

TP 4 : Pointeurs et structures

Semaine du 19 février 2007

Dans tout le TP, *chaque fonction écrite devra impérativement être testée* avant de passer à l'exercice suivant.

1 Introduction

Exercice 1 Exécutez le programme suivant :

```
int *toto(void) {
    int x = 42;
    return &x;
}

void tata(void) {
    int a = 23;
}

int main(int argc, char *argv[]) {
    int *y = toto();
    tata();
    printf("%d\n", *y);
    exit(0);
}
```

Qu'observez-vous ? Qu'en concluez-vous ?

2 Matrices

Dans cette section, on reprendra la structure matrice (peut-être déjà) vue en TD.

```
struct matrice { int lignes, colonnes; double **coefficients; };
```

Exercice 2 Écrivez une fonction **void** affiche_matrice(**struct** matrice A) qui affiche une matrice A ligne par ligne.

Exercice 3 Écrivez une fonction **void** affiche_matrice2(**struct** matrice *A) qui affiche une matrice *A ligne par ligne. Quelles sont les différences avec la fonction précédente ?

Exercice 4 Écrivez une fonction **struct** matrice *alloue_matrice(int colonnes, int lignes) qui alloue une matrice.

Ajoutez de l'affichage permettant d'observer quels sont les pointeurs créés (à chaque utilisation de malloc()). Utilisez le format %p dans printf() pour afficher un pointeur.

Exercice 5 Écrivez une fonction **void** libere_matrice(**struct** matrice *A).

(R)Appel : on doit appeler la fonction free() sur toutes les lignes de la matrice, puis sur le tableau contenant les pointeurs vers les lignes, puis enfin sur la structure elle-même.

Ajoutez de l'affichage permettant d'observer quels sont les pointeurs détruits (à chaque utilisation de `free()`).

Vérifiez que vous avez bien détruit tous les pointeurs créés par `alloue_matrice()`.

Exercice 6 Écrivez la fonction `struct matrice *transpose_matrice(struct matrice *A)` qui renvoie la matrice transposée de la matrice `A` (*Rappel* : si $A = (a_{ij})$ alors la matrice transposée B de A est $B = A^T = (b_{ij})$ avec $b_{ij} = a_{ji}$).

3 Structures et énumérations

On considère les énumérations :

```
enum couleur { COEUR, CARREAU, PIQUE, TREFLE};
enum valeur { SEPT, HUIT, NEUF, DIX, VALET, DAME, ROI, AS};
```

et la structure suivante :

```
struct carte { enum couleur couleur; enum valeur valeur; };
```

Exercice 7 Déclarez l'énumération `couleur`, puis affichez les valeurs (en utilisant `%d`) associées à `COEUR`, `CARREAU`, `PIQUE` et `TREFLE`.

Déclarez maintenant cette énumération :

```
enum couleur2 { COEUR2 = 3, CARREAU2, PIQUE2, TREFLE2};
```

Et affichez de même les valeurs associées. Que remarquez-vous? Vous pouvez essayer d'autres initialisations de l'énumération.

Exercice 8 Reprenez maintenant l'énumération `couleur` de départ et déclarez l'énumération `valeur` et la structure `carte`.

Écrivez la fonction `struct carte *jeu_de_cartes(void)` qui renvoie un tableau de cartes initialisé de manière à représenter un jeu complet de 32 cartes.

Exercice 9 Écrivez une fonction `void affiche_jeu(int cartes, struct carte *jeu)` qui permet d'afficher le nombre indiqué de cartes d'un ensemble de cartes `jeu`.

Remarque : pour simplifier la tâche, on pourra déclarer les deux tableaux suivants :

```
char *chaines_couleur[4] = { "Coeur", "Carreau", "Pique", "Trefle" };
char *chaines_valeur[8] = { "7", "8", "9", "10", "Valet", "Dame", "Roi", "As" };
```

Exercice 10 Écrivez une fonction `void melange_jeu(struct carte *jeu)` qui prend en argument un jeu de carte et le mélange (de la manière qui vous paraît la plus simple).

Rappel : on peut tirer un entier au hasard entre 0 et une valeur donnée avec la fonction `rand()` déclarée dans l'en-tête `<stdlib.h>` et l'opérateur modulo `%`.

Exercice 11 Écrivez une fonction `struct carte **donne(int joueurs, int cartes, struct carte *jeu)` qui prend en argument un jeu de carte mélangé contenant le nombre indiqué de cartes et qui renvoie un pointeur sur les mains obtenues par les différents joueurs (c'est-à-dire un pointeur vers un tableau de pointeurs vers des `struct carte` contenant autant de pointeurs que `jeu`). Tous les joueurs doivent recevoir autant de cartes et s'il reste des cartes en trop, elles ne seront pas données (ainsi pour un jeu de 32 cartes distribué à 5 joueurs, chaque joueur recevra 6 cartes et 2 cartes ne seront pas utilisées).

Exercice 12 On remarque que pour afficher un jeu de carte dans les exercices précédents, on devait préciser dans les arguments de la fonction le nombre de cartes qu'il contenait. Proposez une structure `struct jeu` qui permettrait de remédier à ce problème. Réécrivez ensuite les fonctions précédentes en utilisant cette structure.