Structure de données

TP4

Vector : tableau redimensionnable à la Java/C++

Lorsque l'on crée un tableau en C, automatiquement ou dynamiquement, il faut toujours préciser sa capacité. Cette capacité ne peut pas être changée. Dans le cas d'un tableau alloué dynamiquement, l'utilisateur peut toujours allouer un nouveau tableau de taille plus grande (ou plus petite) qui remplacera le tableau initial.

L'objectif du type *Vector (Le nom est inspiré des langages Java et C++ qui propose un type ayant des propriétés similaires)* est de définir des opérations de haut niveau sur un tableau et qui redimensionnent automatiquement le tableau. Ainsi, si on veut ajouter un élément dans un vecteur mais que le tableau utilisé pour sa représentation est plein, on réalloue le tableau pour augmenter sa capacité et ainsi pouvoir ajouter le nouvel élément.

Cependant, faire une réallocation pour chaque élément ajouté ne serait efficace ni en terme de temps d'exécution, ni en terme d'utilisation de la mémoire. Aussi, si le tableau doit être agrandi, il est augmenté de plusieurs unités, par exemple 10.

Voici les opérations à définir sur le type *Vector* (les conditions d'applicabilité ne sont pas données quand elles sont évidentes/naturelles) :

- create_expert : initialiser un vecteur en donnant la capacité initiale du vecteur et l'incrément utilisé lors des agrandissements. Le vecteur ainsi créé à une taille effective nulle.
- create : initialiser un vecteur avec une capacité de 10 et un incrément de 3.
- destroy : détruire un vecteur. Il ne pourra plus être utilisé, sauf à être de nouveau créé.
- size : obtenir la taille effective du vecteur.
- capacity : obtenir la capacité du vecteur, c'est-à-dire le nombre d'éléments que peut contenir le vecteur sans faire de redimensionnement.
- add : ajouter un nouvel élément à la fin du vecteur.
- get : récupérer l'élément à la position i. Le premier élément est à la position 0 (même convention que pour les tableaux en C) ;
- set : changer la valeur de l'élément en position i.
- remove : supprimer l'élément en position i. La taille effective est donc diminuée de 1.
- adjust : adapter la capacité du vecteur à sa taille effective.
- insert : ajouter un nouvel élément en position i. L'entier i doit être compris entre 0 (ajout en début du vecteur) et size (ajout à la fin du vecteur).
- delete : supprimer les éléments compris entre les indices i_début inclu et i_fin exclu.

Écrire le module *Vector* et les programmes de test associés. Il est conseillé d'écrire (et donc tester) les opérations dans l'ordre où elles sont listées.

Remarque : On utilisera *assert* pour vérifier que l'allocation de mémoire s'est bien passée. Cette technique n'est pas robuste puisqu'elle provoque un plantage du programme et ne permet donc pas de réagir par programme. Elle a cependant l'avantage d'être facile à mettre en œuvre.

Ce sera au programmeur de calculer la complexité de son programme en terme de mémoire utilisée pour s'assurer qu'il fonctionnera...

Attention : Ce travail est à rendre. Vous devez écrire le module *vector* complet tel qu'il est demandé ainsi que fournir les programmes de tests associés.

Le travail sera à rendre par mail à l'adresse m.maachaoui@gmail.com au plus tard le 18/01/2020 à 18h00.

Pour faciliter le traitement des emails merci d'indiquer dans l'objet : ESIEE_VECTOR_VOTRE-NOM.