Examen d'Algorithmique - (2010/2011 session 1)

Durée : 1heure 45 minutes - Aucuns documents autorisés

Formation Ingénieurs CNAM

Ce sujet comporte 3 pages et 2 parties indépendantes. On s'attachera à soigner la présentation du code afin qu'il soit le plus lisible possible. Il ne suffit en aucun cas d'écrire le code d'une fonction, il faut expliquer (i.e. commenter) les choix faits pour l'implantation. Il est de plus indispensable de respecter les notations données dans l'énoncé.

1 Echauffement

On rappelle la structure de données définie pour représenter les arbres binaires d'entiers ainsi que les opérations de base `vide` et `e`. Afin de pouvoir tester notre implantation, on programme également une fonction d'affichage `print`.

```
#include<stdio.h>

typedef struct arbre
{
  int x;
  struct arbre *g, *d;
} Sarbre, *Arbre;

Sarbre *vide()
{ return (NULL); }

Sarbre *e(Sarbre *ag, int v, Sarbre *ad) /* enracinement */
{
  Sarbre *m=(Sarbre *)malloc(sizeof(Sarbre));
  m->g=ag;
  m->x=v;
  m->d=ad;
  return m;
}

void print(Sarbre *a) /* affichage infixe */
{
  if (a!=NULL)
    {
      print(a->g);
      printf("%d ",a->x);
      print(a->d);
   }
```

```
  printf("\n");
}
```

Question 1 Programmer la fonction `fois2` (qui multiple par 2 chaque élement de l'arbre). On procèdera de manière récursive avec des effets de bord (le type de retour de la fonction demandée est `void`).

```
void fois2 (Sarbre *a)
{ ... }
```

Question 2 Programmer de manière récursive une fonction `somme` qui fait la somme de tous les éléments d'une arbre.

```
int somme(Sarbre *a)
{ ... }
```

Question 3 Ecrire un programme de test dans la fonction `main` qui produit la sortie suivante :

```
int main()
{ ... }
```

```
user@computer$ ~/test-arbre
3 6 8
somme = 17
4 7 9
somme (*2)= 34
user@computer$
```

2 Problème : représentation des multi-ensembles

On considère le codage ASCII des caractères usuels. Ceux-ci sont représentés par un code (un entier) compris entre 0 et 255. Par exemple, le caractère 'e' est représenté par le code 101 et le caractère '4' par le code 52.

Dans ce problème, on s'intéresse au codage des multi-ensembles construits à partir de l'ensemble $\mathcal{A}$ des codes ASCII des caractères. Un multi-ensemble peut être vu comme un ensemble d'éléments de $\mathcal{A}$ où un élément peut apparaître plusieurs fois. Par exemple, {a,b,d,a} est un multi-ensemble contenant deux occurences de a, une seule de b et une seule de d.

On souhaite utiliser un tableau de taille 256, contenant des valeurs entières pour représenter le nombre d'occurences de chaque caractère de $\mathcal{A}$ dans un multi-ensemble donné.

Question 4 Comment représenteriez-vous le multi-ensemble vide avec cette structure de données ? Le multi-ensemble {a,c,a,d,f,f,f} ? Le multi-ensemble contenant tous les éléments de $\mathcal{A}$ deux fois ?

On considère maintenant la structure de données suivante pour représenter les multi-ensembles. On présente également les entêtes des fonctions à programmer.

```
#define SIZE 256

struct multiset
```

```
{
  int t[SIZE];
} *Smultiset, Multiset;

  multiset *me_vide();
  void add(char c, Smultiset *m);          // ajoute une occurrence de c
  void remove(char c, Smultiset *m);       // supprime une occurrence de c
  int nb_occ(char c, Smultiset *m);        // retourne le nombre d'occurences de c
  bool appartient(char c, Smultiset *m);   // teste si c appartient au multi-ensemble
  void print(Smultiset *m);                // affiche le multi-ensemble
```

Question 5 Expliquer les raisons pour lesquels on choisit d'encapsuler le tableau statique t à l'intérieur d'une structure.
Question 6 Programmez un constructeur `me_vide` permettant de construire le multi-ensemble vide. Programmez ensuite les opérations `add`, `remove`, `nb_occ` et `appartient`.
Question 7 Programmez une fonction d'affichage d'un multi-ensemble. Montrer sur un exemple ce qu'elle affichera concrètement à l'écran.

On souhaite maintenant ajouter de nouvelles opérations. On rajoute les déclarations suivantes :

```
bool inclus(Smultiset *a, Smultiset *b);// teste l'inclusion de m dans n
bool egal(Smultiset *a, Smultiset *b);  // teste l'égalité de deux multi-ensembles
int cardinal_diff(Smultiset *a);        // compte le nombre d'éléments distincts
int cardinal_total(Smultiset *a);       // compte le nombre total d'éléments
```

Question 8 Programmez la fonction `inclus`.
Question 9 Proposez une implantation du test d'égalité entre deux multi-ensembles. On réutilisera le test d'inclusion de la question précédente si nécessaire.
Question 9bis Implantez la fonction `inclus` en travaillant directement sur le tableau représentant le multiensemble.
Question 10 Programmez les fonctions `cardinal_diff` et `cardinal_total`.
Question 11 On souhaite maintenant définir les opérations d'union et d'intersection de deux multi-ensembles. Proposez des implantations pour ces deux opérations.

```
Smultiset* intersection(Smultiset *a, Smultiset *b);
Smultiset* reunion(Smultiset *a, Smultiset *b);
```