TP Programmation structurée en C

L'objectif de ce TP est de réaliser un projet en langage structuré C. La réalisation du projet doit s'effectuer en groupe de 4 au moins et de 5 étudiants au plus, mais la notation est individuelle. Ainsi la répartition des tâches doit être clairement identifiée. Vous serez évalué sur la qualité du programme produit (modularité, clarté, documentation...) et sur le bon emploi de l'environnement de développement. Chaque projet doit contenir les noms et matricules des membres du groupe. Chaque groupe doit réaliser et présenter au moins cinq (05) projets.

Projet 1 : Mise à jour d'un vecteur

Soit A un tableau des entiers naturels. Écrire un programme qui fait des mises à jour dans le tableau.

Travail demandé :

a) Écrire une procédure/fonction *Ajout*() qui ajoute un entier Val à A.
b) Écrire une procédure/fonction *Supprime*() qui supprime la première occurrence d'un entier Val dans A.
c) Écrire une procédure/fonction *Position*() qui cherche et indique la position de la première occurrence d'un entier Val dans A.
d) Range les éléments de A par ordre décroissant.

Écrire le programme principal qui fait un appel répétitif à l'une des procédures/fonctions selon le choix de l'utilisateur.

N.B : La *suppression* d'un élément du tableau implique de décaler la « *case libre* » vers la fin du tableau afin de supprimer les « *trous* » du tableau.

Projet 2 : Manipulation d'un vecteur trié

Soit A un tableau des entiers naturels. Écrire un programme qui range les éléments de A par ordre croissant, puis fait des mises à jour dans le tableau.

Travail demandé :

a) Écrire une procédure/fonction TriBulle() qui range les éléments de A par ordre croissant, selon le principe du tri par échange.
b) Écrire une procédure *Ajout*() qui ajoute un entier Val à A (déjà trié) et A reste trié.
c) Écrire une fonction *Position*() qui fait le cherche dichotomique sur A et indique la position de Val dans A.

Écrire le programme principal qui fait appel à TriBulle(), puis un appel répétitif à l'une des autres procédures/ fonctions selon le choix de l'utilisateur.

Projet 3 : Tableaux des entiers

Soient A et B deux tableaux des chiffres {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} de taille respective *n* et *m* représentant 2 entiers naturels. On demande de calculer et afficher $C = A + B$; $D = A - B$.

Travail demandé :

a) Écrire une procédure/ fonction *Addition*() qui calcule la somme de A et B dans un autre tableau C.
b) Écrire une procédure/fonction *Difference*() qui calcule la différence entre les tableaux A et B dans un autre tableau D.

Écrire le programme *main*() qui fait un appel répétitif de *Addition*() ou *Difference*() selon le choix de l'utilisateur.

Projet 4 : Calculatrice

Programmer votre propre calculatrice :

Travail demandé :

a) Écrire un programme qui, selon le choix de l'utilisateur, fait l'addition, la soustraction, la multiplication, la division des 2 réels.
b) Écrire une procédure *Interface*() qui dessine l'interface graphique d'une calculatrice.

Écrire le programme principal qui appel *Interface*() et permet à l'utilisateur de faire l'addition, la soustraction, la multiplication, la division des 2 réels de façon répétitive.

Projet 5 : Le Temps

Le temps est une structure composé d'heure, de minute et de seconde (h : m : s). On doit mettre en œuvre une structure dont les champs sont h, m et s, tous des entiers non signés.

```
typedef struct Temps  {
    unsigned short h ;
    unsigned short m ;
    unsigned short s ;
} ;
```

Travail demandé:

a) Écrire une fonction qui prend en entrée une structure temps t1 et affiche 1 si c'est un temps valide et 0 sinon.
b) Écrire une procédure qui prend en entrée un temps et affiche le temps qui sera dans *s* secondes.
c) Écrire une fonction qui prend en paramètre deux temps t1 et t2 et renvoie la somme de ces deux temps.
d) Écrire une fonction qui prend en paramètre deux temps t1 et t2 et renvoie 1 s'ils sont égaux et 0 sinon.

Écrire le programme principal qui fait un appel répétitif à l'une des procédures/ fonctions selon le choix de l'utilisateur.

Projet 6 : Courbe

Soit $P(x) = a_0 + a_1x + a_2x^2 + a_3x^3$un polynôme de degré 3. Sous la forme de Hörner $P(x)$ peut s'écrire : $P(x) = a_0 + x(a_1 + a_2x + a_3x^2) = a_0 + x(a_1 + x(a_2 + xa_3))$. On suppose que les coefficients du polynôme $P(x)$ sont dans le tableau A. Utiliser cette forme pour calculer $P(x_0)$.

Travail demandé :

a) Écrire une fonction qui prend en entrée un réel x_0 et le vecteur A et calcule $P(x_0)$.
b) Tracer la courbe d'une fonction polynomiale de degré ≤ à 3.

Écrire le programme principal qui demande à l'utilisateur de faire entrer les coefficients d'un polynôme, le stock dans le vecteur A et trace la courbe du polynôme.

Projet 7 : Calcul de sin (x)

Le développement de $\sin(x) = x - \frac{x^3}{3!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$ et $\cos(x) = 1 - \frac{x^2}{2!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$. Pour cela on pose $\text{Un} = (-1)^n \times \frac{x^{2n+1}}{(2n+1)!}$ et $\text{Vn} = (-1)^n \times \frac{x^{2n}}{(2n)!}$. Trouver les relations *rs et rc* entre deux termes consécutifs des suites Un et Vn en calculant $\text{rs} = \frac{U_{n+1}}{U_n}$ et $\text{rc} = \frac{V_{n+1}}{V_n}$.

Travail demandé :

a) Écrire une fonction *sin*() qui prend en entrée une valeur réel *x* et une précision $\varepsilon = 0.0001$ et calcule la valeur de *sin*(*x*) jusqu'à l'ordre *n* tel que $\left|\frac{U_{n+1}}{U_n}\right| < \varepsilon$.

b) Écrire une fonction *cos*() qui prend en entrée une valeur réel *x* et une précision $\varepsilon = 0.0001$ et calcule la valeur de *cos*(*x*) jusqu'à l'ordre *n* tel que $\left|\frac{V_{n+1}}{V_n}\right| < \varepsilon$.

Écrire le programme principal qui, de façon répétitive, demande à l'utilisateur de faire entrer un réel *x* puis calcule sin(*x*) ou cos(*x*) selon le choix de l'utilisateur.

Projet 8 : Tableau des complexes

Un nombre complexe est une structure composée d'une partie réelle et d'une autre imaginaire. Il est de la forme *x* + *iy*, où *x* et *y* sont des nombres réels, et *i* un « nombre imaginaire » tel que $i^2 = -1$.

```
typedef struct Complex  {
    float Reel ;
    float Imag;
} ;
```

Travail demandé :

a) On demande de créer un tableau Tab dont les éléments sont des complexes.

b) Écrire une fonction *Norme*() qui prend en entrée un nombre complexe C= *x* + *iy* et renvoie en sortie sa norme norm = $\sqrt{x^2 + y^2}$.

Écrire le programme principal qui trie les éléments du Tab selon l'ordre croissant de leurs normes.

Projet 9 : Tableau des réels

On demande de créer un tableau Tab dont les éléments sont les normes des complexes. Un nombre complexe est une structure composée d'une partie réelle et d'une autre imaginaire. Il est de la forme *x* + *iy*, où *x* et *y* sont des nombres réels, et *i* un « nombre imaginaire » tel que $i^2 = -1$. La norme d'un nombre complexe C= *x* + *iy* est norm = $\sqrt{x^2 + y^2}$.

Travail demandé :

a) Écrire une fonction *Norme*() qui prend en entrée un nombre complexe C= *x* + *iy* et renvoie en sortie sa norme norm = $\sqrt{x^2 + y^2}$.

b) Écrire une fonction qui permet de retrouver la position de la plus petite et la plus grande norme dans tableau.

Écrire le programme principal qui demande d'entrer une série des nombres complexes et range leurs normes dans un tableau Tab. La série est un ensemble de nombre complexe donc le somme des normes est strictement inférieur à 100.

Projet 10 : Les nombres palindromes

Un nombre sera dit « palindrome » s'il peut se lire de gauche à droite et de droite à gauche en représentant le même nombre. Les nombres suivants sont palindromes : 425524, 121, 3, etc.

Travail demandé :

a) Écrire une fonction **extrait** qui reçoit un entier (nb) en paramètre et qui :
   - retourne –1 et ne modifie pas nb si nb<10
   - retourne le 1er chiffre de nb et modifie nb en retirant le chiffre de plus fort poids. Par exemple extrait(4657) retourne 4 et nb vaut au retour 657.

b) Écrire, en utilisant la fonction précédente une fonction « booléenne » palindrome qui recevant un entier en paramètre retourne vrai ou faux selon que cet entier est un palindrome ou non

Écrire le programme principal qui demande de façon répétitive un entier non signé à l'utilisateur et dit si c'est un palindrome ou non.

Projet 11 : Le plus grand diviseur commun de deux entiers positifs

Le plus grand diviseur commun de deux entiers positifs non tous nuls *m* et *n*, dénoté $\gcd(m,n)$, est défini comme le plus grand entier naturel qui divise à la fois *m* et *n*. Il y a plusieurs algorithmes pour calculer le plus grand diviseur commun.

1. Euclide of Alexandria a proposé un algorithme pour résoudre ce problème dans l'un des volumes de ses *Elements*, plus fameux de sa présentation systématique de la géométrie. En termes modernes, l'**Algorithme d'Euclide** est basé sur l'application répétitive de la relation : $\gcd(m,n) = \gcd(n, m \bmod n)$. (où *m* mod *n* est le reste de la division euclidienne de *m* par *n*) jusqu'à ce que *m* mod *n* soit égal à zéro. Comme $\gcd(m,0) = m$ (pourquoi ?), la dernière valeur de *m* est aussi le plus grand diviseur commun de *m* et *n*.

Algorithme d'Euclide pour calculer GCD(*m*, *n*)

Etape 1.	Si *n* = 0, retourner la valeur de *m* comme réponse et **STOP** sinon continuer à l'**Etape 2**.
Etape 2.	Diviser *m* par *n* et affecter la valeur du reste à *r*.
Etape 3.	Affecter la valeur de *n* à *m* et la valeur de *r* à *n*. Aller à l'**Etape 1**.

2. La deuxième est simplement basée sur la définition du plus grand diviseur commun de *m* et *n* comme le plus grand entier qui divise les deux nombres. Clairement, un tel diviseur commun ne peut pas être plus grand que le plus petit de ces nombres, que nous dénotons par $p = \min\{m, n\}$. Ainsi on commence par vérifier si *p* divise les deux nombres, si oui *p* est la réponse, sinon, on décrémente *p* et on essaie encore.

Algorithme de vérification des entiers consécutifs pour le calcul de GCD(*m*, *n*)

Etape 1.	Affecter à *p* la valeur de min{*m*, *n*}
Etape 2.	Diviser *m* par *p*. Si le reste de cette division est zéro, alors aller à l'**Etape 3**, sinon aller à l'**Etape 4.**
Etape 3.	Diviser *n* par *p*. Si le reste de cette division est zéro, retourner la valeur de p comme réponse et **STOP**, sinon continuer à l'**Etape 4**.
Etape 4.	Décrémenter la valeur de p. Aller à l'**Etape 2**.

3. La troisième procédure pour le calcul du plus grand diviseur commun sera familière aux élèves des classes intermédiaires.

Procédure des classes intermédiaires pour le calcul de GCD(*m*, *n*)

Etape 1.	Trouver la décomposition en facteurs premiers de *m*.
Etape 2.	Trouver la décomposition en facteurs premiers de *n*.
Etape 3.	Identifier tous les facteurs communs des décompositions en facteurs premiers trouvées à l'**Etape 1** et à l'**Etape 2**. Si *p* est un facteur commun apparaissant p_m fois et p_n fois dans *m* et *n*, respectivement, il sera répété $\min\{p_m, p_n\}$ fois.
Etape 4.	Calculer le produit de tous les facteurs communs et retourner ce produit comme le plus grand diviseur commun des nombres *m* et *n*.

Présentons maintenant un algorithme simple pour générer les nombres premiers consécutifs inférieurs à un entier donné *n*. L'algorithme commence par initialiser la liste des nombres premiers candidats par les entiers consécutifs de 2 à *n*. Ensuite, à la première itération de l'algorithme, on élimine de la liste tous les multiples de 2. Ensuite on passe au deuxième élément de la liste qui est 3, et on élimine tous ses multiples. Le prochain nombre restant dans la liste et qui est utilisé à la troisième itération est 5. Comme pour 2 et 3, tous les multiples de 5 sont supprimés de la liste. L'algorithme continue de cette façon jusqu'à ce qu'aucun nombre ne puisse plus être supprimé de la liste. Les nombres restant de la liste sont les nombres premiers recherchés.

Travail demandé :

a) Écrire une fonction *PGCDEuclide*() qui reçoit deux positifs non tous nuls *n* et *m* en paramètre et utilise la méthode **d'Euclide** pour retourner le plus grand diviseur commun de *m* et *n*.

b) Écrire une fonction *PGCDSoustraction*() qui reçoit deux positifs non tous nuls *n* et *m* en paramètre et utilise la méthode de **vérification des entiers consécutifs** pour le calcul de plus grand diviseur commun de *m* et *n* tous non nuls.

c) Écrire une fonction *Eratosthenes*() qui recoit *n* entier positif et renvoie un tableau des nombres premiers consécutifs inférieurs à un entier donné *n*.

d) Écrire une fonction *PGCDClasseIntermediaire*() qui reçoit deux positifs non tous nuls *n* et *m* en paramètre et utilise la **procédure des classes intermédiaires** pour le calcul pour le calcul de plus grand diviseur commun de *m* et *n* tous non nuls.

Projet 12 : Le problème de la paire la plus proche

Le problème de la paire la plus proche demande de trouver deux points les plus proches dans un ensemble de *n* points. Pour simplifier, nous considérons le cas bidimensionnel, même si le problème peut aussi bien se poser pour des points appartenant à des espaces de dimension supérieure. Nous supposons que les points en question se spécifiés dans une forme standard par leurs coordonnées cartésiennes (*x*, *y*) et que la distance entre deux points $P_i = (x_i, y_i)$ et $P_j = (x_j, y_j)$ est la distance euclidienne standard

$$d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

L'approche de la force brute pour résoudre ce problème conduit à l'algorithme évident suivant : calculer la distance entre chaque paire de points distincts et trouver la paire qui a la plus petite distance. Naturellement, nous n'avons pas besoin de calculer deux fois la distance associée à la même paire de points. Pour éviter de le faire, nous considérons uniquement les paires de points (P_i, P_j) pour lesquels $i < j$.

```
typedef struct Point  {
    float x ;
    float y ;
} ;
```

Travail demandé :

a) Écrire une fonction *DistanceEuclidienne*( ) qui renvoie la distance euclidienne standard entre deux points.

b) Écrire une fonction *PlusProchePaire*( ) qui prend en paramètre un entier naturel *n* et un tableau *Tab* à deux dimensions et renvoie deux points (coordonnées) les plus proches.

Projet 13 : Décomposition en produit de nombres premiers

Le but de l'exercice est d'écrire un programme qui décompose un entier en produit de nombres premiers. Par exemple **2904** $=2^3 \times 3^1 \times 11^2$. Si *n* est un entier naturel non nul et *p* est un nombre premier, on note $v_p(n)$ le plus grand entier *k* tel que p^k divise *n*. Si *p* ne divise pas *n*, on pose $v_p(n) = 0$. L'entier $v_p(n)$ s'appelle la valuation p-adique de *n*.

Travail demandé :

a) Écrire une fonction booléenne *estPremier*(*n*) qui prend en argument un entier naturel non nul *n* et renvoie *true* si *n* est premier et *false* sinon. On pourra utiliser le critère suivant :
- 0 et 1 ne sont pas des nombres premiers ;
- 2 est un nombre premier ;
- tout entier n >2 qui n'est divisible par aucun entier d tel que $d \geq 2$ et $d^2 \leq n$ est premier.

b) Écrire une fonction *listePremiers*(*n*) qui prend en argument un entier naturel $n > 2$ et renvoie un tableau des nombres premiers inférieurs ou égaux à *n*.

c) Pour calculer la valuation 2-adique de 40, on peut utiliser la méthode suivante :
- 40 est divisible par 2 et le quotient vaut 20
- 20 est divisible par 2 et le quotient vaut 10
- 10 est divisible par 2 et le quotient vaut 5
- 5 n'est pas divisible par 2. La valuation 2-adique de 40 vaut donc 3.

Écrire une fonction *valuation_p_adique*(*n, p*) non récursive qui implémente cet algorithme. Elle prend en arguments un entier naturel *n* non nul et un nombre premier *p* et renvoie la valuation p-adique de *n*. Par exemple, puisque $40=2^3\times5$:

- *valuation_p_adique*(40, 2) renvoie 3 ;
- *valuation_p_adique(40, 5) renvoie 1 ;*
- *valuation_p_adique*(40, 7) renvoie 0.

d) Écrire une fonction *decomposition_facteurs_premiers*(*n*) qui calcule la décomposition en facteurs premiers d'un entier $n\geq2$. Cette fonction doit renvoyer un tableau de deux ligne des couples (p, vp(n)) pour tous les nombres premiers p qui divisent n. Par exemple, *decomposition_facteurs_premiers*(40) renvoie le tableau Tab suivant.

Tab

2	5	Ligne de nombre premier *p*,
3	1	Ligne la valuation p-adique vp(*n*))

e) Écrire une fonction *AfficheDecomposition*(*Tab*) qui prend en paramètre un tableau représentant la décomposition en facteurs premiers d'un entier et renvoie la valeur de *n*, tout en affichant sa décomposition sous la forme $[(p_1, k_1), (p_2, k_2), \ldots, (p_n, k_n)]$. Par exemple, pour le tableau suivant

Tab

2	3	11
3	1	2

La fonction *AfficheDecomposition*(*Tab*) renvoie **2904** et affiche [(2, 3), (3, 1), (11, 2)].

Projet 14 : Système d'équation et Equation du 2nd degré

Travail demandé :

a) Écrire un algorithme qui lit six nombres réels *a, b, c, d, e* et *f* et calcule la solution du système d'équations $\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$.

b) Écrire un algorithme qui lit trois nombres réels *a, b* et *c* puis calcule les solutions réelles de l'équation quadratique $ax^2 + bx + c = 0$.

Projet 15 : Nombre de jours sur terre

On veut écrire un algorithme pour calculer le jour qu'une personne a déjà vécu connaissant la date du jour et sa date de naissance.

Une année est bissextile si elle est multiple de 4, et si de plus elle est multiple de 100 elle doit aussi être multiple de 400. *Par exemple* :

- ✓ 1985 **non,** *car elle n'est pas multiple de* 4 *c'est-à-dire* 1985 *modulo* $4 \neq 0$;
- ✓ 1996 **oui,** *car* 1996 *modulo* 4 =0 *et* 1996 *n'est pas multiple de* 100;
- ✓ 1900 **non,** *car* 1900 *modulo* 4 = 0; 1900 *modulo* 100 = 0 mais 1900 *modulo* $400 \neq 0$
- ✓ 2000 **oui**, *car* 2000 *modulo* 4 = 0; 2000 *modulo* 100 = 0 et 2000 *modulo* 400 = 0.

Travail demandé :

```
typedef struct Date {
    unsigned short Jour ;
    unsigned short Mois ;
    unsigned short Annee ;} ;
```

a) Écrire une fonction *Bissextile*() qui prend en argument une année, puis renvoie 1 si elle est bissextile et 0 sinon.
b) Écrire une fonction *NbrJoursMois*() qui prend en argument *Mois* et *Année*, puis renvoie le nombre de jours que *Mois* a durant l'année *Année*.
c) Écrire une fonction *ValideDate*() qui prend en entrée une structure Date D et affiche 1 si c'est une date valide et 0 sinon.
d) Écrire une fonction *NbrJoursAnnee*() qui calcule le nombre de jours de l'année connaissant le mois, le jour du mois et l'année. Par exemple, si l'entrée de l'algorithme est « 02/07/1983 » la sortie de la fonction serait 38 car « le 02/07/1983 est le 38ème jour de l'année 1983 ».

Écrire le programme principal qui demande la date de naissance *Dn* d'une personne puis, fait appel à ses fonctions pour déterminer le nombre de jour d'une personne à partir de la date courante.

Projet 16 : Pyramide

Créer un programme permettant d'afficher un triangle isocèle formé d'étoiles de N lignes (N étant fourni au clavier) :

Nombre de lignes : 8

```
       *
      *o*
     *ooo*
    *ooooo*
   *ooooooo*
  *ooooooooo*
 *ooooooooooo*
***************
```

Projet 17 : Damier

On veut faire afficher un damier de *n* par *n* cases. Les cases noires (resp. blanches) sont représentées par *p* * *p* un caractère *C1* (resp. *C2*). Par exemple pour *n*=4, *p* = 3, *C1*=&, *C2*=+, le programme doit afficher :

```
&&&+++&&&+++
&&&+++&&&+++
&&&+++&&&+++
+++&&&+++&&&
+++&&&+++&&&
+++&&&+++&&&
&&&+++&&&+++
&&&+++&&&+++
&&&+++&&&+++
+++&&&+++&&&
+++&&&+++&&&
+++&&&+++&&&
```

Les nombres *n* et *p*, les caractères *C1* et *C2* sont choisis par l'utilisateur (*n* et *p* sont supposés être inférieurs ou égaux à 9).

Travail demandé :

a) Écrire le programme de telle façon que :
- ✓ L'entrée des données soit "sécurisée" en ce sens qu'en cas de faute de frappe lors de l'acquisition des nombres *n* et *p*, le programme ne s'arrête pas mais qu'au contraire il demande à nouveau les valeurs *n* ou *p* (par exemple on frappe sur la touche "U" au lieu de la touche "8"). De façon similaire, on s'assurera que le nombre *n* est pair et que les caractères *C1* et *C2* sont différents.
- ✓ Après chaque affichage, le programme "recommence". Il s'arrêtera lorsque la valeur donnée à *n* est nulle.

b) Écrire trois versions de la partie affichage du damier proprement dit n'utilisant que des structures itératives ("pour", "tant que", "jusqu'à ce que"). L'exécution de ces trois versions sera contrôlée par un menu interactif. Par exemple sur l'écran s'affichera le teste suivant :
- ✓ Voulez-vous afficher le damier :
  - en utilisant la structure pour (réponse a ou A) ;
  - en utilisant la structure tant que (réponse b ou B) ;
  - en utilisant la structure pour (réponse c ou C) ;
  - acquérir de nouvelles valeurs pour n, p, C1 et C2 (réponse d ou D).
- ✓ Si la réponse est a, b ou c le damier sera affiché puis le menu réapparaîtra. Si la réponse est d on redemandera les paramètres du problème puis le menu réapparaîtra. Si la réponse est autre, le menu réapparaîtra.

Indications : l'impression se fait caractère par caractère. Pour ce faire on peut utiliser soit la fonction printf() soit la fonction putchar(). Par exemple pour imprimer la lettre 'a', on peut écrire :
- Soit putchar ('a');
- Soit printf("%c",'a');

Projet 18 : Nombre de mots

L'objectif de ce TP est de développer un programme permettant de trier et compter les mots qui sont contenus dans un fichier texte.

Notions utilisées

* Les chaînes de caractères
* Les fichiers
* Les structures

Travail demandé :

a) Il s'agit de lire un fichier contenant du texte, de trier les mots par ordre lexicographique croissant, puis d'afficher la liste triée des mots du texte en indiquant pour chacun d'entre eux leur nombre d'occurrences. Plus précisément, le programme doit permettre :
- ✓ l'*ouverture en lecture d'un fichier contenant un texte*, fichier dont le nom est indiqué sur la ligne de commande. Dans le cas où l'utilisateur omettrait de donner un nom de fichier d'entrée sur la ligne de commande, le programme demande à l'utilisateur d'indiquer le texte directement par le clavier ; il faudra alors terminer la saisie avec une indication de "fin de fichier".

- ✓ le *tri des mots* et le *nombre d'occurrences de chaque mot* ; le tri utilise la dichotomie pour chercher la place d'un mot avant de l'insérer dans le tableau trié s'il n'existait pas encore, ou bien d'incrémenter son nombre d'occurrences.
- ✓ l'*affichage à l'écran de la liste triée des mots*, un mot par ligne, avec pour chaque mot, son *nombre d'occurrences*.

Indications

1/ Exemple de lecture d'un fichier

```
#include "stdio.h"

void main() {

      FILE * fic;
      char c;
      // ouverture du fichier
      fic=fopen("texte.txt","r");
      // boucle sur les caractères lus
      while((c=fgetc(fic)) != EOF) {
            // dans cette boucle, c contient successivement chaque
caractère lu
            printf("%c",c);
      }
}
```

2/ La fonction `strtok()` déclarée dans `string.h` permet de découper une chaîne de caractères en "morceaux" en se basant sur des séparateurs. Pour commencer ce TP, vous pouvez partir du programme ci-dessous qui lit une ligne envoyée au clavier et la découpe avec la fonction `strtok()`.

```
#include <string.h>
#include <stdio.h>
char string[] = "A string\tof ,,tokens\nand some  more tokens";
char seps[]   = " ,\t\n";
char *token;
void main( void )
{
       printf( "%s\n\nTokens:\n", string );
       /* Establish string and get the first token: */
       token = strtok( string, seps );
       while( token != NULL )
       {
              /* While there are tokens in "string" */
              printf( " %s\n", token );
              /* Get next token: */
              token = strtok( NULL, seps );
       }}
```

3/ La fonction `strcmp()` prototypée dans `string.h` compare des chaînes de caractères selon l'ordre lexicographique.

4/ La fonction `strlen()` prototypée dans `string.h` permet de connaître la longueur d'une chaîne de caractères.

Consignes

Pour coder le mot du texte et le nombre d'occurrences de ce mot dans le texte, on utilisera la structure de données suivante :

```
typedef struct CompteMot {
        char mot[20];
        int nbOcc;
};
```

On utilisera un tableau statique pour CompteMot. On peut lire le texte ligne par ligne avec la fonction fgets().

Exemple d'exécution

Supposons que le programme ait été lancé avec un seul argument indiquant le nom d'un fichier contenant le texte suivant :
Bonjour, comment vas-tu ? Très bien. Et toi, comment vas-tu ?
Alors, nous obtiendrons comme résultat :
Bonjour : 1 fois ; Et : 1 fois ; Très : 1 fois ; bien : 1 fois
comment : 2 fois
toi : 1 fois
tu : 2 fois
vas : 2 fois

Projet 19 : Triangle de Pascal

Écrire un programme qui construit le triangle de PASCAL de degré N et le mémorise dans une matrice carrée P de dimension N+1. Une matrice carré est de Pascal si elle est définit par :

- ✓ $M(n,0) = M(n,n) = 1$
- ✓ $M(n,k) = 0$ Si $n < k$
- ✓ $M(n,k) = M(n-1,k-1) + M(n-1,k)$, Si $n > k$ et $k > 0$

Exemple de matrice de Pascal pour N=9

1	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	2	1	0	0	0	0	0	0	0
1	3	3	1	0	0	0	0	0	0
1	4	6	4	1	0	0	0	0	0
1	5	10	10	5	1	0	0	0	0
1	6	15	20	15	6	1	0	0	0
1	7	21	35	35	21	7	1	0	0
1	8	28	56	70	56	28	8	1	0
1	9	36	84	126	126	84	36	9	1

Travail demandé :

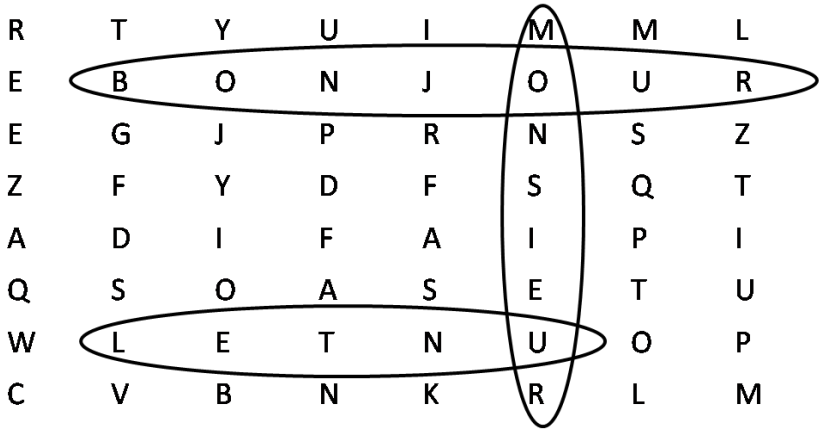
a) Écrire une fonction qui prend en entrée une matrice et renvoie *true* ou *false* selon que la matrice soit de Pascal ou non.

b) Écrire une fonction qui prend en entrée une matrice nulle et la transforme en une matrice de Pascal.

Projet 20 : Mot dans une matrice de caractères

Écrire un programme capable de trouver un mot à l'intérieur d'une matrice de caractères. Le programme devra tout d'abord demander à l'utilisateur la matrice de caractères de dimension N×N (N sera demandé à l'utilisateur). Il demandera ensuite le mot à rechercher, la longueur M du mot devra être inférieure à N. Le programme devra rechercher la présence du mot à l'intérieur de la matrice, horizontalement de gauche à droite et de droite à gauche et verticalement de haut en bas et de bas en haut. Si le mot est trouvé, le programme affichera les coordonnées du caractère de début de la chaîne et sa direction. Si le mot n'est pas trouvé, un message sera affiché.

Exemple de matrice et de mots trouvés:



Groupe_1 = {1, 20, 3, 18, 5, 17}	Groupe_11 = {11, 10, 13, 8, 15, 7}
Groupe_2 = {2, 1, 4, 19, 6, 18}	Groupe_12 = {12, 11, 14, 9, 16, 8}
Groupe_3 = {3, 2, 5, 20, 7, 19}	Groupe_13 = {13, 12, 15, 10, 17, 9}
Groupe_4 = {4, 3, 6, 1, 8, 20}	Groupe_14 = {14, 13, 16, 11, 18, 10}
Groupe_5 = {5, 4, 7, 2, 9, 1}	Groupe_15 = {15, 14, 17, 12, 19, 11}
Groupe_6 = {6, 5, 8, 3, 10, 2}	Groupe_16 = {16, 15, 18, 13, 20, 12}
Groupe_7 = {7, 6, 9, 4, 11, 3}	Groupe_17 = {17, 16, 19, 14, 1, 13}
Groupe_8 = {8, 7, 10, 5, 12, 4}	Groupe_18 = {18, 17, 20, 15, 2, 14}
Groupe_9 = {9, 8, 11, 6, 13, 5}	Groupe_19 = {19, 18, 1, 16, 3, 15}
Groupe_10 = {10, 9, 12, 7, 14, 6}	Groupe_20 = {20, 19, 2, 17, 4, 16}