Algorithmique et Programmation 1
IMAC 1ere année

TP 9

allocation dynamique

Dans cette séance de travaux dirigés, on travaillera sur l'allocation dynamique.

Pendant le TP, vous écrirez vos codes dans des fichiers source. Ayez le réflexe d'enregistrer régulièrement vos sources. À la fin de la séance, mettez-les dans une archive (commande `tar -zcvf nom_archive fichiers_source`) et rendez-le suivant les instructions données sur e-learning. N'oubliez pas de commenter votre code.

Exercice 1. (Fiches d'étudiants)

Dans ce TP, on poursuit le dernier exercice de la semaine précédente sur les fiches d'étudiants.

1. Reprendre le type structuré `Date` ainsi que les deux fonctions `ecrire(Date date)` et `lireDate(Date *date)`.
2. Reprendre le type structuré `Fiche` possédant les champs `char nom[20]`, `char prenom[20]`, `Date dateDeNaissance`, `float notes[MAXNOTES]` et `int nbNotes`.
3. Reprendre les fonctions `lireFiche(Fiche *fiche)`, `ecrireFiche(Fiche *fiche)` et `ajouteNote(Fiche *fiche, float note)`.

Exercice 2. (Classes)

Dans cet exercice, on va définir un type structuré modélisant une classe d'étudiants.

1. Définir une structure

```
1  typedef struct{
2    Fiche *fiches;
3    int taille;
4    int capacite;
5  }Classe;
```

   Le champ `fiches` est un pointeur sur la `fiche` du premier étudiant de la classe. On utilisera ce pointeur comme tableau dynamique. Les champ `taille` et `capacite` correspondent à la taille de la classe et à sa capacité maximale, respectivement.

2. Dans cette question, on définit la fonction de lecture d'une classe en décomposant le problème en différentes étapes.
   (a) En utilisant la fonction `malloc`, définir une fonction `initialiseClasse(Classe *classe)` initialisant une classe : les champs `taille` et `capacite` sont scannés, l'allocation mémoire pour contenir la capacité maximale de la classe est réalisée mais les fiches ne sont pas lues.

(b) Si vous ne l'avez pas déjà fait, modifiez la fonction `initialiseClasse` pour qu'elle retourne un code d'erreur si l'allocation n'a pas fonctionné.

(c) Définir une fonction `dejaPresente(Classe classe, Fiche fiche)` qui regarde si `fiche` correspond à un étudiant déjà présent dans la classe.

(d) En utilisant les deux fonctions précédentes, définir la fonction `lireClasse(Classe *classe)`. On recommence la lecture d'une fiche tant qu'elle correspond à un étudiant déjà présent dans la classe. La fonction doit retourner un code d'erreur si l'allocation de la mémoire n'a pas fonctionné.

3. Définir la fonction `ecrireClasse(Classe classe)` qui écrit les fiches de chaque étudiant de la classe.
4. Définir une fonction `ajouterFiche(Classe *classe, Fiche fiche)` qui ajoute `fiche` dans une classe. Si la capacité maximale est atteinte, il faut réallouer de la mémoire. Il faut gérer les erreurs éventuelles à l'allocation et retourner un code d'erreur si la fiche correspond à un étudiant déjà présent dans la classe.
5. Définir une fonction `supprimerFiche(Classe *classe, Fiche fiche)`. Gérer le cas où la fiche n'est pas dans la classe.