TP11

Langage C (LC4)

semaine du 12 avril 2010

1 Pointeurs vers des fonctions

1.1 Liste

On commence par travailler avec le type de listes suivant :

```
  struct elem_liste {
    int valeur ;
    struct elem_liste * suivant ;
  };

typedef struct elem_liste * Liste;
```

Question 1. Quel est le type d'un pointeur vers une fonction prenant un int en argument et renvoyant un int en résultat ?

Question 2. Écrire une fonction `int pour_tout(Liste l, int (*predicat)(int))` qui prend en argument une liste l, et un pointeur predicat vers une fonction, et qui détermine si la fonction appliquée à tous les éléments de la liste retourne une valeur non-nulle.

Question 3. En utilisant la question précédente, écrire une fonction qui teste si tous les éléments d'une liste sont impairs.
On travaille désormais avec le type suivant :

```
struct elem_liste {
  void * valeur ;
  struct elem_liste * suivant ;
};

typedef struct elem_liste * Liste;
```

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

Question 4. Écrire une fonction

```
  Liste insere (Liste 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. Le résulat renvoyé est le maillon de liste créé pour x. Par exemple, si les objets stockés dans la liste sont des chaines de caractères, inferieur pourrait pointer vers une telle fonction :

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

1.2 Algorithme de tri de tableau

Voici un algorithme de tri des éléments d'un tableau dans l'ordre croissant :

```
void
tri_croissant(int * t, int n)
{
  int i,i_min,j,tmp;
  for ( i = 0; i < n - 1; i++)
    {
      i_min = i;
      for (j = i; j < n; j++)
        {
          if (t[j] < t[i_min])
            {
              i_min = j;
            }
        }
      tmp = t[i_min];
      t[i_min] = t[i];
      t[i] = tmp;
    }
}
```

Voici un algorithme de tri des éléments d'un tableau dans l'ordre décroissant :

```
void
tri_decroissant(int * t, int n)
{
  int i,i_max,j,tmp;
  for ( i = 0; i < n - 1; i++)
    {
      i_max = i;
      for (j = i; j < n; j++)
        {
          if (t[j] > t[i_max])
            {
              i_max = j;
            }
        }
      tmp = t[i_max];
      t[i_max] = t[i];
      t[i] = tmp;
    }
}
```

Ces deux algorithmes sont identiques, à l'exception de l'opérateur de comparaison. Afin d'éviter de dupliquer du code, nous allons utiliser les pointeurs de fonctions.

Question 5. Ecrire une fonction `superieur(int a, int b)` qui renvoie 1 si a est supérieur à b, 0 s'ils sont égaux et -1 sinon

Question 6. Ecrire une fonction `inferieur(int a, int b)` qui renvoie 1 si a est inférieur à b, 0 s'ils sont égaux et -1 sinon

Question 7. Ecrire une fonction `tri`, qui tri un tableau `t` de taille n selon une fonction `compare`

Question 8. Ecrire une fonction `main` qui teste `tri`

2 Macros

Question 9. 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;
}
```

Question 10. 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;
}
```

3 Quelques pièges avec les fichiers

Question 11. La fonction `gets` sert à lire une ligne depuis `stdin`, et l'écrit dans un tableau qu'on lui passe en argument. Entrer le code suivant, et l'exécuter en lui entrant une longue ligne :

```c
#include <stdio.h>

int main(int argc, char** argv) {
  int x=0;
  char buf[10];
  gets(buf);
  printf("%d\n",x);
  exit(0);
}
```

Que se passe-t-il ?

Question 12. Exécuter le code suivant :

```c
#include <stdio.h>
#include <unistd.h>
int main(int argc,char **argv) {
  fputs("ploum",stdout);
  sleep(20); // La fonction sleep sert a mettre le programme en pause
             // pendant le nombre de secondes passees en argument

}
```

Que se passe-t-il ?