

# Le langage Forth

## Brique ROSE

Samuel Tardieu  
sam@rfc1149.net

École Nationale Supérieure des Télécommunications

# Qu'est-ce que Forth ?

- Langage sans syntaxe
  - Mots séparés par des espaces
  - Tous les autres caractères sont des identificateurs valides : « drop », « ( », « s" »
  - Chaque nouveau mot devient partie intégrante du langage
  - Le principe de Forth est d'écrire des phrases décrivant le problème : le langage devient spécifique à l'application
- Historique
  - Créé par Charles H. Moore en 1968
  - Noyau extrêmement petit (typiquement quelques Ko)
  - Pas de cycle éditer/compiler/exécuter/debugger

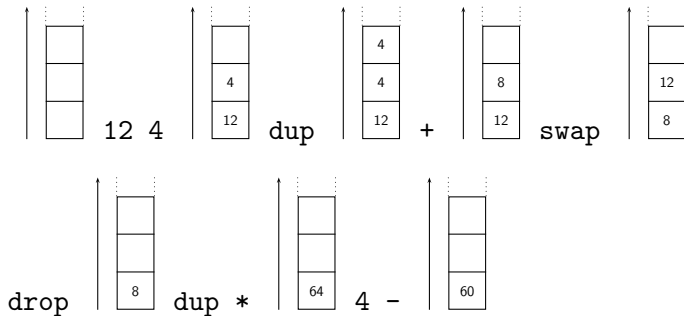
- Les mots peuvent être appelés de manière interactive dès qu'ils sont définis
- Les mots : et ; commencent et terminent une définition
- À partir des mots saisir, tourner et lacher, on peut définir le mot `deplacer` : `deplacer saisir tourner lacher ;`
- Le mot `deplacer` est inséré dans le dictionnaire. Lorsque l'utilisateur tape ce mot, il est recherché dans le dictionnaire et soit appelé (mode interprété) soit compilé (lors de la définition d'un nouveau mot)

# Le système Forth

- Mélange un interpréteur et un compilateur
- Chaque nouveau **mot** (équivalent d'une fonction en Forth) est compilé immédiatement et devient instantanément disponible
- Les mots sont séparés par un blanc ou un retour à la ligne
- Un dictionnaire contient l'adresse du code de chaque mot
- Une pile de données sert à passer les paramètres entre les mots

# Utilisation de la pile

En Forth, chaque mot manipule les objets au sommet de la pile (de données) contenant des entiers. Certaines opérations manipulent la pile.



# Exemples

- Définition d'un mot `carre` qui élève au carré  
: `carre dup *` ;
- Définition d'un mot `cube` qui élève au cube  
: `cube dup carre *` ;
- La phrase suivante affiche 27 (le mot `.` affiche (en le consommant) l'entier au sommet de la pile)  
`3 cube .`
- **Note** : seul le haut de la pile est important

- Forth dispose de deux modes. Le cœur lit un mot, puis agit :
  - **Interprétation** : c'est le mode par défaut ; tout mot qui est tapé est exécuté, tout nombre est placé sur la pile
  - **Compilation** : chaque mot ajoute un appel à ce mot dans la compilation courante, chaque nombre génère le code nécessaire pour placer ce nombre sur la pile
- Le mot `immediate` rend le dernier mot défini immédiat :
  - en mode compilation comme en mode interprétation, le mot est exécuté immédiatement
  - ce mode sert à implémenter de nouvelles constructions du langage

# Changement de mode

- : toto démarre la génération du mot toto
- ; termine la compilation du mot courant
- : carre dup \* ; génère en mémoire, à une adresse libre  $A$ , l'équivalent de

```
call dup  
call mult  
ret
```

et rajoute une entrée dans le dictionnaire pour carre pointant sur l'adresse  $A$ .

# Avantages du modèle

- Parseur simplissime (blanc ou retour-chariot)
- Interpréteur simplissime
- Compilateur simplissime (un système Forth complet tient typiquement en quelques Ko)
- Le système est réflexif
- Code très compact
  - La pile de données permet de ne pas gérer le passage de paramètres
  - Un mot machine par mot appelé
- Pas d'allocation de registres (voir cours de compilation)

- \ demande au parseur de vider le buffer d'entrée  
drop dup \ Commentaire
- ( cherche le caractère ) dans le flux d'entrée et positionne le parseur après  
drop ( supprime ) dup ( duplique )
- On indique souvent l'effet sur la pile de chaque mot dans des commentaires  
: calcule ( n -- x ) ... ;

- `if ...then`
  - `if` teste le sommet de la pile (et le consomme), et saute après le `then` s'il vaut zéro
  - Exemple : `test if agir then`
  - Le sens de `then` est celui de « puis ensuite » (à l'opposé d'autres langages ou il signifie « alors »)
- `if ...else ...then`
  - Si le test est négatif, `if` saute après le `else`
  - Exemple : `test if agir else dormir then`

- `do ...loop`
  - Les bornes (supérieure non incluses puis inférieure) sont placées sur la pile
  - `i` permet de récupérer l'indice de boucle le plus interne
  - Exemple : `10 0 do quelquechose loop`
- `do ...+loop` permet de préciser l'incrément avant `+loop`
- `?do ...loop` permet de ne pas exécuter une boucle si les deux bornes sont égales

## Boucles (2)

- `begin ...again`
  - Boucle infinie
  - Sortie possible par `exit` (sort du mot courant)
- `begin ...while ...repeat`
  - Boucle conditionnelle
  - Exemple : `begin test while quelquechose repeat`
  - Exemple : 7 n va exécuter xyz 7 fois  
: n begin dup while xyz 1- repeat drop ;
- `begin ...until` attend qu'une condition soit vraie

## Exemple : PGCD

Algorithme d'Euclide pour calculer le PGCD (tuck duplique le sommet de la pile après le deuxième élément, par exemple 1 2 3 tuck donne 1 3 2 3) :

```
: pgcd (a b -- n )          (avec a >= b)
  begin   ( a b )
    dup   ( a b b )
    while ( a b )
      tuck ( b a b )
      mod  ( b (a mod b) ) -> ( a b )
    repeat ( a b )
  drop   ( a )
;
```

# En vrai

Un programmeur Forth l'écrirait plutôt comme :

```
: pgcd ( a b -- n )  
  begin dup while tuck mod repeat drop ;
```

ou le factoriserait comme

```
: step ( a b -- b a%b ) tuck mod ;  
: pgcd ( a b -- n )  
  begin dup while step repeat drop ;
```

ou, sans le mot mod, ferait

```
: order ( a b -- a' >= b' ) 2dup < if swap then ;  
: step ( a b -- b a-b ) tuck - ;  
: pgcd ( a b -- n )  
  begin order dup while step repeat drop ;
```

- Le nombre d'octets d'un entier est appelée la **taille de cellule** (par exemple 4 sur un système 32 bits)
- Les mots @ et c@ lisent respectivement une cellule ou un mot à l'adresse se trouvant sur le sommet de la pile
- Les mots ! et c! écrivent respectivement une cellule ou un mot à l'adresse se trouvant sur le sommet de la pile
- Les mots lshift, rshift, and, or, xor et invert permettent de manipuler les bits  
5 2 lshift
- La variable base contient la base courante (de 2 à 36)

- Les entrées/sorties sont généralement à des adresses mémoire
- Supposons qu'un ensemble de 8 LEDs soient à l'adresse 0x1234 et qu'on souhaite les tester. On peut faire :

```
hex                \ 16 base !
ff 1234 c!         \ Allume les 8 leds
0 1234 c!          \ ^^c9teint les 8 leds
aa 1234 c!         \ Une sur deux
55 1234 c!         \ Allume les autres
1234 c@ invert 1234 c! \ Inverse les leds
```

# Allons plus loin

```
hex
: led@ ( -- n ) 1234 c@ ;
: led! ( n -- ) 1234 c! ;
: masque ( n -- 1<<n ) 1 swap lshift ;
: allume ( n -- ) masque led@ or led! ;
: eteint ( n -- ) masque invert led@ and led! ;
: inverse ( n -- ) masque led@ xor led! ;
```

```
0 allume \ Allume la led 0
3 eteint \ Eteint la led 4
7 inverse \ Inverse la led 7
```

- Sur certains systèmes sans fichiers, des **blocs** sont utilisés
- Un bloc fait 1024 octets (16 lignes de 64 caractères)
- Les blocs encourage la factorisation
- Les blocs sont rarement utilisés lorsqu'un système d'exploitation est présent

- Elles sont représentées par deux cellules : adresse et longueur
- Le mot `type` permet d'afficher une chaîne. `cr` passe à la ligne suivante. `s"` crée une chaîne allant jusqu'au `"` suivant. `."` affiche la chaîne allant jusqu'au `"` suivant.
- Exemple :

```
: hello ." Hello world!" cr ;
```

ou

```
: msg s" Hello world!" ;
```

```
: hello msg type cr ;
```

## Exemple : Manipulation de chaînes

Que font les mots suivants ?

```
: lower? ( c -- f ) 97 123 within ;  
( a b c within teste si b <= a < c )  
( 97 est l'ASCII de a, 122 de z, 65 de A )  
  
: upper ( c -- c' ) dup lower? if 32 - then ;  
: upper! ( a -- ) dup c@ upper swap c! ;  
: to-upper ( a n -- ) bounds ?do i upper! loop ;  
( bounds remplace a b par a a+b )  
  
: test s" hElLo WorLd!" 2dup to-upper type cr ;  
( 2dup recopie les deux elements au sommet )
```

- Forth est un standard international depuis 1994
- L'inventeur de Forth (Chuck Moore) s'en plaint. Selon lui
  - cela alourdit le langage
  - cela freine l'innovation
- Beaucoup de Forth ne sont pas conformes au standard (exemple : PicForth)
- Forth a maintenant de la couleur : ColorForth
  - Une couleur définit le rôle du mot : définition, utilisation immédiate, compilation, la base d'un nombre (décimal ou hexadécimal)
  - Plus de : et de `immediate`

- Sur les Forth de Chuck Moore, une partie des structures de contrôle n'existent plus
- `else` est supprimé : il suffit de factoriser

```
: mot test if action1 else action2 then action3 ;
```

devient

```
: sous-mot test if action1 exit then action2 ;
```

```
: mot sous-mot action3 ;
```

## Structures simplifiées (2)

- `begin...again` n'existe plus :  
: mot xyz begin action again ;  
devient  
: actions action recurse ;  
: mot xyz actions ;
- `begin...until non plus` :  
: mot begin action condition? until ;  
devient  
: mot action condition? if exit then recurse ;

# Exemple en ColorForth

- Bras articulé : commande de pince en 0xABCD, commande de moteur en 0x8A00, capteur de position en 0x83FA

```
serrer-pince 1 ABCD ! ;
ouvrir-pince 0 ABCD ! ;
droite 1 8A00 ! 0 8A00 ! ;
position 83FA @ ;
positionner dup position = if drop exit then droite
positionner ;
saisir 1 positionner serrer-pince ;
tourner 5 positionner ;
lacher ouvrir-pince ;
deplacer saisir tourner lacher ;
```

# En Forth classique

```
hex
: serrer-pince 1 abcd ! ;
: ouvrir-pince 0 abcd ! ;
: droite 1 8a00 ! 0 8a00 ! ;
: position 83fa @ ;
: positionner
  begin
    dup position <> while
      droite
    repeat drop ;
: saisir 1 positionner serrer-pince ;
: tourner 5 positionner ;
: lacher ouvrir-pince ;
: deplacer saisir tourner lacher ;
```