Programmation : TP 3

Objectifs : [C] Boucles bornées et non bornées. [O'Caml] Références, boucles bornées et non bornées, exceptions.

On rappelle...

... qu'à la fin du TP il faut envoyer vos fichiers à l'adresse "barnier@recherche.enac.fr". Vous pouvez, pour cela, utiliser le *mailer* disponible sur `https://webmail.ens-mi.enac.fr/`.

Petit rappel de cours

Dans le style impératif de programmation, les variables sont d'abord déclarées (ce qui permet de leur réserver un emplacement mémoire) et éventuellement initialisées avec une certaine valeur, puis elles sont modifiées par une suite de traitements (par exemple une boucle `for` ou `while` lorsque ces traitements sont itératifs). Affecter une nouvelle valeur à une variable revient à faire une écriture dans la case mémoire réservée à celle-ci.

En C, on procède comme suit :

```
int n=0;
n= n+1;
```

En O'Caml, lorsque vous écrivez `let x=2 in ...`, l'identificateur `x` n'est pas une variable modifiable, et le signe `=` qui sert à définir `x` (au sens mathématique), **n'est pas** l'opérateur d'*affectation*. Pour manipuler des variables modifiables en O'Caml vous aurez besoin d'utiliser des *références*, que vous créerez avec le constructeur `ref`. Une *référence* est une structure de données qui modélise une case mémoire de la machine, dans laquelle on peut écrire avec l'opérateur d'affectation ":=" et à laquelle on peut accéder en lecture avec l'opérateur de *déréférencement*, noté "`!`". Les deux lignes suivantes illustrent l'utilisation des références :

```
let n= ref 0 in
n:= !n+1
```

[C, Ocaml] Calcul de $n!$

1. Placez-vous dans votre répertoire `C`, lancez `xemacs &` et ouvrez un nouveau fichier (`^x ^f`) que vous appelerez `tp3.c`.
2. Ecrivez en C un programme calculant et affichant la factorielle d'un entier n (que vous déclarerez et initialiserez à une valeur au choix), en utilisant une boucle `for`.

   ```
   for (i=1;i<=n;i++)
     {
       ...
     }
   ```

3. Modifier le programme précédent pour calculer la factorielle d'un entier n quelconque passé en argument.

   Indication : *L'argument passé à la fonction* `main` *est* `argv[1]`, *que vous transformerez en entier avec la fonction* `atoi` *de la librairie* `stdlib`.

4. Faire la même chose en O'Caml, dans un fichier `tp3.ml`. Syntaxe de la boucle `for` en O'Caml :

   ```
   for i=1 to n do
     ...
   done;
   ```

   Conseil : utilisez le toplevel Caml pour tester votre fonction.

[C, Ocaml] Conjecture de Syracuse

Considérons la suite u_n définie par u_0 son premier terme,et par :

$$u_{n+1} = \begin{cases} \frac{u_n}{2} & \text{si } u_n \text{ pair} \\ 3u_n + 1 & \text{sinon.} \end{cases}$$

Au début des années 1930, Lothar Collatz, un mathématicien alors étudiant à l'université de Hambourg, émit la conjecture que cette suite finissait toujours par donner la valeur 1 pour devenir ensuite cyclique, quelle que soit la valeur du premier terme : $\forall u_0, \exists n_0$ tel que $u_{n_0} = 1$

Cette conjecture, présentée vers les années 50 par un collègue du L.Collatz à l'université de Syracuse, est également connue sous le nom de conjecture de Syracuse. Elle n'est toujours pas démontrée.

1. Vérifiez-la dans la pratique en écrivant un programme C qui calcule les termes de la suite u_n et les affiche, et qui s'arrête lorsque u_n vaut 1.

   Note : *Comme on ne connaît pas a priori la valeur de n_0, il n'est pas possible d'employer une boucle* `for` *comme précédemment. Utilisez une boucle* `while`.

   `while (`*condition*`) { ... }`

2. Même chose en `O'Caml`.

   `while (`*condition*`) do ... done`

[Ocaml] Lecture de données en entrée. Traitement des exceptions

1. Modifier le programme O'Caml de la question précédente de façon à demander à l'utilisateur d'entrer le terme initial de la suite. La fonction `input_line` permet de lire le flot de caractères sur un canal en entrée. Dans notre cas elle sera utilisée comme suit, `stdin` étant l'entrée standard (le clavier) :

   ```
   let u0= int_of_string (input_line stdin) in
   ```

2. Pour l'instant, votre programme ne demande qu'une seule valeur. Modifiez-le afin de redemander une nouvelle valeur à la fin de chaque calcul. Utilisez une boucle infinie. NB : les affichages étant *bufferisés*, c'est-à-dire réalisés à travers une mémoire temporaire dont le contenu est effectivement envoyé à l'écran de façon non systématique, il peut être nécessaire de *flusher* explicitement ce buffer. Pour une écriture sur la sortie standard (l'écran), ceci doit être réalisé avec l'instruction `flush stdout`.

3. Encadrez cette boucle par un traitement d'exception permettant de sortir de la boucle infinie lorsque l'utilisateur entre autre chose qu'un entier.

   Une exception permet d'interrompre la séquence d'instructions en cours lorsqu'elle ne se déroule pas normalement. Ici, la fonction `int_of_string` ne se terminera pas correctement si on entre une lettre au clavier : elle lèvera une exception `Failure("int_of_string")` pour le signaler. Cette exception devra être récupérée et traitée (bloc `try`...`with`) comme suit :

   ```
   try
     while true do  (* la boucle infinie *)
       ...     (* section de code contenant int_of_string (input_line stdin) *)
     done
   with _ ->  (* récupération de n'importe quelle exception *)
     ...      (* section de code à exécuter en cas d'exception *)
   ```