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 `C.tar.gz` qui se trouve dans `/opt/files` dans votre répertoire (commande `cp`) ;
— après l'avoir décompressée (commande `tar` avec les options `zxvf`), renommez le répertoire 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 `tar` avec les options `zcvf`), que vous nommerez comme le répertoire (avec le suffixe `.tar.gz`) ;
— 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 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 Une bibliothèque sur les arbres d'entiers

L'objectif de cet examen est de continuer le développement d'une bibliothèque (`libarbresDEntiers.a`) en C sur les arbres d'entiers. Pour l'instant, cette bibliothèque est composée des modules suivants :

— `ArbreBinaireDEntiers.c` qui propose des fonctions d'encapsulation pour gérer des arbres binaires de `int` en C. Ce module est déjà codé ;

— `AVLDEntiers.c` qui propose des fonctions de gestion d'un AVL d'entier. C'est ce module que vous devez compléter.

Rappels

Nous avons vu que la conception d'un AVL d'entiers est la suivante : **Type** AVL=ArbreBinaire

Avec ce choix de conception, nous avons étudié les algorithmes suivants :

La hauteur d'un arbre

fonction hauteur **(**a : AVL**) : Entier**
debut
    si estVide(a) **alors**
        retourner -1
    sinon
        retourner 1+maximum(hauteur(obtenirFilsGauche(a)),hauteur(obtenirFilsDroit(a)))
    finsi
fin

Les simples et doubles rotations

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

Vous devez être capable d'écrire les algorithmes des rotations à gauche...

La rééquilibrage

procédure equilibrerSiNecessaire **(E/S** a : AVL**)**
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
```

L'insertion

```
procédure inserer (E/S a : AVL, E e : Element)
    Déclaration   temp : AVL
debut
    si estVide(a) alors
        a ← ajouterRacine(arbreBinaireRecherche(),arbreBinaireRecherche(), e)
    sinon
        si e≤obtenirElement(a) alors
            temp ← obtenirFilsGauche(a)
            inserer(temp, e)
            fixerFilsGauche(a, temp)
        sinon
            temp ← obtenirFilsDroit(a)
            inserer(temp, e)
            fixerFilsDroit(a, temp)
        finsi
        equilibrerSiNecessaire(a)
    finsi
fin
```

La suppression

```
fonction plusGrand (a : AVL) : Element
    ⌊précondition(s)   non estVide(a)
debut
    si non estVide(obtenirFilsDroit(a)) alors
        retourner plusGrand(obtenirFilsDroit(a))
    sinon
        retourner obtenirElement(a)
    finsi
fin
procédure supprimer (E/S a : AVL ; E e : Element)
    Déclaration   nouveauSommet : Element
                  temp,tempG,tempD : AVL
debut
    si non estVide(a) alors
        si e < obtenirElement(a) alors
            temp ← obtenirFilsGauche(a)
            supprimer(temp,e)
            fixerFilsGauche(a,temp)
            equilibrerSiNecessaire(a)
        sinon
            si e > obtenirElement(a) alors
                temp ← obtenirFilsDroit(a)
                supprimer(temp,e)
                fixerFilsDroit(a,temp)
                equilibrerSiNecessaire(a)
            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
                        nouveauSommet ← plusGrand(obtenirFilsGauche(a))
                        fixerElement(a,nouveauSommet)
                        temp ← obtenirFilsGauche(a)
```

```
                    supprimer(temp,nouveauSommet)
                    fixerFilsGauche(a,temp)
                finsi
                equilibrerSiNecessaire(a)
            finsi
        finsi
    finsi
finsi
fin
```

Travail à réaliser

L'utilisation du `make` génère, en plus de la bibliothèque (`lib/libarbresDEntiers.a`), un test unitaire (`test/avlTU`)

Vous devez donc compléter le fichier `AVLDEntiers.c` (toutes les fonctions avec les commentaires `code à completer` ou `code à remplacer`) de façon à ce que tous les tests unitaires, des deux séries de tests, passent :

```
$ test/avlTU


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


Suite: Test type AVL (fonction privees)
  Test: hauteur arbre vide ...passed
  Test: hauteur non vide ...passed
  Test: simple rotation a droite ...passed
  Test: simple rotation a gauche ...passed
  Test: double rotation a droite ...passed
  Test: double rotation a gauche ...passed
  Test: equilibre si necessaire par simple rotation a droite ...passed
  Test: equilibre si necessaire par simple rotation a gauche ...passed
  Test: equilibre si necessaire par double rotation a droite ...passed
  Test: equilibre si necessaire par double rotation a gauche ...passed
  Test: plus grand ...passed
Suite: Test type AVL (fonction publiques)
  Test: AVL estVide d'un arbre vide ...passed
  Test: AVL obtenirFilsGauche ...passed
  Test: AVL obtenirFilsDroit ...passed
  Test: AVL obtenirFilsEntier ...passed
  Test: AVL inserer dans arbre vide ...passed
  Test: AVL inserer sans reequilibrage ...passed
  Test: AVL inserer avec reequilibrage ...passed
  Test: AVL supprimer feuille ...passed
  Test: AVL supprimer un fils vide ...passed
  Test: AVL supprimer aucun fils vide ...passed
  Test: AVL supprimer avec reequlibrage ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests     22     22     22      0        0
             asserts     22     22     22      0      n/a

Elapsed time =    0.000 seconds
```