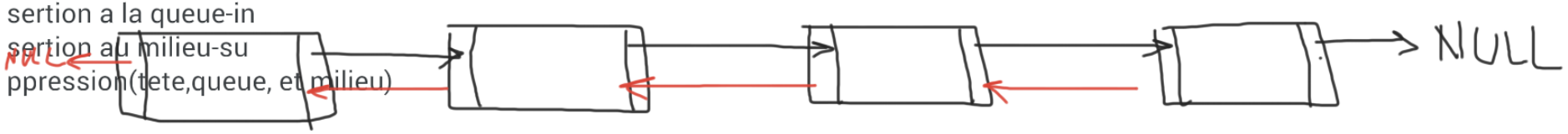
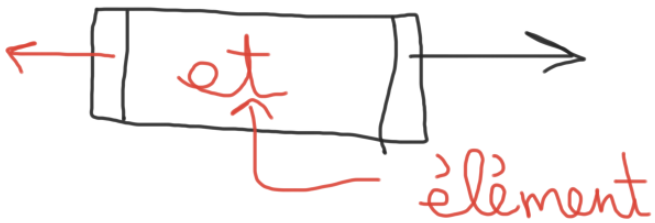
ALGO

Liste bidirectionnelle :
tous les opérations qu on peut faire sur un liste chainé-i
nsertion a la téte-in
sertion a la queue-in
sertion au milieu-su
ppression(tete,queue, et milieu)

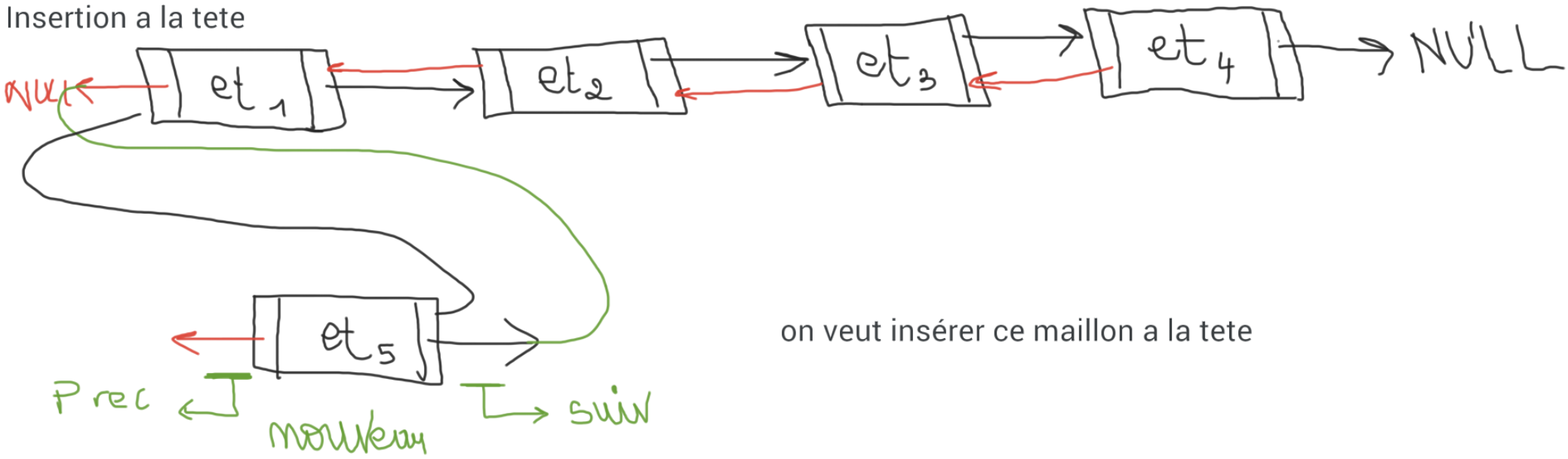
Aspect



Aspect d'un élémént



Insertion a la tete

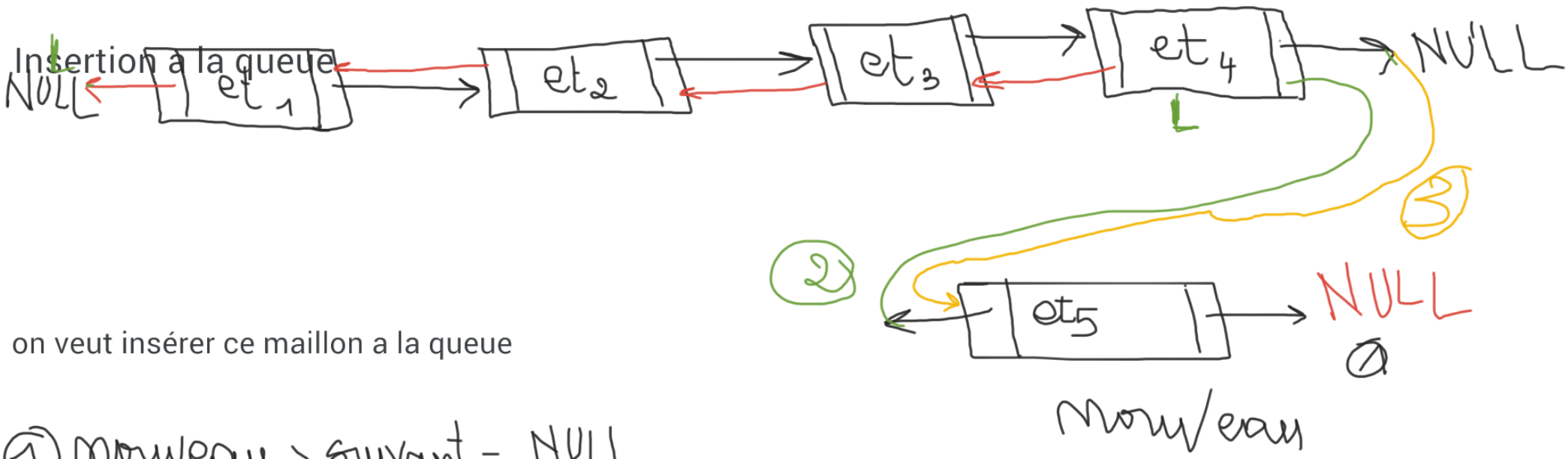


on veut insérer ce maillon a la tete

et 5 : nouveau->suivant = et 1

nouveau->prec = NULL

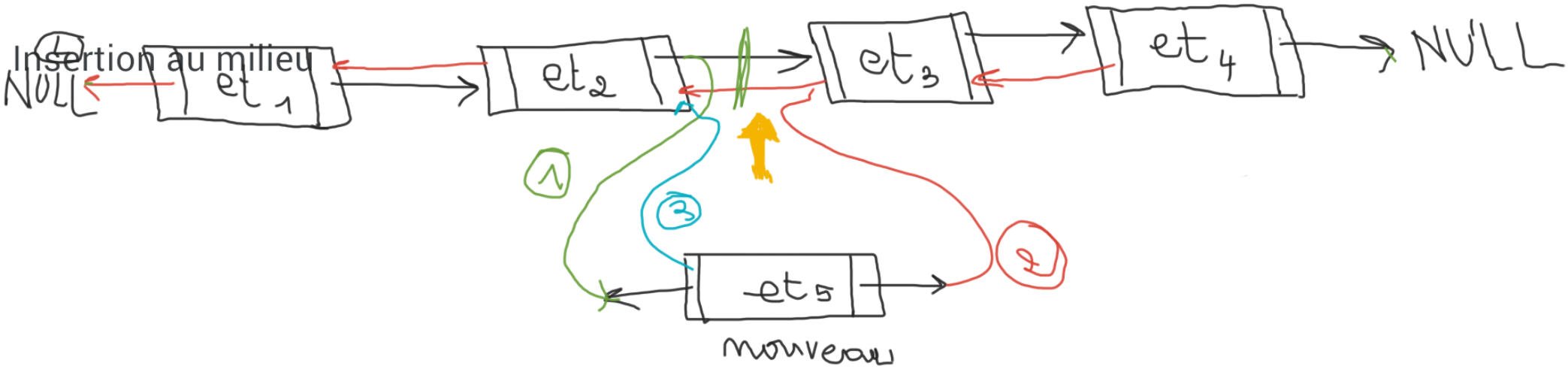
Insertion à la queue



on veut insérer ce maillon a la queue

1. nouveau → suivant = NULL
2. nouveau → prec = et4
3. L → suiv = et5

Insertion au milieu



2) nouveau → suiv = et 3

3) nouveau → prec = et 2

1) L → suiv = et 5

on veut insérer ce maillon à la milieu

DECLARATION D'UNE STRUCTURE D'UNE LISTE DOUBLEMENT CHAINEE

Exemple : Nous prenons un étudiant comme étant un élément

```
//déclaration d'un élémenttyp
edef struct etudiant{cha
r nom[50];cha
r prenom[50];}et
udiant;
```


```
// declaration de la liste
typedef struct Liste {
 etudiant et  ;
struct Liste* suivant;
struct Liste* precedant;
}liste;
```

Insertions

//Fonction qui permet d'insèrer à la tete

DÉCLARATION DE LA FONCTION QUI INSERT À LA QUEUELi

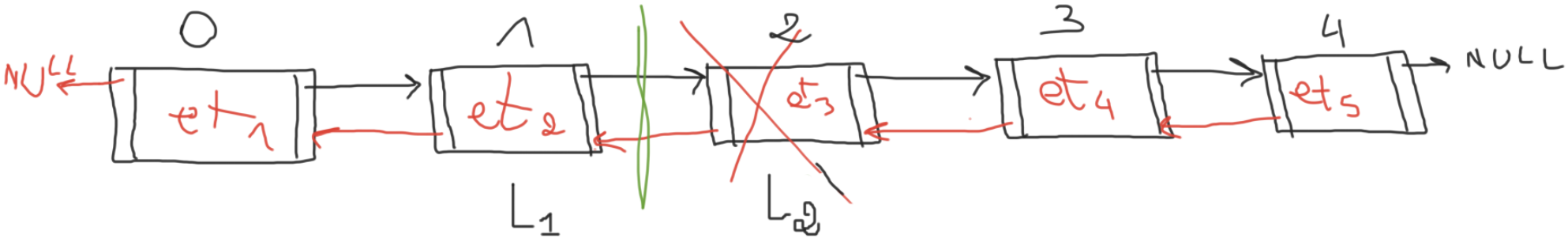
```
ste* ajouter_au_debut(Liste* tete,Etudiant NouveauEt){Li
ste*  listeTampon = tete;Li
ste* nouveau = (Liste*)malloc(sizeof(Liste));no
uveau -> et = NouveauEt;no
uveau -> suivant = listeTampon; //On dit a notre nouveau élément de pointer sur l'ancienne têtenouve
au -> precedant = NULL;liste
Tampon -> precedant = nouveau;retur

n nouveau;}
```

//Fonction qui permet d'insèrer à la queue

DÉCLARATION DE LA FONCTION QUI INSERT À LA QUEUE//

```
Fonction qui insère à la queueList
e* ajouterFin(Liste* L,Etudiant NouveauEt){List
e*  listeTampon = L;List

e* nouveau = (Liste*)malloc(sizeof(Liste));nouv
eau -> et = NouveauEt;//Pa
rcourir la liste Tamponwhil
e (listeTampon  -> suivant != NULL){list
eTampon  = listeTampon -> suivant; // Ici c'est comme si on
disait i++;}lis
t

eTampon -> suivant = nouveau;nouv
eau -> suivant = NULL;nouv
eau -> precedant = listeTampon;retu

rn listeTampon;}
```

```
DÉCLARATION DE LA FONCTION QUI INSERT AU MILIEU/
/Fonction qui insère au milieuLi
ste* ajouterMilieu(Liste* L,Etudiant NouveauEt,int position){
  Liste*  listeTampon = L;
  Liste* nouveau = (Liste*)malloc(sizeof(Liste));
  nouveau -> et = NouveauEt;
  int count = 0;

  while (count < position - 1){
  listeTampon  = listeTampon -> suivant; // Ici c'est comme si on disait i++;
  count++;
  }
  /*.  ou
  for(int i =0;i < position - 1;i++){
  listeTampon  = listeTampon -> suivant; // Ici c'est comme si on disait i++;
  }
  */
   nouveau -> suivant = listeTampon -> suivant ;
   listeTampon -> suivant  = nouveau ;
   nouveau -> precedant = listeTampon;

  return listeTampon;}
```



L'utilisateur veut suprimer le 2é element

```
L2 = L1 -> suiv;
L2 -> prec -> suiv = L2 -> suiv;   // Pour gérer le suivant
```

et2 → suiv et4

```
L2 -> succ -> prec = L2 -> prec;
free (L2);
```

Le successeur de L2 son suivant est le precedant de L2

Supressions à une position donnée

```
DÉCLARATIOQI SUPPRIME A UNE POSITION DONNEE/
/Fonction qui supprime à la pos donnéeLis

te* supprimer(Liste* L,int position){
 Liste*  L1 , L2;
 int count = 0;
  if( position ==0){ //Suppression de la tete
     L1 = L;
     L = L -> suivant;
     L -> precedant = NULL;
     free(L1);
    return L;
  }

 else{
  L1 = L;
 while (count < position - 1){
   L1 = L1 -> suivant; // Ici c'est comme si on disait i++;
   count++;
 }
 /*.  ou
 for(int i =0;i < position - 1;i++){
 listeTampon  = listeTampon -> suivant; // Ici c'est comme si on disait i++;
 }
 */

  L2 = L1-> suivant ;
     L2-> precedant -> suivant = L2-> suivant ;
  L2 -> suivant -> precedant = L2-> prec;
  free(L2);

 return L1:  }
}
```

ALGO

Liste CIRCULAIRE :

Les listes circulaires ou anneaux sont des listes linéaires dans lesquelles le dernier élémentpoi
nte sur le premier. Il n'y a donc ni premier, ni dernier. Il suffit de connaître l'adresse d'unélément po
ur parcourir tous les éléments de la liste