Travaux Dirigés de programmation n°3

Cours de Programmation C

—L2.1—

▸ **Exercice 1.** *Utilisation de* `void *`

1. *Expliquer ce que fait la fonction suivante :*

```
void A(void *a, size_t nbBytes) {
    char *ca = (char *)a;

    int i;

    for(i=0; i<nbBytes; i++) {
        ca[i] = 0
    }

}
```

2. *Écrire une fonction* `main` *qui utilise la fonction* `A` *sur un entier*
3. *Écrire une fonction* `main` *qui utilise la fonction* `A` *sur un tableau d'entiers.*
4. *En s'inspirant de la fonction* `A`*, écrire une fonction* `void Copie(void *a, void *b, size_t n)` *qui copie les données de* `*a` *vers* `*b` *sur* `n` *bytes.*

▸ **Exercice 2.** *Ensemble non typé*

On souhaite créer une structure `Ensemble` *pour représenter en machine des ensembles d'élements dont le type n'est pas connu à l'avance. Une structure* `Ensemble` *doit contenir une zone de mémoire allouée pour recevoir les éléments (le pointeur vers la zone sera de type* `char *`*), le nom du type d'élément qu'elle contient (exemple : "int"), un entier représentant sa capacité maximum et un entier comptant le nombre déléments qu'elle contient. Si en ajoutant un élément, on dépasse la capacité de l'ensemble alors cette dernière doit être augmentée.*

Donner la structure à utiliser puis écrire les fonctions :

1. `Ensemble * CreeEnsemble(int capacite, size_t tailleElts, char *type)` *qui crée un ensemble vide de capacité initiale* `capacite` *pour des éléments de taille* `tailleElts` *bytes et de type* `type`*.*
2. `void LibereEnsemble(Ensemble *e)`
3. `int AugmenteCapacite(Ensemble *e)` *qui multiplie par deux la capacité obtenue et retourne* `1` *en cas de succès et* `0` *en cas d'échec.*

4. `int AjouteElement(Ensemble *e, void *elt)` *qui ajoute une copie de l'élément pointé par* `elt` *et retourne 1 en cas de succès et 0 en cas d'échcec. On ne traitera pas le cas des doublons (un ensemble peut contenir plusieurs fois le même élément). Vous pouvez utiliser les fonctions du premier exercice.*
5. `int SupprimeElement(Ensemble *e, int i)` *qui supprime de l'ensemble l'élément placé en ième position.*
6. `void Concat(Ensemble *e1, Ensemble *e2)` *qui ajoute les éléments de* `e2` *à* `e1` *seulement si les deux ensembles déclarent contenir le même type d'élélments.*

▸ **Exercice 3.** ***A faire chez soi*** *Problème d'égalité*

1. *Copier le programme suivant, le compiler et le tester :*

```
typedef struct {
    char lettre;
    int n;
}essai;

int main(void) {
    printf("taille reelle : %u\n",sizeof(essai));
    printf("taille theorique : %u \n",sizeof(char)+sizeof(int));
    return 0;

}
```

   Quels résultats obtenez-vous ? D'où vient la différence entre les deux valeurs ?

2. *On a écrit la fonction suivante :*

```
int Egal(void *a, void *b, size_t nbBytes) {
    char *ca = (char *)a;
    char *cb =(char *)b;
    int i;

    for(i=0; i<nbBytes; i++) {
        if (ca[i]!=cb[i]) return 0;
    }

    return 1;
}
```

   Expliquez ce que fait cette fonction.

3. *À présent, on teste* `Egal` *avec la fonction* `main` *suivante :*

```
int main(void) {

    int a=4,b=4;
```

```
    essai e1,e2;
    e1.lettre = 'a';
    e2.lettre = 'a';
    e1.n = 5;
    e2.n = 5;
    printf("%d\n", Egal(&a,&b, sizeof(int)));
    printf("%d\n", Egal(&e1,&e2, sizeof(essai)));

    return 1;
}
```

Copier et tester cette fonction, la fonction `Egal` *retourne-t-elle le résultat voulu ? Expliquer le problème qui se pose lorsque l'on teste des structures octet par octet.*

▸ **Exercice 4.** *Matrices d'entiers*

On souhaite travailler avec des matrices d'entiers. Deux structures sont proposées :

```
typedef struct matrice1 {

    int lignes;
    int cols;
    int * nombres;
}Matrice1;

typedef struct matrice2 {
    int lignes;
    int cols;
    int ** nombres;
}Matrice2;
```

Dans `Matrice1`*, les nombres sont représenté par un tableau simple de taille* `lignes` $\times$ `cols`*. Dans* `Matrice2`*, les nombres sont représentés par un double tableau. Pour chacun des deux types, écrire les fonctions :*

1. `Matrice * CreeMatrice(int lignes, int cols)` *qui crée une matrice où toutes les valeurs sont initialisées à zéro.*
2. `void LibereMatrice(Matrice *m)`
3. `int Get(Matrice *m, int l, int c)` *qui retourne la valeur ligne* `l`*, colonne* `c` *de la matrice* `*m`*.*
4. `void Change(Matrice *m, int l, int c, int val)` *qui modifie la valeur ligne* `l`*, colonne* `c`*.*
5. *Expliquer pourquoi la première instruction ci-dessous est correcte et pas la deuxième :*

```
int tableau[] = { 1 , 5 , 45 , 3 , 9 };
int identite[][] = { { 1 , 0 , 0 } , { 0 , 1 , 0 } , { 0 , 0 , 1 } };
```

Comment corriger ?

6. *Pourquoi est-il obligatoire de spécifier la taille de la deuxième dimension d'un tableau statique lors du passage en argument dans une fonction ?*