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 )

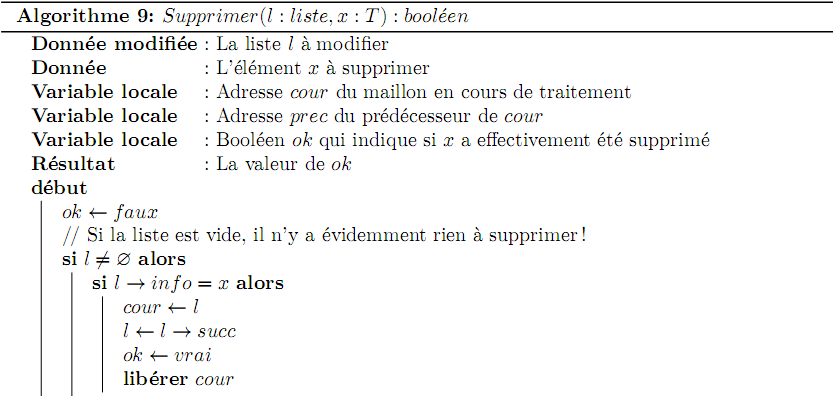
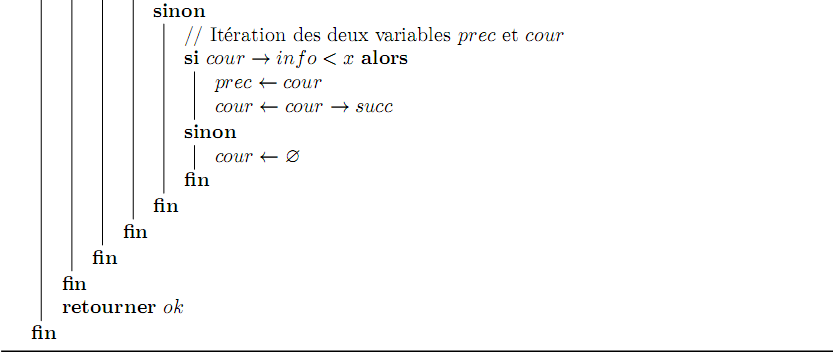
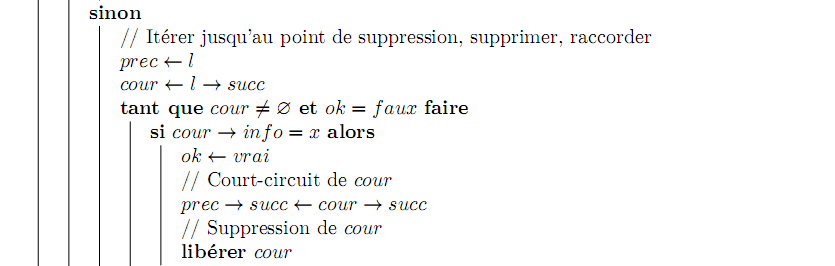
retourner 1

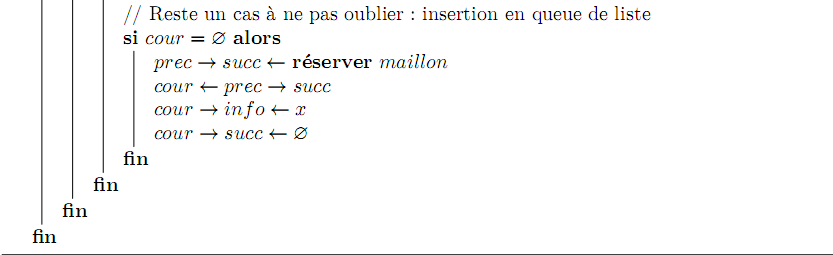
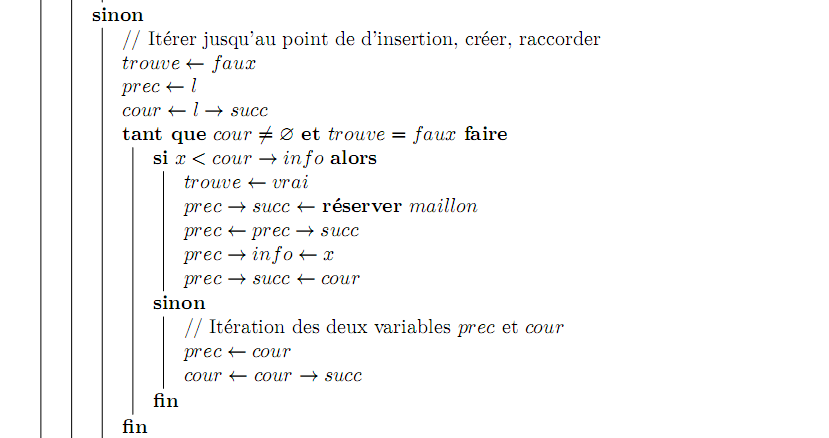
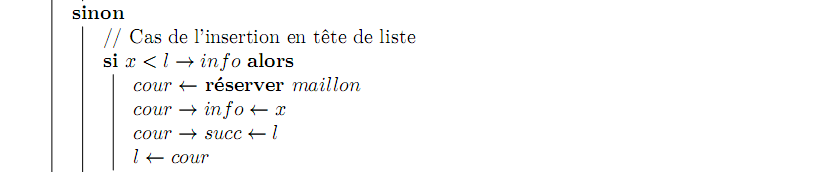
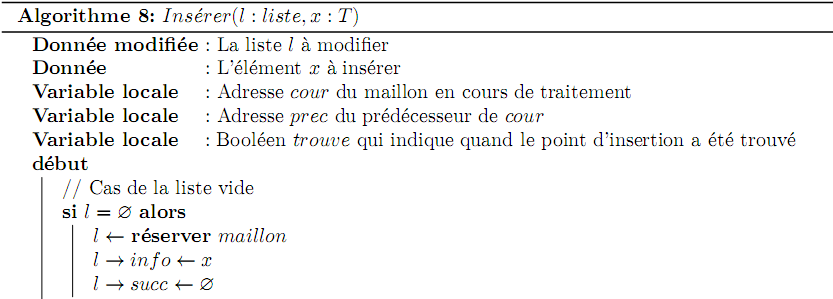
Sinon

Retourner (1 + Position (L.next))

Fin si

Fin



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