GIS2A3 — Structures de Données — Épreuve de Juin 2015
Durée 2 heures — Aucun document autorisé

---

1 Petits exercices (7 points)

1.1 Tables de hachage

On considère l'insertion des clés □, ◯, △, ♢ dans une table de hachage comportant $N = 7$ alvéoles.

Question 1 [2 pts]. Insérer les clés dans la table en supposant qu'elle est gérée avec la technique du double hachage. Les valeurs de hachage des différents éléments sont les suivantes :

	h_1	h_2
□	2	3
◯	5	2
△	5	1
♢	5	1

Question 2 [2 pts]. Insérer les alvéoles dans la table en supposant qu'elle est gérée avec la technique du simple hachage et que la valeur de hachage d'un élément est égale à son nombre de sommets.

1.2 Chaînes de caractères

On considère l'implantation suivante d'un type de chaînes de caractères. Le champ `zone` pointe vers un tableau de caractères alloué dynamiquement, contenant une suite de caractères terminée par `'\0'`. Le champ `alloc` contient le nombre de caractères alloués à `zone`. Le champ `length` contient la longueur de la chaîne et vérifie `length` $+ 1 <$ `alloc`.

```
struct chaine {
    char* zone;
    int alloc;
    int length;
};
```

Question 3 [3 pts]. Écrire une fonction `ajouter`, paramétrée par une chaîne s et un caractère c, qui ajoute c à la fin de s.

2 Problème (13 points)

On s'intéresse à une structure de données pour l'implantation des expressions (voir document préparatoire à l'examen, ci-joint). On souhaite en particulier implanter l'algorithme de la Figure 1. En particulier, on s'intéresse à la complexité $f(n)$ de cet algorithme, où $f(n)$ compte les comparaisons de variables nécessaires pour regrouper les termes dépendant d'une même variable.

$E :=$ l'expression vide (valant zéro)
for chaque terme $T_1, T_2, \ldots, T_n$ do
  $E :=$ l'expression obtenue en ajoutant T_i à E
end do

FIGURE 1 – Construction d'une expression $E = T_1 + T_2 + \cdots + T_n$

2.1 Représentation par listes

Une première solution consiste à représenter une expression par une liste de termes, en adaptant les listes étudiées en cours.

Question 4 [2 pts]. On souhaite implanter en C des listes de termes. Le fichier d'entête doit être distinct de `terme.h`. Indiquer les inclusions de fichiers d'entête par un diagramme. Donner la déclaration C des listes de termes.

Question 5 [2 pts]. Supposons les expressions représentées par des listes de termes non ordonnées. Quel serait un pire des cas pour l'algorithme de la Figure 1 ? Quelle est la complexité de ce pire des cas ?

2.2 Représentation par ABR — Étude a priori

Une autre solution consiste représenter une expression par un arbre binaire de recherche, dont les valeurs seraient des termes, pour accélérer la recherche d'un terme à partir de sa variable.

Question 6 [1 pt]. Donner la déclaration C d'un tel ABR.

Question 7 [1 pt]. On a mentionné en cours (mais on n'a pas étudié) une variante d'ABR équilibrée en hauteur, même dans le pire des cas. Comment s'appelle cette variante ?

Question 8 [1 pt]. Quel est l'ordre de grandeur de la hauteur d'un ABR équilibré en hauteur, comportant k valeurs ?

Question 9 [1 pt]. Supposons les expressions représentées par des ABR équilibrés en hauteur. Quelle est alors la complexité de l'algorithme de la Figure 1 ?

Question 10 [1 pt]. Au lieu d'un ABR, on aurait pu envisager d'implanter une expression par une table de hachage. L'une des deux implantations semble-t-elle meilleure que l'autre (penser à une variante importante de l'algorithme de la Figure 1, où les termes T_i sont obtenus en énumérant les termes d'une ou de plusieurs autres expressions) ?

2.3 Représentation par ABR — Algorithmique

Toutes les expressions sont supposées représentées sous la forme d'un ABR.

Question 11 [2 pts]. Écrire une fonction `est_nulle`, paramétrée par une expression E, qui retourne `true` si tous les termes présents dans E ont un coefficient nul.

Question 12 [2 pts]. Écrire une fonction `imprimer_expression`, paramétrée par une expression E qui affiche cette expression. Utiliser obligatoirement la fonction `imprimer_terme` et faire en sorte que le premier terme imprimé fasse l'objet d'un appel avec `first` = `true` et tous les autres termes fassent l'objet d'un appel avec `first` = `false`. Il est conseillé d'écrire une fonction auxiliaire.

Document préparatoire à l'épreuve écrite

On s'intéresse à des combinaisons linéaires de variables, à coefficients entiers. Le produit d'un coefficient par une variable est appelé un *terme*. Voici deux exemples d'expressions, chacune composée de trois termes :

$$\begin{aligned} E_1 &= 3\,\text{pomme} + \text{poire} + \text{peche}\,, \\ E_2 &= 4\,\text{poire} - 3\,\text{pomme} - 7\,\text{carotte}\,. \end{aligned}$$

Deux expressions peuvent bien sûr s'additionner. Par exemple,

$$E_1 + E_2 = -7\,\text{carotte} + 5\,\text{poire} + \text{peche}\,.$$

L'addition étant commutative, l'ordre dans lequel les termes apparaissent à l'affichage n'a pas d'importance. Les termes sont représentés par une structure de données décrite dans le fichier suivant, disponible (entre autres choses) sur le site du cours.

```
/* terme.h */

#define N 20

#include <stdbool.h>

struct terme
{
  int coeff;        /* le coefficient (peut être nul) */
  char variable[N]; /* le nom de la variable */
};

/*
 * Affecte (c, v) à *T
 */

extern void set_terme_coeff_variable (struct terme * T, int c, char *v);

/*
 * Retourne true si le coefficient de T est nul
 */

extern bool est_nul_terme (struct terme *T);

/*
 * Affiche le terme T. Le booléen first indique si T est le premier
 * terme d'une expression. Si c'est le cas, T est affiché normalement.
 * Sinon, le signe '+' ou '-' du coefficient est affiché devant T.
 */

extern void imprimer_terme (struct terme *T, bool first);
```