LMD MIAS L2 S3 Examen Algorithmique et Structures de Données I durée : 1h 30

Exercice 1 (5,5pts)

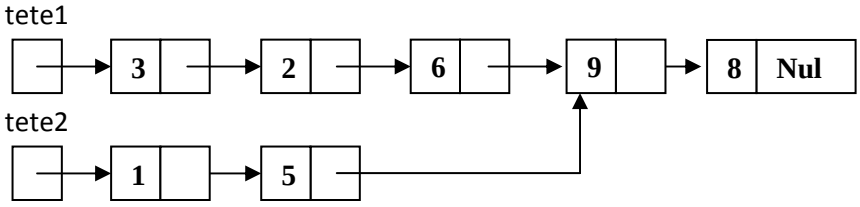
On dispose d'une suite de valeurs qu'on peut stocker dans une table de hachage «interne» ou bien dans un arbre binaire ordonné.

	clé	suiv	Occ
0	35	-1	1
1	10	4	1
2	12	-1	1
3			0
4	15	0	1

1) Dans le cas où ces valeurs sont déjà stockées dans une table de hachage interne (voir figure où le champs « Occ » désigne si la case est occupée (1) ou libre (0)) :
   a) Déterminer les valeur de la fonction de hachage pour chaque clé stockée
   b) Donner le schéma de la table dans le cas de suppression de la clé «15»
   c) Donner le schéma de la table dans le cas d'insertion de la clé «20», avec h(20) = 0
2) Dans le cas d'utilisation d'un arbre de recherche, donner le schéma de l'arbre à la fin des insertions 35, 10, 12, 15, 20
3) Quelle est la complexité moyenne de la recherche d'une valeur dans une liste de n éléments, dans le cas d'une table de hachage de taille m (chainage interne), et d'un arbre binaire ordonné ?

Exercice 2 (4pts)

On dispose de deux listes chaînées liste1 de tete1 et liste2 de tete2 qui ont une partie commune. Ecrire la fonction qui retourne le nombre des éléments appartenant uniquement à liste1. (dan l'exemple, on a 3 éléments qui appartiennent uniquement à liste1 : 3, 2, 6

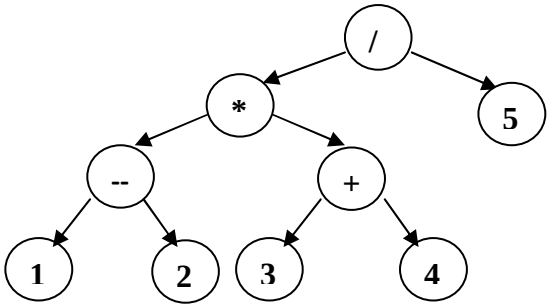


Exercice 3 (10,5pts)

```
const max = 50;
typedef struct pile{
 int tab[max];
 int  indiceSommet;
};
```

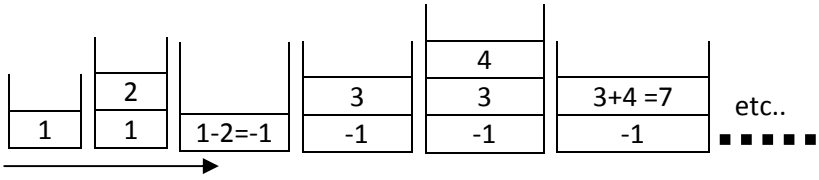
1) Soit la déclaration d'une pile contigües d'entiers, écrire les routines de gestion de cette structure:
   a) **void initialiserPile(pile &p)**
   b) **void empiler(pile &p, int x);**
   c) **int depiler (pile &p)**

2) Les expressions arithmétiques peuvent être représentées dans un arbre binaire comme le montre l'exemple suivant qui représente l'arbre de l'expression (1-2)*(3+4)/5.
   a) Donner les parcours en ordre postfixé(GDR), infixé (GRD), préfixé (RGD) et par niveau ;
   b) Ecrire la fonction de parcours postfixé d'une telle structure (version récursive) (utiliser la déclaration donnée)




```
typedef struct noeud {
 char val;
 struct noeud *gauche ;
 struct noeud *droit
};
```

3) Une méthodes pour calculer le résultats d'une telle expression consiste à faire un parcours postfixé de l'arbre et de traiter chaque nœud de la façon suivante : si le nœud est un opérande (un nombre), on l'empile dans une pile, si c'est une opération (+, -,*,/), on dépile les deux sommets et on applique dessus l'opération (voir l'exp.). Ecrire une fonction **void evaluer (noeud *racine,pile &p)** qui implémente cette méthode en effectuant un parcours postfixée (version recursive) et en utilisant la pile déclarée dans 1). Comment peut on afficher le résultat final ?



Indication :
- On suppose que les opérandes sont les nombres 0..9,
- Une méthode de transformer le caractère « car » ∈ 0..9 en un entier est d'écrire int x = car-48

Corrigé Type

Exercice 1 (5,5pts)

1)

a) Clés et valeurs de h

0.25 x 4 h(10) = 1; h(15) = 1 ; h(35) = 1 ; h(12) = 2 ;

b) Suppression de 15

	clé	suiv	Occ
0	35	-1	1
1	10	**0**	1
2	12	-1	1
3			0
4	15	0	**0**

0.5 (ligne 1) — 0.25 (ligne 4)

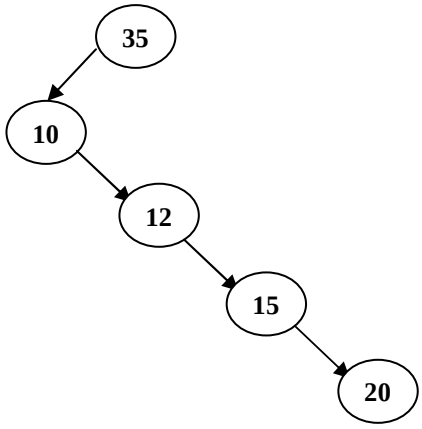
c) Ajout de 20, h(20) =0 (collision secondaire)

	clé	suiv	Occ
0	**20**	-1	1
1	10	**4**	1
2	12	-1	1
3			0
4	**35**	**-1**	**1**

0.5 (ligne 0) — 0.5 (ligne 1) — 0.5 (ligne 4)

2) arbre de recherche après insertion de 35, 10, 12, 15, 20

0.25 x 5



3) table de hachage : O(n/m) ; arbre de recherche $O(\log_2(n))$

0.5 x 2

Exercice 2 (4pts)

```
typedef struct bloc {.....0,25
 int val ;
 struct bloc* suiv ;
}
 bloc* courant 1= tete1 ;.....0.25
 int compteur = 0 ;.....0.25
 bool egal = false ; .....0.25
 while  (courant1 != NULL && !egal)  {.....0.25
   courant2 = tete2 ;.....0.25
   while (courant2 != NULL && !egal).....0.25
    if (courant1 == courant2) egal = true ;.....0.5
    else courant2 = courant2 ->suiv ;.....0.5
   if (!egal) {.....0.25
     compteur++ ;.....0.5
     courant1 = courant1 ->suiv ;.....0.5
   }
 }
```

N.B :

- d'autres méthodes sont envisageables comme l'utilisation d'un tableau de pointeurs, stockage de pointeurs des deux listes, ensuite recherche des deux pointeurs égaux
- si utilisation d'une autre méthode, évaluer selon le principe général utilisé

Exercice 3 (10,5pts)

1) Routine de gestion de la pile contigue

a) Initialisation **(0,5pts)**

```
void initialiserPile(pile &p){
     p.indiceSommet = -1..................................... 0.5
}
```

b) Empiler **(1pt)**

```
void empiler(pile &p, int x){
     p.tab[indiceSommet+1] = x ;............................. 0.5
     p.indiceSommet++ ;...................................... 0.5
}
```

c) Dépiler **(1,5 pt)**

```
int depiler (pile &p){
     if (p.indiceSommet != -1) {.............................. 0.5
       int x = p.tab[indiceSommet] ;.......................... 0.25
       p.indiceSommet-- ;...................................... 0.5
       return x ............................................... 0.25
     }
}
```

2) a) Parcours de l'arbre binaire **(2 pt)**

- Postfixé (GDR) : 12-34+*5/ 0.5
- Infixé (GRD) : 1-2*3+4/5 .....0.5
- Prefixe (RGD) : /*-12+345 .....0.5
- Par niveaux : /*5-+1234 .....0.5

b) Fonction parcours postfixe **(1.5 pt)**

```
int postfixe (noeud* racine){
     if (racine != NULL) {.................................... 0.25
       postfixe(racine->gauche) ;............................. 0.5
       postfixe(racine->droit) ;.............................. 0.5
       cout<<racine->val ..................................... 0.25
     }
}
```

3) Fonction Evaluation (4pts)

```
void  GDR (noeud *r, pile &p) {
int a,b, res;
 if (r !=NULL) {..................................................... 0.25
  GDR (r->gauche,p);................................................. 0.25
  GDR (r->droit,p);.................................................. 0.25
  if (r->val!='+' && r->val!='-' && r->val!= '*' && r->val!='/'){ ... 0.25
  empiler(r->val-48,p); ............................................. 0.5
  }
  else {
       a = depiler(p); .............................................. 0.25
       b = depiler (p); ............................................. 0.25
       if (r->val == '+') res = a+b; ................................ 0.25
       if (r->val == '-') res = b-a; ................................ 0.25
       if (r->val == '*') res = b*a; ................................ 0.25
       if (r->val == '/') res = b/a; ................................ 0.25
       empiler(res,p); .............................................. 0.5
   }
 }
}
```

- En accédant à p.tab[0] après la fin d'exécution de GDR **0,5**