Feuille de travaux pratiques n° 2
Types abstraits

Exercice 2.1

On souhaite manipuler des arbres binaires de caractères.

1. Définir le type nœud d'arbre binaire de caractères `ctree_t` et le type pointeur sur `ctree_t`, `ctreep_t` ;
2. Écrire la fonction `ctreep_t create_ctree(char v, ctreep_t left, ctreep_t right)` créant un arbre avec une racine contenant `v` et des sous-arbres gauche et droit `left` et `right` ;
3. Écrire la fonction `void delete_ctree(ctreep_t root)` détruisant l'arbre de racine `root` ;

Dans la suite, on considère le type « *pointeur sur fonction prenant en entrée un paramètre de type* `char` *et ne retournant rien* » `visit_node_f`.

4. Écrire la fonction `void prefix(ctreep_t root, visit_node_f f)` visitant l'arbre de racine `root` en ordre préfixe et appliquant la fonction `f` sur la valeur stockée dans chaque nœud ;
5. Écrire la fonction `void infix(ctreep_t root, visit_node_f f)` visitant l'arbre de racine `root` en ordre infixe et appliquant la fonction `f` sur la valeur stockée dans chaque nœud ;
6. Écrire la fonction `void postfix(ctreep_t root, visit_node_f f)` visitant l'arbre de racine `root` en ordre postfixe et appliquant la fonction `f` sur la valeur stockée dans chaque nœud ;
7. Écrire le programme principal créant l'arbre de la figure 1, l'affichant en ordre préfixe, infixe, puis postfixe et le détruisant.

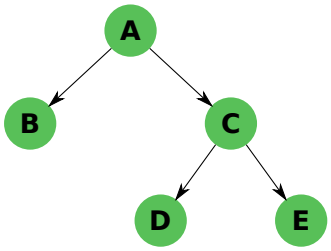

Figure 1 – Arbre de caractères pour l'exercice 2.1.

Exercice 2.2

L'objectif de cet exercice est d'implémenter le type « *ensemble de couleurs* ». Pour cela, on choisit d'utiliser une table de hachage avec fonction de hachage imparfaite (voir figure 2). Les questions ci-dessous vont vous permettre de créer dans l'ordre les différentes structures nécessaires. On s'attachera à séparer les différentes structures et leurs fonctionnalités associées dans des fichiers différents. On prendra aussi soin de masquer le plus possible les implémentations concrètes des types abstraits.

Le type `color_t`

1. Définir le type `color_t` représentant différentes couleurs (au moins une dizaine) ;
2. Écrire la fonction `const char *color2str(color_t c)` retournant la chaîne de caractères représentant la couleur `c` (exemple : `"BLEU"` pour la couleur bleue).

Le type `collist_t`

3. Définir le type « *liste doublement chaînée de couleurs* » `collist_t` et son pointeur associé `collistp_t` ;
4. Écrire la fonction ajoutant un nœud de liste de couleur `c` entre les nœuds pointés par `before` et `after`, `collistp_t collist_add(color_t c, collistp_t before, collistp_t after)` ;
5. Écrire la fonction `void collist_remove(collistp_t n)` détruisant le nœud `n` d'une liste chaînée en prenant soin de reconstituer les chaînages ;
6. Écrire la fonction `void collist_delete(collistp_t n)` détruisant toute la liste chaînée de tête `n` ;
7. Écrire la fonction `color_t collist_color_value(collistp_t n)` retournant la couleur contenue dans le nœud `n` ;
8. Écrire la fonction `collistp_t collist_next(collistp_t n)` retournant le nœud suivant `n` dans la liste ;
9. Écrire la fonction `collistp_t collist_prev(collistp_t n)` retournant le nœud précédent `n` dans la liste ;
10. Écrire le programme créant une liste chaînée de quatre couleurs, affichant les éléments de la liste à l'écran, retirant le troisième nœud de la liste, affichant la nouvelle liste en partant de la fin, puis la détruisant.

Le type `colhash_t`

11. Définir le type « *table de hachage de couleurs* » `colhash_t` et son pointeur associé `colhashp_t`. La table sera représentée par un tableau de pointeurs `collistp_t`, chaque élément du tableau pointant vers une liste doublement chaînée de couleurs, éventuellement vide ;
12. Écrire la fonction `colhashp_t colhash_create(size_t nbuckets)` créant une table de hachage avec `nbuckets` *buckets* (nombre de cases du tableau de pointeurs) ;
13. Écrire la fonction `void colhash_delete(colhashp_t h)` détruisant l'intégralité de la table de hachage `h` ;
14. Écrire la fonction `void colhash_add(colhashp_t h, color_t c)` ajoutant la couleur `c` dans la table `h` ;
15. Définir le type « *itérateur sur table de hachage* » `colhash_iterator_t`. Un itérateur est une structure permettant d'énumérer les éléments de la table de hachage en se souvenant de la dernière position vue. On définira aussi le pointeur associé `colhash_iteratorp_t` ;
16. Écrire la fonction `colhash_iteratorp_t colhash_begin(colhashp_t s)` retournant un itérateur sur le début de la table de hachage `s` ;
17. Écrire la fonction `bool colhash_end(colhash_iteratorp_t iter)` retournant `true` si l'itérateur `iter` est arrivé au bout de la table et `false` sinon. On détruira l'itérateur passé en paramètre si la fin de la table est atteinte ;
18. Écrire la fonction `colhash_iteratorp_t colhash_next(colhash_iteratorp_t iter)` retournant l'itérateur `iter` mis à jour pour pointer sur la position suivante dans la table ;
19. Écrire la fonction `color_t colhash_color_value(colhash_iteratorp_t iter)` retournant la couleur se trouvant à la position donnée par l'itérateur `iter` ;
20. Définir le type `colhash_iteratorp_or_color_t` permettant de représenter soit une couleur, soit un itérateur sur table de hachage ;
21. Écrire `bool colhash_remove(colhashp_t h, colhash_iteratorp_or_color_t v)`, la fonction retirant une couleur de la table ; la couleur peut être spécifiée directement par un itérateur pointant sur la case à retirer, ou indirectement en en donnant le nom. Dans ce dernier cas, on retirera la première occurrence rencontrée ;
22. Écrire `colhash_iteratorp_t colhash_contains(colhashp_t h, color_t c)`, la fonction retournant un itérateur sur la première occurrence de la couleur `c`, ou `nullptr` si elle n'est pas présente dans la table. L'itérateur sera créé dynamiquement dans la fonction et il reviendra à l'utilisateur de libérer la mémoire correspondante ;
23. Écrire le programme créant une table de hachage avec trois *buckets* et ajoutant six couleurs dedans. On affichera ensuite un message si la table contient la couleur `VERT`, puis l'on affichera l'ensemble des couleurs stockées dans la table ; on retirera ensuite une des couleurs stockées et l'on réaffichera toutes les couleurs de la nouvelle table.

Le type `colset_t`

24. Définir le type « *ensemble de couleurs* » `colset_t` et son pointeur associé `colsetp_t`. L'ensemble sera implémenté par une table de hachage pour pouvoir déterminer rapidement si un élément est déjà présent. On s'assurera de pouvoir définir une fonction `cardinal()` en $\mathcal{O}(1)$;
25. Écrire la fonction `colsetp_t colset_create(void)` créant un ensemble de couleurs vide ;
26. Écrire la fonction `void colset_delete(colsetp_t s)` détruisant un ensemble ;
27. Écrire la fonction `bool colset_add(colsetp_t s, color_t c)` ajoutant une couleur `c` à un ensemble `s`. La fonction retournera `true` si la couleur n'était pas déjà présente et `false` sinon ;
28. Écrire la fonction `bool colset_remove(colsetp_t s, color_t c)` retirant la couleur `c` de l'ensemble `s`. La fonction retournera `true` si la couleur était bien présente et `false` sinon ;
29. Écrire la fonction `bool colset_contains(colsetp_t s, color_t c)` retournant `true` si l'ensemble `s` contient la couleur `c` et `false` sinon ;
30. Écrire la fonction `size_t colset_cardinal(colsetp_t s)` retournant le nombre d'éléments de `s` ;
31. Définir le type « *itérateur sur ensemble de couleurs* » `colset_iterator_t` et son pointeur associé `colset_iteratorp_t` ;
32. Écrire la fonction `colset_iteratorp_t colset_begin(colsetp_t s)` retournant un itérateur sur le début de l'ensemble ;
33. Écrire la fonction `bool colset_end(colset_iteratorp_t iter)` retournant `true` si l'itérateur `iter` est à la fin de l'ensemble et `false` sinon. On détruira l'itérateur `iter` s'il est arrivé à la fin de l'ensemble ;
34. Écrire la fonction `colset_iteratorp_t colset_next(colset_iteratorp_t iter)` retournant l'itérateur `iter` mis à jour pour pointer sur l'élément suivant de l'ensemble ;
35. Écrire la fonction `color_t colset_color_value(colset_iteratorp_t iter)` retournant la couleur de l'élément pointé par l'itérateur `iter` ;
36. Écrire le programme créant un ensemble de couleurs vide, ajoutant cinq couleurs, dont l'une par deux fois, affichant l'ensemble des éléments de l'ensemble et son cardinal, puis affichant un message si l'ensemble contient la couleur `NOIR`, retirant l'une des couleurs présentes, puis réaffichant tous les éléments et enfin, détruisant l'ensemble.

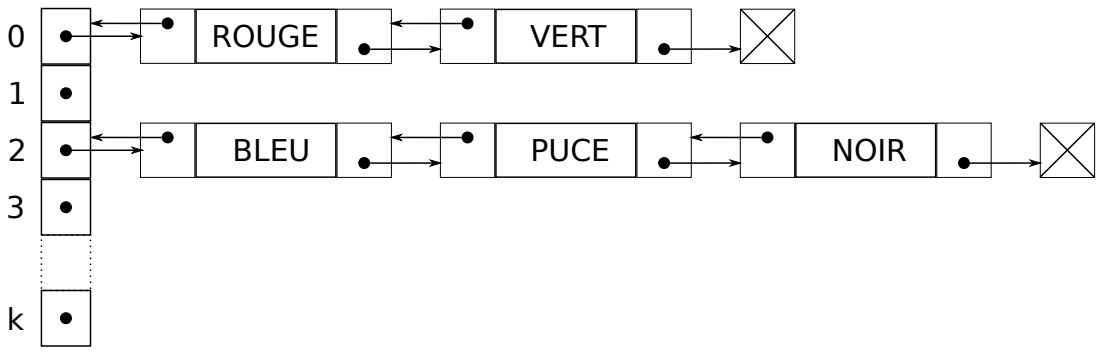

Figure 2 – Table de hachage de l'exercice 2.2