▶ **HX4 & HX1 – Option Informatique**
Sixième TP Caml
Vendredi 1 avril 2011

Brainfuck et Langton

Extrait de Wikipedia : *Brainfuck est un langage de programmation minimaliste, inventé par Urban Müller en 1993. Il tire son nom de l'union de deux mots anglais, brain (cerveau) et fuck (foutre), allusion transparente à l'expression « masturbation intellectuelle ». Ce vocabulaire peu flatteur lui a d'ailleurs valu d'être écrit sous d'autres orthographes plus prudes, telles que Brainf*ck, Brainf*** ou encore BF. [...]*
L'objectif de Müller était de créer un langage de programmation simple, destiné à fonctionner sur une machine de Turing, et dont le compilateur aurait la taille la plus réduite possible. Le langage se satisfait en effet de seulement huit instructions. La version 2 du compilateur originel de Müller, écrit pour l'Amiga, ne pesait lui-même que 240 octets, la version actuelle se contentant de 171 octets. Le brainfuck est pourtant un langage Turing-complet, ce qui signifie que, malgré les apparences, il est théoriquement possible d'écrire n'importe quel programme informatique en brainfuck. La contrepartie est que, comme son nom ne le suggère peut-être pas pour un non-anglophone, le langage brainfuck produit des programmes difficiles à comprendre. Il suit un modèle de machine simple, consistant en un tableau d'octets initialisés à 0, d'un pointeur sur le tableau (positionné sur le premier octet du tableau) et de deux files d'octets pour les entrées et sorties.
Les instructions du BF :

- *'>' et '<' déplacent le pointeur vers la case de droite ('>') ou de gauche ('<') de la case courante*
- *'+' et '-' modifient la valeur de la case courante : '+' ajoute 1, '-' enlève 1*
- *'.' affiche le caractère ASCII de l'entier contenu dans la case courante*
- *',' lit un caractère ASCII, et le stocke sous forme d'entier dans la case courante*
- *'[' saute l'instruction suivant le ']' correspondant si la case courante vaut 0*
- *']' revient l'instruction suivant le '[' correspondant si la case courante ne vaut pas 0*

1 Brainf*ck

1.1 Programmes en BF

▶ **Question 1** *Que font les trois programmes suivants :*

```
,+.    [-.]      [[-]>]
```

?

▶ **Question 2** *Écrire un programme en BF qui fasse la somme de deux entiers, puis un qui fasse le produit.*

1.2 Analyse syntaxique

▶ **Question 3** *Écrire une fonction qui étant donné une chaîne de caractère renvoie une liste de caractère.*

1.3 Analyse sémantique

```
type instruction =
  | Bouge of int
  | Ajoute of int
  | Lire
  | Ecrire
  | Boucle of instruction list
```

▶ **Question 4** *Écrire une fonction qui étant donné une liste de (+ - . , < > [ ]) renvoie une paire (instruction list×char list) (le premier élément de cette liste correspond à tout ce que la fonction a pu lire, le second correspond à ce que la fonction n'a pas pu lire,ie rien ou ce qui suit un]). Ce n'est pas grave si votre programme accepte des sources où il manque des] à la fin.*

1.4 Interprétation

▶ **Question 5** *Écrire une fonction qui évalue une instruction list. On pourra définir un tableau, une référence vers la tête de la machine puis une sous-fonction qui évalue une instruction list. Pour l'instruction ',' on peut utiliser la fonction read_char, disponible sur mon site.*

▶ **Question 6** *En déduire une fonction qui fait tout.*

▶ **Question 7** *Essayer votre programme avec des programmes BF que vous pouvez écrire, regarder sur internet, récupérer sur la page des TPs.*

2 La fourmi de Langton

2.1 Règles

Les cases d'une grille peuvent être blanches ou noires. On considère arbitrairement l'une de ces cases comme étant la "fourmi". Dans l'état initial, toutes les cases sont de la même couleur.

La fourmi peut se déplacer à gauche, à droite, en haut ou en bas d'une case à chaque fois selon les règles suivantes :

- Si la fourmi est sur une case noire, elle tourne de 90 ° vers la droite, change la couleur de la case en blanc et avance d'une case.
- Si la fourmi est sur une case blanche, elle tourne de 90 ° vers la gauche, change la couleur de la case en noir et avance d'une case.

2.2 Simulation

▶ **Question 8** *Écrire une fonction $fourmi$ qui prend en entrée un quadruplet (x, y, dir, col) x,y la position,dir la direction courante, couleur la couleur de la fourmi (on dessinera plusieurs fourmis ensuite) et renvoie $(nx, ny, ndir, col)$ correspond à la position et la direction mise à jour.*

▶ **Question 9** *En déduire une fonction qui affiche la suite des itérations d'une fourmi de Langton.*

▶ **Question 10** *En déduire une fonction qui affiche la suite des itérations de trois fourmis de Langton.*

▶ **Question 11 *** *Selon vous, les fourmis de Langton atteindront elles une orbite périodique ?*

▶ **HX4 & HX1 – Option Informatique**
Sixième TP Caml
Vendredi 1 avril 2011

Brainfuck et Langton

Un corrigé

▶ **Question 1** Ajouter un au caractère ASCII, afficher une suite décroissante pour leur code ASCII de caractère, remettre la bande à zéro.

▶ **Question 2** Cela donne

```
,>,[-<+>]<.

,>,<[ > [->+>+<<] >[-<+>] <<- ]>>>.
```

▶ **Question 3**

```
let decomp s =
  let rec foo i =
    if string_length s = i
    then []
    else if mem s.[i] ['+';'-';'<';'>';'.';',';'[';']']
    then s.[i]::(foo (i+1))
    else (foo (i+1))
  in
  foo 0
```

▶ **Question 4**

```
let get = function
  | '+' -> Ajoute(1)
  | '-' -> Ajoute(-1)
  | '>' -> Bouge(1)
  | '<' -> Bouge(-1)
  | '.' -> Ecrire
  | ',' -> Lire
(* renvoie la liste etudiee et le reste *)
let rec parse = function
  | [] -> ([],[])
  | ']'::q -> ([],q)
  | '['::q ->
    let (lu,reste) = parse q in
    let (nlu,nreste) = parse reste in
    (Boucle(lu)::nlu,nreste)
  | a::q ->
      let (lu,reste) = parse q in
      ((get a)::lu,reste)
```

▶ **Question 6**

```
let eval s =
  let t = make_vect 100000 0 in
  let i = ref 0 in
  let rec foo = function
    | [] -> ()
    | Bouge(dec)::q -> i:=!i+dec;foo q
    | Ajoute(dec)::q ->t.(!i)<-((t.(!i)+dec)+256) mod 256;foo q
    | Lire::q -> t.(!i) <- int_of_char(read_char ()) ;foo q
    | Ecrire::q -> print_char (char_of_int t.(!i)) ; foo q
    | Boucle(l)::q ->
        while t.(!i)<>0 do
          foo l
        done ;
        foo q
  in
foo (fst (parse ( decomp s )) )
```

▶ **Question 8**

```
let fourmi (x,y,dir,c) =
  let des_c,n_dir = (if point_color (x*taille) (y*taille) <> black
    then (black,(dir+1) mod 4)
    else (c,(dir+3) mod 4)) in
      set_color des_c ;
  fill_rect (x*taille) (y*taille) taille taille ;
  ((x+dir_x.(n_dir)+800/taille) mod (800/taille),
  (y+dir_y.(n_dir)+800/taille) mod (800/taille),n_dir,c)
```

▶ **Question 9**

```
let rec dooone f1 =
  dooone (fourmi f1)
```

▶ **Question 10**

```
let rec doo f1 f2 f3 =
  doo (fourmi f1) (fourmi f2) (fourmi f3)

let c = set_color black ; fill_rect 0 0 800 800 ;
 doo (45,45,0,red) (145,145,0,green) (245,245,0,blue)
```