ESILV LÉONARD DE VINCI | ÉCOLE D'INGÉNIEURS PARIS-LA DÉFENSE

Langage C

A3 - TD N

Réalisation d'une application de gestion d'un annuaire de fonctionnaires

PICARD Nicolas - PIBERNE Thomas

Décembre 2017

Explication du type table

Tout d'abord, nous avons réfléchi à la logique derrière la table de fonctionnaires demandée. Nous avons réalisé un schéma afin de bien distinguer chaque type, puisque nous avons créé une structure pour chaque besoin : une structure table, une structure vecteur, et une structure fonctionnaire.

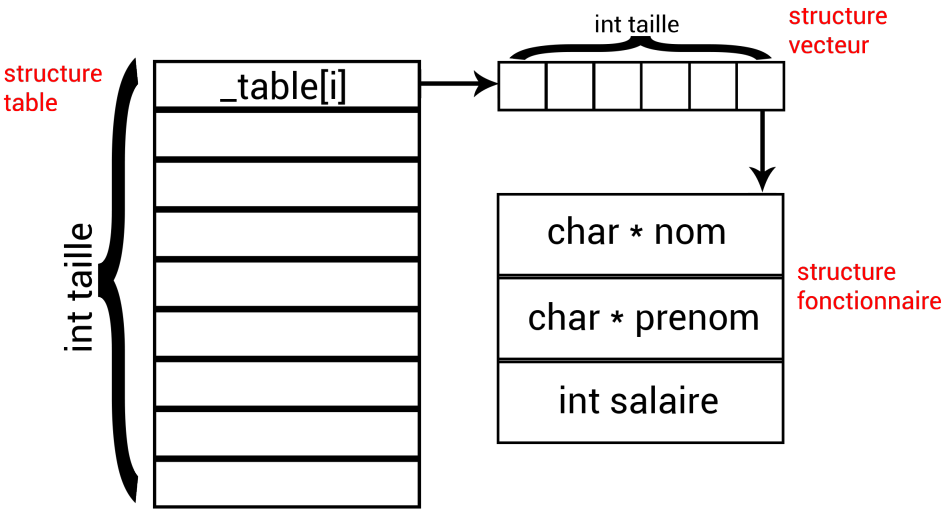

Figure 1: Organisation des structures

Nous avons donc commencé par affecter une ***taille*** à cette structure ***table***. C'est le nombre de vecteurs dont elle disposera. Nous lui avons par la suite ajouté un contenu de type ***vecteur*** (vecteur * _table).
Chaque vecteur dispose d'une ***taille*** (ie. un nombre de fonctionnaires), et d'un contenu de ***fonctionnaires*** (fonctionnaire * vec_fonctionnaires). En effet, ce vecteur sera constamment sollicité puisqu'un nombre variable de fonctionnaires se situera à l'intérieur.
Chaque fonctionnaire dispose lui, d'un ***nom*** (char * nom), d'un ***prénom*** (char * prenom), et d'un ***salaire*** (int salaire).

Descriptif des fonctions

1. Index : La fonction index est codée de telle façon qu'elle reçoit des entrées utilisateurs (*scanf*) pour le nom et prénom du fonctionnaire. Les fonctions *formatNom* et *formatPrenom* s'assurent de formater correctement les entrées utilisateurs afin de rendre celles-ci plus simples (peu importe les majuscules ou non entrées). Une fonction index réalise ensuite le calcul de l'index (sur la concaténation des 4 premières lettres du nom et des deux premières lettres du prénom) à l'aide de la formule suivante :

$$\boxed{index = \left(\sum_{i=0}^{5} ascii(c_i).B^i\right) \pmod N}$$

2. Ajouter : Tout comme la fonction *index*, *ajouter* prend des entrées utilisateurs, formate le nom et prénom du fonctionnaire à ajouter et mémorise le salaire entré. La méthode *addToTable* s'occupe de trouver le vecteur où placer le fonctionnaire, grâce au calcul de son index, puis d'ajouter (ou non) le fonctionnaire avec *ajouterFonctionnaire*, qui va réaliser une vérification de l'existence de ce fonctionnaire. Un nouvel espace mémoire de taille *n+1* est alors alloué pour y placer ce nouveau fonctionnaire s'il n'existe pas. Enfin, on augmente la taille du vecteur et on affecte ce nouvel espace mémoire au contenu du vecteur, puis on le trie à l'aide de *triAlpha*.

3. Charger : Cette fonction attend une entrée utilisateur qui correspond au nombre de fonctionnaires à charger. Si le nombre entré est bien inférieur au nombre maximum de fonctionnaires du fichier Chicago.txt (et que le fichier existe évidemment), alors l'ajout des n premiers fonctionnaires du fichier se fait, en récupérant pour chaque ligne le nom, prénom et salaire du fonctionnaire puis en appelant la fonction *addToTable* réalisée auparavant.

4. Afficher Salaire : On récupère tout d'abord le nom et le prénom du fonctionnaire recherché et on les formate. On calcule ensuite son index pour trouver le vecteur correspondant. On crée un fonctionnaire (temporaire) qui servira à la comparaison lors de la vérification de son existence au sein du vecteur, et enfin si celui-ci existe on parcourt le vecteur jusqu'à trouver le fonctionnaire correspondant, puis on affiche le salaire correspondant.

5. Afficher entre : L'utilisateur entre d'abord deux index, l'un de début et l'autre de fin. Si ces index sont bien dans la table, alors pour chaque vecteur entre ces index on affiche son contenu, c'est à dire *NOM Prénom Salaire* de chaque fonctionnaire, ou *Pas de fonctionnaire* sinon.

6. Nombre de conflits : Cette fonction parcourt toute la table et regarde pour chaque vecteur si sa taille est supérieure à 1, c'est à dire si son contenu comporte deux fonctionnaires ou plus. Si c'est le cas, alors le compteur de conflits augmente de 1.

7. Taille moyenne des conflits: Lorsqu'il y a conflit, (*cf* fonction Nombre de conflits), alors on compte le nombre de fonctionnaires du vecteur correspondant et on l'ajoute à la somme. Lorsque la table a été entièrement parcourue, et si $nbConflits \neq 0$ (pas de division par 0) alors on affiche la somme du nombre de fonctionnaires divisée par le nombre de conflits.

8. Supprimer : *supprimer* prend des entrées utilisateurs, formate le nom et prénom du fonctionnaire à supprimer. La méthode *deleteFromTable* s'occupe de trouver le vecteur où placer le fonctionnaire, grâce au calcul de son index, puis d'appeler *supprimerFonctionnaire*, qui réalise une vérification de l'existence de ce fonctionnaire. Si il existe, on supprime le fonctionnaire avec *free*, et on decale les autres fonctionnaires qui le suivaient. Enfin, on diminue la taille du vecteur et on le trie à l'aide de *triAlpha*.
Cas particulier : vecteur de taille 1 : on *free* directement le contenu du vecteur et on affecte sa taille à 0.

9. Supprimer entre : L'utilisateur entre d'abord deux index, l'un de début et l'autre de fin. Si ces index sont bien dans la table, alors pour chaque vecteur dont le contenu n'est pas vide entre ces index on *free* son contenu et on lui affecte une taille de 0.

10. Quitter : Quitte le programme avec $EXIT\_SUCCESS \Leftrightarrow return\ 0$, qui indique que le programme a pris fin et que tout s'est bien passé.