Programmation C

TP n° 4 : Allocation de mémoire (vecteur dynamique et structures)

Exercice 1 : Vecteur dynamique

Le but de cet exercice est d'implémenter en C un vecteur dynamique d'entiers. On travaillera avec un zone mémoire allouée par `malloc`, d'adresse `vec`. La *capacité* d'un vecteur est son nombre maximal déléments (c'est-à-dire sa taille d'allocation). Sa *taille*, variable au cours du temps, est le nombre courant d'éléments stockés dans le vecteur – à des positions consécutives à partir, de son adresse de base. Implémenter :

1. une fonction `int *creation(long capacite)` qui crée un vecteur d'entiers de capacité initiale `capacite` (implicitement, la taille initiale du vecteur est `0`). Est-ce que le code suivant est correct ? Expliquer pourquoi.

```c
int *creation (long capacite) {
  int vec[capacite];
  return vec;
}
```

2. une fonction `void supprimer(int *vec)` qui supprime le vecteur `vec` en libérant son espace mémoire.
3. une fonction `int *ajouter(int *vec, long *taille, long *capacite, long pos, int val)` qui insère la valeur `val` à la position `pos` du vecteur :
   - la position d'insertion doit être inférieure ou égale à la taille courante du vecteur – utiliser `assert` pour vérifier cette condition ;
   - si la taille courante a déjà atteint la capacité du vecteur, la fonction doit commencer par doubler sa capacité à l'aide de `realloc` – utiliser `assert` pour vérifier le succès de la réallocation (ne pas oublier de mettre à jour `*capacite`). ;
   - tous les éléments du vecteur à partir de la position `pos` sont d'abord décalés d'un cran vers la droite à l'aide de la fonction `memmove` ;
   - enfin, la valeur est écrite à la position spécifiée (ne pas oublier de mettre à jour `*taille`).

   Remarque : Pourquoi la fonction retourne-t-elle `int *` et non pas `void` ?
4. une fonction `void effacer(int *vec, long *taille, long pos)` qui supprime l'élément à la position `pos`, avec un décalage vers la gauche de tous les éléments qui suivent l'élément supprimé. (ne pas oublier de mettre à jour `*taille`).
5. une fonction `void modifier(int *vec, long taille, long pos, int val)` qui remplace une valeur à une position donnée.
6. une fonction `int obtenir(int *vec, long taille, long pos)` qui retourne la valeur stockée à l'indice `pos` du vecteur. Utiliser `assert` pour s'assurer que `pos` est plus petit que `taille`.

Écrire un programme principal qui permet de tester toutes les fonctions – en particulier, augmenter suffisamment la taille du vecteur pour arriver à la limite de la capacité de `tab` et vérifier que la réallocation du vecteur marche bien.

Question bonus : Au petit 3. la capacité du vecteur est doublée lorsqu'il est rempli. Pourquoi avoir choisi cette solution plutôt que d'avoir simplement alloué une case supplémentaire lors du débordement ?

Exercice 2 : Vecteur dynamique structuré

Le premier exercice était consacré aux opérations de gestion d'un vecteur dynamique. Pour maintenir les informations nécessaires au fonctionnement du vecteur ; sa capacité, sa taille et l'adresse de son premier élément, nous avons utilisé des variables indépendantes. Si nous désirions utiliser plusieurs vecteurs dynamiques lors de l'exécution du programme, maintenir trois variables indépendantes par vecteur deviendrait rapidement complexe et alourdirait le code.
Le but de cet exercice est de stocker toutes les informations propres à un vecteur dans une structure. En plus de simplifier le programme, les fonctions de gestion du vecteur ne prendront plus en paramètre qu'un pointeur vers la structure à considérer.
Pour cela, nous allons utiliser les types suivants.

```c
typedef struct {
  long capacite;
  long taille;
  int *t;
} vecteur;
```

1. Ré-implémenter les fonctions de l'exercice 1 en les adaptant au nouveau contexte.

```c
vecteur *creation(long capacite);
void supprimer(vecteur *q);
vecteur *ajouter(vecteur *q, long pos, int val);
void effacer(vecteur *q, long pos);
void modifier(vecteur *q, long pos, int val);
int obtenir(vecteur *q, long pos);
```

2. Comparer les solutions proposées aux deux exercices en allouant un vecteur contenant plusieurs vecteurs d'entiers (représenter une matrice par exemple). Quelle approche vous semble être la plus concise ?