InitProg par l'exemple

Martin Monperrus

16 décembre 2011

Ce document illustre avec des programmes Ocamlcartes et Ocaml le polycopié du cours *Initiation à la programmation (InitProg)* de l'Université Lille 1. Si vous êtes capables de comprendre et d'écrire par vous-mêmes des programmes de ce calibre, vous avez atteint l'objectif du cours.

Table des matières

1 init_tas et deplacer_sommet

```
(* {{init_tas et deplacer_sommet}}
Utilisation:
- init_tas
- deplacer_sommet

Ce programme initialise le tas 1 avec une alternance de Pique et de Coeur
sur le tas 1 et rien sur les autres tas, puis trie toutes les cartes du tas 1 par couleur
sur les tas 2 (qui reçoit les Piques) et 3 (qui reçoit les Coeurs).
*)

(* Rappel: ceci est un commentaire, que ocaml et ocamlcartes ignorent. *)
(* Les commentaires vous aident à structurer et comprendre votre propre programme,
et, pour les professeurs, à corriger vos travaux. *)

(* Nous utilisons le module Cartes *)
open Cartes;;

(* Initialisation du tas 1 avec 4 cartes. Il y a une alternance de Pique et Coeur.
La fonction init_tas prend deux paramètres:
- le premier est un numéro de tas entre 1 et 4
- le deuxième est une chaîne de caractères décrivant le tas

C -> Coeur
K -> Carreau
T -> Trèfle
P -> Pique
La carte du dessous correspond au premier caractère:
init_tas(1,"CPT");;
          |_La carte du dessous est un coeur
Les numéros de carte sont tirés au hasard.
*)
init_tas(1,"CPCP");;

(* Les autres tas sont vides. Un tas vide est représenté par une chaîne de caractères vide notée en Ocaml "" *)
init_tas(2,"");;
init_tas(3,"");;
init_tas(4,"");;

(* Nous savons que la carte du dessus est un Pique, donc va sur le tas 2 *)
deplacer_sommet(1,2);;

(* La suivante est un coeur *)
deplacer_sommet(1,3);;

(* La suivante est un pique *)
deplacer_sommet(1,2);;

(* La dernière est un coeur *)
deplacer_sommet(1,3);;

(* Toutes les cartes sont triées, le programme est terminé! *)
pause("fini");
```

2 Le if/then et if/then/else

```
(* {{Le if/then et if/then/else}}
Utilisation:
 - le + et () dans init_tas
 - if/then/else

Ce programme initialise les tas 1 avec des valeurs aléatoires.  Les autres tas sont vides.
Ensuite, dépendamment de l'initialisation du tas 1, les coeurs partent
sur le tas 2, les trèfles sur le tas 3 et les piques sur le tas 4 *)

open Cartes;;

(* Initialisation du tas 1:
- le + veut dire "ou": (C+P) donne un coeur ou un pique aléatoirement
- les parenthèses permettent de délimiter des groupes
Ici, les tas possibles sont:
T
CP
PP
*)
init_tas(1,"((C+P)P)+T");;

(* Les autres tas sont vides. Un tas vide est représenté par
une chaîne de caractères vide notée en Ocaml "" *)
init_tas(2,"");;
init_tas(3,"");;
init_tas(4,"");;

(* Déplacements des cartes *)

if sommet_trefle(1)
then (* un bloc then prend une seule instruction ou un bloc begin-end *)
begin
  deplacer_sommet(1,3);
end (* jamais de ; avant le else *)
else
begin (* un bloc else prend une seule instruction ou un bloc begin-end *)

  (* nous déplacons le pique du sommet *)
  deplacer_sommet(1,4);

  (* sous le pique, avons-nous un coeur ou un autre pique ? *)
  if sommet_coeur(1)
  then
    deplacer_sommet(1,2) (* jamais de ; avant le else *)
  else
    (* c'est nécessairement un pique (d'après le init_tas) *)
    deplacer_sommet(1,4);

end;; (* end bloc else, premier niveau donc deux point-virgules *)

pause("fini");;
```

3 Algèbre booléenne

```
(* {{Algèbre booléenne}}
Utilisation:
 - la fonction tas_vide, tas_non_vide
 - l'algèbre booléenne not && ||

Ce programme initialise les tas 1 avec des valeurs aléatoires.
Les autres tas sont vides.

Ensuite, dépendamment de l'initialisation du tas 1, les rouges partent
sur le tas 2, les noires sur le tas  *)

open Cartes;;

(* Initialisation du tas 1
Le tas peut être vide à cause du () *)
init_tas(1,"((P+T)+(C+K))+()");;

(* Les autres tas sont vides. Un tas vide est représenté par
une chaîne de caractères vide notée en Ocaml "" *)
init_tas(2,"");;
init_tas(3,"");;
init_tas(4,"");;


(* Si le tas 1 n'est pas vide *)
(* NB: faire un sommet_coeur, sommet_carreau, sommet_pique
ou sommet_trefle sur un tas vide casse l'automate *)
if not(tas_vide(1))
then
begin

  (* si carte est rouge *)
  if sommet_coeur(1) || sommet_carreau(1)
  then
    deplacer_sommet(1,2);

  (* si la carte est noire *)
  (* nous avons peut-être déplacé une carte, donc on rajoute un test *)
  (* notez que tas_vide(x) = not(tas_non_vide(x)) *)
  if tas_non_vide(1) && (sommet_pique(1) || sommet_trefle(1))
  then
    deplacer_sommet(1,3);
end;;

pause("fini");;
```

4 La boucle while

```
(* {{La boucle while}}
Utilisation:
 - le [] dans init_tas
 - la boucle while

Ce programme initialise les tas 1 et 2 avec n coeurs ou trèfles aléatoirement.
Puis déplace tous les coeurs du tas 1 sur le tas 4 et tous les coeurs du sommet du tas 2 sur le tas 4.
Le tas 3 est utilisé comme entrepôt *)

open Cartes;;

(* les crochets veulent dire n au hasard dans [0,52]
NB: l'expression entre crochet est exécutée n fois, donc les tas 1 et 2
contiennent une alternance aléatoire de coeur et trèfle *)
init_tas(1,"[C+T]");;
init_tas(2,"[C+T]");;

(* Les autres tas sont vides. *)
init_tas(3,"");;
init_tas(4,"");;


(* déplace tous les coeurs du tas 1 sur le tas 4 *)
while tas_non_vide(1)
do
  if sommet_coeur(1)
  then
    deplacer_sommet(1,4)
  else
    deplacer_sommet(1,3); (* dans l'entrepôt *)
done;;


(* tous les coeurs du sommet du tas 2 sur le tas 4. *)
(* Le tas 2 peut être vide si [] donne n=0 ou si il n'y a que des coeurs sur le tas 2
et qu'ils ont tous été déplacés *)
while tas_non_vide(2) && sommet_coeur(2)
do
  deplacer_sommet(2,4);
done;;

pause("fini");;
```

5 Définition d'une procédure

```
(* {{Définition d'une procédure}}
Utilisation:
 - définition de procédures

Ce programme déplace toutes les cartes des tas 1, 2 et 3 sur le tas 4 *)

open Cartes;;

(** Initialisation des tas: tous les tas possibles *)
init_tas(1,"[P+C+K+T]");;
init_tas(2,"[P+C+K+T]");;
init_tas(3,"[P+C+K+T]");;
(* Le tas 4 est vide *)
init_tas(4,"");;

(* Déplace toutes les cartes du tas x vers le tas y.
Types: x et y sont des numéros de tas
Contrainte d'utilisaton: x et y dans {1,2,3,4} *)
let deplacer_tas(x,y) =
begin
  while tas_non_vide(x)
  do
    deplacer_sommet(x,y);
  done;
end;;

(* déplacement des tas en utilisant la procédure deplacer_tas *)
deplacer_tas(1,4);;
deplacer_tas(2,4);;
deplacer_tas(3,4);;

pause("fini");;
```

6 Appels de procédure

```
(* {{Appels de procédure}}
Ce programme trie toutes les cartes de tous les tas. Les rouges finissent sur le tas 3 et les noires sur le tas 4.

trie_tous_tas appelle trie_tas_par_couleur et trie_tas_par_couleur appelle bouge_noir_sur et bouge_rouge_sur
Comme au légo, on commence par les plus petites briques:
l'ordre de définition des procédures commence par les "petites" procédures qui sont appelées par d'autres .
*)
open Cartes;;

(* déplace la carte du sommet du tas x sur le tas y
si et seulement si c'est une carte rouge *)
let bouge_rouge_sur(x,y) =
begin
  if tas_non_vide(x)
  then
  begin
    if sommet_coeur(x) || sommet_carreau(x) then deplacer_sommet(x,y);
  end
end;;

(* déplace la carte du sommet du tas x sur le tas y
si et seulement si c'est une carte noire *)
let bouge_noir_sur(x,y) =
begin
  if tas_non_vide(x)
  then
  begin
    if sommet_pique(x) || sommet_trefle(x) then deplacer_sommet(x,y);
  end
end;;

(* trie les cartes du tas x en mettant toutes les cartes rouges sur y et les cartes noires sur z *)
let trie_tas_par_couleur(x,y,z) =
begin
  while tas_non_vide(x)
  do
    bouge_rouge_sur(x,y);
    bouge_noir_sur(x,z);
  done
end;;

(* trie toutes les cartes de tous les tas en mettant toutes les cartes rouges sur y et les cartes noires sur z *)
let trie_tous_tas(y,z) =
begin
  (* on libère les tas de destination *)
  trie_tas_par_couleur(y,1,2);
  trie_tas_par_couleur(z,1,2);
  (* et on finit le tri *)
  trie_tas_par_couleur(1,y,z);
  trie_tas_par_couleur(2,y,z);
end;;

trie_tous_tas(3,4);;
pause("fini");;
```

7 Arithmétique et variables

```
(* {{Arithmétique et variables}}

Ce programme teste si 999763 est un nombre premier.
*)

(* y est une variable constante de type entier (int) *)
let x = 999763;;

(* y est une variable mutable de type entier (int) *)
let y = ref 2;;

(* est_entier est une variable mutable de type booléen (bool) *)
let est_entier = ref true;;

while !y < x (* y est mutable, donc on accède à sa valeur avec ! *)
do
  (* !!Attention!! la division de deux entiers est euclidienne *)
  if x = ( x / !y) * !y
  then
    est_entier := false;

  (* on incrémente y *)
  y := !y + 1;
done;;

(* affichage du résultat *)
print_int(x);
if !est_entier = true
then
begin
  print_string(" est premier");
  print_newline(); (* retour à la ligne *)
end
else
  (* print_endline est équivalent à print_string suivi de print_newline *)
  print_endline(" n'est pas premier");;
```

8 Définition d'une fonction

```
(* {{Définition d'une fonction}}
Utilisation:
 - définition d'une fonction qui renvoie une valeur
 - variable locale
 - print_string, print_int, print_newline

Ce programme calcule la somme des carrées des entiers naturels de 1 à 10

*)

(* retourne le carré de n (de type entier) *)
(* NB: cette fonction est consituté d'une seule instruction, la valeur renvoyée par la fonction
est la valeur de cette instruction *)
let carre(n) = n*n;;

(* retourne la somme des carrées des entiers naturels de 1 à *)
(* NB: cette fonction est consituté de plusieurs instructions, la valeur renvoyée par la fonction
est la valeur de la dernière instruction (ici resultat) *)
let somme_carre(n) =
  let resultat = ref 0 and i = ref 1 in (* ce sont des variables locales *)
  begin
    while !i <= n
    do
      resultat := !resultat + carre(!i);
      i := !i +1;
    done;
    !resultat;
  end;;

(* Nous appelons notre fonction *)
print_string("somme_carre(2)=");;
print_int(somme_carre(2));;
print_newline();;


print_string("somme_carre(5)=");;
print_int(somme_carre(5));;
print_newline();;

print_string("somme_carre(500)=");;
print_int(somme_carre(500));;
print_newline();;
```

9 La boucle for

```
(** {{La boucle for}}
Utilisation:
- boucle for/pour (to et downto)
- variable locale (resultat)

Ce programme calcule la factorielle d'un entier *)

(* retourne la factorielle de l'entier n
Contrainte d'utilisation: n>=1 *)
let factorielle(n) =
  let resultat = ref 1 in
  begin
    for i = 1 to n
    do
      resultat := !resultat * i;
    done;
    !resultat;
  end;;


(* affiche toutes les valeurs de k! pour k in [ 1 , 20 ] *)
(* NB: l'utilisation du downto *)
for k = 10 downto 1
do
  print_int(k);
  print_string("! = ");
  print_int(factorielle(k));
  print_string("\n");
done;;
```

10 Calcul de π avec la suite de Madhava

```
(* {{Calcul de $\pi$ avec la suite de Madhava}}
Ce programme calcule un approximation de Pi selon une suite de Madaha *)


(* Retourne la valeur de a puissance n avec a entier et n entier *)
let puissance(a,n) =
  let resultat = ref 1 in
  begin
  for i = 1 to n
  do
    resultat := !resultat*a;
  done;
  !resultat;
  end;;

(* Renvoie une approximation de pi en utilisant la suite de madhava.
plus n est grand, meilleure est l'approximation de pi.
n est nombre entier *)
let calcul_pi_madhava(n) =
  let resultat = ref 0. and terme = ref 0. in
  begin
  for i = 0 to n
  do
    terme := (float_of_int(puissance(-1,i))/.float_of_int(2*i+1));
    resultat := !resultat +. !terme;
  done;
  4. *. !resultat;
  end;;

(* affiche l'evolution des valeurs de la suite *)
for k = 0 to 20
do
  let n = k * 500 in
  begin
    print_string("calcul_pi_madhava(");
    print_int(n);
    print_string(")=");
    print_float(calcul_pi_madhava(n));
    print_string("\n");
  end
done
```

11 Manipulation des chaînes de caractères

```
(* {{Manipulation des chaînes de caractères}}
Utilisation:
- String.length, String.create
- affectation d'un caractère s[i] <- 'a'

Ce programme transforme un chaîne de caractère s en vertical *)



(* transforme un chaîne de caractère s en vertical *)
let transforme_en_vertical(s) =
  let longueur = String.length(s) in
  (* renvoie une nouvelle chaîne de taille 2*String.length(s) *)
  let resultat = String.create (2 * longueur) in
  begin
   for i = 0 to longueur-1
   do
     resultat.[2*i] <- s.[i] ;
     (* Entre chaque caractère de s, i.e. aux indices impaires,
     la fonction insère un retour à la ligne (en OCcaml s.[x] <- '\n';). *)
     resultat.[2*i+1] <- '\n' ;
   done;
   resultat;
  end;;


(* appels de la fonction *)
print_string(transforme_en_vertical("InitProg, je suis au point"));;
print_string(transforme_en_vertical("Ocaml? C'est super!"));;

(* que se passe-t-il ici ? *)
print_string(transforme_en_vertical(transforme_en_vertical("VERTIGE!")));;
```

12 Tri des cartes

```
(* {{Tri des cartes}}
Ce programme trie un tas aléatoire par ordre croissant *)
open Cartes;;

(* Initialisation des tas *)
init_tas(1,"[T+C+P+K]");;
init_tas(2,"");;
init_tas(3,"");;
init_tas(4,"");;

(* Déplace tout le tas x sur y et met la plus petite carte (une seule carte) sur z
Contrainte d'utilisation: y doit être vide
Note: si n est le nombre de cartes de x en entrée, le tas y contient n-1 cartes à la fin *)
let deplace_plus_petite_carte(x,y,z) =
begin
  (* on fait l'hypothèse que la première carte est la plus petite *)
  deplacer_sommet(x,z);
  while tas_non_vide(x)
  do
    (* hypothèse invalidée *)
    if superieur(z,x) then
    begin
      deplacer_sommet(z,y);
      deplacer_sommet(x,z);
    end
    else (* z est plus petit que x, on bouge sur y *)
      deplacer_sommet(x,y);
  done;
end;;

(* Déplace un tas x sur un tas y *)
let deplacer_tas(x,y) =
begin
  while tas_non_vide(x)
  do
    deplacer_sommet(x,y);
  done;
end;;

(* Trie le tas x par ordre croissant. Met le tas trié sur le tas y en se servant du tas tmp comme tas intermédiaire
Contrainte d'utilisation: tmp est vide *)
let trier_tas_croissant(x,y,tmp) =
begin
  while tas_non_vide(x)
  do
    deplace_plus_petite_carte(x,tmp,y);
    deplacer_tas(tmp,x);
  done;
end;;

(* Trie le tas 1, le tas trié se trouve en 4 à la fin. Le tas 2 est le tas intermédiaire *)
trier_tas_croissant(1,4,2);;

pause("fini");;
```