Exercice 1 :

Exécuter le programme suivant :

```
1
2
3  #include <stdio.h>
4  int main(void)
5  {
6      int t[3]={10,-20,320};
7      printf ("les adresses de t et de t+1 sont  : %p et %p \n", &t, &t+1);
8      printf ("les adresses de t[0] et   t[0]+1 sont : %p et %p\n", &t[0], &t[0]+1);
9      printf ("les valeurs de t et t+1 sont : %p et %p \n", t, t+1);
10     return 0;
11 }
```

Que faut-il conclure ?

Exercice 2:

Ecrire une fonction qui a l'entête suivante :

```
int nbre_mot_de_taille_n (char *ch, int n)
```

Cette fonction retourne le nombre de mots de taille n (avec $n > 0$) dans une chaîne de caractères *ch*. Nous supposons que la chaîne de caractères ch est composée uniquement de lettre minuscules et du caractère blanc (' '). Un mot est une suite contingente de lettres minuscules (les mots sont donc séparés par un ou plusieurs caractères blanc).

Par exemple, considérons la chaine de caractères ch suivante :

```
"bonjour    vous avez pris place   dans       le train le plus rapide au monde         "
```

- si $n = 4$ alors la fonction retournera la valeur 5. En effet, il y a les cinq mots de taille 4 : "vous", "avez", "pris", "dans" et "plus".
- si $n = 7$ alors La fonction retournera la valeur 1. En effet, il n'y a qu'un seul mot de taille 7 : "bonjour".
- si $n = 8$ alors La fonction retournera la valeur 0. En effet, il n'y a aucun mot de taille 8.

Remarque : La fonction strlen permet de calculer la longueur d'une chaîne de caractères. N'oubliez pas d'inclure la bibliothèque $< string.h >$.

Exercice 3:

Ecrire une fonction, appelée strstr_fin, qui a l'entête suivante :

```
const char * strstr_fin( const char * grande, const char * petite )
```

Cette fontion (strstr_fin) recherche la **dernière** occurrence d'une sous-chaine (pointée par la variable petite) dans la chaîne qui se trouve à l'adresse grande. Si la sous-chaîne est trouvée, la fonction retourne un pointeur où se trouve l'adresse début de la dernière occurrence de la sous-chaine. La fonction retournera la valeur NULL dans le cas contraire.

Exercice 4:

Objectifs :

Nous nous intéressons dans cet exercice au jeu d'échec et plus particulièrement aux situations dites d'échec. Le but est d'écrire des fonctions qui vérifient si le roi d'un joueur se trouve dans une situation d'échec et mat. Nous décomposons le problème en sous-problèmes avec l'écriture des fonctions simples qui vérifient si un roi se trouve en situation d'échec par une tour, puis par un fou, puis par un cavalier et ainsi de suite.

Partie déclarations :

- Déclarer une constante N. Cette constante représente la taille de l'échiquier. Dans ce exercice, nous considérons que $N = 8$ (comme dans un jeu d'échec standard).
- Déclarer (dans le main) un tableau de caractères de taille $N \times N$.
- Intuitivement, ce tableau codera les différentes pièces d'un échiquier. De ce fait, nous utiliserons la convention suivante :
  - les lettres minuscules 't', 'f', 'k', 'p', 'q', 'c' sont utilisées pour désigner respectivement les pièces de l'échiquier "tour", "fou", "roi", "pion", "reine", "cavalier" du joueur humain;
  - les lettres majuscules 'T', 'F', 'K', 'P', 'Q', 'C' sont utilisées pour désigner respectivement les pièces de l'échiquier "tour", "fou", "roi", "pion", "reine", "cavalier" du joueur ordinateur (votre programme);
  - ' ' (blanc) pour désigner une case vide d'un échiquier.

Dans ce projet, il n'est pas demandé d'utiliser les fonctions pré-définies de la bibliothèque de chaînes de caractères "string.h".

I Fonctions préliminaires à écrire

- Ecrire une fonction, appelée **initialiser_echiquer**, qui initialise toutes les cases du tableau d'un échiquier avec le caractère ' ' (blanc).
- Ecrire une fonction, appelée **imprimer_echiquer**, qui imprime le tableau qui représente un échiquier (l'impression se fait ligne par ligne).
- Ecrire une fonction, appelée **est_entre_0_et_7**, qui prend en paramètre un caractère c. Si le caractère c est compris entre '0' et '7' alors la fonction retourne l'entier associé au caractère. Sinon elle retournera la valeur -1. Par exemple, si le paramètre c est égal à '3' alors la fonction retournera l'entier 3. Si le paramètre c est égal à '9' ou 'A' alors la fonction retournera l'entier -1.
  Vous pouvez utiliser l'opérateur du "conditionnement ternaire" pour traiter cet exercice.
- Ecrire une fonction, appelée **est_piece_echiquier**, qui prend en paramètre un caractère c et elle retourne 1 si le caractère c correspond à un caractère associé à une pièce d'un échiquier, c'est-à-dire si c appartient à l'ensemble {'t', 'f', 'k', 'p', 'q', 'c', 'T', 'F', 'K', 'P', 'Q', 'C'}. La fonction retourne 0 sinon.
  Il est recommandé d'utiliser l'instruction "switch" pour écrire cette fonction.
- Ecrire une fonction booléenne, appelée **lire_echiquier**, qui lit depuis le clavier toutes les pièces de l'échiquier et les enregistre dans un tableau (qui représente l'échiquier à lire) $N \times N$ de caractères. Le tableau sera passé en paramètres. La lecture se fera caractère par caractère. Il s'agit d'une suite de triplets de caractères, séparés par un espace et terminés par le caractère dollar. Chaque triplet est composé de deux caractères chiffres et d'un caractère lettre qui représente la pièce d'un échiquier. Une exemple de lecture de pièces d'un échiquier est :

  03r 12c 13q 14t 25T 23Q 27K 34k$

  Par exemple, le triplet 12c signifie que la pièce du joueur humain "cavalier" se trouve sur la ligne numéro 1 et colonne numéro 2 de l'échiquier. Les lignes ainsi que les colonnes sont numérotées de 0 à $N - 1$ (ici de 0 à 7).
  La fonction retourne 1 si la suite, de triplets, saisie est valide. Elle retourne 0 sinon.
  Il est demandé d'utiliser les fonctions **est_entre_0_et_7** et **est_piece_echiquier** pour écrire la fonction lire **lire_echiquier**. Par ailleurs, il est demandé de verifier qu'il ne peut pas exister deux triplets avec les mêmes coordonnées (lignes et colonnes). Par exemple :
  - La séquence :

    03q 12c 13k 14t 25T 23Q 27K 34k$

    est valide.
  - La séquence suivante :

    12c 13q 14t 25T 12Q 27K 34k$

    n'est pas valide (car la case qui se trouve sur la ligne numéro 1 et la colonne numéro 2 est précisée deux fois).
  - La séquence suivante :

    12c 13q 14t 25T 12Q 29K 34k$

    n'est pas valide pour n=8 (le triplet 29K n'est pas valide car la colonne 9 n'existe pas).

- Ecrire une fonction, appelée **nbre_occurrences**, qui prend en paramètre un tableau $N \times N$ de caractères (qui représente un échiquier), un caractère c et retourne le nombre d'occurrences de c dans l'échiquier.
- Ecrire une fonction booléenne, appelée **echiquier_valide**, qui prend en paramètre un tableau $N \times N$ de caractères (qui représente un échiquier) et retourne 1 si l'échiquier est valide. La fonction retourne 0 sinon. Un échiquier est dit valide si :
  - il existe exactement une seule pièce "roi" pour chaque joueur (humain et ordinateur),
  - il existe au plus 8 pièces "pion" pour chaque joueur,
  - il existe au plus 2 pièces "fou" pour chaque joueur,
  - il existe au plus 2 pièces "tour" pour chaque joueur,
  - il existe au plus 2 pièces "cavalier" pour chaque joueur, et
  - il existe au plus une pièce "reine" pour chaque joueur.

  Si l'échiquier n'est pas valide, un message d'erreur est imprimé.

Situations d'échec par une tour :

Le travail demandé dans cette section est d'écrire une fonction qui vérifie si la pièce "roi" du joueur humain est en situation d'échec par une pièce "tour" du joueur ordinateur.

- Nous rappelons que la tour se déplace (d'une ou de plusieurs cases libres) en horizontal et en vertical comme indiqué dans le schéma suivant :

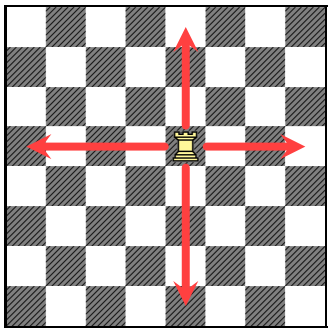

Figure 1: Déplacements possibles d'une tour

- Nous rappelons que la pièce "roi" du joueur humain est en situation d'échec par une pièce "tour" du joueur ordinateur, si les deux pièces se trouvent sur la même ligne ou sur la même colonne et aucune autre pièce ne se trouve entre les pièces "roi" et "tour". Elle retourne faux sinon.
- Ecrire une fonction **est_echec_tour** qui prend en paramètre :
  - un tableau "echiquier" de caractères de taille NxN,
  - les coordonnées x_T et y_T de la pièce "tour" du joueur ordinateur.

  La fonction retourne la valeur 1 si la pièce "roi" du joueur humain est en situation d'échec par la pièce "tour" du joueur ordinateur. Elle retournera -1 sinon (elle retournera également -1 si la pièce "tour" du joueur ordinateur n'est pas à la case x_T et y_T).

Exercice 5:

Nous reprenons notre exercice sur le jeu d'échec (voir l'énoncé du TP précédent) et nous nous intéressons plus particulièrement aux situations dites d'échec.

Nous demandons d'écrire une fonction **est_echec_cavalier** qui permet de vérifier si la pièce la "roi" du joueur humain est en une situation d'échec par une pièce "cavalier" du joueur ordinateur.
Nous rappelons que le cavalier effectue des déplacements dits en "L", comme indiqué dans le schéma suivant :

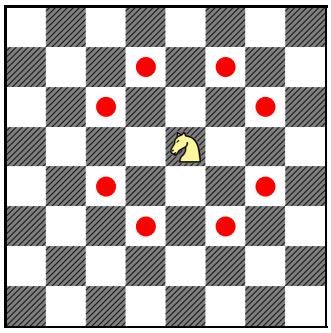

Figure 2: Déplacements possibles d'un cavalier. Les cases représentées par des cercles pleins rouges indiquent les huit positions interdites de la pièces "roi" du joueur humain.

Exercice 6:

La fonction "free" permet de libérer un espace mémoire (que nous supposons, dans cet exercice, composé des entiers) pointé par p et qui a été préalablement alloué de manière dynamique. Cependant, la fonction free ne met pas la valeur du pointeur p à NULL.

- Ecrire une fonction, appelée liberer, qui permet de libérer un espace mémoire pointé par p et qui réinitialise la variable p à NULL. La fonction ne retourne aucune valeur.
- Tester votre fonction depuis la fonction mail.

Indications : Le travail demandé consiste à compléter le programme suivant :

```
#include <stdio.h>
#include <stdlib.h>
#define Taille 2000

void liberer (...){
    ......
}

int main (void)
{
    void libérer (....);
    int *p=(int *) malloc (Taille*sizeof (int));
    liberer (....);
    printf ("La nouvelle valeur du pointeur est : %p \n", p);
    return 0;
}
```