CORRECTION DE L'EPREUVE D'INFORMATIQUE SEPTEMBRE 2006

Problème 1 (Programmation en C)

a) Création d'un monôme

Créer un monôme ici revient à réserver un espace mémoire pouvant être utilisé pour stocker :

- Un entier le coefficient
- Un entier le degré
- Une adresse le lien vers le monôme suivant (qui sera inexistant dans ce cas)

La procédure permettant de créer un monôme est la suivante :

```
Monome* CréerMonome (int a, int n){
        Monome*p ;
        p->coef=a ;
        p->deg=n ;
        p->suivant=NULL ;
        return p ;
}
```


b) Dérivation de polynôme

Nous allons construire et retourner un polynôme qui est la dérivée de celui pris en entrée (en considérant que la dérivée du polynôme nul est lui-même). La procédure correspondante est la suivante

```
Polynôme Deriver (polynôme p) {
        Polynôme t, q, r, s ;
        T=q=NULL ;
        if (p!=NULL) {
                r=p;
                While (r! polynôme) malloe (sizeof (monomer));
                        S= (polynomme) malloe (siweof(monomer));
                        s->coef=(r->coef)*r-deg;
                        s->deg=r->deg-1;
                        if (t==NULL) t=q=s;
                        else{
                                q->suivant= s;
                                q=s
                        }
                        r=r-> suivant;
                }
        }
        return t;
}
```


c) <u>Somme de polynôme</u>

Nous allons comme à la question précédente construire et retourner un polynôme qui est la somme des deux pris en entrée (en considérant ici que si l'un des deux polynômes est nul, la somme se réduit à l'autre et si les deux sont nuls la somme l'est aussi). La procédure qui effectue cette tache est la suivante :

```
Polynôme Somme(polynome p, polynome q){
    Polynome t,q,r,s,m ;

    If((p==NULL) && (q==NULL)) return NULL ;
    If(( p!=NULL)&& (q=NULL)) m=p;
    If(( p==NULL)&& (q!=NULL)) m=q;
    else m=p;
    r= m; t=q=NULL;
    while (r !=NULL){
        s=(polynome) malloe (sizeof (monome )) ;
        *s=*r;
        s->suivant =NULL;
            if (t==NULL) t=q=s;
            else{
                q->suiant = s;
                q=s;
            }
        r= r->suivant;
    }
    If ((p! =NULL)&& (q!=NULL) {
        r=q;
        while (r!=NULL) {
            b=0; ,=t;
            while (( m!=NULL) && (b==0)) {
                b=(m->deg)==(r->deg);
                if (b)  m->coef +=r->coef;
                m=m->suivant ;
            }
            If'!b){
                S=(polynome) malloe (sizeof(monomer));
                *s=*r
                s->suivant=NULL;
                q->suivant=s;
                q=s;
            }
            r=r->suivant;
        }
    }
    return t;
}
```


d) <u>Produit d'un polynôme et un monôme</u>

Nous proposons ici une procédure récursive basée sur la relation de récurrence suivante :

$$ax^n * p(x) = ax^n * \sum b_i x^i = (a * b_0)x^{n+0} + ax^n * (\sum b_i x^i)$$

Ou encore

Produit Monotone (a,n,p) = Somme ((a*p->coef , n=p->deg), ProduitMonotone(a,n,p->suivant))

En se basant sur cette relation de récurrence on obtient lecode de la procédure suivante :

```
Polynome ProduitMonotone(int a, int n , polynome p){
        If (p==NULL) return NULL ;
        Q= (polynome) malloe (sizeof (monomer));
        q->coef=p->coef;
        q->deg = p->deg ;
        return Somme (q, ProduitMonome (a,n,p->suivant)) ;

}
```

e) Produit de deux Polynômes

Comme à la question précédente nous avons utilisé une procédure récursive basée sur la relation

$$P(x) * Q(x) = (\sum_{i=0}^{n} a_i x^i) * Q(x)$$

$$= a_0 x^0 * Q(x) + (\sum_{i=1}^{n} a_i x^i) * Q(x)$$

Produit (P ;Q)= Somme (ProduitMonome(P->coef, P->deg,Q)n Produit(P->suivant, Q)) si les deus polynômes P et Q sont nuls.

La procédure ontenue en s'appuyant sur cette relation de récurrence est :

```
Polynome Produit (polynomme p, polynome q) {
        If(( p== NULL) ||  (q==NULL) return NULL ;
        return Somme (ProduitMonome (p->coef,p->deg),q),Produit (p->suivant,q)) ;
}
```


Problème 2 (Programmation en C)

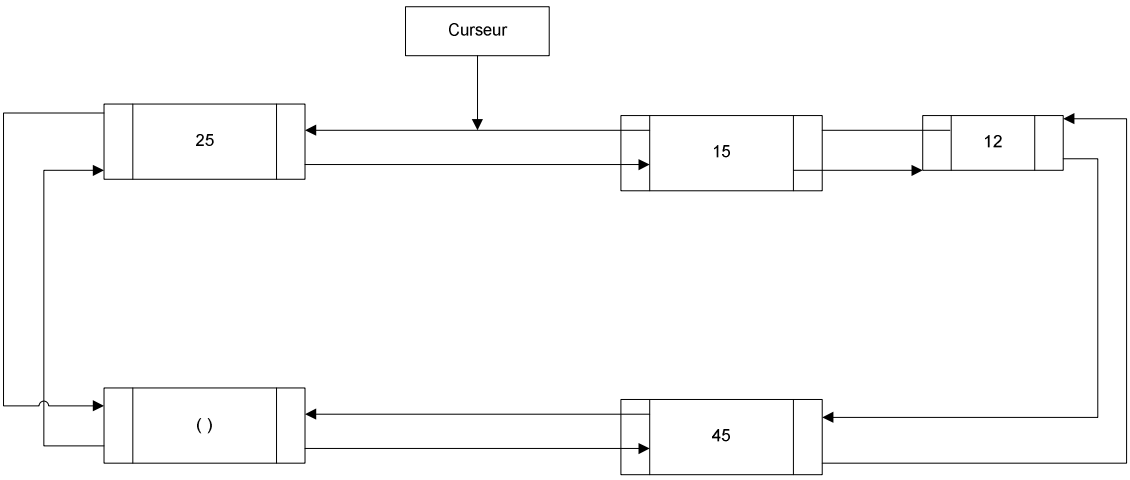
1) Nous donnons ici un fochier point h répondant à cette question. Ce fichier contient les déclarations d'une liste de fonctions permettant d'effectuer les opérations définies dur les listes avec curseur. Le fichier s'appelle curliste h ; son contenu est le suivant :

```
#ifndef_CURLISTE_H_
#define_CURLISTE_H
Curliste CrétionCurliste(void);
Void DestructionCurlite (curliste*);
Curliste DeplacementDebut (curliste*) ;
Curliste DeplacementFin (curliste*) ;
CURLISTE Deplacementsuivant5curliste*)
Curliste DeplacementPrecedent (curliste*) ;

//en supposant que element soit un type définit
Void InsertionDevantCurseur (curliste*, element)
Void SuppressionSuivant Curseur (curliste*) ;

//1 pour vrai 0 pour faux
Int CurseurEstAuDebut (curliste) ;
Int CurseurEstEnFin (curiste) :

//nombre d »élements
Int Cardinal (curliste) ;

« endif
```


2) Le fichier makefile

```
Listing : curlite.o listing.o
Gcc-o curliste curliste.o  listing.o

Curlite.o: curliste.c
Gcc-o curliste.o – c curliste.c-W- Wall- qnsi- pedantic

Listing.o: listing.c curliste.h
Gcc –o listing.o –o listing.c –W –Wall – ansi -pedantic
```

3) <u>Dessin</u>



Déclaration de types

Le curseur étant la position entre deux éléments pour le repérer nous utiliserons les adresses de ses deux éléments. On obtient la déclaration de types suivants pour une liste avec curseur.


```
Struct nœud {
        Info elt ;
        Struct nœud* next ;
        Struct noeud* old;
}
Typedef strucr noeud* pointeur;

Struct enreg{
        Pointeur tete;
        Pointeur avant;
        Pointeur apres ;
}
Typedef strucr enreg curliste ;
```

Le type info présent dans la déclaration du type struct nœud paut etre n'importe quel type élémentaire – entier, réel, caractère ou complexe (tableau, enregistrement, chaine, liste0..)

Le type curliste se définie donc comme un enregistrement contenant trois champs :

Les pointeurs avant et après désignent respectivement les adresses de l'élément avant le curseur et l'élément après le curseur.

Le pointeur tete représentant comme son nom l'indique l'adresse de la tête de notre liste doublement chaînée.

4) Fonction de création d'une curliste.

Voici le code de la fonction qui contient et retourne une liste valide :

```
Curliste Création Curliste (void){
        Curliste e ;
        e.tete= NULL
        e.avant = NULL
        e.après = NULL ;
        return e ;
}
```

Puisque la liste est vide les adresses de position du curseyr prennent la valeur NULL.

Fonction de destruction d'une curliste

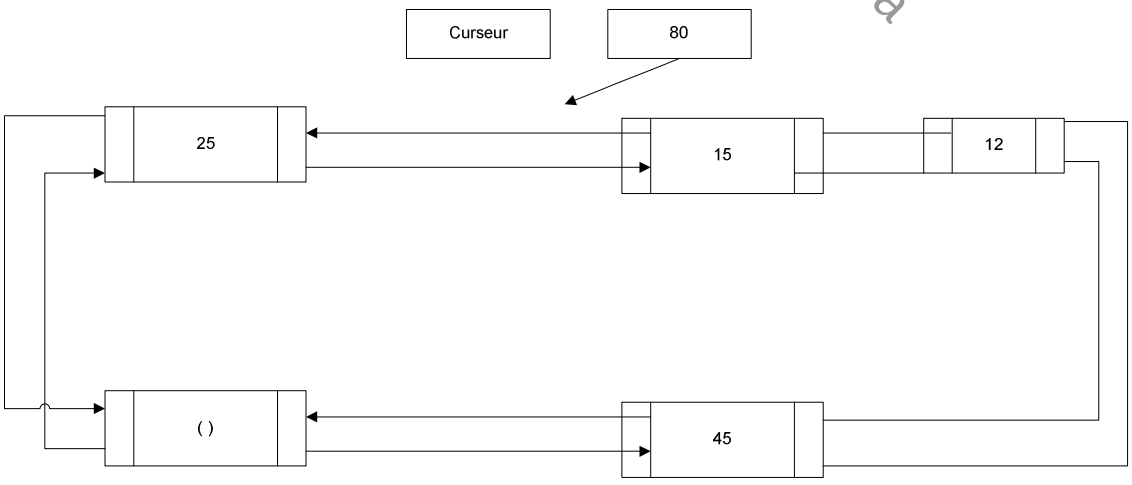
La destruction d'une liste chaînée en générale consiste à libérer l'espace mémoire occupé par celle-ci, cela se fait successivement pour chacun de ses nœuds. D'où le code suivant :

```
Void DestructionCurliste (curliste*c){
        Pointeur p ;
        White ((c->tete !=NULL) && (e->tete->next!=tete)){
                c->tete=c->tete->next ;
                free (p);
        }
        If(c->tete!=NULL) free(c->tete->next;
        c->tete= c->tete=c->avant = c->après= NULL;
}
```

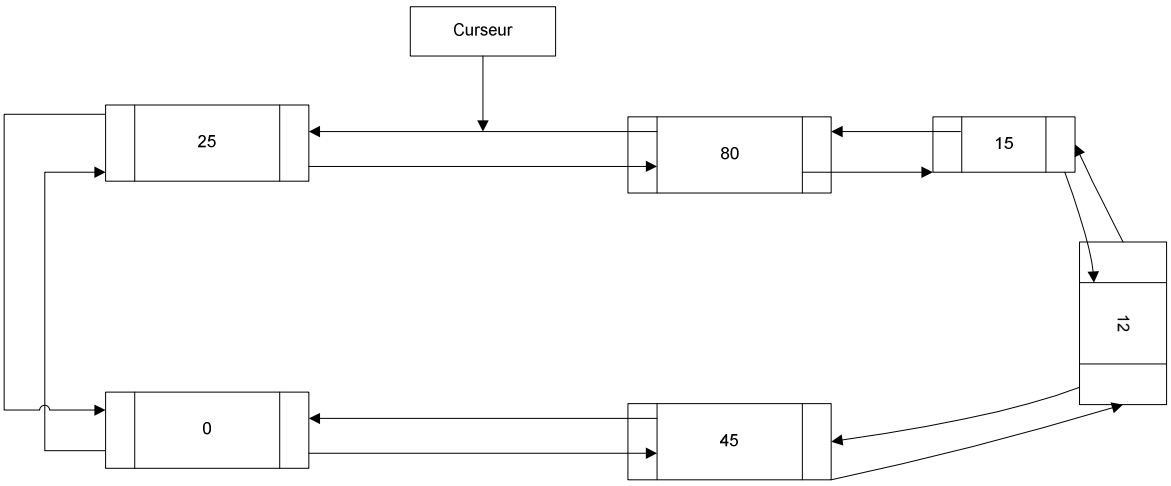

5) Fonction Insertion d'un élément devant le curseur

Cette opération peut se schématiser comme suit :

Avant l'insertion on a



Apres l'insertion on a :

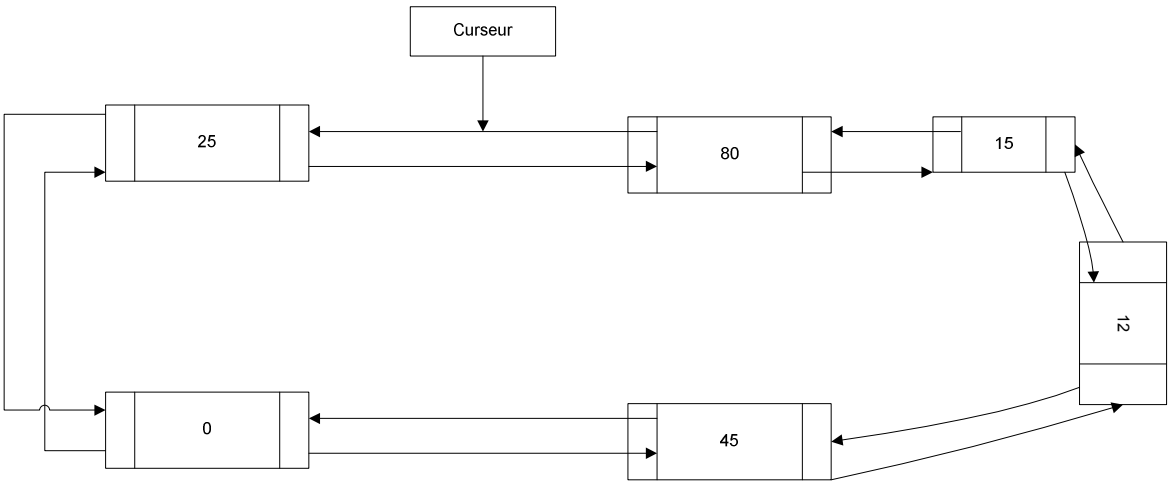


L code correspondant est le suivat :

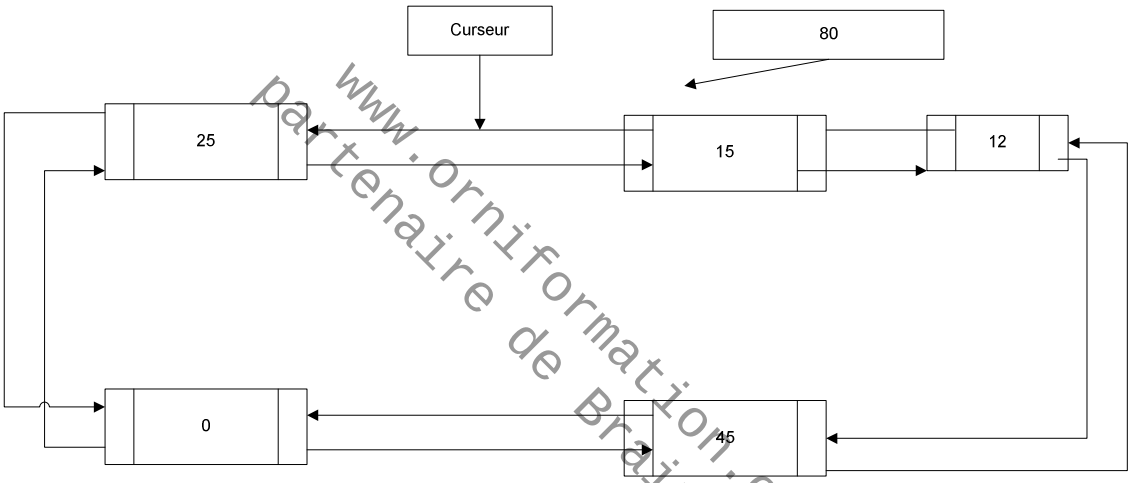

```
Vold InsertionDevant Curseur( curliste* c,
éléùent e){
        Pointeur p ;
        P= (pointeur) mlloe (siweof(struct nœud)) ;
        P->elt=e :

        If(c->tete==NULL){ /* si la liste est vide*/
                p->next=p ;
                p->old=p ;
                c->tete=p
                c->avant = c->apres=tete;

        }
        Else {/*si la liste a au loins un  élément */
                c->avant->next=p ;
                p->old = c->apres ;
                c->apres ->old = p ;
                c->après-> old = p ;
                c-> après = p ;
        }
}
```

Suppression de l'élément suivant le curseur.

Cette opération peut se schématiser comme suit :
Avant la suppression on a



Après la suppression on a



Le code de la procédure correspondante à ce schéma est :

```
Void SuppressionSuivantCurseur (curseur(curliste* c{
        If (c->tete != NULL){/* si laliste a moins d'un élément*/
                If (c->tete->next != tete){ /* si la lsite a plus d'un element*/
                        c->avant->next=c->apres->next ;
                        c->après->next->old=c->avant;
                        free (c->après);
                }
                Else{/* si la liste a un seul élément*/
                        Free (c->tete) ;
                        c->tete= c->avant = c->après = NULL;
                }

        }
}
```