Programmation C
TP 10

Allocation, `void*`, généricité...

Exercice 1 Échange générique

Dans un ordinateur et sur les matériaux supports de la mémoire, les données sont binaires. C'est un fait, même si la couche logiciel fait tout son possible pour rendre l'utilisation d'un ordinateur quelque chose de simple. Primitivement, les informations sont codées par des bits et le langage C, bas niveau, peut permettre de creuser un peu pour manipuler ces données brutes.

Voici une fonction en C échangeant deux entiers :

```
1 void swap_integer(int* a, int* b){
2   int tmp;
3
4   tmp = *a;
5   *a = *b;
6   *b = tmp;
7 }
```

En C, comme le code source est relativement fortement typé statiquement, il faudrait écrire une fonction d'échange pour chaque type manipulé. Au lieu de ça, si on considère les données comme binaires, on va pouvoir échanger n'importe quel type en les considérant seulement comme des zones mémoires.

Écrire une fonction `void swap_mem(void* z1, void* z2, size_t size)` qui échange les données binaires contenues dans les zones `z1` et `z2`. Tester votre fonction sur des entiers, des doubles, des tableaux voire des structures de votre choix.

Astuce : il faut lire les zones mémoires caractères par caractères (c'est à dire octets par octets) en utilisant de bons casts...

Exercice 2 Liste chaînée de structure

Télécharger à partir de la plate-forme elearning le fichier `liste_nom.c`. Ce fichier contient ligne par ligne des données relatives à des personnes fictives : leur prénom, leur nom puis leur âge. Pour manipuler ces personnes, nous allons utiliser la structure chaînée suivante :

```
1 typedef struct cell{
2   char* first_name;
3   char* last_name;
4   int age;
5   struct cell* next;
6 }Cell, *List;
```

L'objectif est de trier ce fichier de personnes selon certains critères d'ordre. Pour cela, nous allons entièrement parser le fichier, fabriquer une cellule pour chaque personne rencontrée puis en manipulant des listes chaînées triées et en insérant de manière triée, nous allons reconstituer une liste chaînée de toutes les données triées selon les critères voulus.

On voudra trier selon deux manières : tout d'abord par l'âge puis alphabétiquement pour les personnes ayant le même âge. On essayera ensuite de trier alphabétiquement puis par âge pour les personnes ayant les mêmes nom et prénom.

Voici une liste de prototypes de fonction raisonnables pour mener à bien cet exercice.

- `Cell* allocate_cell(char* first, char* last, int age)`
- `int age_order(Cell* p1, Cell* p2)`
- `int name_order(Cell* p1, Cell* p2)`
- `void ordered_insertion(List* l, Cell* new, int order_func(Cell*, Cell*))`
- `void print_list(List l)`
- `void free_list(List l)`

La fonction `ordered_insertion` possède pour un de ses arguments, une autre fonction. Cela s'appelle en C un pointeur de fonction. Ici, la fonction `order_func` est considérée comme une variable, son type est décrit par le type de ses arguments et son type de retour. C'est une des solutions pour essayer de produire du code générique en C (langage absolument pas conçu pour faire des choses génériques). Remarquez que les deux fonctions `age_order` et `name_order` prennent le même nombre d'argument avec les mêmes types et retournent le même type, ce sont ces deux fonctions qui ont vocation à être données à la fonction assurant l'insertion triée.