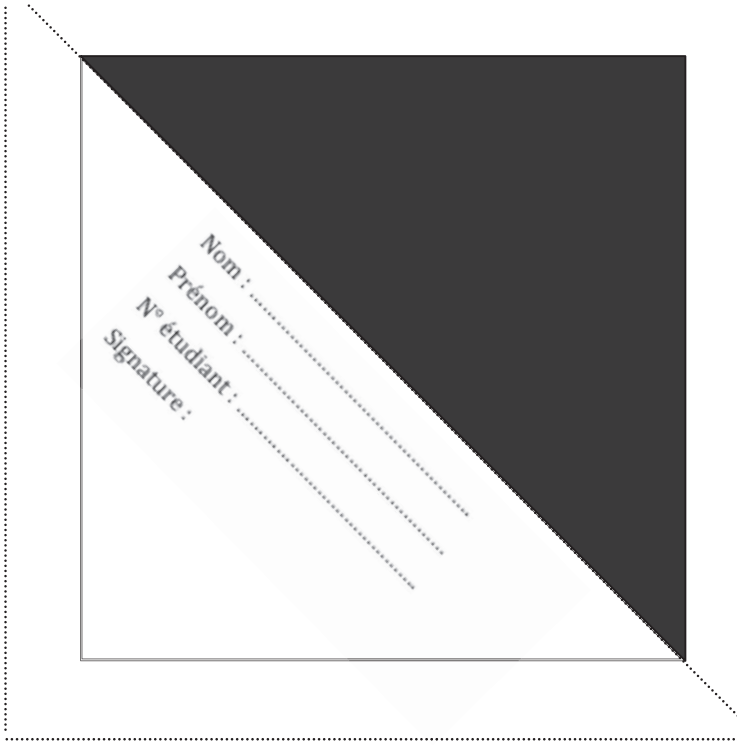
Université Claude Bernard Lyon 1
Licence Sciences, Technologies, Santé – L2
Année 2010-2011, 2ème semestre

LIF5 – Algorithmique & Programmation procédurale

Contrôle final du 20 juin 2011

Durée : **1h30**

Note :



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

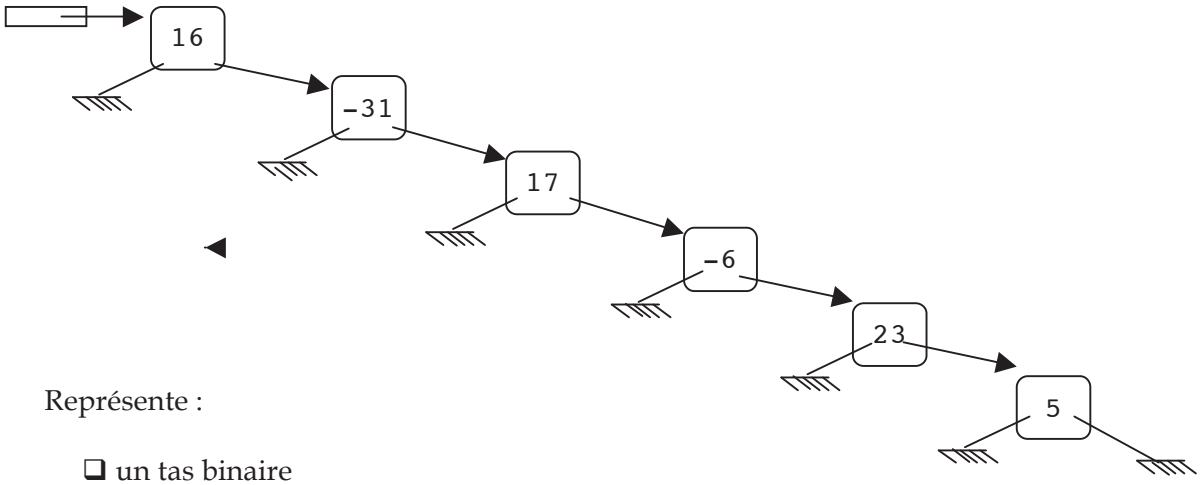
Le barème est donné à titre indicatif.

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. La représentation graphique suivante :



Représente :

- ❑ un tas binaire
- ❑ un arbre binaire
- ❑ un arbre binaire bien équilibré
- ❑ un arbre binaire de recherche
- ❑ une liste doublement chaînée

2. Cochez la ou les propositions correctes au sujet d'un tas binaire décroissant.
   ❑ {7, 6, 5, 4, 3, 2, 1} est un tas binaire décroissant.
   ❑ Tout tas binaire décroissant est trié dans l'ordre croissant.
   ❑ Tout tas binaire décroissant est trié dans l'ordre décroissant.
   ❑ Un tas binaire décroissant peut être stocké dans un tableau dynamique.
   ❑ k éléments initialement dans un ordre quelconque peuvent être réorganisés en un tas binaire décroissant en $O(\log_2(k))$.

3. Parmi les structures de données suivantes, laquelle ou lesquelles permettent d'accéder en temps constant au i-ème élément, quel que soit i ?
   ❑ tas binaire décroissant
   ❑ liste simplement chaînée circulaire
   ❑ liste doublement chaînée circulaire
   ❑ arbre binaire de recherche
   ❑ tableau statique

4. Cochez la ou les affirmations correctes au sujet d'une File d'Elements implantée sous forme de liste chaînée.
   ❑ les Elements sont stockés dans le segment Tas de l'espace d'adressage
   ❑ les Elements sont stockés de façon contiguë en mémoire vive
   ❑ les Elements sont stockés dans un fichier binaire sur le disque dur
   ❑ il faut mémoriser des pointeurs en plus des éléments eux-mêmes

5. Qu'appelle-t-on le « coût amorti » d'une insertion dans une structure de données dynamique ?
   ❑ le nombre d'opérations élémentaires effectuées dans le pire des cas
   ❑ le nombre d'opérations élémentaires effectuées dans le meilleur des cas
   ❑ le nombre d'opérations élémentaires moyenné sur plusieurs insertions successives
   ❑ le nombre d'opérations élémentaires exprimé relativement à celui pour une suppression

Exercice 2 : Fusion de deux listes chaînées (9 points)

Dans cet exercice, on considère un module Liste permettant de gérer des listes de « doubles » simplement chaînées, non circulaires.

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

```
struct sListe
{
  Cellule *prem;
};
typedef struct sListe Liste;
```

A. Ecrivez en C une fonction qui décroche la cellule de tête d'une liste chaînée bien initialisée, éventuellement vide et qui renvoie l'adresse de cette cellule. Il ne faut faire appel à aucune procédure ou fonction vue en cours, en td ou en tp.

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

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

   Résultat : .........................................................
*/

.................  decrocheTete(.........................)
/* complétez le type de retour de la fonction et la liste des paramètres */
{









}
```

B. Ecrire en C une procédure qui chaîne en tête d'une liste, éventuellement vide mais bien initialisée, une cellule déjà créée et bien initialisée dont l'adresse est passée en paramètre de la procédure. La liste est elle aussi passée en paramètre de la procédure.

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

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

*/

......... chaineTete(.........................................)
/* complétez la liste des paramètres */
{









}
```

C. Soit deux listes chaînées L1 et L2, triées dans l'ordre CROISSANT. Ecrivez en C une procédure qui fusionne ces deux listes pour obtenir une liste triée dans l'ordre DECROISSANT. Exemple, L1 = {5, 7, 9, 11} ; L2 = {6, 10, 15}. Le résultat obtenu est {15, 11, 10, 9, 7, 6, 5}. On interdit la création de nouvelles cellules, c'est-à-dire qu'à la fin de la procédure, les deux listes L1 et L2 sont vides.

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

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

                    ..................................................

*/
..... FusionSansCreation(.....................................)
/* complétez la liste des paramètres */
{









}
```

D. On suppose donnée une procédure dont le prototype est

```
void CreeListeTriee (Liste * l, int taille) ;
/* Précondition : la liste l est bien initialisée, taille est valide.
   Postcondition : construit aléatoirement l triée par ordre croissant avec
taille éléments */
```

Ecrivez le programme principal qui utilise cette procédure pour créer 2 listes triées par ordre croissant et qui les fusionne en appelant la procédure **`FusionSansCreation()`**. Toute procédure appelée doit être préalablement définie, à l'exception de CreeListeTriee.

```
..........main(.......................)
{









}
```

Exercice 3 : Procédures sur les arbres binaires (6 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;
  struct sNoeud * fg;
  struct sNoeud * fd;
};
typedef struct sNoeud Noeud;
```

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

A. On considère un arbre binaire de recherche a non vide. Ecrire en C, la fonction qui renvoie l'adresse du Nœud contenant le minimum dans l'arbre a.

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

   Résultat.....................................................................

*/
.............. minimum(.....................................................)
/* paramètre(s) à compléter */
{









}
```

B. On considère un arbre binaire a non vide. Ecrire en C une fonction ITERATIVE qui recherche l'adresse du père d'un nœud de l'arbre a, contenant l'Elem e donné. On supposera que le nœud recherché n'est pas à la racine de a. 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 : ..........................................................

   Résultat : ..............................................................

*/
..................recherchePere(.................................)
/* paramètre(s) à compléter */
{









}
```