Exercice 1:

Considérons les quatre fonctions suivantes :

```
float copier_un_reel_v1(float b) {
    float a;
    a=b;
    return a;
}
void copier_un_reel_v2(float *a, float b) {
    *a=b;
}
void copier_un_reel_v3(float *a, float *b) {
    *a=*b;
}

void copier_un_tableau(int a[], int b[], int n){
    int i;
    for (i=0; i<n; i++) a[i]=b[i];
}
```

- Proposer une ré-écriture des fonctions ci-dessus, sans utiliser le symbole d'affectation "=" et en utilisant une des fonctions de la bibliothèque "string.h" vue en cours.
- Tester vos nouvelles fonctions depuis le programme principal main.
- Ecrire une fonction, appelée memcpy_version_etudiant, qui a l'entête suivante :

  ```
  void * memcpy_version_etudiant(void * destination, const void * source, size_t n)
  ```

  Cette fonction recopie les n premiers octets depuis l'adresse mémoire source vers la zone mémoire pointée par destination. La fonction retourne un pointeur sur destination. Il n'est pas permis d'utiliser la fonction memcpy définie dans la bibliothèque <string.h>.
- Quelle est la principale différence entre la fonction memcpy et la fonction strncpy.

Exercice 2:

- Ecrire une fonction qui a comme entête

  ```
  int majuscule (const char c)
  ```

  Cette fonction retourne :

  - 1 : si c est une lettre majuscule
  - 0 : sinon
- Ecrire une fonction qui a comme entête

  ```
  int compte_majuscules (const char *s)
  ```

  Cette fonction retourne le nombre de lettres majuscules dans une chaîne de caractères.
- Tester vos fonctions depuis le programme principal main.

Exercice 3:

- Qu'affiche le programme suivant :

  ```
  #include <stdio.h>
  #include <string.h>
  #include <stdlib.h>
  #define taille 10

  int main(void)
  {
  ```

```
    char chaine[taille]={'L','E','N', 'S', '\0'};
    char chaine1[taille]={'A','R','R', 'A', 'S', '\0'};
    char *chaine2=malloc(taille * sizeof(char));
    *chaine2='A';
    *(chaine2+1)='R';
    *(chaine2+2)='R';
    *(chaine2+3)='A';
    *(chaine2+4)='S';
    *(chaine2+5)='\0';
    printf ("La taille de la chaine est  : %lu \n", strlen(chaine));
    printf ("Le résultat de la comparaison de deux chaines : %d \n", strcmp (chaine, chaine1));
    printf ("Le résultat de la comparaison de deux chaines : %d \n", strcmp (chaine, chaine2));
    printf ("Le résultat de la comparaison de deux chaines : %d \n", strcmp (chaine1, chaine2));
    return 0;
}
```

- Ecrire une fonction qui a comme entête

```
int comparer_chaine (const char* s1, const char* s2)
```

Cette fonction retourne :

- -1 : si s1 est strictement inférieur à s2
- 0 : si s1 est égale à s2
- 1 : s1 est strictement supérieur à s2

Remarque : Sauf indication contraire, dans les exercices sur les listes chaînées, donnés ci-dessous, nous utiliserons les déclarations suivantes :

```
struct _noeud
{
 int element;
 struct _noeud *suivant;
};

typedef struct _noeud noeud;
```

Toujours, sauf indication contraire, les listes considérées sont simplement chaînées.

Exercice 4:

Ecrire une fonction itérative qui imprime tous les éléments d'une liste chaînée (du premier élément de la liste jusqu'au dernier élément). Le pointeur qui contient l'adresse de début de la liste est donnée en paramètre.

Cette fonction a l'entête suivante :

```
void imprimer_liste_normal (noeud *tete)
```

Exercice 5 :

- Ecrire une fonction qui a l'entête suivante :

```
noeud *creation_de_liste(void)
```

  - Cette fonction ne prend pas de paramètres.
  - Elle crée une liste chaînée simple à partir d'entiers lus depuis une entrée standard (un clavier). La fonction lit donc, au fur et à mesure, des entiers puis les insère dans la liste. Les insertions se font au début de la liste. L'insertion des entiers s'arrête lorsqu'un nombre négatif est rentré (ce nombre négatif sert de condition d'arrêt et n'est donc pas inséré dans la liste).
  - La fonction retourne un pointeur vers la tête de la liste (l'adresse du maillon du premier élément inséré).

- Tester votre fonction depuis le main.
- Reprendre la fonction de création de liste précédente avec l'entête suivante :

```
void creation_de_liste_V2(noeud **tete)
```

- Tester votre nouvelle fonction depuis le main.

Exercice 6:

- Ecrire une fonction qui prend en paramètre une liste simplement chaînée et un entier nb. La fonction retourne 1 si l'entier nb est présent dans la liste et retourne 0 sinon

  Cette fonction a l'entête suivante :

```
int existe (noeud *tete, int nb){
```

- Proposer une version de la fonction précédente (y compris l'entête) où la fonction retourne l'adresse du maillon qui contient l'entier nb (si l'entier nb est présent dans la liste) et retourne NULL sinon

Exercice 7:

Considérons un échiquier de taille $N \times N$. Nous supposons que les lignes, ainsi que les colonnes, sont numérotées de 0 à $N-1$. Le but de cette exercice est de vérifier si deux reines placées sur un échiquier ne sont pas en conflit. Deux reines sont dites être en conflit si elles sont :

- sur la même ligne, ou
- sur la même colonne, ou
- sur la même diagonale.

Ecrire un programme C qui :

- déclare à l'aide de #define une constante N (qui représente la taille de l'échiquier);
- lit deux entiers positifs qui représentent les coordonnées (numéro de ligne, numéro de colonne) de la première reine;
- lit deux entiers positifs qui représentent les coordonnées (numéro de ligne, numéro de colonne) de la deuxième reine;
- affiche le nombre 1 si les deux reines sont en conflits et le nombre 0 sinon.

Nous supposons que l'échiquier ne contient que ces deux reines.

Exercice 8 :

Ecrire un programme qui résout, avec un minimum de déplacements de la case vide, le taquin 3×3.