


Institut Supérieur d'Informatique et de Mathématiques 	Année Universitaire : 2022-2023
	<p style="text-align: center;">Examen</p> Matière : Atelier de programmation 1 Filières : L1 Info Enseignant : Sakka Rouis Taoufik

Exercice 1 : (8 Points)

On souhaite gérer les produits d'un magasin en utilisant un tableau de structures permettant l'enregistrement des informations concernant les produits. Pour cela on considère les champs suivants :

Une **Date** est caractérisée par :

- ✓ mois (entier)
- ✓ année (entier)

Un **Produit** est caractérisé par :

- ✓ code (code de l'adhérent : entier)
- ✓ nom (chaîne de 15 caractères)
- ✓ date_production : (structure de type Date)
- ✓ date_expiration: (structure de type Date)
- ✓ prix (réel)
- ✓ quantité (entier)

On vous demande d'écrire:

- 1- Définir une structure nommée **Date**,
- 2- Définir une structure nommée **Produit**,
- 3- Une fonction qui permet de comparer deux dates d1 et d2, et retourne (-1 si d1 avant d2, 0 si d1 et d2 sont identiques , 1 si d1 après d2)
- 4- Une fonction qui permet de saisir les informations de **N** produits dans un tableau **TP**.
NB : la date d'expiration doit être supérieure à la date de production.
- 5- Une fonction qui permet d'afficher les informations des produits du tableau **TP**.
- 6- Une fonction qui permet de trier le tableau TP en ordre croissant suivant le champ date de production.
- 7- Une fonction qui permet d'afficher la liste des produits expirés par rapport à une date donnée.
- 8- La fonction main qui permet de lire la taille de **N** ($3 \leq N \leq 100$) et tester les fonctions ci-dessus.

Exercice 2 : (6 Points)

Un entier positif est dit méchant dont les facteurs premiers sont **uniquement** 2, 3 ou 5.

- **Exemples de nombres méchants** : 150, 12, 4, 243, 90 : car par exemple ($90=2*3*3*5$)
- **Exemples de nombres non méchants** : 14, 19, 33 : car par exemple ($33=3*11$)

On vous demande d'écrire la (ou les) fonction(s) nécessaire(s) permettant de vérifier si un entier positif est méchant ou non.

Exercice 3 : (6 Points)

La version classique de tri par sélection (donné en annexe) utilise deux boucles « **for** » imbriquées.

Le but de cet exercice est d'écrire deux versions récursives pour cette méthode. On vous demande d'écrire :

- 1- En utilisant **la récursivité non terminale**, proposer une deuxième implémentation pour la fonction «**POSMIN** » qui retourne la position de la valeur minimale.
- 2- En utilisant **la récursivité terminale**, proposer une troisième implémentation pour la fonction «**POSMIN** » qui retourne la position de la valeur minimale. **NB.** Vous pouvez modifier la liste des paramètres.
- 3- En utilisant la fonction de la question 1, écrire une version récursive du tri par sélection « **tri_selection_rec1** » qui n'utilise aucune structure de boucle.
- 4- En utilisant la fonction de la question 2, écrire une version récursive du tri par sélection « **tri_selection_rec2** » qui n'utilise aucune structure de boucle.

Annexe :

```
int POSMIN (int T[], int g, int N) {
    int j, posmin=g;
    for(j=g+1; j<N; j++)
        if(T[j]<T[posmin])
            posmin=j;

    return posmin;
}

void TRI_SELECTION (int T[], int N){
    int i, pos, aux;
    for (i=0; i<N-1;i++) {
        pos= POSMIN(T,i,N);
        if (pos != i) {
            aux=T[i];
            T[i]=T[pos];
            T[pos]=aux;
        }
    }
}
```