TP n°11 - Types personnalisés en C.

On rappelle que pour pouvoir exécuter les programmes compilés sur les machines du lycée, votre répertoire de travail doit être situé en dehors du répertoire `Documents`

Il est possible de récupérer des informations tapées au clavier par l'utilisateur en cours d'exécution grâce à la commande `scanf`.
Celle-ci fonctionne comme la fonction de lecture d'un fichier `fscanf` sauf que le flux de données ne vient pas d'un canal ouvert depuis un fichier, mais de l'entrée standard appelée `stdin` (*standard input*), à savoir, par défaut, le clavier, ce canal étant automatiquement ouvert.
Ainsi, la fonction `scanf` est équivalente à la fonction `fscanf` pour laquelle on a fixé le premier argument à `stdin` : `fscanf(stdin,...)`

Exercice 1 (Interactions avec l'utilisateur).
On considère le jeu de devinette suivant. Il se joue de la manière suivante : un maître du jeu choisit secrètement un entier aléatoire entre 0 et 100 (inclus). Le joueur cherche à deviner ce nombre. A chaque essai, le maître du jeu lui indique s'il a trouvé le nombre mystère et, si ce n'est pas le cas, lui indique si le nombre mystère est plus grand ou plus petit que celui proposé. Le jeu s'arrête lorsque le joueur a trouvé le nombre mystère. Le but du jeu est bien sûr de trouver le nombre mystère avec le moins d'essais possibles. A la fin du jeu, le maître du jeu indique au joueur le nombre d'essais qu'il a dû faire avant de trouver le nombre mystère.

Coder ce jeu de devinette dans un fichier source `NOM_devinette.c`, la machine jouant le rôle de maître du jeu et l'utilisateur le rôle du joueur.

Nous avons vu en OCaml qu'il était possible de définir un nouveau type personnalisé avec le mot-clé `type` :

Type somme : le type est défini en décrivant les différentes alternatives permettant de construire des objets de ce nouveau type à l'aide de constructeurs (constructeurs avec ou sans paramètres)

Type produit : n-uplets ou enregistrement (produit nommé, les champs sont repérées par un nom et non plus seulement par leur position)

Bien entendu, il est également possible de définir de nouveaux types personnalisés en C.

Type somme en C. Il n'y a pas d'équivalent aussi riche que les types somme OCaml, qui décrivent les alternatives sous forme de constructeurs. En C, on parle d'énumération : il s'agit simplement de permettre de créer un type regroupant une simple énumération finie de possibilités (couleurs, jours de la semaine par exemple...), chaque possibilité étant désignée par un identificateur (un nom) choisi. On peut aussi voir le type somme en C comme un type somme OCaml qui n'autoriserait pas l'utilisation de constructeurs avec paramètres. Voici un exemple ci-dessous :

```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

// définition d'un type groupe par énumération
// équivalent très appauvri du type somme OCaml
// en fait, derrière, ce sont des entiers (4 octets)
// MPII est un alias pour l'entier 0, MPSI1 un alias pour l'entier 1...etc
enum groupe {MPII, MPSI1, MPSI2};
```

Pour créer un type somme (type énuméré), on utilise le mot clé `enum`.

- Le programmeur donne un nom au type regroupant les différentes alternatives. Ici, le nom du type énuméré est `groupe`
- Le programmeur attribue un identificateur de son choix à chacune des alternatives énumérées. L'usage veut que ces identificateurs d'énumérations commencent par une majuscule. Ici, il y a seulement 3 alternatives énumérées : `MPII`, `MPSI1`, `MPSI2`

Remarque. Il est important de comprendre que, derrière ces noms explicites et parlant choisis par le programmeur, se cachent en fait des entiers. Le compilateur associera simplement l'identificateur `MPII` à l'entier 0, l'identificateur `MPSI1` à l'entier 1...etc par ordre d'écriture entre les accolades.
Les énumérations servent donc surtout à améliorer la lisibilité du code, en évitant d'avoir à se rappeler que l'on a identifié la `MPII` par l'entier 0, la `MPSI1` par l'entier 1...etc .

Types produits en C. En C, il n'existe pas de n-uplets à proprement parler. Il n'existe que des types produits nommés comme les enregistrements OCaml. Ce n'est absolument pas un problème car les types produits nommés apportent une flexibilité en plus par rapport au n-uplets : il est donc possible de retrouver toutes les possibilités des n-uplets avec des enregistrements. Voici un exemple de définition d'un nouveau type produit nommé en C :

```c
// définition du type appelé "struct s_eleve", qui possède 4 champs
// équivalent du type produit enregistrement
struct s_eleve
{
  char nom[256]; // 256 octets
  int age;       // 4 octets
  enum groupe classe; // 4 octets
  float notes[3];     // 3*4 octets
};

typedef struct s_eleve eleve;
// cela permet d'appeler notre type "eleve", au lieu de  "struct s_eleve",
//qui est plus long et lourd à écrire
```

Pour créer un type produit nommé en C, on utilise le mot-clé `struct`. Ici, le nouveau type crée s'appelle `struct s_eleve`. Il possède 4 champs :

- un champ nommé `nom` qui est un tableau de 256 caractères (256 octets)
- un champ nommé `age`, de type entier (4 octets)
- un champ nommé `classe` qui a pour type le type personnalisé `enum groupe` défini au-dessus (4 octets)
- un champ nommé `notes` qui est un tableau de 3 nombres flottants stockés sur 4 octets (12 octets)

Chaque variable de type `struct s_eleve` nécessite donc au moins 276 octets de mémoire pour être stockée. [1]

Le mot-clé `typedef` permet de donner un **alias**, c'est-à-dire de créer un nom supplémentaire, plus simple, pour désigner le type `struct s_eleve`. Ici, on définit un deuxième nom plus pratique, `eleve`, pour désigner le type `struct s_eleve`. Grâce à cette ligne de code, on pourra donc dans la suite du code utiliser indifféremment `struct s_eleve` ou `eleve` pour désigner notre nouveau type.

Le code ci-dessous montre comment déclarer des variables de ce nouveau type, et comment affecter des valeurs aux différents champs avec le `.`, comme cela se fait avec les enregistrements en OCaml.

```c
int main(void)
{
  printf("Un objet de type enum groupe occupe %ld octets\n", sizeof(enum groupe));

  eleve e1 ;  // déclaration d'une variable de type "eleve"
              // dans le bloc d'activation du main, dans la pile

  // affectation des valeurs des différents champs
  strcpy(e1.nom, "Marque");
  e1.age = 18;
  e1.classe = MPII;
  e1.notes[0] = 19;
  e1.notes[1] = 17;
  e1.notes[2] = 19.5;

  // autre manière d'initialiser un objet de type eleve
  // (toujours stocké dans la pile)
  eleve e2 = {"Kitten", 18, MPII, {13.5, 18.5, 19.75} };

  // le point "." sert à accéder aux différents champs de l'objet,
  // comme en OCaml
  printf("%d\n", e2.age);
  printf("%f\n", e2.notes[1]);
```

Attention. Par défaut, en C, tous les champs sont mutables, c'est l'inverse de ce qui se passe en OCaml. [2]

Enfin, on peut tout à fait définir des pointeurs pointant vers des variables de ce nouveau type.

1. Parfois, pour des raisons de stockage mémoire (mécanisme de *padding* pour garantir l'alignement mémoire), le nombre d'octet occupé par un type peut-être plus grand que la somme des octets nécessaires à chaque champ.
2. On rappelle qu'il faut rajouter le mot-clé `mutable` en OCaml devant le champ lors de la déclaration du type enregistrement pour rendre ce champ mutable.

```c
// allocation d'une variable de type eleve dans le tas
eleve *pe = NULL; // l'objet sera stocké dans le tas, mais le pointeur,
                  // qui contiendra l'adresse de l'objet, est lui stocké
                  // dans la pile, dans le bloc d'activation du main
pe = malloc(sizeof(eleve));
printf("Un objet de type eleve occupe %ld octets\n", sizeof(eleve));


// acces aux champs à partir du POINTEUR avec ->
pe->age = 19;
strcpy(pe->nom, "Misplon");
pe->classe = MPII;

printf("%s\n", pe->nom);

if (pe != NULL)
  free(pe);
pe = NULL;

return 0;
}
```

Dans ce cas, on peut utiliser la syntaxe avec `->` pour accéder directement aux champs de l'objet depuis le pointeur. En fait, `pe->age` désigne la même variable (même zone mémoire) que `(*pe).age`

Exercice 2 (Types structurés en C).
Le code C de cet exercice sera enregistré dans un fichier `NOM_geom.c`

1. Dans le fichier `NOM_geom.c`, définissez un type `point` permettant de stocker les informations relatives à un point 2D : nom du point (2 caractères max) et coordonnées. Compiler et faire de petits tests à la main dans la fonction principale pour valider la création du nouveau type.
2. Écrire une fonction qui demande 3 points à l'utilisateur en ligne de commande et les stocke dans un tableau. Écrire une fonction qui affiche un tableau de points pour pouvoir tester et valider votre fonction.
3. Écrire une fonction qui calcule l'aire du triangle formé par les trois points du tableau. On rappelle que :
$$\mathcal{A}(ABC) = \frac{1}{2}\left|\det\left(\overrightarrow{AB}, \overrightarrow{AC}\right)\right| = \frac{1}{2}\left|\det\left(\overrightarrow{BA}, \overrightarrow{BC}\right)\right| = \frac{1}{2}\left|\det\left(\overrightarrow{CA}, \overrightarrow{CB}\right)\right|.$$
Tester votre fonction sur des triangles simples dont l'aire est facile à calculer, et faire afficher le résultat dans le `main`.

Définition 1 (Format CSV (*comma-separated values*))

Le format CSV (*comma-separated values*) est un format très simple permettant de stocker des informations tabulées dans un fichier texte. Un fichier au format CSV est conçu de la manière suivante :

- chaque ligne du fichier correspond à une ligne du tableau
- chaque ligne est formée de différents champs, séparés par un caractère appelé séparateur, généralement une virgule

Ce format est reconnu pour la quasi-totalité des logiciels de calcul de type tableurs (`Libre Office Calc`, `Excel`, ...etc)

Exercice 3 (I/O et types personnalisés.).
Le code C de cet exercice sera enregistré dans un fichier `NOM_anniversaire.c`

1. Sur Moodle, sous l'énoncé de ce TP, un fichier nommé `MPII.csv` est mis à votre disposition. Il a été exporté depuis le logiciel `Pronote`, qui connaît lui aussi le format CSV. Téléchargez ce fichier (clic droit, enregistrer la cible du lien...) et utilisez les commandes Linux dans le Terminal pour déplacer ce fichier dans votre répertoire de travail.
2. Ouvrir ce fichier avec `emacs` puis avec `Libre Office Calc` et observez les informations qu'il contient et comment elles sont structurées.
3. Dans le fichier `NOM_anniversaire.c`, écrire une fonction `decouper_chaine` qui prend en entrée une chaîne de caractères et un caractère considéré comme séparateur et qui retourne le nombre de sous-chaînes séparées par le caractère séparateur et un tableau contenant les sous-chaînes de caractères, dans le bon ordre. Par exemple, si la chaîne donnée en entrée est `toto;tutu;12;2.5` et que le caractère séparateur est le point-virgule, alors la fonction retourne un tableau dont la première case contient la chaîne de caractère `''toto''`, la seconde `''tutu''`, la troisième `'12''` et la dernière case `''2.5''`. **On considère dans tout le code qu'il ne peut pas y avoir plus de 5 sous-chaînes de caractères. On ne fera aucune copie de chaîne de caractères, et on ne manipulera que des pointeurs.**
4. Tester votre fonction sur différentes chaînes et différents séparateurs directement dans le `main`. Testez-la également sur l'une des lignes du fichier `MPII.csv`.
5. Créer un (ou plusieurs !) types personnalisés permettant de stocker de manière structurée les informations de chaque élève. Compiler et tester votre nouveau type dans le `main` avec des valeurs de votre choix pour les différents champs.
6. Écrire une fonction permettant de récupérer toutes les informations stockées dans ce fichier sous la forme d'un tableau pour lequel chaque case correspond aux informations pour un élève. *Indication : réfléchissez à toutes les fonctions dont vous allez avoir besoin et pensez l'architecture de votre code. Compilez et déboguez dès que possible il y a plusieurs possibilités d'implémentation, nous discuterons des avantages et des inconvénients de chacune.*
7. Écrire une fonction `anniversaire` qui prend en entrée le numéro d'un mois de l'année et qui affiche à l'écran le nom et le prénom de tous les élèves dont c'est l'anniversaire ce mois-là, ainsi que le jour de leur anniversaire.
8. Tester votre code sur les fichiers des MPSI1 et MPSI2, disponibles sur Moodle également.

*Nous venons de constituer un embryon de **base de données** et quelques fonctions sur cette base de données, mais elle n'est pas du tout optimisée ! Mais ici, cela n'a aucune importance car nous avons tout au plus quelques dizaines d'enregistrements dans notre base de données. Nous verrons au second semestre des stockages sous forme de tables de hachage qui permettent d'accéder très rapidement à l'ensemble des enregistrements vérifiant les critères souhaités par l'utilisateur dans de grosses bases de données.*