INFORMATIQUE
Sciences
Université Paris Cité

Langage C –Cours 6

Lélia Blin

lelia.blin@irif.fr

2023 - 2024

Lélia Blin

Parcours de chaînes de caractères
Structures de données

Recherche dans une chaîne

- On peut chercher un caractère dans une chaîne (**dans <string.h>**) :
  - `char * strchr(const char *s, int c)`
  - `char * strrchr(const char *s, int c)`
- strchr cherche la première occurence du caractère c dans la chaîne s
  - Cette fonction renvoie un pointeur vers la première position où c apparaît
  - Si c n'apparaît pas, la fonction renvoie NULL
- strrchr fait la même chose mais en renvoyant un pointeur vers la dernière position

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
  char *st="ABRA CADA BRA!";
  char *st1=strchr(st,'R');
  char *st2=strrchr(st,'R');
  char *st3=strchr(st,'M');
  printf("Premiere occurence de R : %s\n",st1);
  printf("Derniere occurence de R : %s\n",st2);
  if(!st3){
    puts("M n'est pas présent dans la chaîne!");
  }
  return EXIT_SUCCESS;
}
```

recherche-char.c

Recherche dans une chaîne

- On peut chercher une sous-chaîne dans une chaîne (**dans <string.h>**) :
  - `char * strstr(const char *haystack, const char *needle);`
- Cherche la première occurence de la chaîne needle dans la chaîne haistack
  - Cette fonction renvoie haystack si needle est la chaîne vide
  - Elle renvoie NULL si needle n'est pas la chaîne vide, mais n'apparaît pas dans haystack
  - Elle renvoie une pointeur vers la première position de needle dans haystack

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main () {
  char *st="ABRA CADA BRA!";
  char *st1=strstr(st,"");
  char *st2=strstr(st,"DA B");
  char *st3=strstr(st,"CATA");
  printf("Si le deuxieme argument est vide : %s\n",st1);
  printf("Premier occurence de DA B : %s\n",st2);
  if(!st3){
    puts("CATA n'est pas dans la chaîne");
  }
  return EXIT_SUCCESS;
}
```

recherche-chaine.c

Découper une chaîne

- On peut découper une chaîne en différentes sous-chaînes en précisant quelles sont les ‘caractères’ que l’on veut ignorer, c’est-à-dire les délimiteurs
- Pour cela on a la fonction (**dans <string.h>**) :
  - `char *strtok(char *restrict str, const char *restrict sep);`
- Le premier argument str contient la chaîne que l’on veut découper
- Le deuxième contient les séparateurs
- **Attention :** cette fonction modifier la chaîne str (il faut donc mieux copier au préalable la chaîne à découper)
- Au premier appel, on donne la chaîne à parser et ensuite on met NULL
  - Le programme se rappelle quelle chaîne il est en train de parser !!!
- À chaque appel, la fonction renvoie un pointeur vers une chaîne commençant après le prochain délimiteur et met ‘\0’ à la fin du morceau de chaîne lorsque qu’elle rencontre un délimiteur
- Au dernier appel, quand on arrive au bout de la chaîne à parcourir, la fonction renvoie NULL

Exemple

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>

int main () {
  char *st="--- Les enfants-du+paradis -Tres bon film!";
  char *temp=malloc((strlen(st)+1)*sizeof(char));
  assert(temp!=NULL);
  strcpy(temp,st);
  char *lex=strtok(temp,"-  +");
  while(lex!=NULL){
    printf("%s\n",lex);
    lex=strtok(NULL,"-  +");
  }
  return EXIT_SUCCESS;
}
```

parsing-chaine.c

Fonctionnement de strtok

- Intuition de comment marche strtok (pas forcément implémenté de cette façon)

```
char *s="---He--llo--";
char *tmp=malloc((strlen(s)+1)*sizeof(char));
strcpy(tmp,s);
char *lex=strtok(tmp,"-"); // on veut chercher les mots entre les -
while(lex!=NULL){
  lex=strotok(NULL,"-");
}
```

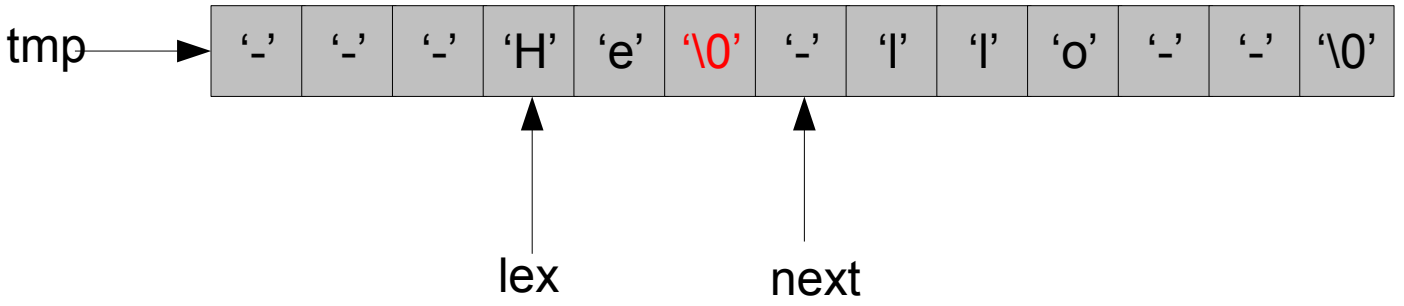
- 

- Le pointeur next indique où commencer la prochaine recherche.

Fonctionnement de strtok

- Intuition de comment marche strtok (pas forcément implémenté de cette façon)

```
char *s="---He--llo--";
char *tmp=malloc((strlen(s)+1)*sizeof(char));
strcpy(tmp,s);
char *lex=strtok(tmp,"-"); // on veut chercher les mots entre les -
while(lex!=NULL){
  lex=strotok(NULL,"-");
}
```

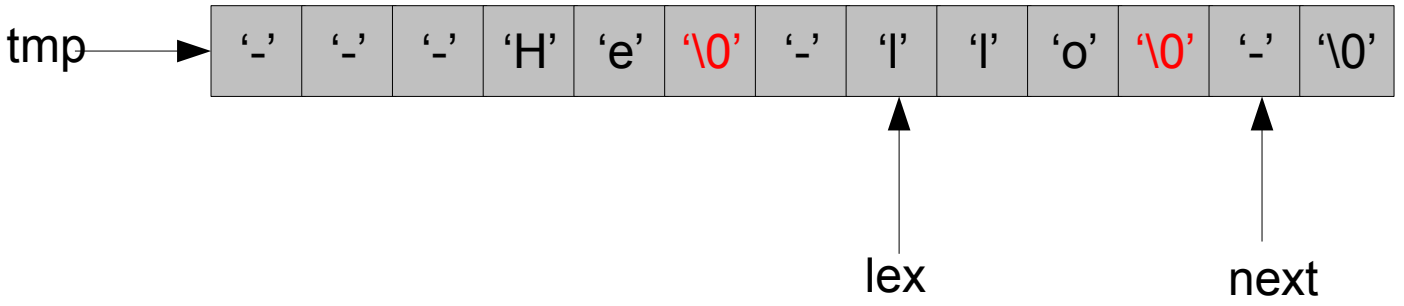
- 

- Le pointeur next indique où commencer la prochaine recherche (variable globale utilisée par strtok...elle ne s'appelle pas forcément comme cela).

Fonctionnement de strtok

- Intuition de comment marche strtok (pas forcément implémenté de cette façon)

```
char *s="---He--llo--";
char *tmp=malloc((strlen(s)+1)*sizeof(char));
strcpy(tmp,s);
char *lex=strtok(tmp,"-"); // on veut chercher les mots entre les -
while(lex!=NULL){
  lex=strotok(NULL,"-");
}
```

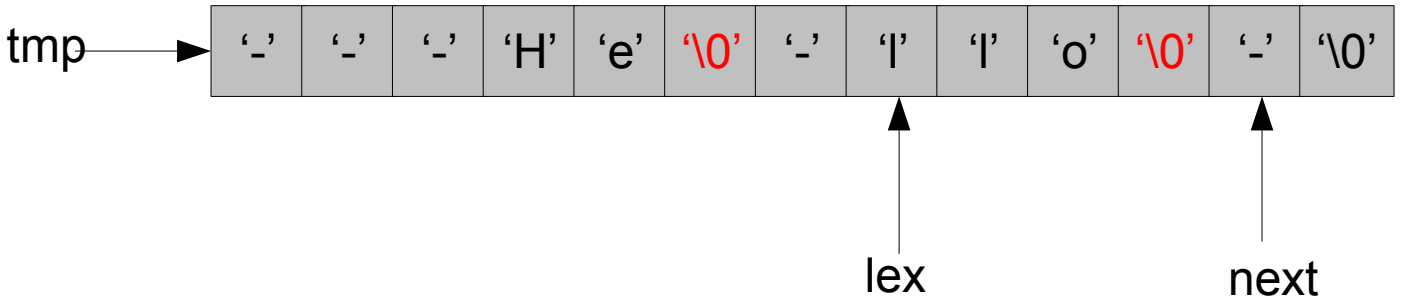
- 

- Le pointeur next indique où commencer la prochaine recherche (variable globale utilisée par strtok...elle ne s'appelle pas forcément comme cela).

Fonctionnement de strtok

- Intuition de comment marche strtok (pas forcément implémenté de cette façon)

```
char *s="---He--llo--";
char *tmp=malloc((strlen(s)+1)*sizeof(char));
strcpy(tmp,s);
char *lex=strtok(tmp,"-"); // on veut chercher les mots entre les -
while(lex!=NULL){
  lex=strotok(NULL,"-");
}
```

- 

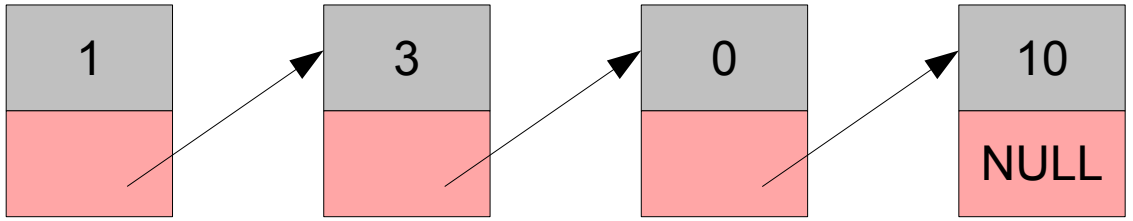
- **Remarque : si vous pouvez éviter l'utilisation de cette fonction, et programmer un découpage vous même c'est mieux**

Structures de données

- Avec les structures et l'allocation dynamique, on peut reprogrammer les structures de données à taille variable classique :
  - Listes simplement chaînées
  - Liste doublement chaînées
  - Liste de listes
  - Arbres binaires
  - etc
- Méthodes :
  - Définir une structure correspondante
  - Définir les fonctions de manipulation de la structure
  - Faire attention à comment est alloué/libéré la mémoire

Listes simplement chaînées

- Liste d'éléments, chaque élément ayant une donnée et un unique successeur



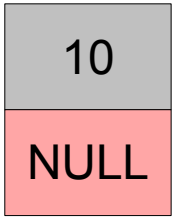
- On peut représenter une telle liste par la structure suivante (si les données sont des entiers signés) :

```
struct noeud{
  int val;
  struct noeud *succ;
};

typedef struct noeud noeud;
```

Listes simplement chaînées

Création d'un noeud simple

- Création d'un noeud simple avec une valeur v dedans

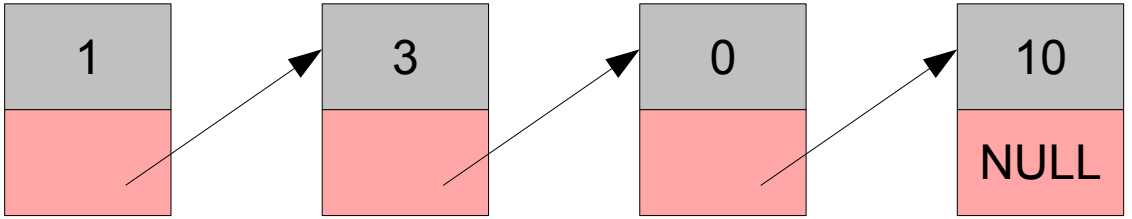



```
struct noeud{
  int val;
  struct noeud *succ;
};

typedef struct noeud noeud;

noeud *creation_noeud(int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  return n;
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

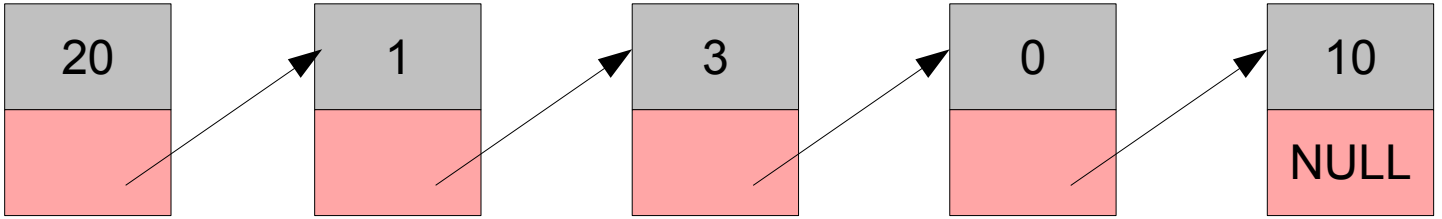



```
struct noeud{
  int val;
  struct noeud *succ;
};

typedef struct noeud noeud;

noeud *insertion_tete(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=li;
  return n;
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

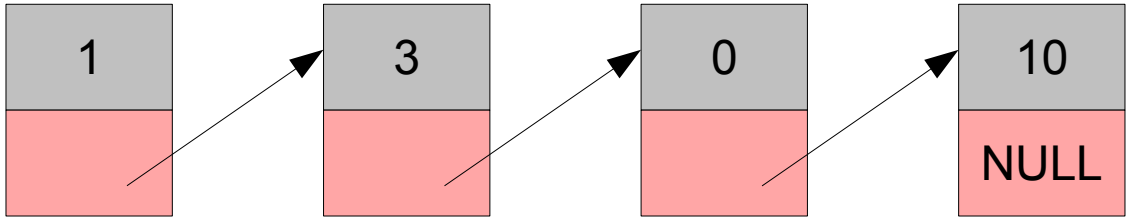



```
struct noeud{
  int val;
  struct noeud *succ;
};

typedef struct noeud noeud;

noeud *insertion_tete(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=li;
  return n;
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

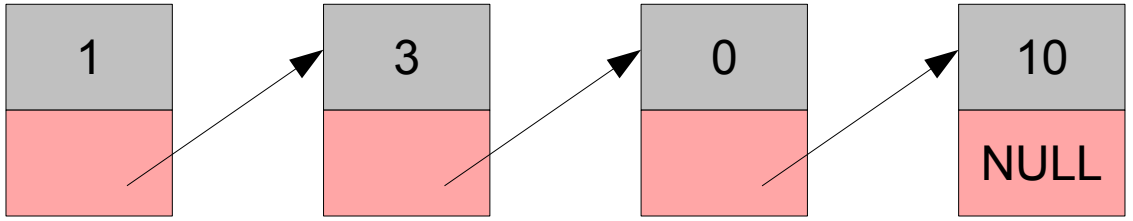



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

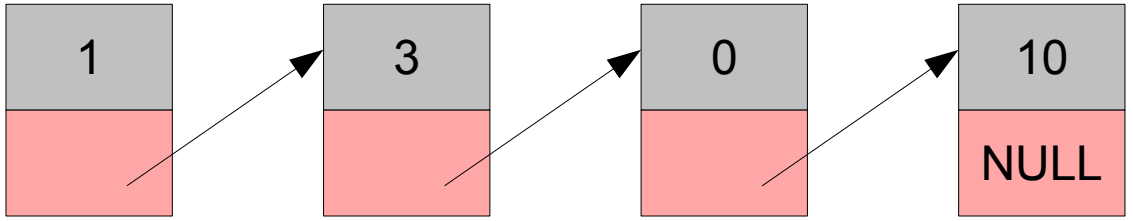



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

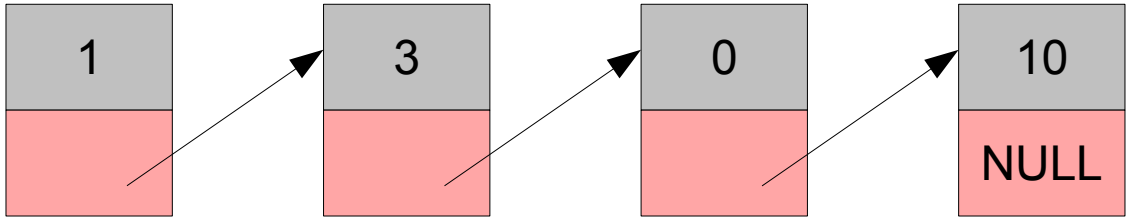



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

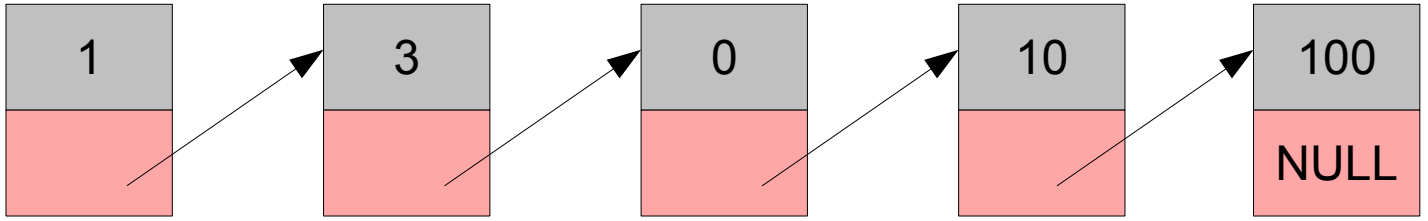



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

- On peut insérer un noeud ou une valeur soit en tête, soit en queue de liste

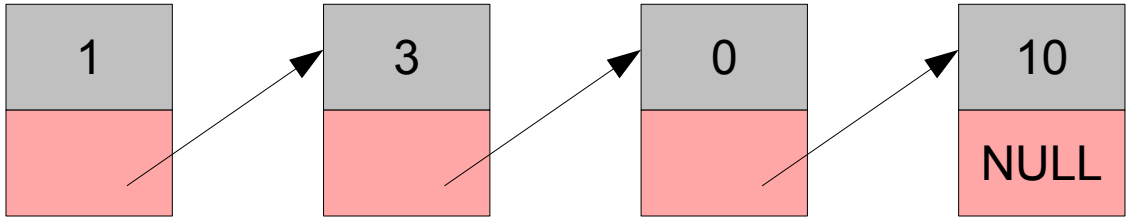



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Affichage des éléments d'une liste

- On parcourt les éléments d'une liste pour les afficher

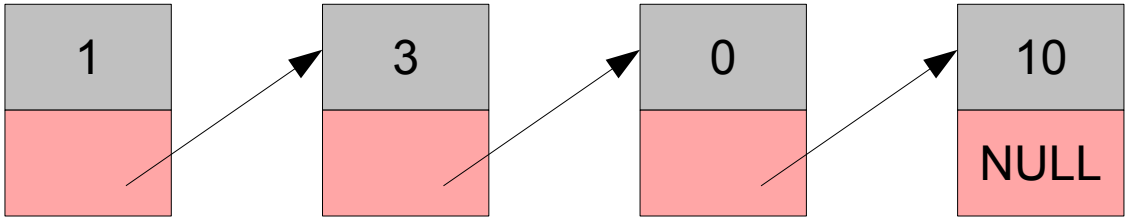



```
struct noeud{
  int val;
  struct noeud *succ;
};

typedef struct noeud noeud;

void print_list(noeud *li){
  while(li!=NULL){
   printf("%d->",li->val);
   li=li->succ;
  }
  printf("NULL\n");
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de tête (**ne pas oublier de libérer la mémoire!!**)

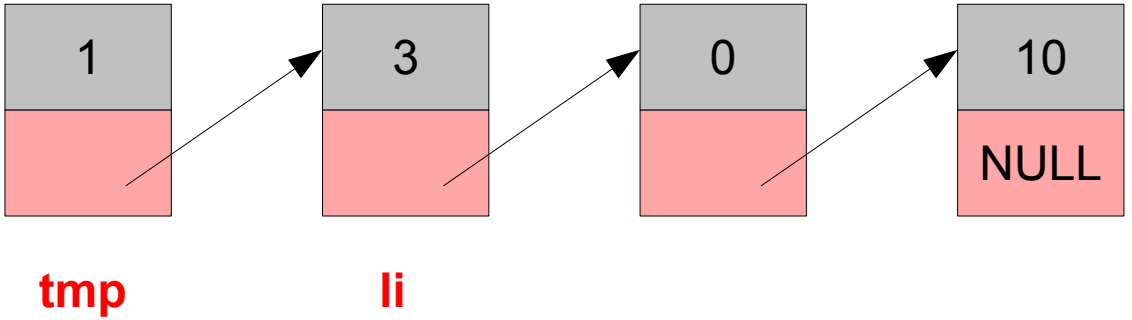



```
noeud *efface_tete(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    noeud *tmp=li;
    li=li->succ;
    free(tmp);
    return li;
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de tête (**ne pas oublier de libérer la mémoire!!**)

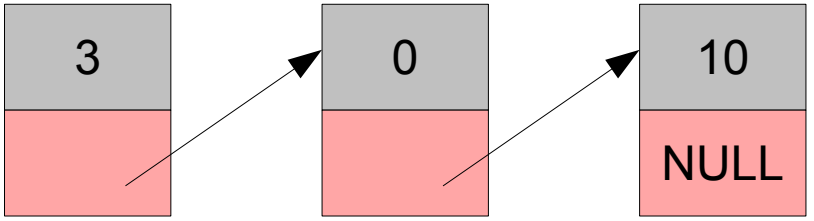



```
noeud *efface_tete(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    noeud *tmp=li;
    li=li->succ;
    free(tmp);
    return li;
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de tête (**ne pas oublier de libérer la mémoire!!**)

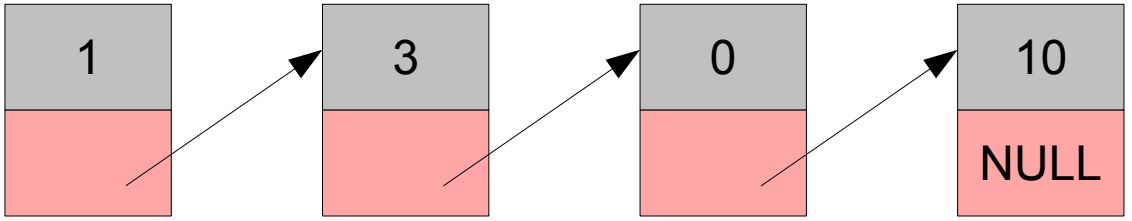



```
noeud *efface_tete(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    noeud *tmp=li;
    li=li->succ;
    free(tmp);
    return li;
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de queue (**ne pas oublier de libérer la mémoire!!)**

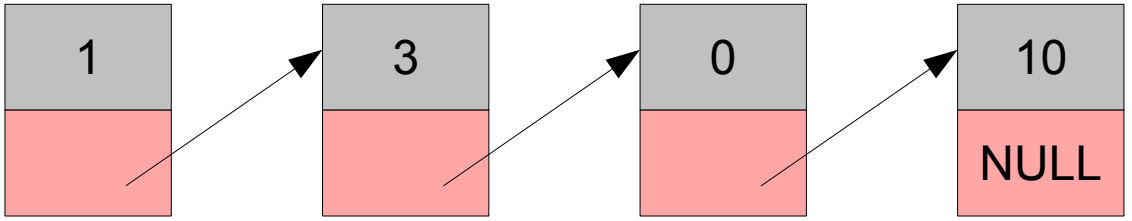



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
       }
       pred->succ=NULL;
       free(courant);
       return li;
    }
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de queue (**ne pas oublier de libérer la mémoire!!)**

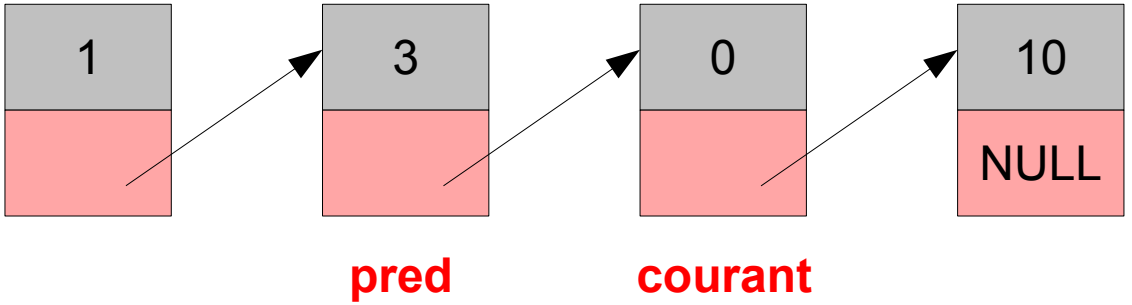



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
       }
       pred->succ=NULL;
       free(courant);
       return li;
    }
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de queue (**ne pas oublier de libérer la mémoire!!)**

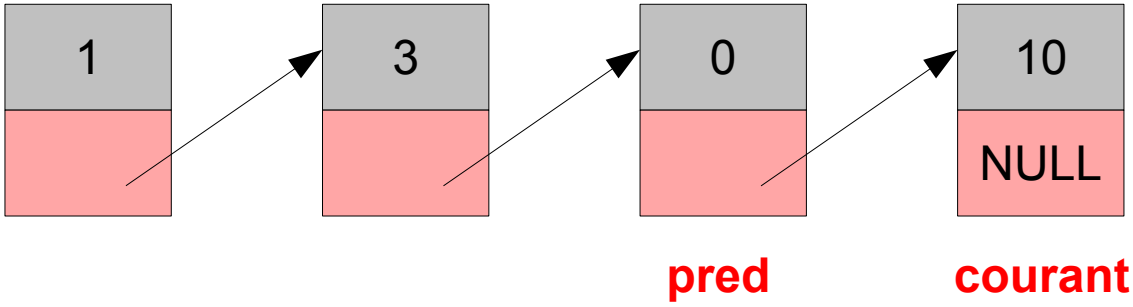



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
       }
       pred->succ=NULL;
       free(courant);
       return li;
    }
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

- On peut effacer l'élément de queue (**ne pas oublier de libérer la mémoire!!)**

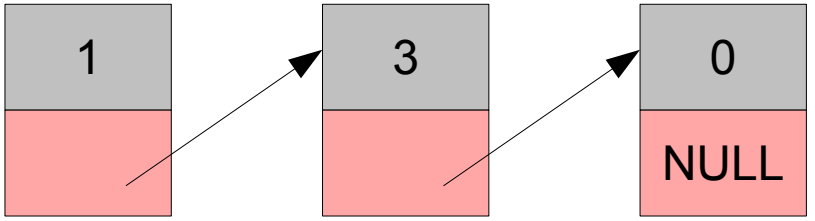



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
       }
       pred->succ=NULL;
       free(courant);
       return li;
    }
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

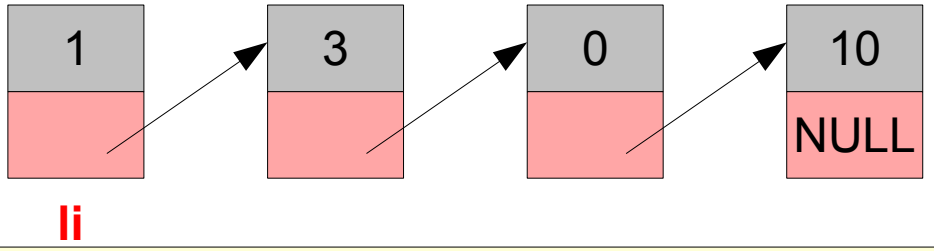
- On peut effacer l'élément de queue (**ne pas oublier de libérer la mémoire!!)**




```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
       }
       pred->succ=NULL;
       free(courant);
       return li;
    }
  }
}
```

Listes simplement chaînées

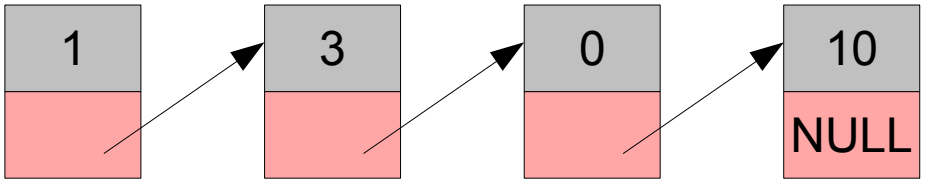
- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!)**




```
noeud *efface_val(noeud *li,int v){
  if(li==NULL){
    return NULL;
  }else{
    if(li->val==v){
      noeud *tmp=li->succ;
      free(li);
      return tmp;
    }else{
      noeud *pred=NULL;
      noeud *courant=li;
      noeud *suivant=li->succ;
      while(courant->val!=v && suivant!=NULL){
        pred=courant;
        courant=suivant;
        suivant=suivant->succ;
      }
      if(courant->val==v){
        pred->succ=suivant;
        free(courant);
      }
      return li;
    }
  }
}
```

Listes simplement chaînées

- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!) -> par exemple 0**

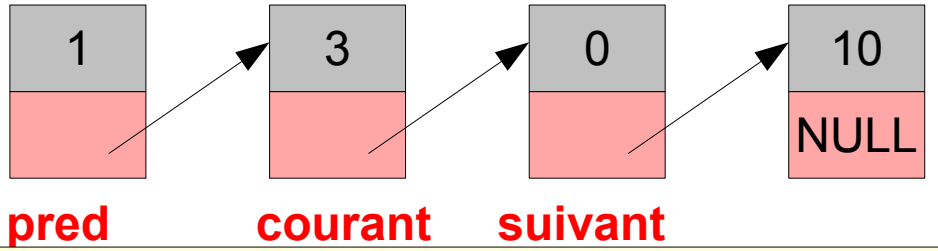


pred **courant** **suivant**

```
noeud *efface_val(noeud *li,int v){
  if(li==NULL){
    return NULL;
  }else{
    if(li->val==v){
      noeud *tmp=li->succ;
      free(li);
      return tmp;
    }else{
      noeud *pred=NULL;
      noeud *courant=li;
      noeud *suivant=li->succ;
      while(courant->val!=v && suivant!=NULL){
        pred=courant;
        courant=suivant;
        suivant=suivant->succ;
      }
      if(courant->val==v){
        pred->succ=suivant;
        free(courant);
      }
      return li;
    }
  }
}
```

Listes simplement chaînées

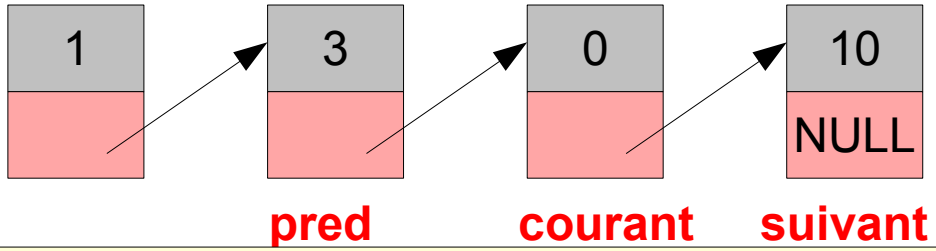
- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!) -> par exemple 0**




```
noeud *efface_val(noeud *li,int v){
  if(li==NULL){
    return NULL;
  }else{
    if(li->val==v){
      noeud *tmp=li->succ;
      free(li);
      return tmp;
    }else{
      noeud *pred=NULL;
      noeud *courant=li;
      noeud *suivant=li->succ;
      while(courant->val!=v && suivant!=NULL){
        pred=courant;
        courant=suivant;
        suivant=suivant->succ;
      }
      if(courant->val==v){
        pred->succ=suivant;
        free(courant);
      }
      return li;
    }
  }
}
```

Listes simplement chaînées

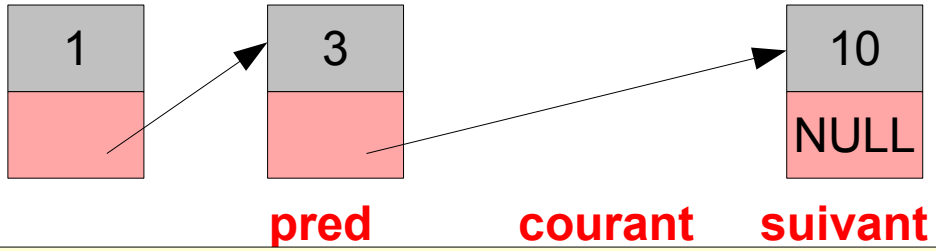
- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!) -> par exemple 0**




```
noeud *efface_val(noeud *li,int v){
  if(li==NULL){
    return NULL;
  }else{
    if(li->val==v){
      noeud *tmp=li->succ;
      free(li);
      return tmp;
    }else{
      noeud *pred=NULL;
      noeud *courant=li;
      noeud *suivant=li->succ;
      while(courant->val!=v && suivant!=NULL){
        pred=courant;
        courant=suivant;
        suivant=suivant->succ;
      }
      if(courant->val==v){
        pred->succ=suivant;
        free(courant);
      }
      return li;
    }
  }
}
```

Listes simplement chaînées

- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!) -> par exemple 0**

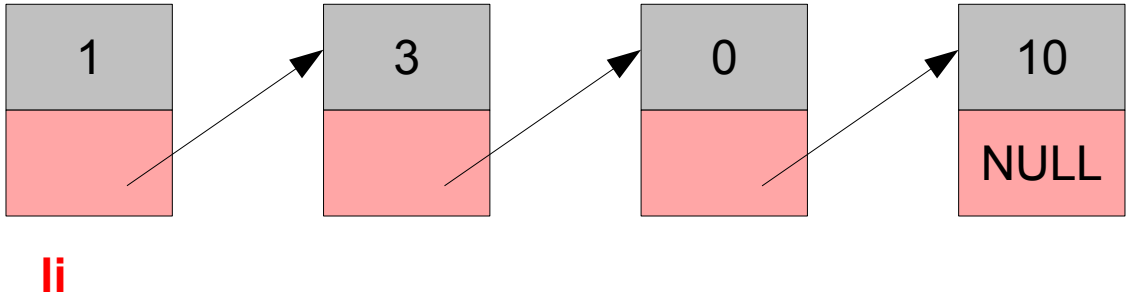



```
noeud *efface_val(noeud *li,int v){
  if(li==NULL){
    return NULL;
  }else{
    if(li->val==v){
      noeud *tmp=li->succ;
      free(li);
      return tmp;
    }else{
      noeud *pred=NULL;
      noeud *courant=li;
      noeud *suivant=li->succ;
      while(courant->val!=v && suivant!=NULL){
        pred=courant;
        courant=suivant;
        suivant=suivant->succ;
      }
      if(courant->val==v){
        pred->succ=suivant;
        free(courant);
      }
      return li;
    }
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

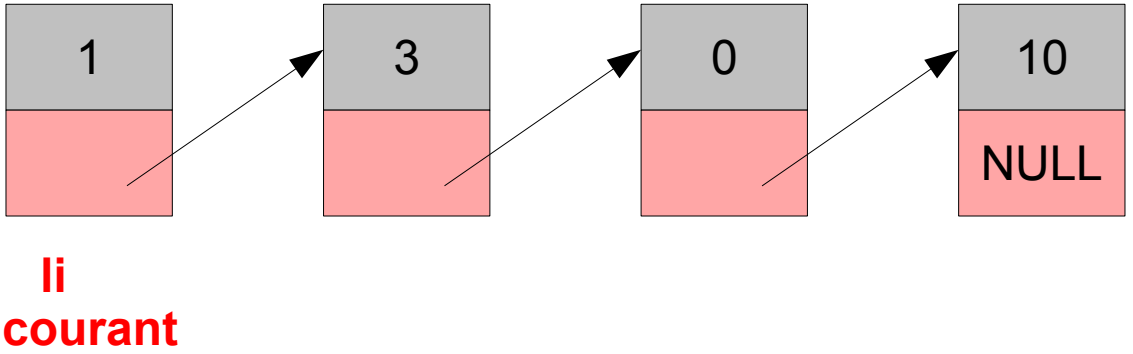



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

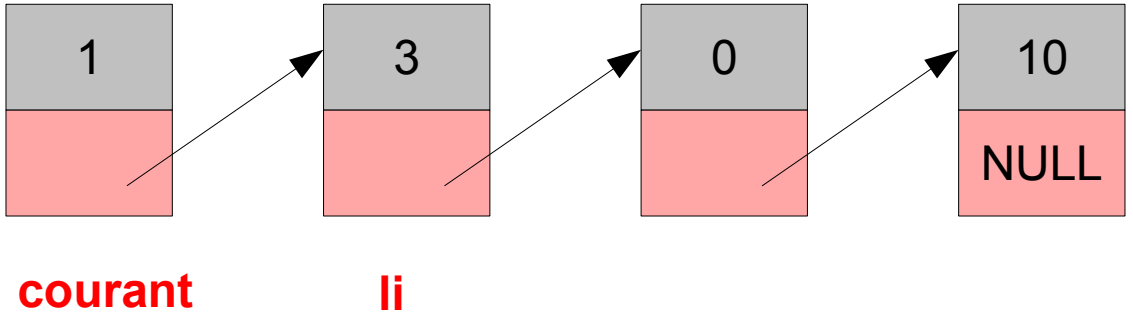



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

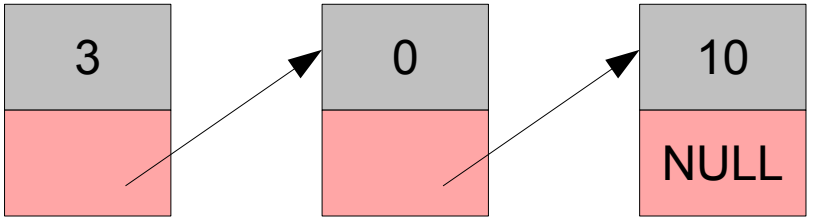



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



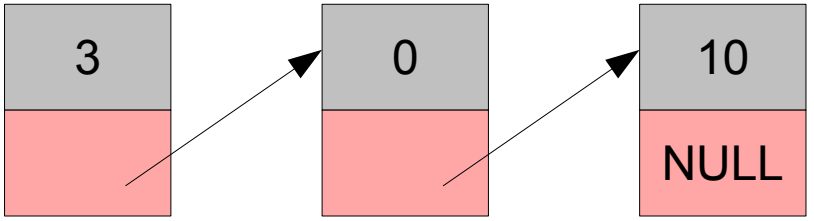
courant **li**

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



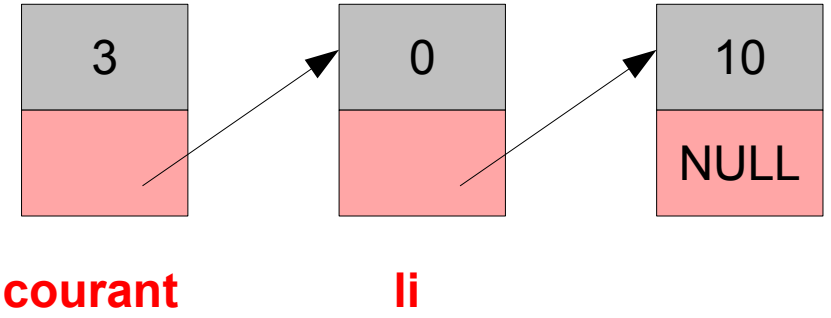
li
courant

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

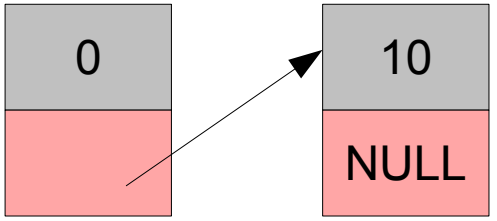



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

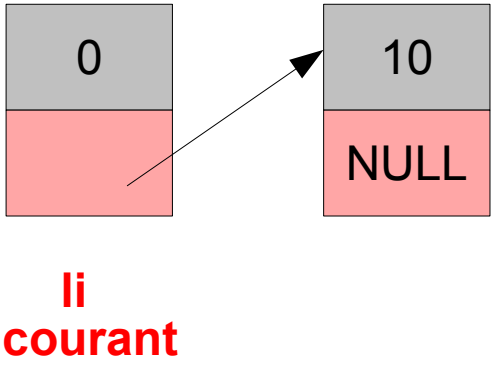



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

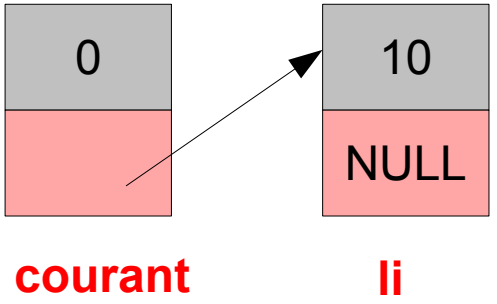



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

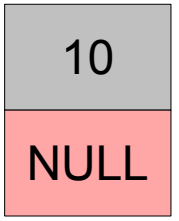



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



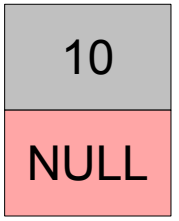
courant **li**

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



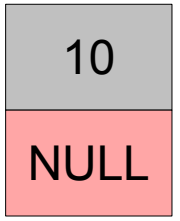
li
courant

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



courant **li**

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

courant **li**

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

li

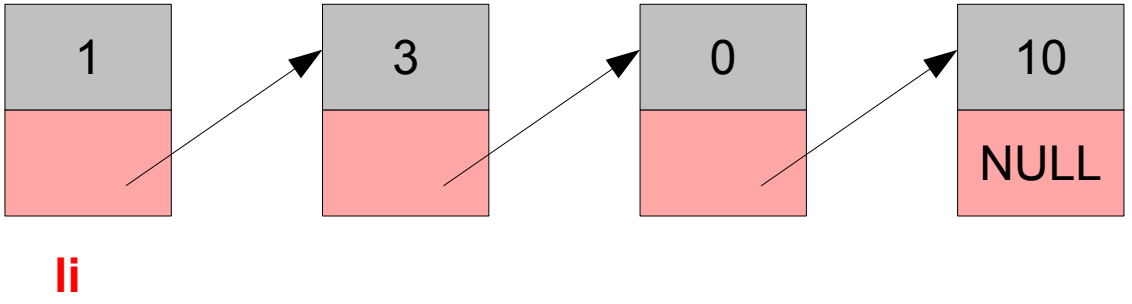
courant

```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

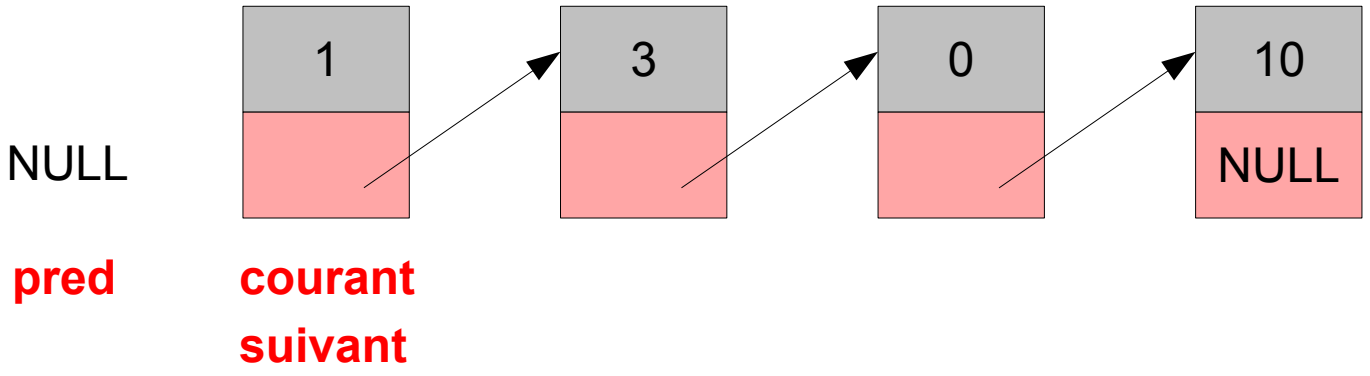



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

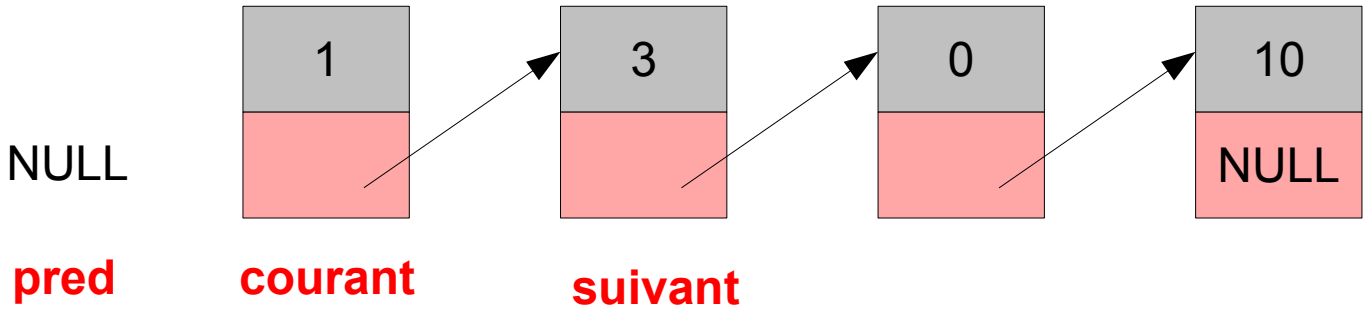



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

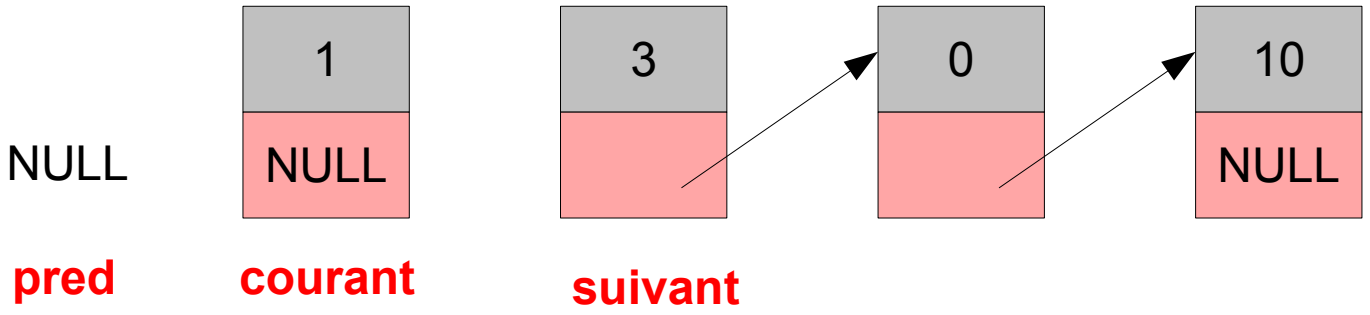



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

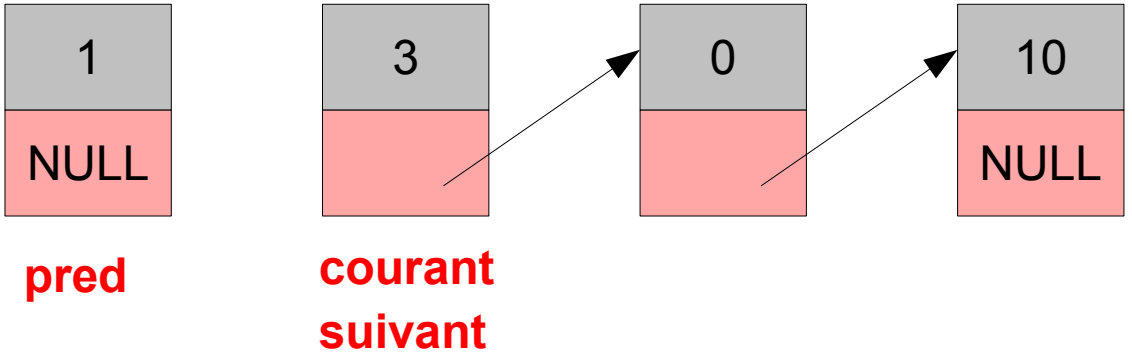



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

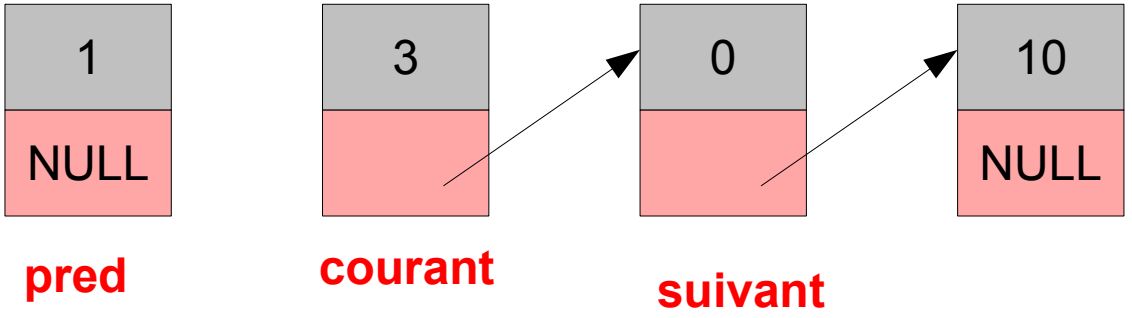



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

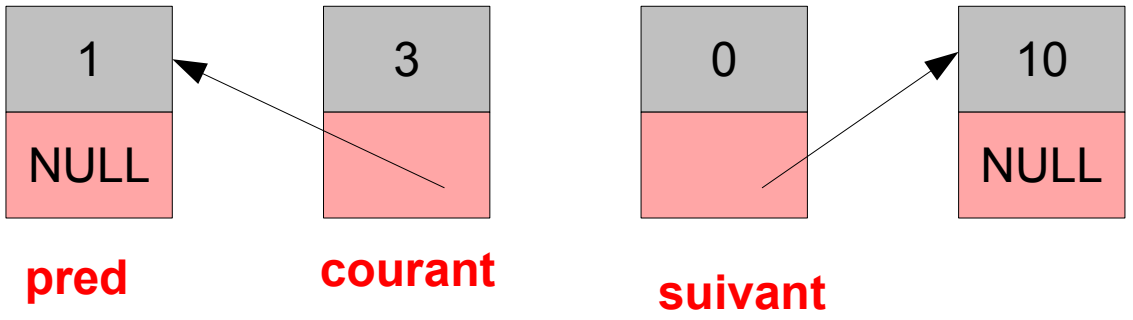



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

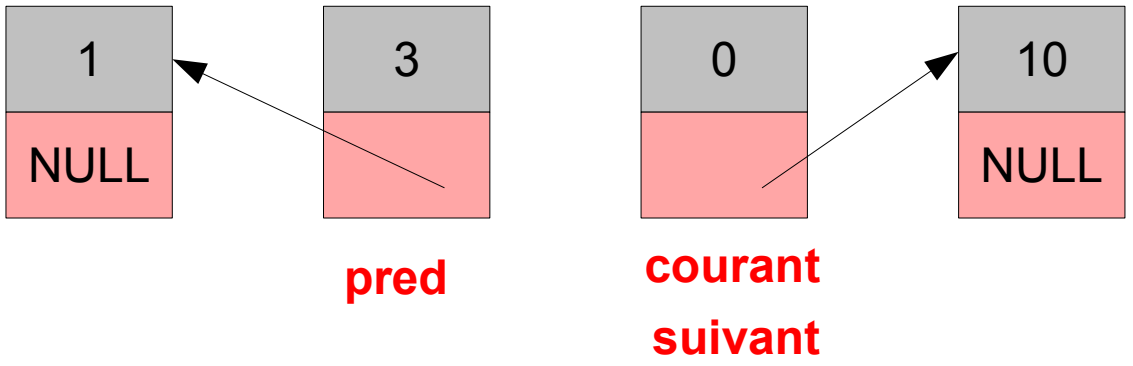



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

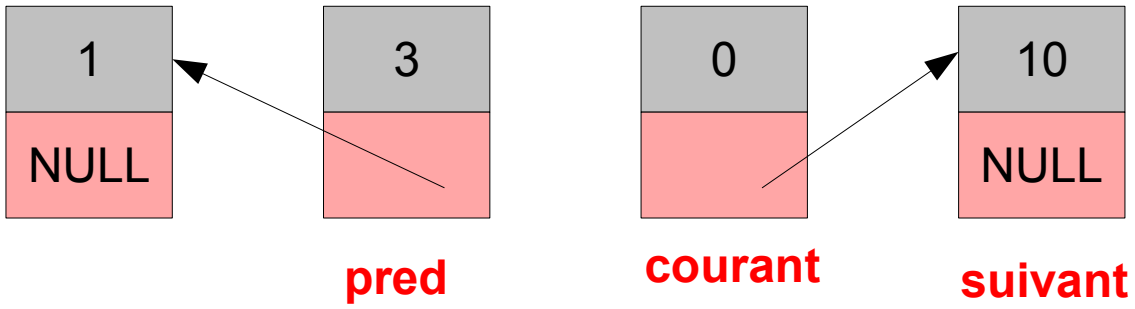



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

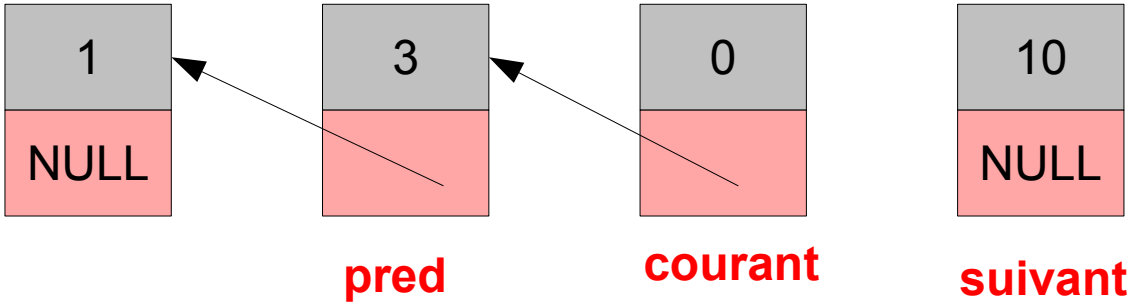



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

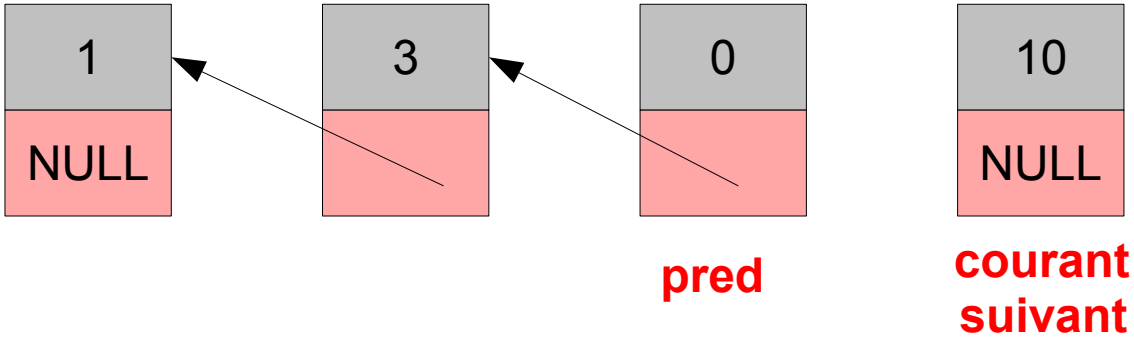



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

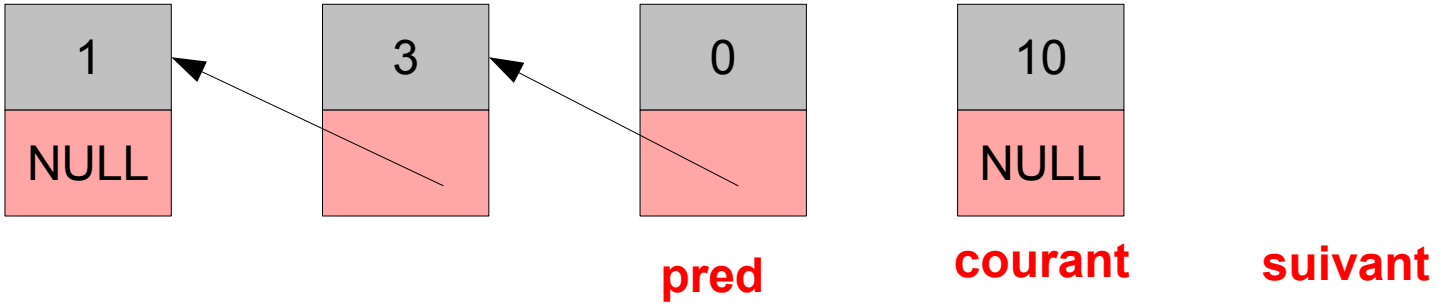



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

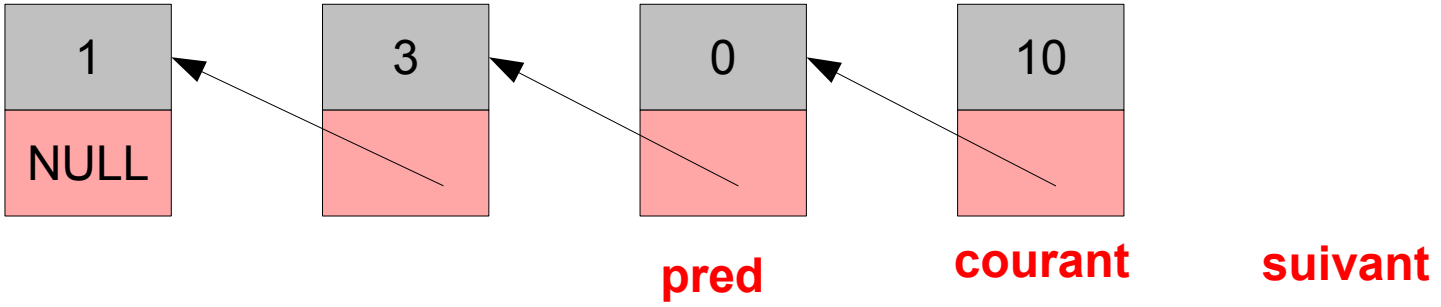



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste

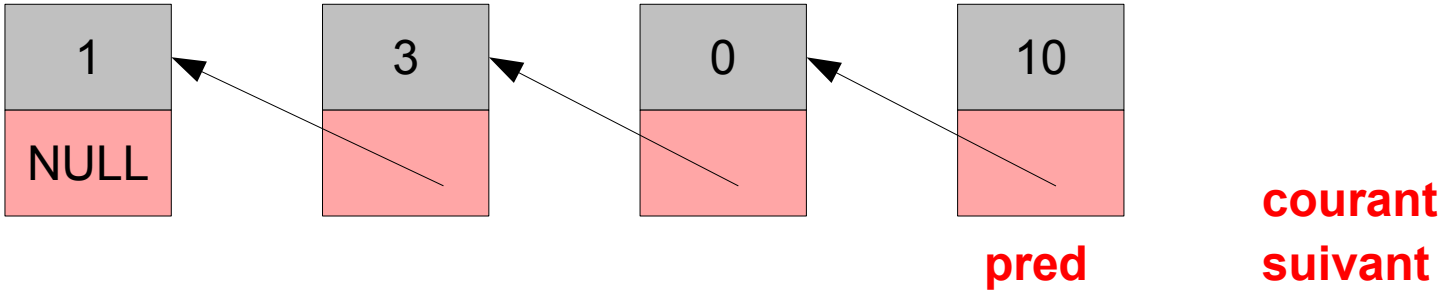



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

- On veut changer l'ordre des éléments d'une liste




```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```