TP numéro 1 : Modularité

Programmation avancée, ENSIIE

Semestre 2, 2013–14

Nous avons vu qu'il était possible de manipuler les arbres binaires à l'aide d'un petit nombre d'opérations de base. Le but de ce TP est d'implémenter ces techniques en OCaml et en C en fournissant différentes implémentations pour les arbres binaires sans changer les algorithmes de haut niveau qui les manipulent. Vous trouverez à l'url `http://www.ensiie.fr/~guillaume.burel/cours/IAP3/` une archive avec les fichiers nécessaires pour ce TP.

Exercice 1

On rappelle qu'OCaml peut fonctionner en mode non interactif de la même manière que `gcc`. On dispose de deux types de fichiers sources : les fichiers d'interfaces (`.mli`) et les fichiers d'implémentation (`.ml`). Ils sont respectivement compilés vers des fichiers `.cmi` qui contiennent les informations de l'interface et des fichiers `.cmo` qui contiennent l'implémentation en elle-même. Le fichier `ab.mli` vous est donné et spécifie les opérations de base sur les arbres binaires vus en TD. Nous allons utiliser les fonctions qu'il déclare pour écrire les fonctionnalités de haut niveau sur les arbres binaires dans un fichier `algos.ml`.

Question 1.1

Implémentez les fonctions `hauteur`, `nombre_de_feuilles`, `est_equilibre` vues en TD. Implémentez les afficheurs `print_prefix`, `print_infix` et `print_postfix` vus en TD.

Nous pouvons maintenant commencer par compiler l'interface `ab.cmi` grâce à la commande

```
ocamlc -c ab.mli
```

puis produire un fichier objet `algos.cmo`, en utilisant la commande

```
ocamlc -c algos.ml
```

Question 1.2

Le fichier `Makefile` qui vous est fourni permet de fabriquer un exécutable `algos_ml` à partir des fichiers `ab.cmo` et `algos.cmo`. Nous devons donc écrire l'implémentation `ab.ml` correspondant à l'interface `ab.mli` pour les arbres binaires. Pour des raisons de simplicité nous éviterons les types polymorphes et le type `ab` des arbres binaires sera donné par

```
type arbre_binaire =
  | Vide
  | Noeud of arbre_binaire * int * arbre_binaire
```

Implémenter le fichier `ab.ml` et compléter le fichier `algos.ml` avec des tests pour faire tourner l'exécutable. On rappelle que la fonction `print_string` permet d'afficher une chaîne de caractères et que la fonction `print_newline` imprime un saut de ligne.

Question 1.3 (Implémentation en C)

Comme pour OCaml nous allons coder des fichiers `ab.c` et `algos.c`, le fichier d'interface `ab.h` vous est donné. Il permet de décrire les fonctionnalités des arbres binaires indépendamment de leur implémentation que l'on voit comme un simple pointeur vers une structure de nom `donnee_arbre` (`arbre_binaire` peut donc être considéré comme un type abstrait). En s'inspirant de ce qui a été fait en TD, implémenter les différentes structures nécessaires dans le fichier `ab.c`.

Question 1.4

Dans le fichier `algos.c` implémenter les fonctions `hauteur`, `nombre_de_feuilles`, `est_equilibre` vues en TD. Implémenter les afficheurs `print_prefix`, `print_infix` et `print_postfix` vus en TD. Implémenter une fonction `main` qui fabrique un arbre binaire et qui l'imprime.

Exercice 2

Il est possible d'utiliser d'autres structures de données pour représenter des arbres binaires. Par exemple, on peut ne pas utiliser l'arbre vide et avoir des constructeurs différents pour les nœuds à deux fils, à un seul fils gauche, à un seul fils droite et les feuilles.

Question 2.1

Sauvegarder le fichier `ab.ml` en un fichier `ab_avec_vide.ml`, puis réécrire le fichier `ab.ml` en utilisant le type suivant :

```
type arbre_binaire =
   Noeud2 of arbre_binaire * int * arbre_binaire   (* noeud avec deux fils *)
 | Noeudg of arbre_binaire * int      (* noeud avec un seul fils, à gauche *)
 | Noeudd of int * arbre_binaire      (* noeud avec un seul fils, à droite *)
 | Feuille of int                     (* noeud sans fils *)
```

Relancer `make`. Remarquer que vous n'avez pas eu à modifier ni à recompiler `algos.ml`, l'interface de `Ab` n'ayant pas changée.

Question 2.2

Créer un nouveau fichier `ab_algebrique.c`. Y inclure l'interface `ab.h`. Encoder le type algébrique d'OCaml ci-dessus à l'aide de la première technique vue en cours :

- créer une énumération `enum cas_ab` qui distingue les quatre constucteurs ;
- créer une union `union union_ab` qui regroupe les quatre contenus possibles ; pour `Noeud2` (resp. `Noeudg` et `Noeudd`) on utilisera un pointeur vers une structure `struct donnee_noeud2` (resp. `struct donnee_noeudg` et `struct donnee_noeudd`) ;
- créer une structure `struct donnee_arbre` qui est le produit de l'énumération et de l'union ; attention, contrairement au cours, le type `arbre_binaire`, défini dans l'interface, est un pointeur vers cette structure, et non pas juste un alias ; ceci est nécessité par le fait de laisser ce type abstrait dans l'interface ;
- créer les structures `struct donnee_noeud2`, `struct donnee_noeudg` et `struct donnee_noeudd`.

Réimplémenter les fonctions demandées dans l'interface avec cette nouvelle structure.

Ajouter une règle dans le `Makefile` pour qu'un exécutable `algos_algebrique` soit créé à partir des fichiers objets `ab_algebrique.o` et `algos.o`. On ajoutera les dépendances nécessaires. Ici encore, pas besoin de modifier ou de recompiler `algos.c`.