Langage C : TD5

Exercice 1 : Expliquer ce que fait le code suivant (la fonction strlen est définie dans string.h et retourne la longueur de la chaîne de caractères passée en argument).

```
int i , l;
char *s , *sp;

scanf("%s ", s);
printf("%s \n", s);

l = strlen(s);
sp = (char *) malloc(l * sizeof(char));

for (i = 0 ; i < l ; i++) {sp[i] = s[i];}
printf("%s \n", sp);
free(sp);
```

- Ligne 3 : on essaie de lire une chaîne de caractères et l'affecter à s qui n'est pas initialisé. Correction :

```
int i , l;
char s[100] , *sp;                          //MODIFICATION

scanf("%s ", s);
printf("%s \n", s);

l = strlen(s);
sp = (char *) malloc(l * sizeof(char));

for (i = 0 ; i < l ; i++) {sp[i] = s[i];}
printf("%s \n", sp);
free(sp);
```

OU

```
int i , l;
char *s = (char *)malloc(100*sizeof(char)), *sp;    //MODIFICATION

scanf("%s ", s);
printf("%s \n", s);

l = strlen(s);
sp = (char *) malloc(l * sizeof(char));

for (i = 0 ; i < l ; i++) {sp[i] = s[i];}
printf("%s \n", sp);
free(sp);
```

- Ligne 8 : notre chaîne de caractères sp ne se finit pas par \0. Correction :

```
int i , l;
char *s = (char *)malloc(100*sizeof(char)), *sp;

scanf("%s ", s);
printf("%s \n", s);

l = strlen(s);
sp = (char *) malloc((l + 1) * sizeof(char));       //MODIFICATION

for (i = 0 ; i <= l ; i++) {sp[i] = s[i];}          //MODIFICATION
```

```
printf("%s \n", sp);
free(sp);
```

Exercice 2 : Expliquer ce que font les trois fonctions suivantes :

```
int *f1()
{
    int *a;
    *a = 1;
    return a;
}

int *f2()
{
    int a;
    a = 1;
    return &a;
}

int *f3()
{
    int *a;
    a = (int *)malloc(sizeof(int));
    *a = 1;
    return a;
}
```

- Dans la fonction f1, le pointeur n'est pas initialisé, ile ne pointe nulle part.
- Dans la fonction f2, on retourne l'adresse d'une variable locale. Lorsqu'une fonction se termine, toutes ses variables locales sont supprimées.
- La fonction f3 fonctionne très bien car on allou l'espace pour stocker un entier sur le tas et non la pile. Tant qu'on' ne fait pas de free et que le programme tourne, le programme ne le libérera pas.

Exercice 3 : Écrire une fonction qui alloue dynamiquement un tableau permettant de stocker n valeurs de type int, initialise chacune des cases avec son indice, et retourne l'adresse du tableau ainsi alloué.

```
int *tab_dyn(int n)
{
    int *t = NULL;
    t = (int *)malloc(n * sizeof(int));
    if (t != NULL)
    {
        for (int i = 0 ; i < n ; i++ )
        {
            t[i] = i;
        }
    }

    return t;
}
```

Exercice 4 : Expliquer ce que fait ce code :

```
void f(int *a)
{
    a = (int*)malloc(sizeof(int));
}

int main()
{
    int *x;
    f(x);
    *x = 6;
    printf("%d\n", *x);

    return 0;
}
```

La fonction f travaille sur une copie du pointeur et non le pointeur lui même. Deux façons de corriger :

```
int *f(int *a)                          //MODIFICATION
{
    a = (int*)malloc(sizeof(int));
    return a;                           //MODIFICATION
}

int main()
{
    int *x;
    x = f(x);                           //MODIFICATION
    *x = 6;
    printf("%d\n", *x);

    return 0;
}
```

OU

```
void f(int **a)                         //MODIFICATION
{
    *a = (int*)malloc(sizeof(int));     //MODIFICATION
}

int main()
{
    int *x;
    f(&x);                              //MODIFICATION
    *x = 6;
    printf("%d\n", *x);

    return 0;
}
```

Exercice 5 : Écrire une fonction qui

1. Alloue dynamiquement la mémoire pour stocker une matrice de nl lignes et nc colonnes de réels.

```
int **mat_dyn(int nl, int nc)
{
    int i, double **t = NULL;
    t = (int **)calloc(nl, sizeof(int *));
```

```
}
```

2. Libère la mémoire occupée par une matrice ayant nl lignes et nc colonnes.