Université Paris 6
Licence d'informatique

Module de Programmation
ANNÉE 2000-2001

Travaux Dirigés n° 8

Exercice 1 Soit la déclaration suivante, effectuée dans un environnement `Env` :

```
let f = function m -> function n ->
  let res = g (m * n) in
    (((h res) = res), res);;
```

On cherche quelles sont les hypothèses minimales à faire sur `Env` pour que cette déclaration soit acceptable par le système CAML, en donnant à `f` le type le plus général (autrement dit, le plus polymorphe) possible.

Exercice 2 Typer la déclaration :

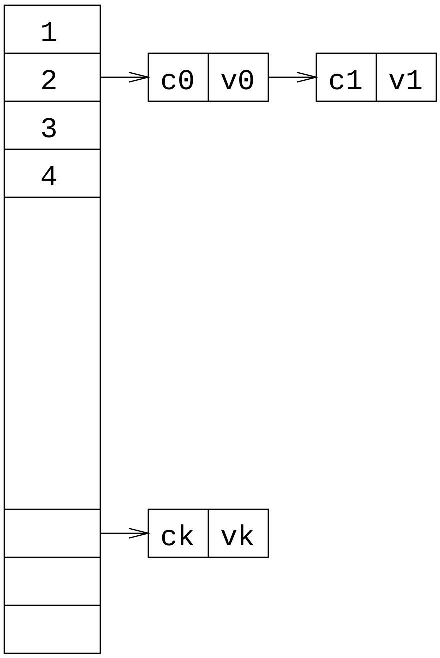
```
let f = function (x,z,y)  ->
  let r = ref (!z) in
    r := !r + 1;
    y := !y + 1 ;
    if x then z := !r else failwith " erreur";;
```

Exercice 3 *(Tables de hachage)* Dans le td précédent, vous avez vu comment représenter une table d'associations de couples *(clef, valeur)* par une liste. Une telle représentation est assez inefficace puisque l'ajout d'un couple a une complexité en $\mathcal{O}(1)$ mais la recherche d'une valeur associée à une clef a une complexité en $\mathcal{O}(n)$ dans le pire cas (si n est la taille de la liste).

Les tables de hachage sont des structures de données très efficaces permettant d'implanter une table d'associations entre des clefs et des valeurs. En pratique, ces tables permettent de retrouver la valeur associée à une clef en temps $\mathcal{O}(1)$.

Une table de hachage est la donnée :
- d'un tableau `T` de listes de couples *(clef, valeur associée)*, appelée liste de conflits.
- d'une fonction `hash` dite "fonction de hachage" permettant d'associer un indice entier à toute clef.

Graphiquement :



La recherche d'une valeur associée à une clef donnée `c` est faite ainsi :
- si `hash c` est une liste vide, aucune valeur n'est associée à la clef `c`
- sinon, on recherche dans la liste le premier couple `(c, v)` et on retourne `v`.

En pratique, l'ajout et la recherche d'un élément dans une table de hachage est très efficace à condition de disposer d'une "bonne" fonction de hachage, c'est-à-dire une fonction qui permette de minimiser les conflits.

Pour plus de détails on pourra se reporter au chapitre 12 du livre "Introduction à l'algorithmique" de Cormen, Leiserson et Rivest publié chez Dunod.

1 *(Définition du type)* Étant donnée une fonction de hachage `hash: 'a -> int`, proposez un type de données `('a, 'b) hash_table` permettant de représenter les tables de hachage de valeurs `'a * 'b`.

2 *(Création)* Écrire la fonction `creer` de type `int -> 'a hash_table` telle que `creer n` retourne une table de hachage de taille `n`.

3 *(Ajout)* Écrire une fonction `ajout` telle que `ajout table clef valeur` ajoute le couple `(clef, valeur)` dans la liste contenue à l'indice `table.(hash (clef))`. On décide ici que l'élement est ajouté en tête de la liste.

4 *(Recherche)* Écrire une fonction `rechercher` telle que `rechercher table clef` retourne la valeur associée à une `clef`. Si la clef apparaît plusieurs fois, vous retournerez la dernière valeur entrée. Cette fonction lèvera l'exception `Non_trouve` si aucune valeur n'est associée à une clef donnée.

5 *(Conversion)* Écrire une fonction `liste_de_table` permettant de convertir une table en une liste d'associations.

6 *(Itération)* Écrire une fonction `iterer_table` telle que `iterer_table f table` exécute la fonction `f` pour tous les éléments d'une table de hachage.

Exercice 4 *(Arbres d'évaluation (exo 3, feuille 7))* Reprendre l'exercice de la feuille de td 7 en implantant les listes d'associations par des tables de hachage.