

Année universitaire : 2019 / 2020

LIFAP3 : Algorithmique et programmation avancée

ECA – Epreuve Commune Anonyme
8 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 : Recherche dans un arbre binaire de recherche (ABR) (6 points)

Dans cet exercice, on s'intéresse aux arbres binaires de recherche implémentés par la classe **Arbre** vue en CM/TD/TP. Pour rappel, la classe **Arbre** a une seule donnée membre : **adRacine** de type pointeur sur **Noeud**. La structure **Noeud** a trois champs : **info** de type **ElementA**, **fg** et **fd** tous deux de type pointeur sur **Noeud**. Voir l'annexe pour les fonctions membres de la classe **Arbre**.

Question 1.1 : Donner le code C++ de la fonction membre :

```
Noeud * rechercherElement (ElementA e) const;
```

Pour rappel, cette fonction a pour but de retourner l'adresse du **Noeud** de l'arbre contenant l'élément passé en paramètre. Si l'élément n'est pas présent dans l'arbre la fonction retourne **NULL**.



Question 1.2 : Donner le code C++ de la fonction membre de la classe **Arbre** dont l'entête en notation algorithmique est donné ci-dessous. Vous pourrez notamment utiliser la fonction **rechercherElement** que vous venez d'écrire.

Fonction tabNoeudsDepuisListe (l : Liste d'ElementA) : lien sur TableauDynamique de liens sur Noeud
Pré-condition : la liste est une liste simplement chaînée non circulaire d'ElementA (comme vu en CM/TD)

Post-condition : l'arbre de l'instance est inchangé

Paramètres en mode donnée : l

Résultat : retourne l'adresse d'un tableau dynamique, alloué sur le tas, qui contient des liens sur Noeud. Pour chaque élément de la liste l le tableau contiendra le lien sur son Noeud lorsque l'élément appartient à l'arbre. Si aucun élément n'est présent la fonction retournera l'adresse d'un tableau vide.

Exemple : si l=(8 4 2 7) et que l'arbre contient les éléments 8 et 2 (mais pas 4 ni 7), alors la fonction retournera un tableau de deux éléments : l'adresse du nœud contenant l'élément 8 suivie de l'adresse du nœud contenant l'élément 2.

Exercice 2 : Implémentation d'une pile avec une liste chaînée (7 points)

Dans le TP7, vous avez implémenté une pile grâce à un tableau dynamique. Dans cet exercice, vous allez l'implémenter grâce à une liste chaînée. La classe **Pile** aura donc une seule donnée membre **l** de type **Liste**. Voir l'annexe pour les fonctionnalités disponibles dans la classe **Liste**.

Donner le code C++ de chacune des fonctions membres de la classe **Pile** suivantes.

```
//Constructeur. Postcondition : la pile est initialement vide
Pile::Pile () {

}

// Destructeur. Postcondition : la mémoire allouée sur le tas est libérée
Pile::~Pile () {

}

// Empile l'élément e. Postcondition : le sommet de la pile est une copie de e
void Pile::empiler (ElementP e) {

}

// Dépile le sommet de la pile. Précondition : la pile n'est pas vide
// Postcondition : le sommet de la pile est dépilé
void Pile::depiler () {

}

// Consulte le sommet de la pile. Précondition : la pile n'est pas vide
// Résultat : l'élément qui se trouve au sommet de la pile
ElementP Pile::consulterSommet () const {

}

// Dépile et retourne le sommet de la pile. Précondition : la pile n'est pas vide
// Postcondition : le sommet de la pile est dépilé
// Résultat : l'élément qui se trouvait au sommet de la pile
ElementP Pile::traiterPile () {

}

// Retourne si la pile est vide. Résultat : vrai si la pile est vide, faux sinon
bool Pile::estVide () const {

}

// Affiche le contenu de la pile. Postcondition : les éléments contenus dans la pile sont
// affichés à l'écran (le sommet est affiché en dernier)
void Pile::afficher() const {

}
```

Exercice 3 : Questions courtes (7 points)

Répondez succinctement, sans justification approfondie, aux questions suivantes.

Question 3.1 : Sur une machine à architecture 64 bits, sur combien d'octets est stockée une adresse mémoire ?

Question 3.2 : Comment passe-t-on un tableau statique en paramètre en mode donnée ?

Question 3.3 : Comment doit-on déclarer un pointeur **p** en paramètre en mode donnée-résultat ?

Question 3.4 : Que doit-on écrire pour récupérer l'adresse d'une variable **v** ?

Question 3.5 : Donner l'entête du constructeur par copie de la classe **MaClasse**.

Question 3.6 : Donner l'entête de l'opérateur membre **+** surchargé dans la classe **Date**.

Question 3.7 : Pourquoi le pire cas est le cas le plus intéressant à étudier en analyse algorithmique ?

Question 3.8 : Que vérifie-t-on dans la condition de conservation d'un invariant de boucle ?

Question 3.9 : Quel est l'invariant du tri par sélection ?

Question 3.10 : Donner l'instruction permettant d'écrire la valeur contenue dans la variable **montant** suivie d'un espace puis de la chaîne de caractères "euros" dans une variable fichier nommé **monFichier**.

Question 3.11 : Quelle est la commande g++ à effectuer pour créer un exécutable à partir de deux fichiers objets **objet1.o** et **objet2.o** ?

Question 3.12 : Combien de règles doit comporter, au minimum, un fichier makefile devant créer un exécutable à partir de deux modules et un programme principal ?

Question 3.13 : Quelle est la complexité amortie, en notation O , d'un ajout d'élément dans un tableau dynamique lorsque l'on double la capacité s'il n'y a plus d'espace libre ?

Question 3.14 : Dans quel ordre doit-on ajouter en tête les éléments 1, 2 et 3 pour obtenir une liste triée dans l'ordre décroissant ?

**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;
```