ASI3 19 Novembre 2019

Algorithmique avancée et programmation C

Durée : *1h30*
Documents autorisés : **AUCUN**

Remarques :
— Vos réponses aux exercices 2 et 3 seront sur des copies doubles différentes
— Veuillez lire attentivement les questions avant de répondre ;
— Rendez une copie propre ;
— N'utilisez pas de crayon à papier sur votre copie ;
— Le pourcentage indiqué pour chaque exercice est le temps approximatif que vous devez passer sur l'exercice.

1 QCM (15%)

Répondez au qcm ci joint (à rendre avec votre copie). La note de chaque question peut varier de -1 à $+1$.

Soit B le nombre de bonnes réponses à une question et soit M le nombre de mauvaises réponses à cette même question ($B + M$ est égal au nombre de réponses à la question). Chaque bonne réponse cochée rapporte $1/B$ points et chaque mauvaise réponse $-1/M$ points.

Solution proposée :

Compétences évaluées

— DEV001 : Savoir compiler et linker un programme C (options de base de gcc)
— DEV008 : Savoir traduire des passages de paramètre algorithme en passage de paramètre C
— DEV009 : Maîtriser les pointeurs, tableaux et chaînes de caractères

2 Multiensemble (50%)

D'après Wikipédia, un « multiensemble [...] est une sorte d'ensemble dans lequel chaque élément peut apparaître plusieurs fois. [...] On nomme multiplicité d'un élément donné le nombre de fois où il apparaît.

Formellement, un multiensemble est un couple (A, m) où A est un ensemble appelé support et m une fonction de A dans l'ensemble des entiers naturels, appelée multiplicité. Dans le multiensemble (A, m), l'élément x apparaît $m(x)$ fois. »

2.1 Analyse

Soit le TAD `MultiEnsemble` possédant les opérations suivantes :
— obtenir un multiensemble vide ;
— savoir si un multiensemble est vide ;
— ajouter un élément ;
— savoir si un élément est présent ;
— obtenir la multiplicité d'un élément (la multiplicité d'un élément non présent est de 0) ;
— supprimer un élément ayant une multiplicité strictement positive. Cette opération décrémente la multiplicité de l'élément. L'élément n'est alors plus considéré comme présent si sa multiplicité est de 0 ;
— obtenir le support.

Formaliser (axiomes compris) le TAD `MultiEnsemble`.

Solution proposée :

Compétences évaluées

- AN201 : Savoir identifier les dépendances d'un TAD
- AN202 : Savoir définir des TAD générique
- AN203 : Savoir si une opération identifiée fait partie du TAD à spécifier
- AN204 : Savoir formaliser des opérations d'un TAD
- AN205 : Savoir formaliser les préconditions d'une opération d'un TAD
- AN206 : Savoir formaliser des axiomes ou savoir définir la sémantique d'une opération d'un TAD

Nom: MultiEnsemble
Paramètre: Element
Utilise: **Booleen**, **Naturel**, Ensemble
Opérations: multiensemble: → MultiEnsemble
estVide: MultiEnsemble → **Booleen**
ajouter: MultiEnsemble × Element → MultiEnsemble
estPresent: MultiEnsemble × Element → **Booleen**
multiplicite: MultiEnsemble × Element → **Naturel**
supprimer: MultiEnsemble × Element ↛ **Naturel**
support: MultiEnsemble → Ensemble<Element>
Préconditions: supprimer(m,e): estPresent(m,e)
Axiomes:
- estVide(multiensemble())
- non estVide(ajouter(m,e))
- estPresent(ajouter(m,e),e)
- multiplicite(ajouter(m,e),e)=multiplicite(m,e)+1
- multiplicite(supprimer(m,e),e)=multiplicite(m,e)-1
- non estPresent(m,e) et multiplicite(m,e)=0
- estVide(m) et estVide(support(m))
- estPresent(m,e) et estPresent(support(m),e)
- non estPresent(m,e) et non estPresent(support(m),e)

2.2 Conception préliminaire

Donnez les signatures des fonctions et procédures du type `MultiEnsemble`.

Solution proposée :

Compétences évaluées

- CP003 : Savoir choisir entre une fonction et une procédure
- CP004 : Savoir concevoir une signature (préconditions incluses)
- CP005 : Savoir choisir un passage de paramètre (E, S, E/S)

- **fonction** multiensemble **() :** MultiEnsemble
- **fonction** estVide (m : MultiEnsemble**) : Booleen**
- **procédure** ajouter **(E/S** m : MultiEnsemble,**E** e : Element**)**
- **fonction** estPresent (m : MultiEnsemble, e : Element**) : Booleen**
- **fonction** multiplicite (m : MultiEnsemble, e : Element**) : Naturel**
- **procédure** supprimer **(E/S** m : MultiEnsemble,**E** e : Element**)**
  ⌊**précondition(s)** estPresent(m,e)
- **fonction** support (m : MultiEnsemble**) :** Ensemble<Element>

2.3 Conception détaillée

Proposez une conception détaillée pour le type `MultiEnsemble` (uniquement le type, on ne vous demande pas les algorithmes de ses opérations).

Solution proposée :

Compétences évaluées

— CD901 : Savoir concevoir un type de données adapté à la situation en terme d'espace mémoire et d'efficacité

— **Type** MultiEnsemble **=** Dictionnaire<Element,**Naturel**>

2.4 Utilisation : l'intersection

D'après Wikipédia anglais, l'intersection de deux multiensembles A et B est un multiensemble C tel que le support de C est l'intersection des supports de A et de B et tel que la multiplicté de ses éléments est le min des multiplicités de ces éléments dans A et B.

1. Proposez l'analyse descendante de l'intersection de deux multiensembles en faisant apparaître toutes les opérations que vous utilisez (du TAD `MultiEnsemble` ou autre).
2. Donnez l'algorithme correspondant.

Solution proposée :

1. L'analyse descendante :

   Compétences évaluées

   — AN101 : Savoir identifier les entrées et sorties d'un problème
   — AN102 : Savoir décomposer logiquement un problème

   intersection : MultiEnsemble × MultiEnsemble → MultiEnsemble
   support : MultiEnsemble → Ensemble<Element>
   intersection : Ensemble × Ensemble → Ensemble
   multiensemble : → MultiEnsemble
   multiplicite MultiEnsemble × Element → **Naturel**
   min : **Naturel** × **Naturel** → **Naturel**
   ajouterNFois : MultiEnsemble × Element × **NaturelNonNul** → MultiEnsemble
   ajouter : MultiEnsemble × Element → MultiEnsemble

2. L'algorithme :

   Compétences évaluées

   — CD001 : Savoir dissocier les deux rôles du développeur : concepteur et utilisateur
   — CD002 : En tant qu'utilisateur, savoir respecter une signature
   — CD004 : Savoir écrire des algos avec le pseudo code utilisé à l'INSA
   — CD005 : Savoir écrire un pseudo code lisible (indentation, identifiant significatif)
   — CD006 : Savoir choisir la bonne itération
   — CD009 : Savoir écrire un algorithme qui résout le problème

   fonction interscetion **(**m1, m2 : MultiEnsemble**) :** MultiEnsemble
   Déclaration res : MultiEnsemble
   debut
   res ← multiensemble()
   pour chaque e **de** intersection(support(m1), support(m2))
   ajouterNFois(res, element, min(multiplicite(m1,e), multiplicite(m2,e)))
   finpour
   retourner res
   fin

3 L'indice de la dernière occurrence d'un entier (35%)

Écrire une fonction récursive, `indiceMax`, qui détermine, en $O(log(n))$, le plus grand indice d'un entier présent dans un tableau t de nb entiers, trié dans l'ordre croissant. Cet entier peut bien entendu être présent plusieurs fois dans le tableau t.

Solution proposée :

Compétences évaluées

- CP004 : Savoir concevoir une signature (préconditions incluses)
- CD004 : Savoir écrire des algos avec le pseudo code utilisé à l'INSA
- CD005 : Savoir écrire un pseudo code lisible (indentation, identifiant significatif)
- CD009 : Savoir écrire un algorithme qui résout le problème
- CD104 : Savoir écrire un algorithme d'une complexité donnée
- CD201 : Savoir identifier et résoudre le problème des cas non récursifs
- CD202 : Savoir identifier et résoudre le problème des cas récursifs

fonction indiceMax (t : **Tableau**[1..MAX] **d'Entier**, e : **Entier**, nb : **NaturelNonNul) : NaturelNonNul**

⌊**précondition(s)** nb≤MAX

debut
  retourner indiceMaxR(t,e,1,nb)
fin
fonction indiceMaxR (t : **Tableau**[1..MAX] **d'Entier**, e : **Entier**, d,f : **NaturelNonNul) : NaturelNonNul**

⌊**précondition(s)** d≤f et f≤MAX

```
debut
  si d=f alors
    retourner d
  sinon
    si d=f+1 alors
      si t[d]=e alors
        retourner d
      sinon
        retourner f
      finsi
    sinon
      m ← (d+f) div 2
      si t[m]=e alors
        retourner indiceMaxR(t,e,m,f)
      sinon
        si t[m]>e alors
          retourner indiceMaxR(t,e,d,m-1)
        sinon
          retourner indiceMaxR(t,e,m+1,f)
        finsi
      finsi
    finsi
  finsi
fin
```

Annexe

Pour rappel le TAD Ensemble vu en cours est :

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)
- non estPresent(retirer(s,e),e)
- cardinalite(ensemble())=0
- cardinalite(ajouter(s,e))=1+cardinalite(s) et non estPresent(s,e)
- cardinalite(ajouter(s,e))=cardinalite(s) et estPresent(s,e)

...