Structures simplement chaînées

Objectifs

— Écrire une structure de données simplement chaînée.

Rappel : Comme pour tous les TP, il faut commencer par faire un « `git pull` » depuis votre dossier « pim/tp » pour récupérer les fichiers fournis pour cette séance.

Exercice 1 : Vecteurs creux

Les calculs numériques sur ordinateur font appel à des structures de données qui sont principalement les *vecteurs* et les *matrices*. Lorsque les problèmes traités nécessitent un volume de données important – on parle alors de problèmes de *grande taille* – la question du stockage de ces données devient cruciale. Or, il se trouve que dans de nombreuses applications, les matrices ou les vecteurs de grande taille manipulés ne contiennent en fait qu'un *petit nombre* (par rapport à la taille) d'*éléments non nuls* : on parle alors de matrices et vecteurs « creux ». L'idée est de ne stocker que l'information utile, à savoir les éléments non nuls. Encore faut-il disposer d'une technique de stockage permettant d'accéder aux données comme si elles étaient stockées dans des matrices ou des vecteurs « pleins » (c'est-à-dire où tous les éléments, mêmes nuls, sont représentés).

On s'intéresse ici aux vecteurs creux que l'on représentera par une structure chaînée. Chaque cellule de cette structure contient l'indice et la valeur d'une composante non nulle du vecteur.

Un vecteur creux doit pouvoir être manipulé comme un vecteur plein et propose donc les opérations usuelles telles que l'initialisation (le vecteur est alors nul), la destruction, l'accès à la i^e composante et sa modification, l'égalité entre deux vecteurs, le carré de la norme, le produit scalaire et l'addition.

Démarche : Quand on développe un module, il faut commencer par identifier les opérations qui seront accessibles aux utilisateurs de ce module. Ici, les opérations peuvent être déduites du texte qui décrit les vecteurs creux. On peut alors écrire la spécification du module. La spécification du module vecteurs creux est fournie (`vecteurs_creux.ads`).

On peut aussi écrire les programmes de test qui permettront de tester ces différentes opérations. Ceci permet de valider que les opérations du module sont bien comprises. Un programme de tests est fourni (`test_vecteurs_creux.adb`).

La connaissance et la compréhension des opérations du module permettent de choisir une représentation des données efficace en terme d'espace mémoire et temps de calcul.

Une fois le choix de représentation des données fait, il s'agit d'implanter le module et de le tester. Pour cette dernière partie, il est souhaitable d'implanter et de tester au fur et à mesure les sous-programmes.

1. Le type `T_Vecteur_Creux` est déjà défini. Un invariant spécifie que les composantes sont stockées par ordre croissant de leurs indices. Ceci permet d'implanter de manière efficace (en temps

linéaire) les opérations telles que le produit scalaire, l'égalité ou la somme de deux vecteurs. Expliquer pourquoi.

2. Pour suivre la démarche préconisée, et comme les tests sont déjà fournis, il est demandé d'écrire un programme d'exemple (`exemple_vecteurs_creux.adb`) qui décrira un scénario particulier d'utilisation des vecteurs creux. Ainsi, pour chacun des sous-programmes à implanter, il s'agit de :

1. compléter le scénario en utilisant ce sous-programme,
2. implanter le sous-programme,
3. compiler et exécuter le scénario pour voir si des erreurs apparaissent (et les corriger).
4. valider les modifications sur Git et les pousser sur Gitlab (en indiquant quelle opération a été implantée et testée).

Appliquer cette démarche et pousser sur Gitlab les modifications après chaque sous-programme implanté et testé (le message mentionnera le nom du sous-programme). Le sous-programme Afficher est déjà implanté. On traitera les sous-programmes dans l'ordre suivant :

1. Initialiser un vecteur creux. Dans le scénario, on peut initialiser le vecteur V et l'afficher.
2. déterminer si un vecteur est nul. Dans le scénario, on peut vérifier que V est nul.
3. Détruire un vecteur creux. Dans le scénario, on peut détruire V.
4. Obtenir une composante de V. Dans le scénario, on peut vérifier que la composante d'indice 18 vaut 0. Mais on aura du mal à vraiment la tester tant qu'on n'aura pas écrit le sous-programme qui permet de modifier une composante. C'est aussi le cas pour les sous-programmes précédents.

   On écrira deux versions de ce sous-programme, l'une itérative, l'autre récursive.
5. Modifier une composante d'un vecteur creux. Dans le scénario, on peut modifier plusieurs composantes de V et vérifier que la modification est effective.
6. Déterminer si deux vecteurs creux sont égaux. Ici, on peut basculer sur les tests fournis...

   On écrira deux versions de ce sous-programme, l'une itérative, l'autre récursive.
7. Additionner à un vecteur un autre vecteur.
8. Calculer le carré de la norme d'un vecteur creux.
9. Calculer le produit scalaire de deux vecteurs.
10. Bien d'autres opérations pourraient être définies...