.....	14		