TRAVAUX PRATIQUES VII

Tableaux dynamiques

L'objectif de ce TP est de coder en C une librairie de tableaux dynamiques, afin de s'en resservir à l'avenir.
Appelez-moi si quoi que ce soit n'est pas clair !

Remarque. Mon colorieur de code souligne certains termes dans ce type. Ignorez les soulignements (ils induisent en erreur, je dois les enlever...)

A Enregistrements

Comme nous l'avons vu en cours, un tableau dynamique est composé de 3 choses : le tableau en lui-même, son nombre de cases utilisées, et son nombre de cases existantes. Ainsi, si nous voulons les coder... il faudra manipuler 3 variables pour représenter 1 seule chose (1 seul tableau dynamique). Pas vraiment pratique.

Ce que l'on aimerait, c'est avoir une variable de type « tableau dynamique » qui contienne trois « sous-variables » : ça tombe bien, ça existe ! On appelle cela un **type enregistrement**, ou de manière familière une **struct** . Les « sous-variables » sont appelées des **champs** :

Définition 1 (Type enregistrement en C).
La syntaxe pour définir un type enregistrement en C est la suivante :

```c
struct nom_s {
  type0 identifiant_champ_0;
  ...
  typeK identifiant_champ_K;
}; // notez le ; après l'accolade
```

Attention, cela créé un type nom `struct nom_s` et non simplement `nom_s` .
Une structure est passé par valeur (sauf pour ses champs qui doivent s'affaiblir en pointeur, comme les tableaux).
En particulier, on peut renvoyer une structure déclarée localement.

Remarque.

- Les différents champs peuvent tout à fait être de types différents.
- La création d'un type ne peut pas se faire dans une fonction.
- Si vous avez fait NSI Terminale, les struct sont l'ancêtre des classes : que des attributs, pas de méthodes (ni héritage).

Par exemple, voici comment déclarer un type enregistrement nommé `struct pc_s` qui contient 3 champs : `physique` , `chimie` , `mathematiques` (tous trois de type `double`) :

```c
struct pc_s {
  double physique;
  double chimie;
  double mathematiques;
};
```

Une fois ce type créé, on peut crée des variables de ce type. On peut donner une valeur à leurs champs : si la variable s'appelle `var` , pour accéder au contenu de son champ `ch` on utilise `var.ch` :

```c
// Je vais créer une variable nommée viviane, de type struct pc_s,
// dont les champs contiennent :
//    physique : 15.6 ; chimie : 18.9 ; mathématiques : 22.5
struct pc_s viviane;
viviane.physique = 15.6;
viviane.chimie = 18.9;
viviane.mathematiques = 22.5;

// Je vais maintenant modifier le champ mathematiques de viviane :
viviane.mathematiques = 24.8;
```

Remarque. Il est peu agréable d'avoir un type nommé `struct pc_s` : on aimerait juste `pc` . Il existe un moyen de donner un nom supplémentaire à un type :

Définition 2 (typedef).
On peut donner un nouveau nom supplémentaire à un type via la syntaxe :

```c
typedef id_type nouveau_id_type;
```

Exemple. Pour renommer `struct pc_s` en juste `pc` , on peut faire : `typedef struct pc_s pc;`

1. Modifiez `partie-A/main.c` pour :
   - Y créer un type enregistrement nommé `struct mp2i_s` , ayant pour champs 3 entiers : `physique` , `informatique` , `sciences_industrielles` .
   - Doner un nouveau nom à ce type : `mp2i` .
   - Modifier les champs de la variable `toto` .

Définition 3 (Initialisateur de struct).
On peut utiliser un initiasateur pour créer une `struct` comme suit :

```c
struct id_s variable = {.champ0 = val0, .champ1 = val1, ..., .champK = valK};
```

Cela crée une variable nommée `variable` dont le champ `champ0` contient la valeur `val0`, le champ `champ1` contient la valeur `val1`, etc.

Exemple.

```c
// Je vais créer une variable nommé joel, de type struct pc, dont les champs contiennent :
//  physique : 15.6; chimie : 18.9; mathématiques : 22.5
struct pc_s joel = {.physique = 15.6, .physique = 18.9, .mathematiques = 22.5};
```

B Tableaux dynamiques

Nous allons coder des tableaux dynamiques. Voici le type enregistrement que nous utiliserons pour les représenter :

dynArray.h

```c
/** Structure d'un tableau dynamique */
struct dynArray_s {
  int* arr;          // pointeur vers la zone contenant les cases
  unsigned len;      // nombre de cases utilisées de arr
  unsigned len_max; // nombre de cases de arr
};
typedef struct dynArray_s dynArray;
```

Dans le dossier `partie-B/` , vous trouverez :

- `dynArray.h` , un fichier header qui décrit les différentes fonctions que vous devez coder. **Vous ne devez pas modifier ce fichier.**
- `test.c` , un fichier contenant des tests de tableaux dynamiques et un main qui les lance.

Vous noterez que toutes les fonctions qui prennent en argument un tableau dynamique prennent en fait un *pointeur* vers un tableau dynamique : c'est voulu, plusieurs de ces fonctions devant avoir comme effet secondaire de modifier certains champs. Vous ferez attention à ne pas oublier de déréférencer lorsqu'il le faut.

2. Dans `partie-B/`, créez `dynArray.c` : c'est là que vous coderez les tableaux dynamiques.

   Commencez le fichier par `#include "dynArray.h"` : cela inclut le header, lequel contient les autres inclusions dont vous avez besoin (`stdlib` par exemple). Ainsi, l'inclusion du header est la seule inclusion que `dynArray.c` a besoin de faire.

   De même, comme cela inclut le header, la définition du type `dynArray` est incluse : vous n'avez pas à la recoder.
3. Codez les fonctions décrites dans le header, les unes après les autres.

   Le testeur contient des tests de toutes les fonctions. N'hésitez pas à le modifier pour passer en commentaire certains tests, ou pour rajouter des tests, afin de pouvoir tester votre code au fur et à mesure.

   NB : Je conseille simplement de garder `dynarray_of_array` pour la fin.

Remarque. Vous remarquerez que vous écrirez souvent `(*ptr).ch` . À la longue, c'est agaçant. Là aussi, il existe une syntaxe pour raccourcir cela :

Définition 4 (Déréférécement+accès à un champ).
La syntaxe `ptr->ch` est un raccourci pour `(*ptr).ch`