ITI3

Algorithmique avancée et Programmation C

Durée : *1h50*
Documents autorisés : **AUCUN**

Informations

Avant de commencer l'examen

Connectez vous à la machine et suivez les instructions suivantes :
— copiez l'archive `examen.zip` qui se trouve dans `/opt/files` dans votre répertoire (commande `cp`) ;
— après l'avoir décompressée (commande `unzip` [1]), renommez le répertoire `C` en lui donnant comme nom : votre nom suivi de votre prénom, sans espace, sans caractères accentués et en minuscule sauf pour les premières lettres du prénom et du nom. Par exemple, si « Paul Du Villaré » devait passer cet examen, il nommerai son répertoire « DuvillarePaul ».

À la fin de l'examen

Cinq minutes avant la fin de l'examen :
— assurez vous que les fichiers sont au format `UTF-8` (c'est par défaut le cas) ;
— assurez vous que votre projet compile ;
— créez une archive de votre répertoire (commande `zip`), que vous nommerez comme le répertoire ;
— déposez sur moodle votre archive (cours « algorithmique avancée et programmation C », section « examens pratiques »).

Attention :
— le lien de dépôt est actif uniquement 10 minutes, si vous dépassez le délai, votre note sera 0 ;
— on ne peut déposer qu'une seule fois le projet sur Moodle.

Quelques conseils

Pour réussir votre examen suivez les conseils suivants :
— respectez bien les consignes données ci dessus. **Le fait de ne pas les respecter vous fera perdre 2 points sur la note finale** ;
— par défaut le projet compile et s'exécute. Après le développement de chaque fonction C, compilez votre projet (`make`) et ne passez à la fonction suivante que lorsque votre projet compile : **le fait de rendre un projet qui ne compile pas ou ne s'exécute pas (par exemple à cause d'un *segmentation fault*), divisera par deux votre note finale** ;
— la note finale sera fonction :
  — du nombre de tests unitaires et d'assertions qui seront valides ;
  — du nombre de *Warning* (compilation avec l'option `-Wall`) qui seront affichés ;
  — de la quantité, de la qualité et de la lisibilité du code (indentation et nom des identifiants).

1. si vous décompressez cette archive à partir de l'interface graphique d'Ubuntu, elle produira le répertoire `examen/C`, ce que je ne veux pas, je veux uniquement le répertoire `C` !

Une bibliothèque proposant des AVL génériques

L'objectif de cet examen est le développement d'un module (`avl`) en C proposant le type `AVL_ArbreBinaireRecherche` générique. La conception de ce type utilise le type `AB_ArbreBinaire`, lui aussi générique, proposé par le module `arbreBinaire` vu en cours.

Analyse

Le type C `AVL_ArbreBinaireRecherche` est l'implantation en C du TAD ABR (pour Arbre Binaire de Recherche) suivant :

Nom:	ABR (ArbreBinaireDeRecherche)	
Paramètre:	Element ($\forall e_1 \in Element, e_2 \in Element, e_1 \neq e2 \Rightarrow e_1 < e_2$ *ou* $e_1 > e_2$)	
Utilise:	**Booleen**	
Opérations:	aBR:	$\rightarrow$ ABR
	estVide:	ABR $\rightarrow$ **Booleen**
	inserer:	ABR $\times$ Element $\nrightarrow$ ABR
	supprimer:	ABR $\times$ Element $\rightarrow$ ABR
	estPresent:	ABR $\times$ Element $\rightarrow$ **Booleen**
	obtenirElement:	ABR $\nrightarrow$ Element
	obtenirFilsGauche:	ABR $\nrightarrow$ ABR
	obtenirFilsDroit:	ABR $\nrightarrow$ ABR
Préconditions:	obtenirElement(a):	$non(estVide(a))$
	...:	...

Conception

Dans le paradigme de la programmation structurée, ces opérations correspondent aux signatures des procédures et fonctions suivantes :

— **fonction** aBR () **:** ABR
— **fonction** estVide (unArbre : ABR) **: Booleen**
— **procédure** insérer (**E/S** unArbre : ABR, **E** element : Element)
— **procédure** supprimer (**E/S** unArbre : ABR, **E** element : Element)
— **fonction** estPresent (unArbre : ABR, element : Element) **: Booleen**
— **fonction** obtenirElement (unArbre : ABR) **:** Element
  ⌊**précondition(s)** non estVide(unArbre)
— **fonction** obtenirFilsGauche (unArbre : ABR) **:** ABR
  ⌊**précondition(s)** non estVide(unArbre)
— **fonction** obtenirFilsDroit (unArbre : ABR) **:** ABR
  ⌊**précondition(s)** non estVide(unArbre)

Conception détaillée

L'algorithme de la fonction `estPresent` est :

fonction estPresent (a : ABR, e : Element) **: Booleen**
  Déclaration temp : ABR
debut

```
   si estVide(a) alors
      retourner FAUX
   sinon
      si e=obtenirElement(a) alors
         retourner VRAI
      sinon
         si e<obtenirElement(a) alors
            retourner estPresent(obtenirFilsGauche(a),e)
         sinon
            retourner estPresent(obtenirFilsDroit(a),e)
         finsi  '
      finsi
   finsi
fin
```

Nous avons vu en cours et en TD que l'on peut concevoir un ABR à l'aide du SDD `ArbreBinaire` :

Type ArbreBinaireRecherche = ArbreBinaire

Pour que la recherche d'un élément dans un arbre binaire de recherche soit efficace, il faut utiliser les algorithmes d'insertion et de suppression AVL, utilisant les algorithmes de rotation. Par exemple l'algorithme des simples et doubles rotations à droite sont :

```
procédure faireSimpleRotationADroite (E/S a : ABR)
   ⌊précondition(s)  non(estVide(a)) et non(estVide(obtenirFilsGauche(a)))
   Déclaration  temp : ABR
debut
   temp ← obtenirFilsGauche(a)
   fixerFilsGauche(a,obtenirFilsDroit(temp))
   fixerFilsDroit(temp,a)
   a ← temp
fin
procédure faireDoubleRotationADroite (E/S a : ABR)
   ⌊précondition(s)  non(estVide(a)) et non(estVide(obtenirFilsGauche(a)))
                     et non(estVide(obtenirFilsDroit(obtenirFilsGauche(a)))
   Déclaration  temp : ABR
debut
   temp ← obtenirFilsGauche(a)
   faireSimpleRotationAGauche(temp)
   fixerFilsGauche(a,temp)
   faireSimpleRotationADroite(a)
fin
```

L'algorithme de la procédure inserer est alors :

```
procédure inserer (E/S a : ABR, E e : Element)
   ⌊précondition(s)  temp : ABR
debut
   si estVide(a) alors
      a ← ajouterRacine(arbreBinaireRecherche(), arbreBinaireRecherche(), e)
   sinon
      si e≤obtenirElementRacine(a) alors
         temp ← obtenirFilsGauche(a)
```

```
            inserer(temp, e)
            fixerFilsGauche(a, temp)
        sinon
            temp ← obtenirFilsDroit(a)
            inserer(temp, e)
            fixerFilsDroit(a, temp)
        finsi
    finsi
    equilibrerSiNecessaire(a)
fin
```

Elle utilise la procédure de rééquilibrage suivante :

```
procédure equilibrerSiNecessaire (E/S a : ABR)
debut
    si hauteur(obtenirFilsGauche(a))> hauteur(obtenirFilsDroit(a))+1 alors
        si hauteur(obtenirFilsGauche(obtenirFilsGauche(a))) ≥ hauteur(obtenirFilsDroit(obtenirFilsGauche(a)))
        alors
            faireSimpleRotationDroite(a)
        sinon
            faireDoubleRotationDroite(a)
        finsi
    sinon
        si hauteur(obtenirFilsDroit(a))> hauteur(obtenirFilsGauche(a))+1 alors
            si hauteur(obtenirFilsGauche(obtenirFilsDroit(a))) ≤ hauteur(obtenirFilsDroit(obtenirFilsDroit(a)))
            alors
                faireSimpleRotationGauche(a)
            sinon
                faireDoubleRotationGauche(a)
            finsi
        finsi
    finsi
fin
```

La procédure de suppression est :

```
procédure supprimer (E/S a : ABR ; E e : Element)
    Déclaration  temp,tempG,tempD : ABR
debut
    si non estVide(a) alors
        si e < obtenirElement(a) alors
            temp ← obtenirFilsGauche(a)
            supprimer(temp,e)
            fixerFilsGauche(a,temp)
        sinon
            si e > obtenirElement(a) alors
                temp ← obtenirFilsDroit(a)
                supprimer(temp,e)
                fixerFilsDroit(a,temp)
            sinon
                si estVide(obtenirFilsGauche(a)) et estVide(obtenirFilsDroit(a)) alors
                    supprimerRacine(a,tempG,tempD)
                sinon
                    si estVide(obtenirFilsGauche(a)) ou estVide(obtenirFilsDroit(a)) alors
```

```
                supprimerRacine(a,tempG,tempD)
                si estVide(tempG) alors
                    a ← tempD
                sinon
                    a ← tempG
                finsi
            sinon
                e ← obtenirPlusGrand(obtenirFilsGauche(a))
                fixerElement(a,e)
                temp ← obtenirFilsGauche(a)
                supprimer(temp,e)
                fixerFilsGauche(a,temp)
            finsi
        finsi
    finsi
  finsi
  equilibrerSiNecessaire(a)
fin
```

Elle utilise la fonction qui permet d'obtenir l'élément le plus grand d'un arbre :

```
fonction obtenirPlusGrand (a : ABR) : Element
  ⌊précondition(s)  non estVide(a)
debut
  si non estVide(obtenirFilsDroit(a)) alors
    retourner obtenirPlusGrand(obtenirFilsDroit(a))
  sinon
    retourner obtenirElement(a)
  finsi
fin
```

Travail à réaliser

L'utilisation du `make` génère un test unitaire (`tests/testsAVL`). L'utilisation de la commande `doxygen` permet de générer la documentation HTML et LaTeX dans le répertoire `doc`.

Vous devez compléter le fichier `src/avl.c` (toutes les fonctions avec les commentaires `code à compléter` ou `code à remplacer`) de façon à ce que tous les tests unitaires passent :

```
$ tests/testsAVL


     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: Tests boite noire du type AVL_ArbreBinaireRecherche
  Test: AVL_arbreBinaireRecherche est vide ...passed
  Test: AVL_arbreBinaireRecherche obtenir element à la racine ...passed
  Test: AVL_inserer sans rotation ...passed
  Test: AVL_inserer avec simple rotation droite ...passed
  Test: AVL_inserer avec simple rotation gauche ...passed
  Test: AVL_inserer avec double rotation droite ...passed
  Test: AVL_inserer avec double rotation gauche ...passed
  Test: AVL_supprimer non présent ...passed
  Test: AVL_supprimer feuille ...passed
  Test: AVL_supprimer avec fils gauche vide ...passed
```

```
  Test: AVL_supprimer avec fils droit vide ...passed
  Test: AVL_supprimer avec fils gauche et droit sans rééquilibrage ...passed
  Test: AVL_supprimer avec fils gauche et droit avec rééquilibrage ...passed
Suite: Tests boite blanche du type AVL_ArbreBinaireRecherche
  Test: AVL_arbreBinaireRecherche est de hauteur -1 ...passed
  Test: Arbre non vide hauteur 1/2 ...passed
  Test: Arbre non vide hauteur 2/2 ...passed
  Test: AVL_obtenirElement ...passed
  Test: AVL_obtenirFilsGauche ...passed
  Test: AVL_obtenirFilsDroit ...passed
  Test: AVL_faireSimpleRotationADroite ...passed
  Test: AVL_faireSimpleRotationAGauche ...passed
  Test: AVL_faireDoubleRotationADroite ...passed
  Test: AVL_faireDoubleRotationAGauche ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests     23     23     23      0        0
             asserts     23     23     23      0      n/a

Elapsed time =    0.000 seconds
```