ITI3 3 Novembre 2025

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 ;
— Pour chaque exercice, il est indiqué, entre parenthèses, le nombre d'Attendus d'Apprentissage Disciplinaires (AAD) évalués ;
— Vous trouverez en annexe les signatures de fonctions et procédures sur des collections, SDD nécessaires à certains exercices.

1 QCM (3 AAD)

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 :

Attendus d'apprentissages disciplinaires évalués

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

2 L'indice de la dernière occurrence d'un entier (8 AAD)

Écrire une fonction récursive, `indiceMax`, qui détermine, en $O(log_2(n))$, le plus grand indice d'un entier e, obligatoirement 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 :

Attendus d'apprentissages disciplinaires évalués

— CP004 : Concevoir une signature (préconditions incluses)
— CD004 : Écrire des algos avec le pseudo code utilisé à l'INSA
— CD005 : Écrire un pseudo code lisible (indentation, identifiant significatif)
— CD009 : Écrire un algorithme qui résout le problème
— CD104 : Écrire un algorithme d'une complexité donnée
— CD201 : Identifier et résoudre le problème des cas non récursifs(pour g=d)
— CD201 : Identifier et résoudre le problème des cas non récursifs(pour d=g+1)
— CD202 : Identifier et résoudre le problème des cas récursifs
— CD204 : Penser à avoir une fonction/procédure récursive privée pour respecter une signature générale

fonction indiceMax (t : **Tableau**[1..MAX] **d'Entier**, e : **Entier**, nb : **NaturelNonNul**) **: NaturelNonNul**
⌊**précondition(s)** nb≤MAX
$\exists i \in 1..nb$ tel que $t[i] = e$
debut

```
    retourner indiceMaxR(t,e,1,nb)
fin
fonction indiceMaxR (t : Tableau[1..MAX] d'Entier, e : Entier, g,d : NaturelNonNul) : NaturelNonNul
    ⌊précondition(s)   g≤d et d≤MAX
debut
    si g=d alors
        retourner g
    sinon
        si d=g+1 alors
            si t[d]=e alors
                retourner d
            sinon
                retourner g
            finsi
        sinon
            m ← (g+d) div 2
            si t[m]=e alors
                retourner indiceMaxR(t,e,m,d)
            sinon
                si t[m]>e alors
                    retourner indiceMaxR(t,e,g,m-1)
                sinon
                    retourner indiceMaxR(t,e,m+1,d)
                finsi
            finsi
        finsi
    finsi
fin
```

3 Multiensemble (10 AAD)

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. »

3.1 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 :

Attendus d'apprentissages disciplinaires évalués

- AN201 : Identifier les dépendances d'un TAD
- AN202 : Définir des TAD génériques
- AN203 : Savoir si une opération identifiée fait partie du TAD à spécifier
- AN204 : Formaliser des opérations d'un TAD
- AN205 : Formaliser les préconditions d'une opération d'un TAD
- AN206 : 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 ↛ MultiEnsemble
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) ⇒ cardinalite(support(m))=0
- estPresent(m,e) ⇒ estPresent(support(m),e)
- non estPresent(m,e) ⇒ non estPresent(support(m),e)

3.2 Conception préliminaire

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

Solution proposée :

Attendus d'apprentissages disciplinaires évalués

- CP003 : Choisir entre une fonction et une procédure
- CP004 : Concevoir une signature (préconditions incluses)
- CP005 : 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>

3.3 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 :

Attendus d'apprentissages disciplinaires évalués

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

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

3.4 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 multiplicité 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) en respectant le nommage de Wikipédia.
2. Donnez l'algorithme correspondant.

Solution proposée :

1. L'analyse descendante :

   Attendus d'apprentissages disciplinaires évalués

   — AN101 : Identifier les entrées et sorties d'un problème
   — AN102 : 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 :

   Attendus d'apprentissages disciplinaires évalués

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

   fonction intersection **(**m1, m2 : MultiEnsemble**) :** MultiEnsemble

   Déclaration res : MultiEnsemble
   e : Element

   debut
   res ← multiensemble()
   pour chaque e **de** Ensemble.intersection(support(m1), support(m2))
   ajouterNFois(res, e, min(multiplicite(m1,e), multiplicite(m2,e)))
   finpour
   retourner res
   fin

Annexe

Pour rappel le TAD Ensemble vu en cours est :

Nom: Ensemble

Paramètre: Element

Utilise: **Booleen**,**Naturel**

Opérations:

ensemble:		$\rightarrow$ Ensemble
ajouter:	Ensemble $\times$ Element	$\rightarrow$ Ensemble
retirer:	Ensemble $\times$ Element	$\rightarrow$ Ensemble
estPresent:	Ensemble $\times$ Element	$\rightarrow$ **Booleen**
cardinalite:	Ensemble	$\rightarrow$ **Naturel**
union:	Ensemble $\times$ Ensemble	$\rightarrow$ Ensemble
intersection:	Ensemble $\times$ Ensemble	$\rightarrow$ Ensemble
soustraction:	Ensemble $\times$ Ensemble	$\rightarrow$ 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)

  ...