Liste Chainée

```c
/*représentation physique chainée*/

struct element
{
    char cle;
    struct element*suivant;
};


struct liste
{
    struct element* premier;
    struct element* dernier;
};

/*services à exporter*/

void creer_liste(struct liste*);  /*service de création*/
unsigned liste_vide(struct liste); /*service de consultation*/
void ajout_entete(char, struct liste*); /*service de modification*/
void ajout_enqueue(char, struct liste*);/*service de modification*/
void ajouter_apres_ele_ref(char,struct element*,struct liste*);/*service de modif
ication*/
void ajouter_avant_ele_ref(char,struct element* ref,struct liste*);/*service de m
odification*/
void supprimer_entete(struct liste *);/*service de modification*/
void supprimer_enqueue(struct liste *);/*service de modification*/
void supprimer_ele_ref(struct liste *,struct element *);/*service de modification
*/
void effacer(struct liste *);/*service de modification*/
struct element * chercher(char c, struct element*p);/*service de consultation*/
void visiter(struct element*p,void(*oper)(struct element*));/*service de consulta
tion*/
unsigned liste_longueur(struct element *p);/*service de consultation*/
void trier(struct element *p);/*service de modification*/
struct element* has_cle(struct element *);/*service de consultation*/
void affiche_liste_chainée(struct liste*);/*service d'affichage*/


#include<stdlib.h>
#include<assert.h>
#include "Ex1_Liste_lineraire.h"

void creer_liste(struct liste* ll)
{
```

```
    assert(ll!=NULL);
    ll->premier=NULL;
    ll->dernier=NULL;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

unsigned liste_vide(struct liste ll)
{
    return ll.premier==NULL && ll.dernier==NULL;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void ajout_entete(char c, struct liste* ll)
{
    struct element *q;
    assert(ll!=NULL);
    q=(struct element*)malloc(sizeof(struct element));
    assert(q!=NULL);
    q->cle=c;
    q->suivant=ll->premier;
    ll->premier=q;
    if(ll->dernier==NULL)/*if (liste_vide(*ll))*/
        ll->dernier=q;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void ajout_enqueue(char c, struct liste *ll)
{
    struct element*q;
    assert(ll!=NULL);
    q=(struct element*)malloc(sizeof(struct element));
    assert(q!=NULL);
    q->cle=c;
    q->suivant=NULL;
    if(liste_vide(*ll))
    {
        ll->premier=q;
    }
    else
    {
        ll->dernier->suivant=q;
    }
    ll->dernier=q;
}
```

```c
/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++*/

void ajouter_apres_ele_ref(char c,struct element* ref,struct liste* ll)
{
    struct element *e;
    assert(ll!=NULL);
    assert(ref!=NULL);
    assert(!liste_vide(*ll));
    e=(struct element*)malloc(sizeof(struct element));
    assert(e!=NULL);
    e->cle=c;
    e->suivant=ref->suivant;
    if(ref==ll->dernier)
        ll->dernier=e;
    ref->suivant=e;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++*/

void ajouter_avant_ele_ref(char c,struct element* ref,struct liste* ll)
{
    struct element*e;
    assert(ll!=NULL);
    assert(ref!=NULL);
    assert(!liste_vide(*ll));
    e=(struct element*)malloc(sizeof(struct element));
    assert(e!=NULL);
    *e=*ref;
    /*en c l'affectation est définie sur les variables ayant un type struct :
    e->cle=ref->cle et e->suivant=ref->suivant*/
    /*mise à jour l'espace référencé par p*/
    ref->cle=c;
    ref->suivant=e;
    if(ref==ll->premier)
        ll->premier=e;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++*/

void supprimer_entete(struct liste *ll)
{
    struct element *p,*q;
    assert(ll!=NULL);
    p=ll->premier;
    if(p->suivant== NULL)
    {
        free(p);
```

```
        return creer_liste(ll);
    }
    else
    {
        ll->premier=p->suivant;
        free(p);
    }
    if(ll->premier==NULL) ll->dernier==NULL;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void supprimer_enqueue(struct liste *ll)
{
    struct element *p;
    if(ll->premier==ll->dernier) supprimer_entete(ll);
    else
    {
        p=ll->premier;
        while (p->suivant!=ll->dernier)
        {
            p=p->suivant;
        }
        p->suivant=NULL;
        free(ll->dernier);
        ll->dernier=p;
    }
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void supprimer_ele_ref(struct liste *ll,struct element *ref)
{
    assert(ref!=NULL);
    struct element *p;
    p=ll->premier;
    while (p->suivant!=ref)
    {
        p=p->suivant;
    }
    p->suivant=ref->suivant;
    free(ref);

}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/
```

```c
void effacer(struct liste *ll)
{
    if(liste_vide(li))
        return creer_liste(ll);

    while(ll->premier != NULL)
        supprimer_enqueue(ll);
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

struct element * chercher(char c, struct element*l)
{
    while(l&&(l->cle!=c))
    l=l->suivant ;
    //échec : !l => l== NULL
    //succès :l->cle==info*/
    return(l) ;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void visiter(struct element *l,void(*oper)(struct element*))
{
    while(l)
    {
    /*appliquer à l'élément porté par l le traitement est fourni par oper*/
        (*oper)(l) ;
    /*passer à l'élément suivant */
        l=l->suivant ;
    }
    /*    Exemple:
    struct element * point_de_depart ;
    void afficher(struct element * q)
    {
        printf (''%u d\n '', q->cle) ;
    }
    visiter(point_de_depart, afficher) ;
    void incrementer (struct element *q)
    {
        q->cle++ ;
    }
    visiter(point_de_depart, incrementer) ;
    */
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/
```

```c
unsigned liste_longueur(struct element *l)
{
    int length = 0;

    if(liste_vide(p))
        return length;

    while(l != NULL)
    {
        length++;
        l = l->suivant;
    }

    return length;
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void trier(struct element *l)
{
    struct element *tmp;
    while(l<l->suivant)
    {
        *tmp=*l;
        *l=*l->suivant;
        *l->suivant=*tmp;
    }
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

struct element* has_cle(struct element *ref)
{
    assert(ref!=NULL);
    return(ref->cle);
}

/*++++++++++++++++++++++++++++++++++++----------
++++++++++++++++++++++++++++++++++++++*/

void affiche_liste_chainée(struct liste *ll)
{
    struct element*e;
    e=ll->premier;
    while(e)
    {
        printf("%d\n",e->cle);
```

```
        e=e->suivant;
    }
}
```