Auteurs : R. Namyst, C. Nichitiu, I. Todinca
Enseignants : L. Bougé, A.Bonhomme
Email : {Luc.Bougé,Alice.Bonhomme}@ens-lyon.fr

le 23/10/2000

TD Programmation 4 : Exceptions.

1 Utiliser les exceptions

1.1 pour manipuler des listes

1.1.1 Définition

- Définissez un type `ilist` avec ses constructeurs et extracteurs (`empty`, `cons`, `car`, `cdr`, `is_empty`).
- Ecrire une fonction `generate n` qui génère une liste de `n` éléments qui ont une valeur tirée aléatoirement entre `0` et `n`.

1.1.2 Recherche dans une liste

- Ecrire une fonction itérative de recherche d'un élément dans une liste (`searchList a l`).
- Réécrire cette fonction en utilisant comme boucle principale `while true` et en utilisant les exceptions pour terminer la fonction.

1.2 pour manipuler des arbres

1.2.1 Définition

- Définissez un type arbre avec ses constructeurs et extracteurs (`empty`, `make`, `get_item`, `get_left`, `get_right`, `is_empty`).
- Ecrire une fonction `generate n` qui génère un arbre de `n` éléments qui ont une valeur tirée aléatoirement entre `0` et `n`.

1.2.2 Recherche dans un arbre

- Ecrire une fonction récursive de recherche d'un élément dans un arbre (`searchTree a t`).

– Réécrire cette fonction en utilisant les exceptions.
– Dans cette deuxième fonction de recherche, rajouter l'affichage des éléments parcourus. Que peut-on en déduire ?
– Reprendre la deuxième fonction de recherche pour qu'elle renvoie le sous-arbre associé à l'élément trouvé, ou un arbre vide si l'élément n'est pas présent dans l'arbre.

1.3 Recherche avancée dans les listes

A partir de la définition du type `ilist` faite dans le paragraphe 1.1.1, définir une nouvelle fonction de recherche `searchList2 a b l` qui renvoie `true` si on peut trouver l'élément `a` dans la liste tel que l'élément `b` se situe avant `a` dans la liste. Cette fonction renvoie `false` dans les autres cas.

2 Calculatrice

Remarque. Pour vous aider dans la mise en œuvre, vous trouverez à l'URL

`http://www.ens-lyon.fr/~bouge/Teaching/Programmation/TD/tpcaml-4.ml`

un harnais général pour les entrées-sorties et la boucle d'évaluation (*toplevel*). Il se lance en évaluant la fonction `run()`. La fonction principale à modifier est `handle_char()`.

Pour simplifier, on ne traite *en entrée* que des nombres formés d'un seul chiffre.

2.1 Ordre préfixé, ordre infixé, ordre postfixé

Nous avons l'habitude d'écrire les expressions arithmétiques en ordre infixé. Il existe aussi deux autres types de représentation des expressions arithmétiques qui ont l'avantage de ne pas avoir besoin de parenthèses. Ce sont les ordres préfixé et postfixé. L'ordre préfixé précise les opérateurs avant les arguments alors que c'est l'inverse pour l'ordre postfixé. Par exemple `((2+3)*6)` s'écrit `* + 2 3 6` en préfixé et `2 3 + 6 *` en postfixé.

2.2 Pile

Implémenter une structure de données de type *pile* : les opérations principales seront `push` et `pop`, qui permettent de rajouter et respectivement d'enlever un élément en tête de la pile d'évaluation . Il faudra aussi des opérations pour initialiser la pile, tester si elle est vide, etc.

2.3 Réalisation d'un évaluateur d'expression d'ordre postfixé

Écrire un programme qui calcule la valeur d'une expression postfixée. Le programme est constitué d'une boucle qui lit une entrée[1] (chaîne de caractère) par itération et lui applique le traitement adéquate. Si c'est un opérateur `+`, `-`, `*`, `/`, `=`, il calcule à partir de la pile. Sinon,

[1] Pour la partie interface, se reporter au chapitre 16, "The core library", du manuel OCaml.

si c'est une valeur numérique, il la stocke dans une pile. L'opérateur `=` affiche la valeur du sommet de pile. Rajouter un symbole spécial ! qui a comme effet de vider la pile, et un symbole `.` qui termine le programme.

Exemple. On veut calculer : `2 3 + 6 *`. On rentre 2, qui est stocké dans la pile. Puis 3, qui va aussi dans la pile.`+` est un opérateur qui prend les deux éléments du dessus de la pile, calcule leur somme et la stocke dans la pile, i.e. 5. Puis, 6 est rentré et stocké dans la pile. `*` multiplie les 2 valeurs du dessus de la pile et stocke le résultat au sommet de la pile.

Essayer le calculateur avec les expressions suivantes :

A : `4 5 8 3 * + - =` **B** : `7 4 + 5 - 3 5 + 2 1 * - + =`
C : `1 2 3 4 5 6 7 8 1 1 + + - * * + - + / =`

2.4 Extensions

2.4.1 Corrections

Permettre à l'utilisateur d'avorter l'évaluation d'une ligne ou d'effacer le dernier caractère rentré (touches `AC` et `C` sur la plupart des calculatrices).

2.4.2 Erreurs

Des message d'erreur explicites doivent être affichés. Proposer un système de “récupération”, pour pouvoir reprendre le calcul même en cas d'erreur.

2.4.3 Terminaison

Demander confirmation à l'utilisateur avant de quitter le programme.

2.5 Compilation

Créer un fichier executable a partir de votre code OCaml, en utilisant la commande

ocamlc -o ⟨fichier_executable⟩ ⟨fichier_source.ml⟩

Pour des informations complémentaires sur la compilation, vous pouvez consulter le chapitre 7 “Batch compilation” du manuel en ligne.

3 Ordonnanceur de tâches (si vous avez du temps)

On se propose ici de simuler un petit ordonnanceur de tâches. Les tâches seront simplement des fonctions acceptant en argument un numéro d'itération (un appel à $f(n)$ signifiera que l'ordonnanceur donne la main à la tâche f pour la $n$ième fois). Lorsqu'une tâche désire rendre la main à l'ordonnanceur pour donner sa chance à une autre tâche, elle lève une exception. Cette exception indique si cette préemption est purement volontaire ou si la tâche désire se terminer. Lorsqu'un tel événement se produit, l'ordonnanceur doit donc élire une autre tâche et l'exécuter. Le travail de l'ordonnanceur s'arrête lorsqu'il n'y a plus aucune tâche à exécuter dans le système.

Voici un petit exemple de tâche pour illustrer notre propos :

```
let g x =
  if x < 10 then
    begin
      print_string("C'est la ");
      print_int(x);
      print_string(" ieme fois que j'ai la main");
      print_newline();
      raise Yield
    end
  else
    raise Terminate;;

let f = function
  0 ->
    print_string("Ceci est mon premier quantum de temps!");
    print_newline();
    raise Yield |
  _ ->
    print_string("Et celui-la le deuxieme...");
    print_newline();
    raise (ContinueWith(g));;
```

Écrivez une fonction (l'ordonnanceur de tâches) qui prenne une liste de fonctions en paramètre et qui leur attribue tour à tour le processeur comme indiqué précédemment.