LANGAGE C AVANCÉ

TD 2 : Exercices sur les pointeur

Manipulation, tableaux, chaînes de caractères, listes chaînée

---

Objectif : le présent TD a pour but de manipuler les pointeurs sous plusieurs formes.

1. Soit le code suivant

```
int a=3;
int b=10;
int c, *pa, *pb, *pc;

pa = &a;
*pa = *pa * 2;
pb = &b;
c = 3 * (*pb - *pa);
pc = pb;
pa = pb;
pb = pc;
```

À la fin de l'exécution, que vallent `*pa`, `*pb`, `a`, `b`, et `c` ?

2. Le code suivant est-il correct, et si non pourquoi ?

```
{
  int i, *p;
  p = (int*) malloc(10 * sizeof(int));
  for (i = 1; i <= 10; i++)
    p[i] = i * i;
}
```

3. Le code suivant est-il correct, et si non pourquoi ?

```
{
  int i, *p;
  p = (int*) malloc(10 * sizeof(int));
  for (i = 1; i <= 10; i++)
    *p++ = i * i;
}
```

4. Analyser le programme suivant (donnez la séquence des affichages produits) :

```
int f1(int i) { return i+1;                   }
int f2(int i) { return i++;                   }
int f3(int i) { printf("%d\n",i==0); return i;}
int f4(int i) { printf("%d\n",i=0);  return i;}
int f5(int *i){ return ++(*i);                }
int f6(int *i){ (*i)++;         return (*i)++;}
int main(){
        int a,b;
        a=f1(0);    b=f2(1);  printf("a=%d, b=%d\n",a,b);
        a=f3(a);    b=f4(a);  printf("a=%d, b=%d\n",a,b);
        a=f5(&a);   b=f6(&a); printf("a=%d, b=%d\n",a,b);
}
```

5. Écrire une fonction qui prend comme arguments `a`, `b`, `c` de type `float`, et des pointeurs vers un entier `n`, et deux flottants `x1` et `x2`. Après l'appel de la fonction, `*n` doit contenir le nombre de solutions de l'équation $\mathtt{a}\,x^2 + \mathtt{b}\,x + \mathtt{c} = 0$, et `*x1`, `*x2` cette ou ces solution(s).

Chaînes de caractères

6. Une chaîne de caractères w est un palindrôme si elle est identique à la chaîne retournée. Écrire la fonction `palindrome` qui prend en argument une chaîne de caractères et retourne 1 si la chaîne est un palindrôme et 0 sinon. Par exemple

```
palindrome ("eluparcettecrapule") --> 1
palindrome ("eluparcetabruti") --> 0
palindrome ("esoperesteicietserepose") --> 1
```

7. Écrire une fonction qui prend comme argument deux chaînes de caractères `str` (une longue chaîne) et `wrd` (un mot), et qui renvoie le nombre de fois que `wrd` est présent dans `str`.

8. Écrire une fonction qui prend comme argument deux chaînes de caractères `str` (une longue chaîne) et `wrd` (un mot), et qui renvoie la première position de `wrd` dans `str` si `wrd` est présent dans `str`, et -1 sinon.

Tableaux

9. Écrire une fonction `int max_tab(int *tab, int size)` qui renvoie l'indice de l'élément maximum du tableau `tab` de taille `size` (`size`> 0).

10. Écrire une fonction `int * add_tab(int *tab, int size, int elmt)` qui ajoute elmt dans le tableau trié `tab` de taille `size`, en le maintenant trié.

11. On dispose d'un ensemble de coupures (pièces d'un euro, de 2, billets de 5, 10, 20, 50, 100, 200, 500). Écrire une fonction `coupure` qui prend comme argument un tableau de coupures, le nombre d'éléments dans le tableau, et une somme, et qui renvoie le nombre de façons de constituer la somme au moyen des coupures.

Listes

Pour les exercices de cette section, on considère que la structure d'un élément de liste liée suivante

```
struct Tlist
{
  int nb;
  struct Tlist * next;
}
```

Les éléments dans la liste sont des entiers signés, les algorithmes ne dépendent cependant pas du type des éléments.

12. Écrire un programme qui demande à l'utilisateur de donner des nombres entiers, et qui les met

dans une liste liée. Afficher ensuite tous ces nombres.

13. Construire une fonction `list_print` qui prends une liste en argument et imprime les éléments sous la forme `(l1 l2 ... ln)`.

14. Construire une fonction `list_free` prenant une liste liée en argument, et libère la mémoire prise par celle-ci.

15. Construire une fonction `list_length` prenant une liste liée en argument, et renvoie le nombre d'éléments dans la liste.

16. Définir la fonction `list_member` qui renvoie 1 si le premier argument (entier) est un élément de la liste donnée comme second argument et 0 sinon.

17. Définir la fonction `list_remove_last` qui renvoie la liste (non vide) donnée en argument, privée de son dernier élément.

18. Définir la fonction `list_append` qui prend deux listes en argument et qui renvoie la concaténation de ces deux listes.