▶ **HX4 & HX1 – Option Informatique**
Troisième TP Caml
Vendredi 4 février 2011

Programmation impérative et backtrack

1 Programmation impérative

Dans toute cette partie, il est interdit d'utiliser des fonctions récursives.

1.1 for

La syntaxe du for est la suivante :

```
for nom_variable = valeur_debut to valeur_fin do
        expr_de_type_unit
done

(* expr_de_type_unit peut ne pas etre de type unit mais c'est moche! *)
```

par exemple

```
for i = 0 to 5 do
        print_int i
done
(*affiche "012345".*)
```

La syntaxe du while c'est :

```
while condition do
        expr
done
```

▶ **Question 1** *Faites un programme qui prend en entrée n qui affiche une ligne composée de n 'X'*

▶ **Question 2** *Faites un programme, sans appels de fonctions auxiliaires, qui prend en entrée n et qui affiche un triangle composé de 'X' : par exemple pour $n = 2$ il doit afficher*

```
X
XX
```

et pour $n = 3$

```
X
XX
XXX
```

1.2 ref

La syntaxe des références est la suivante :

```
let ma_ref = ref ma_valeur
(* Declare une reference qui contient ma_valeur *)
!ma_ref
(* Pour acceder a la valeur contenue dans la reference*)
ma_ref := nouvelle_valeur
(*Pour changer la valeur d'une reference *)
```

▶ **Question 3** *Qu'affiche le programme suivant*

```
let a = ref 5 ;;
let b = ref 13 ;;
a:=!b+!a*!a;;
b:=!a+4;;
print_int !b;;
```

1.3 Tableaux

Un tableau est une structure de donnée qui permet d'accéder au i^eélément en temps constant et de le modifier. La syntaxe est la suivante :

```
make_vect taille_tableau valeur_defaut (*fabrique un tableau *)
[|v_1;v_2;v_3;v_4|] (*fabrique aussi un tableau *)
mon_tab.(i) (*accede au i+1eme element de mon_tab *)
mon_tab.(i) <- nouvelle valeur (*remplace la i+1eme case du tableau *)
vect_length mon_tab (*renvoie la taille du tableau *)
```

▶ **Question 4** *Que renvoie le programme suivant :*

```
let a = make_vect 10 5 in
let b = [|5;3;4;6;2;3;2|] in
a.(0) <- b.(a.(0));
b.(a.(0))+(vect\_length b)
```

1.4 strings

La syntaxe des chaînes de caractères (type string) est très proche de celle des tableaux :

```
"ma chaine de caractere" (* creation d'une chaine de caractere *)
make_string taille_chaine 'car' (* Peu utile mais existe *)
ma_chaine.[i] (* acces au ieme caractere, attention les indices commencent a 0*)
ma_chaine.[i] <- nouveau_caractere (* remplace le ieme caractere *)
string_length ma_chaine (* taille de la chaine*)
```

▶ **Question 5** *Faites un programme qui retourne une chaine de caractère. Exemple : le retourné de "maman" est "namam".*

1.5 Quelques entrainements

▶ **Question 6** *Faites un programme qui renvoie le maximum d'un tableau.*

▶ **Question 7** *Faites un programme qui renvoie l'indice du maximum d'un tableau.*

▶ **Question 8** ***Tri à bulle*** *Faites un programme qui prend un tableau et qui renvoie le tableau trié par la méthode du tri à bulles. (Tant que le tableau n'est pas trié on le parcourt et on inverse les paires d'éléments consécutifs qui ne sont pas dans le bon ordre.)*

2 Backtrack

Le backtrack c'est une classe d'algorithme pour résoudre des problèmes. L'idée c'est que pour trouver l'existence de solution à un problème, on va distinguer plusieurs cas et tenter de résoudre chacun des cas séparément. Par exemple si je veux résoudre un sudoku, je vais distinguer selon le chiffre qu'il peut y avoir dans tel case, essayer de résoudre le sudoku avec cette valeur et si cela ne marche pas je supposerais une autre valeur pour la case et recommencerais.

2.1 Sudoku solver

Dans cette partie on va écrire un programme capable de résoudre un sudoku. Le sudoku est représenté par un tableau 9×9. Pour construire un tel tableau on utilise la fonction make_matrix tailleX tailleY valeur.

▶ **Question 9 *** *Pourquoi serait-il faux de construire un tableau bidimensionnel avec*
make_vect tailleX (make_vect tailleY valeur) ?

▶ **Question 10** *Faites un programme qui prend en entrée un tableau 9×9, un x, un y et un entier c avec* $1 \leq c \leq 9$ *renvoie true si poser c en (x,y) ne rentre pas en conflit avec les cases déjà remplies de la grille.*

▶ **Question 11 *** *En vous inspirant de la méthode du backtrack, faites un programme qui résout le sudoku.*

2.2 Le problème des huit dames

Le problèmes des huit dames est de trouver la position de huit dames sur un échiquier de telle manière qu'elles ne puissent pas se prendre les unes les autres.

▶ **Question 12 *** *En appliquant la même méthode que dans la section précédente, trouvez des solutions au problème des huit dames. Combien de solutions y a-t-il ?*
■

▶ **HX4 & HX1 – Option Informatique**
Troisième TP Caml
Vendredi 4 février 2011

Programmation impérative et backtrack

Un corrigé

▶ Question 1

```
let ligneX n =
  for i = 1 to n do
    print_char 'X'
  done
```

▶ Question 2

```
let triangleX n =
  for i = 1 to n do
    for j = 1 to i do
      print_char 'X'
    done;
    print_char '\n'
  done
```

▶ Question 3 42

▶ Question 4 $a.(0)$ vaut 5 initialement donc b.(a.(0)) vaut initialement $b.(5) = 3$. Donc après modification $a.(0) = 3$ et $b.(3) = 6$. vect_length $b = 7$ donc le programme renvoie 13.

▶ Question 5

```
let retourne ch =
  let n = string_length ch in
  for i = 0 to (n-1)/2 do
    let temp = ch.[i] in
    ch.[i] <- ch.[n-1-i] ;
    ch.[n-1-i] <- temp
  done ;
  ch
```

▶ Question 6

```
let maximum tb =
  let min = ref tb.(0) in
  for i = 1 to pred (vect_length tb) do
    if !min < tb.(i) then min:=tb.(i)
  done;
  !min
```

▶ Question 7

```
let ind_max tb =
  let min = 0 in
  for i = 1 to pred (vect_length tb) do
    if tb.(!min) < tb.(i) then min:=i
  done;
  !min
```

▶ Question 8

```
let tri tb =
  let modif = ref true in
  while !modif do
    modif := false;
    for i = 0 to vect_length tb - 2 do
      if tb.(i)>tb.(i+1) then
        let temp = tb.(i) in
        tb.(i) <- tb.(i+1) ;
        tb.(i+1) <- temp ;
        modif := true
    done
  done
```

▶ Question 9 Parce que chaque ligne du tableau pointerait vers la même colonne.

▶ Question 10

```
let peut_poser g x y c =
  let peut = ref true in
  for i = 0 to 8 do
    if g.(x).(y) = c then peut := false
  done;
  for nx = (x/3)*3 to (x/3)*3+2 do
    for ny = (y/3)*3 to (y/3)*3+2 do
      if g.(x).(y) = c then peut := false
    done
  done;
  !peut
```

▶ Question 11

```
let resoudre g =
  let peut_poser x y c =
    let peut = ref true in
    for i = 0 to 8 do
      if g.(x).(i) = c || g.(i).(y) =c then peut := false
    done;
    for nx = (x/3)*3 to (x/3)*3+2 do
      for ny = (y/3)*3 to (y/3)*3+2 do
        if g.(nx).(ny) = c then peut := false
    done
    done;
    !peut
  in

  let rec foo (x,y) =
    if y > 8
    then foo (x+1,0)
    else if (x,y) = (9,0)
    then
      begin
        for x = 0 to 8 do
          for y = 0 to 8 do
            print_int g.(x).(y)
          done ;
          print_newline () ;
        done;
        failwith "trouve"
      end
    else
      if g.(x).(y) = 0
      then
        for c = 1 to 9 do
          if peut_poser x y c
          then (g.(x).(y) <- c ; foo (x,y+1) ; g.(x).(y) <- 0)
        done
      else
        foo (x,y+1)
  in
  foo (0,0)
```

▶ Question 12

```
let huit_dame ()=
  let g = make_matrix 24 24 0 in
  let aff =
    let s = make_string 72 ' ' in
    for i = 0 to 7 do
      s.[i*9+8] <- '\n'
    done;
    s
  in

  let affiche = let r = ref 0 in fun () -> incr r;print_int !r ;print_newline();print_string aff; print_newline() in

  let pose x y v =
    for i = -7 to 7 do
      g.(x).(y+i) <- g.(x).(y+i)+ v ;
      g.(x+i).(y) <-g.(x+i).(y)+ v ;
      g.(x+i).(y+i) <-g.(x+i).(y+i)+v ;
      g.(x+i).(y-i) <- g.(x+i).(y-i)+ v
    done ;
    if v > 0
    then aff.[(x-8)*9+y-8]<- '*'
    else aff.[(x-8)*9+y-8]<- ' '
  in
  let rec foo y =
    if y > 15 then affiche ()
    else
      for x = 8 to 15 do
        if g.(x).(y) = 0
        then ( pose x y 1 ; foo (y+1) ; pose x y (-1))
      done
  in
  foo 8
```