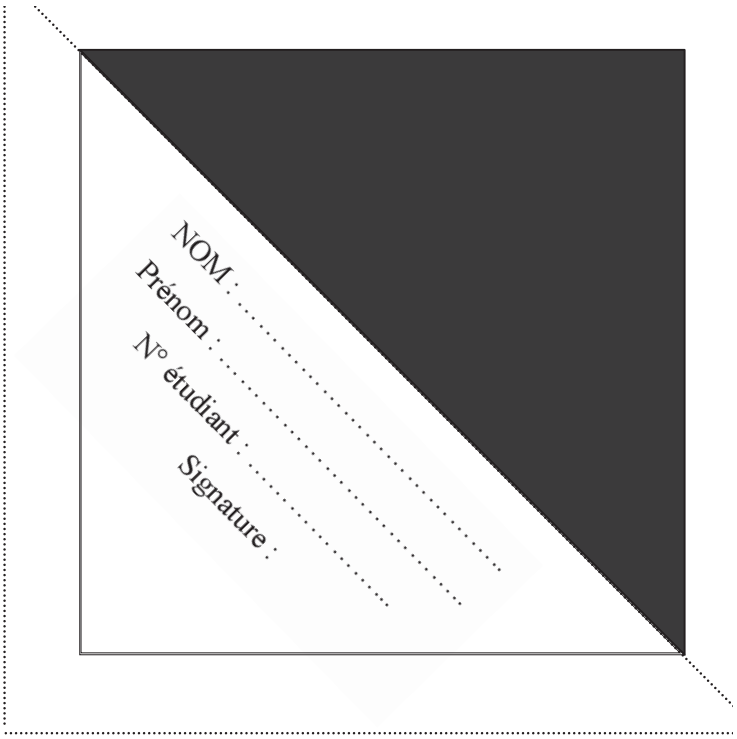
Université Claude Bernard Lyon 1
Licence Sciences, Technologies, Santé – L2
Année 2010-2011, 1er semestre

LIF5 – Algorithmique & Programmation procédurale

Contrôle final
18 janvier 2010

Note :



Documents, calculatrices, ordinateurs, lecteurs mp3 et téléphones portables interdits.

Le barème est donné à titre indicatif.

Travaillez au brouillon d'abord de sorte à rendre une copie propre. Nous ne pouvons pas vous garantir une copie supplémentaire si vous vous trompez.

Exercice 1 : Questions diverses (5 points)

Chaque question est sur un point. Pour les questions à choix multiples : 0 si une proposition fausse est cochée ; sinon, pourcentage de propositions justes cochées.

1. Dessiner l'arbre binaire de recherche obtenu lorsqu'on insère successivement aux feuilles de l'arbre les éléments du jeu de données suivant : 12, 25, 14, 5, 18, 9, 41, 8.

2. Dans quel ordre faut-il fournir le jeu de données de la question 1 pour obtenir un arbre binaire de recherche totalement dégénéré (déséquilibré) ?

3. Parmi les structures de données suivantes, laquelle ou lesquelles accèdent en temps linéaire (O(N)) à un élément quelconque?
   - ☐ tableau dynamique
   - ☐ liste doublement chaînée
   - ☐ liste simplement chaînée circulaire avec sentinelle
   - ☐ arbre binaire de recherche bien équilibré.

4. Le coût d'un ajout en queue dans un tableau statique (de taille > k) contenant déjà k éléments est au pire :
   - ☐ $O(k^2)$
   - ☐ O(k.ln(k))
   - ☐ O(ln(k))
   - ☐ O(k)
   - ☐ O(1)

5. Le coût d'un ajout dans un arbre binaire de recherche contenant déjà k éléments est au pire :
   - ☐ $O(k^2)$
   - ☐ O(k.ln(k))
   - ☐ O(ln(k))
   - ☐ O(k)
   - ☐ O(1)

Exercice 2 : Suppression d'éléments dans une liste doublement chaînée (5 points)

Dans cet exercice, on considère un module Liste permettant de gérer des listes doublement chaînées, non circulaires.

```
typedef double Elem;
struct sCellule
{
  Elem info;
  struct sCellule *suivant;
  struct sCellule *pred;
};
typedef struct sCellule Cellule;
```

```
struct sListe
{
  Cellule *adPremiere;
  Cellule *adDerniere;
};
typedef struct sListe Liste;
```

Ecrivez une procédure C qui prend une liste **l** et un élément **e** en paramètres et qui libère toutes les cellules de la liste l contenant **e**. Remarque : le champ « pred » de la première cellule et le champ « suivant » de la dernière cellule pointent sur NULL.

```
/* Préconditions : ....................................................

   Postconditions : ...................................................

                    ..................................................... */


void supprime(.....................................................)
/* complétez la liste des paramètres */
{

/* Complétez les variables locales et le code de cette procédure. Vous ferez
attention aux cas particuliers où e est en tête ou en fin de liste de liste */









}
```

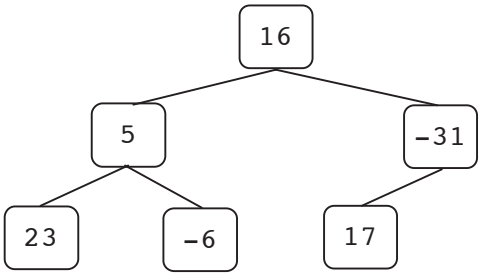
Exercice 3 : Procédures sur les arbres binaires (10 points)

Dans cet exercice, on considère le module Arbre permettant de gérer des arbres binaires d'entiers signés.

```
typedef int Elem;
struct sNoeud {
  Elem info;
  int niveau;
  struct sNoeud * fg;
  struct sNoeud * fd;
};
```

```
typedef struct sNoeud Noeud;


struct sArbre {
  Noeud * adRacine;
};
typedef struct sArbre Arbre;
```

1. Quel affichage obtient-on pour l'arbre suivant lorsqu'on effectue un parcours **postfixé** ?

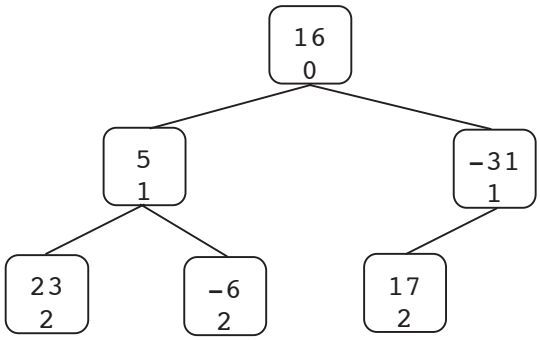


2. Donnez l'implantation en langage C du parcours **en largeur** pour afficher un arbre binaire. La procédure devra être **itérative** et non récursive. Vous pourrez appeler les sous-programmes suivants sans en donner le code :

```
void afficherElem(Elem e);
/* Précondition : aucune
   Postcondition : la valeur de e est affichée sur la sortie standard,
       suivie d'un espace */

void initialiserFile(File *pF);
/* Précondition : *pF non préalablement initialisée
       Postcondition : *pF initialisée en File vide */

void testamentFile(File *pF);
/* Précondition : *pF préalablement initialisée
       Postcondition : *pF prête à disparaître (ne doit plus être
       utilisée) */
void enFiler(File *pF, Noeud * adrNoeud);
/* Précondition : *pF initialisée
       Postcondition : adrNoeud est placée en fin de (*pF) */

void deFiler(File *pF);
/* Précondition : *pF initialisée, *pF non vide
       Postcondition : le premier élément de *pF est retiré */

Noeud * consulterPremier(File F);
/* Précondition : F non vide
       Résultat : adresse située au début de F */

int testFileVide(File F);
/* Précondition : F initialisée
       Résultat : 1 si F est vide, 0 sinon */
```

```
/* Précondition : ..................................................

   Postcondition : ..............................................

                   ..............................................
*/

void affichageIteratif(.....................................................)
/* paramètre(s) à compléter */
{
  File FileA;
  Noeud * pN;

  /* complétez le code de la procédure */









}
```

3. On veut à présent indiquer dans chaque nœud le niveau dans lequel il se situe, sachant que la racine est au niveau 0 (voir schéma ci-dessous). Ecrivez une procédure **C** qui effectue cette opération. Cette procédure pourra être itérative ou récursive, et pourra utiliser ou non les sous-programmes du module File.




```
/* Précondition : l'arbre passé en paramètre est initialisé, il peut être vide
 Postcondition : tous les nœuds de l'arbre sont numérotés avec le niveau dans
lequel ils sont  */

void numNiveau(.....................................................)
/* paramètre(s) à compléter */
{









}
```