TP 4 : Pointeurs et structures

Semaine du 19 février 2007

1 Introduction

▶ **Exercice 1**

La valeur affichée est 23 et non 42. Le pointeur sur la variable locale x de la fonction toto() n'est valable qu'au sein de cette fonction.

2 Matrices

▶ **Exercice 2**

```
void affiche_matrice(struct matrice A) {
    int i, j;
    for (i = 0; i < A.lignes; i++) {
        for (j = 0; j < A.colonnes; j++)
            printf("%g ", A.coefficients[i][j]);
        printf("\n");
    }
}
```

▶ **Exercice 3**

```
void affiche_matrice2(struct matrice *A) {
    int i, j;
    for (i = 0; i < A->lignes; i++) {
        for (j = 0; j < A->colonnes; j++)
            printf("%g ", A->coefficients[i][j]);
        printf("\n");
    }
}
```

▶ **Exercice 4**

```
struct matrice *alloue_matrice(int lignes, int colonnes) {
    struct matrice *A = malloc(sizeof(struct matrice));
    int i;
    printf("alloue_matrice() :\n");
    printf("    A                   : %p\n", A);
    A->colonnes = colonnes;
    A->lignes = lignes;
    A->coefficients = malloc(lignes * sizeof(double *));
    printf("    A->coefficients     : %p\n", A->coefficients);
    for (i = 0; i < lignes; i++) {
        A->coefficients[i] = malloc(colonnes * sizeof(double));
        printf("    A->coefficients[%i] : %p\n", i, A->coefficients[i]);
    }
    return A;
}
```

▶ **Exercice 5**

```
void libere_matrice(struct matrice *A) {
    int i;
    printf("libere_matrice() :\n");
    for (i = 0; i < A->lignes; i++) {
        free(A->coefficients[i]);
        printf("    A->coefficients[%i] : %p\n", i, A->coefficients[i]);
    }
    free(A->coefficients);
    printf("    A->coefficients    : %p\n", A->coefficients);
    free(A);
    printf("    A                  : %p\n", A);
}
```

▶ **Exercice 6**

```
struct matrice *transpose_matrice(struct matrice *A) {
    int i, j;
    struct matrice *B = alloue_matrice(A->colonnes, A->lignes);
    for (i = 0; i < B->lignes; i++)
        for (j = 0; j < B->colonnes; j++)
            B->coefficients[i][j] = A->coefficients[j][i];
    return B;
}
```

3 Structures et énumérations

▶ **Exercice 7**

La valeur associée à COEUR est 0. Les symboles suivants des énumérations prennent automatiquement comme valeur les entiers suivant l'entier associé à COEUR.

▶ **Exercice 8**

```
struct carte* jeu_de_cartes(void) {
    int i, j, n = 0;
    struct carte *jeu = malloc(32 * sizeof(struct carte));
    for (i = 0; i < 4; i++)
        for (j = 0; j < 8; j++) {
            jeu[n].couleur = i;
            jeu[n].valeur = j;
            n++;
        }
    return jeu;
}
```

▶ **Exercice 9**

```
void affiche_jeu(int cartes, struct carte *jeu) {
    int i;
    for (i = 0; i < cartes; i++)
        printf("%s de %s\n",
               chaines_valeur[jeu[i].valeur],
               chaines_couleur[jeu[i].couleur]);
}
```

▶ **Exercice 10**

```
void melange_jeu(struct carte *jeu) {
    int n, i, j;
```

```
    struct carte t;
    for (n = 0; n < 100; n++) {
        i = rand() % 32;
        j = rand() % 32;
        t.valeur = jeu[i].valeur;
        t.couleur = jeu[i].couleur;
        jeu[i].valeur = jeu[j].valeur;
        jeu[i].couleur = jeu[j].couleur;
        jeu[j].valeur = t.valeur;
        jeu[j].couleur = t.couleur;
    }
}
```

► **Exercice 11**

```
struct carte **donne(int joueurs, int cartes, struct carte *jeu) {
    int i, cartes_par_main = cartes / joueurs;
    struct carte **mains = malloc(joueurs * sizeof(struct carte *));
    for (i = 0; i < joueurs; i++)
        mains[i] = malloc(cartes_par_main * sizeof(struct carte));
    for (i = 0; i < cartes_par_main * joueurs; i++)
        mains[i % joueurs][i / joueurs] = jeu[i];
    for (i = 0; i < joueurs; i++) {
        printf("Joueur %i :\n", i + 1);
        affiche_jeu(cartes_par_main, mains[i]);
    }
    return mains;
}
```

► **Exercice 12**

```
struct jeu { int nb_cartes; struct carte *cartes; };

struct jeu *jeu_de_cartes2(void) {
    int i, j, n = 0;
    struct jeu *jeu = malloc(sizeof(struct jeu));
    jeu->nb_cartes = 32;
    jeu->cartes = malloc(jeu->nb_cartes * sizeof(struct carte));
    for (i = 0; i < 4; i++) {
        for (j = 0; j < 8; j++) {
            jeu->cartes[n].couleur = i;
            jeu->cartes[n].valeur = j;
            n++;
        }
    }
    return jeu;
}

void affiche_jeu2(struct jeu *jeu) {
    int i;
    for (i = 0; i < jeu->nb_cartes; i++)
        printf("%s de %s\n",
               chaines_valeur[jeu->cartes[i].valeur],
               chaines_couleur[jeu->cartes[i].couleur]);
}

void melange_jeu2(struct jeu *jeu) {
    int i, j, n;
```

```
    struct carte t;
    for (n = 0; n < 100; n++) {
        i = rand() % jeu->nb_cartes;
        j = rand() % jeu->nb_cartes;
        t.valeur = jeu->cartes[i].valeur;
        t.couleur = jeu->cartes[i].couleur;
        jeu->cartes[i].valeur = jeu->cartes[j].valeur;
        jeu->cartes[i].couleur = jeu->cartes[j].couleur;
        jeu->cartes[j].valeur = t.valeur;
        jeu->cartes[j].couleur = t.couleur;
    }
}

struct jeu *donne2(int joueurs, struct jeu *jeu) {
    int i, cartes_par_main = jeu->nb_cartes / joueurs;
    struct jeu *mains = malloc(joueurs * sizeof(struct jeu));
    for (i = 0; i < joueurs; i++) {
        mains[i].cartes = malloc(cartes_par_main * sizeof(struct carte));
        mains[i].nb_cartes = cartes_par_main;
    }
    for (i = 0; i < cartes_par_main * joueurs; i++)
        mains[i % joueurs].cartes[i / joueurs] = jeu->cartes[i];
    for (i = 0; i < joueurs; i++) {
        printf("Joueur %i :\n", i + 1);
        affiche_jeu2(&mains[i]);
    }
    return jeu;
}
```