Travaux pratiques 3

En 1958, un jeune informaticien du nom de Charles H. Moore, entreprend le développement d'un langage de programmation pour faciliter son travail, qui consistait à calculer des trajectoires de satellites. Ce langage, prévu initialement pour les ordinateurs dits de *quatrième génération*, est nommé "Forth", et présente l'originalité d'utiliser explicitement la pile de données.

Dans ce TP, nous vous proposons d'implémenter un interpréteur pour un sous ensemble du langage Forth. Ce TP permettra de mobiliser des notions importantes et avancées du langage C telles que les structures, les unions, les pointeurs et les pointeurs de fonctions.

---

1 Implémentation d'une calculatrice en notation polonaise inversée

Dans un premier temps, on implémentera une simple calculatrice permettant de réaliser des calculs arithmétiques simples sur des nombres entiers. Ces calculs sont écrits sous la forme de la notation dite "polonaise inversée" : ainsi, le calcul :

$$(3 + 5) \times (4 - 2)$$

s'écrira sous la forme polonaise inversée de la manière suivante :

$$3\ 5\ +\ 4\ 2\ -\ *$$

Le programme fourni en entrée à l'interpréteur est constitué de mots ('3', '5', l'opérateur '+', etc) qu'on appelera *tokens*. Le comportement de l'algorithme attendu est le suivant :

```
Pour chaque token:
    - si le token est un nombre entier :
        empiler le nombre sur la pile
    - si le token est un opérateur arithmétique :
        dépiler les deux derniers éléments sur la pile
        réaliser l'opération arithmétique
        empiler le résultat sur la pile
```

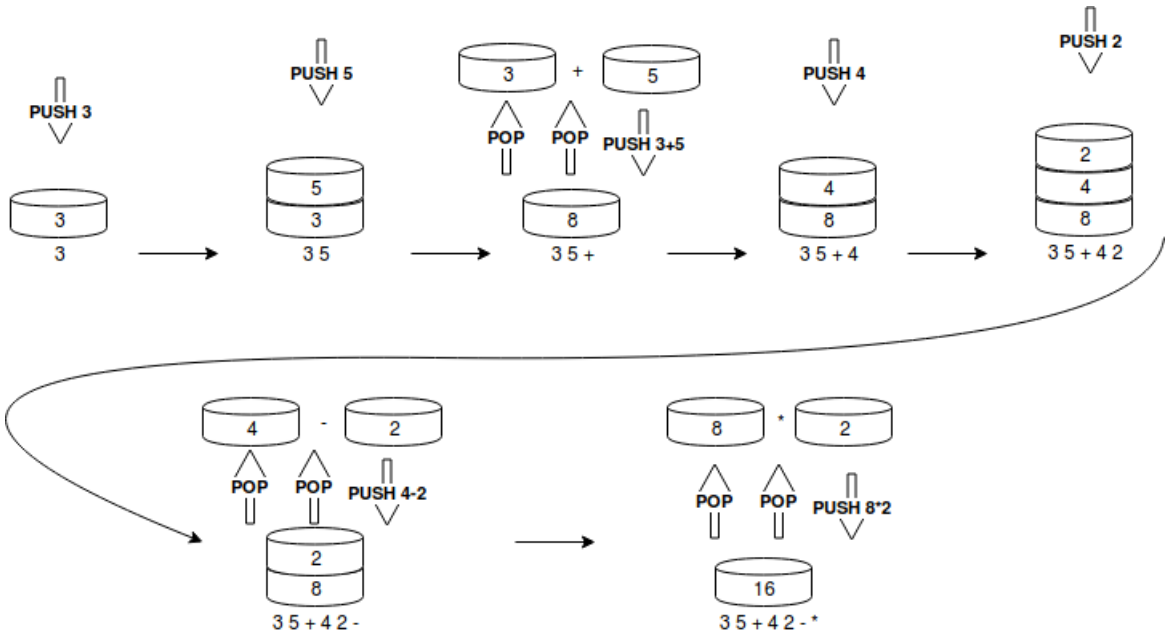

FIGURE 1 – Exemple d'exécution de l'algorithme pour *3 5 + 4 2 - **

1. Implémentez une structure de données de type Pile (*First In Last Out*) capable de manipuler des nombres entiers. Ecrivez des fonctions permettant :
   — la création de la pile
   — l'empilage d'un élément
   — le dépilage d'un élément
   — l'affichage de tous les éléments de la pile
2. Ajoutez le code suivant à votre projet :

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

typedef struct Programme {
    char **tokens;
    int taille;
} Programme;

int numberOfDelimiters(char* string) {
    int count = 0;
    for (int i=0;i<strlen(string);i++) {
        if (string[i] == ' ') count++;
    }
    return count;
}

Programme* lexer(char* chaine) {
    char *token,*str;
    str = strdup(chaine);
    int i=0;
    int arraysize = (numberOfDelimiters(str)+1);
    char** programme = (char**)malloc(sizeof(char*)*arraySize);

    while ( (token = strsep(&str, " ")) ) {
        programme[i] = token;
        i++;
    }
    Programme *retour = (Programme*)malloc(sizeof(Programme));
    retour->tokens = programme;
    retour->taille = i;
    return retour;
}
```

En utilisant la fonction *lexer*, générez un tableau de *tokens* à partir du premier argument reçu par le programme principal. Affichez les différents *tokens* sur la sortie standard au format suivant :

```
$ ./forth '3 5 + bonjour pouet'
    TOKEN : 3
    TOKEN : 5
    TOKEN : +
    TOKEN : bonjour
    TOKEN : pouet
```

3. Écrivez une fonction *void executer(Etat *etat)*, qui prendra en paramètre un pointeur vers une structure nommée Etat, contenant une pile de données et la structure Programme (qui contient elle même le tableau de *tokens*). Implémentez l'algorithme précédemment décrit, et affichez l'élément au sommet de la pile en fin d'exécution.

   Voici quelques exemples d'entrée que le programme devra être capable de gérer :

   ```
   $ ./forth '3 5 +'
   SORTIE : 8
   $ ./forth '3 5 + 4 2 -'
   SORTIE : 2
   $ ./forth '3 5 + 4 2 - *'
   SORTIE : 16
   $ ./forth '1 +'
   SORTIE : ERROR
   ```

4. Ajoutez à votre structure de Pile le support des nombres réels, et modifiez le code de la fonction exécuter pour prendre en compte cet ajout. On pourra utiliser la notion d'*union* afin d'adapter la pile à ce nouveau comportement, dont la syntaxe est la suivante :

   ```
   union Test {
       int ValEntier;
       float ValReel;
   };
   ```

2 Ajout des mots de manipulation de pile

Dans cette partie, on ajoutera des mots prédéfinis du langage permettant à l'utilisateur de manipuler la pile. Les mots que vous pouvez ajouter sont les suivants :

— DROP : supprime l'élément au sommet de la pile
— DUP : duplique l'élément au sommet de la pile
— SWAP : intervertit les deux éléments au sommet de la pile
— ROT : modifie l'ordre des trois éléments au sommet de la pile

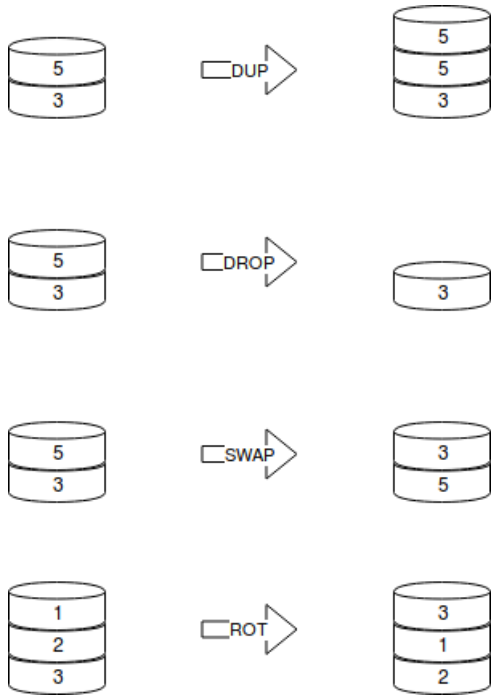

FIGURE 2 – Exemple d'exécution des mots clés DROP, DUP, SWAP, ROT

1. Ajoutez une structure de type "Liste Chaînée", dont les éléments (que nous appelerons *symboles*) sont capables de stocker un *token* et un pointeur de fonction correspondant à la définition de type suivante :

```
typedef int (*Commande)(Etat*) ;
```

Les pointeurs de fonction permettent de stocker l'adresse d'une fonction, permettant ainsi de l'exécuter facilement. Ainsi, un exemple d'utilisation d'un tel type pourrait être le suivant :

```
typedef int (*Commande)(Etat*) ;

int op_test(Etat* etat) {
```

```
    return 0;
}
int main(int argc, char **argv) {
    // Declaration du pointeur de fonction
    Commande f = op_test;
    // Appel de la fonction pointee par le pointeur
    f(NULL);
}
```

Ajoutez des fonctions permettant de manipuler cette structure pour :
- créer la table des symboles
- ajouter une entrée à la table des symboles
- chercher une entrée correspondant à un *token* dans la table des symboles
- initialiser la table des symboles avec une série de symboles existants

2. Définissez et implémentez les fonctions correspondant aux mots DUP, ROT, SWAP et DROP. Ajoutez les dans la fonction d'initialisation de la table des symboles.
3. Modifiez la structure *Etat* afin qu'elle puisse intégrer un pointeur vers la table des symboles.
4. Modifiez la fonction *executer* afin que, lorsqu'elle examine un *token* défini dans la table des symboles, la fonction dont l'adresse est stockée dans le pointeur de fonction soit exécutée. Au terme de cette modification, l'interpréteur doit être capable d'interpréter les entrées suivantes :

```
$ ./forth '1 2 + DUP *'
SORTIE : 9
$ ./forth '1 2 DROP'
SORTIE : 1
$ ./forth '2 3 SWAP -'
SORTIE : 1
$ ./forth '3 2 1 ROT'
SORTIE : 3
```

3 Ajout des mécanismes d'affichage

Le langage Forth dispose de trois mécanismes d'affichage :
— le mot clé . (*point*), qui dépile l'élément au sommet de la pile et l'affiche
— le mot clé *.S*, qui affiche le contenu de la pile (non destructif)
— les mots clés *."* et *"*, qui permet de réaliser un affichage de chaîne de caractères. Par exemple :

```
./forth '." bonjour tout le monde "'
```

affichera la chaîne de caractères 'bonjour tout le monde'.

1. Supprimez le mécanisme d'affichage du dernier élément de la pile en fin d'exécution. Ajoutez dans la table des symboles une entrée permettant de gérer le token . (*point*), et implémentez la fonction correspondante. Pensez à adapter la chaîne de formatage de votre affichage en fonction du type de données à afficher, votre pile étant capable de gérer des nombres entiers et des nombres réels.
2. Ajoutez dans la table des symboles une entrée permettant de gérer le token *.S*, et implémentez la fonction correspondante.
3. Ajoutez dans la structure *Etat* un ou plusieurs champs, correspondant au mode actuel. Ce mode peut prendre deux valeurs :
   — EXECUTER : interprète le token courant
   — AFFICHER : affiche le token courant sous la forme d'une chaîne de caractères

   Vous pouvez implémenter ce mode sous la forme d'un type énuméré.
4. Modifiez la fonction *executer* afin qu'elle passe en mode AFFICHER lorsque le token *."* est rencontré, et qu'elle repasse en mode EXECUTER lorsque le token *"* est rencontré. Adaptez le code afin que nombres et mots clés ne soient pas exécutés en mode AFFICHER.
5. Ajoutez dans la table des symboles une entrée permettant de gérer le token *CR* (*Carriage Return*), dont le comportement est de provoquer un saut de ligne dans l'affichage. Implémentez la fonction correspondante.

   À ce stade, les exemples d'exécution suivants devraient être fonctionnels :

```
$ ./forth '." hello world 3 DUP pouet"'
hello world 3 DUP pouet
```

```
$ ./forth '1 2 . .'
1 2
$ ./forth '1 2 3 .S CR DROP .S CR DROP .S CR'
3 2 1
2 1
1
$ ./forth '." le carré de trois est " 3 DUP * . CR'
le carré de trois est 9
```

4 Ajout des structures conditionnelles

Le langage Forth dispose d'un mécanisme de structures conditionnelles, sous la forme :

`IF ... [ELSE ...] THEN`

Lorsque le token *IF* est rencontré, il examine l'élément au sommet de la pile comme un booléen. Arbitrairement, la valeur TRUE est représentée par -1 et la valeur FALSE par 0.

1. Implémentez les opérateurs binaires de comparaison arithmétique =, < et >. Ajoutez les entrées correspondantes dans la table des symboles.
2. Ajoutez un mode IGNORER à votre type énuméré *Mode*.
3. Ajoutez les tokens *IF*, *ELSE* et *THEN* à votre table des symboles, et implémentez les comportements correspondants. L'algorithme à implémenter est le suivant :

```
Lorsqu'on observe le token IF :
    - dépiler l'élément au sommet de la pile
    - s'il vaut 0, passer en mode IGNORER
    - sinon, rester en mode EXECUTER
Lorsqu'on observe le token ELSE :
    - si on est en mode IGNORER, passer en mode EXECUTER
    - si on est en mode EXECUTER, passer en mode IGNORER
Lorsqu'on observe le token THEN :
    - si on est en mode IGNORER, passer en mode EXECUTER
```

A ce stade, l'interpréteur devrait être capable d'exécuter les programmes suivants :

```
$ ./forth '19 DUP 18 < IF ." mineur " ELSE 18 = IF ." 18 ans "
ELSE ." majeur " THEN THEN CR'
majeur
$ ./forth '12 DUP 18 < IF ." mineur " ELSE 18 = IF ." 18 ans "
ELSE ." majeur " THEN THEN CR'
mineur
$ ./forth '18 DUP 18 < IF ." mineur " ELSE 18 = IF ." 18 ans "
ELSE ." majeur " THEN THEN CR'
18 ans
```

5 Pour aller plus loin ...

Si vous le souhaitez, vous pouvez implémenter en autonomie les deux fonctionnalités supplémentaires suivantes :

— La fonctionnalité de boucle, dont la syntaxe est la suivante :

```
BEGIN ... UNTIL
```

qui a pour effet de répéter les opérations contenues entre les deux mots clés tant que l'élément au sommet de la pile correspond à un booléen évalué à TRUE.

— La fonctionnalité de définition de mots, dont la syntaxe est la suivante :

```
: NOUVEAUMOT ... ;
```

Cette fonctionnalité permet d'enregistrer une séquence d'opérations à effectuer si le mot NOUVEAUMOT est rencontré plus tard lors de l'exécution. Par exemple :

```
: CARRE DUP * ;
```

définira un nouveau mot CARRE, dont l'exécution provoquera l'exécution de la séquence de tokens :

```
DUP *
```

Pour implémenter ces fonctionnalités, il est pertinent d'ajouter une seconde pile, contenant des "adresses" (correspondant à l'indice d'un token

dans le tableau généré par la fonction *lexer*). Cette pile permettra de stocker l'emplacement auquel sauter lors de la fin de l'exécution d'un mot défini par l'utilisateur, ou l'indice du début d'une boucle.

Il est également nécessaire d'enrichir la table des symboles afin qu'elle soit capable d'enregistrer dynamiquement de nouveaux mots définis par l'utilisateur. Les entrées doivent pouvoir indiquer une "adresse" (ou indice d'un token) correspondant au début d'un mot (on ne définira pas de pointeur de fonction pour les mots de l'utilisateur).

Attention, le code précédemment écrit (notamment l'algorithme implémentant le *IF*) nécessite quelques adaptations, afin de gérer les cas complexes mélangeant boucles, mots et/ou conditions.

Les exécutions suivantes doivent être fonctionnelles au terme de ces ajouts :

```
$ ./forth ': CARRE DUP * ; : CUBE DUP CARRE * ; ." Le cube de 5 est "
5 CUBE . CR'
Le cube de 5 est 125

$ ./forth ': MAJEUR DUP 18 < IF ." mineur " ELSE 18 = IF ." 18 ans "
ELSE ." majeur " THEN THEN ; 2 MAJEUR CR'
mineur
$ ./forth ': ETOILE ." * " ; : LIGNE DUP BEGIN ETOILE 1 - DUP UNTIL ;
: TRIANGLE DUP BEGIN DUP LIGNE SWAP 1 - DUP CR UNTIL ; 10 TRIANGLE'
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```