DS n° 2 - Informatique II

Calculatrice et documents non autorisés

> *Une fonction / procédure peut être écrite en plusieures fonction si besoin.*
> *Si vous ne parvenez pas à répondre à une question, vous pouvez passer à la suivante en supposant que la réponse précédente a été fournie correctement).*
> *Les fonctions déjà écrites peuvent être ré-utilisées dans les questions suivantes.*
> *On ne vous demande pas d'écrire la fonction principale du programme.*
> **Vos programmes doivent être rédigés en C !**
> **Pretez attention à la ROBUSTESSE de vos programmes !!!**

Gestion de la masse salariale d'une entreprise

Le but de cet exercice est d'implémenter un programme permettant à une PME de gérer sa masse salariale.
Chaque employé est caractérisé par les informations suivantes :
— Nom
— Numéro d'employé (entier qui devra être toujours positif)
— Numéro de département (entier entre 1 et 95 inclus)
— Salaire mensuel brut
qui seront traduites en C par la structure `Employe` suivante :

```
typedef struct _emplo_{
    char*        nom;
    unsigned int num;
    int          dep;
    float        salaire;
} Employe;
```

1. Écrire la fonction constructeur d'un `Employe` (toutes les données seront saisies par l'utilisateur).
2. Écrire une fonction/procédure `creationEntreprise(...)` qui va demander à l'utilisateur le nombre d'employés de l'entreprise et va créer et remplir un tableau d'employés. Ce tableau ainsi que toutes les informations utiles permettant de l'exploiter doivent pouvoir être récupérées par le reste du programme.
3. Écrire une fonction `SalaireDepartement(...)` qui prend, au minimum, en paramètres un tableau d'employés, sa taille et un numéro de département. La fonction doit retourner la somme des salaires des employés de ce département. Cette fonction doit être **récursive terminale obligatoirement**.
4. Quelle est la complexité de la fonction précédente ? Argumenter.
5. Chaque fin d'année, l'entreprise augmente de 2.5% les salaires des employés dont le salaire brut est inférieur à 2500 euros. Écrire une fonction qui permet de mettre en place l'augmentation des employés avec les salaires les plus bas. Cette fonction va également retourner le nombre d'employés concernés par cette mesure. Cette fonction/procédure doit être **récursive non-terminale obligatoirement**.
6. On suppose que le tableau d'employés a été rangé par ordre croissant de numéro d'employé. Écrire une fonction `RechercheEmploye(...)` qui prendra en paramètre le tableau des tous les employés et un numéro d'employé que l'on recherche.

   Cette fonction va retourner **l'adresse** de l'employé dans le tableau dont le numéro correspond au numéro recherché. S'il n'existe pas, elle retournera **NULL**. *Attention* : cette fonction doit avoir une **complexité logarithmique**.

En plus de ses activités salariales "classiques", un employé peut s'investir dans des activités annexes au sein de l'entreprise comme la participation au Comité d'Entreprise. La participation à ces activités peut être associée à des primes. De plus, un même employé peut participer à plusieurs de ces activités.
Une activité est donc caractérisée par :
— Le nombre d'employés maximum pouvant participer à cette activité annexe (à définir dans le programme)
— Le tableau d'employés impliqués dans cette activité.

— la prime mensuelle associée à la participation à cette activité.
Ce qui se traduira en C par

```
typedef struct{
    int taille;
    Employe **tab;
    float prime;
} Activite;
```

7. Que va contenir le tableau `tab` de la structure `Activite` ? Quelle est la raison pratique de ce choix ?
8. Écrire la fonction constructeur d'une `Activite`. Cette fonction :
   — Demanderait à l'utilisateur le nombre de personnes impliquées et créerait le tableau en fonction.
   — Demanderait à l'utilisateur de saisir la prime
   — Demanderait à l'utilisateur le numéro de chaque employé impliqué et s'en servirait pour remplir le tableau (on pourra utiliser une des fonctions déjà écrites).
9. Écrire une fonction `participation(Activite a, int num)` qui prend en paramètre une activité et un entier correspondant à un numéro d'employé et qui retournera 1 si cet employé participe à l'activité et 0 sinon.
   Bonus : Écrire cette fonction de manière récursive (vous pouvez modifier la liste des paramètres et/ou la découper en plusieurs fonctions).
10. Quelle est la complexité de cette fonction ? Peut-on obtenir la même complexité que celle imposée en question 6 ? Argumenter.
11. Implémenter une fonction `majSalaire(...)` qui prend en paramètre :
    — Le tableau réunissant tous les employés.
    — Sa taille.
    — Le tableau réunissant toutes les Activités annexes de l'entreprise.
    — le nombre d'activité annexe de l'entreprise.
    Cette fonction va mettre à jour le salaire de tous les employés en ajoutant les potentielles primes de leur participations aux activités annexes (on rappelle qu'un employé peut participer à aucune, une ou plusieurs activités). Cette fonction peut être découpée en plusieurs fonctions.
12. De quoi dépend la complexité de la fonction précédente ? Quel est son ordre ? Argumenter.

Annexe : quelques fonctions utiles

string.h

strlen

Prototype : `size_t strlen(char *s);`
Description : La fonction `strlen` calcule la longueur d'une chaîne de caractères, sans compter le caractère nul '\0'.
Paramètre :
— `s` : Pointeur vers la chaîne dont la longueur est calculée.
Retour : Le nombre de caractères dans la chaîne pointée par `s`.

strcpy

Prototype : `char *strcpy(char *dest, char *src);`
Description : La fonction `strcpy` copie la chaîne pointée par `src`, y compris le caractère nul '\0', dans le tableau pointé par `dest`.
Paramètres :
— `dest` : Pointeur vers le tableau de destination.
— `src` : Pointeur vers la chaîne de caractères à copier.
Retour : Un pointeur vers la chaîne de destination `dest`.

strcmp

Prototype : `int strcmp(char *s1, char *s2);`
Description : La fonction `strcmp` compare les deux chaînes `s1` et `s2`.
Paramètres :

— `s1` : Première chaîne à comparer.
— `s2` : Seconde chaîne à comparer.

Retour : 0 si `s1` et `s2` sont égales, une valeur négative si `s1` est inférieure à `s2`, et une valeur positive si `s1` est supérieure à `s2`.

strcat

Prototype : `char *strcat(char *dest, char *src);`
Description : La fonction `strcat` ajoute la chaîne `src` à la fin de `dest`. Elle suppose que le tableau `dest` a assez d'espace pour contenir les deux chaînes.
Paramètres :
— `dest` : Pointeur vers la chaîne de destination.
— `src` : Chaîne à ajouter à `dest`.
Retour : Un pointeur vers la chaîne de destination `dest`.

---

stdlib.h

malloc

Prototype : `void *malloc(size_t size);`
Description : La fonction `malloc` alloue un bloc de mémoire de taille spécifiée et renvoie un pointeur vers le début du bloc.
Paramètre :
— `size` : La taille en octets de la mémoire à allouer.
Retour : Un pointeur vers le bloc de mémoire alloué ou NULL si l'allocation échoue.

calloc

Prototype : `void *calloc(size_t nmemb, size_t size);`
Description : La fonction `calloc` alloue un bloc de mémoire pour un tableau de `nmemb` éléments de `size` octets chacun. La mémoire est initialisée à zéro.
Paramètres :
— `nmemb` : Nombre d'éléments.
— `size` : Taille de chaque élément.
Retour : Un pointeur vers le bloc de mémoire alloué ou NULL si l'allocation échoue.

atof

Prototype : `double atof(const char *str);`
Description : La fonction `atof` convertit la chaîne de caractères `str` en un nombre à virgule flottante de type double.
Paramètre :
— `str` : Chaîne représentant le nombre à convertir.
Retour : La valeur convertie en double. Si `str` ne contient pas un nombre valide, zéro est retourné.