ASI3 Mardi 17 Janvier 2017

Algorithmique et Bases de la programmation

Durée : *1h55*
Documents autorisés : **AUCUN**

L'objectif de cet examen pratique est de développer une liste d'entiers vu en cours et en TD.

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 : 'EXAM-ALGO-' (les tirets sont ceux du signe moins) suivi de votre nom et 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 EXAM-ALGO-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 qu'il n'y a aucun accent dans les commentaires que vous auriez pu ajouter ;
— 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 préfixe `.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 Liste D'Entiers : LDE

1.1 Analyse

Pour rappel le TAD `Liste` est :

Nom: Liste
Paramètre: Element
Utilise: **Booleen**, **NaturelNonNul**, **Naturel**
Opérations:

liste:	$\rightarrow$ Liste
estVide:	Liste $\rightarrow$ **Booleen**
inserer:	Liste $\times$ **NaturelNonNul** $\times$ Element $\nrightarrow$ Liste
supprimer:	Liste $\times$ **NaturelNonNul** $\nrightarrow$ Liste
obtenirIemeElement:	Liste $\times$ **NaturelNonNul** $\nrightarrow$ Element
longueur:	Liste $\rightarrow$ **Naturel**

Axiomes:
- estVide(liste())
- non estVide(inserer(l,i,e))
- suprimer(inserer(l,i,e),i)=l
- obtenirElement(inserer(inserer(l,i,$e1$),i,$e2$),$i+1$)=$e1$
- longueur(liste())=0
- longueur(inserer(l,i,e))=1+longueur(l)

Préconditions:

inserer(l,i,e):	$i \leq longueur(l)+1$
supprimer(l,i):	$i \leq longueur(l)$
obtenirElement(l,i):	$i \leq longueur(l)$

1.2 Conception

1.2.1 Conception préliminaire

Le TAD `Liste` permet d'identifier les fonctions et procédures suivantes pour le type `ListeDEntiers` :

— **fonction** listeOrdonneeDEntiers **() :** ListeDEntiers
— **fonction** estVide (uneListe : ListeDEntiers) **: Booleen**
— **procédure** inserer **(E/S** uneListe : ListeDEntiers, **E** pos : **NaturelNonNul**, element : **Entier)**

  ⌊**précondition(s)** $position \leq longueur(uneListe)+1$

— **procédure** supprimer **(E/S** uneListe : ListeDEntiers, **E** pos : naturelNonNul)

  ⌊**précondition(s)** $position \leq longueur(uneListe)$

— **fonction** obtenirIemeElement (uneListe : ListeDEntiers, i : **Naturel) : Entier**

  ⌊**précondition(s)** $position \leq longueur(uneListe)$

— **fonction** longueur (uneListe : ListeDEntiers) **: Naturel**

À ces fonctions et procédures, nous pouvons ajouter la procédure qui supprime tous les éléments d'une liste ordonnée :

— **procédure** supprimerLDE **(E/S** uneListe : ListeDEntiers**)**

1.2.2 Conception détaillée

Nous avons vu en cours que l'on peut concevoir une liste d'entiers à l'aide de la SDD `ListeChainee` :

Type ListeDEntiers = **Structure**
  nb : **NaturelNonNul**
  entiers : ListeChainee<Entier>
finstructure

Ainsi la plupart des opérations du type `ListeDEntiers` utilisent des fonctions ou procédures manipulant le champ `entiers`. Par exemple, l'opération `inserer` peut utiliser la procédure suivante qui insère à une certaine position un entier dans une liste chaînée d'entiers :

procédure insererDansListeChainee **(E/S** l : ListeChainee<**Entier**> ; **E** pos : **NaturelNonNul**, element : **Entier)**
  Déclaration temp : ListeChainee<**Entier**>
debut
  si pos=1 **alors**
    ajouter(l,e)
  sinon
    temp ← obtenirListeSuivante(l)
    insererDansListeChainee(temp,pos-1,element)
    fixerListeSuivante(l,temp)
  finsi
fin

1.3 Développement

Nous décidons d'implanter le type `ListeDEntiers` en C en utilisant le type `ListeChaineeDEntiers`. Pour cela, nous disposons des fichiers suivants :

— `include/ListeChaineeDEntiers.h` qui déclare les fonctions permettant d'utiliser une liste chaînée d'entiers ;
— `include/ListeDEntiers.h` qui déclare les fonctions permettant d'utiliser une liste d'entiers ;
— `src/ListeChaineeDEntiers.c` qui définit les fonctions déclarées dans `include/ListeChaineeDEntiers.h`
— `src/ListeDEntiers.c` qui définit les fonctions déclarées dans `include/ListeDEntiers.h`
— `src/ldeTU.c` le programme des tests unitaires des fonctions déclarées dans `include/ListeDEntiers.h` ;

L'exécution du `make` génère le programme des tests unitaires `test/ldeTU`.

2 Travail à réaliser

Complétez le fichier `ListeDEntiers.c` de façon à ce que le maximum de tests unitaires fonctionnent (`test/ldeTU`).