Application - Récursivité

Exo 1 : Ecrire une fonction récursive qui affiche les différents termes de la suite suivante :

$$U_n = \begin{cases} 5 & Si\ n = 0 \\ \sqrt{2 + U_{n-1}} & si\ n \geq 1 \end{cases}$$

Ecrire le programme principal permettant de tester la fonction précédente.

Exo 2 : Dans cet exercice, il est recommandé d'utiliser la récursivité dès que possible.

On se propose d'écrire une application qui permet de mettre à jour des factures d'un magasin qui n'ont pas encore été payées. Les définitions suivantes seront utilisées :

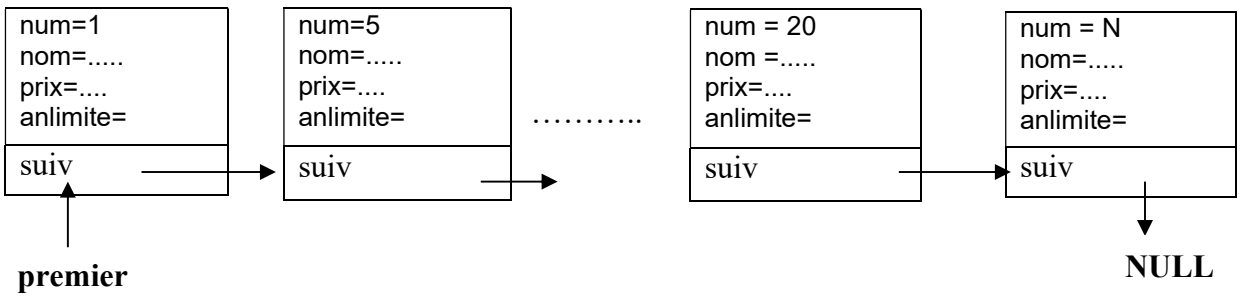
```
typedef struct {
        int num ;               // numéro de la facture
        char nom[20];           // le nom du client
        float prix ;            // le prix à payer
        int anlimite ;          // année limite de paiement
} Facture ;

typedef struct noeud  {
        Facture  info ;         // données de la facture
        struct noeud *suiv;     // adresse de l'élément suivant
} T_NOEUD;

const int anCourant = 2024 ;    // année en cours (à déclarer comme variable globale)
```

Dans cette liste les éléments sont reliés par ordre croissant de numéros.

Exemple de représentation de cette liste de factures



Questions :

1. Ecrire une fonction de prototype « **Facture saisir_Facture(void)** » permettant à l'utilisateur de saisir les attributs d'une facture.
2. Ecrire une fonction de prototype « **void affiche_Facture(Facture f)** » permettant d'afficher les attributs d'une facture.

3. Ecrire une fonction de prototype T_NOEUD * creer_noeud(void) qui permet de créer un maillon.

4. Ecrire une fonction qui permet d'insérer, dans le bon ordre suivant le numéro, dans liste T un maillon N précédemment créé. Cette fonction renvoie 1 ou -1 suivant si tout s'est bien passé ou non. Le prototype de la fonction est le suivant :

   int ins_noeud(T_NOEUD **T, T_NOEUD *N).

   En cas de problème avec le prototype précédent dû à la manipulation de pointeurs de pointeurs, vous pouvez aussi utiliser un prototype qui retourne la tête de la liste T_NOEUD * ins_noeud(T_NOEUD *T, T_NOEUD *N).

5. Faire une fonction qui permet d'afficher un maillon dont l'adresse est passée en paramètre. Le prototype sera le suivant : void aff_noeud(T_NOEUD *N). Vous ferez un affichage sous forme de liste.

6. Faire une fonction qui permet d'afficher tous les maillons de la liste et qui renvoie le nombre d'éléments. Le prototype de la fonction sera le suivant : int aff_liste(T_NOEUD *T). Un affichage post ordre est recommandé.

7. Ecrire une fonction de prototype « **int numero(**T_NOEUD *T**)** » qui retourne le plus petit numéro d'une facture dont l'année limite est supérieure ou égale à l'année en cours (le plus petit numéro d'une facture en règle). Cette fonction doit retourner 0 si toutes les factures de la liste ont une année limite de paiement strictement inférieure à l'année en cours.
8. Ecrire une fonction de prototype **« void supprimer(T_NOEUD **T, int val) »** qui supprime de la liste la facture de numéro val (cette facture vient d'être payée). Dans le cas où val ne correspond à aucun numéro d'une facture de la liste, la fonction affiche le message « cette facture n'existe pas ! ».
9. Ecrire une fonction de prototype « **void penalite (T_NOEUD *T) »** qui permet d'augmenter de 10% par an de retard les prix de toutes les factures de la liste dont l'année limite de paiement est strictement inférieure à l'année en cours.

Exemple :

Soit la facture ayant l'année limite de paiement 2022 et le prix=100 euros. Si l'année en cours est 2024, le nouveau prix deviendra 121 euros (de 2022 à 2023 on augmente le prix de 10% soit 110, de 2023 à 2024 (l'année en cours), on augmente 110 euros de 10% soit 121 euros).

10. Ecrire le programme principal qui permet de mettre en œuvre toutes les fonctionnalités. Il utilisera une fonction menu() qui proposera les options suivantes :
    (0) quitter le programme,
    (1) ajout d'une nouvelle facture,
    (2) afficher toutes les factures,
    (3) numéro de la première facture en règle,
    (4) pénalité,
    (5) supprimer une facture