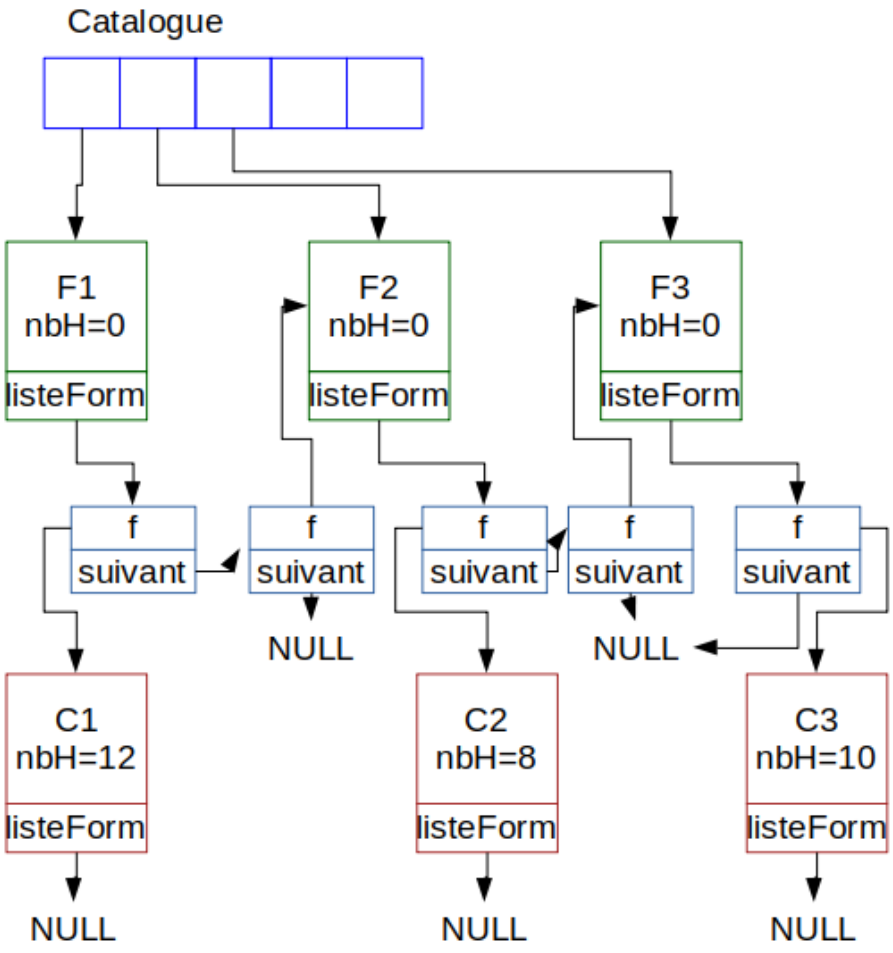
Structures de données
TD5 Structures non linéaires

1 Exercice 1 Formations



```c
typedef struct form Formation; // indique au compilateur que ce type existe

typedef struct element{
    Formation* f;
    struct element* suivant;
} Element;

struct form{
  char* nom;
  int nbHeures;

  Element* listeFormations
};

typedef struct catalogue {
    unsigned nb_formations; // unsigned et unsigned int c'est le même type
    unsigned nb_max_formations; // taille du tableau
    Formation** formations_tab; // malloc donc non contigue
} Catalog;

void affichage_formation(Formation* f)
{
    if(f->nbHeures == 0) {
        printf("Formation %s : ", f->nom);
        Element* cur = f->listeFormations;
        while(cur!=NULL) {
            printf("%s ", cur->f->nom);
            cur = cur->suivant;
        }
    } else {
        printf("Cours %s (%d heures)", f->nom, f->nbHeures);
    }
    printf("\n");
}

void affichage_catalogue(Catalog* c)
{
    for(int i=0;i<c->nb_formations;i++) {
        affichage_formation(c->formations_tab[i]);
    }
}

int nbheures_formation(Formation* f)
{
```

```
    int nbH = f->nbHeures;
    Element* cur = f->listeFormations;
    while(cur!=NULL) {
        nbH += nbheures_formation(cur->f);
        cur = cur->suivant;
    }
    return nbH;
}

void ajouter_formation_dans_formation(Formation* f, Formation* nouvelle)
{
    // rechercher sur nouvelle est deja incluse dans f (ou inversement)
    // on cherche a empecher la creation de cycle
    // si la recherche indique que les formations sont déjà liées alors
    // on annule l'ajout
}
```

2 Maison des associations

```
typedef struct membre{
    char* nom;
    structu membre* suiv;
} Membre;
typedef struct asso{
    char* nomAssoc;
    Membre* liste_membres;
} Association;
typedef struct maison{
    Association* assoc;
    struct maison* suiv;
} Maison;
Association* creerAsocciation(char* nomA) {
    Association* a = (Association*)malloc(sizeof(Association));
    a->nomAssoc = strdup(nomA); // necessite un free à la fin
    a->liste_membres = NULL;
    return a;
}
void ajouterPersonne(Association* a, char* nom)
{
    Membre* m = (Membre*)malloc(sizeof(Membre));
    m->nom = strdup(nom); // free a la fin !!
    m->suiv = a->liste_membres;
    a->liste_membres = m; // m devient la tete de liste
```

```
}
void supprimerPersonne(Association* a, char* nom)
{
    Membre* cur = a->liste_membres;
    // A -> B -> C
    // si vous supprimez B il faut que A pointe vers C
    Membre* prev = NULL;
    while(cur!=NULL && strcmp(nom, cur->nom) != 0) {
        prev = cur;
        cur = cur->suiv;
    }
    if(cur != NULL) {
        prev->suiv = cur->suiv;
        free(cur->nom); // car strdup
        free(cur);
    }
}
void ajouterAssociation(Maison** m, Association* a)
{
    Maison* new = creerMaisson();
    new->assoc = a;
    new->suiv = *m; // on pointe sur la tete actuelle
    *m = new; // on met a jour la tete
}
void enleverAssociation(Maison** m, Association* a)
{
    Maison* cur = *m;
    if(strcmp(a->nomAssoc, cur->assoc->nomAssoc)==0) // si cest le meme nom
    {
        *m = cur->suiv;
        free(cur);
    } else {
        Maison* prev = cur;
        cur = cur->suiv;
        while(cur!=NULL && strcmp(a->nomAssoc, cur->assoc->nomAssoc) != 0) {
            prev=cur;
            cur=cur->suiv;
        }
        if(cur != NULL) {
            prev->suiv=cur->suiv;
            free(cur);
        }
    }
}
```

On peut recevoir deux fois le courrier si on une personne physique est membre

- de deux associations liées
- si on a au moins 2 associations qui sont liées (A ¡=¿ B) alors on a un cycle et donc boucle infinie

Pour éviter les doublons on peut utiliser une table de hachage dont la clé est le nom de la personne et le contenu ça peut être un booleen où 0 = courrier pas envoyé et 1 = courrier déjà envoyé. ⇒ on utilise une table par courrier

Pour éviter la boucle infinie il faudrait "marquer" les éléments que l'on a visité ⇒ parcourir un graphe

On parcourt toutes les associations, le problème c'est si la maison des asso est très grande alors c'est très lent l'idée c'est de dire qu'on utilise plus la complexité du temps d'execution mais la compelxite de la consommation mémoire