Language C

Matej Stehlik
matej@irif.fr

avec des contributions de Iyan Nazarian

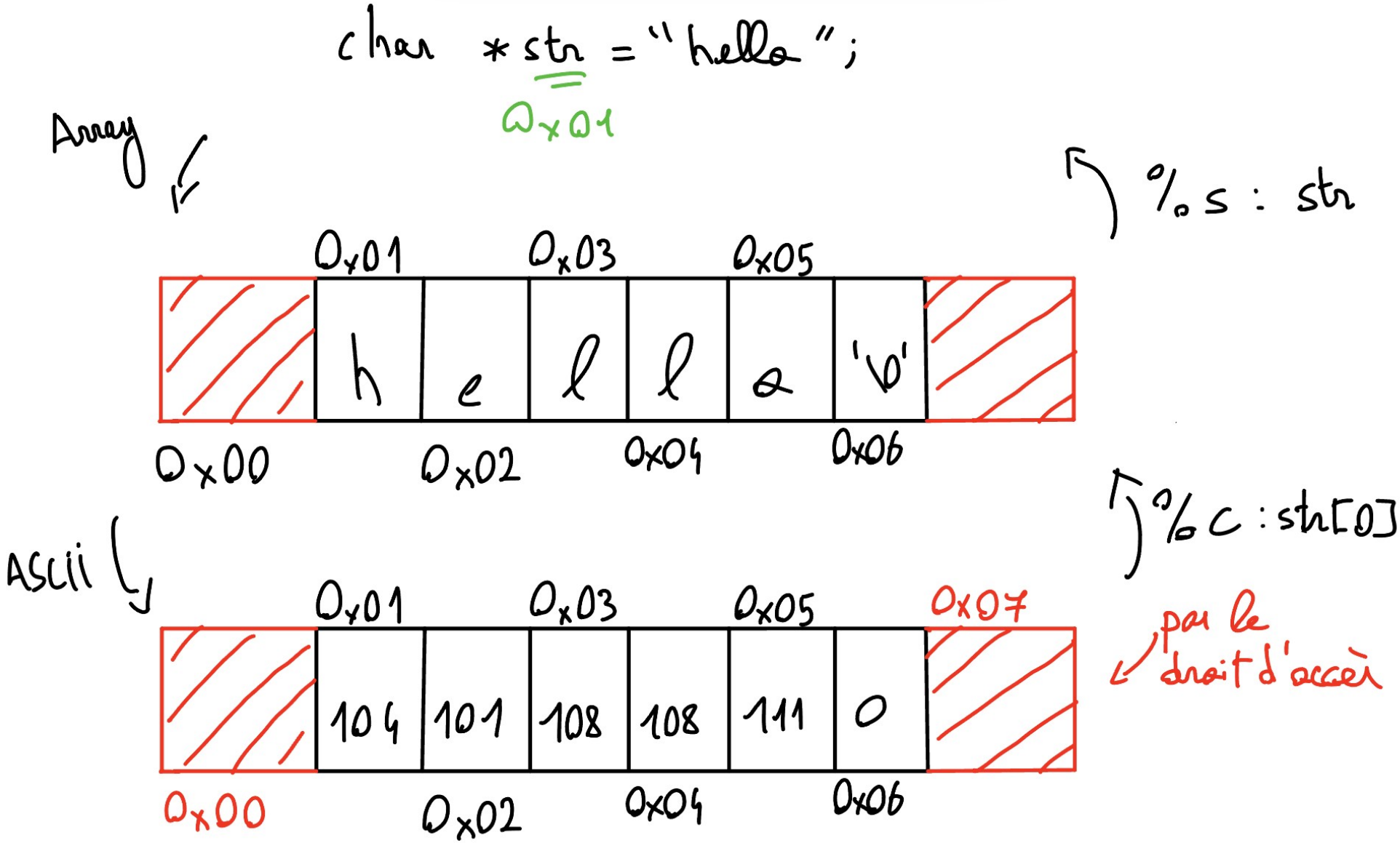
Parcours de chaînes de caractères & manipulations de structures de données

Petit rappel sur les strings en C

- D’autres langages comme Python ou Java possèdent un type String, mais pas le C

- en C, un string est une suite (array) de **caractères individuels** qui se termine par un caractère spécial : ‘\0’ (null terminating character)

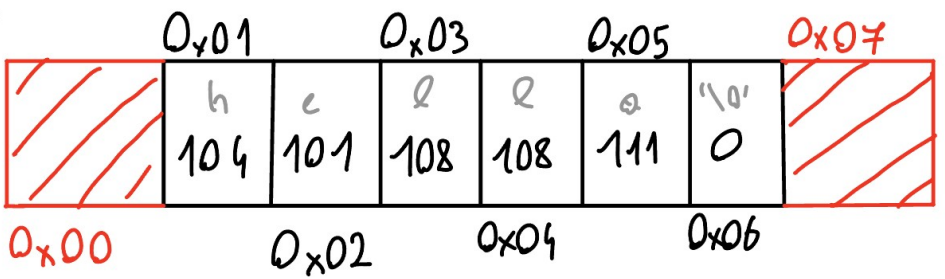
- Voilà un exemple afin de mieux comprendre :


char *str = "hello";
0x01
Array
%s : str
0x01
0x03
0x05
h
e
l
l
o
'\0'
0x00
0x02
0x04
0x06
%c : str[0]
ASCII
0x01
0x03
0x05
0x07
par le droit d'accès
104
101
108
108
111
0
0x00
0x02
0x04
0x06

```
char *str = "hello";
```

0x01

ASCII



par le droit d'accès

char : 1 byte $\in [\![0 ; 127]\!]$

5 caractères manuel

Quelle est la longueur (length) de ce string : 5

Quelle est la taille (size) de ce string : 6

nombre de bytes disponible

Comment traverser des strings

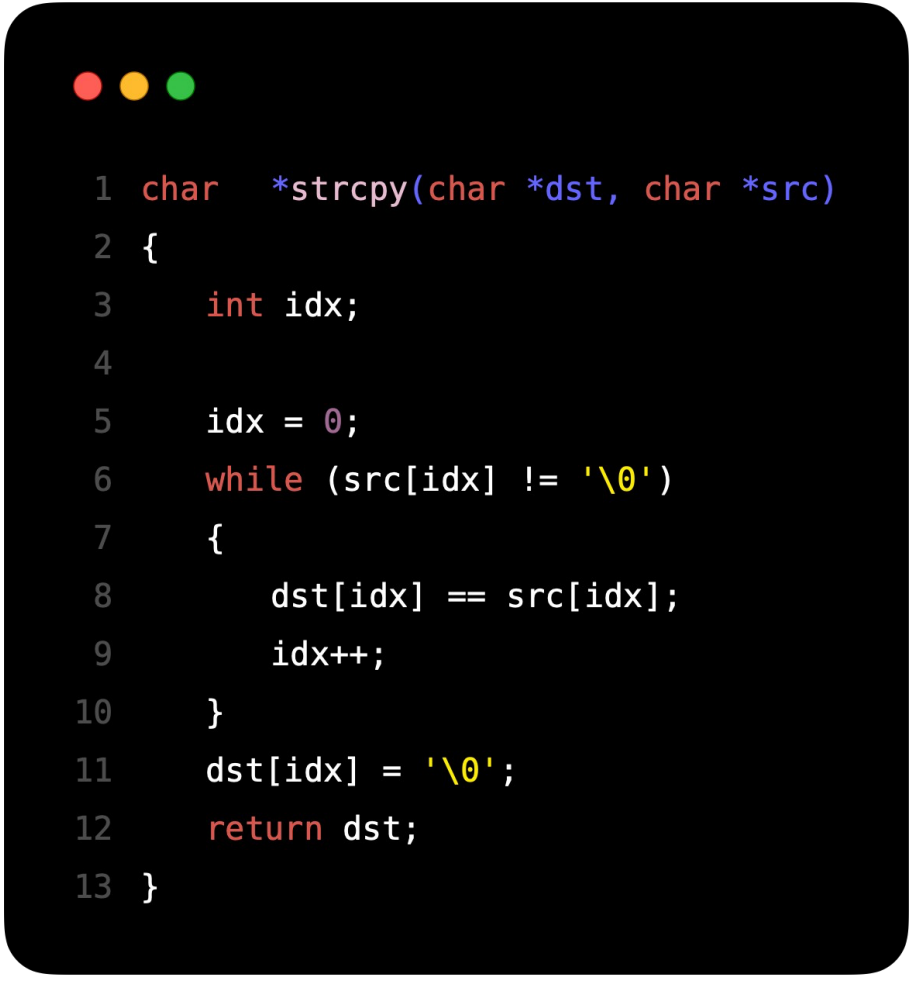
- Quasiment tout le temps vous allez devoir traverser des strings, que ce soit pour copier son contenu, le modifier, etc.
- Nous allons essayer de recréer une fonction très utile : **strcpy()** qui a pour prototype (signature) :

```
char   *strcpy(char *dst, char *src)
```

Comment fonctionne-t-elle ?

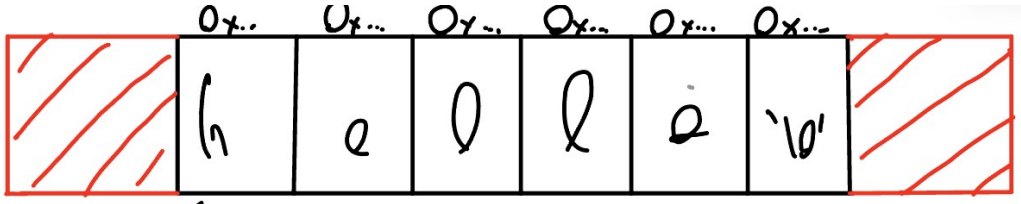
```
char   *strcpy(char *dst, char *src)
```

- Elle part du principe que l'on peut écrire dans *dst* (que ce n'est pas en **read-only**) et qu'on ait la place de copier *src*
- Elle itère simultanément sur *src* et *dst* et assigne les valeurs consécutifs de la 1ere dans la 2eme, sans se préoccuper des problèmes de taille.
- Elle ajoute également un '\0' dans *dst* une fois que *src* a été traversé



```c
char   *strcpy(char *dst, char *src)
{
    int idx;

    idx = 0;
    while (src[idx] != '\0')
    {
        dst[idx] == src[idx];
        idx++;
    }
    dst[idx] = '\0';
    return dst;
}
```

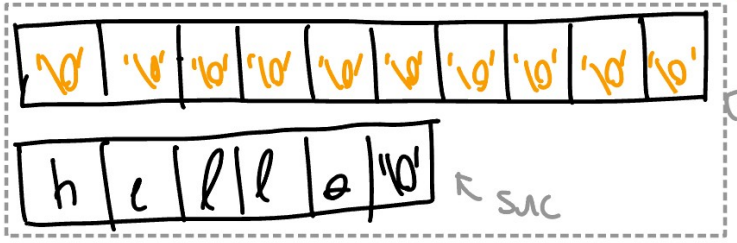
1 char *src ;



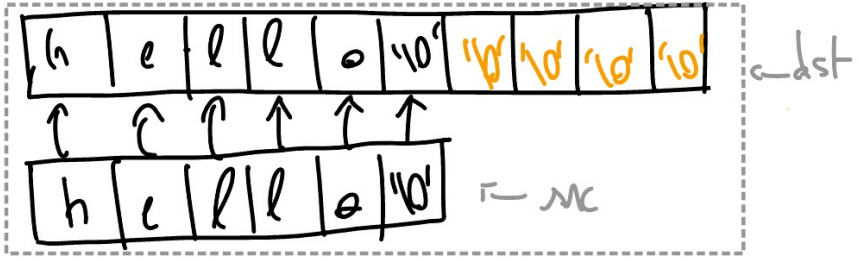
= 1, c'est plus pour le concept

2 char *dst = calloc (10 * sizeof (char));

demande 10 bytes contigus de mémoire initialisé à 0



3 strcpy (dst, src);



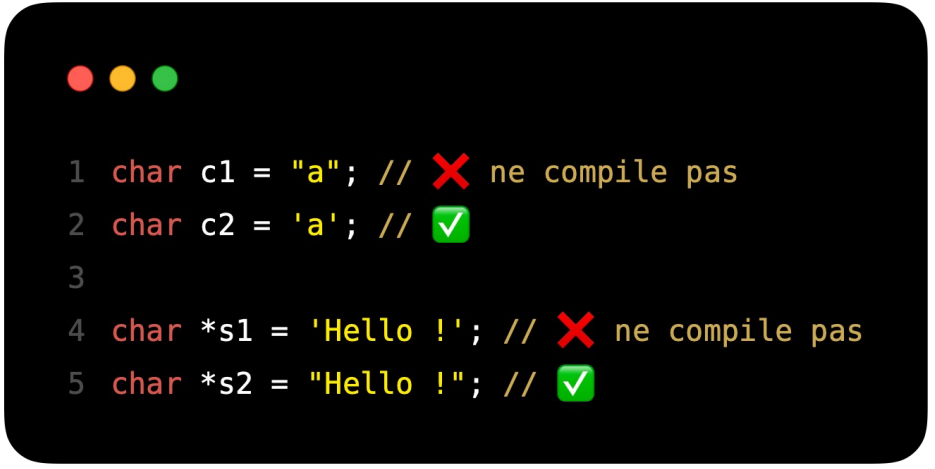
Une fonction parmi d'autres

- De nombreuses fonctions qui permettent de manipuler des strings sont disponible a travers des librairies : **string.h**, **stdio.h**, **stdlib.h**, …
- Par ex : int strlen(char *str) – renvoie la **longueur** de str
- char *strstr(char *s1, char *s2) – renvoie l'adresse de la 1ère occurrence **totale** de s2 dans s1, NULL sinon
- char *strchr(char *s2, char c) – renvoie l'adresse de la 1ère occurrence de c dans s2, NULL sinon
- Etc...

Une distinction importante

Contrairement au Python où l'on peut utiliser de façon interchangeable le single quote (') et le double quote ("), le C ne le permet pas.

- Le single quote est réservé pour les **caractères**
- Le double quote est réservé pour les **strings** :



```
~/Desktop
❯ cat t.c
int main(void)
{
        char c1 = "a"; // ❌ ne compile pas
        char c2 = 'a'; // ✅

        char *s1 = 'Hello !'; // ❌ ne compile pas
        char *s2 = "Hello !"; // ✅
}

~/Desktop
❯ cc t.c
t.c:3:7: error: incompatible pointer to integer conversion initializing 'char' with an expression of type 'char[2]' [-Wint-conversio
    3 |         char c1 = "a"; // ❌ ne compile pas
      |              ^    ~~~
t.c:6:13: warning: multi-character character constant [-Wmultichar]
    6 |         char *s1 = 'Hello !'; // ❌ ne compile pas
      |                    ^
t.c:6:13: warning: character constant too long for its type
t.c:6:8: error: incompatible integer to pointer conversion initializing 'char *' with an expression of type 'int' [-Wint-conversion]
    6 |         char *s1 = 'Hello !'; // ❌ ne compile pas
      |              ^    ~~~~~~~~~
2 warnings and 2 errors generated.
```

Parenthèse sur « NULL »

- « NULL », est l'équivalent du **caractère** '\0', mais pour les adresses !
- C'est une macro qui vaut : « (void *) 0 »
- Elle fait référence a l'adresse 0, et est utilisée comme valeur « d'erreur », par exemple quand on appelle malloc mais on n'a plus de mémoire, NULL doit être renvoyée par malloc
- Une fonction qui renvoie char *, int *, …, peut renvoyer NULL, vu que NULL est une adresse : (void *) 0

Pardon ?? « void * » ??

- On sait qu'en C, **void** veut juste dire .. rien, littéralement.
- Quand une fonction ne renvoie rien.. elle renvoie void !
- Quand une fonction ne prend rien en paramètre.. elle prend void !

On a vu que un « int * », est un pointeur vers un entier, un « char * » est un pointeur vers le 1er caractère d'un string, etc.. mais « void * » pointe sur void ? Loin de là.

Pardon ?? « void * » ??

- Quand on parle de « void * » il faut penser au terme void comme voulant dire « un pointeur qui pointe sur quelque chose ».

- Ce quelque chose en soi contient des informations qui sont libres d'être utilisées et formatées comme l'utilisateur le souhaite.

- Par exemple, malloc renvoie un void *, et laisse à l’utilisateur le choix d’utiliser cette mémoire comme il le veut.

Pardon ?? « void * » ??

- Nous sommes capables d'écrire « char *s = malloc(10) ; » sans problèmes car depuis le standard C89, « void * » est converti implicitement au type de la valeur de gauche dans l'initialisation

```
1 char *s1 = malloc(10);
2 char *s2 = (char *) malloc(10);
```

← correct ✓

← redondant, inutile

void * et un peu comme <T> en Java

Parlons structure de données

Nous allons faire une librairie qui permet a un utilisateur d’employer des liste chaine qui on était allouer avec malloc

Tout d'abord, la base

```
1 struct noeud {
2     int           val;
3     struct noeud *succ;
4 };
5
6 typedef struct noeud noeud
```

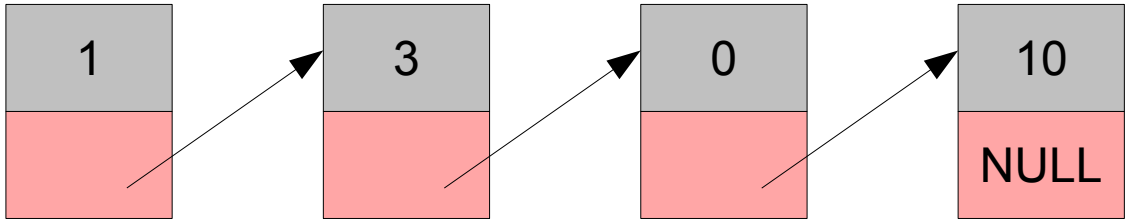
← le contenu sera un entier

← un pointeur vers le prochain maillon

← se lit : typedef struct nœud nœud et permet de référer a « struct noeud » en disant uniquement noeud

Listes simplement chaînées

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

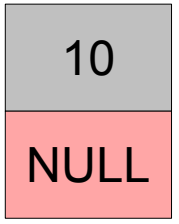


-

Listes simplement chaînées

Création d'un noeud simple

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

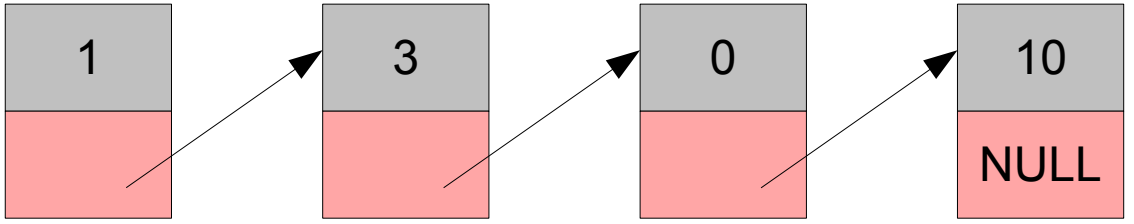



```
struct noeud{
  int val;
  struct noeud *succ;
};

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

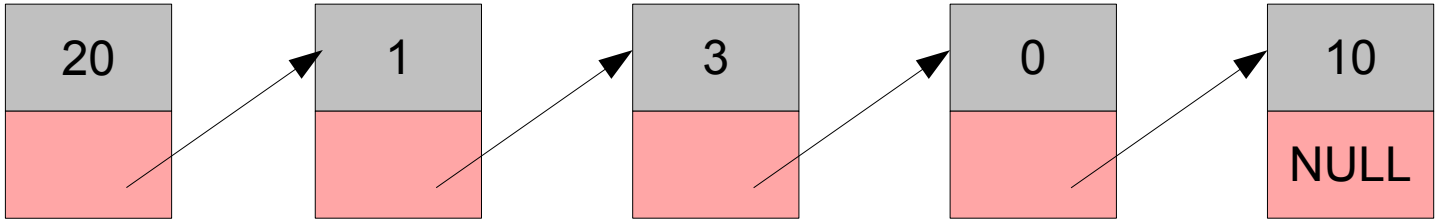



```
struct noeud{
  int val;
  struct noeud *succ;
};

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

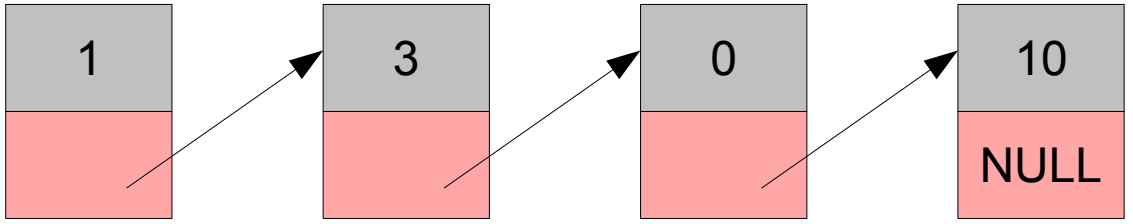



```
struct noeud{
  int val;
  struct noeud *succ;
};

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



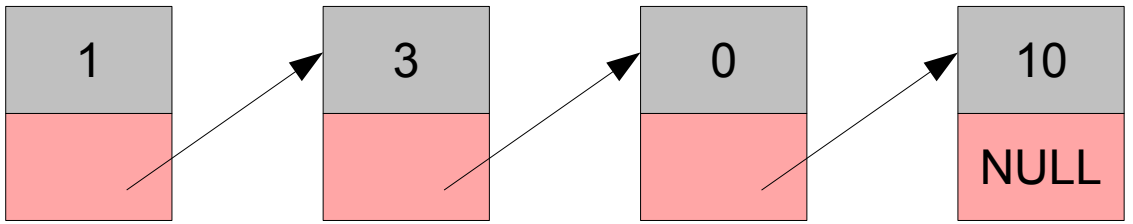
courant

```
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

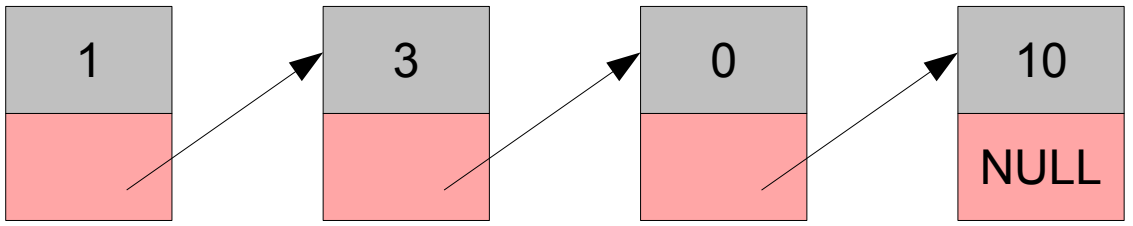



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

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

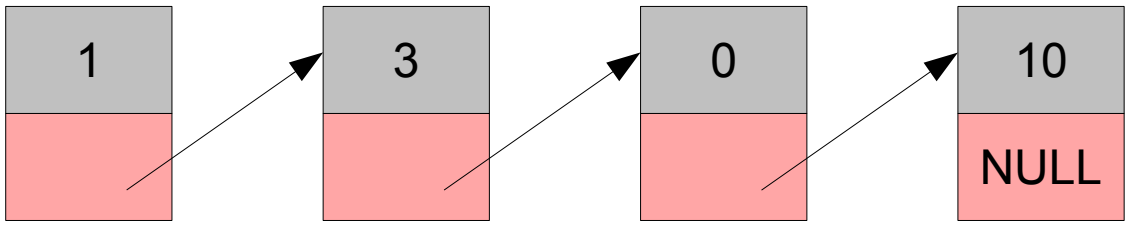



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

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

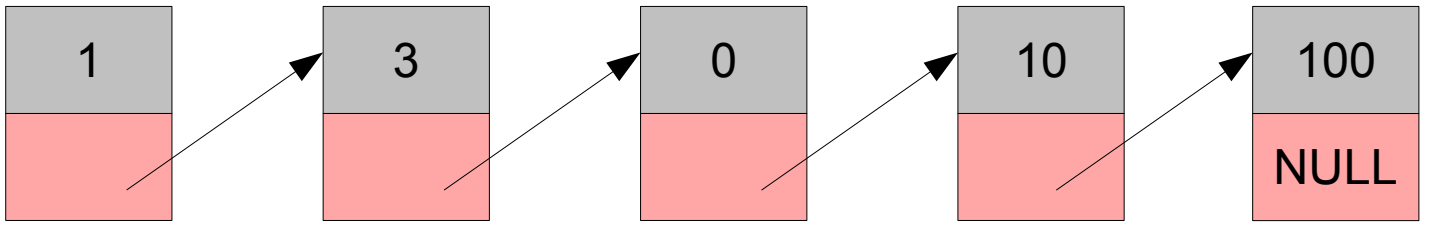



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Insertion dans une liste

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

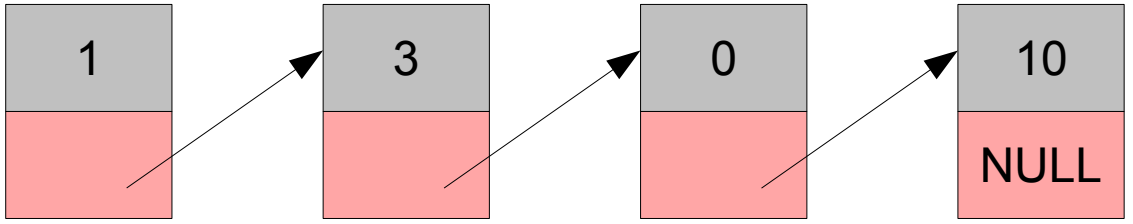



```
noeud *insertion_queue(noeud *li,int v){
  noeud *n=malloc(sizeof(noeud));
  n->val=v;
  n->succ=NULL;
  if(li==NULL){
    return n;
  }else{
    noeud *courant=li;
    while(courant->succ!=NULL){
      courant=courant->succ;
    }
    courant->succ=n;
    return li;
  }
}
```

Listes simplement chaînées

Affichage des éléments d'une liste

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

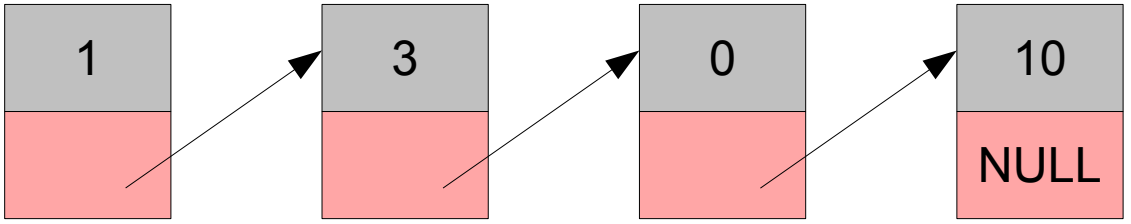



```
struct noeud{
  int val;
  struct noeud *succ;
};

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!!)**

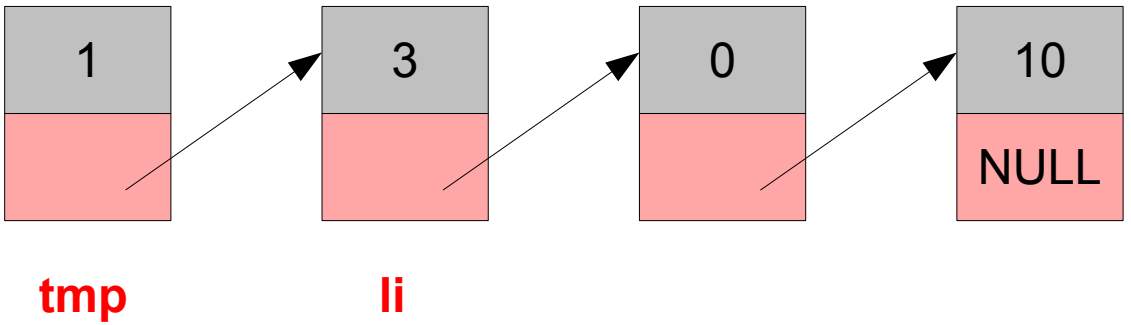



```
noeud *efface_tete(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    noeud *tmp=li;
    li=li->succ;
    free(tmp);
    return li;
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

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

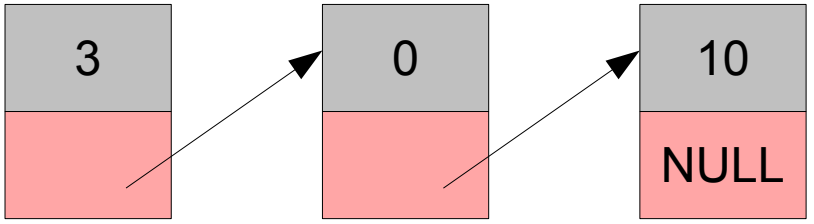



```
noeud *efface_tete(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    noeud *tmp=li;
    li=li->succ;
    free(tmp);
    return li;
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

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

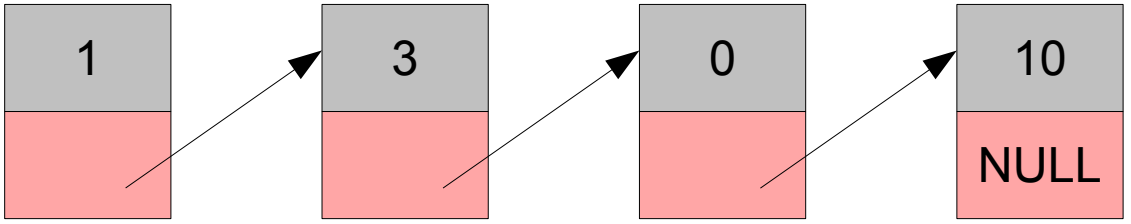



```
noeud *efface_tete(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    noeud *tmp=li;
    li=li->succ;
    free(tmp);
    return li;
  }
}
```

Listes simplement chaînées

Effacer un élément d'une liste

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

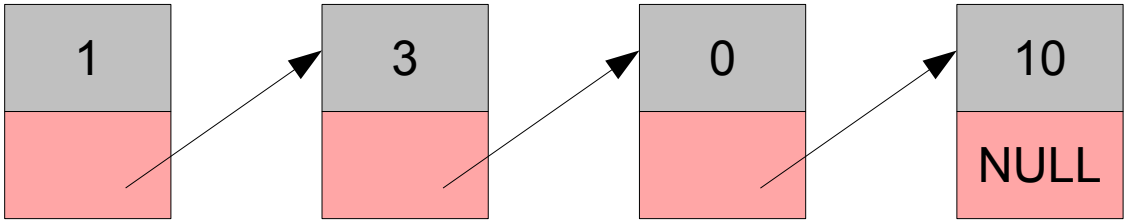



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
     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

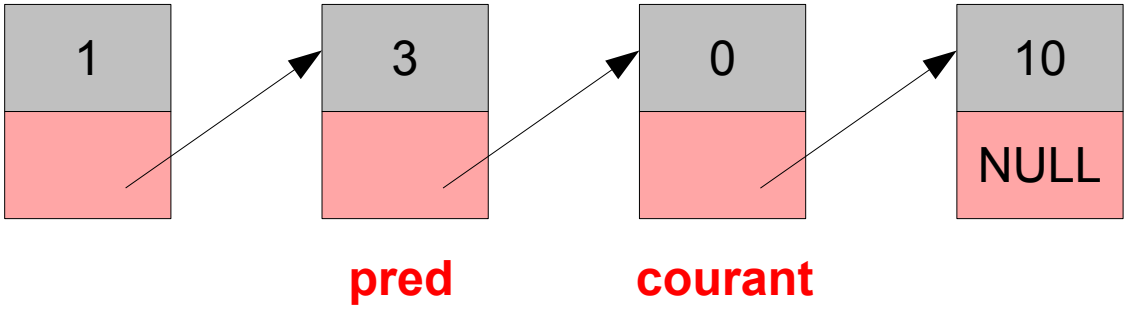



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
     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

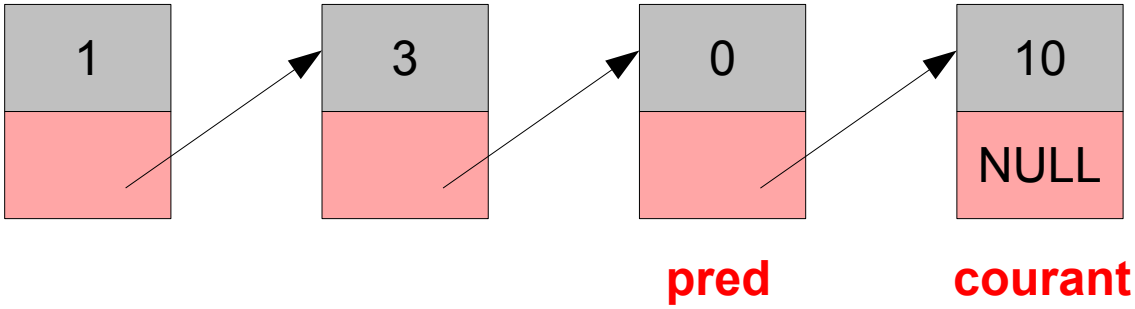



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
     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

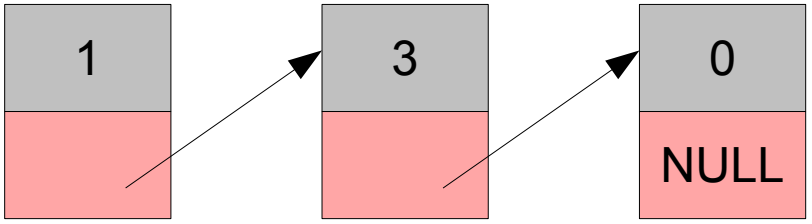



```
noeud *efface_queue(noeud *li){
  if(li==NULL){
    return NULL;
  }else{
    if(li->succ==NULL){
       free(li);
       return NULL;
    }else{
       noeud *pred=li;
       noeud *courant=li->succ;
       while(courant->succ!=NULL){
         pred=courant;
         courant=courant->succ;
     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

Listes simplement chaînées

Effacer un élément d'une liste

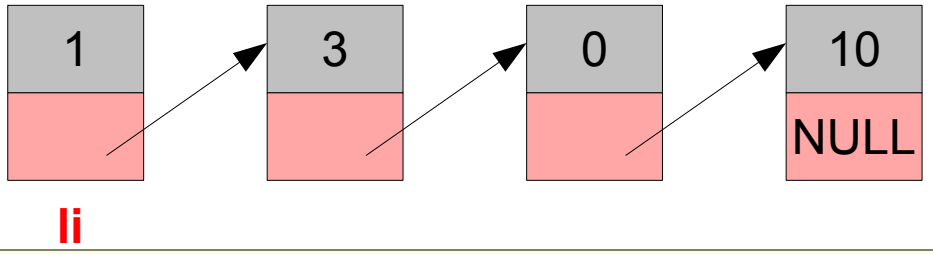
- On peut effacer l'élément de queue (**ne pas oublier de libérer la mémoire!!)**




```
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

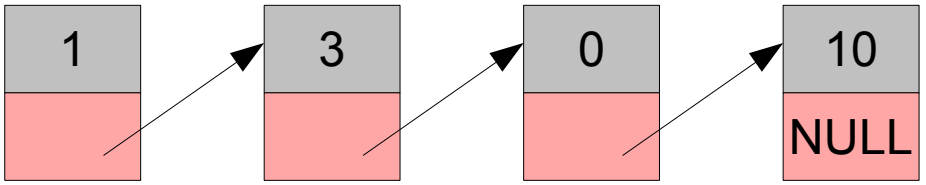
- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!)**




```
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**

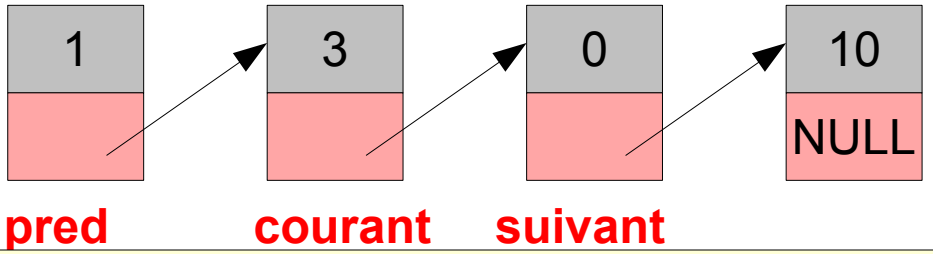


pred **courant** **suivant**

```
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

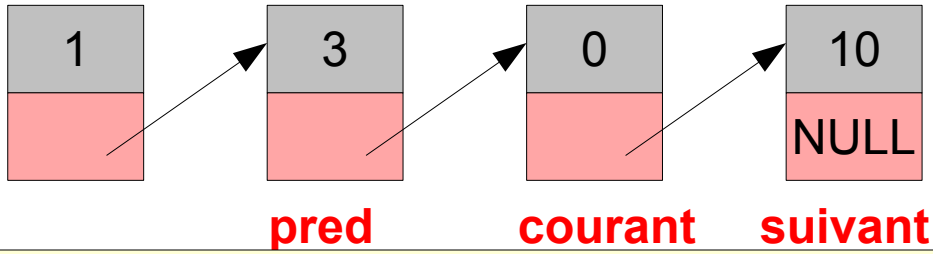
- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!) -> par exemple 0**




```
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

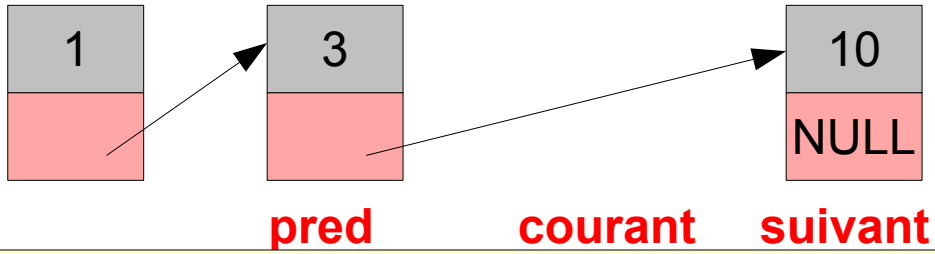
- On peut effacer un élément particulier (**ne pas oublier de libérer la mémoire!!) -> par exemple 0**




```
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**

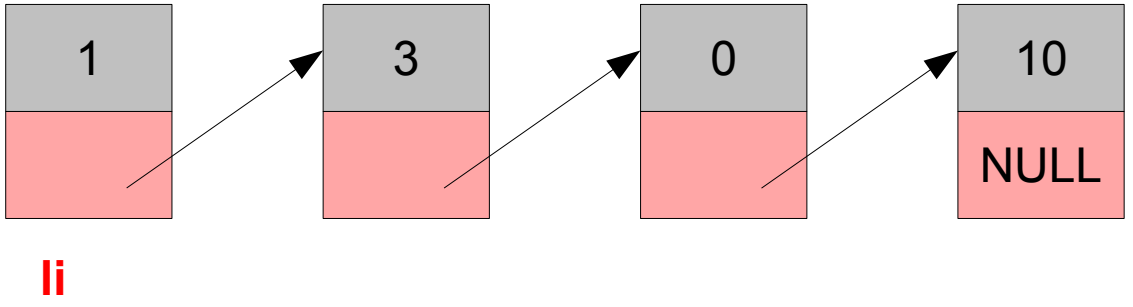



```
noeud *efface_val(noeud *li,int v){
  if(li==NULL){
    return NULL;
  }else{
    if(li->val==v){
      noeud *tmp=li->succ;
      free(li);
      return tmp;
    }else{
      noeud *pred=NULL;
      noeud *courant=li;
      noeud *suivant=li->succ;
      while(courant->val!=v && suivant!=NULL){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

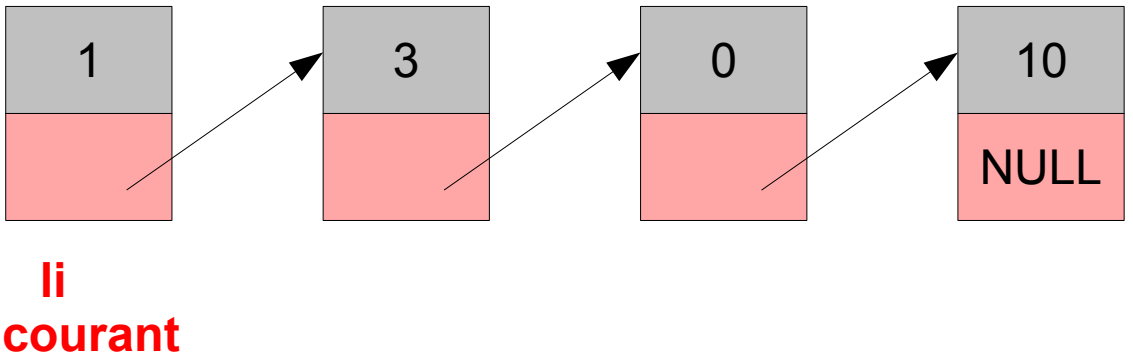



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

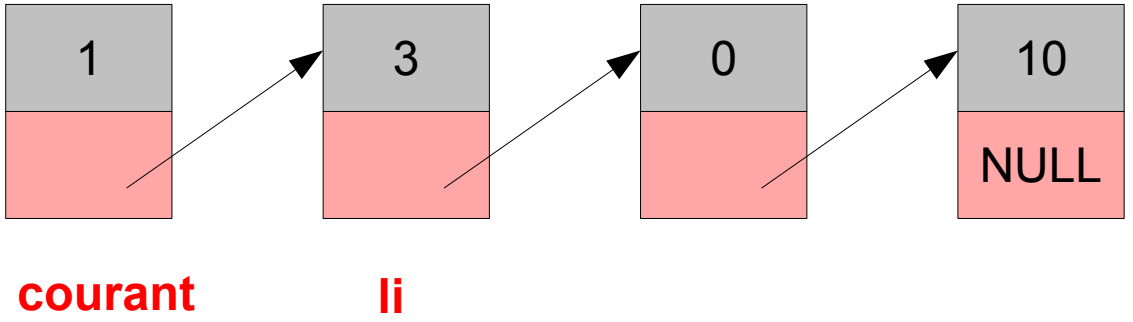



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste

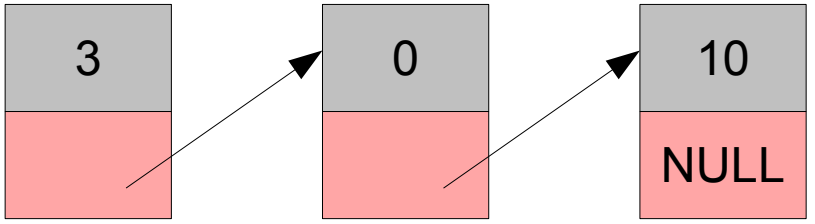



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



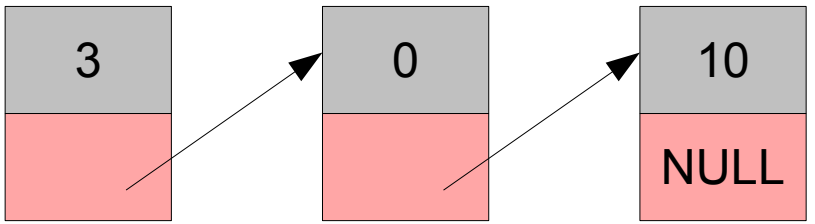
courant **li**

```
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

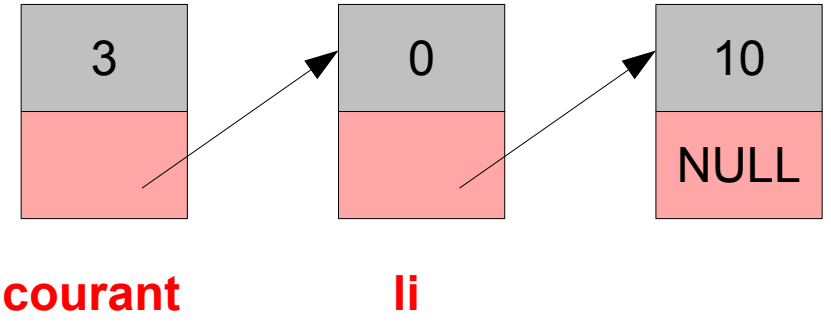
courant

```
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

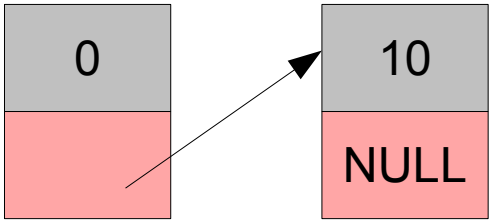



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



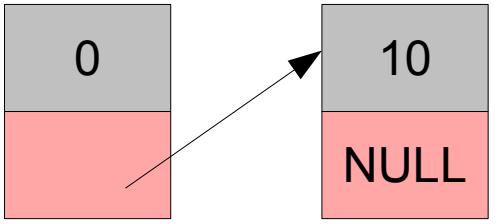
courant **li**

```
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



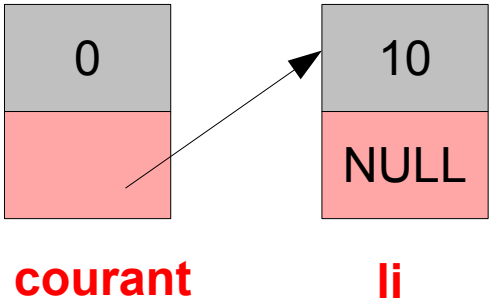
li
courant

```
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

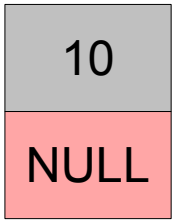



```
void efface(noeud *li){
  noeud *courant=li;
  while(courant!=NULL){
    li=courant->succ;
    free(courant);
    courant=li;
  }
}
```

Listes simplement chaînées

Effacer une liste

- On peut libérer tous les éléments d'une liste



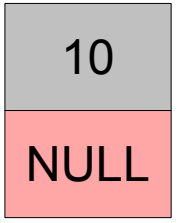
courant **li**

```
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



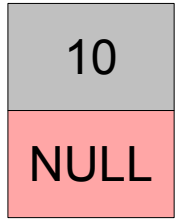
li
courant

```
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

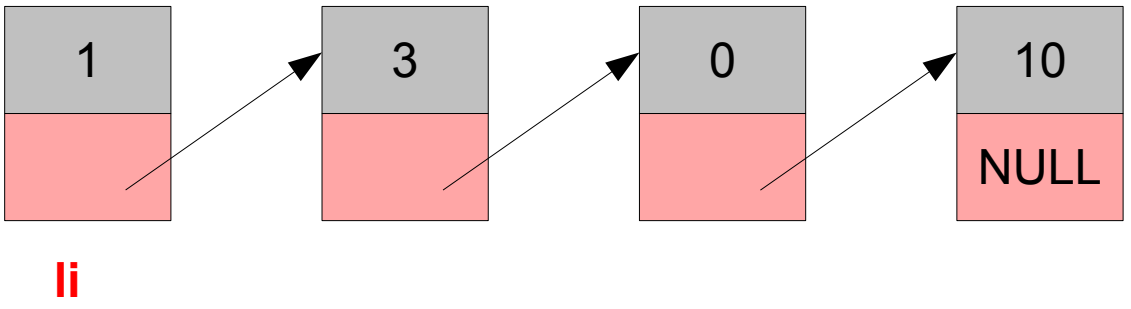
courant

```
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

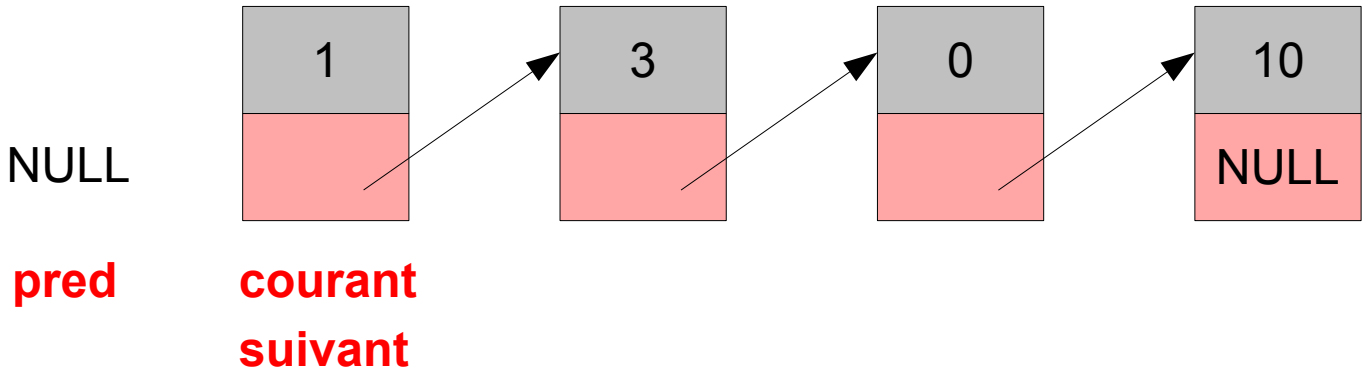



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

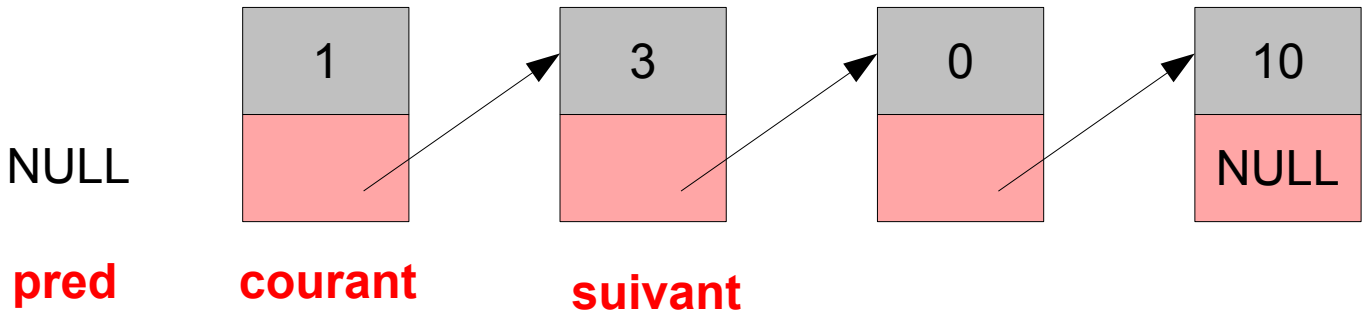



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

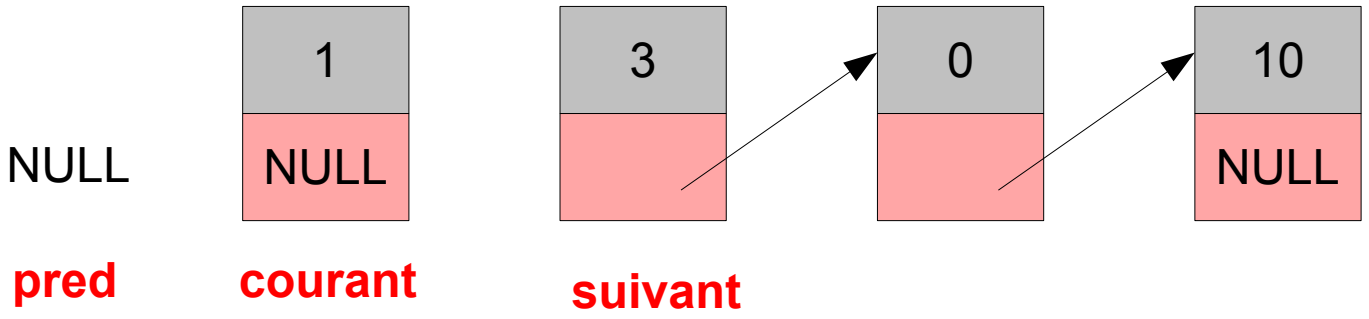



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

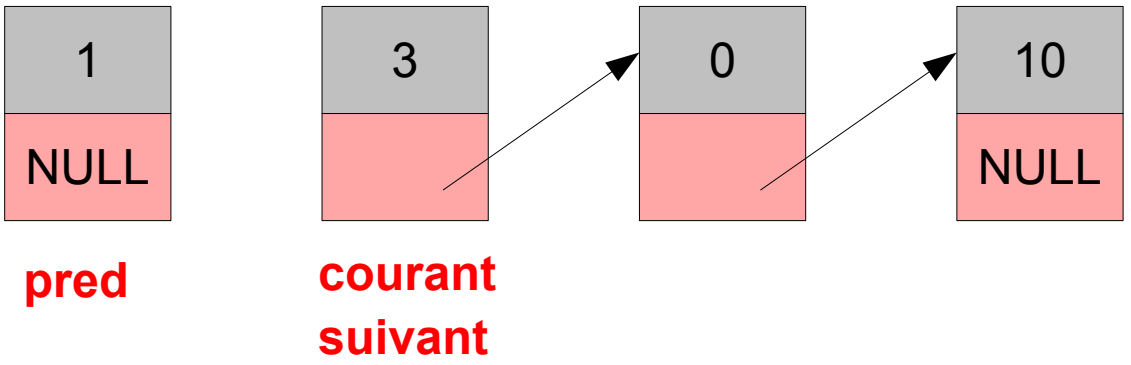



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

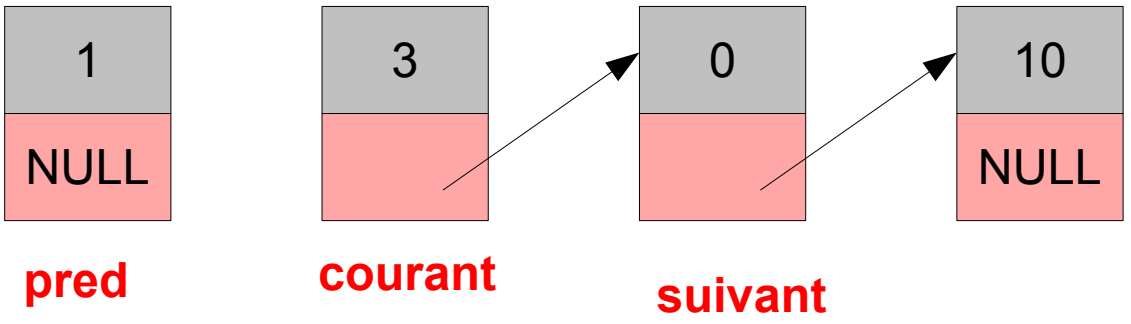



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

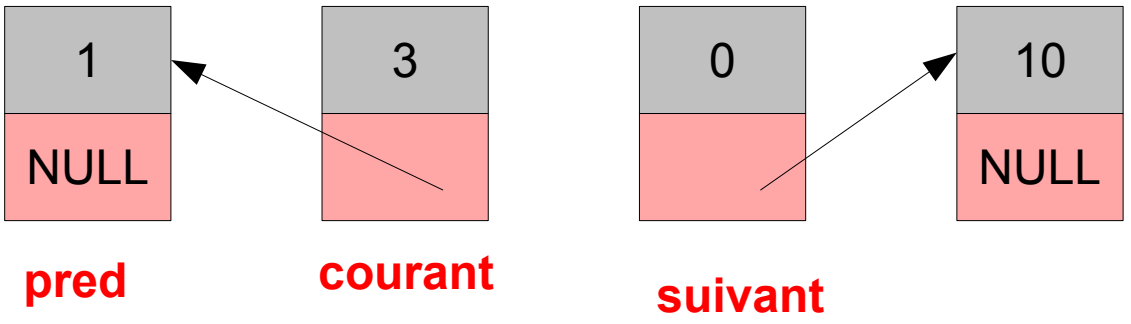



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

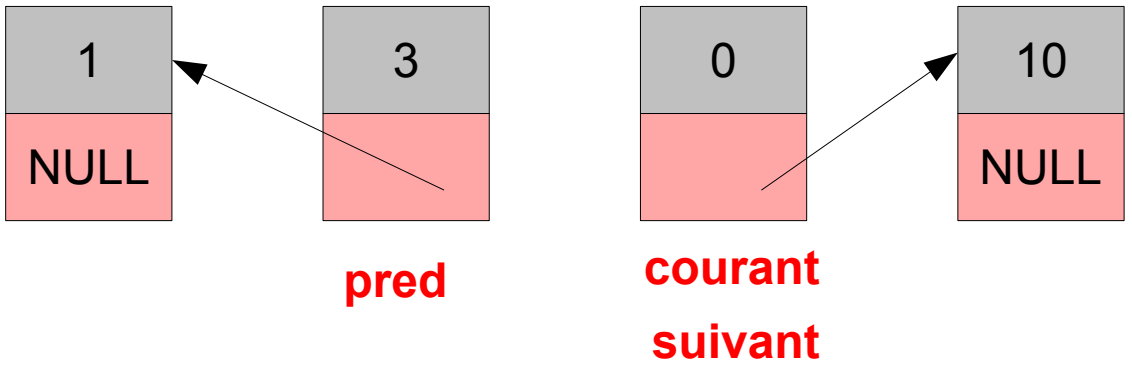



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

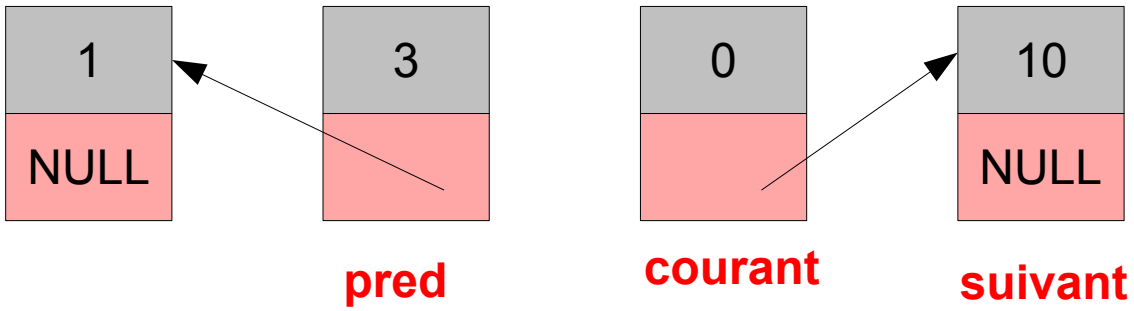



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

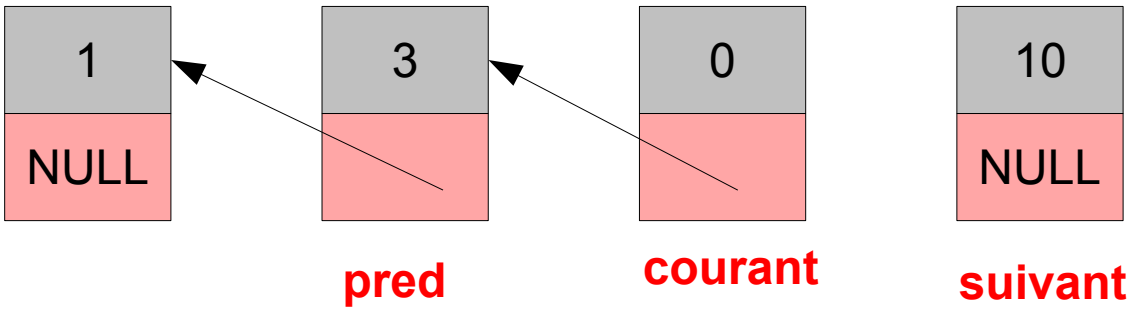



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

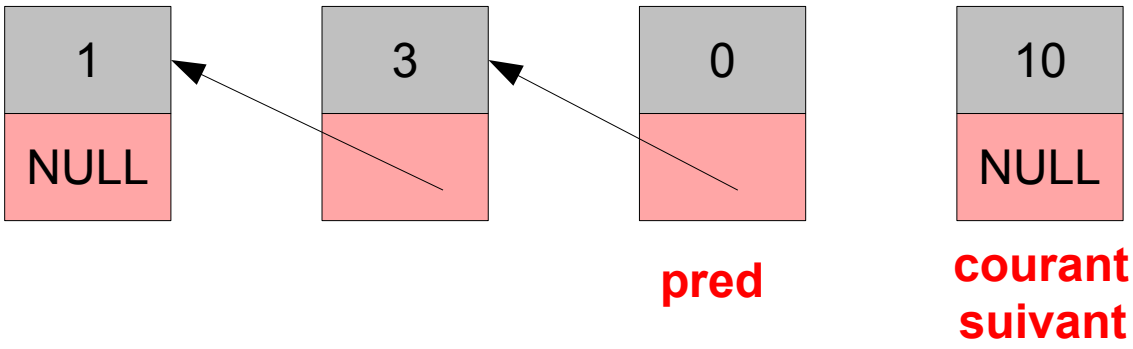



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

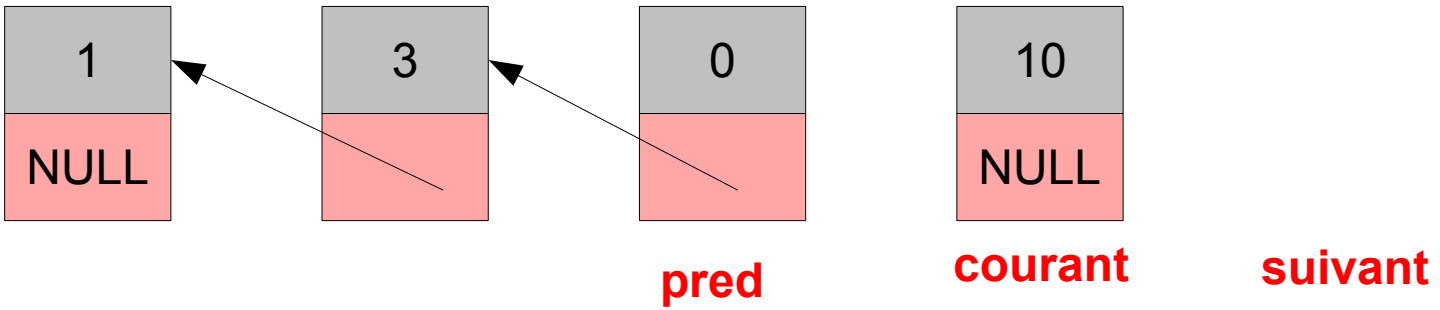



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

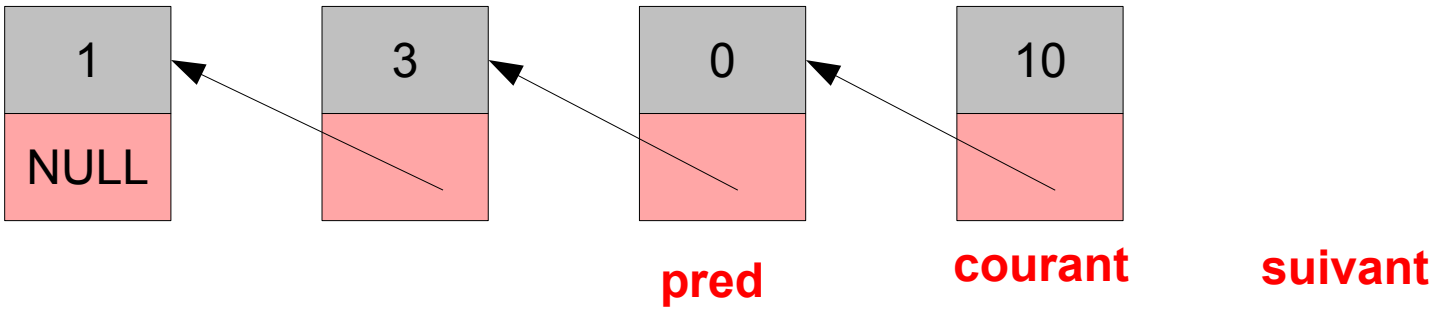



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

Listes simplement chaînées

Renverser une liste

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

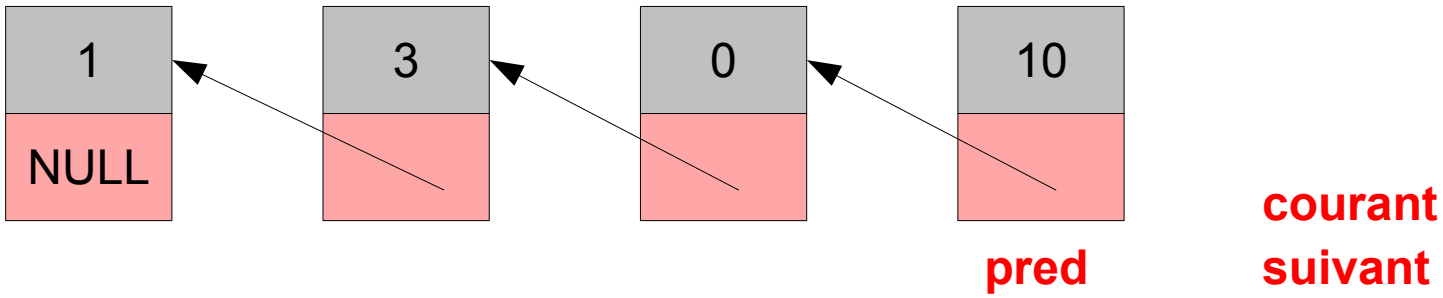



```
noeud *reverse(noeud *li){
  noeud *pred=NULL;
  noeud *courant=li;
  noeud *suivant=li;
  while(courant!=NULL){
    suivant=courant->succ;
    courant->succ=pred;
    pred=courant;
    courant=suivant;
  }
  return pred;
}
```

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;
}
```