Licence de Mathématiques et Informatique — Université de Perpignan Via Domitia
Semestre 2

Algorithmique

Correction du contrôle continu de mars 2016

Durée : 2 heures. Aucun document autorisé.

Modalités : Répondre uniquement dans les cadres prévus à cet effet. Le barème est indiqué à droite de chaque question. La qualité de la rédaction sera prise en compte dans la notation.

Exercice 1. (12 points) **/12**

1. Un sous-programme `min` qui retourne le minimum de 2 entiers (de signe quelconque) est une fonction ou une procédure ? Justifier votre réponse. **/1**

```
Une fonction : elle retourne une valeur sans modifier les variables entrées.
```

2. Écrire l'en-tête du sous-programme `min`. **/1**

```
  fonction min(a : in entier, b : in entier) retourne entier
```

3. Écrire un algorithme qui utilise `min` pour afficher le minimum de deux valeurs entrées au clavier. **/1**

```
declare
  v1, v2, m : entiers
debut
  lire(v1)
  lire(v2)
  m = min(v1, v2)
  afficher(m)
fin
```

4. Écrire un algorithme qui utilise `min` pour afficher la valeur minimale d'un nombre arbitraire $n > 0$ de valeurs entrées au clavier. On veillera à ce que la valeur entrée pour n soit strictement positive ; dans le cas contraire, l'entrée d'une nouvelle valeur pour n sera répétée autant de fois que nécessaire. **/3**

```
declare
  v, n,  m, i : entiers
debut
  repeter
    afficher(``n :'')
    lire(n)
  jusqua (n > 0)

  afficher(``val :'')
  lire(m)

  pour i de 1 à n-1 faire
    afficher(``val :'')
    lire(v)
    m = min(m, v)
  finpour

  afficher(m)
fin
```

5. Écrire l'en-tête du sous-programme `min3` qui retourne la valeur minimale de 3 entiers. **/1**

```
fonction min3(a : in entier, b : in entier, b : in entier) retourne entier
```

6. Écrire le corps du sous-programme `min3` en utilisant `min`. **/2**

```
// On peut éviter d'utiliser res avec 2 retourne
fonction min3(a : in entier, b : in entier, b : in entier) retourne entier
  res : entier
  avec fonction min(a : in entier, b : in entier) retourne entier

debut
  si a < b alors
    res = min(a,c)
  sinon
    res = min(b,c)
  finsi
  retourner res
fin
```

7. Écrire le corps du sous-programme `min3` sans utiliser `min`. **/3**

```
fonction min3(a : in entier, b : in entier, b : in entier) retourne entier
  res : entier
debut
  si a < b alors
    res = a
  sinon
    res = b
  finsi

  si c < res
    res = c
  finsi

  retourner res
fin

// autre solution bcp plus lourde (mais peut éviter la variable locale res)
fonction min3(a : in entier, b : in entier, b : in entier) retourne entier
```

```
  res : entier
debut
  si a < b alors
    si a < c
      res = a
    sinon
      res = c
    finsi
  sinon
    si b < c
      res = b
    sinon
      res = c
    finsi
  finsi
  retourner res
fin
```

Exercice 2. (10 points) **/8**

Cet exercice s'intéresse aux *palindromes*. Un palindrome est un mot, ou un groupe de mots, dont l'ordre des lettres reste le même qu'on le lise de la droite vers la gauche ou inversement. Des exemples bien connus sont "été", "kayak", "mon nom", "élu par cette crapule". Ce dernier permet d'illustrer qu'on ne tient pas compte en général des accents, trémas, cédilles, ni des espaces.

Dans cet exercice :
— un mot ou un groupe de mots est un tableau 1D de caractères de longueur maximale 512,
— ces caractères sont sans accent, tréma, ni cédille : "ete"
— les espaces sont considérés comme des caractères : ainsi "elu par cette crapule" n'est pas un palindrome ici.

1. Compléter le début d'algorithme suivant et afficher si `m` est, ou non, un palindrome. **/2**

```
declare
   m : tableau[7] de caractères = ``mon nom''
   ...
```

```
declare
  m : tableau[7] de caractères = [m,o,n, ,n,o,m]
  i : entier
  res : booléen = vrai
début
  pour i de 0 à 2
    si m[i] != m[6-i] alors
      res = faux
    finsi
  finpour

  afficher(``palindrome (V/F) :'', res)
fin
```

2. Préciser le sens du résultat de la division entière de 2 entiers positifs $a//b$ où $b \neq 0$.
   Donner 2 exemples significatifs. **/1**

```
Le résultat de a//b est le quotient q de la division euclidienne :
a = b . q + r  avec 0 <= r < b-1.
2 // 2 = 1   3 // 2 = 1
```

3. Modifier l'algorithme précédent pour qu'il s'applique à un mot de longueur arbitraire `n` sans se soucier de son initialisation. Compléter le début d'algorithme suivant et utiliser une boucle `pour`. **/2**

```
declare
  n : constante entier = ...
  m : tableau[n] de caractères = ...
  ...
```

```
declare
  n : constante entier = ...
  m : tableau[n] de caractères = ...
  i : entier
  res : booléen = vrai
début
  pour i de 0 à n//2
    si m[i] != m[n-1-i] alors
      res = faux
    finsi
  finpour

  afficher(``palindrome (V/F) :'', res)
fin
```

4. Écrire l'en-tête du sous-programme `VerifPalindrome` qui vérifie si un mot donné est, ou non, un palindrome. **/1**

```
  fonction VerifPalindrome(n : entier, m : tableau[n] de caractères) retourne booleen
```

5. Écrire le corps du sous-programme `VerifPalindrome` en utilisant une boucle `tant que`. **/2**

```
fonction VerifPalindrome(n : entier, m : tableau[n] de caractères) retourne booleen
  i : entier =0
  res : booléen = vrai
début
  tant que (i <= n//2) ET (res == vrai) faire
    si m[i] != m[n-1-i] alors
      res = faux
    finsi
    i = i+1
  fintantque

  retourne res
fin
%-------------------------------------------------------------------------------
// Autre solution plus courte
fonction VerifPalindrome(n : entier, m : tableau[n] de caractères) retourne booleen
  i : entier =0
début
  tant que (i <= n//2) ET (m[i] == m[n-1-i]) faire
    i = i+1
  fintantque

  retourne (i == n//2+1) // qui est vrai si la boucle tantque a repéré un palindrome
fin

%-------------------------------------------------------------------------------
```

Exercice 3. (33 points) **/33**

On va manipuler et transformer des tableaux.

1. On commence avec des tableaux 1D d'entiers.

   (a) Soient trois tableaux 1D d'entiers de taille 10 et d'identifiants respectifs `t1, t2, t3`.
   Écrire un algorithme qui les initialise avec les valeurs successives 1, 2, …, 10, en procédant comme suit :

   — `t1` est initialisé lors de sa la déclaration, **/1**

   — `t2` est initialisé avec une boucle `pour`, **/1**

   — `t3` est initialisé avec une boucle `tant que`. **/1**

```
declare
  t1 : tableau[10] d'entiers = [1,2,3,4,5,6,7,8,9,10]
  t2, t3 : tableau[10] d'entiers
  i : entier
debut
  pour i de 0 à 9 faire
    t2[i] = i+1
  finpour

  i = 0
  tantque i < 10 faire
    t[i] = i+1
    i = i+1
  fintantque
fin
```

   (b) ($\star$) On suppose que les tableaux sont de taille n. Quelle est la complexité asymptotique de chacune des initialisations ? Justifier votre réponse. **/2**

```
La complexité de ces 3 initialisations est linéaire : il faut parcourir le tableau
de longueur n une fois exactement.
Ce résultat asymptotique est le même que l'on mesure le nombre d'affectations ou
le nombre d'additions.
```

(c) Modifier l'algorithme précédent pour effectuer, dans l'ordre, les traitements suivants :
- une valeur entière k est "entrée au clavier" ;
- `t2` est initialisé avec k et ses neuf valeurs entières immédiatement supérieures (et successives) en utilisant une boucle `pour` ; **/1**
- `t3` est initialisé avec les 10 valeurs **paires** immédiatement supérieures ou égale à k en utilisant avec une boucle `tant que` (aucune hypothèse est faite sur k) ; **/2**
- les valeurs d'indice impair de `t1` sont mises à zéro (son initialisation lors de sa déclaration n'a pas été modifiée). **/1**

```
declare
  t1 : tableau[10] d'entiers = [1,2,3,4,5,6,7,8,9,10] //inchangé
  t2, t3 : tableau[10] d'entiers
  k, i, valPaire : entier
debut
  afficher(``entrer une valeur entière svp :'')
  lire(k)

  pour i de 0 à 9 faire
    t2[i] = k+i
  finpour

  // on commence bien avec k ou son successeur selon sa parité
  si k mod 2 == 0 alors
    valPaire = k
  sinon
    valPaire = k+1
  fin si
  // on y va
  i = 0
  tantque i < 10 faire
    t[i] = valPaire + 2
    i = i+1
  fintantque

  pour i de 1 à 9 par pas de 2 faire
    t1[i] = 0
  finpour
fin
```

2. On continue avec des tableaux 2D d'entiers. On dit qu'une ligne (ou une colonne) d'un tableau 2D est *constante* si elle est composée de valeurs identiques. Par exemple, une ligne composée uniquement de la valeur 0 est dite constante et nulle, et une colonne de 1 est dite constante et égale à 1.

   (a) Écrire un algorithme qui initialise `t` : un tableaux 2D d'entiers de 5 lignes et 10 colonnes avec des lignes constantes, alternativement égales à 0 et à 1, en commençant par une ligne égale à 0. **/2**

```
declare
  t : tableau[5,10] d'entiers
  i, j : entiers
debut
  pour i de 0 à 5 faire
    pour j de 0 à 9 faire
      si i % 2 == 0 alors
        t[i,j] = 0
      sinon
        t[i,j] = 1
      finsi
    finppour
  finpour
fin
%-------------------------------------------------------------------------------
// autre solution écrite pour un nb de lignes quelconque (pair ou impair)
declare
  nl : contante entiere = 5
  t : tableau[nl,10] d'entiers
  i, j : entiers
  n : entier = nl // n=nl si n pair, sinon n=nl-1
debut
  si nl % 2 == 1 alors
    n = nl-1
  finsi

  // traitement des lignes 2 par 2 à partir de la première
  pour i de 0 à n par pas de 2 faire
    pour j de 0 à 9 faire
      t[i  ,j]   = 0
      t[i+1,j] = 1
    finpour
  finpour

  // la dernière ligne si nl est impair
  si n != nl alors
    pour j de 0 à 9 faire
      t[nl,j] = 0
    finpour
  finsi
fin
%-------------------------------------------------------------------------------
// encore une autre solution avec 2 fois 2 boucles pour imbriquées
declare
  t : tableau[5, 10] d'entiers
  i, j : entiers
debut
  // les lignes de 0 à partir de la première
  pour i de 0 à 4 par pas de 2 faire
    pour j de 0 à 9 faire
      t[i,j] = 0
    finpour
  finpour

  // les lignes de 1 à partir de la suivante
  pour i de 1 à 4 par pas de 2 faire
    pour j de 0 à 9 faire
```

```
      t[i,j] = 1
    finpour
  finpour
fin
```

(b) Modifier l'algorithme précédent pour que :
— une valeur entière k est "entrée au clavier" ;
— la première ligne de `t` est une ligne constante égale à k,
— ($\star$) chaque ligne suivante est constante et égale à la somme des valeurs de la ligne précédente.

Par exemple si $k = 1$, la deuxième ligne est une ligne de 10. **/3**

On écrira une solution indépendante de la taille de `t`.

```
declare
  t : tableau[5, 10] d'entiers
  i, j, k : entiers
  s, next_s : entier = 0
debut
  afficher(``entrer une valeur entière svp :'')
  lire(k)

  // initialisation de la première ligne et calcul de sa somme
  pour j de 0 à 9 faire
    t[0,j] = k
    s = s + k
  finpour

  // les suivantes
  pour i de 1 à 4 faire
    pour j de 0 à 9 faire
      t[i,j] = s
      next_s = next_s + s
    finpour
    s = next_s
  finpour
fin
```

(c) ($\star$) On suppose que le tableau `t` est carré de taille $n \times n$. Quelle est la complexité asymptotique de l'algorithme précédent ? Justifier votre réponse. **/2**

```
La complexité de ces 3 initialisations est quadratique : il faut parcourir le
tableau 2D de taille n x n une fois exactement.
Ce résultat asymptotique est le même que l'on mesure le nombre d'affectations ou
le nombre d'additions.
```

(d) ($\star$) Modifier l'algorithme précédent pour que :

— la première ligne de `t` est initialisée à 1, 2, . . ., 10. **/1**

— chaque case de `t` vaut la somme des cases "au dessus d'elle" dans la même colonne. **/3**
Par exemple, la deuxième ligne commence par un 1 et la suivante par un 2.
On écrira une solution indépendante de la taille de `t`.

```
declare
  t : tableau[5, 10] d'entiers
  i, j, k : entiers
debut

  // initialisation de la première ligne
  pour j de 0 à 9 faire
    t[0,j] = j + 1
  finpour

  pour i de 1 à 4 faire
    pour j de 0 à 9 faire
      t[i,j] = 0                  // SURTOUT NE PAS OUBLIER
      pour k de 0 à i-1 faire
        t[i,j] = t[i,j] + t[k, j]
      finpour
    finpour
  finpour
fin
```

3. Soit `t` un tableau 2D d'entiers de n lignes et n colonnes.

(a) On suppose que $n = 10$. Écrire un algorithme qui initialise `t10` avec, pour chaque case, la somme de ces indices de ligne et de colonne. **/2**

```
declare
  t10  : tableau[10, 10] d'entiers
  i, j : entiers
debut
  pour i de 0 à 9 faire
    pour j de 0 à 9 faire
      t10[i,j] = i+j
    finpour
  finpour
fin
```

(b) `t10` est-il symétrique par rapport à sa première diagonale ? Justifier votre réponse. **/1**

```
Oui car :
. t10 est un tableau carré
. et t[i,j] = i+j = j+i = t[j,i]
```

(c) Écrire une fonction `VerifSym` qui vérifie qu'un tableau 2D de taille arbitraire est symétrique par rapport à sa première diagonale. D'abord l'en-tête : **/1**

```
fonction VerifSym(n: entier, t: tableau[n,n] d'entiers) retourne booleen
```

(d) Puis le corps de `VerifSym` : **/3**

```
fonction VerifSym(n: entier, t: tableau[n,n] d'entiers) retourne booleen
  i, j : entier
  res : booleen = vrai
debut
  pour i de 0 à n-1 faire
    pour j de 0 à i-1 faire  // partie triangulaire inferieure
      si t[i,j] != t[j,i] alors
        res = faux
```

```
      finsi
    finpour
  finpour

  retourne res
fin
%-----------------------------------------------
// autre solution plus économique : 2 boucles tantque

fonction VerifSym(n: entier, t: tableau[n,n] d'entiers) retourne booleen
  i : entier = 0
  j : entier = 0
  res : booleen = vrai
debut
  tant que (res == vrai) et (i < n) faire
    tant que (res == vrai) et (j < i) faire
      si t[i,j] != t[j,i] alors
        res = faux
      finsi
    j = j+1
    fintantque
  i = i+1
  fintantque

  retourne res
fin
```

4. (⋆) Écrire une propriété qui permette de vérifier qu'un tableau 2D carré est symétrique par rapport à sa **seconde** diagonale. **/2**

```
t[i,j] = t[n-1 -j, n-1 -i]
```

5. (⋆) Écrire un algorithme qui génère un tableau 2D carré,
   — de taille arbitraire (qui pourra être initialisée par une constante),
   — symétrique par rapport à sa seconde diagonale
   — et qui contienne au moins $2n$ valeurs différentes.
   Cette dernière condition interdit de générer des tableaux remplis d'une unique valeur ou d'autres répétitions triviales. **/4**

```
declare
  n : constante entiere = ...
  t  : tableau[n, n] d'entiers
  i, j : entiers
  val : entier = 0  // pour remplir le tableau avec des val distinctes
debut
  // on remplit le triangle inférieur
  // colonne par colonne en partant de la première
  pour j de 0 à n-1 faire
    pour i de j à n-1 faire  // on descend la colonne depuis la diagonale
      t[i,j] = val
      t[n-1 -i, n-1- j] = t[i, j]  // on symétrise
      val = val + 1
    finpour
  finpour
fin
```