Les listes chaînées : Un Essai didactique

Proposé par Pr. D. El Ghanami

Ecole Mohammadia d'Ingénieurs

Mai 2011

Des fonctions de manipulation d'une Liste

1. Définition du nœud :

```
typedef struct un_noeud{
        int x;
        struct un_noeud *s;
} nœud ;
```

Déclaration d'une liste ***L*** vide *:*

*nœud *L = NULL ;*

L	Signifie la liste (pointeur sur le 1er nœud s'il existe ou *NULL* sinon).
**L*	1er nœud de la liste s'il existe sinon cette utilisation est fausse.
**L.x et *L.s*	Entier et pointeur du 1er nœud (autre écriture *L→s*, ce dernier pointe sur le 2ème nœud)
**(*L.s).s*	ou *L→s→s* : Pointeur sur le 3ème nœud
L→	est équivalent à **L*. c-à-d qu'à la droite de → on ne peut avoir qu'un pointeur.

Cas d'une liste d'enregistrements (ex. des points de coordonnées *x* et *y*) alors accès aux variables d'un l'enregistrement: **L.p.x* (autrement *L→p.x*)

Si le passage de ***L*** à une fonction se fait par valeur (ex. ***f(L)*** **)**, alors les modifications de *L* seront fait sur la copie.

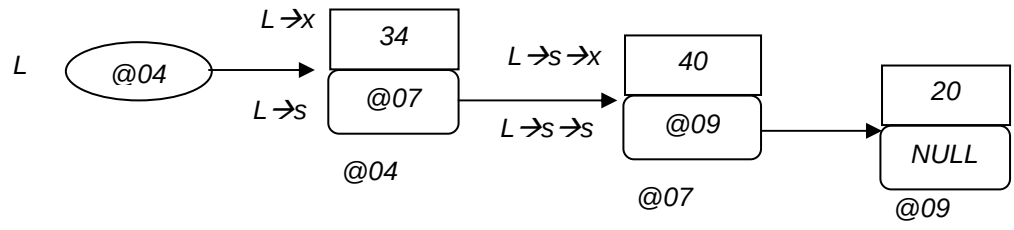
Dans le cas de modification de *L* au sein de ***f*** (cas : ajout, suppression d'un nœud au début) alors la nouvelle valeur de *L* doit être retournée. Attention au cas où la modification se fera sous condition *if*.

Autre manière est le passage par adresse de *L* (ex. *f(&L)*), alors le prototype de la fonction sera : *f(nœud **pL)*. La manipulation la plus simple au sein de *f* est la suivante :
*nœud *p ;*
*p=*pL ;*
La manipulation de la liste (accès aux nœuds se feront par le pointeur *p*).
Si modification de *L* (ex. suppression du nœud début) alors : **pL = p→s*

Si la liste *L* est déclarée variable globale, alors elle n'est ni passée en paramètre ni retournée et sa modification dans une fonction est valable partout.

L'allocation des nœuds ne doit pas se faire dans la PILE au risque d'avoir une rupture dans le chaînage de la liste mais dans le TAS grâce à l'instruction malloc.
La libération de la mémoire (instruction free) n'est pas importante dans le programme info-CPGE (rappel : intérêts de malloc et les pointeurs)

Soit la liste suivante :



Distance entre nœuds :
Soit *p1=@04*, *p2=@07*, *p3=@09* et *p*

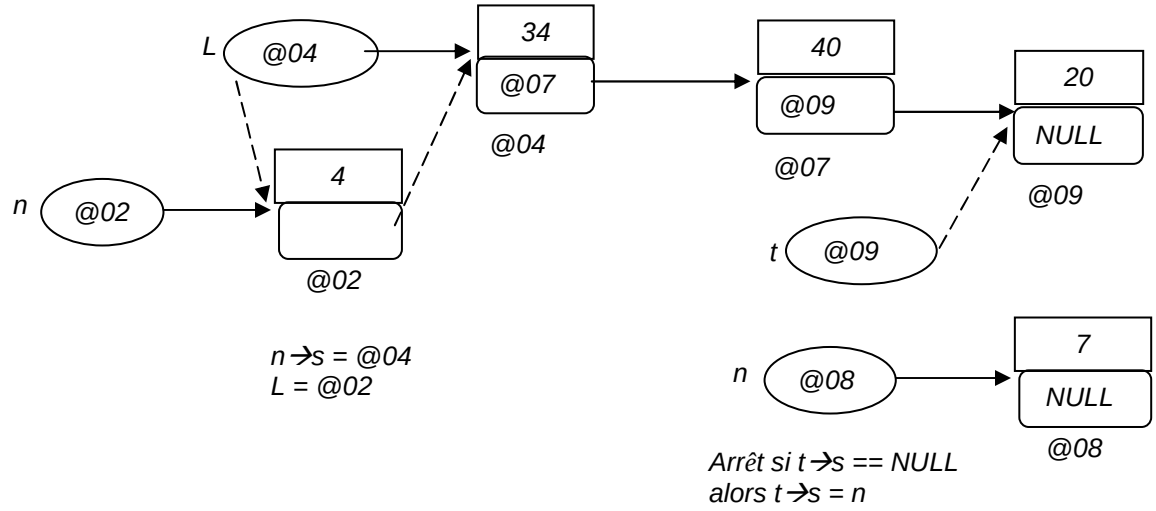
Pour que *p* indique le 1er nœud, il faut que : *p == p1* : distance *0* entre **p* et **p1* (même nœud)

p→s==p2 : distance *1* entre **p* et **p2* (deux nœuds voisins)
p→s→s==p3 : distance *2* entre **p* et **p3*

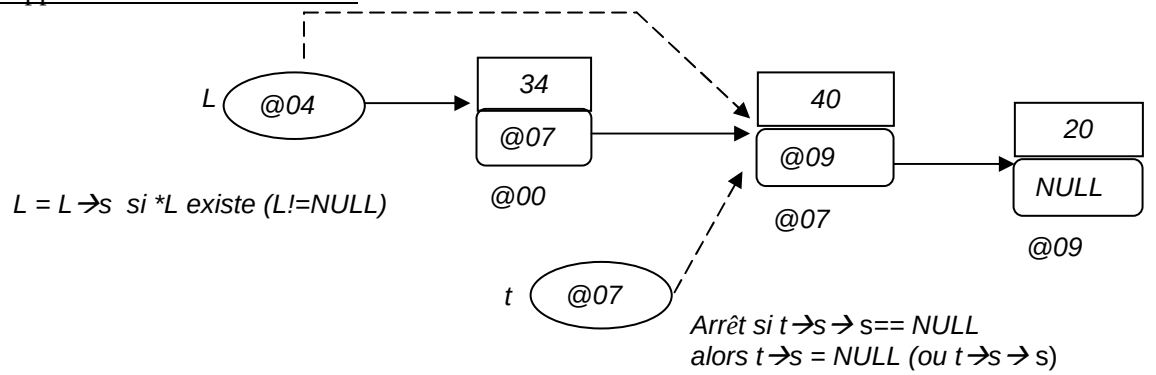
Exemples:

- Pour supprimer le dernier nœud, il faut que *p* pointe sur l'avant dernier nœud, alors la distance entre *p* et *NULL* est de *2* : *while(p→s→s !=NULL) p=p→s ;*
- Pour ajouter un nœud à la fin de la liste, il faut que *p* pointe sur le dernier nœud alors la distance entre *p* et *NULL* est de *1* : *while(p→s !=NULL) p=p→s ;*
- Pour traiter une liste, il faut la parcourir et pointer sur chaque nœud. A la fin du traitement le pointeur sera *NULL* : *while(p→s !=NULL){Traitement ; p=p→s ;}*
  Attention au cas particuliers : liste vide, liste avec un seul élément, ajout fin équivalent à ajout début.
- Le déplacement de *p (p=p→s)* peut se faire par rapport à une autre condition d'arrêt (pointeur *q* équivalent à *NULL*) suivant la même logique précédente.
- Le déplacement de *p* peut se faire d'une manière déterministe (un certain nombre de fois)

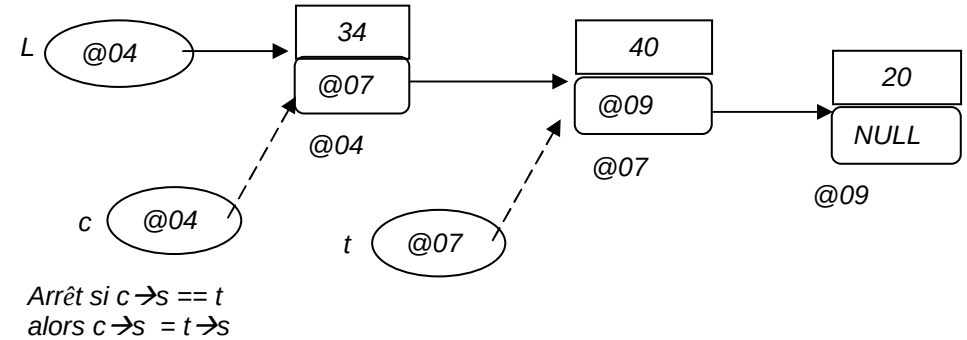
Ajout début et à la fin de la liste:



Supprimer au début et à la fin :



Supprimer une valeur :



noeud creerN(int i, nœud *p){* *noeud *n =(noeud*)malloc(sizeof(noeud));* *n→x = i;* *n→s = p;* *return n;* *}*	Création d'une case mémoire initialisée par les paramètres i et p et retour de son adresse.
noeud ajoutD(noeud *l, noeud* n){* *n→s = l;* *return n;* *}*	Vers2 (passage par @ de L) *void ajoutD(noeud **pl, noeud* n){* *noeud *p=*pl;* *n→s = p;* **pl=n;* *}*
noeud supprimeD(noeud *l){* *if(l != NULL) l = l→s;* *return l;* *}*	*void supprimeD(noeud **pl){* *noeud *p=*pl;* *if(p != NULL) *pl = p→s;* *}*
nœud sommetPile(noeud *l){* *return l;* *}*	Retour de l'@ du nœud de la tête de liste. L'avantage c'est le retour de plusieurs variables d'un nœud.
int nbNoeud(noeud l){* *int i = 0;* *while(l != NULL){* *i++; l = l→s;* *}* *return i;* *}*	
noeud seDeplacer(noeud* l, int i){* *int j;* *for(j = 0 ; j < i ; j++) l = l→s;* *return l;* *}* *void inverserListe(noeud *l){* *noeud *t1, *t2;* *int a, i, j, n = nbNoeud(l);* *for(i = 0; i < n/2; i++){* *t1 = seDeplacer(l, i);* *t2 = seDeplacer(l, n-1-i);* *a = t1→x;* *t1→x = t2→x;* *t2→x = a;* *}}*	l = l→s est le déplacement d'un nœud à l'autre i = 0 se déplacer 0 fois → le 1er nœud i = 1 se déplacer 1 fois → le 2ème nœud i = nbNoeud-1 → le dernier nœud i = nbNoeud → NULL donc : 0≤i<nbNoeud
noeud ajouterFin(noeud *l, noeud* n){* *noeud *c;* *if(l == NULL)* *return ajouterDeb(l, n);* *c = seDeplacer(l, nbNoeud(l) – 1);* *n→s = NULL ;* *c→s = n;* *return l;* *}*	Si liste est vide alors ajouter fin revient à ajouter au début (la nouvelle tête à retourner) sinon il faut déplacer *c* sur le dernier nœud et retourner quelque chose qui conserve *L*. *void ajouterFin(noeud **pl, noeud* n){* *noeud *c=*pl;* *n→s=NULL ;* *if(c == NULL) *pl=n ;* *else{ while(c→s!=NULL) c=c→s;* *c→s = n; }* *}*
noeud supprimerFin(noeud *l){*	Pour supprimer le dernier nœud, il faut mettre la valeur du pointeur de l'avant dernier nœud à NULL.

*noeud *c;* *if(l == NULL) return NULL;* *if(l→s == NULL)* *return supprimerD(l);* *c = seDeplacer(l, nbNoeud(l) – 2);* *c→s = NULL ;* *return l;* *}*	Il faut se déplacer jusqu'à ce nœud ou c = l ; while(c→s→s!=NULL) c=c→s; Cas particuliers : si liste vide : rien à supprimer Si la liste contient un seul nœud, ce nœud n'a pas de précédent et donc le faut le traiter comme supprimer début.
nœud sommetFile(noeud *l){* *noeud *c;* *if(l→s == NULL)* *return sommetPile(l);* *c = seDeplacer(l,nbNoeud(l) – 1);* *return c;* *}*	Déplacer le pointeur jusqu'au dernier nœud c = l ; while(c→s!=NULL) c=c→s;
noeud rechVal(noeud* l, int v){* *while(l != NULL && l→x != v)* *l = l→s;* *return l;* *}*	Rechercher une valeur dans la liste. Retour de NULL s'il n'est pas dans la liste ou sa position (pointeur qui l'indique) sinon. Si l est NULL alors l→x est interdite mais dans le while, l!=NULL la précède
void modifVal(noeud l, int v){* *noeud *t = rechVal(l,v);* *if(t != NULL) t→x++;* *}*	
noeud supprimerVal(noeud* l, int v){* *noeud *t,*c = l;* *t = rechVal(l, v);* *if(t == NULL) return l;* *if(t == l) return supprimerD(l);* *while(c→s != t)* *c = c→s;* *c→s = t→s;* *return l;* *}*	*void supprimerVal(noeud**p l, int v){* *noeud *t,*c = *pl;* *t = rechVal(c, v);* *if(t != NULL){* *if(t == c) *pl= c→s;* *else{* *while(c→s != t) c = c→s;* *c→s = t→s;* *}}* *}*
*void afficherDirect(noeud *l){* *while(l != NULL){* *printf("%d ", l→x);* *l = l→s;* *}* *}*	
*void afficherInverse(noeud *l){* *if(l != NULL){* *afficherInverse(l→s);* *printf("%d ", l→x);* *}* *}*	version itérative : *for(i= nbNoeud(l)-1 ; i>=0 ; i--){* *c = seDeplacer(l, i);* *printf("%d", c→x) ;* *}* *On peut utiliser une pile explicite*
*void triBulle(noeud *l){* *noeud *t;* *int i, j, a, n = nbNoeud(l);* *for(i=0; i <n–1; i++){* *t = l;* *for(j=0; j<n-1-i; j++){*	Tri à bulle d'une liste. Grâce à nbNoeud et le principe du tri qui consiste à comparer les valeurs 2 à 2 (valeur avec son suivant) et ceci jusqu'à la fin d'un parcours et reprendre le même principe, l'écriture devient similaire au cas de tableau. t=t→s (équivalent à j++) et t=l (éq. à remettre j à 0). Vers.2: *for(i=0; i <n–1; i++){* *t = l ;*

```
                    if(t→x > t→s→x){
                        a = t→x;
                        t→x = t→s→x;
                        t→s→x = a;
                    }
                    t = t→s;
                }
}}
```

```
                while(t→s→s != NULL{
                    if(t→x > t→s→x){
                        a = t→x;
                        t→x = t→s→x;
                        t→s→x = a;
                    }
                    t = t→s;
}}
```

```
void triSelect(noeud *l){
        int a;
        noeud *t = l, *pmax;
        while(t != NULL){
                pmax = maxListeP(t);
                a = t→x;
                t→x = pmax→x;
                pmax→x = a;
                t = t→s;
        }
}
```

Tri par sélection se base sur la recherche du max dans une liste paramétrée par le pointeur d'où commencer le permuter avant le premier. Dans le cas de recherche du max dans toute la liste, il faut supprimer t du paramètre et initialiser pmax à l.

```
noeud* maxListeP(noeud *t){
        noeud *pmax = t;
        while(t != NULL){
                if(t→x > pmax→x) pmax = t;
                t = t→s;
        }
        return pmax; }
```

```
noeud* convEntCh(noeud *l, int x){
   while(x != 0){
        l=ajouterDeb(l,creerN(x%10+'0'));
        x=x/10;
   }
  return l;
}
```

Conversion d'entier en chaîne en gérant une liste qui permet d'inverser les chiffres de l'entier. Ex. Récursive avec ajoutFin

```
noeud* convEntChR(noeud *l, int x){
    if(x!=0) l = convEntChR(l, x/10);
            return ajouterFin(l, creerN(x%10+'0'));  }
     else return NULL;        // liste encore vide au départ
}
```

```
int verifExpression(noeud *l, char T[]){
        int i = 0, b = 1;
        while(T[i] != '\0' && b == 1){
           if(T[i] == '(')
             l=ajouterDeb(l, creerN(T[i]));
             if(T[i] == ')')
                 if(sommetPile(l) == '(')
                     l = supprimerDeb(l);
                 else b = 0;
           i++;
        }
        if(l != NULL) b = 0;
        return b;
}
```

La vérification d'une expression si elle est bien parenthésée : Ex. ((a+b)+c)……

L'expression peut avoir d'autres symboles en plus de '(pour regrouper les termes de l'expression '{' , '[' , … Le principe de la solution reste le même.

La solution consiste à chaque fois qu'on rencontre une '(' de l'empiler dans la liste et si en rencontre une ')' alors en dépile (dans le cas de plusieurs symboles il faut que ça coïncide ex. '}' avec '{' en sommet de la liste)

A la fin, il faut que la liste soit vide sinon il y a un symbole supplémentaire dans l'expression qui reste non dépilé.

```
int maxListeV(noeud *l){
        int vmax;
        if(l == NULL)  return -1;
        vmax = l→x;
        while(l != NULL){
            if(l→x > vmax)  vmax=l→x;
            l = l→s;
        }
        return vmax;
}
```

Recherche de la valeur max dans une liste. La valeur -1 si la liste est vide. Rechercher le pointeur du max :

```
noeud* maxListeP(noeud *l){
        noeud *pmax = l;
        while(l != NULL){
                if(l→x > pmax→x)        pmax = l;
                l = l→s;
        }
        return pmax;
}
```

```
void ajoutApVal(noeud* l, noeud *n, int v){
        noeud *t = rechVal(l, v);
```

```
void supprimerApVal(noeud* l, int v){
        noeud *t = rechVal(l, v);
```

```
        if(t != NULL){
                n→s = t→s ;
                t→s = n;
}}
```

```
        if(t == NULL || t→s == NULL)
                printf("impossible");
        else    t→s = t→s→s;
}
```

```
nœud*  ajoutAvV(noeud* l,noeud *n,int v){
        noeud *c = l, *t = rechVal(l, v);
        if(t == NULL)  return l ;
        if(t == l)  return ajoutDeb(l, n) ;
        while(c→s != t) c = c→s;
        n→s = t ;
        c→s = n;
        return l;
}
```

```
nœud* supprimerAvVal(noeud* l, int v){
        noeud *c=l, *t = rechVal(l, v);
        if(t == NULL || t == l) return l ;
        if(t→s == NULL)
                return supprimerDeb(l);
        while(c→s→s != t)
                c = c→s;
        return l;
}
```