NOM :	Prénom :

INF102 – Programmation en C

Durée : 1h30,
Nombre de pages : 9
Documents et calculatrices autorisés

Les exercices seront réalisés directement sur les sujets.

Exercice 1 – Mystère *(2 points)*
On considère le programme en C suivant :

```c
#include <stdio.h>

char Mystere(char c, int code){
	switch(code){
	case 1:{
		if(c>='a' && c<='z')
			return (c - ('a'-'A'));
		else return c;
		} break;
	case -1:{
		if(c>='A' && c<='Z')
			return (c + ('a'-'A'));
		else return c;
		 }
		break;
	}
}
// fonction principale
int main()
{
	char s[100];
	printf("Entrez uns phrase\n");
	gets(s);
	int i=0;
	int j=1;
	while(s[i] != '\0'){
		while (s[i] != ' ' && s[i] != '\0'){
		s[i] = Mystere(s[i], j);
		i++;
		}
		j=-j;
		i++;
	}
	printf ("%s\n", s);
	return 0;
}
```

Q 1.1) Que fait la fonction Mystere?

Q 1.2) Quel est le résultat qu'il affiche (la fonction principale) si l'on saisit la chaine suivante :
"**Ceci Est Un Exemple**"

Q 1.3) Que fait ce programme ?

Exercice 2 – Chaînes de caractères (*4 points)*

Q 2.1) Écrire une fonction *int occurenceLettre(char c, char chaine[])* qui retourne le nombre d'occurrences de la lettre *c* donnée dans une chaîne de caractères *chaine*, sans distinction entre les minuscules et les majuscules.

PS : Le nombre d'occurrences d'une lettre correspond à son nombre d'apparition dans la chaîne. Par exemple, dans la chaîne "Ceci est Exemple", la lettre 'e' (ou E) a une fréquence de 5 car elle apparait 5 fois dans la chaîne. Par contre, la fréquence de la lettre 't' dans cette même chaîne vaut 1 tandis que celle de la lettre 'd' vaut 0.

```
int occurenceLettre ( char c, char chaine[] ) {

}
```

Q 2.2) Écrire une fonction *int numbreMots (char chaine[])* qui retourne le nombre de mots contenus dans la chaîne de caractères *chaine*.

```
int numbreMots ( char chaine[] ) {
```

}

Q 2.3) Écrire une fonction *int SupprimeEspace (char chaine[])* qui supprime tous les espaces dans une chaîne de caractères *chaine*.

int SupprimeEspace (char chaine[]){

}

Exercice 3 – Tableaux (*4 points)*

Q 3.1) Écrire une fonction *float Produit (float Tab[])* qui calcule retourne le produit des éléments du tableau.

float Produit (float Tab[]) {

}

Q 3.2) Écrire une fonction *float Ecarttype (float Tab[])* qui calcule retourne l'écart-type de ce tableau (des éléments dans le tableau).

```
float Ecarttype (float Tab[]) {




}
```

Q 3.3) Écrire une procédure void *Transpose (int M[m][n], int Trans[n][m])* qui permet de calculer la matrice transposée *Trans* d'une matrice *M* passée en paramètre.

$$M = \begin{bmatrix} 1 & 4 & 5 \\ 2 & 9 & 5 \\ 4 & 3 & 7 \end{bmatrix} \quad Tranpose(M) = \begin{bmatrix} 1 & 2 & 4 \\ 4 & 9 & 3 \\ 5 & 5 & 7 \end{bmatrix}$$

```
void Transpose (int M[m][n], int Trans[n][m]) {




}
```

Exercice 4 – Mon réfrigérateur végétalien 2.0 *(7 points)*

L'objectif de cette partie est d'implémenter un logiciel permettant de gérer automatiquement le stockage d'un réfrigérateur végétalien.
Les fonctions à développer devront permettre de :

- Lister **tous les fruits et légumes** présents dans le frigo. ;
- Ajouter un **fruit** dans le frigo. ;
- Mettre à jour la date de consommation de tous les **légumes**;
- Supprimer automatiquement les **légumes** non consommables.

La structure Fruit (resp. Legume) contient le nom d'un fruit (resp. légume) et la durée pendant laquelle il est encore consommable. Les fruits et légumes sont représentés par les structures :

```
struct FruitSt{
        char nomF[Taille] ;             // Le nom du fruit
        int nbJFrais ;          // Le nombre de jours où le fruit est encore frais
};
typedef struct fruitSt Fruit;

struct LegumeSt{
        char nomL[Taille] ;     // Le nom du légume
        int nbJFrais ;          // Le nombre de jours où le legume est encore frais
};
typedef struct legumeSt Legume;
```

Le réfrigérateur contient des fruits et des légumes. Il est représenté par la structure suivante :

```
#define Taille 20
struct frigoSt{
    Fruit tabF[Taille];             // Le tableau de fruits
    Legume tabL[Taille];            // Le tableau de légumes
    int nbF ;                   // Le nombre de fruits présents dans le frigo.
    int nbL ;                   // Le nombre de légumes présents dans le frigo.
};
typedef struct frigoSt Frigo;
```

Q 4.1) Écrivez une procédure *void inventaireFrigo(Frigo f)* qui affiche **tous les fruits et légumes** qui sont dans le *Frigo f* avec le formatage suivant :

Par exemple, avec un frigo. contenant une banane, une pomme et un navet, on aurait l'affichage :

Il y a *2 fruit et 1 légume dont :* *- Un(e) Banane : 3 jours*
- Un(e) pomme : 7 jours
- Un(e) navet : 10 jours

```
void inventaireFrigo(Frigo f) {
```

}

Q 4.2) Écrivez une fonction int *nbrFruits(Frigo f, char nomDuFruit[Taille])* qui prend comme argument un *Frigo f,* une chaîne de caractères *nomDuFruit* et qui renvoie le nombre de fruits qui portent le nom de *nomDuFruit*.

int nbrFruits(Frigo f, char nomDuFruit[Taille]){

}

Q 4.3) Écrivez une fonction *Frigo AjouteFruit(Frigo f, char nomDuFruit[Taille], int nbJFrais)* qui prend comme argument un *Frigo f,* une chaîne de caractères *nomDuFruit,* un entier nbJFrais spécifiant la durée pendant laquelle le fruit est encore consommable. Elle ajoute le fruit si le frigo. n'est pas rempli, sinon, elle ne fait rien. Enfin, elle renvoie le frigo.

```
Frigo AjouteFruit (Frigo f, char nomDuFruit[Taille], int nbJFrais ){




}
```

Q 4.4) Écrivez une fonction *Frigo SupprimeLegume(Frigo f, int i)* qui prend comme argument un *Frigo f* et qui supprime le légume qui se trouve à l'indice *i* (donné en paramètre de la fonction) dans le tableau de légume et retourne le frigo. mis à jour. N.B. : pensez à tester la valeur de *i* !

```
Frigo SupprimeLegume(Frigo f, int i)  {




}
```

Q 4.5) Tous les soirs, un script exécute la fonction Frigo *MaJLegumes(Frigo f)* qui prend comme argument un *Frigo f.* Celui-ci décrémente d'un jour le *nbJoursFrais* de tous les légumes, supprime les légumes devenus non consommables (i.e. les légumes dont le *nbJoursFrais* est inférieur ou égal à 0) et renvoie le frigo. mis à jour. Écrivez la fonction *MaJLegumes*.

```
Frigo MaJLegumes(Frigo f)  {




}
```

Q 4.6) Ecrire un programme principal **int main()** qui permet d'appeler toutes les fonctions et procédures précédentes, en n'oubliant pas de déclarer les variables nécessaires.

```
int main(){
// déclaration et initialisation d'un frigo
Frigo f ;
f.tabF[0].nomFruit = "Banane"  ;  f.tabF[0].nbJoursFrais  =  3;
f.tabF[1].nomFruit = "Pomme" ;   f.tabF[1].nbJoursFrais  =  7 ;
f.tabL[0].nomLegume = "Navet" ; f.tabL[0].nbJoursFrais  =  10 ;
f.nbF=2 ;
f.nbL=1 ;
// Afficher l'état du frigo



// Ajoutez un fruit Pomme avec une durée de consommation de 20 jours



// Affichez le nombre de Pommes



// Mettez à jour les légumes du frigo



}
```

Exercice 5 – Les pointeurs *(3 points)*

Corrigez directement sur la feuille les fonctions suivantes **sans ajouter de lignes de code ! Les seuls caractères à ajouter judicieusement sont : * et &.** NB : le nombre de caractères à ajouter est de 12.

Le programme doit afficher la somme des nombres de 0 à N élevés à la puissance P. Par exemple, avec N=3 et P=2, le résultat est 14 (soit $1^2 + 2^2 + 3^2$).

```
void NpuissP( int A , int P , int res2 ){
        int i ;
        res2 = 1 ;
        for( i = 0 ; i < P ; i ++)
                res2 = res2 * A ;
}

void sommeNbrPuissN( int N , int P , int res1 ) {
        int i ;
        int res2 ;
        res1 = 0 ;
        for( i = 0 ; i ≤ N ; i ++ ){
                NpuissP( i , P , res2 ) ;
                res1 = res1 + res2 ;
        }
}

int main(){
        int N ;
        int P ;
        int res ;
        scanf(''%d'', N ) ;
        scanf(''%d'', P ) ;
        sommeNbrPuissN( N , P , res ) ;
        printf(''Le résultat est %d'', res ) ;
        return 0 ;
}
```