1) La procédure creeListe initialise une liste :

```
Procédure creerListe( r L : pointeur vers cellule)
Début
L←NULL ;
Fin ;
```

```
void creerListe(cellule** L){
        *L=NULL;
    }
```

Remarque : Lorsque le pointeur L est NULL, cela signifie que la liste est vide.

2) ajoute un élément (de type entier) en début de chaîne

voici une procédure qui ajoute une valeur 'v' dans une liste L

Algo	C

procédure ajoutDeb(d/r L : pointeur vers cellule, v : entier)
variables
p: pointeur vers cellule;
debut
allouer(p);
P→donnee← v ;
P→suivant← L ;
L ←p ;
Fin
	void ajoutDeb(cellule** L, int v){
	cellule *p;
	p=(cellule*)malloc(sizeof*p);
	p->donnee=v;
	p->suivant=*L;
	*L=p;
	}

Algo	C

Procédure ajoutFin (d/r L :Liste, v :t)
variables p, q: pointeur vers cellule;
debut
allouer(p);
P->donnee←v ; p→suivant←NULL ;
si(L !=NULL)alors
Q← L ;
tant que (q→suivant !=NULL)
q←q→suivant ;
finTque ;
q→suivant←p ;
sinon
L←p ;
FINSI.
Fin
	void ajoutFin(cellule** L, int v)
	{
	cellule *p, *q;
	p=(cellule*)malloc(sizeof(cellule));
	p->donnee=v ;
	p->suivant=NULL ;
	if(*L !=NULL){
	q=*L ;
	while(q->suivant!=NULL){
	q=q->suivant ;
	}
	q->suivant=p ;
	}
	else
	*L=p ;
	}

4) une fonction qui Recherche un élément ayant une valeur v dans une liste

Algo	C

Fonction rechercher(L :Liste,v :entier)**:booleen**
 	
Début
SI(L<> NULL) **alors**
Tantque (L -> suivant <> NULL **ET** L->donnee<> v)**faire**
L<-L-->suivant:
FinTQ
si(L->donnee=v) **alors**
retourner vrai;
Sinon
Retourner faux;
FinSi;
Sinon
Retourner faux;
FinSi;
FIN ;
	int rechercher(cellule* L, int v) **{**
	
	If(L!=NULL){
	While((l->suivant !=NULL) &&(L->donnee!=v))
	L=L->suivant;
	If(L->donnee==V) return 1;
	else return 0;
	}
	else return 0;
	}

5) SUPPRIME UN ÉLÉMENT EN DÉBUT DE CHAÎNE

DANS CE CAS, IL FAUT FAIRE POINTER LA LISTE SUR LE NŒUD SUCCESSEUR DU NŒUD À SUPPRIMER. SINON, APRÈS LA LIBÉRATION DU PREMIER NŒUD/MAILLON/CELLULE DE LA LISTE, LE RESTE DE LA LISTE CHAÎNÉE EST PERDU.

ALGO

```
Procédure suppDeb(d/r L :Liste)
  R: pointeur vers cellule;
  Debut
  R<-L;
  Si(L<> NULL) alors
  L<-L->suivant;
  Liberer(R);
  Finsi;

  Fin ;
```

C

```
void suppDeb(cellule** L){

Liste R=*L;
if(*L!=NULL){
*L=(*L)->suivant;
free(R);
}


}
```

6) SUPPRIME UN ÉLÉMENT EN FIN DE CHAÎNE

Algo

Procédure suppFin(d/r L :pointeur vers cellule)
R: pointeur sur cellule;
Debut
R<-L;
Si(L<>NULL) alors
Si(L->suivant <> NULL) alors
Tant que (R-> suivant->suivant<> NULL) faire
R<-R->suivant;
Fintq;
Liberer (R->suivant);
R->suivant<-NULL;
Sinon
Liberer(L);
L<-NULL;
FinSi
FinSi;

Fin ;

version avec un seul pointeur,

C

```
void suppFin(cellule** l)
{ cellule *q,*p; p=*l;
if(*l!=NULL)
{
if(p->suivant!=NULL) // cas de +eurs elt dans la liste
{
q=p->suivant;
while (q->suivant!= NULL) //2 pointeurs qui avancent l'un
devant l'autre
{
q=q->suivant;
p=p->suivant;
}
p->suivant=NULL;
free(q);
}
else // Cas où il y a un seul elt dans la liste
{
*l=NULL;
free(p);
}
}
}
```

7) SUPPRIME TOUTE LA CHAÎNE

LORSQU'UNE LISTE CHAÎNÉE N'EST PLUS UTILE AU PROGRAMME, IL FAUT LA DÉTRUIRE. SINON, LA MÉMOIRE RESTE OCCUPÉE POUR RIEN

Algo

Procédure SuppList(d/r L :liste)

Debut

Procédure SuppList(d/r L :liste)
q :pointeur sur cellule ;
Debut
Tant que (L <> NULL) faire
q←L;
L←q->suivant ;
Libérer(q) ;
FINTQ
Fin ;

C

```
void suppList(cellule* l)
{

cellule *q;
while(*l!=NULL)
{
q=*l;
*l=q->suivant;
free(q);
}
 }
```

;

8) AFFICHE UNE CHAÎNE

Algo

```
Procédure afficheList(L :liste)
Debut

Fin ;
```

C

```
void afficheList(cellule* L)
 {

à faire version itérative et version
                récursive
}
```

EXERCICES

1- compte le nombre d'éléments de la chaîne (version itérative et version récursive)
2- calcule la somme des valeurs de la chaîne (version itérative et version récursive)
3-Retravaillez toutes les procédures en les transformant en des fonctions
4-Retravaillez toutes les procédures avec une liste composée d'une tête et une taille:

Type Structure etudiant
Nom: chaine;
...
fin
Type Structure cellule
donnee: etudiant ;
suivant : pointeur vers cellule;
fin
Type structure Liste
Tete: pointeur vers cellule;
Taille: entier;
Fin