TD 11 : static, macros, pointeurs sur des fonctions

Programmation en C (LC4)

Semaine du 23 avril 2007

1 Mot clé static (dans une fonction)

► **Exercice 1**

Déclaration avec static :

```
1
2
3
----
4, 2
5, 4
6, 8
```

Déclaration sans static :

```
1
1
1
----
1, 2
1, 2
1, 2
```

2 Macros

► **Exercice 2**

```
7 9
```

Il faut protéger les expressions passées en arguments à la macro des opérateurs plus prioritaires qui figurent dans le corps de la macro en rajoutant des parenthèses :

```
#define FOIS_MACRO(a, b) ((a) * (b))
```

► **Exercice 3**

```
x=3, y=9, tmp1=3
x=5, y=20, tmp2=4
```

On utilise un do/while (0) pour avoir un bloc d'instructions sous la forme d'une seule instruction compatible avec un if / else sans accolade :

```
#define M(a, b, tmp) do { int tmp = a; b = tmp * a; } while (0)
```

3 Pointeurs sur des fonctions

► **Exercice 4**

```c
#include <stdio.h>

double dichotomie(double (*f)(double), double a, double b, double precision) {
    double c;
    int sgn_c, sgn_a = ((*f)(a) > 0);
    while ((b - a) > precision) {
        c = (a + b) / 2;
        printf("a=%f b=%f c=%f\n", a, b, c);
        sgn_c = (*f)(c) > 0;
        if (sgn_c == sgn_a)
            a = c;
        else
            b = c;
    }
    return c;
}

double cube(double a) {
    return (a * a * a);
}

double f(double a) {
    return (-cube(a) + 2);
}

int main(void) {
    double precision = 0.001;
    printf("cube: %g\nf: %g\n", dichotomie(&cube, -2, 1, precision),
                               dichotomie(&f, -10, 10, precision));
    return 0;
}
```

► **Exercice 5**

```c
#include <stdio.h>

int accumule(double (*f)(double), liste_t l) {
    double resultat = 0;
    while (l) {
        resultat += (*f)(l->valeur);
        l = l->suivant;
    }
    return resultat;
}

double un(double x) {
    return 1;
}

double identite(double x) {
    return x;
}

double carre(double x) {
    return (x * x);
}
```

```
/* mu_{n+1} = mu(x_1, ..., x_{n+1}) = mu_n + (x_{n+1} - mu_n)) / (n+1) */
double mu(double x) {
    static double n = 0;
    static double m = 0;
    double d = x - m;

    n++;
    m += d / n;

    return (d / n);
}

/* s_{n+1} = n sigma^2(x_1, ..., x_{n+1})
           = s_n + (x_{n+1} - mu_n) * (x_{n+1} - mu_{n+1}) */
double nsigma2(double x) {
    static double n = 0;
    static double m = 0;
    double d = x - m;

    n++;
    m += d / n;

    return (d * (x - m));
}

int main(void) {
    liste_t l = /* À COMPLÉTER */;

    printf("%g %g, %g, %g, %g\n", accumule(&un, l),
                                  accumule(&identite, l),
                                  accumule(&carre, l),
                                  accumule(&mu, l),
                                  accumule(&nsigma2, l));
    return 0;
}
```

4 Reprise du TD 10

► **Exercice 6**

```
#include <stdlib.h>

liste_t insere(liste_t l, void *x, int (*inferieur)(void *, void *)) {
    liste_t *precedent = &liste;
    liste_t k = l;
    liste_t t = malloc(sizeof(struct liste_s));
    while (k && (*inferieur)(k->valeur, x)) {
        *precedent = &k->suivant;
        k = k->suivant;
    }
    t->valeur = x;
    t->suivant = l;
    *precedent = t;
    return l;
}
```