Université Claude Bernard Lyon 1

Année universitaire : 2020 / 2021

LIFAP3 : Algorithmique et programmation avancée

ECA – Epreuve Commune Anonyme – Session 1
5 janvier 2020
Durée : 1h30

Note : / 20

coller ici

Nom :
Prénom :
N° d'étudiant :
Signature :

coller ici

Documents et téléphones portables interdits. Le barème est donné à titre indicatif. Répondez dans les emplacements prévus à cet effet. Travaillez au brouillon d'abord de sorte à rendre une copie propre – nous ne pouvons pas vous garantir une copie supplémentaire. **Il sera tenu compte de la présentation et de la clarté de vos réponses**.

Exercice 1 : Arbre binaire de recherche (15 points)

Dans cet exercice nous nous intéressons aux arbres binaires de recherche (ABR).

Pour rappel, dans un ABR tous les éléments dans le fils gauche d'un nœud sont plus petits que ce nœud et tous les éléments à droite sont plus grands (cf. Annexe pour les fonctions membres de la classe **`Arbre`**).

Pour rappel également, un **`Noeud`** est défini comme une structure contenant trois champs : **`info`** de type **`ElementA`**, **`fg`** et **`fd`** tous les deux de type pointeur sur **`Noeud`**. La seule donnée membre de la classe **`Arbre`** est **`adRacine`**, un pointeur sur le **`Noeud`** racine de l'arbre.

Question 1.1 : Donner le code C++ de la fonction membre **`void`** `insererElement (ElementA e);` faite en TP. Si vous avez besoin de créer une autre fonction membre, donner également son code.

Dans la suite de cet exercice, nous allons ajouter et tenir à jour un nouveau champ de la structure **`Noeud`** : le **`statut`**. Le statut d'un nœud est soit : une feuille (**`feuille`**), un nœud avec uniquement un fils gauche (**`peregauche`**), un nœud avec uniquement un fils droit (**`peredroit`**), un nœud avec deux fils (**`pere`**). Les 4 valeurs possibles d'un statut de nœud sont représentées par une énumération nommée **`statutNoeud`** :

```
enum statutNoeud {feuille, peregauche, peredroit, pere};
```

Il sera donc possible dans la suite de cet exercice de créer et manipuler une variable de type `statutNoeud` (qui sera en fait un entier). Par exemple l'instruction : `statutNoeud s = feuille;` crée une variable `s` de type `statutNoeud` qui a comme valeur `feuille`.

Question 1.2 : Donner le code C++ de la nouvelle déclaration de la structure **`Noeud`** contenant le nouveau champ **`statut`**.

Question 1.3 : Donner la nouvelle version du code de la fonction membre `void insererElement (ElementA e);` Si vous avez besoin de créer une autre fonction membre, donner également son code. Cette nouvelle version doit bien entendue mettre à jour les champs **`statut`** des nœuds concernés par l'insertion.

Soit la nouvelle fonction membre de la classe **Arbre** suivante qui compte et retourne la quantité de nœuds de chacun des 4 statuts possibles.

```
unsigned int * Arbre::compteStatut () const {
    // Création d'un tableau de 4 entiers sur le tas
    unsigned int * compteur = new unsigned int [4];
    // Initialisation des valeurs à zéro
    compteur[feuille] = compteur[peregauche] = compteur[peredroit] = compteur[pere] = 0;
    // Appel à la fonction récursive
    compteStatutDepuisNoeud(adRacine,compteur);
    // Retour du résultat (le tableau)
    return compteur;
}
```

La fonction retourne les quantités calculées dans un tableau de taille 4 alloué sur le tas dans la fonction. Les types énumérés étant des entiers commençant à zéro, nous pouvons les utiliser comme indices dans le tableau. Après initialisation, et avant de retourner le résultat, la fonction appelle la fonction récursive `compteStatutDepuisNoeud` qui effectue, de manière récursive, un parcours infixe de l'arbre pour calculer les valeurs du tableau.

Question 1.4 : Compléter le code de la fonction récursive `compteStatutDepuisNoeud`.

```
void Arbre::compteStatutDepuisNoeud (…… , ……) const {









}
```

Question 1.5 : Donner le code C++ d'un programme principal qui crée un objet de la classe **Arbre**, y ajoute 20 éléments (entiers) aléatoires et qui affiche le nombre de nœuds de chacun des 4 statuts possibles.

```
#include "Arbre.h"
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;

int main () {
    srand(time(NULL));
```

```
}
```

Question 1.6 : Quelle relation existe entre les valeurs calculées (nombres de nœuds de chaque statut) et la propriété d'équilibre de l'arbre ?

Question 1.7 : Un autre choix que de stocker le statut d'un nœud dans la structure **`Noeud`** serait de stocker le degré de ce nœud (nombre de fils). Sans donner de code, quelles seraient les modifications à apporter à la structure **`Noeud`** et à la fonction d'insertion d'un élément dans l'arbre pour tenir compte de ce nouveau choix.

Exercice 2 : Questions rapides (5 points)

Répondez succinctement, sans justification approfondie, aux questions suivantes.

Question 2.1 : Quelle est la complexité, dans le pire cas, de l'algorithme de tri par sélection ? Justifier brièvement votre réponse.

Question 2.2 : Pourquoi le contenu d'un tableau statique n'est pas recopié lorsqu'on l'affecte à un autre tableau ?

Question 2.3 : Quelle est l'entête complète de la fonction C++ permettant de tester si un fichier a bien été ouvert ?

Question 2.4 : Quelle est la commande g++ à effectuer pour créer un exécutable à partir de deux fichiers objets `objet1.o` et `objet2.o` ?

Question 2.5 : Combien de règles doit comporter, au minimum, un fichier makefile devant créer un exécutable à partir de deux modules et un programme principal ?

Question 2.6 : Quel est le principal avantage d'un tableau dynamique par rapport à un tableau statique ?

Question 2.7 : Quelles sont les complexités des procédures d'ajout en tête et d'ajout en queue lorsque la liste est simplement chaînée et non circulaire (comme en CM/TD) ?

Question 2.8 : Dans quel ordre doit-on ajouter en tête les éléments 1, 2 et 3 pour obtenir une liste triée dans l'ordre décroissant ?

Question 2.9 : Quelles sont les complexités (amorties) des fonctions `empiler` et `depiler` si on implémente une pile avec un tableau dynamique et que le sommet de pile est la dernière case du tableau ?

Question 2.10 : Quelles sont les complexités (amorties) des fonctions `enfiler` et `defiler` si on implémente une file avec un tableau dynamique et que le premier de la file est la dernière case du tableau ?

Annexe : liste des fonctions membres des classes TableauDynamique, Liste, File, Pile et Arbre (constructeurs et destructeurs omis)

Classe TableauDynamique

```
void vider ();
void ajouterElement (ElementTD e);
ElementTD valeurIemeElement (unsigned int indice) const;
void modifierValeurIemeElement (ElementTD e, unsigned int indice);
void afficher () const;
void supprimerElement (unsigned int indice);
void insererElement (ElementTD e, unsigned int indice);
int rechercherElement (ElementTD e) const;
```

Classe Liste

```
void vider ();
bool estVide () const;
unsigned int nbElements () const;
ElementL iemeElement (unsigned int indice) const;
void modifierIemeElement (unsigned int indice, ElementL e);
void afficherGaucheDroite () const;
void afficherDroiteGauche () const;
void ajouterEnTete (ElementL e);
void ajouterEnQueue (ElementL e);
void supprimerTete ();
int rechercherElement (ElementL e) const;
void insererElement (ElementL e, unsigned int indice);
void supprimerElement (ElementL e);
```

Classe File

```
void enfiler (ElementF e);
ElementF premierDeLaFile () const;
void defiler ();
bool estVide () const;
unsigned int nbElements () const;
```

Classe Pile

```
void empiler (ElementP e);
ElementP consulterSommet () const;
void depiler ();
bool estVide () const;
```

Classe Arbre

```
bool estVide () const;
void vider ();
void insererElement (ElementA e);
void supprimerElement (ElementA e);
Noeud * rechercherElement (ElementA e) const;
```