TP10 : Lisibilité et efficacité du code

Christophe Rose

Vendredi 13 juin 2008

christophe.rose@ens.fr

http ://www.eleves.ens.fr/home/rose/caml

Dans ce TP, nous allons voir à travers plusieurs exemples comment rendre lisibles, executables, corrects et rapides les programmes en Caml.

1 Lisibilité

L'indentation permet de mettre en valeur la structure du programme. Par exemple, le début et la fin d'une boucle doivent prendre chacun une ligne entière, sauf cas particulier.

Question 1. Réécrivez le code suivant :

```
let max_vv tab = let n=vect_length tab and
maximum=ref 0 in for i=0 to (n-1) do let
p=vect_length tab.(i) in for j=0 to (p-1) do if
tab.(i).(j) > !maximum then maximum:=tab.(i).(j);
done; done; !maximum;;
```

Les noms de variables et de fonctions ne doivent jamais tenir en une lettre, sauf dans des cas précis : `n` pour la longueur d'un tableau, `i` ou `j` pour des indices de boucle, `t::q` ou `h::t` pour la tête et la queue d'une liste (atttention à ne pas mélanger les deux), `m` et `M` pour une borne inférieure ou supérieure d'un ensemble.

En règle générale, il faut donner le plus d'information et de sens dans le nom d'une variable ou d'une fonction, sans dépasser les 10 lettres.

Question 2. Réécrivez le code suivant :

```
let f x = let a = vect_length x and b = ref true
in for c = 0 to (a-2) do if x.(c)>x.(c+1) then
b:= false else (); done; !b;;
```

Vous pouvez utiliser à volonté les fonctions comme `max`, `min`, `incr`, `decr`, `list_length` et `map`. Par contre, il est déconseillé d'utiliser des fonctions comme `vect_to_list` ou `list_to_vect` sauf si c'est explicitement autorisé.

Soyez clair et précis. N'écrivez pas de code inutile. Par exemple `if (prop=true) then true else false` se condense en `prop`.

Question 3. Réécrivez le code suivant :

```
let decale tab =
  let t::q= list_of_vect tab in
    vect_of_list(q@[t]);;
```

Conseils pour les concours

Soyez cohérents d'un bout à l'autre de l'épreuve : gardez le même style d'indentation, la langue (français ou anglais) et le style d'écriture des variables, ainsi que le parenthésage.

Il faut réutiliser les noms utilisés dans les énoncés, cela permet au correcteur de trouver ses repères. De même, une fonction écrite dans une question peut être réutilisée dans tout le reste de l'épreuve, même si on a sauté cette question.

Écrire avec un stylo ou avec un clavier sont deux choses différentes. Il est plus facile d'écrire `UneFonctionQuelconque` que `une_fonction_quelconque`. L'écriture script n'est pas obligatoire et les accents sont autorisés.

Le correcteur n'est pas un compilateur : une erreur de syntaxe peut passer inaperçue, mais les programmes illisibles sont mal vus des correcteurs. N'utilisez pas de fonctions anonymes comme arguments, sauf pour les fonctions très simples (comme `(fun x -> x+1)`).

Un programme et ses commentaires doivent tenir sur une page, au pire sur deux pages qui se font face. Les commentaires ne doivent être insérés à l'intérieur des programmes que s'ils sont assez courts, avec `(* ... *)`. Sinon il faut les écrire avant ou après, et éventuellement numéroter les lignes du code pour y faire référence.

Écrivez vos programmes une ligne sur deux, cela permet de les corriger plus facilement.

2 Erreurs et warnings

Lorsqu'on définit une fonction, Caml cherche les erreurs de programmation qui peuvent poser problème au moment de l'exécution. Les erreurs les plus courantes repérées par Caml sont les erreurs de syntaxes, ainsi que la non-correspondance des types.

Caml annonce un *warning* lorsqu'une fonction risque de renvoyer une erreur avec certains arguements. S'il est possible de prouver rigoureusement qu'une telle erreur n'intervient jamais, alors on peut ignorer le warning.

Question 4. Corrigez le code suivant :

```
let rec decompose n = match n with
      1 -> []
    | _ -> let i = 2 and continue = true in
      while continue do
        if n mod i = 0
        then continue:= false;
        else i:= i+1;
      done;
      [i] :: (decompose n/i);;
```

Il ne faut pas oublier d'écrire un `let rec` pour définir une fonction récursive. Parfois, une erreur se produit lorsqu'on a renommé une variable ou une fonction par erreur.

Question 5. Que se passe-t-il avec le code suivant ?

```
let f x = x+1;;
let f n = match n with
      0 -> 0
    | 1 -> 1
    | _ -> f(n-1)+f(n-2);;
```

3 Bogues

On appelle *bogue* une erreur de codage qui empêche un programme de fonctionner comme souhaité. Un bogue peut se déclarer uniquement sur certains arguments, d'où la nécéssité de tester les fonctions sur de nombreux exemples.

Question 6. Testez la fonction suivante sur quelques cas, puis corrigez-la :

```
let premier n =
  let i = ref 2 and est_premier = ref true in
    while (!i * !i)<n do
      if n mod !i = 0
      then (est_premier:=false)
      else incr i;
    done;
    !est_premier;;
```

Il est parfois difficile de faire la distinction entre un programme qui est entré en boucle infinie et un programme qui fait un calcul très long. Le meilleur moyen pour s'en apercevoir est de commencer par utiliser des petits arguments.

Question 7. Testez la fonction suivante sur des listes de longueur variant entre 0 et 100000. Améliorez-la de sorte que sa complexité soit en $O(n)$.

```
let rec melange l = match l with
      [] -> []
    | [t] -> [t]
    | t1::t2::q -> t1::(melange q)@[t2];;
```

Au cours de l'exécution d'un programme, il arrive qu'on fasse plusieurs fois le même calcul. Le fait de garder en mémoire les résultats des calculs permet d'améliorer la vitesse du programme.

Question 8. Améliorez la fonction suivante :

```
let rec arbres n = match n with
      0 -> 1
    | _ -> let reponse = ref 0 in
      for i=0 to (n-1) do
        reponse:= !reponse+
          (arbres i)*(arbres (n-1-i))
      done;
      !reponse;;
```

Pour trouver un bogue, on peut commencer par diviser la fonction en plusieurs parties, puis par chercher dans quelle partie se trouve l'erreur. On recommence jusqu'à trouver un *exemple minimal*, c'est-à-dire le plus petit exemple qui reproduit le bogue.

On apporte alors des modifications par petites étapes jusqu'à son élimination.

On peut également demander à la fonction d'imprimer (à l'aide de `print_int` pour des variables entières), les valeurs de certaines variables ou références au cours du temps. Après l'élimination du bogue, il ne faut pas oublier de retirer l'instruction d'impression.

Question 9. La fonction suivante calcule à partir d'un entier $n > 0$ le plus petit entier a tel que n divise une certaine puissance de a. Corrigez la fonction suivante et améliorez sa vitesse :

```
let radical n = let tab = make_vect (n+1) 1 in
  for i=2 to n do
    let diviseur=ref 2 in
      while(n mod !diviseur<>0) do
        incr diviseur;
      done;
    let quotient=(i / !diviseur) in
      if quotient mod !diviseur = 0
      then tab.(i)<- tab.(quotient)
      else tab.(i)<- !diviseur*(tab.(quotient))
  done;
  tab.(n);;
```