Informatique S3 **L2 CUPGE 2018–2019**

Contrôle continu

Durée : 1h30. Aucun document n'est autorisé.

I Exercice : *Codage (4 points)*

On rappelle qu'un nombre :

$$x = (-1)^s \cdot 2^{e-127} \cdot (1 + \frac{M}{2^{23}})$$

est codé sous la forme :

0	1 … 8	9 … 31
s	e	M

(a) (2 pts) Donnez le code hexadécimal du signe s, de l'exposant e, et de la mantisse M, ainsi que le code entier sur 32 bits, du nombre $-10,5$.

(b) (2 pts) On utilise la méthode par multiplications successives (en base 16).

II Exercice : *Test de compréhension (4 points)*

> Comme cela n'a pas été beaucoup vu en TP, on rappelle que si x est une valeur, alors :
>
> `( t ) x`
>
> est la traduction de cette valeur dans le type `t`.

On considère la fonction `main` suivante :

```
int main ( int argc , char * argv[] )
{
  int t[3] = {1,2,3} ;
  int * p1 = ( int * ) ( t + 1 ) ;
  int * p2 = ( int * ) ( &t + 1 ) - 1 ;
  * ( t + 2 ) = p1 [ 1 ] - t [ 0 ] ;
  p2[-2] = p1[1] ;
  printf ( "%d %d %d\n" , t[0] , t[1] , t[2] ) ;
  return 0 ;
}
```

(a) (2 points) Expliquez, en vous aidant si possible de schémas, quelles sont les valeurs, en tant qu'entier, de p_1 et p_2 en fonction de la valeur de t en tant qu'entier. Pour simplifier, on suppose que `sizeof ( int ) = 4`.

III Exercice : *Somme des éléments d'un tableau (4 points)*

On veut écrire une fonction `somme_partielle` qui prend en entrée l'adresse de la première case d'une tableau d'entiers, le nombre n de cases de ce tableau, un entier i qui devrait être l'indice d'une case dans le tableau, et l'adresse d'une variable (de type `int`) dans laquelle il faudra stocker :

$$\Sigma_{j=0}^{i} t[j]$$

c'est-à-dire la somme des éléments du tableau de l'indice 0 à l'indice i (inclus). Cette fonction renvoie 0 si le calcul est possible, et 1 sinon.

(a) (1 point) Donnez la déclaration de cette fonction.

(b) (0,5 points) Décrivez avec une expression C les cas d'erreur.

(c) (2,5 points) Écrivez la fonction.

IV Exercice : *Tas (8 points)*

Préliminaires. Un *arbre* est soit un arbre vide, soit est un nœud (la *racine* de cet arbre) qui est le père de deux *fils*, son fils droit et son fils gauche, qui sont eux-mêmes des arbres.

Les *tas* sont des tableaux qui codent certains arbres de la manière suivante :

— la valeur de la racine de l'arbre est dans la case 0 ;
— si la valeur d'un nœud est stockée dans la case i, alors la valeur de la racine de son fils gauche est stockée dans la case $2 \times i + 1$, et la valeur de la racine de son fils droit est stockée dans la case $2 \times i + 2$;
— la valeur d'un nœud est toujours plus petite que la valeur de ses fils.

Pour savoir quand s'arrêtent les valeurs du tas (les cases blanches de la Figure 1), on a besoin, en plus du tableau, du nombre d'éléments dans le tas. Pour décrire des tas en C, on utilise les déclarations suivantes :

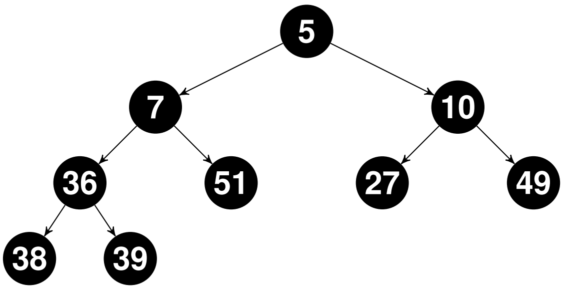
```
struct tas_s {
    int nb_elts; /* nombre d'éléments dans le tas */
    int * val; /* tableau des valeurs */
};
typedef struct tas_s * tas;
```

(a) (0.5 + 0.5 + 1 = 2 pts) Écrire trois fonctions fils_gauche, fils_droit, et pere qui, en fonction d'un indice i, calculent l'indice correspondant respectivement au fils gauche, au fils droit, et au père du nœud dont la valeur est stockée dans la case i. *Par exemple, pere(5)=2, et fils_gauche(2)=5.*

L'algorithme permettant d'insérer un élément dans un tas est donné informellement dans la Figure 3a.

(b) (1 pt) Justifiez (informellement) que l'insertion d'un élément dans un tas permet d'obtenir un tas, *i.e.*, que dans le tas obtenu, tout élément est plus petit que ses fils.

(c) (1 pt) De quels arguments (signification de la variable et son type) est-ce qu'une fonction qui prend en entrée un tas aura besoin ?

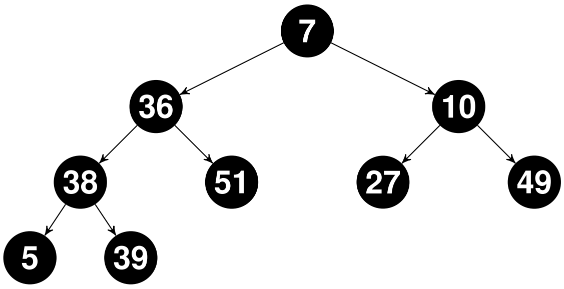


(a) Un tas, dont la racine a pour valeur 5, son fils gauche a une racine de valeur 7 et son fils droit a pour racine un nœud dont la valeur est 10. Les arbres vides ne sont pas représentés.

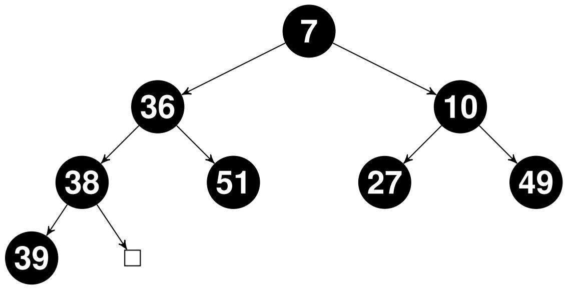
5	7	10	36	51	27	49	38	39			
0		2			5						

(b) Le tableau correspondant à ce tas : la valeur de la racine est dans la case 0, celle de son fils droit dans la case $2 = 2 \times 0 + 2$, et celle du fils gauche de ce fils droit est dans la case $5 = 2 \times 2 + 1$.

FIGURE 1 – Tas et leur représentation par un tableau



(a) Le tas de la Figure 1a après l'étape 2 de la suppression.



(b) Le tas de la Figure 1a après l'étape 5 de la suppression. Le nœud vide représente la décrémentation du nombre d'éléments. 39 est plus grand que 38, donc l'étape 6 ne fait rien

FIGURE 2 – Tas aux différentes étapes de la suppression.

(d) (2 pts) Écrire la fonction d'insertion d'un élément en C. On suppose qu'il y a strictement moins d'éléments dans le tas que le nombre maximal qu'il ne peut en contenir.

(e) (2 pts) Écrire la fonction de suppression du plus petit élément en C en se basant sur l'algorithme de la figure 3b. On suppose qu'il y a au moins un élément dans le tas.

1. On note i l'indice courant. Au départ, l'indice courant est l'indice de la première case vide dans le tableau (dans la Figure 1b, i vaut au départ 9) ;
2. mettre l'élément à ajouter dans la case d'indice i ;
3. Tant que :
   — l'indice courant est strictement positif ;
   — et la valeur du père de l'indice courant est plus grande que la valeur de l'indice courant

   échanger ces deux valeurs, et positionner l'indice courant sur son père ;
4. rendre le tas ;

(a) Algorithme d'insertion d'un élément.

1. On note i l'indice courant. Au départ, l'indice courant vaut 0 ;
2. Tant que le fils droit de l'indice courant est dans le tableau :
   — échanger la valeur du nœud courant avec celle du plus petit de ses fils ;
   — l'indice courant devient celui du fils avec lequel l'échange a été fait.
3. Si l'élément courant a un fils gauche, on échange ces deux éléments, et l'élément courant devient l'indice du fils gauche ;
4. on échange ensuite la valeur de l'élément courant et celle du dernier élément du tas ;
5. on décrémente de 1 le nombre d'éléments dans le tas ;
6. on remonte, comme pour l'insertion, l'élément courant tant qu'il est plus petit que son père.

(b) Algorithme de suppression du plus petit élément.

FIGURE 3 – Algorithmes sur les tas