```
/**
 * Détermine la position de la première occurrence d'un élément dans une liste
 * linéaire. La position du maillon de tête est 1.
 *
 * La fonction doit être implémentée en 2 versions : en itératif et en récursif.
 *
 * Paramètre l : liste à examiner.
 * Paramètre e : élément à trouver.
 *
 * Retourne : la position de la première occurrence de e dans l, ou 0 si e n'est
 * pas dans l.
 */
int position(liste l, int e)
{
    // Déclaration des variables de travail
    int pos;

    // Recherche de e
    pos = 1;
    while ( (l != NULL) && (l->info != e) ) {
        pos++;
        l = l->succ;
    }

    // Cas e pas trouvé
    if ( l == NULL ) {
        return 0;
    }

    // Cas e trouvé à la position pos
    return pos;
}

int position_rec(liste l, int e)
{
    // Cas liste vide
    if ( l == NULL ) return 0;

    // Cas e en premier élément
    if ( l->info == e ) return 1;

    // Cas autre
    int pos = position_rec(l->succ, e);
    if ( pos == 0 ) return 0; // e pas trouvé
    return 1 + pos;           // e trouvé à la position 1+pos
}


// Exemple d'appel sur une liste L pour déterminer la position de la première
// occurrence d'une valeur v
// posi = position(&L, v);    ou     posi = position_rec(&L, v);
//
```

```
/**
 * Insère un élément dans une liste doublement chaînée linéaire après une
 * position donnée. Si la position spécifiée est 0, l'insertion se fait en
 * tête de liste.
 *
 * Paramètre l   : liste à modifier.
 * Paramètre pos : position après laquelle Insèrer.
 * Paramètre e   : élément à Insèrer.
 *
 * Retourne : 1 si succès ou 0 sinon (un paramètre est incorrect).
 */
int inserer(liste* l, int pos, int e)
{
    // Cas paramètre(s) incorrect(s)
    if ( (l == NULL) || (pos < 0) ) return 0;

    // Cas paramètres corrects

    // 1. Créer le nouveau maillon à insérer
    maillon* pnv = new maillon;
    pnv->info = e;

    // 2. Cas insertion en tête de liste
    if ( (*l == NULL) || (pos == 0) ) {
        pnv->succ = *l;
        pnv->pred = NULL;
        if ( *l != NULL ) (*l)->pred = pnv;
        *l = pnv;
        return 1;
    }

    // 3. Cas insertion en coeur ou fin de liste

    // 3.1. Chercher le lieu d'insertion
    maillon* p = *l;
    while ( (pos > 1) && (p->succ != NULL) ) {
        p = p->succ;
        pos--;
    }

    // 3.2. Insérer le nv maillon après le maillons pté par p
    //      (dans tous les cas, même si pos demeure > 1)
    pnv->succ = p->succ;
    pnv->pred = p;
    if ( p->succ != NULL ) p->succ->pred = pnv;
    p->succ = pnv;

    return 1;
}


// Exemple d'appel sur une liste L pour insérer un maillon de valeur v à la
// position posi+1 :
// ok = inserer(&L, v, posi);
```

```
/**
 * Supprime d'une liste doublement chaînée linéaire l'élément situé à une
 * position donnée. Si la position est supérieure à la longueur de la liste,
 * le dernier maillon est supprimé.
 *
 * Paramètre l   : liste à modifier.
 * Paramètre pos : position de l'élément à supprimer.
 *
 * Retourne : 1 si succès ou 0 sinon (un paramètre est incorrect).
 */
int supprimer(liste* l, int pos)
{
    // Cas paramètre(s) incorrect(s)
    if ( (l == NULL) || (pos <= 0) ) return 0;

    // 1. Cas liste vide
    if ( *l == NULL ) return 1;

    // 2. Cas suppression du maillon de tête
    maillon* p = *l;
    if ( pos == 1 ) {
        *l = p->succ;
        if ( *l != NULL ) (*l)->pred = NULL;
        free(p);
        return 1;
    }

    // 3. Cas suppression d'un maillon du coeur ou de fin de liste

    // 3.1. Chercher le maillon à supprimer
    while ( (pos > 1) && (p->succ != NULL) ) {
        pos--;
        p = p->succ;
    }

    // 3.2. Supprimer le maillon pointé par p
    p->pred->succ = p->succ;
    if ( p->succ != NULL ) p->succ->pred = p->pred;
    free(p);

    return 1;
}

// Exemple d'appel sur une liste L pour supprimer le maillon de position posi :
// ok = supprimer(&L, posi) ;
```