Programmation fonctionnelle (Ocaml)
Utilisation de `Caml` sous Linux

Il existe, en première approche, 3 méthodes pour programmer en Ocaml sous Linux :

1. interpréteur en ligne de commande ;
2. fichier source inclus dans l'interpréteur ;
3. compilation puis exécution.

Dans le cadre des TP, je conseille d'utiliser la deuxième méthode.

Interpréteur en ligne de commande

Pour lancer l'interpréteur Ocaml, il suffit, dans un terminal, de taper `ocaml`. Comme vous pouvez rapidement le constater, l'interpréteur ne dispose pas de fonctionnalités d'édition (notamment, on ne peut pas utiliser les flèches pour modifier le contenu d'un ligne (on doit effacer) ou pour naviguer dans l'historique des lignes déjà tapées, ...) Aussi, pour une bonne ergonomie de l'interpréteur, je conseille de l'utiliser couplé à `ledit` [1] en utilisant la commande : `ledit ocaml`.

On gagnera à faire dès maintenant un alias dans ses fichiers de configurations...

Inclure un fichier

Au cours d'un TP, nous allons écrire des programmes assez longs. L'édition purement à l'intérieur de l'interpréteur Ocaml serait alors assez fastidieuse, en particulier dès qu'il s'agit de corriger des erreurs ou de sauvegarder son travail. Je conseille plutôt d'utiliser un éditeur de texte classique (`kwrite`, `vim`, `emacs`, ...) pour taper et sauvegarder l'ensemble des définitions de fonctions, puis d'importer tout le fichier dans l'interpréteur (ce qui permet d'avoir ensuite la main dans l'interpréteur pour effectuer facilement des tests).

Pour importer un fichier dans l'interpréteur, on utilise la commande :

```
#use "toto.ml"
```

Attention ! Le `#` fait ici partie intégrale de la commande ! Il ne s'agit pas du prompt de l'interpréteur !

En ce qui concerne le choix de l'éditeur, il n'y a pas de vraies contraintes. Comme toujours, il est agréable d'avoir un éditeur puissant incluant notamment une bonne coloration syntaxique conçue pour le langage. Dans ce but, je conseille le couple `emacs` (éditeur) et `tuareg-mode` (coloration syntaxique). Je ne sais pas si les autres éditeurs (`vim`, `kwrite`, ...) possèdent une bonne coloration syntaxique pour Ocaml.

Pour utiliser le `tuareg-mode`, il faut rajouter les lignes suivantes à la fin de votre fichier `~/.emacs` :

```
;; Tuareg-mode
(setq auto-mode-alist (cons '("\\.ml\\w?" . tuareg-mode) auto-mode-alist))
(autoload 'tuareg-mode "tuareg" "Major mode for editing Caml code" t)
(autoload 'camldebug "camldebug" "Run the Caml debugger" t)
(add-hook 'tuareg-mode-hook 'auto-fill-on)
```

Si vous n'avez pas encore de fichier `.emacs` (c'est-à-dire si vous n'avez encore jamais utilisé `emacs` ou modifié sa configuration), vous pouvez tout simplement recopier dans votre répertoire personnel le fichier `~jean-yves.moyen/.emacs`

1. qui est lui-même un programme écrit en Ocaml...

Compilation

Finalement, on peut aussi compiler puis exécuter le programme Ocaml comme on le ferait pour un programme C. La commande pour compiler est `ocamlc`. Elle s'utilise, globalement, de la même manière que `gcc` (voir le manuel pour plus de détails).

`ocamlc` produit non pas directement un exécutable mais un fichier *bytecode* (code objet) qui est une sorte d'assembleur intermédiaire et doit de nouveau être interprété à l'aide de la machine virtuelle `ocamlrun`. C'est le même mécanisme qui est utilisé par Java. En pratique, tout ceci est fait de manière transparente et le fichier produit par `ocamlc` se comporte comme un exécutable (si Ocaml et `ocamlrun` sont bien installés sur la machine...) De même que les fichiers "compilés" par Java se comportent comme des exécutables si la machine possède bien une machine virtuelle Java.

Si on souhaite générer un "vrai" exécutable, c'est-à-dire du vrai code assembleur qui aura l'avantage d'être utilisable sans machine virtuelle (mais l'inconvénient de dépendre de l'architecture de l'ordinateur, comme avec un programme C), on peut utiliser la commande `ocamlopt` qui s'utilise elle aussi comme `gcc` (ou `ocamlc`).

Dans l'absolu, la compilation n'est utilisée qu'un fois les tests effectués et le logiciel finalisé. En effet, pendant la phase de tests et de débuggage, lancer l'interpréteur et inclure les sources est plus commode puisqu'on dispose alors d'une ligne de commande pour tester facilement et rapidement les fonctions écrites. C'est donc cette méthode qui est à préférer.