Mini-projet 2

Rappel : Comme pour tous les TP, il faut commencer par faire un « svn update » depuis votre dossier « pim/tp » pour récupérer les fichiers fournis.

1 Cahier des charges

Quand on veut stocker un nombre variable d'éléments, on peut utiliser un tableau. Ce tableau doit avoir une capacité suffisante et contient généralement moins d'éléments. On gère alors une taille effective comprise entre 0 et la capacité du tableau. Ces deux informations, le tableau et sa taille, peuvent être regroupées dans un enregistrement.

Sur un tableau en Ada, on peut essentiellement consulter le i^e élément et modifier le i^e élément. Si nous définissons notre type Tableau, nous devons fournir ces opérations de base mais nous avons aussi intérêt à définir des opérations de plus haut niveau comme ajouter un élément à la fin du tableau, insérer un élément à un certain indice, savoir si un élément est présent dans le tableau, ajouter à la fin d'un tableau les éléments d'un autre...

Ce nouveau type Tableau et les sous-programmes associés seront définis dans un module. De plus, pour que ce module soit utilisable dans différents contextes, il faut permettre à son utilisateur de choisir le type des éléments et la capacité du tableau.

L'objectif est donc de définir le module générique Tableaux, le programme de tests associé et un exemple d'utilisation. La suite décrit plus précisément ce qui est attendu.

E_1 Le module Tableaux doit fournir le type Tableau dont la définition ne sera pas accessible des utilisateurs de ce module.

E_2 Le module doit être générique sur le type des éléments et la capacité du tableau. Par exemple, on peut avoir un tableau d'entiers de capacité 10 ou un tableau de 50 réels.

E_3 Les éléments doivent être stockés de manière contiguë (il n'y aura pas de « case vide » dans le tableau) sur les petits indices (la partie gauche du tableau).

E_4 Pour les exemples qui suivent, on considèrera que le tableau `Tab1` est un tableau de 10 entiers qui contient les 5 éléments : 4, 3, 1, 3 et 5 que l'on notera [4, 3, 1, 3, 5]. Le module doit fournir les sous-programmes correspondants aux opérations suivantes.

- SP_1 Initialiser un tableau. Sa taille est nulle. Il ne contient aucun élément.
- SP_2 Obtenir la taille, le nombre d'éléments, du tableau. La taille de `Tab1` est 5.
- SP_3 Obtenir l'élément à un indice valide du tableau. L'élément à l'indice 1 de `Tab1` est 4, celui à l'indice 4 est 3.

- SP_4 Modifier l'élément à un indice valide du tableau. Si on modifie l'indice 2 de `Tab1` pour y mettre -7, on obtient le tableau [4, -7, 1, 3, 5]
- SP_5 Ajouter un élément à la fin du tableau. Si on ajoute 2 à la fin de `Tab1`, le tableau devient [4, 3, 1, 3, 5, 2].
- SP_6 Savoir si un élément est présent dans un tableau. L'élément 3 est présent dans `Tab1`, 11 ne l'est pas.
- SP_7 Insérer un élément à un indice donné. On décale à droite les éléments à partir de l'indice donné pour faire de la place au nouvel élément. Par exemple, insérer 9 à l'indice 2 dans `Tab1` donne le tableau [4, 9, 3, 1, 3, 5] ; insérer -1 à l'indice 6 donne [4, 3, 1, 3, 5, -1]
- SP_8 Supprimer l'élément à un indice valide donné. Le tableau contient un élément de moins et les éléments à droite de l'indice donné sont décalés à gauche. Si on supprime l'élément à l'indice 2 de `Tab1`, on obtient le tableau [4, 1, 3, 5].
- SP_9 Supprimer toutes les occurrences d'un élément dans un tableau. Ce sous-programme doit s'exécuter en temps linéaire (proportionnel au nombre d'éléments du tableau). Si on supprime l'élément 3 de `Tab1`, on obtient [4, 1, 5].
- SP_{10} Appliquer une opération sur chaque élément du tableau.

  Souvent on a besoin de faire une même opérations sur tous les éléments d'un Tableau. Par exemple, sur un tableau d'entiers, on peut vouloir mettre à 0 tous les éléments, ajouter un à tous les éléments, mettre au carré tous les éléments, remplacer par 0 les éléments impairs, etc. Ainsi, si on met au carré tous les éléments de `Tab1`, le tableau devient [16, 9, 1, 9, 25].

  On ne peut pas écrire tous ces sous-programmes dans le module Tableaux. On veut éviter que les utilisateurs du module aient à écrire de multiples parcours du tableau pour modifier les éléments du tableau.

  Une solution consiste à définir un sous-programme `Appliquer_Sur_Chaque` avec un paramètre de généricité, la procédure qui modifie un élément du tableau. Ce sous-programme applique cette procédure sur chaque élément du tableau.

- E_5 Pour chaque sous-programme, sa spécification doit être complète. Les préconditions et post-conditions seront formalisées dans la syntaxe Ada (dans la mesure du possible).
- E_6 Un programme de tests du module doit être réalisé. Il devrait y avoir au moins une procédure de test pour chaque sous-programme important (SP_5 et suivants).
- E_7 Un programme montrera l'utilisation du module Tableaux. La seule contrainte sur ce programme est d'utiliser deux tableaux de types et capacités différents.

2 Livrables

Les squelettes des livrables sont fournis sur le SVN, dossier `pr2`. Les versions intermédiaires et finales devront y être poussées.

L_1 `tableaux.ads` : la spécification du module Tableaux. Les sous-programmes doivent **impérativement** apparaître dans l'ordre du sujet.

L_2 `tableaux.adb` : l'implantation du module Tableaux. Les sous-programmes apparaîtront **impérativement** dans l'ordre du sujet.

L_3 `test_tableaux.adb` : le programme de tests du module Tableaux.

L_4 `exemple_tableaux.adb` : le programme qui utilise le module Tableaux.

L_5 `LISEZ-MOI.txt` : fichier fourni à compléter.

3 Échéances

Ce travail est à rendre au plus tard le 17 juin.