

Romain SAHUT
Gpe 2A1

LDC: Dossier de programmation

TP : LISTE DOUBLEMENT CHAINEE

Définition des types

TypeListeDC = structure

tete : pointeur de TypeCellule

TypeCellule = structure

pred : pointeur de TypeCellule

cle : entier

succ : pointeur de TypeCellule

Prototypes des fonctions

- Fonction menu (vide) : ret Entier
- Fonction creationListeAvecVerification(in-out liste : pointeur sur TypeListeDC) : ret pointeur sur TypeListeDC
- Fonction creationListe() : ret pointeur sur TypeListeDC
- Fonction creationCellule() : ret Pointeur sur TypeCellule
- Fonction listeVide(in liste : Pointeur sur TypeListeDC) : ret booléen
- Fonction listeExiste(in-out l: Pointeur sur TypeListeDC) : ret booléen
- Fonction insereListe(in-out l: Pointeur sur TypeListeDC) : ret Vide
- Fonction insereEltListe(in-out l: Pointeur sur TypeListeDC ,in x : entier) : ret Entier
- Fonction insereAvant (in-out l: Pointeur sur TypeListeDC ,in-out celluleAinserrer: Pointeur sur TypeCellule,in-out cellule: Pointeur sur TypeCellule) : ret Entier
- Fonction insereApres (in-out l: Pointeur sur TypeListeDC ,in-out celluleAinserrer: Pointeur sur TypeCellule,in-out cellule: Pointeur sur TypeCellule) : ret Entier
- Fonction rechercheListe(in-out l: Pointeur sur TypeListeDC) : ret vide
- Fonction recherche(in-out l: Pointeur sur TypeListeDC ,x : entier) : ret Pointeur sur TypeCellule
- Fonction rechercheEltListe(in-out l: Pointeur sur TypeListeDC ,x : entier) :ret Pointeur sur TypeCellule
- Fonction supprimeElt(in-out l: Pointeur sur TypeListeDC) : ret vide
- Fonction supprimeEltListe(in-out l: Pointeur sur TypeListeDC ,in-out cellule: Pointeur sur TypeCellule) : ret entier
- Fonction moyenne(in-out l: Pointeur sur TypeListeDC) : ret vide
- Fonction moyenneListe(in-out l: Pointeur sur TypeListeDC) : ret réel
- Fonction afficheListe(in-out l: Pointeur sur TypeListeDC) : ret vide
- Fonction libereEltsListe(in-out l: Pointeur sur TypeListeDC) : ret vide

Préambule

Pour ce projet, il fallait bien faire attention à ne faire de mauvaises affectations, par exemple déclarer un pointeur de cellule en l'affectant (sur la même ligne) de l'adresse donnée par l↑.tete, or si l n'existe pas : plantage.

De même, il faut initialiser le pointeur de liste a NIL au début et à chaque libération mémoire, sinon l pointe n'importe ou (ancienne adresse de la structure, mais qui peut être affectée a un autre programme a ce moment) et le programme pense que la liste existe et plante.

La principale difficulté de ce projet réside, je pense, dans la rigueur demandée par l'utilisation des pointeurs (une erreur de conception est dure a retrouver quand le source se compile sans problème) et dans l'allocation dynamique de la mémoire qui oblige, encore, a être très rigoureux afin de ne pas affecter les variables qui, peut être, n'ont pas pu être allouées par manque de mémoire

Dans ce dossier de programmation, Je tenterai être aussi clair que possible concernant l'algorithme, mais la trivialité de la conception de certaines fonctions est telle que je ne les commenterai pas (ou du moins juste en décrivant le résultat).

Fonction principale

Description du résultat :

Désallouer la liste doublement chaînée après l'avoir créé et modifié a volonté.

Idée de l'algo :

TANT QUE l'utilisateur de quitte pas

- Afficher le menu et demander à l'utilisateur de choisir
- Lancer la fonction demandée
- Afficher le résultat de la fonction

FTANT

Désallouer la liste SI elle existe.

Lexique des variables :

l pointeur sur TypeListeDC Pointeur sur la liste à créer

quit booléen VRAI SI l'utilisateur veut quitter l'application

Algo :

l ← NIL

quit ← FAUX //initialisations de nos variables

TANT QUE(quit ← FAUX)

 SELON QUE menu() est

 CAS 1) l ← creationListe()

 CAS 2) insereListe(l)

 CAS 3) rechercheListe(l)

 CAS 4) supprimeElt(l)

 CAS 5) moyenne(l)

 CAS 6) afficheListe(l)

 CAS 7) libereEltsListe(l)

 CAS 8) quit ← VRAI

 Defaut) écrire « Choix entre 1 et 8! Merci »

 FSELON

FTANT

Ecrire « Fin d'exécution du programme »

SI l != NIL ALORS

 libereEltsListe(l)

 l ← NIL

FSI

Fonctions secondaires

Fonction menu (vide) : ret Entier

Description du résultat :

Retourne le choix de l'utilisateur, lui demande de ressaisir s'il ne tape pas ce qui est attendu

Idée de l'algo :

TANT QUE saisie incorrecte

-Afficher tous les choix possibles

-Demander le choix de l'utilisateur

FTANT

Lexique local des variables :

Choix	caractère	La valeur saisie par l'utilisateur
Continue	booléen	VRAI si saisie incorrecte

Algo :

TANT QUE continue

écrire « _____,s\$* Manipulation d'une liste doublement chaînée *\$,_____ »

écrire « 1.Créer une liste »

écrire « 2.insérer un élément »

écrire « 3.Rechercher un élément »

écrire « 4.Supprimer un élément »

écrire « 5.Calculer la moyenne de la liste »

écrire « 6.Afficher la liste »

écrire « 7.Liberer la mémoire »

écrire « 8.Quitter l'application »

écrire « Votre choix: »

choix ← lire

SI choix<'0' OU choix>'9' ALORS

continue=FAUX

FSI

FTANT

retour_de_menu ->choix-'0'

Fonction creationListeAvecVerification(in-out liste : pointeur sur TypeListeDC) : ret pointeur sur TypeListeDC

Description du résultat :

Vérifie si une liste existe déjà, sinon demande a creationListe() de la créer

Lexique des fonctions :

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen

listeVide(in liste: pointeur sur TypeListeDC) : ret booléen

creationListe() : ret pointeur sur TypeListeDC

Idée de l'algo :

SI la liste existe déjà

-on retourne l'adresse de notre ancienne liste (pour ne pas la désaffecter)

SINON

-on crée une nouvelle liste et on renvoie son adresse

FSI

Algo :

```
SI listeExiste(liste) ALORS
    écrire « Une liste a la fois, supprimez la liste courante avant! »
    retour_de_creationListeAvecVerification ← liste
SINON
    retour_de_creationListeAvecVerification ← creationListe()
FSI
```

Fonction creationListe() : ret pointeur sur TypeListeDC

Description du résultat:

Alloue l'espace mémoire nécessaire à l'existence de la liste et renvoie un pointeur sur la liste

Lexique des fonctions:

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen

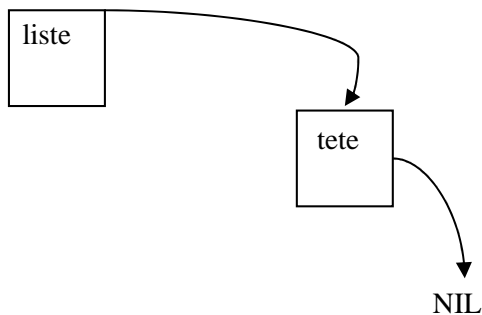
Lexique des variables :

liste Pointeur sur TypeListeDC

Algo :

```
liste ← Alloue (TypeListeDC)
SI listeExiste(liste) ALORS //SI la liste a été correctement créée
    liste↑.tete ← NIL //On initialise la tete
FSI
retour_de_creationListe ← liste //On ne vérifie pas ici qu'elle a bien pu être créée
```

Schéma :



Fonction creationCellule() : ret Pointeur sur TypeCellule

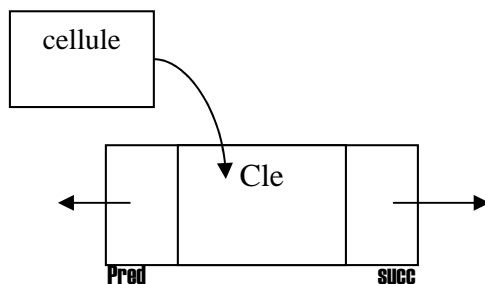
Description du résultat :

Alloue l'espace mémoire nécessaire à l'existence de la cellule et renvoie un pointeur sur la cellule

Algo :

```
retour_de_creationCellule ← alloue (TypeCellule) //On ne vérifie pas ici qu'elle a bien pu être créée
```

Schema :



Fonction listeVide(in liste : Pointeur sur TypeListeDC) : ret booléen

Description du résultat :

Retourne VRAI si la liste est vide (mais elle existe)

Algo :

retour_de_listeVide ← (l↑.tete = NIL) //Une liste vide existe et sa tete pointe sur NIL

Fonction listeExiste(in-out l: Pointeur sur TypeListeDC) : ret booléen

Description du résultat :

Retourne VRAI si la liste existe

Algo :

retour_de_listeExiste ← (l != NIL) //Une liste existe SI son pointeur ne pointe pas sur NIL

Fonction insereListe(in-out l: Pointeur sur TypeListeDC) : ret Vide

Description du résultat :

Demande la valeur à insérer, tente de insérer et affiche le résultat selon le retour de insereEltListe(l,x)

Lexique des fonctions

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

insereEltListe(in-out l: Pointeur sur TypeListeDC ,in x : entier) : ret Entier

Lexique des variables

x Entier

a Entier

Algo :

SI listeExiste(l) ALORS

 écrire « Tapez la valeur a insérer:»

 x ← lire

FSI //SI la liste n'existe pas, on ne demande rien et insereEltListe nous

a ← insereEltListe(l,x) //retournera le cas d'erreur et on sera prévenu

SELON QUE a EST

 CAS 0) écrire «la mémoire est pleine, la cellule n'a pas pu être créée »

 CAS 1) écrire «L'insertion a été effectuée dans la liste vide»

 CAS 2) écrire «L'insertion a été effectuée en tete de liste»

 CAS 3) écrire «L'insertion a été effectuée en milieu de liste»

 CAS 4) écrire «L'insertion a été effectuée en queue de liste»

 CAS 5) écrire «La liste n'existe pas!»

FSELON

Fonction insereEltListe(in-out l: Pointeur sur TypeListeDC ,in x : entier) : ret Entier

Description du résultat :

Vérifications de base (liste existe...) et insère élément x AVANT élément immédiatement supérieur, ou en queue

Lexique des fonctions

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

insereAvant (in-out l: Pointeur sur TypeListeDC ,in-out celluleAinserrer: Pointeur sur TypeCellule,in-out cellule: Pointeur sur TypeCellule) : ret Entier

insereApres (in-out l: Pointeur sur TypeListeDC ,in-out celluleAinserrer: Pointeur sur TypeCellule,in-out cellule: Pointeur sur TypeCellule) : ret Entier

Idée de l'Algo

Si la liste existe et contient au moins 1 élément et mémoire allouée

Tant qu'on est pas a la fin de la liste ET qu'on pointe sur une cellule dont la cle est < x

Pointeur ← celluleSuivante

Ftant

→ maintenant on pointe juste après x ou en queue

Si on est en queue

→ c'est la plus grande valeur de la liste

On insère x en tant que dernier élément

Sinon on est en milieu ou en tete de liste

On insère x avant la cellule que l'on pointe

fsi

fsi

Lexique des variables

ptr Pointeur sur TypeCellule

cellule Pointeur sur TypeCellule

Algo :

cellule ← creationCellule()

SI listeExiste(l) ALORS //SI l existe

SI cellule != NIL ALORS //emplacement Mémoire de la cellule correctement allouée

cellule↑.cle ← x

SI listeVide(l) ALORS //Liste vide

cellule↑.pred ← NIL

cellule↑.succ ← NIL

l↑.tete ← cellule

retour_de_insereEltListe ←(1)

SINON

ptr ← recherche(l,x) //on insère soit avant la première valeur>x ,soit après le dernier

SI cellule↑.cle<ptr↑.cle ALORS //SI on veut insérer la plus petite valeur de la liste ou en milieu

retour_de_insereEltListe ←(insereAvant(l,cellule,ptr))

SINON

retour_de_insereEltListe ←(insereApres(cellule,ptr))

FSI

FSI

FSI

retour_de_insereEltListe ←(0)

SINON

retour_de_insereEltListe ←(5) //retour de fonction additionnel par rapport au sujet, SI la liste n'existe

pas...

FSI

Fonction insereAvant (in-out l: Pointeur sur TypeListeDC, in-out celluleAinsérer: Pointeur sur TypeCellule, in-out cellule: Pointeur sur TypeCellule) : ret Entier

Description du résultat :

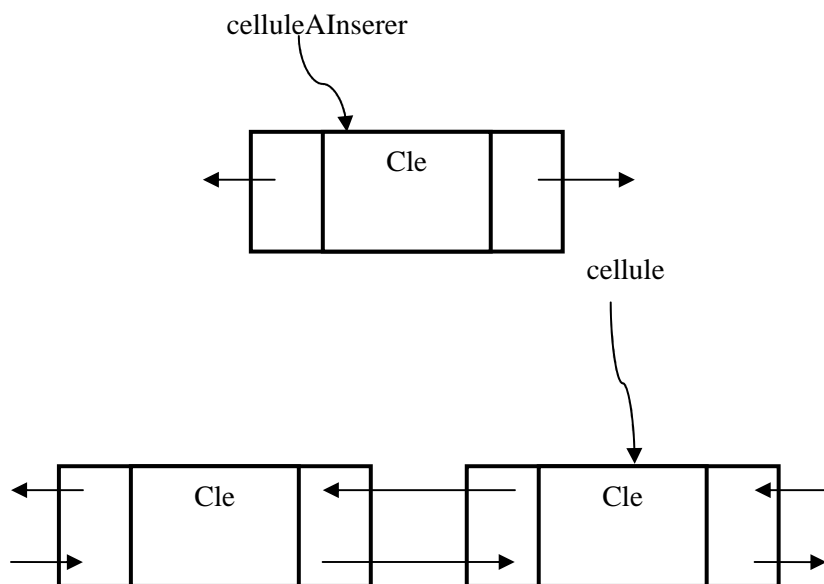
Insère la cellule 'celluleAinsérer' AVANT 'cellule' et retourne si l'insertion s'est faite en tete ou non

Algo :

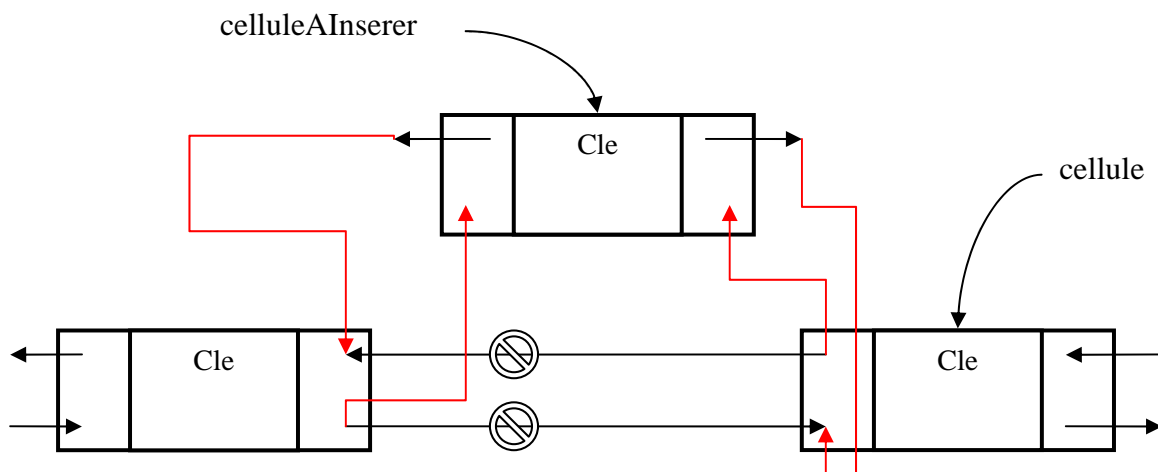
```
celluleAinsérer↑.pred ← cellule↑.pred  
celluleAinsérer↑.succ ← cellule  
cellule↑.pred ← celluleAinsérer  
SI celluleAinsérer↑.pred = NIL ALORS  
  l↑.tete ← celluleAinsérer  
  retour_de_insereAvant ← (2)  
SINON  
  (celluleAinsérer↑.pred) ↑.succ ← celluleAinsérer  
  retour_de_insereAvant ← (3)  
FSI
```

Schéma :

Avant :



Après :



Fonction insereApres (in-out l: Pointeur sur TypeListeDC ,in-out celluleAinsérer: Pointeur sur TypeCellule,in-out cellule: Pointeur sur TypeCellule) : ret Entier

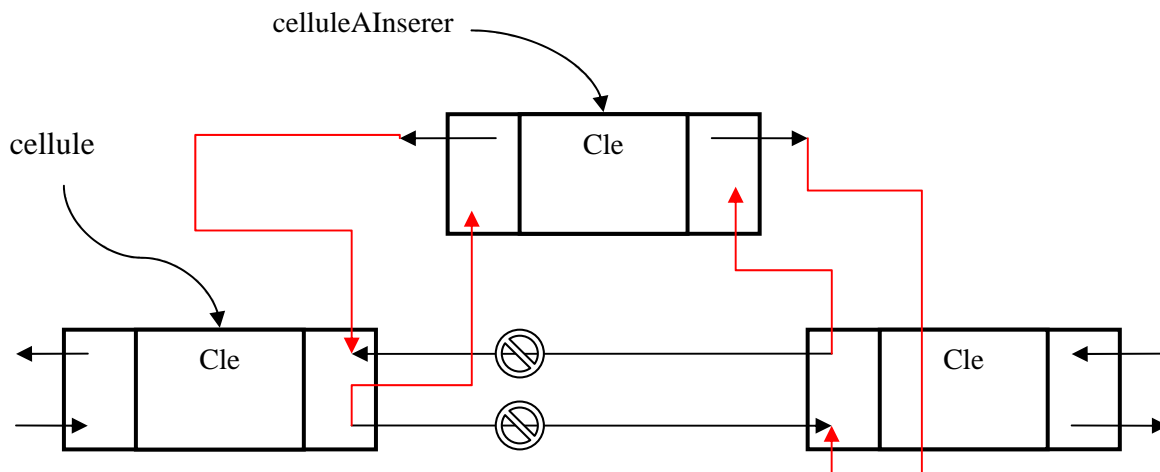
Description du résultat :

Insère la cellule 'celluleAinsérer' APRES 'cellule' et retourne si l'insertion s'est faite en queue ou non

Algo :

```
celluleAinsérer↑.succ ← cellule↑.succ
celluleAinsérer↑.pred ← cellule
cellule↑.succ ← celluleAinsérer
SI celluleAinsérer↑.succ = NIL ALORS
    retour_de_insereApres ← (4)
SINON
    retour_de_insereApres ← (3)
FSI
```

Schéma :



Fonction rechercheListe(in-out l: Pointeur sur TypeListeDC) : ret vide

Description du résultat :

Demande a l'utilisateur la valeur a chercher et demande a rechercheEltList de la trouver, affiche « trouvée » ou « pas trouvée »

Lexique des fonctions

```
rechercheEltListe(in-out l: Pointeur sur TypeListeDC ,x : entier) :ret Pointeur sur TypeCellule
listeExiste(in liste : pointeur sur TypeListeDC) : ret booléen
listeVide(in liste : pointeur sur TypeListeDC ) : ret booléen
```

Lexique des variables

x Entier

Algo :

```
SI listeExiste(l) ALORS
    SI listeVide(l) ALORS
        écrire «La liste est vide»
    SINON
        écrire « Tapez la valeur a chercher:»
        x ←lire
        SI (rechercheEltListe(l,x) = NIL) ALORS
            écrire «élément cherché n'appartient pas à la liste»
        SINON
            écrire «élément a été trouvé»
```

```

FSI
FSI
SINON
    écrire «La liste n'existe pas!»
FSI

```

Fonction recherche(in-out l: Pointeur sur TypeListeDC ,x : entier) : ret Pointeur sur TypeCellule

Description du résultat :

Fonction de recherche utilisée par insereEltListe pour retourner un pointeur sur TypeCellule élément immédiatement supérieur a x ou la tete (si $x < l \uparrow .tete$)

Lexique des fonctions :

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen
 listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Lexique des variables :

cellule Pointeur sur TypeCellule
 celluleSauve Pointeur sur TypeCellule

Algo :

```

SI listeExiste(l) ALORS
    cellule ← l↑.tete
    TANT QUE (cellule↑.cle < x ET cellule↑.succ != NIL) //On est sur le premier supérieur a x ou sur le dernier
        cellule ← cellule↑.succ
    FTANT
    retour_de_recherche ← (cellule) //On retourne ce pointeur
SINON
    écrire «La liste n'existe pas!»
    retour_de_recherche ← NIL // a défaut de retourner la cellule recherchée, on retourne NIL
FSI

```

Fonction rechercheEltListe(in-out l: Pointeur sur TypeListeDC ,x : entier) :ret Pointeur sur TypeCellule //recherche la valeur EXACTE

Description du résultat :

Recherche la valeur x EXACTE, renvoie un pointeur sur une cellule dont la cle est x, NIL sinon

Lexique des fonctions

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen
 listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Lexique des variables

cellule Pointeur sur TypeCellule

Algo :

```

SI listeExiste(l) ALORS
    cellule ← l↑.tete
    TANT QUE (cellule↑.cle < x ET cellule↑.succ != NIL) //On parcourt TANT QUE c'est trop petit et qu'il reste
des cellules
        cellule ← cellule↑.succ
    FTANT
    SI cellule↑.cle != x ALORS //c'est que x n'est pas dans la liste
        cellule ← NIL
FSI

```

```
    retour_de_rechercheEltListe←(cellule) //retourne NIL SI x n'est pas dans l, ou un pointeur sur élément
recherché
    SINON
    écrire «La liste n'existe pas!»
    retour_de_rechercheEltListe ←NIL
FSI
```

Fonction supprimeElt(in-out l: Pointeur sur TypeListeDC) : ret vide

Description du résultat :

Supprime une cellule dont la cle est choisie par l'utilisateur et lui affiche le résultat de cette suppression

Lexique des fonctions

listeExiste(in liste : pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Lexique des variables

x Entier

a Entier

c Pointeur sur TypeCellule

Algo :

SI listeExiste(l) ALORS

SI listeVide(l) ALORS

a ← 6

SINON //SI l existe et n'est pas vide

écrire ("Tapez la valeur a supprimer:»

x ← lire

c ← rechercheEltListe(l,x)

a ← supprimeEltListe(l,c)

FSI

SINON

a ← 5

FSI

SELON QUE a est

CAS 0) écrire «élément à effacer n'existe pas!»

CAS 1) écrire «La suppression s'est effectuée sur une liste à un élément»

CAS 2) écrire «La suppression a été effectuée en tete de liste»

CAS 3) écrire «La suppression a été effectuée en milieu de liste»

CAS 4) écrire «La suppression a été effectuée en queue de liste»

CAS 5) écrire «La liste n'existe pas!»

CAS 6) écrire «La liste est vide»

FSELON

Fonction supprimeEltListe(in-out l: Pointeur sur TypeListeDC, in-out cellule: Pointeur sur TypeCellule) : ret entier

Description du résultat :

Supprime la cellule pointée par cellule de la liste (avec MàJ des pointeurs des cellules juxtaposées à notre cible bien sur...)

Lexique des fonctions

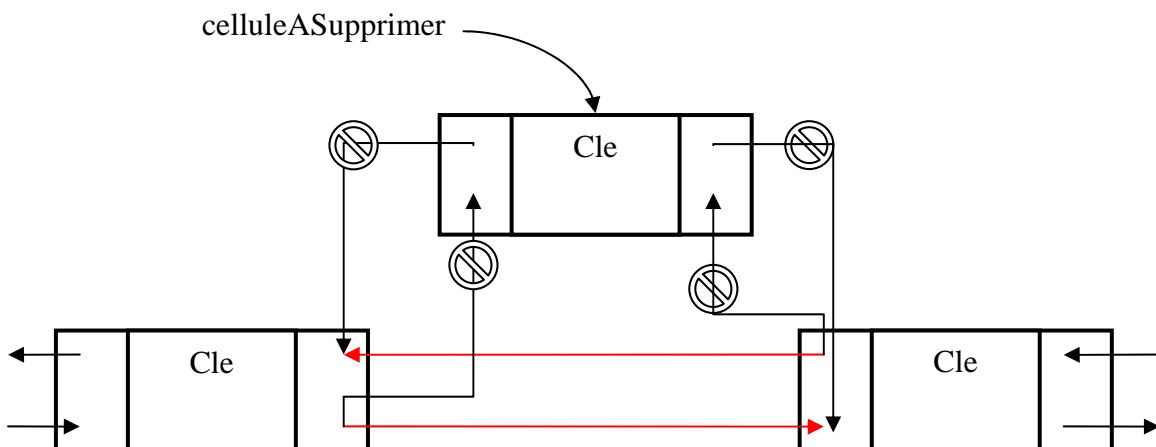
listeExiste(in liste : pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Algo :

```
SI listeExiste(l) ALORS //Je vérifie dans presque chaque fonction, comme ça on peut
                        //l'extraire et la réutiliser presque à l'identique
SI (cellule!= NIL) ALORS //S'il y a des cellules dans l
    SI (cellule↑.pred = NIL ET cellule↑.succ = NIL) ALORS //cas Unique élément
        l↑.tete ← NIL
        libere(cellule)
        retour_de_supprimeEltListe←(1)
    SINON
        SI (cellule↑.pred = NIL) ALORS //cas 1er élément de la liste
            l↑.tete ← cellule↑.succ
            (cellule↑.succ)↑.pred ← NIL
            libere(cellule)
            retour_de_supprimeEltListe←(2)
        SINON // cas Dernier élément de la liste
            SI (cellule↑.succ = NIL) ALORS
                (cellule↑.pred)↑.succ ← NIL
                libere(cellule)
                retour_de_supprimeEltListe←(4)
            SINON //cas En milieu de liste
                (cellule↑.pred)↑.succ ← cellule↑.succ //On coupe les liens, laissant notre cellule inaccessible
                (cellule↑.succ)↑.pred ← cellule↑.pred
                libere(cellule)
                retour_de_supprimeEltListe ←(3)
        FSI
    FSI
FSI
FSI
FSI
FSI
retour_de_supprimeEltListe ←(0)
FSI
```

Schéma :



Fonction moyenne(in-out l: Pointeur sur TypeListeDC) : ret vide

Description du résultat :

Vérifie la validité de la liste vis-à-vis d'un calcul de moyenne (doit exister), affiche cette moyenne

Lexique des fonctions

listeExiste(in liste : pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Algo :

```
SI listeExiste(l) ALORS
  SI listeVide(l) ALORS
    écrire «La liste est vide! Elle n'a pas de moyenne...»
  SINON
    écrire « Moyenne de la liste = « ,moyenneListe(l)
  FSI
SINON
  écrire «La liste n'existe pas!»
FSI
```

Fonction moyenneListe(in-out l: Pointeur sur TypeListeDC) : ret réel

Description du résultat :

Calcule la moyenne de la liste et retourne cette valeur

Lexique des fonctions

listeExiste(in liste : pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Lexique des variables

i	entier
somme	réel
cellule	Pointeur sur TypeCellule

Algo :

```
i ← 0
somme ← 0
SI listeExiste(l) ALORS
  cellule ← l↑.tete
  TANT QUE(cellule!= NIL) //Tant qu'il reste des cellules
    somme ← cellule↑.cle + somme
    cellule ← cellule↑.succ
    i++
  FTANT
  retour_de_moyenneListe ←((somme*1.0)/i)
SINON
  écrire «La liste n'existe pas!»
  retour_de_moyenneListe ←(0.0) // on considère que pas de liste, c'est pas élément, donc 0
FSI
```

Fonction afficheListe(in-out l: Pointeur sur TypeListeDC) : ret vide

Description du résultat :

Affiche à l'écran, dans l'ordre, de la tete vers la queue, les valeurs des clés des cellules de la liste

Lexique des fonctions

listeExiste(in liste :pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Lexique des variables

nbLigne Entier

nbVal Entier

Cellule Pointeur sur TypeCellule

Algo :

nbVal ← 0

nbLigne ← 1

SI listeExiste(l) ALORS

SI listeVide(l) ALORS

 écrire «La liste est vide! on a que la tete!»

SINON

 écrire « |»

 cellule ← l↑.tete

 TANT QUE (cellule!= NIL){ //Tant qu'il reste des cellules

 écrire cellule↑.cle

 écrire « | »

 cellule ← cellule↑.succ

 nbVal++

 SI (nbVal%15 = 0) ALORS //Ts les 15 valeurs, on saute une ligne

 écrire « |»

 SI (nbLigne%25 = 0) ALORS //On montre les résultats de la page chaque 25 lignes

 PAUSER LE PROGRAMME POUR FACILITER LA LECTURE (getchar())

 FSI

 nbLigne++

 FSI

FTANT

 écrire « Appuyez sur ENTREE pour continuer»

FSI

SINON

 écrire «La liste n'existe pas!»

FSI

Fonction libereEtsListe(in-out l: Pointeur sur TypeListeDC) : ret vide

Description du résultat :

Libère TOUTE la liste (y compris la structure pointée par l) si elle existe, sinon dit qu'il faut en créer une avant.

Lexique des fonctions

listeExiste(in liste : pointeur sur TypeListeDC) : ret booléen

listeVide(in liste : pointeur sur TypeListeDC) : ret booléen

Lexique des variables

cellule Pointeur sur TypeCellule

Algo:

```
SI listeExiste(l) ALORS
  cellule ← l↑.tete
  TANT QUE(NON listeVide(l))
    cellule ← l↑.tete
    l↑.tete ← cellule↑.succ
    SI (cellule↑.succ!= NIL) ALORS
      (cellule↑.succ)↑.pred ← NIL      //On n'allait pas assigner une valeur à NIL↑.pred!
    FSI
    libere(cellule)                    //cellule courante desallouée
  FTANT                               //La tete de la liste pointe a NIL , il n'y a plus de cellule
  libere(l)                            /liste desallouée
  écrire «La liste a été correctement effacée»
SINON
  écrire «La liste n'existe pas, il faut en créer une avant.»
FSI
```