Examen d'Algorithmique - (2014/2015 session 1)

Durée : 1heure 30 minutes - Aucuns documents autorisés

Formation Ingénieurs CNAM

Ce sujet comporte 4 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. Enfin, il faut respecter les consignes notamment en ce qui concerne l'utilisation ou non de la récursivité.

1 Echauffement (8 points)

On rappelle la structure de données définie pour représenter les arbres d'entiers ainsi que les opérations de base `vide` et `e` (enracinement). 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 `oppose` (qui transforme chaque étiquette x en son opposé -x). On procèdera de manière récursive avec des effets de bord (le type de retour de la fonction demandée est `void`).

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

Question 2 Programmer de manière *récursive* une fonction `tous_positifs` qui vérifie que toutes les étiquettes de l'arbre sont positives.

```
bool tous_positifs(Sarbre *a)
{ ... }
```

Question 3 Programmer de manière *récursive* une fonction `nb_positifs` qui compte le nombre d'étiquettes positives dans un arbre.

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

Question 4 Ecrire un programme de test dans la fonction `main` qui produit la sortie suivante :

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

```
user@computer$ ~/test-arbre
3 6 8
-3 -6 -8
tous_positifs ? = true
4 7 -9
nb_positifs = 2
-4 -7 9
nb_positifs = 1
user@computer$
```

2 Petit problème : les listes d'association (12 points)

On cherche à représenter un environnement par une liste de couples formés d'un caractère `char` et d'un entier `int`. Par exemple, la liste [('x',3) ; ('y',-9) ;('z',0)] permet de représenter un état de la mémoire où la variable x contient la valeur 3, la variable y la valeur -9 et la variable z la valeur 0.

On suppose que l'on a défini une structure de données pour représenter les couples ainsi qu'une structure par chaînage pour représenter les listes de couples. L'extrait du fichier `.h` suivant précise comment tout cela est défini :

```
typedef struct scouple
{char variable; int valeur;} couple;
```

```
typedef struct liste
{
  couple c;
  struct liste *suivant;
} Sliste, *Liste;

Sliste* vide() /* construit une liste vide */
{
  return NULL;
}

Sliste* ajout(char v, int e, Sliste *l) /* ajoute le couple (v,e) en tete de la liste l */
{
  Sliste *m = (Sliste *)malloc(sizeof(Sliste));
  m->suivant = l;
  (m->c).variable=v;
  (m->c).valeur=e;
  return m;
}
```

Question 5 Programmer une variante `ajout_couple` qui utilise la fonction `ajout` pour ajouter un couple c à une liste l. Utiliser cette fonction ainsi que le constructeur `vide` pour construire la liste de couples [('x',3) ; ('y',-9) ;('z',0)] .

```
Sliste *ajout_couple(couple c, liste* l)
{ ... }
```

Question 6 Programmer une fonction d'affichage `affiche_liste` d'une liste de couples. On procèdera de manière itérative.

```
void affiche_liste(Sliste *l)
{ ... }
```

Question 7 Programmer une fonction `appartient` qui teste si une variable t appartient à la liste. On procèdera également de manière récursive.

```
int appartient(char t, Sliste *l)
{ ... }
```

Question 8 Programmer, en utilisant la fonction `appartient` de la question 7, une fonction `sans_doublons` qui vérifie que toutes les variables n'apparaissent qu'une seule fois dans la liste.

```
int sans_doublons(Sliste *l)
{ ... }
```

Question 9 Ecrire une fonction `renverse` qui retourne une liste avec les éléments dans l'ordre inverse. Par exemple, pour la liste [('x',3) ;('y',-9) ;('z',0)] donnée en exemple, on devra obtenir [('z',0) ; ('y',-9) ;('x',3)].
Indication : on pourrait utiliser une fonction auxiliaire et utiliser un accumulateur...

```
Sliste *renverse(Sliste *l)
{ ... }


Sliste *renv_acc(Sliste *l1, Sliste *l2)
{ ... }
```

Question 10 Programmer une fonction `maj` de mise à jour d'un couple (variable, valeur) dans la liste. Par exemple, on pourra vouloir faire passer la valeur contenue dans la variable 'x' à la valeur 12. Dans ce cas, la nouvelle liste obtenue en résultat sera [('x',12) ;('y',-9) ;('z',0)].

```
Sliste *maj(char t, int val, Sliste *l)
{ ... }
```