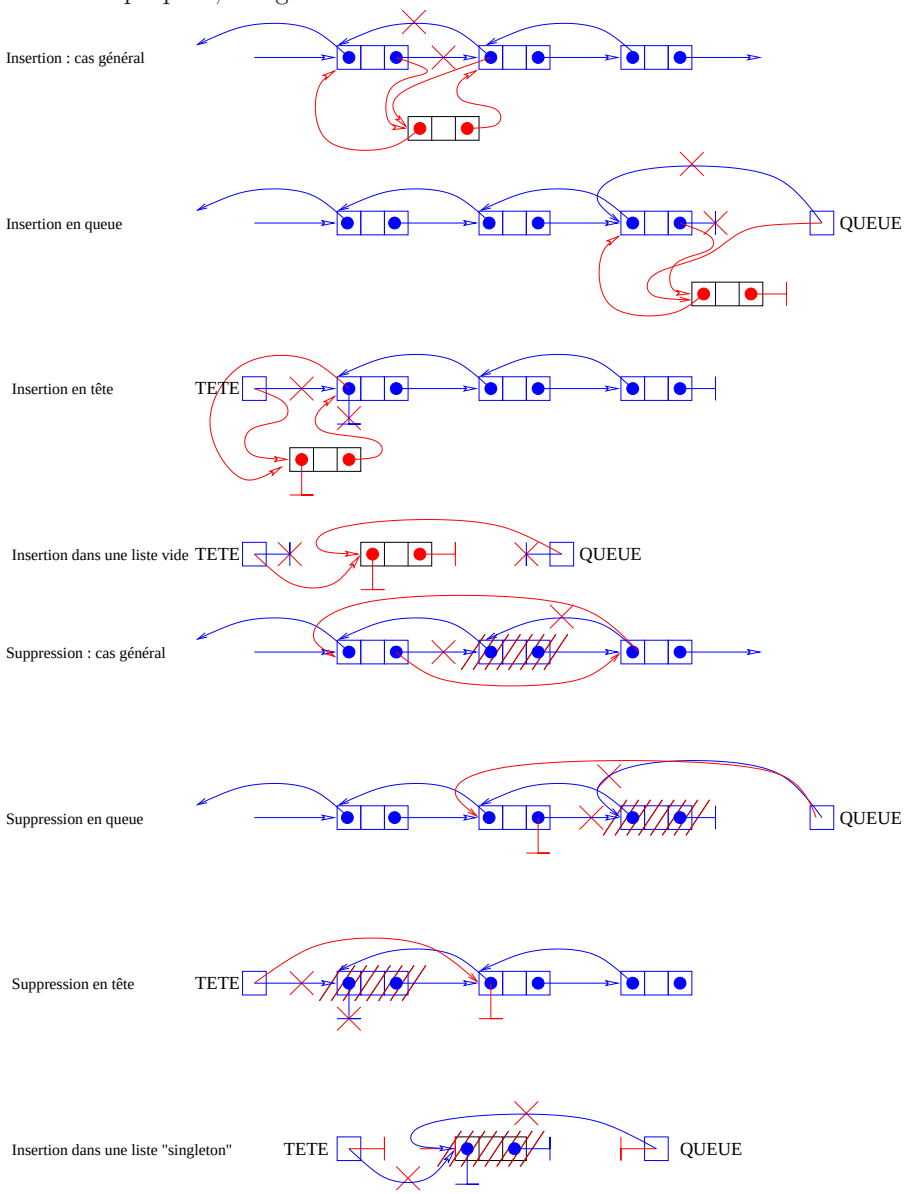
1. **liste doublement chaînée :**
   (a) Voir code proposé (pages suivantes)
   (b) Voir code proposé, et figures suivantes :



   (c) Calcul simple, à condition de prendre en compte la complexité des primitives elles-même lorsqu'elles sont utilisées
   (d)
```
cellule c = tete ;
entier cpt = 0 ;
while (c != Nil)
  if (element(c) == valeur)
    cpt ++ ;
afficher (cpt) ;
```

2. **longueur moyenne des branches** : voir cours.

```c
#include<stdio.h>
#include<stdlib.h>
#define TMax 10
#define Nil -1

typedef int cellule ;
typedef int position ;
typedef int valeur ;

struct liste {
  cellule tete ;
  cellule queue ;
  cellule libre ;
  valeur val[TMax] ;
  cellule prec[TMax] ;
  cellule suiv[TMax] ;
} l ;
// Une seule liste, déclarée en variable globale, argument implicite
// de toutes les opérations.
// ------------------------------------------------------------------------
// Fonctions utilitaires
// ------------------------------------------------------------------------
void erreur(char * s)
{
  fprintf(stderr, "ERREUR FATALE: %s\n", s) ;
  exit (-1) ;
}
void init_liste()
{
  int i ;
  l.tete = Nil ;
  l.queue = Nil ;
  l.libre= 0 ;
  for (i=0 ; i<TMax-1 ; i++)
    l.suiv[i] = i+1 ;
  l.suiv[TMax-1] = Nil ;
}
void affiche_table()
{
  int i ;
  printf("   | Val|Prec|Suiv|  ") ;
  printf("Tete: %d ; Queue : %d ; Libre : %d", l.tete, l.queue, l.libre) ;
  printf("\n-------------------\n") ;
  for (i=0 ; i<TMax ; i++)
    printf("%2d | %2d | %2d | %2d |\n", i, l.val[i], l.prec[i], l.suiv[i]) ;
}
void affiche_liste()
{
  cellule c ;
  for (c = l.tete ;  c != Nil ; c = l.suiv[c])
    printf("%2d|", l.val[c]) ;
  printf("\n") ;
}
// ------------------------------------------------------------------------
// Gestion des cases libres ("pile des cases libres")
// ------------------------------------------------------------------------
cellule newCell()
{
  cellule temp = l.libre ;
  if (temp == Nil) erreur("Plus de cellules") ;
  l.libre = l.suiv[temp] ;
  return temp ;
}

void freeCell(cellule c)
{
  l.suiv[c] = l.libre ;
  l.libre = c ;
}
```

```
// ---------------------------------------------------------------------
// Primitives de bas niveau
// ---------------------------------------------------------------------
cellule suivant (cellule c)
{
  return l.suiv[c] ;
}

cellule precedent (cellule c)
{
  return l.prec[c] ;
}

cellule pos2cel (position p)
{
  cellule c ;
  int n ;

  if (p==1) return l.tete ;
  for (n=1, c=l.tete ; c != Nil && n < p ; c = l.suiv[c], n++)
    ;
  if (n == p) return c ;
  erreur("position inexistante") ;
}

cellule cel2pos (cellule c)
{
  cellule c_cour ;
  int n ;

  for (n=1, c_cour=l.tete ; c_cour != Nil && c_cour != c ; c_cour = l.suiv[c_cour], n++)
    ;
  if (c_cour == c) return n ;
  erreur("cellule pas dans la liste") ;
}


// ---------------------------------------------------------------------
// Primitives de "haut" niveau
// ---------------------------------------------------------------------

cellule tete ()
{
  return l.tete ;
}

valeur element(cellule c)
{
  return l.val[c] ;
}

void ajouter(valeur v , position p)
{
  cellule y ;
  cellule x = newCell() ;

  l.val[x] = v ;
  //  printf("j'ajoute %d, position %d\n", v, p) ;

  if (p==1) { // insertion en tête, avec ou sans liste vide
   if (l.tete == Nil) { // liste vide
      l.tete = l.queue = x ;
      l.prec[x] = l.suiv[x] = Nil ;
   }
   else {
     l.prec[x] = Nil ;
     l.suiv[x] = l.tete ;
     l.tete = x ;
     l.prec[l.suiv[x]] = x ;
   }
  }
```

```
  else { // cas général : on cherche la cellule précedent la nouvelle
    y = pos2cel(p-1) ;
    l.suiv[x] = l.suiv[y] ;
    l.prec[x] = y ;
    if (l.suiv[y] == Nil) // insertion en queue
      l.queue = x ;
    else  // insertion quelconque
      l.prec[l.suiv[x]] = x ;
    l.suiv[y] = x ;
  }
}

void supprimer(position p)
{
  cellule c ; // pointera dans tous les cas sur la cellule à supprimer
  //  printf("je supprime la position %d\n", p) ;

  if (p == 1) {
    c = l.tete ;
    if (l.queue == l.tete) { // liste singleton
      l.tete = l.queue = Nil ;
    }
    else { // suppression en tête
      l.prec[l.suiv[c]] = Nil ;
      l.tete = l.suiv[c] ;
    }
  }
  else { // suppression pas en tete
    c = pos2cel(p) ;
    if (l.suiv[c] == Nil)
      l.queue = l.prec[c] ;
    else
      l.prec[l.suiv[c]] = l.prec[c] ;
    l.suiv[l.prec[c]] = l.suiv[c] ;
  }
  freeCell(c) ;
}

// ------------------------------------------------------------------------
// Programme principal
// ------------------------------------------------------------------------
main()
{
  init_liste() ;
  ajouter(11,1) ;
  ajouter(12,2) ;
  ajouter(13,3) ;
  ajouter(14,4) ;
  ajouter(15,5) ;
  ajouter(16,6) ;
  ajouter(17,7) ;
  ajouter(18,8) ;
  ajouter(19,9) ;
  ajouter(20,10) ;
  supprimer(4) ;
  ajouter(3,1) ;
  supprimer(9) ;
  ajouter(1,2) ;
  supprimer(1) ;
  ajouter(4,3) ;
  supprimer(2) ;
  ajouter(1,4) ;
  affiche_liste() ;
  affiche_table() ;
}
```