Cahier de TP n° 4

Cahier de TP n° 4

Représentation d'un ensemble : allocation et réallocation dynamique

Exercice 1 : Tableaux

1) Ecrire une fonction d'affichage et une fonction d'affichage inverse d'un tableau d'entiers

   Paramètres :

   - tab pointeur sur le tableau
   - taille_utile : le nombre d'éléments présents dans le tableau

   void AfficheTab(const int * tab, int taille_utile);

   void AfficheInvTab(const int * tab, int taille_utile);

2) Ecrire une fonction d'insertion d'une valeur entière dans un tableau d'entiers, supposé trié et sans doublon.

   La fonction doit conserver le tableau trié et sans doublon.

   Paramètres :

   - tab pointeur sur le tableau (supposé être déjà alloué en amont dynamiquement)
   - taille_utile : le nombre d'éléments présents dans le tableau, qu'il convient de modifier en conséquence
   - taille : la taille maximale du tableau (à ne pas dépasser : gestion des débordements)

     remarque : pour le moment, on n'essaiera pas de comprendre pourquoi on utilise un pointeur pour la taille, alors que la taille ne doit pas être modifiée! Cela paraît paradoxal ….

   - valeur : la valeur à insérer, s'il n'est pas déjà présente (pas de doublon), l'ordre est conservé

   int InsertionTrieeTab(int ** tab, int * taille_utile, int * taille, int valeur);

3) Ecrire une fonction d'insertion d'une valeur entière dans un tableau d'entiers, supposé trié et sans doublon.

   La fonction doit conserver le tableau trié et sans doublon.

   Paramètres :

   - tab pointeur sur le tableau, supposé être déjà alloué dynamiquement en amont
   - taille_utile : le nombre d'éléments présents dans le tableau, , qu'il convient de modifier en conséquence
   - taille : la taille maximale du tableau à mettre à jour en cas d'aggrandissement du tableau
   - valeur : la valeur à insérer, s'il n'est pas déjà présente (pas de doublon), l'ordre est conservé

   int InsertionTrieeTabDynamique(int ** tab, int * taille_utile, int * taille, int valeur);

   Fonctionnalité importante :

   Cette insertion doit être dynamique : le tableau étant déjà alloué dynamiquement, il faut le **réallouer dynamiquement** par un incrément donné lorsqu'il est plein. Le tableau est donc aggrandi.

   Pour cela on définit :

   /** Le pas d'incrémentation pour une réallocation dynamique d'un tableau */

   #define INCREMENT 10

Rappel de cours :

void* realloc (void* ptr, size_t size);

Reallocate memory block

Changes the size of the memory block pointed to by *ptr*.

The function may move the memory block to a new location (whose address is returned by the function).

The content of the memory block is preserved up to the lesser of the new and old sizes, even if the block is moved to a new location. If the new *size* is larger, the value of the newly allocated portion is indeterminate.

In case that *ptr* is a null pointer, the function behaves like malloc, assigning a new block of *size* bytes and returning a pointer to its beginning.

If the function fails to allocate the requested block of memory, a null pointer is returned, and the memory block pointed to by argument *ptr* is not deallocated (it is still valid, and with its contents unchanged).

Parameters

ptr

Pointer to a memory block previously allocated with malloc, calloc or realloc.
Alternatively, this can be a *null pointer*, in which case a new block is allocated (as if malloc was called).

size

New size for the memory block, in bytes.
size_t is an unsigned integral type.

Exercice 2 : Ensemble

On représente un ensemble par une structure contenant :

- le cardinal de l'ensemble
- la taille du tableau
- un tableau alloué dynamiquement (les éléments de l'ensemble)

```
typedef struct
{
   int cardinal;
   int taille;
   int * elements;
} Ensemble;
```

Un ensemble est initialisé ainsi : **Ensemble e = {0, 0, NULL};**

1) Ecrire une fonction d'affichage et une fonction d'affichage inverse d'un ensemble

   void AfficheEnsemble(const Ensemble * ensemble);

   void AfficheInvEnsemble(const Ensemble * ensemble);

2) Ecrire une fonction de création d'un ensemble demandant à l'utilisateur de saisir les éléments de l'ensemble.

Si l'ensemble est plein alors la valeur saisie n'est pas prise en compte : pas d'insertion possible

La fonction doit appeler une des fonctions d'insertion déjà créees.

int CreationEnsemble(Ensemble * ensemble, int taille);

Exercice 3 : Ensemble dynamique

3) Ecrire une fonction de création d'un ensemble demandant à l'utilisateur de saisir les éléments de l'ensemble.

Si l'ensemble est plein alors la valeur saisie est bien prise en compte : insertion possible

La fonction doit appeler une des fonctions d'insertion déjà créees.

int CreationEnsembleDynamique(Ensemble * ensemble, int taille);

4) Ecrire une fonction d'ajout d'un élément à un ensemble dynamique

void Ajout(Ensemble * ensemble, int valeur);

Exercice 4 : Pointeurs de fonctions

1) Les 2 fonctions de création d'ensemble ont le même code sauf en un point. Lequel?
2) Par suite, afin de n'avoir qu'une seule fonction de création d'ensemble paramétrée par le traitement d'insertion, le recours à un pointeur de fonction est nécessaire.

a) Définir un ***nouveau type PF_INSERTION*** en accord avec les déclarations prototypes des 2 fonctions d'insertion.

b) Définir une fonction unique de création d'ensemble paramétrée avec le pointeur de la fonction d'insertion :

int CreationEnsemble_ChoixInsertion(Ensemble * ensemble, int taille, PF_INSERTION pf);

Exercice 5 : Lecture et écriture dans un fichier

1) Ecrire un programme qui lit un fichier d'entiers, et recopie dans un autre fichier les entiers lus à raison d'un entier par ligne (tous les caractères d'espacement sont ignorés). Les noms de fichiers sont saisis par l'utilisateur.

2) Modifiez le programme pour que les noms des fichiers soient transmis sur la ligne de commande.

Exercice 6 : Ensembles

On va maintenant ajouter aux fonctions de manipulations d'ensemble, la lecture et l'écriture d'un ensemble dans un fichier.

1) Ecrire une fonction SauveEnsemble(const Ensemble * ens, FILE * sortie) qui sauvegarde dans un fichier les éléments d'un ensemble. L'ensemble et le fichier seront fournis en argument de la fonction.
Le format de sauvegarde est :

Un entier par ligne, le premier entier est le nombre d'élément de l'ensemble.

2) Ecrire une fonction int LitEnsemble(Ensemble * ens, FILE * entree) qui construit l'ensemble à partir des données du fichier fourni en argument.
Effectuer les tests nécessaires pour s'assurer que le fichier lu a le bon format, c'est-à-dire le même format que celui utilisé dans la fonction précédente.
Cette fonction retournera 0 en cas de problème et 1 sinon.