ASI3 21 Novembre 2017

Algorithmique avancée et programmation C

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

Remarques :
- Veuillez lire attentivement les questions avant de répondre.
- Le barème donné est un barème indicatif qui pourra évoluer lors de la correction.
- Rendez une copie propre.
- N'utilisez pas de crayon à papier sur votre copie.

1 QCM (5 points)

Répondez au qcm ci joint (à rendre avec votre copie). La note de chaque question peut varier de -1 à $+1$. La note à l'exercice (0 au minimum) est la moyenne des notes de chaque question multipliée par le nombre de points de l'exercice.

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 égale au nombre de réponses à la question). Chaque bonne réponse cochée rapporte $1/B$ points et chaque mauvaise réponse $-1/M$ points.

2 Amélioration du tri à bulles (4 points)

Pour rappel le principe du tri à bulles est de (re)parcourir entièrement le tableau, de nb éléments significatifs, jusqu'à ce que celui ci soit trié. Durant ces parcours si deux éléments consécutifs ne sont pas en ordre, on les échange et on note que le tableau n'était pas trié.

Une des améliorations consiste à limiter le parcours du tableau en prenant comme borne maximale la position du dernier échange dans le précédent parcours (bien entendu lors du premier parcours cette borne est fixée à nb).

Proposez l'algorithme de ce tri à bulles amélioré en supposant que l'on veuille trier en ordre croissant un tableau (de 1 à MAX entiers) avec nb entiers significatifs.

Solution proposée :

```
procédure triABullesAmeliore (E/S t :Tableau[1..MAX] d'Entier,E nb :NaturelNonNul)
   Déclaration  estTrie : Booleen
                i,n,indiceDernierEchange : Naturel
debut
   n ← nb
   tant que n ≠ 0 faire
      indiceDernierEchange ← 0
      pour i ←1 à n-1 faire
         si t[i]>t[i+1] alors
            echanger(t[i],t[i+1])
            indiceDernierEchange ← i
         finsi
      finpour
      n ← indiceDernierEchange
   fintantque
fin
```

3 Traitement d'image (5 points)

3.1 Le Type Abstrait de Données ImageNG

Une image bitmap en 256 niveaux de gris est constituée de pixels. À chaque pixel sont associés des coordonnées (x, y) et un naturel (entre 0 et 255), déterminant son niveau de gris (0 pour noir ; 255 pour blanc). Le

pixel de coordonnées $(0, 0)$ se trouve en haut à gauche de l'image et celui de coordonnées $(largeur-1, hauteur-1)$ se trouve en bas à droite. Une image peut être utilisée à l'aide du TAD `ImageNG` (pour Image en Niveau de Gris) suivant (sans les parties axiomes et préconditions) :

Nom:	ImageNG	
Utilise:	**Naturel**	
Opérations:	imageNG:	**Naturel** × **Naturel** × 0..255 → ImageNG
	largeur:	ImageNG → **Naturel**
	hauteur:	ImageNG → **Naturel**
	obtenirNG:	ImageNG × **Naturel** × **Naturel** ⇸ 0..255
	fixerNG:	ImageNG × **Naturel** × **Naturel** × 0..255 ⇸ ImageNG
Sémantiques:	imageNG(l,h,ng):	permet de créer une image en niveau de gris de largeur l, de hauteur h avec comme couleur de fond le niveau de gris ng
	largeur(i):	permet d'obtenir la largeur d'une image i
	hauteur(i):	permet d'obtenir la hauteur d'une image i
	obtenirNG(i,x,y):	permet d'obtenir le niveau de gris d'un pixel (x, y) d'une image i
	fixerNG(i,x,y,ng):	permet de fixer le niveau de gris ng à un pixel (x, y) d'une image i

Donnez les parties préconditions et axiomes de ce TAD.

Solution proposée :

Préconditions: obtenirNG(i,x,y): x<largeur(i) et y<hauteur(i)
fixerNG(i,x,y,ng): x<largeur(i) et y<hauteur(i)

Axiomes:
- largeur(imageNG(l,h,ng))=l
- hauteur(imageNG(l,h,ng))=h
- obtenirNG(imageNG(l,h,ng),x,y)=ng
- obtenirNG(fixerNG(i,x,y,ng),x,y)=ng

3.2 Filtre moyenneur

En traitement d'image, un filtre est une fonction qui calcule une image destination à partir d'une image source. Chaque pixel de l'image destination est fonction des niveaux de gris des pixels de l'image source et optionnellement des niveaux de gris des pixels de l'image destination déjà calculées.

Le principe du filtre moyenneur est que le niveau de gris de chaque pixel (x, y) de l'image destination est égale à la moyenne des niveaux de gris des pixels appartenant à une fenêtre carrée centrée en (x, y) de l'image source. La longueur des côtés de cette fenêtre carrée est obligatoirement impaire. L'image destination, qui a la même taille que l'image source, semble alors floue.

La figure 1 présente l'utilisation d'un filtre moyenneur avec une fenêtre de taille 5 (exemple issue de `http://perso.telecom-paristech.fr/~maitre/BETI//filtres_lin_nlin/filtres.html`).

(a) Image source (b) Image destination

FIGURE 1 – Utilisation d'un filtre moyenneur de taille 5

Pour calculer un filtre moyenneur, il faut donc être capable d'identifier les bornes (coin haut gauche et bas droit) des pixels de la fenêtre pour une coordonnée donnée. Il faut aussi calculer la moyenne des niveaux de gris des pixels à l'intérieur de cette fenêtre. Et pour calculer cette moyenne, il faut être capable de calculer la somme des niveaux de gris des pixels de cette fenêtre.

Donnez l'analyse descendante du filtre moyenneur à partir de la description ci dessus.

Solution proposée :

L'analyse descendante :

— filtreMoyenneur : ImageNG × **NaturelNonNul** ⇸ ImageNG
  — bornesFenetre : **Naturel** × **Naturel** × **NaturelNonNul** ⇸ **Naturel** × **Naturel** × **Naturel** × **Naturel**
    — min : **Naturel** × **Naturel** → **Naturel**
    — max : **Naturel** × **Naturel** → **Naturel**
  — moyenneFenetree : ImageNG × **Naturel** × **Naturel** × **Naturel** × **Naturel** → 0..255
    — sommeFenetree ImageNG × **Naturel** × **Naturel** × **Naturel** × **Naturel** → **Naturel**

4 Expression arithmétique (7 points)

Une expression arithmétique est soit :

— un nombre réel ;

— une opération binaire composée d'un opérateur et de deux opérandes qui sont des expressions arithmétiques.

On suppose posséder le type suivant :

Type Operateur = {ADDITION, SOUSTRACTION, MULTIPLICATION, DIVISION}

1. Proposez le TAD `ExpressionArithmetique` sans donner les axiomes ou les sémantiques
   Solution proposée :

Nom:	ExpressionArithmetique	
Utilise:	**Reel**, **Booleen**, Operateur	
Opérations:	aPartirDUnReel:	**Reel** → ExpressionArithmetique
	aPartirDUneOperation:	Operateur × ExpressionArithmetique × ExpressionArithmetique → ExpressionArithmetique
	estUneOperation:	ExpressionArithmetique → **Booleen**
	valeur:	ExpressionArithmetique ⇸ **Reel**
	operateur:	ExpressionArithmetique ⇸ Operateur
	operandeG:	ExpressionArithmetique ⇸ ExpressionArithmetique
	operandeD:	ExpressionArithmetique ⇸ ExpressionArithmetique
Préconditions:	valeur(e):	non estUneOperation(e)
	operateur(e):	estUneOperation(e)
	operandeG(e):	estUneOperation(e)
	operandeD(e):	estUneOperation(e)

2. Donnez les signatures des fonctions et procédures de ce TAD
   Solution proposée :
   — **fonction** aPartirDUnReel (v : **Reel**) **:** ExpressionArithmetique
   — **fonction** aPartirDUneOperation (op : Operateur, opG,opD : ExpressionArithmetique) **:** ExpressionArithmetique
   — **fonction** estUneOperation (e : ExpressionArithmetique) **: Booleen**
   — **fonction** valeur (e : ExpressionArithmetique) **: Reel**
     ⌊**précondition(s)** non estUneOperation(e)
   — **fonction** operateur (e : ExpressionArithmetique) **:** Operateur
     ⌊**précondition(s)** estUneOperation(e)
   — **fonction** operandeG (e : ExpressionArithmetique) **:** ExpressionArithmetique
     ⌊**précondition(s)** estUneOperation(e)
   — **fonction** operandeD (e : ExpressionArithmetique) **:** ExpressionArithmetique
     ⌊**précondition(s)** estUneOperation(e)
3. Donnez l'algorithme de la procédure suivante qui permet de calculer la valeur v d'une expression arithmétique e si aucune erreur err n'intervient dans ce calcul :
   — **procédure** evaluer (**E** e : ExpressionArithmetique, **S** v : **Reel**, err : **Booleen**)
   Solution proposée :
   procédure evaluer (**E** e : ExpressionArithmetique, **S** v : **Reel**, err : **Booleen**)

```
    Déclaration   vG,vD : Reel
                  errG, errD : Booleen
debut
    si non estUneOperation(e) alors
        v ← valeur(e)
        err ← FAUX
    sinon
        evaluer(operandeG(e),vG,errG)
        si non errG alors
            evaluer(operandeD(e),vD,errD)
            si non errD alors
                err ← FAUX
                cas où operateur(e) vaut
                    ADDITION:
                        v ← vG+vD
                    SOUSTRACTION:
                        v ← vG-vD
                    MULTIPLICATION:
                        v ← vG*vD
                    DIVISION:
                        si vD≠0 alors
                            v ← vG/vD
                        sinon
                            err ← VRAI
                        finsi
                fincas
            sinon
                err ← VRAI
            finsi
        sinon
            err ← VRAI
        finsi
    finsi
fin
```