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 Qu'affiche le programme suivant ? Qu'afficherait-il si les deux variables cpt n'étaient pas déclarées static ?

```
#include<stdio.h>
int f() {
    static int cpt = 0;
    cpt++;
    return cpt;
}
int g() {
    static int cpt = 1;
    cpt *= 2;
    return cpt;
}
int main(void) {
    int i;
    for(i = 0; i < 3; i++)
        printf("%d\n", f());
    printf("----\n");
    for(i = 0; i < 3; i++)
        printf("%d, %d\n", f(), g());
    return 0;
}
```

2 Macros

Exercice 2 Qu'affiche le programme suivant ? Pourquoi ? Comment y remédier ?

```
#include <stdio.h>
#define FOIS_MACRO(a, b) a * b
int fois_fonction(int a, int b) {
    return (a * b);
}
int main(void) {
    int x = 1, y = 2, z = 3;
    printf("%d %d\n", FOIS_MACRO(x + y, z), fois_fonction(x + y, z));
    return 0;
}
```

Exercice 3 Qu'affiche le programme suivant ? Quels autres problèmes se posent si l'on veut utiliser la macro M dans un if / else sans accolade ? Comment la modifier en conséquence ?

```
#include <stdio.h>
#define M(a, b, tmp) int tmp = a; b = tmp * a
int main(void) {
    int x = 3, y;
    M(x, y, tmp1);
    printf("x=%d, y=%d, tmp1=%d\n", x, y, tmp1);
    M(++x, y, tmp2);
    printf("x=%d, y=%d, tmp2=%d\n", x, y, tmp2);
    return 0;
}
```

3 Pointeurs sur des fonctions

Exercice 4 Écrivez une fonction racine() qui prend en argument :
- un pointeur sur une fonction : double f(double)
- trois réels : double a, b, precision

et qui cherche par dichotomie une racine de f comprise entre a et b à precision près.

Exercice 5 On considère le type liste de doubles suivant :

```
typedef struct liste_s {
    double valeur; struct liste_s *suivant;
} *liste_t;
```

Écrivez une fonction accumule() prenant en argument :
- un pointeur sur une fonction : double f(double)
- une liste de doubles : liste_t l

et qui calcule (et renvoie) la somme : $\sum_{i=1}^{n} f(l_i)$ où l_i est le $i^{\text{ème}}$ élèment de la liste l et n est sa taille.

Utilisez cette fonction pour calculer n, $\sum_{i=1}^{n} l_i$, $\sum_{i=1}^{n} {l_i}^2$.

Utilisez cette fonction pour calculer $\mu = \frac{\sum_{i=1}^{n} l_i}{n}$ (sans avoir à diviser par n le résultat d'accumule() ?), $s = n\sigma^2 = \sum_{i=1}^{n}(l_i - \mu)^2$ (sans avoir à calculer μ au préalable ?).

4 Reprise du TD 10

On travaille avec le type suivant :

```
typedef struct liste_s {
    void *valeur; struct liste_s *suivant;
} *liste_t;
```

L'idée est que l'on met dans le champ `valeur` un pointeur vers un objet dont on a fait oublier le type au compilateur. Cela permet d'écrire du code travaillant sur des listes, indépendamment du type des objets manipulés.

Exercice 6 Écrire une fonction

```
liste_t insere(liste_t l, void *x, int (*inferieur)(void *,void *))
```

qui prend en argument une liste, un pointeur x vers un objet de type inconnu, et un pointeur inferieur vers une fonction qui est supposée implémenter une relation d'ordre pour laquelle la liste l est triée ; et qui insère x dans la liste, de sorte qu'elle soit toujours triée.

Par exemple, si les objets stockés dans la liste sont des chaînes de caractères, inferieur pourrait pointer vers une telle fonction :

```
int compare_chaine(void *s, void *t) {
    return (strcmp((char *) s, (char *) t) > 0);
}
```