Algorithmique et Bases de la programmation

Durée : *1h45*
Documents autorisés : **AUCUN**

L'objectif de cet examen pratique est de développer une librairie C proposant le type `EnsembleDEntiers`.

Informations concernant le déroulement de l'examen

Avant de commencer l'examen, connectez vous à la machine et suivez les instruction suivantes :
- copiez l'archive `C.tgz` qui se trouve dans `/tmp` 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' suivi de votre nom et prenom, sans espace et en minuscule sauf pour le première lettre du prénom et du nom (si je devais passer l'examen, je nommerai le répertoire `mv C EXAM-ALGO-DelestreNicolas`)
- à la fin de l'examen :
  - vous créerez une archive de votre répertoire (commande `tar` avec les options `zcvf`), que vous nommerez comme le répertoire (avec le préfixe `.tgz`) ;
  - vous déposerez l'archive sur moodle.

Quelques conseils pour réussir votre examen
- après le développement de chaque fonction, 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, divisera par deux votre note finale ;
- la note finale sera fonction :
  - du nombre de tests unitaires qui seront valides (exécution du programme `bin/testEDE_CUnit`) ;
  - de la quantité et de la lisibilité du code (indentation et nom des identifiants).

1 Rappels

1.1 Analyse

Nous avons vu en cours que le TAD collection Ensemble est défini de la manière suivante :

Nom:	Ensemble	
Paramètre:	Element	
Utilise:	**Booleen**,**Naturel**	
Opérations:	ensemble:	→ Ensemble
	ajouter:	Ensemble × Element → Ensemble
	retirer:	Ensemble × Element → Ensemble
	estPresent:	Ensemble × Element → **Booleen**
	cardinalite:	Ensemble → **Naturel**

union: Ensemble × Ensemble → Ensemble
intersection: Ensemble × Ensemble → Ensemble
soustraction: Ensemble × Ensemble → Ensemble

Axiomes:
- ajouter(ajouter(s,e),e)=ajouter(s,e)
- retirer(ajouter(s,e),e)=s
- estPresent(ajouter(s,e),e)
- ¬estPresent(retirer(s,e),e)
- cardinalite(ensemble())=0
- cardinalite(ajouter(s,e))=1+cardinalite(s) ∧ ¬estPresent(s,e)
- cardinalite(ajouter(s,e))=cardinalite(s) ∧ estPresent(s,e)

...

1.2 Conception préliminaire

Nous avons aussi vu en cours que dans la paradigme de la programmation structurée, les opérations de ce TAD sont traduites en fonctions et procédures :

- **fonction** ensemble () **:** Ensemble
- **procédure** ajouter (**E/S** unEnsemble : Ensemble,**E** element : Element)
- **procédure** retirer (**E/S** unEnsemble : Ensemble,**E** element : Element)
- **fonction** estPrésent (unEnsemble : Ensemble, element : Element) **: Booleen**
- **fonction** cardinalite (unEnsemble : Ensemble) **: Naturel**
- **fonction** union (ens1,ens2 : Ensemble) **:** Ensemble
- **fonction** intersection (ens1,ens2 : Ensemble) **:** Ensemble
- **fonction** soustraction (ens1,ens2 : Ensemble) **:** Ensemble

Nous pouvons aussi ajouter la procédure `vider` :

- **procédure** vider (**E/S** unEnsemble : Ensemble)

2 Conception détaillée

On se propose de représenter le TAD Ensemble à l'aide d'un structure utilisant une liste chaînée pour stocker les entiers et un naturel pour le nombre d'entiers présents dans l'ensemble (afin de proposer la cardinalité de l'ensemble à l'aide d'une opération en $O(1)$.

Type Ensemble = **Structure**
lesElements : ListeChainee
nbElements : **Naturel**
finstructure

Pour rappel, les fonctions et procédures du TAD `ListeChainee` sont :

- **fonction** listeChainee () **:** ListeChainee
- **fonction** estVide (l : ListeChainee) **: Booleen**
- **procédure** ajouter (**E/S** l : ListeChainee,**E** e : Element)
- **fonction** obtenirElement (l :ListeChainee) **:** Element
  ⌊**précondition(s)** non estVide(l)
- **procédure** fixerListeSuivante (**E/S** l : ListeChainee,**E** l' : ListeChainee)
  ⌊**précondition(s)** non estVide(l)
- **fonction** obtenirListeSuivante (l : ListeChainee) **:** ListeChainee
  ⌊**précondition(s)** non estVide(l)
- **procédure** supprimerTete (**E/S** l : ListeChainee)

⌊**précondition(s)** non estVide(l)

Voici les algorithmes de quelques fonctions et procédures du type `Ensemble` :

fonction ensemble **() :** Ensemble
  Déclaration resultat : Ensemble
debut
  resultat.lesElements ← listeChainee()
  resultat.nbElements ← 0
  retourner resultat
fin
procédure ajouter **(E/S** unEnsemble : Ensemble,**E** element : Element**)**
debut
  si non estPresent(unEnsemble,element) **alors**
    ajouter(unEnsemble.lesElements,element)
    unEnsemble.nbElements ← unEnsemble.nbElements+1
  finsi
fin
procédure supprimerElement **(E/S** liste : ListeChainee,**E** element : Element**)**
  Déclaration temp : ListeChainee
debut
  si non estVide(liste) **alors**
    si obtenirElement(liste)=element **alors**
      supprimerTete(l)
    sinon
      temp ← obtenirListeSuivante(liste)
      supprimerElement(temp,element)
      fixerListeSuivante(l,temp)
    finsi
  finsi
fin
procédure retirer **(E/S** unEnsemble : Ensemble,**E** element : Element**)**
debut
  si estPresent(unEnsemble,element) **alors**
    supprimerElement(unEnsemble.lesElements,element)
    unEnsemble.nbElements ← unEnsemble.nbElements-1
  finsi
fin
fonction estPresent **(**unEnsemble : Ensemble, element : Element**) : Booleen**
  Déclaration l : ListeChainee
    trouve : **Booleen**
debut
  l ← unEnsemble.lesElements
  trouve ← FAUX
  tant que non estVide(l) et non trouve **faire**
    si obtenirElement(l)=element **alors**
      trouve ← VRAI
    sinon
      l ← obtenirListeSuivante(l)
    finsi
  fintantque

retourner trouve
fin
fonction cardinalite (unEnsemble : Ensemble) **: Naturel**
debut
retourner unEnsemble.nbElements
fin
procédure ajouterElements (**E** source : Ensemble,**E/S** destination : Ensemble)
Déclaration l : ListeChainee
debut
l ← source.lesElements
tant que non estVide(l) **faire**
ajouter(destination,obtenirElement(l))
l ← obtenirListeSuivante(l)
fintantque
fin
fonction union (ens1,ens2 : Ensemble) **:** Ensemble
Déclaration resultat : Ensemble
debut
resultat ← ens1
ajouterElements(ens2,resultat)
retourner resultat
fin

3 Développement

Le projet C contient les fichiers suivants :
- `include/ListeChaineeDEntiers.h` : partie publique du type `ListeChaineeDEntiers`
- `include/EnsembleDEntiers.h` : partie publique du type `EnsembleDEntiers`
- `src/ListeChaineeDEntiers.c` : partie privée du type `ListeChaineeDEntiers`
- `src/EnsembleDEntiers.c` : partie privée du type `EnsembleDEntiers`
- `src/testLCDE_CUnit.c` : code source du test unitaire du type `ListeChaineeDEntiers`
- `src/testEDE_CUnit.c` : code source du test unitaire du type `EnsembleDEntiers`

L'exécution du `make` produit :
- `lib/libListeChaineeDEntiers.a` : la librairie proposant le type `ListeChaineeDEntiers`
- `lib/libEnsembleDEntiers.a` : la librairie proposant le type `EnsembleDEntiers`
- `bin/testLCDE_CUnit` : test unitaire du type `ListeChaineeDEntiers`
- `bin/testEDE_CUnit` : test unitaire du type `EnsembleDEntiers`

Travail à réaliser Complétez le fichier `src/EnsembleDEntiers.c`.