Les tableaux
Corrigé

Objectifs

— Savoir définir et manipuler les types utilisateurs (énuméré, enregistrement et tableau)

— Spécifier, implanter et tester des sous-programmes... Toujours !

Exercice 1 : Tri par sélection

Soit A un vecteur de N entiers relatifs quelconques, l'objectif est de trier le vecteur A en utilisant le tri par sélection. Le vecteur A est trié si $A[i] \leq A[i+1]$.

Le tri par sélection est un tri en $(N-1)$ étapes. L'étape i consiste à ranger à sa place le i^{e} plus petit élément du vecteur.

Exemple : Voici les différentes valeurs du vecteur 8 2 9 5 1 7 après chaque étape (la partie encadrée correspond à la partie du vecteur déjà traitée et donc triée) :

vecteur initial	:	8 2 9 5 1 7
après l'étape 1	:	[1] 2 9 5 8 7
après l'étape 2	:	[1 2] 9 5 8 7
après l'étape 3	:	[1 2 5] 9 8 7
après l'étape 4	:	[1 2 5 7] 8 9
après l'étape 5	:	[1 2 5 7 8 9]

1. Écrire un sous-programme qui trie un vecteur en utilisant le tri par sélection. On complètera le fichier `tri_selection.adb`.

Solution : Voir le code.

Exercice 2 : Vérifier un générateur aléatoire

Écrire un sous-programme (fichier `evaluer_alea.adb`) qui évalue la qualité d'un générateur de nombres aléatoires. L'idée est de vérifier que tous les nombres que peut produire le générateur ont la même probabilité d'apparaître. Pour ce faire, on prendra un échantillon (nombres aléatoires tirés) de grande taille et on affichera la plus petite et la plus grande fréquence absolue ainsi que la fréquence absolue théorique.

Par exemple, si on considère un dé à 6 faces, on a 6 entiers possibles (de 1 à 6). Si on considère une taille d'échantillon de 20, on réalise 20 tirages aléatoires. On peut par exemple obtenir les résultats suivants : 5, 2, 1, 2, 4, 5, 3, 2, 2, 5, 2, 3, 5, 3, 2, 2, 2, 6, 4, 3. On calcule alors que la fréquence absolue de 1 (le nombre d'occurrences de 1) est 1, celle de 2 est 8, celle de 3 est 4, celle de 4 est 2, celle de 5 est 4, celle de 6 est 1. La fréquence minimale est donc 1 (fréquence et 1

ou 6) et la maximale est 8 (fréquence de 2). La fréquence absolue théorique est 3,33 (20 / 6). Cet exemple n'est pas pas convaincant mais l'échantillon est trop petit pour en tirer des conclusions !
Indication : Doit-on conserver tous les nombres tirés ?
Solution : Non. Il faut juste savoir combien de fois est sortie chaque nombre. Il est donc inutile de conserver dans un tableau tous les nombres tirés aléatoirement.

Utiliser un tel tableau nous contraindrait sur la taille de l'échantillon qui ne pourrait pas être plus grand que la capacité du tableau.

Même si on n'avait pas de contrainte de capacité sur le tableau, pourquoi stocker en mémoire tous ces nombres alors que ce n'est pas nécessaire ?

Quelles informations faut-il conserver ?
Solution : Ce qui nous intéresse, c'est seulement la fréquence de chaque nombre dans l'échantillon. On doit donc juste conserver la fréquence de chaque nombre.

En quoi un tableau peut être utile ?
Solution : Le tableau permet de conserver la fréquence (nombre d'occurrences) d'un nombre. Ce nombre est l'indice. La valeur à cet indice est la fréquence.

A-t-on besoin de gérer une taille effective sur ce tableau ?
Solution : Non car on utilise toutes les cases du tableau. On commencera à initialiser toutes les fréquences et donc toutes les cases du tableau à 0 (fréquence nulle au départ).
Solution :

Voir le code.

Exercice 3 : Supprimer certains éléments d'un tableau
Écrire un sous-programme (fichier `tableau_supprimer.adb`) qui supprime tous les entiers strictement négatifs d'un tableau ainsi que tous les entiers strictement supérieurs à 20. L'algorithme mis en œuvre devra être en $O(n)$ pour les affectations et les comparaisons, n étant le nombre d'éléments dans le tableau.
Solution : Voir le code.

Pour aller plus loin...

Exercice 4 : Tester le sous-programme de tri
L'objectif de cet exercice est d'écrire un programme de test du sous-programme qui trie un tableau. Vérifier que le sous-programme de tri a bien fonctionné, c'est vérifier que [1] :

1. le tableau est trié : les éléments sont dans l'ordre croissant,
2. le tableau initial et le tableau trié contiennent les mêmes éléments.

Pour vérifier cette deuxième propriété, on peut vérifier que :

1. tableau initial et tableau trié ont la même taille,
2. tous les éléments du tableau initial sont présents dans le tableau trié avec le même nombre d'occurrences dans les deux tableaux.

1. Ces propriétés sont en fait les postconditions du sous-programme de tri.

1. Écrire une procédure (fichier `tri_selection.adb`) qui vérifie que le sous-programme de tri fonctionne bien pour un tableau donné en vérifiant les propriétés ci-dessus.
Remarque : On aura intérêt à définir des sous-programmes supplémentaires pour écrire plus simplement ces propriétés.
2. Écrire une procédure de test qui appelle la procédure précédente avec différents tableaux. On pourra par exemple utiliser les tableaux suivants :

- `[1, 3, 4, 2]`
- `[4, 3, 2, 1]`
- `[-5, 3, 8, 1, -25, 0, 8, 1, 1, 1]`

3. Compléter la procédure de test pour faire 10 tests sur un tableau dont les valeurs ont été initialisées aléatoirement.
Solution : Voir le code.