Université Caen Normandie
DUT informatique
Algorithmique avancée

Année 2015-2016
Examen
14 décembre 2015
1 heure 30

Remarques :
- Les téléphones portables doivent être éteints.
- Aucun document ni machine électronique n'est permis.
- Sauf mention particulière, les questions peuvent être résolues de façon indépendante. Il est possible, voire même utile, pour répondre à une question, d'utiliser les réponses qui sont l'objet des questions précédentes (même si vous n'y avez pas répondu !).
- La clarté des réponses et la présentation des programmes seront appréciées.
- Le barème (total sur 20) n'est donné qu'à titre indicatif.

Exercice I : La fonction de Fibonacci

La fonction de Fibonacci exprime sous la forme suivante :

$$\begin{cases} Fibo(n) = Fibo(n-1) + Fibo(n-2) \\ Fibo(0) = 1 \\ Fibo(1) = 1 \end{cases}$$

Question 1 (2 points) :

Représenter sous forme d'un arbre les appels à la fonction $Fibo$ engendrés par $Fibo(4)$.
Combien de fois la fonction $Fibo$ est-elle appelée ?
Quel est le résultat de l'évaluation ?

Exercice II : réécriture des opérateurs

On se positionne dans un monde particulier dans lequel les étudiants ne savent faire que les choses suivantes :
- travailler sur des entiers positifs ou nuls
- ajouter 1 à un entier
- retirer 1 à un entier
- tester si un entier vaut 0 ou vaut 1.

Ils ont par contre une très bonne maitrise des fonctions récursives et savent réutiliser le travail déjà fait.

Question 1 (2 points) :

Compléter la méthode récursive add qui prend en entrée 2 entiers **positifs** et retourne la somme des deux entiers.

```
public static int add(int a, int b){
        if ( ..... ) return .....;
        else return 1+add(...., .....);
    }
```

Question 2 (2 points) :

Écrire une méthode récursive mul qui prend en entrée 2 entiers **positifs** et retourne le produit des deux entiers.
Votre méthode devra fonctionner notamment avec les cas suivants : $mul(4,5), mul(4,0), mul(4,1), mul(0,3), mul(1,3)$

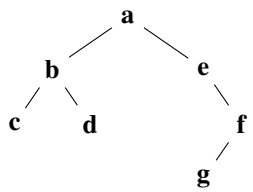
Exercice III : Parcours d'arbre

Question 1 (1 point) :
Soit la classe `ArbreBinaire` et la méthode récursive `affiche` suivante :

```
class ArbreBinaire{
  char v;
  ArbreBinaire filsG, filsD;

   public static void affiche(ArbreBinaire a)
   {
        if (a==null) return;
        else {
            System.out.print(a.v+"  ");
            affiche(a.filsG);
            System.out.print(a.v+"  ");
            affiche(a.filsD);
        }
    }
```

Soit une instance `A` de la classe `ArbreBinaire`. `A` est une référence sur l'arbre suivant :



Dans un progamme, on exécute l'instruction suivante :

```
affiche(A);
```

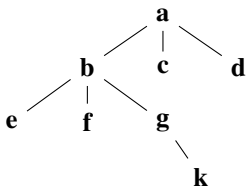
Que s'affiche-t-il sur la sortie standard ?

Question 2 (2 points) :
Soit la classe `ArbreTernaire` et la méthode récursive `affiche` suivante :

```
class ArbreTernaire{
  char v;
  ArbreTernaire filsG, filsM, filsD;

   public static void affiche(ArbreTernaire a)
   {
        if (a==null) return;
        else {
            System.out.print(a.v+"  ");
            affiche(a.filsG);
            affiche(a.filsM);
            System.out.print(a.v+"  ");
            affiche(a.filsD);
        }
    }
}
```

Soit une instance `B` de la classe `ArbreTernaire`. `B` est une référence sur l'arbre suivant :



Dans un progamme, on exécute l'instruction suivante :

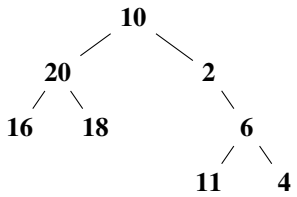
```
affiche(B);
```

Que s'affiche-t-il sur la sortie standard ?

Question 3 (3 points) :

Soit une structure d'arbre binaire dont les noeuds contiennent des nombres entiers. Les éléments importants de la classe sont donnés ci-dessous ainsi qu'un exemple d'arbre.

```
class ArbreBinaireInt{
  int v;
  ArbreBinaire filsG, filsD;
  ....
  public int getMin(){
     ....
  }
}
```



Écrire le code de la méthode `getMin` qui retourne la plus petite valeur contenue dans un arbre.

Exercice IV : Arbre lexicographique

On représente un dictionnaire sous forme d'arbre binaire. Dans un tel arbre :

— chaque lettre d'un mot est stockée dans un noeud différent. On suppose que les mots stockés dans le dictionnaire ne contiennent que des lettres.

— les premières lettres possibles pour le mot sont obtenues en suivant la branche de droite à partir de la racine (par exemple, sur la figure ci-dessous, les premières lettres possibles sont a, b ou c)

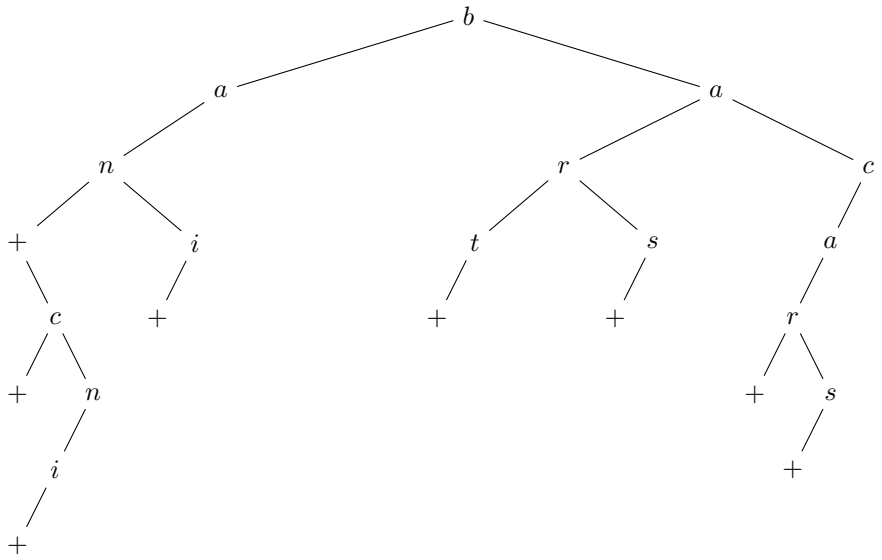
— le fils gauche d'un noeud contient la liste des lettres suivantes possibles. Cette liste est constituée des fils droits de ce fils gauche. Le caractère '+' est utilisé pour indiquer la fin d'un mot.

— pour accéder aux deuxièmes lettres des mots, on passe au fils gauche de la première lettre. Toutes les deuxièmes lettres possibles sont alors obtenues en prenant la branche de droite (par exemple, avec la première lettre a, les deuxièmes lettres possibles sont r et s). Pour avoir la troisième lettre, on passe au fils gauche de la deuxième lettre et ainsi de suite jusqu'à trouver '+'.

Soit l'arbre lexicographique suivant :

Question 1 (4 points) :

Écrire une fonction `afficheMots` qui affiche l'ensemble des mots de l'arbre à raison d'un mot par ligne. Pour l'arbre ci-dessus, le résultat attendu est :



`ban, banc, banni, bai, art, as, car, cas,`

Un prototype possible pour la méthode est :

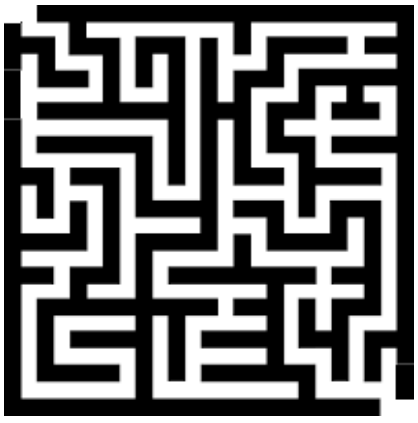
```
public void afficherMots(String s)
```

Le parametre `s` sert à construire le mot à afficher.

Exercice V : Sortie d'un labyrinthe

Question 1 (4 points) :

Soit le labyrinthe suivant :



Ce labyrinthe est stocké dans un tableau dans lequel une case peut contenir 3 caractères possibles :
— espace indique que la case est vide.
— le caractère X qui indique la présence d'un mur
— le caractère . qui indique que l'on est passé par ce point
Ecrire une méthode récursive qui s'appuie sur le backtracking pour rechercher le chemin qui permet d'aller du point d'entrée en haut à gauche au point de sortie en bas à droite.
Il est conseillé de laisser une trace (le caractère .) pour marquer le chemin en cours de construction.