ASI3 Mardi 23 Janvier 2007

Algorithmique et Bases de la programmation

Durée : *3h*
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.

1 Tri par fusion (2,5 points)

Donnez le corps de la procédure suivante qui permet de trier un tableau d'entiers à l'aide de l'algorithme du tri par fusion.

- **procédure** trierParFusion **(E/S** t : **Tableau**[1..MAX] **d'**Entier ; **E** debut,fin : **Naturel**)

Vous donnerez aussi les algorithmes de toutes les fonctions/procédures qui sont utilisées par cette dernière.

Solution proposée :

procédure fusionner **(E/S** t : **Tableau**[1..MAX] **d'**Entier ; **E** debut,milieu,fin : Entier **)**
Déclaration i,j,k : Entier, temp : **Tableau**[1..MAX] **d'**Entier
début
i ← debut
j ← milieu+1
pour k ←1 **à** fin-debut+1 **faire**
si i≤milieu et j<fin **alors**
si t[i]≤t[j] **alors**
temp[k] ← t[i]
i ← i+1
sinon
temp[k] ← t[j]
j ← j+1
finsi
sinon
si i≤milieu **alors**
temp[k] ← t[i]
i ← i+1
sinon

```
            temp[k] ← t[j]
            j ← j+1
         finsi
      finsi
   finpour
   pour k ←1 à fin-debut+1 faire
      t[debut+k-1] ← temp[k]
   finpour
fin
procédure trierParFusion ( E/S t : Tableau[1..MAX] d'Entier ; E debut,fin : Naturel )
   Déclaration  m : Naturel
début
   si debut < fin alors
      m ← (debut+fin)div 2
      trierParFusion(t,debut, m)
      trierParFusion(t,m+1,fin)
      fusionner(t,debut,m,fin)
   finsi
fin
```

2 Position d'une sous-chaîne (2,5 points)

Donnez le corps de la fonction suivante qui retourne l'indice de la première occurrence de *sousChaine* dans *chaine*. Si *sousChaine* n'est pas présente dans *chaine*, alors la fonction retourne -1.

- **fonction** position (chaine,sousChaine : **Chaîne de caractères) : Entier**

Vous pouvez utiliser les fonctions suivantes :

- **fonction** longueur (uneChaine : **Chaîne de caractères) : Naturel**
  ...avec $longueur("") = 0$
- **fonction** iemeCaractere (uneChaine : **Chaîne de caractères**, iemePlace : **Naturel) : Caractère**

  ⌊**précondition(s)** $0 < iemePlace$
  $iemePlace \leq longueur(uneChaine)$

Solution proposée :

```
fonction position (chaine,sousChaine : Chaîne de caractères) : Entier
   Déclaration  i,j : Entier
                trouve : Booléen
début
   i ← 1
   j ← 1
   trouve ← FAUX
   tant que i<longueur(chaine)-longueur(sousChaine) et non trouve faire
```

```
      si j>longueur(sousChaine) alors
        trouve ← VRAI
      sinon
        si iemeCaractere(chaine,i+j-1)=iemeCaractere(sousChaine,j) alors
          j ← j+1
        sinon
          i ← i+1
          j ← 1
        finsi
      finsi
    fintantque
    si trouve alors
      retourner i
    sinon
      retourner -1
    finsi
fin
```

3 Un peu de C (3 points)

Soit le programme C suivant :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #define TAILLE_BUFFER 100
4
5  char* saisirChaine() {
6    char buffer[TAILLE_BUFFER];
7    scanf("%s",buffer);
8    return buffer;
9  }
10
11 void afficherChaine(char* chaine) {
12   printf("Vous avez tapé %s\n",chaine);
13 }
14
15 int main() {
16   afficherChaine(saisirChaine());
17   return EXIT_SUCCESS;
18 }
```

On compile ce programme :

```
> gcc -c -Wall -pedantic main.c
main.c: In function 'saisirChaine':
main.c:8: warning: function returns address of local variable
```

Répondez aux questions suivantes (à chaque fois en 10 lignes maximum) :

1. Que signifient les options `-c`, `-Wall` et `-pedantic` de *gcc* ?
2. Que signifie le *warning* affiché par *gcc* ? Expliquez comment résoudre ce problème (donnez le code C) ?

Solution proposée :

1. `-c` pour compiler, `-Wall` pour afficher tous les *warning*, `-pedantic` pour avoir du code compatible ISO
2. Le *warning* prévient que le pointeur retourné pointe sur une variable locale (zone mémoire de la pile) qui n'existera plus en sortie de la fonction. Solution : faire une allocation dynamique (zone mémoire du tas).

4 Distance dans un arbre binaire (8 points)

4.1 Question de cours (3 points)

4.1.1 TAD

Donnez le TAD ArbreBinaire et le TAD Dictionnaire à l'aide du formalisme vu en cours (sans la partie axiomatique et sémantique). Donnez aussi la signature des fonctions et procédures de leurs opérations.

Solution proposée :
Les TAD :

Nom:	ArbreBinaire	
Paramètre:	Element	
Utilise:	**Booléen**	
Opérations:	arbreBinaire:	$\rightarrow$ ArbreBinaire
	ajouterRacine:	Element $\times$ ArbreBinaire $\times$ ArbreBinaire $\rightarrow$ ArbreBinaire
	estVide:	ArbreBinaire $\rightarrow$ **Booléen**
	obtenirElement:	ArbreBinaire$-\{\emptyset\} \rightarrow$ Element
	obtenirFilsGauche:	ArbreBinaire$-\{\emptyset\} \rightarrow$ ArbreBinaire
	obtenirFilsDroit:	ArbreBinaire$-\{\emptyset\} \rightarrow$ ArbreBinaire
	fixerFilsGauche:	ArbreBinaire$-\{\emptyset\} \times$ ArbreBinaire $\rightarrow$ ArbreBinaire
	fixerFilsDroit:	ArbreBinaire$-\{\emptyset\} \times$ ArbreBinaire $\rightarrow$ ArbreBinaire
Nom:	Dictionnaire	
Paramètre:	Element,Clé	
Utilise:	**Booléen**	
Opérations:	dictionnaire:	$\rightarrow$ Dictionnaire
	ajouter:	Dictionnaire $\times$ Clé $\times$ Element $\rightarrow$ Dictionnaire
	retirer:	Dictionnaire $\times$ Clé $\rightarrow$ Dictionnaire
	estPresent:	Dictionnaire $\times$ Clé $\rightarrow$ **Booléen**

rechercher: Dictionnaire × Clé → Element

Les fonctions et procédures :
fonction arbreBinaire () **:** ArbreBinaire
fonction ajouterRacine (element : Element, ssArbreG, ssArbreD : ArbreBinaire) **:** ArbreBinaire
fonction estVide (lArbre : ArbreBinaire) **: Booléen**
fonction obtenirElement (lArbre : ArbreBinaire) **:** Element

⌊**précondition(s)** non estVide(lArbre)

fonction obtenirFilsGauche (lArbre : ArbreBinaire) **:** ArbreBinaire

⌊**précondition(s)** non estVide(lArbre)

fonction obtenirFilsDroit (lArbre : ArbreBinaire) **:** ArbreBinaire

⌊**précondition(s)** non estVide(lArbre)

procédure fixerFilsGauche (**E/S** lArbre : ArbreBinaire , **E** ssArbreG : Arbre)

⌊**précondition(s)** non estVide(lArbre)

procédure fixerFilsDroit (**E/S** lArbre : ArbreBinaire , **E** ssArbreD : Arbre)

⌊**précondition(s)** non estVide(lArbre)

fonction dictionnaire () **:** Dictionnaire
procédure ajouter (**E/S** leDictionnaire : Dictionnaire , **E** clé : Clé, element : Element)
procédure retirer (**E/S** leDictionnaire : Dictionnaire , **E** clé : Clé)
fonction estPrésent (leDictionnaire : Dictionnaire, clé : Clé) **: Booléen**
fonction rechercher (leDictionnaire : Dictionnaire, clé : Clé) **:** Element

⌊**précondition(s)** estPresent(leDictionnaire,clé)

4.1.2 Parcours

Soit l'arbre présenté par la figure n° 1. Donnez les parcours RGD, GRD et GDR.

Solution proposée :

RGD a b d c e f g

GRD d b a e c f g

GDR d b e g f c a

4.2 Dictionnaire de pères (2,5 points)

Donnez le corps de la fonction `obtenirPères` suivante :
– **fonction** obtenirPères (a : ArbreBinaire<Element>) **:** Dictionnaire<Element,Element>

⌊**précondition(s)** non estVide(a)

...qui pour un arbre binaire non vide d'éléments distincts permet d'obtenir un dictionnaire dont :

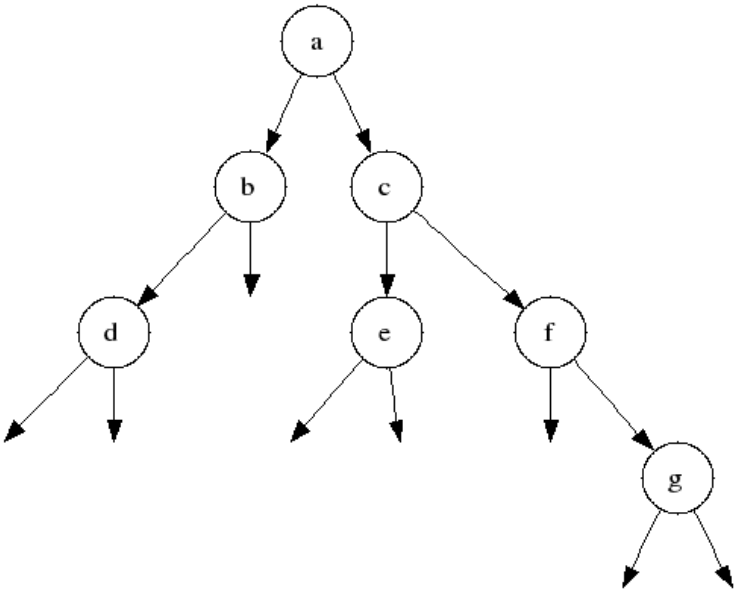

FIG. 1 – Un arbre de caractères

- les clés sont les éléments de l'arbre qui possède un père (c'est-à-dire tous les éléments de l'arbre sauf celui contenu dans la racine)
- la valeur associée à chaque clé est l'élément qui se trouve dans le nœud père.

Par exemple, pour l'arbre présenté par la figure n° 1, le dictionnaire correspondant possédera les couples (clé,valeur) suivante : {('b','a'),('c','a'),('d','b'),('e','c'),('f','c'),('g','f')}

Solution proposée :

procédure remplirDictionnaire **(E/S** d : Dictionnaire<Element,Element> ,
E a : ArbreBinaire<Element>, pere : Element **)**
début
  si non estVide(a) **alors**
    ajouter(d,pere,obtenirElement(a))
    remplirDictionnaire(d,obtenirFilsGauche(a),obtenirElement(a))
    remplirDictionnaire(d,obtenirFilsDroit(a),obtenirElement(a))
  finsi
fin
fonction obtenirPères **(**a : ArbreBinaire<Element>**) :** Dictionnaire<Element,Element>
  ⌊**précondition(s)** non estVide(a)
  Déclaration d : Dictionnaire<Element,Element>
début
  d ← Dictionnaire
  remplirDictionnaire(d,obtenirFilsGauche(a),obtenirElement(a))
  remplirDictionnaire(d,obtenirFilsDroit(a),obtenirElement(a))
  retourner d

fin

4.3 Distance dans un arbre (2,5 points)

Donnez le corps de la fonction suivante :

- **fonction** distanceDansUnArbre (a : ArbreBinaire<Element>, e1,e2 : Element) **: Naturel**

  ⌊**précondition(s)** non estVide(a)

...qui pour un arbre binaire donné non vide d'éléments distincts et deux éléments de cet arbre donne la distance en nombre d'arêtes qu'il y a entre les deux nœuds stockant ces deux éléments. Par exemple dans l'arbre présenté par la figure n° 1, la distance entre 'd' et 'c' est de 3.

Solution proposée :

fonction distance (d : Dictionnaire<Element,Element>,e1,e2 : Element) **: Naturel**
début
  si e1=e2 **alors**
    retourner 0
  sinon
    si non estPresent(d,e1) **alors**
      retourner 1+distance(d,e1,rechercher(e2))
    sinon
      si non estPresent(d,e2) **alors**
        retourner 1+distance(d,rechercher(e1),e2)
      sinon
        retourner min(1+distance(d,e1,rechercher(e2)), 1+distance(d,rechercher(e1),e2))
      finsi
    finsi
  finsi
fin
fonction distanceDansUnArbre (a : ArbreBinaire<Element>, e1,e2 : Element) **: Naturel**
  ⌊**précondition(s)** non estVide(a)
  Déclaration d : Dictionnaire<Element,Element>
début
  d ← obtenirPères(a)
  retourner distance(d,e1,e2)
fin

5 Parcours d'un arbre binaire en largeur (4 points)

5.1 Hauteur d'un arbre (1 point)

Donnez le corps de la fonction suivante qui retourne la hauteur d'un arbre binaire (pour rappel la hauteur d'un arbre vide est de -1) :

- **fonction** hauteur (a : ArbreBinaire<Element>) **: Naturel**

Solution proposée :

fonction hauteur (a : ArbreBinaire<Element>) **: Naturel**
début
  si estVide(a) **alors**
    retourner -1
  sinon
    retourner 1+max(hauteur(obtenirFilsGauche(a)),hauteur(obtenirFilsDroit(a)))
  finsi
fin

5.2 Enfiler des éléments d'un certain niveau (1,5 points)

Donnez le corps de la procédure suivante qui enfile dans f tous les éléments d'un arbre a se trouvant au niveau n :

- **procédure** enfilerElements **(E/S** f : File<Element> , **E** a : ArbreBinaire<Element>, niveau : **Naturel)**

  ⌊**précondition(s)** $0 \leq$ n
  n $\leq$ hauteur(a)

Solution proposée :

procédure enfilerElements **(E/S** f : File<Element> , **E** a : ArbreBinaire<Element>, niveau : **Naturel)**

⌊**précondition(s)** $0 \leq$ n
n $\leq$ hauteur(a)

début
  si niveau=0 **alors**
    enfiler(f,obtenirElement(a))
  sinon
    enfilerElements(f,obtenirFilsGauche(a),niveau-1)
    enfilerElements(f,obtenirFilsDroit(a),niveau-1)
  finsi
fin

5.3 Parcours en largeur (1,5 points)

Donnez le corps de la fonction suivante :

- **fonction** parcoursEnLargeur (a : ArbreBinaire<Element>) **:** File<Element>

...qui à partir d'un arbre a retourne une file qui contient tous les éléments de a rangés suivant un parcours en largeur.
Par exemple la file correspondante à l'arbre de la figure n° 1 est ('a','b','c','d','e','f','g') avec 'a' en tête de file.

Solution proposée :

```
fonction parcoursEnLargeur (a : ArbreBinaire<Element>) : File<Element>
   Déclaration  f : File<Element>
                i : Naturel
début
   f ← file()
   si non estVide(a) alors
      pour i ←0 à hauteur(a) faire
         enfilerElements(f,a,i)
      finpour
   finsi
   retourner f
fin
```