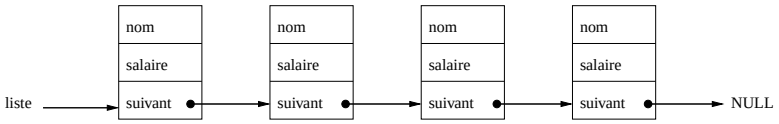
Programmation système et réseaux

Corrigé du TP 7: Listes chaînées

Arnaud Giersch Guillaume Latu

1. On souhaite réaliser une gestion de fiches concernant les employés d'une petite structure. La structure de données que vous allez employer dans votre programme pour manipuler ces fiches est une *liste chaînée*. Dans ce type de liste, chaque élément possède un pointeur sur l'élément *suivant* de la liste. Le dernier élément possède un *suivant* égal à NULL.

```
#define TAILLE_CHAINE 32
struct Employe {
    char nom[TAILLE_CHAINE];
    int salaire ;
    struct Employe *suivant;
};
```



Implémentez les différentes fonctions dont les prototypes vous sont présentés ci-dessous :

```
void affichageEmploye (const struct Employe *e);
void affichageEmployes (const struct Employe * liste );
struct Employe *ajouteEmploye (struct Employe *prec, char *nom, int salaire );
struct Employe *soustraitEmploye ( struct Employe * liste , struct Employe *cible );
void supprimeEmployes (struct Employe * liste );
struct Employe * triSalaire ( struct Employe *lentree );
struct Employe *triNom (struct Employe *lentree );
```

- La fonction ajouteEmploye ajoute la fiche du nouvel employé *avant* celui passé en paramètre et renvoie le pointeur sur la fiche du nouvel employé.
- La fonction soustraitEmploye parcourt la liste chaînée à la recherche de la fiche employé passée en paramètre. Elle renvoie la liste chaînée, dans laquelle on a soustrait la fiche indiquée.
- La fonction supprimeEmployes libère la mémoire allouée pour la liste.
- Le but des deux fonctions de tri est de classer les employés selon des critères différents. Pour ces fonctions, vous utiliserez l'algorithme de tri qui suit. Celui-ci manipule deux listes (*lentrée* et *lsortie*) :
  - Au départ, la liste *lsortie* est vide.
  - Tant que *lentree* n'est pas vide,
    - rechercher l'élément maximum dans la liste *lentrée*, et le soustraire de cette liste ;
    - ajouter cet élément en tête de la nouvelle liste *lsortie*.

Correction :

```
#include < stdlib .h>
#include < string .h>
#include < stdio .h>

void affichageEmploye (const struct Employe *e)
{
```

```
    printf ("Salaire : %6d, ", e->salaire);
    printf ("Nom de l'employé : %s\n", e->nom);
}

void affichageEmployes (const struct Employe *liste)
{
    while (liste != NULL) {
        affichageEmploye (liste);
        liste = liste->suivant;
    }
}

struct Employe *ajouteEmploye (struct Employe *prec, char *nom, int salaire)
{
    struct Employe *nouv = malloc (sizeof (struct Employe));
    if (nouv == NULL) {
        perror ("malloc()");
        exit (EXIT_FAILURE);
    }
    strncpy (nouv->nom, nom, TAILLE_CHAINE-1);
    nouv->nom[TAILLE_CHAINE-1] = '\0';
    nouv->salaire = salaire;
    nouv->suivant = prec;
    return nouv;
}

struct Employe *soustraitEmploye (struct Employe *liste, struct Employe *cible)
{
    if (cible == liste) {
        liste = cible->suivant;
    } else {
        struct Employe *e = liste;
        while (e->suivant != cible)
            e = e->suivant;
        e->suivant = cible->suivant;
    }
    cible->suivant = NULL;
    return liste;
}

void supprimeEmployes (struct Employe *liste)
{
    if (liste != NULL) {
        supprimeEmployes (liste->suivant);
        free (liste);
    }
}

struct Employe *tri (struct Employe *liste,
                     int (*cmp)(struct Employe *, struct Employe *))
{
    struct Employe *elt, *max, *newliste = NULL;

    while (liste != NULL) {
        max = liste;
        for (elt = liste->suivant; elt != NULL; elt = elt->suivant) {
            if (cmp (max, elt) < 1)
                max = elt;
```

```
            }
            liste = soustraitEmploye ( liste , max);
            max->suivant = newliste;
            newliste = max;
        }
        return newliste;
}

int cmpNom (struct Employe *e1, struct Employe *e2)
{
    return strcmp ( e1->nom, e2->nom);
}

struct Employe *triNom (struct Employe * liste )
{
    return tri ( liste , cmpNom);
}

int cmpSalaire ( struct Employe *e1, struct Employe *e2)
{
    return ( e1->salaire < e2->salaire )? -1: ( e1->salaire != e2->salaire );
}

struct Employe * triSalaire ( struct Employe * liste )
{
    return tri ( liste , cmpSalaire );
}

int main (void)
{
    struct Employe *liste = NULL;

    liste = ajouteEmploye ( liste , "Bob", 1600);
    liste = ajouteEmploye ( liste , "Albert", 2000);
    liste = ajouteEmploye ( liste , "Denise", 1900);
    liste = ajouteEmploye ( liste , "Florence", 1800);
    liste = ajouteEmploye ( liste , "Céline", 1700);
    liste = ajouteEmploye ( liste , "Edmond", 1500);

    printf ( "Liste:\n");
    affichageEmployes ( liste );

    printf ( "Liste par nom:\n");
    liste = triNom ( liste );
    affichageEmployes ( liste );

    printf ( "Liste par salaire:\n");
    liste = triSalaire ( liste );
    affichageEmployes ( liste );

    supprimeEmployes ( liste );

    return EXIT_SUCCESS;
}
```

2. On souhaite disposer de listes rassemblant les différentes catégories de salariés : secrétaires, assistants de directions, etc.
   – Mettez au point la structure de données Categorie (sans modifier la structure Employe).

– Implémentez les fonctions de mise à jour d'une structure Categorie : insertion, suppression, recherche, changement de catégorie.
– Écrivez une fonction qui, à partir d'une structure Categorie, crée une *nouvelle* liste contenant tous les employés.
– Construisez une fonction qui affiche le contenu d'une structure Categorie.