Une donnée modifiée est une variable source passée « telle quelle » à un algorithme tandis qu'une donnée copiée est la copie locale à l'algorithme d'une telle variable. Il s'ensuit que les modifications effectuées par l'algorithme sur une donnée modifiée impactent directement la variable source tandis que celles effectuées sur une donnée n'impactent sont sans effet sur elle.

```
Jenesuispasenretard : (L : liste)
Donnée modifier : L
Variable locale : temp, liste, stocker la liste
Début
    Si (L =/= NULL) alors
        temp <- L
        Tant que (temp.next =/= NULL) faire
            Temp <- temp.next
        Fin tant que
        Temp.next <- L
    Fin si
FIN

Retard (L : liste) : entier
Donnée copiée : L
Variable locale : cpt, entier, compteur
Début
    cpt <- 0
    Si (L =/= NULL) alors
        Faire
            cpt <- cpt +1
            L <- L.next
        Tant que (L.next =/= NULL)
    sinon retourner 0
    Fin si
    Retourner cpt
Fin
```

```
Début
        cpt <- 0
        tant que(L=/=NULL) faire
                cpt<-cpt+1
                L <- L.next
        Fin tant que
        Retourner cpt

RetardKadaRécursif (L : liste) : entier
Donnée copiée : L
Début
        Si (L = NULL) alors
                retourner 0
        Sinon
                Retourner 1 + RetardKadaRécursif (L.next)
        Fin si
Fin

Position (L : liste, x : entier) : entier
Donnée copiée : L, liste
                 x, entier
Variable locale :cpt
Début
        Si (L = NULL) alors
                retourner 0
        sinon si (L.date = x )
                retourner 1
        Sinon
                Retourner (1 + Position (L.next))
         Fin si
Fin
```

**Algorithme 9:** $Supprimer(l : liste, x : T) : booléen$

```
Donnée modifiée : La liste l à modifier
Donnée          : L'élément x à supprimer
Variable locale : Adresse cour du maillon en cours de traitement
Variable locale : Adresse prec du prédécesseur de cour
Variable locale : Booléen ok qui indique si x a effectivement été supprimé
Résultat        : La valeur de ok
début
    ok ← faux
    // Si la liste est vide, il n'y a évidemment rien à supprimer !
    si l ≠ ∅ alors
        si l → info = x alors
            cour ← l
            l ← l → succ
            ok ← vrai
            libérer cour
        sinon
            // Itérer jusqu'au point de suppression, supprimer, raccorder
            prec ← l
            cour ← l → succ
            tant que cour ≠ ∅ et ok = faux faire
                si cour → info = x alors
                    ok ← vrai
                    // Court-circuit de cour
                    prec → succ ← cour → succ
                    // Suppression de cour
                    libérer cour
                sinon
                    // Itération des deux variables prec et cour
                    si cour → info < x alors
                        prec ← cour
                        cour ← cour → succ
                    sinon
                        cour ← ∅
                    fin
                fin
            fin
        fin
    fin
    retourner ok
fin
```

**Algorithme 8:** *Insérer*$(l : liste, x : T)$

**Donnée modifiée** : La liste $l$ à modifier
**Donnée** : L'élément $x$ à insérer
**Variable locale** : Adresse *cour* du maillon en cours de traitement
**Variable locale** : Adresse *prec* du prédécesseur de *cour*
**Variable locale** : Booléen *trouve* qui indique quand le point d'insertion a été trouvé

```
début
    // Cas de la liste vide
    si l = ∅ alors
        l ← réserver maillon
        l → info ← x
        l → succ ← ∅
    sinon
        // Cas de l'insertion en tête de liste
        si x < l → info alors
            cour ← réserver maillon
            cour → info ← x
            cour → succ ← l
            l ← cour
        sinon
            // Itérer jusqu'au point de d'insertion, créer, raccorder
            trouve ← faux
            prec ← l
            cour ← l → succ
            tant que cour ≠ ∅ et trouve = faux faire
                si x < cour → info alors
                    trouve ← vrai
                    prec → succ ← réserver maillon
                    prec ← prec → succ
                    prec → info ← x
                    prec → succ ← cour
                sinon
                    // Itération des deux variables prec et cour
                    prec ← cour
                    cour ← cour → succ
                fin
            fin
            // Reste un cas à ne pas oublier : insertion en queue de liste
            si cour = ∅ alors
                prec → succ ← réserver maillon
                cour ← prec → succ
                cour → info ← x
                cour → succ ← ∅
            fin
        fin
    fin
fin
```

```
Suppr_récursif (l : liste)
Donnée modifiée : l, la liste à supprimer
début
        si l non vide alors
                Suppr_récursif(l->succ)
                liberer l
                l <- vide // ici ce n'est pas surperflu et même indispensable !
        fin(si)
fin
Suppr_iteratif (l : liste)
Donnée modifiée : l, la liste à supprimer
Variable locale : kill, le prochain maillon à supprimer
début
        tant que l non vide faire
                kill <- l
                l <- l->succ
                libérer kill
        fin
        // superflu : l <- vide
Fin
```

```
Creation_1_n (n : entier) : liste
Donnée : n, le nombre d'éléments de la liste à créer
Variables locales : l, la référence de la tête de la liste à créer
                    q, la référence de la queue de liste
                    i, compteur de boucle
Retourne : la liste créée
début
        si n = 0 alors retourner vide
        l <- réserver Maillon
        l->info <- 1
        q <- l
        i <- 1
        tant que i < n faire
                i <- i + 1
                q <- q->succ <- réserver Maillon
                q->info <- i
        fin
        q->succ <- vide
        retourner l
fin


void print_list_rec (list l) {
        if (!l) printf("->x");
        else {
                printf("->(%d)", l->info);
                print_list_rec(l->succ);
        }
}
```

Exo 1 Derniere_Position (l : liste, x : T)

Données : l, la liste, et x, l'élément dont on cherche la dernière position

Variables locales : p, position courante, pos de la dernière occurrence de x

début

```
p <- pos <- 0
tant que l non vide faire
    p <- p + 1
    si l->info = x alors pos <- p
    l <- l->succ
fin
retourner pos
```

fin