TD 9 : static,macros listes chainées

Programmation en C (LC4)

Semaine du 24 mars 2008

1 Mot clé static, dans une fonction

Exercice 1 Qu'affiche le programme suivant ?

```
#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;
}
```

► **Correction**

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

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;
}
```

▶ **Correction**

7 9

Exercice 3 Qu'affiche le programme suivant ?

```
#include <stdio.h>
#define f(a,b, _tmp) int _tmp = a;\
                     b = _tmp * a

int main(void){
  int x=3, y;
  f(x, y, tmp1);
  printf("x=%d, y=%d, tmp1=%d\n", x, y, tmp1);
  f(++x, y, tmp2);
  printf("x=%d, y=%d, tmp2=%d\n", x, y, tmp2);
  return 0;
}
```

▶ **Correction**

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

3 Listes doublement chainées

On veut implémenter des listes circulaires, où chaque élément pointe vers son prédécesseur et son successeur. On peut utiliser le type suivant :

```
struct liste_circulaire {
  struct liste_circulaire * precedent;
  struct liste_circulaire * suivant;
  int contenu;
};
```

Exercice 4 Ecrire une fonction

```
struct liste_circulaire * insere(struct liste_circulaire* l,int x)
```

qui insère l'entier `x` entre `l` et `l.suivant` si `l` n'est pas `NULL`, et crée une liste contenant juste `x` dans le cas contraire. Elle doit renvoyer en résultat un pointeur vers le nœud de la liste créé pour contenir `x`.

▶ **Correction**

```
struct liste_circulaire * insere(struct liste_circulaire* l,int x) {
  struct liste_circulaire * p=malloc(sizeof(struct liste_circulaire));
  p->contenu=x;
  if (l) {
    p->suivant=l->suivant;
    l->suivant->precedent=p;
    p->precedent=l;
    l->suivant=p;
  } else {
    p->precedent=p;
    p->suivant=p;
  }
  return p;
}
```

Exercice 5 Ecrire une fonction

struct liste_circulaire * supprime(**struct** liste_circulaire * l)

qui efface l'élément pointé par `l` de la liste circulaire à laquelle il appartient. Elle doit renvoyer `NULL` si la liste est vide après l'effacement de `l`, et `l.precedent` sinon.

▶ **Correction**

```
struct liste_circulaire * supprime(struct liste_circulaire * l) {
  if (l->suivant==l) {
    free(l);
    return NULL;
  } else {
    struct liste_circulaire * p=l->precedent;
    p->suivant=l->suivant;
    l->suivant->precedent=p;
    free(l);
    return p;
  }
}
```

Exercice 6 Ecrire une fonction

int compte(**struct** liste_circulaire * l)

qui compte le nombre d'éléments dans la liste pointée par `l`.

▶ **Correction**

```
int compte(struct liste_circulaire * l) {
  struct liste_circulaire * p=l;
  int res=0;
  do {
    res++;
    p=p->suivant;
  } while (p!=l);
  return res;
}
```

Exercice 7 Ecrire une fonction

void inverse(**struct** liste_circulaire * l)

qui inverse l'ordre des élḿents de la liste `l` (sans toucher aux champs `contenu`).

▶ **Correction**

```
void inverse(struct liste_circulaire * l) {
  struct liste_circulaire * p=l;
  struct liste_circulaire * q=l;
  do {
    q=p->suivant;
    p->suivant=p->precedent;
    p->precedent=q;
    p=q;
  } while (p!=l);
}
```