TP 11 : pointeurs de fonctions et préprocesseur

Programmation en C (LC4)

Semaine du 23 avril 2007

1 Les pointeurs de fonction

▶ **Exercice 1**

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

int fois_deux(int i) {
    return 2 * i;
}

void appliquer_tableau(int (*f)(int), int *t, int n) {
      int i;
      for (i = 0; i < n; i++)
          t[i] = (*f)(t[i]);
}

int main(void) {
    int tab[] = {1, 2, 3, 4};
    int taille = sizeof(tab) / sizeof(int);
    int i;
    appliquer_tableau(&fois_deux, tab, taille);
    for (i = 0; i < taille; i++)
        printf("%d ", tab[i]);
    printf("\n");
    exit(EXIT_SUCCESS);
}
```

▶ **Exercice 2**

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

int superieur(int a, int b) {
    return (a > b) ? 1 : ((a == b) ? 0 : -1);
}

int inferieur(int a, int b) {
    return -superieur(a, b);
}

void tri(int * t, int n, int (*compare_int)(int a, int b)) {
    int i_min, i, j, tmp;
    for (i = 0; i < n - 1; i++){
        i_min = i;
        for (j = i; j < n; j++)
```

```
            if ((*compare_int)(t[j], t[i_min]) < 0)
                i_min = j;
        tmp = t[i_min];
        t[i_min] = t[i];
        t[i] = tmp;
    }
}

void afficher_tableau(int *t, int n) {
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", t[i]);
    printf("\n");
}

int main(void) {
    int tab[] = { 6, 2, 8, 6, 4, 8, 2, 9, 2, 1};
    int taille = sizeof(tab) / sizeof(int);
    tri(tab, taille, &superieur);
    afficher_tableau(tab, taille);
    tri(tab, taille, &inferieur);
    afficher_tableau(tab, taille);
    exit(EXIT_SUCCESS);
}
```

2 Préprocesseur

3 Macros

► **Exercice 3**

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

#define PI 3.14159265358979323846
#define CUBE(x) ((x) * (x) * (x))
#define SOMME(x, y) ((x) + (y))
#define VOLUME(x) ((4 * PI * CUBE(x)) / 3)

int main(int argc, char *argv[]) {
    double d;
    int i;
    for (i = 1; i < argc; i++) {
        printf("Le volume de la sphere %d est %g.\n",
               i, VOLUME(1 + strtod(argv[i], NULL)));
        d = SOMME(d, VOLUME(1 + strtod(argv[i], NULL));
    }
    printf("Le volume total est %f\n", d);
    return 0;
}
```

4 Compilation conditionnelle

► **Exercice 4**

`toto.h` :

```c
#ifndef TOTO_H
#define TOTO_H

#include "tata.h"

int somme(int x, int y);

#endif /* TOTO_H */
```

toto.c :

```c
#include "toto.h"

int somme(int x, int y) {
    return (x + y);
}
```

tata.h :

```c
#ifndef TATA_H
#define TATA_H

#include "toto.h"

int produit(int x, int y);

#endif /* TATA_H */
```

tata.c :

```c
#include "tata.h"

int produit(int x, int y) {
    return (x * y);
}
```

main.c :

```c
#include "toto.h"
#include "tata.h"

int main(int argc, char *argv[]) {
    int a = 2;
    int b = 3;
    int c, d;
    c = somme(a, b);
    d = produit(d, a);
    return 1;
}
```

Makefile :

```
CC = gcc

circulaire: tata.o toto.o main.o
	$(CC) tata.o toto.o main.o -o circulaire

tata.o: tata.c toto.h
	$(CC) -c tata.c -o tata.o

toto.o: toto.c tata.h
```

```
	$(CC) -c toto.c -o toto.o

main.o: main.c toto.h tata.h
	$(CC) -c main.c -o main.o

clean:
	rm -rf circulaire tata.o toto.o main.o
```

▶ **Exercice 5**

```
/* #define NDEBUG */
#include <assert.h>

int main(int argc, char *argv[])
{
    assert(argc >= 4);

    return 0;
}

#ifndef NDEBUG
# define mon_assert(expr)                                   \
    do {                                                    \
        if (!(expr)) {                                      \
            fprintf(stderr, "assertion failed: " #expr      \
                            " in file \"" __FILE__          \
                            "\", line %d\n", __LINE__);     \
            exit(1);                                        \
        }                                                   \
    } while (0)
#else
# define mon_assert(expr) do { } while (0)
#endif /* NDEBUG */
```

5 Programmation modulaire

▶ **Exercice 6**

Cf. correction de l'exercice suivant.

▶ **Exercice 7**

`vecteur.h` :

```
#ifndef VECTEUR_H
#define VECTEUR_H

#define CONCAT_REEL2(nom, suffixe) nom##suffixe
#define CONCAT_REEL(nom, suffixe) CONCAT_REEL2(nom, suffixe)

#define REEL double
#define SUFFIXE_REEL d
#define VECTEUR_REEL CONCAT_REEL(vecteur, SUFFIXE_REEL)
# include "vecteur-decl.h"
#undef VECTEUR_REEL
#undef SUFFIXE_REEL
#undef REEL

#define REEL float
#define SUFFIXE_REEL f
```

```
#define VECTEUR_REEL CONCAT_REEL(vecteur, SUFFIXE_REEL)
# include "vecteur-decl.h"
#undef VECTEUR_REEL
#undef SUFFIXE_REEL
#undef REEL

#endif /* VECTEUR_H */
```

vecteur-decl.h :

```
typedef struct VECTEUR_REEL *VECTEUR_REEL;

VECTEUR_REEL CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(REEL x, REEL y, REEL z);

void CONCAT_REEL(detruit_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v);

void CONCAT_REEL(affiche_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v);

VECTEUR_REEL CONCAT_REEL(ajoute_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v1,
                                                       VECTEUR_REEL v2);
VECTEUR_REEL CONCAT_REEL(soustrait_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v1,
                                                          VECTEUR_REEL v2);

VECTEUR_REEL CONCAT_REEL(normalise_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v);
VECTEUR_REEL CONCAT_REEL(multiplie_vecteur, SUFFIXE_REEL)(REEL r, VECTEUR_REEL v);
VECTEUR_REEL CONCAT_REEL(vecteur_oppose, SUFFIXE_REEL)(VECTEUR_REEL v);

REEL CONCAT_REEL(produit_scalaire, SUFFIXE_REEL)(VECTEUR_REEL v1,
                                                 VECTEUR_REEL v2);
```

vecteur.c :

```
#include "vecteur.h"

#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define REEL double
#define SUFFIXE_REEL d
#define VECTEUR_REEL CONCAT_REEL(vecteur, SUFFIXE_REEL)
# include "vecteur-impl.c"
#undef VECTEUR_REEL
#undef SUFFIXE_REEL
#undef REEL

#define REEL float
#define SUFFIXE_REEL f
#define VECTEUR_REEL CONCAT_REEL(vecteur, SUFFIXE_REEL)
# include "vecteur-impl.c"
#undef VECTEUR_REEL
#undef SUFFIXE_REEL
#undef REEL
```

vecteur-impl.c :

```
struct VECTEUR_REEL {
    REEL x, y, z;
};
```

```
VECTEUR_REEL CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(REEL x, REEL y, REEL z)
{
    VECTEUR_REEL v = malloc(sizeof(struct VECTEUR_REEL));

    v->x = x;
    v->y = y;
    v->z = z;

    return v;
}

void CONCAT_REEL(detruit_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v)
{
    free(v);
}

void CONCAT_REEL(affiche_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v)

{
    printf("[%g;%g;%g]\n", v->x, v->y, v->z);
}

VECTEUR_REEL CONCAT_REEL(ajoute_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v1,
                                                       VECTEUR_REEL v2)
{
    return CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(v1->x + v2->x,
                               v1->y + v2->y,
                               v1->z + v2->z);
}

VECTEUR_REEL CONCAT_REEL(soustrait_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v1,
                                                          VECTEUR_REEL v2)
{
    return CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(v1->x - v2->x,
                                                      v1->y - v2->y,
                                                      v1->z - v2->z);
}

VECTEUR_REEL CONCAT_REEL(normalise_vecteur, SUFFIXE_REEL)(VECTEUR_REEL v)
{
    REEL norme = sqrt(CONCAT_REEL(produit_scalaire, SUFFIXE_REEL)(v, v));

    return CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(v->x / norme,
                                                      v->y / norme,
                                                      v->z / norme);
}

VECTEUR_REEL CONCAT_REEL(multiplie_vecteur, SUFFIXE_REEL)(REEL r, VECTEUR_REEL v)
{
    return CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(r * v->x,
                                                      r * v->y,
                                                      r * v->z);
}

VECTEUR_REEL CONCAT_REEL(vecteur_oppose, SUFFIXE_REEL)(VECTEUR_REEL v)
```

```
{
    return CONCAT_REEL(nouveau_vecteur, SUFFIXE_REEL)(-v->x,
                                                      -v->y,
                                                      -v->z);
}

REEL CONCAT_REEL(produit_scalaire, SUFFIXE_REEL)(VECTEUR_REEL v1,
                                                 VECTEUR_REEL v2)
{
    return (v1->x * v2->x + v1->y * v2->y + v1->z * v2->z);
}
```

main.c :

```
#include <stdlib.h>

#include "vecteur.h"

int main(int argc, char *argv[])
{
    /* r = 2 * v . (u / |u|) u / |u| - v */
    vecteurf u = nouveau_vecteurf(1, 2, 3);
    vecteurf v = nouveau_vecteurf(4, 5, 6);
    vecteurf n = normalise_vecteurf(u);
    float f = 2 * produit_scalairef(n, v);
    vecteurf w = multiplie_vecteurf(f, n);
    vecteurf r = soustrait_vecteurf(w, v);

    affiche_vecteurf(r);

    detruit_vecteurf(r);
    detruit_vecteurf(w);
    detruit_vecteurf(n);
    detruit_vecteurf(u);
    detruit_vecteurf(v);

    exit(0);
}
```

Makefile :

```
CC = gcc
LIBS = -lm

vecteur: main.o vecteur.o
	$(CC) -o vecteur vecteur.o main.o $(LIBS)

vecteur.o: vecteur.c vecteur-impl.c vecteur.h vecteur-decl.h
	$(CC) -o vecteur.o -c vecteur.c

main.o: main.c vecteur.h vecteur-decl.h
	$(CC) -o main.o -c main.c

clean:
	rm -f vecteur *.o
```