Licence de Mathématiques et Informatique
Semestre 2

Université de Perpignan Via Domitia

Algorithmique

Contrôle continu de mars 2016

NOM : **Prénom :** **Groupe TD :**

Durée : 2 heures. Aucun document ni téléphone 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. Un barème est indiqué à droite de chaque question. La note finale sur 20 est proportionnellement déduite de ce barème.

Ne pas dégrafer les feuilles.

/ 20

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**

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

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

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**

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

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

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

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''
   ...
```

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**

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 = ...
  ...
```

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

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

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**

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

(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**

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**

(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`.

(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**

(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`.

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**

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

(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**

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

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

5. ($\star$) 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**