

CYCLE L (L'3 2012)

LANGAGES ET COMPILATION TP N°2

But: Application concrète construite avec LEX et YACC.

Sujet : BFSHell, BF2C, C2BF, LL Parser.BF ★★★

Introduction

Le brainfuck (BF), langage constitué de seulement huit instructions, est (presque¹) le plus petit langage Turing Complet du monde. Le caractère Turing-complet signifie qu'il est possible (en théorie), de réaliser toute machine de Turing, donc en particulier, un analyseur syntaxique.

Langage issu de la contre-culture geek, il est destiné, par facétie, à maximiser l'effet connu de l'arithmétique des pointeurs sur le programmeur C débutant.

Indépendamment de son caractère amusant, ce langage comporte la vertu pédagogique de permettre, sur la base d'un noyau de langage minimal, d'aborder toute la panoplie théorique et pratique associée à la théorie des langages et aux techniques de compilation.

Pour en savoir d'avantage, une sélection de ressources Internet est donnée en fin de sujet.

Architecture et fonctionnement

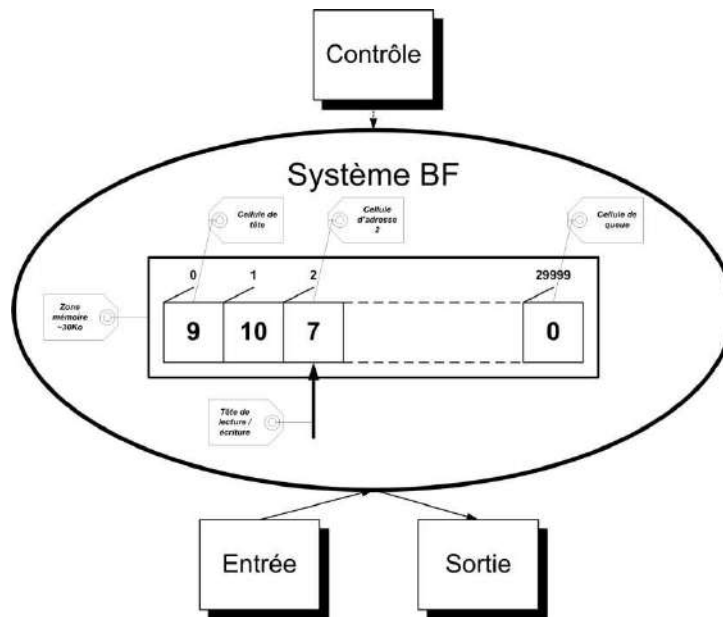
Brainfuck fonctionne sur une architecture très primitive :

- Une zone mémoire linéaire de 30000 cellules (en 8 bits ou en 16 bits).
- Une tête de lecture / écriture (LE pointeur) qui peut se déplacer séquentiellement vers la droite ou vers la gauche de la zone mémoire, à raison d'un pas d'une cellule à la fois, et peut y incrémenter ou décrémenter le contenu (à raison de l'unité).
- L'entrée et la sortie standards, à partir desquelles il est possible de lire ou écrire à raison d'un caractère à la fois.
- Une séquence d'instructions (le programme) qui contrôle l'exécution, et qui comporte en outre la possibilité de reprise en amont d'une série d'instructions (boucle) si la cellule pointée au moment de faire le choix de boucler, ne contient pas 0).

¹ Voir le langage $\mathcal{P}$ inventé en 1964 par Corrado Böhm : <http://en.wikipedia.org/wiki/P>

"On a family of Turing machines and the related programming language", ICC Bull. 3, 185-194, July 1964.

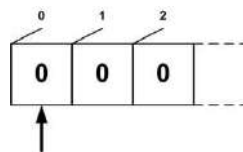
"Flow diagrams, Turing machines and languages with only two formation rules", CACM 9(5), 1966.



Architecture

La configuration initiale (avant d'exécuter la première instruction) :

- Toutes les cellules de la zone mémoire sont initialisées à zéro.
- LE pointeur est positionné sur la première cellule (la plus à gauche).

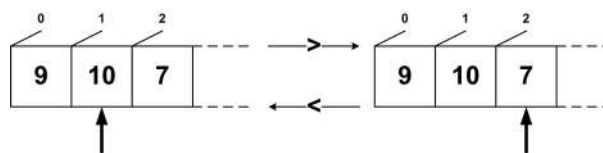


Configuration initiale

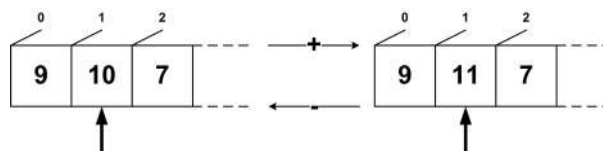
Il existe bien entendu des situations d'erreur en cas de débordement de la zone.

Langage, sémantique et équivalence en C

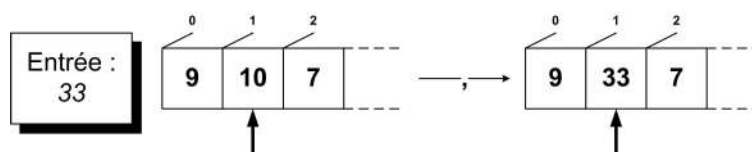
Les huit instructions élémentaires, lesquelles viennent par couples d'opposés sont les suivantes :



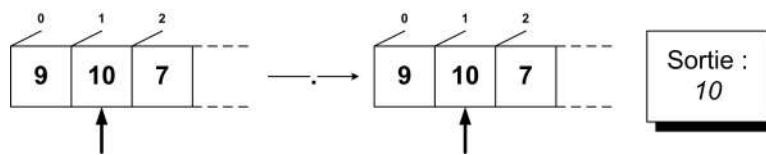
Instructions '<' et '>' de contrôle du déplacement de la tête (qui indique la cellule courante)



Instructions '+' et '-' pour modifier le contenu de la cellule courante



Instruction ';' pour copier le contenu de l'entrée dans la cellule courante



Instruction ‘.’ pour copier le contenu de la cellule courante sur la sortie

Les deux instructions ‘[’ et ‘]’ n’agissent que sur le flux de contrôle, la première en marquant un point de reprise, la seconde en déclenchant la réinterprétation de toute la séquence d’instructions amont, depuis le point de reprise, si et seulement si la valeur de la cellule courante est non nulle.

Equivalence BF et C

Tableau d’association sémantique (instruction BF et son équivalent en C)

But	Direction 1		Direction 2	
	BF	C	BF	C
Adressage	>	++p;	<	--p;
Affectation	+	++*p;	-	--*p;
Entrées/Sorties	,	*p = getchar();	.	putchar(*p);
Boucles	[	while (*p) {	]	}

Cette équivalence s’établit dans le contexte de l’insertion de la traduction C de la séquence d’instructions BF au cœur du squelette suivant :

```
#include <stdlib.h>
#include <stdio.h>

unsigned char m[30000] = {0};
unsigned char *p = m;

int main()
{
    ...
    return EXIT_SUCCESS;
}
```

Partie I (Base minimale pour viser la moyenne)

Il vous est demandé de construire à l’aide de LEX, YACC, et des librairies standards du C, la boîte à outils suivante :

1. **BFShell** – Un interpréteur BF : il exécute un programme BF.
2. **BF2C** – Un traducteur (générateur de code) de BF vers C : il produit un programme C compilable et exécutable.
3. **C2BF** – Un traducteur de C vers BF : il prend en entrée une version d’un fichier C qui contient le sous-ensemble d’instructions ci-dessus, et produit en sortie la traduction BF
4. Vous pouvez étendre le traducteur pour qu’il sache interpréter des macros : cela pourra vous faciliter la tâche pour les parties suivantes.

Partie II (Pour creuser l'écart)

Il vous est demandé de réaliser en BF un analyseur LL.

Ce programme lit en entrée un mot constitué de 'a' et de 'b' et produit au fur et à mesure de son exécution la trace de ses décisions (décalages, réductions, ERR, ACC).

Vous devez permettre qu'il puisse être configuré (ou produire trois programmes distincts) pour reconnaître les mots de chacune des grammaires suivantes :

$$S \rightarrow aS \mid \varepsilon$$

$$S \rightarrow aSa \mid bSb \mid \varepsilon$$

$$S \rightarrow aSb \mid bSa \mid \varepsilon$$

Partie IIIA (Pour aller chercher une note d'excellence)

Pour celles et ceux qui souhaitent mettre en œuvre une analyse LL(k) en mode LL(1), une surprise intéressante les attend avec les deux grammaires suivantes :

$$S \rightarrow aSbS \mid bSaS \mid \varepsilon$$

$$S \rightarrow aSaS \mid bSbS \mid \varepsilon$$

Partie IIIB (Pour aller chercher une note d'excellence)

Au-delà de l'interprétation et de la traduction sur base d'analyses lexicale et syntaxique, il est un sous-domaine important de l'art de la compilation : l'optimisation de code.

En raisonnant à partir d'équivalence logique entre différents jeux d'instructions (par exemple succession de déplacements droits et gauches de la tête de lecture, ou bien succession d'incrémentations et de décrétements du contenu d'une cellule), vous pouvez vous servir de l'arbre d'analyse comme pivot à un raisonnement sémantique dont le but est de réécrire le code dans une version qui fait la même chose, mais en mieux.

Vous pouvez vous donner trois objectifs d'optimisation, et les tester sur des programmes BF existants : programme le plus court, programme le plus rapide, programme le moins gourmand en mémoire.

Livraison

Pour chaque binôme, remise des travaux (une seule par binôme) au plus tard lundi quinze février 2010 à 23:59 (au-delà, pénalité de trois points/jours au prorata horaire).

Envoi par email à franck.lepoivre@gmail.com , en précisant « TP LC 2 LP3 – <NOM 1> – <NOM 2> » dans l'objet.

Forme et contenu du livrable : une archive (RAR), de nom « TP LC 2 LP3 – <NOM 1> – <NOM 2> » contenant votre code source, vos exécutables et un README.txt pour mode d'emploi contenant l'entête :

```
Equipe
<Prénom1> <NOM1>
<Prénom2> <NOM2>
Groupe <Lettre Groupe>
TP 2 - Brainfuck
```

Ressources externes

A propos de BF

<http://fr.wikipedia.org/wiki/Brainfuck>

<http://fr.wikipedia.org/wiki/Turing-complet>

<http://esoteric.voxelp perfect.net/wiki/Brainfuck>

Doc technique potentiellement utile

<http://www.lysator.liu.se/c/ANSI-C-grammar-y.html>

<http://www.delafond.org/traducmanfr/man/man1/flex.1.html>

Quelques autres langages minimaux

<http://en.wikipedia.org/wiki/P>

<http://en.wikipedia.org/wiki/Malbolge>

<http://fr.wikipedia.org/wiki/Whitespace>

http://esoteric.voxelp perfect.net/wiki/Language_list

Un exemple d'interpréteur BF écrit en C

```

/** pepper labs inside (copyleft) */
#include <stdlib.h>
#include <stdio.h>

unsigned char m[30000] = {0};
unsigned char *p = m;

void exec(char cmd, FILE *fp)
{
    char c;
    long pos;
    switch (cmd)
    {
        case '<' : --p; break;
        case '>' : ++p; break;
        case '+' : ++*p; break;
        case '-' : --*p; break;
        case '.' : putchar(*p); break;
        case ',' : *p = getchar(); break;
        case '[' :
            /// mémorise la position de '[' dans le programme BF
            pos = ftell(fp);
            /// ttq l'itérateur n'est pas null
            while(*p != 0)
            {
                /// repositionne la lecture du programme BF en début de boucle
                fseek(fp, pos, SEEK_SET);
                /// appel récursif de la fonction principale d'interprétation..
                do exec(c =getc(fp), fp);
                /// .. ttq ] ou EOF ne sont pas rencontrés
                while (c != ']' && c != EOF);
            }
        }
    }
}

```

```

int interpret(const char *bf_file)
{
    char cmd;
    FILE *fp = fopen(bf_file, "r");

    if (!fp)
    {
        fprintf(stderr, "Error: no %s file !!\n", bf_file);
        return EXIT_FAILURE;
    }

    while ((cmd = getc(fp)) != EOF) exec(cmd, fp);

    fclose(fp);

    return EXIT_SUCCESS;
}

int main(int argc, char **argv)
{
    if (argc > 1) return interpret(argv[1]);
    else
    {
        fprintf(stderr, "Error: bf file awaited as first argument !!\n");
        return EXIT_FAILURE;
    }
}

```