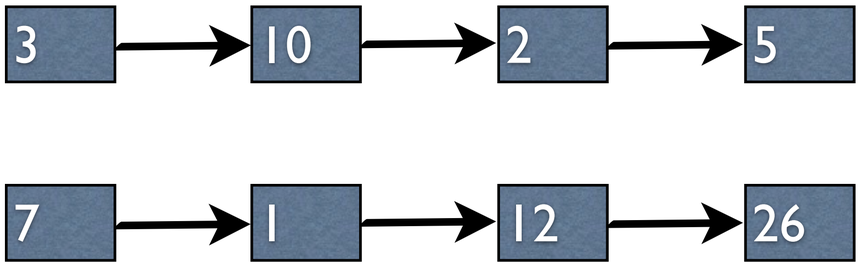
Listes

Dr MAMBE Moïse

Objectif

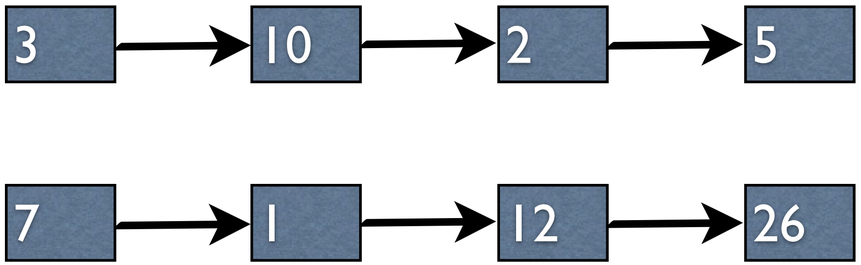
- Structure dynamique permettant:
  - Ajouter un élément en début / fin
  - Supprimer un élément en début / fin
  - Concaténer deux listes

Exemples



Exemples

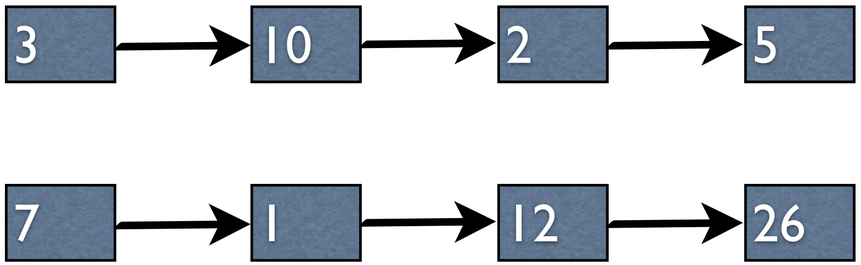
Ajout en tête



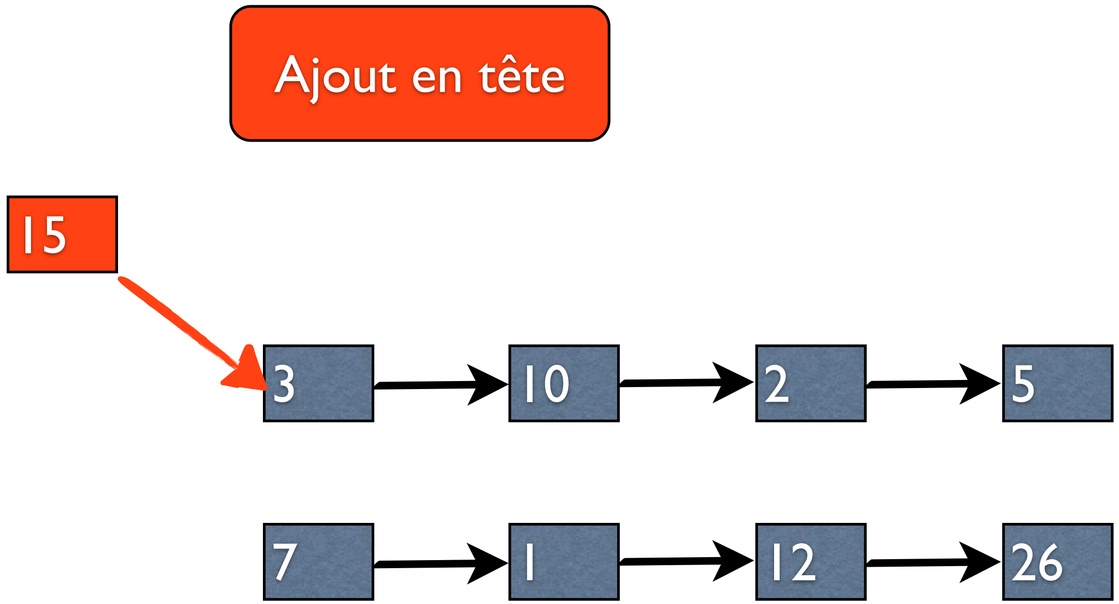
Exemples

Ajout en tête

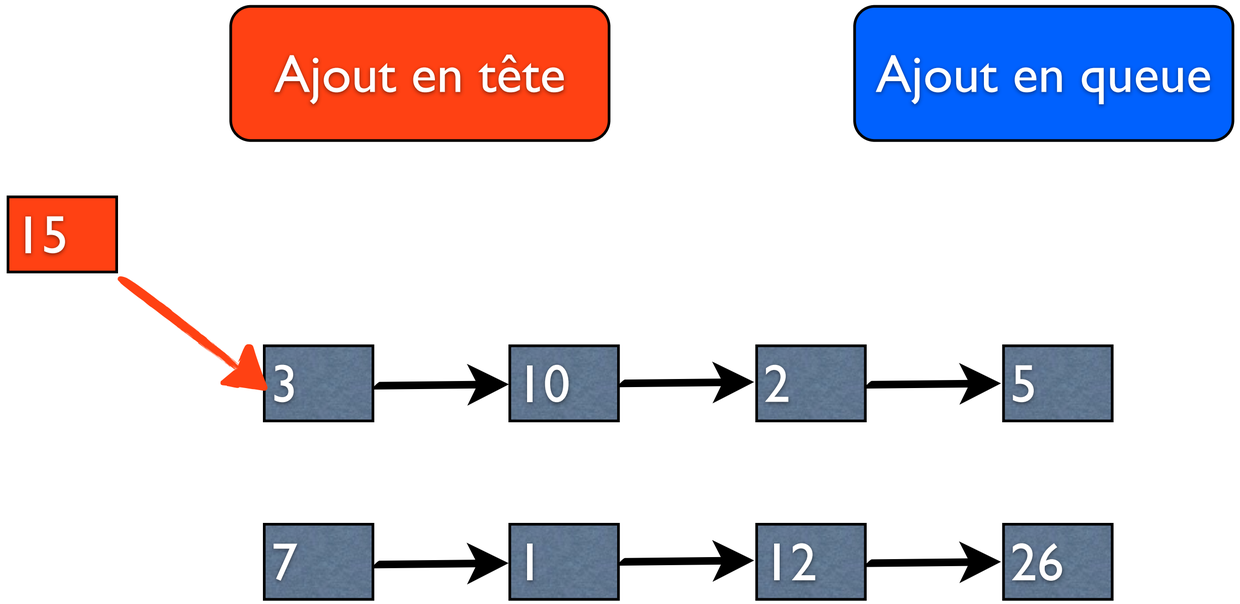
15



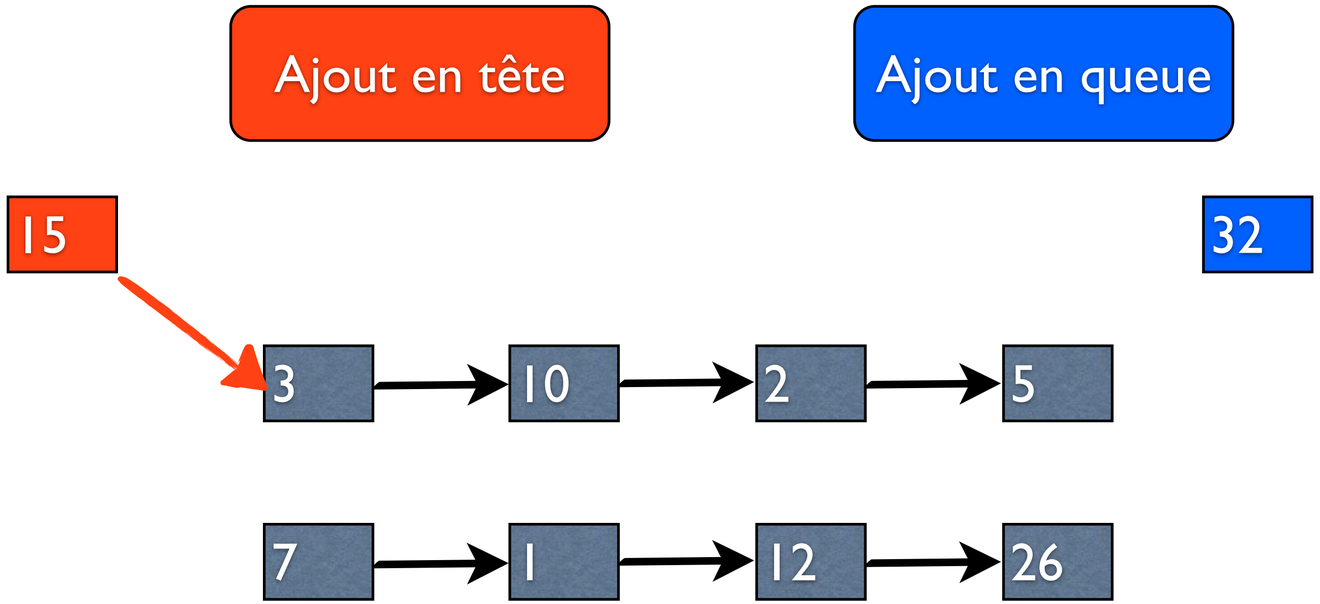
Exemples



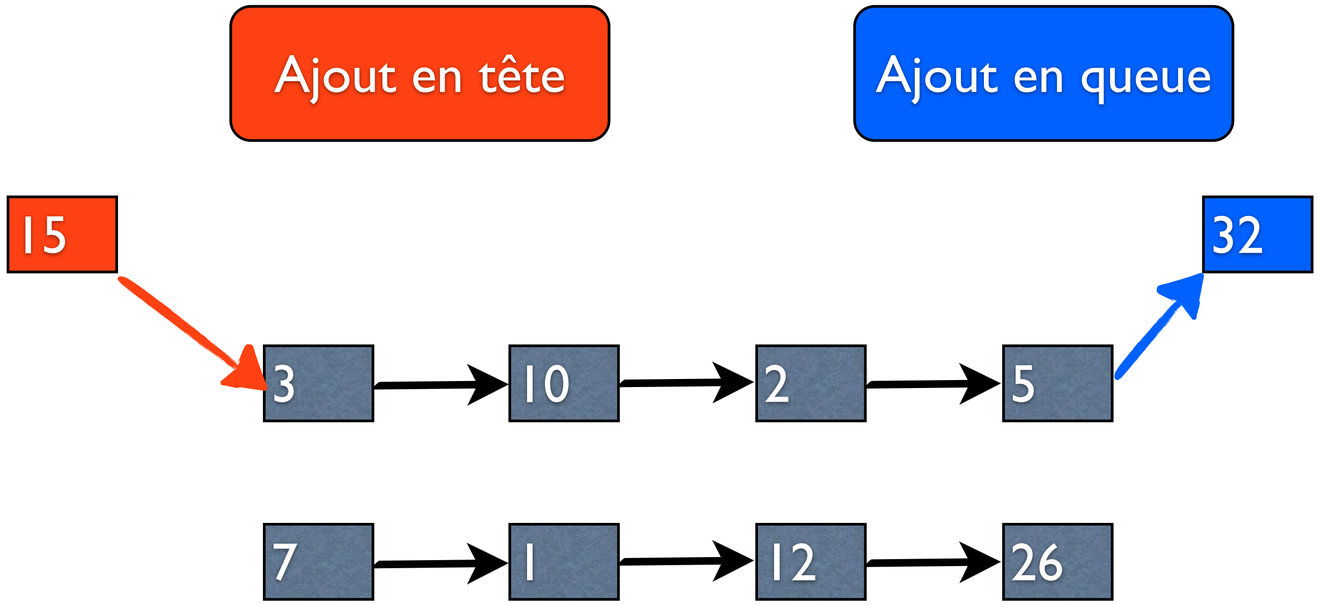
Exemples



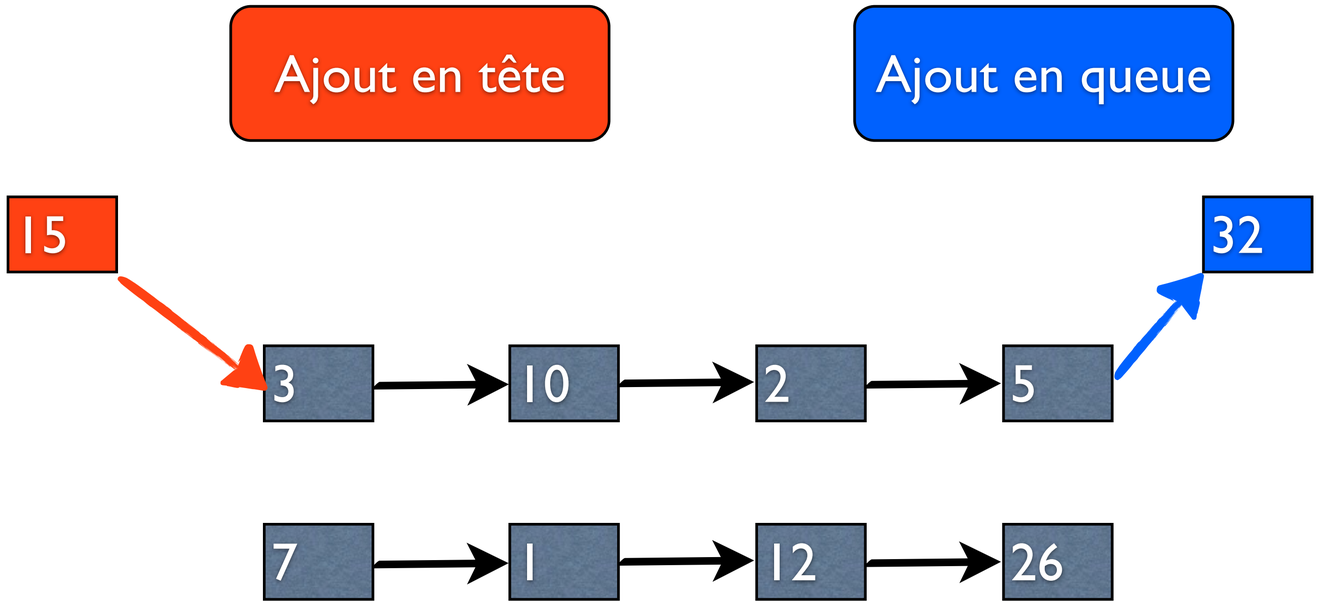
Exemples



Exemples

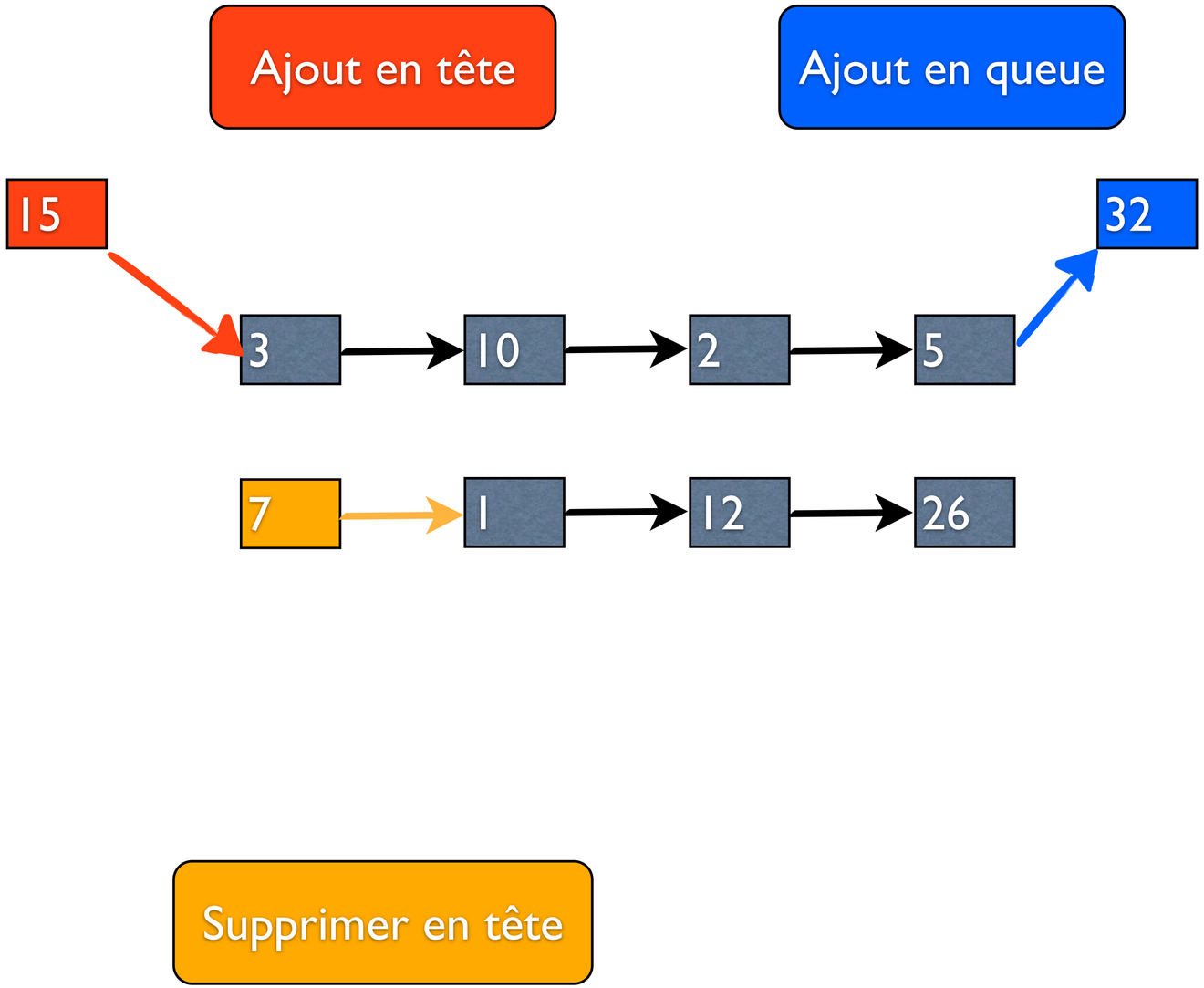


Exemples

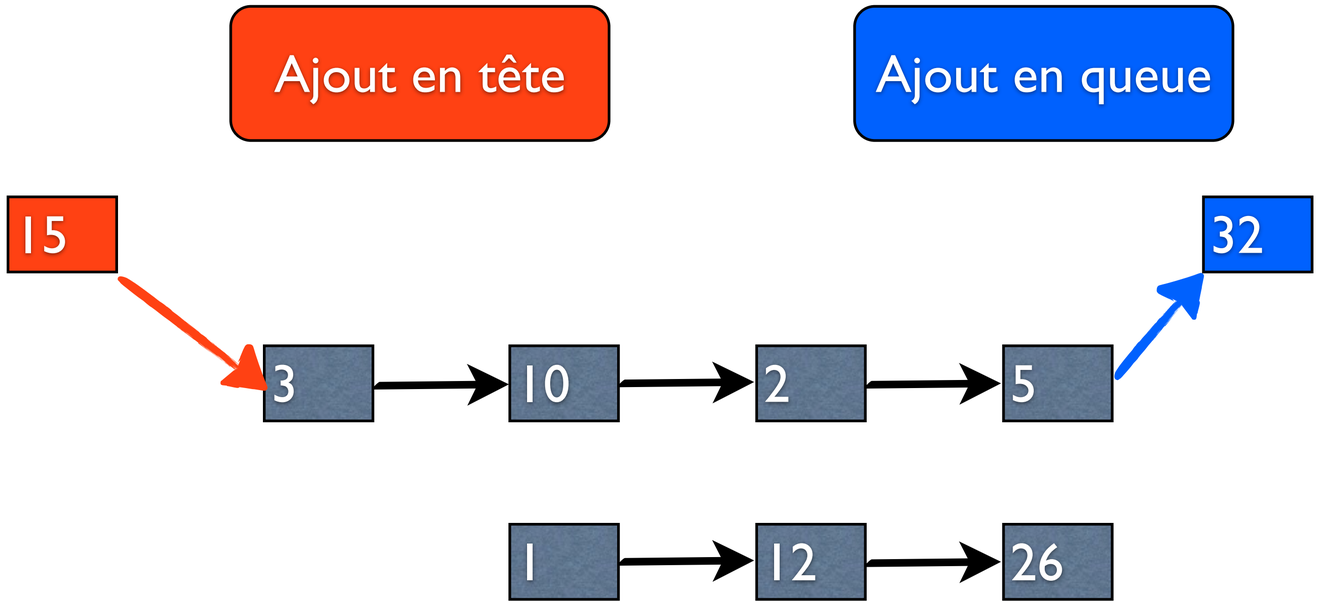


Supprimer en tête

Exemples

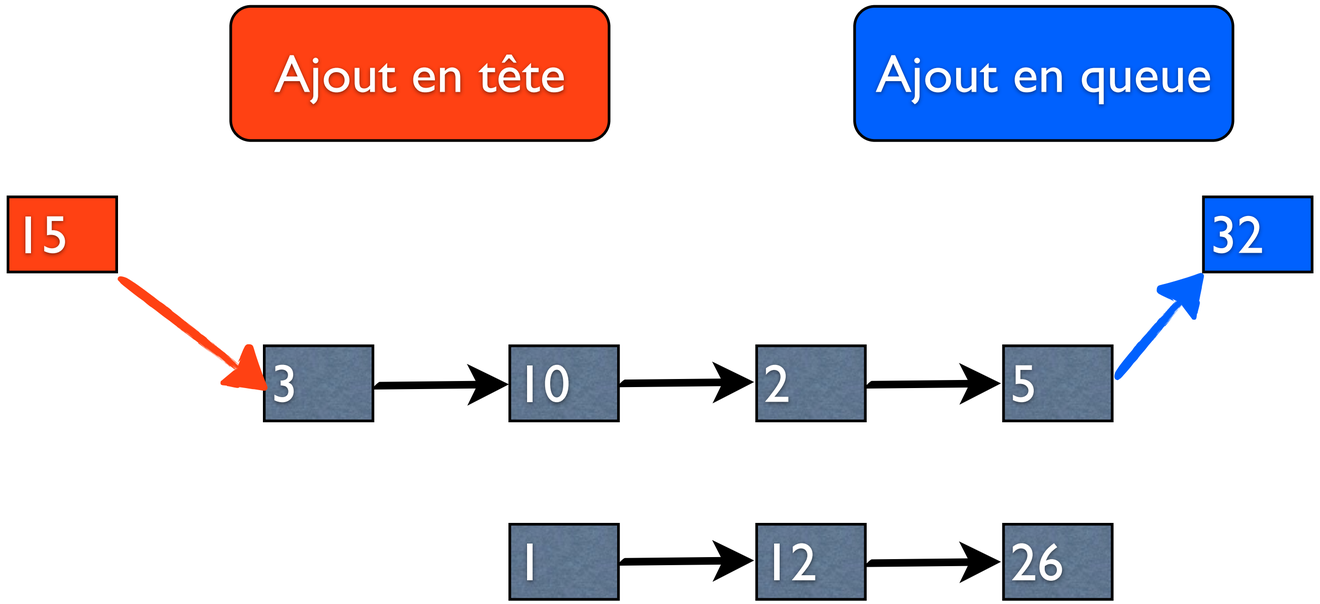

Ajout en tête
Ajout en queue
15
32
3
10
2
5
7
1
12
26
Supprimer en tête

Exemples

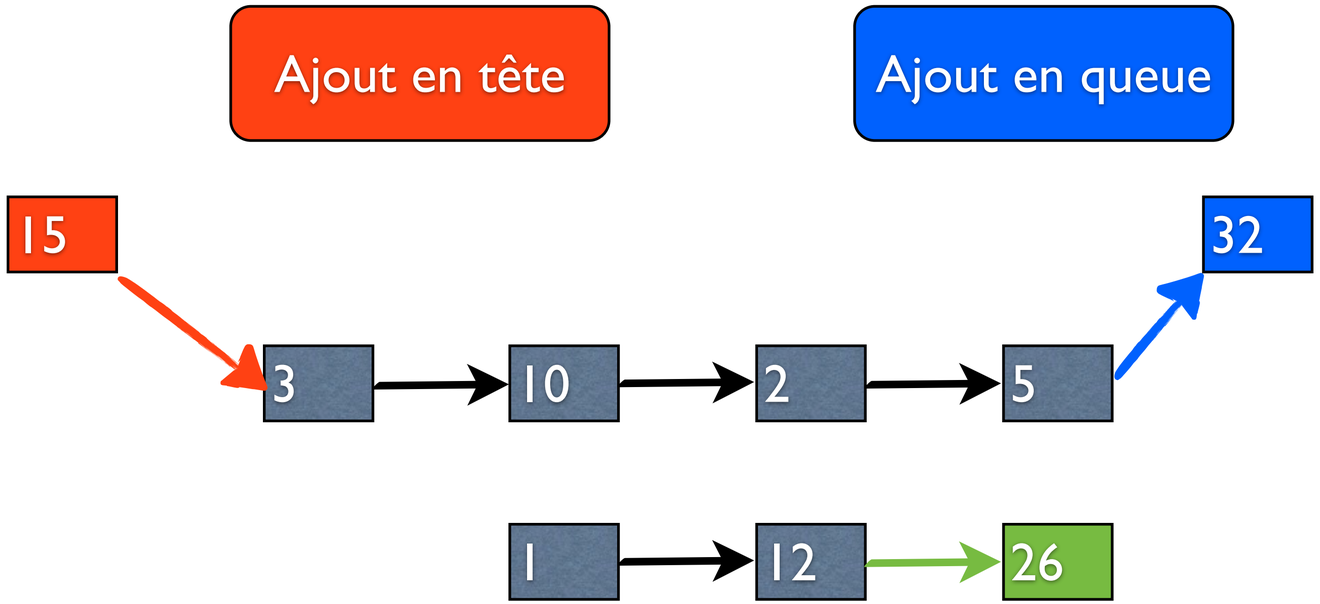


Supprimer en tête

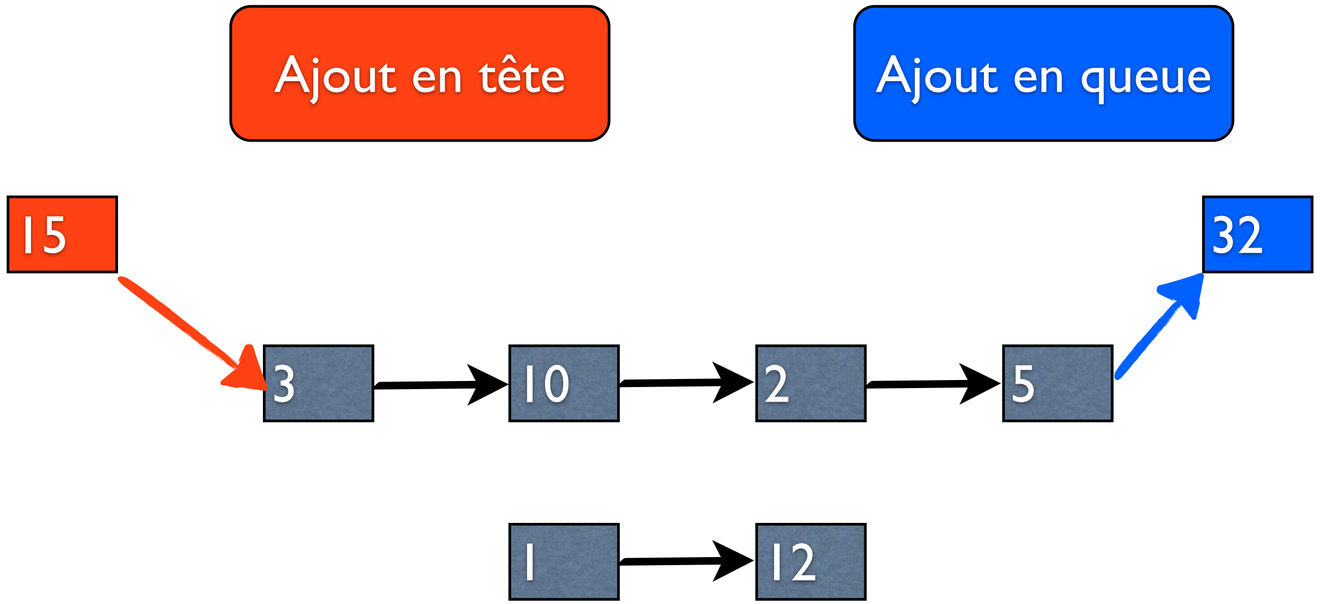
Exemples


Ajout en tête
Ajout en queue
15
32
3
10
2
5
1
12
26
Supprimer en tête
Supprimer en queue

Exemples


Ajout en tête
Ajout en queue
15
32
3
10
2
5
1
12
26
Supprimer en tête
Supprimer en queue

Exemples



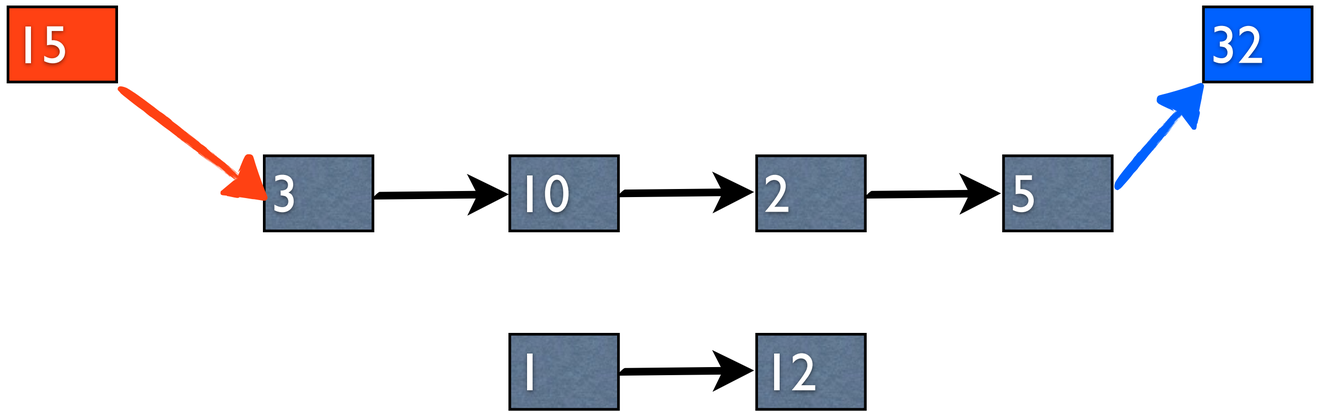
Supprimer en tête

Supprimer en queue

Exemples

Ajout en tête

Ajout en queue



Concaténer les deux listes

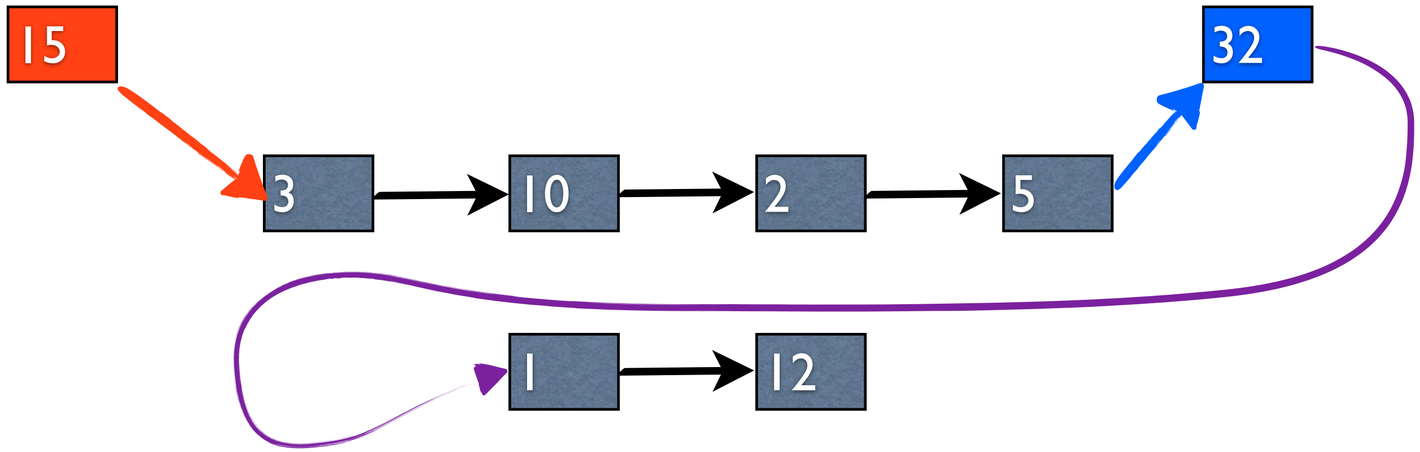
Supprimer en tête

Supprimer en queue

Exemples

Ajout en tête

Ajout en queue

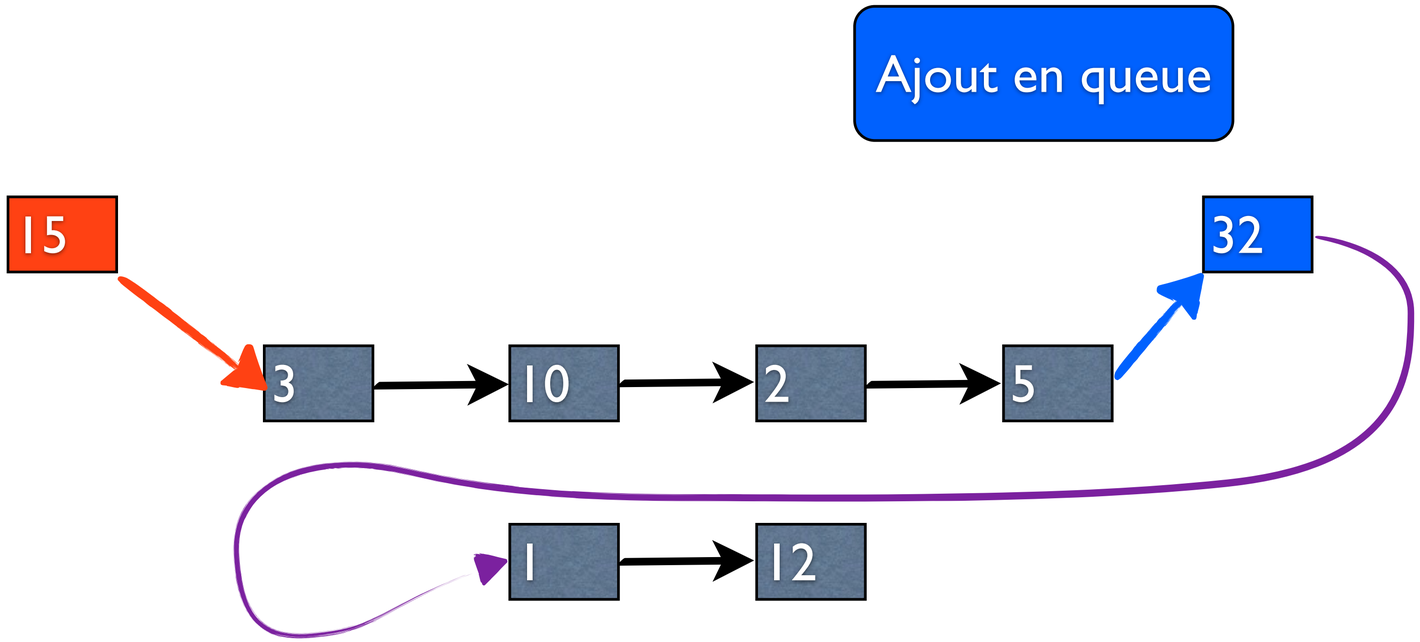


Concaténer les deux listes

Supprimer en tête

Supprimer en queue

Exemples



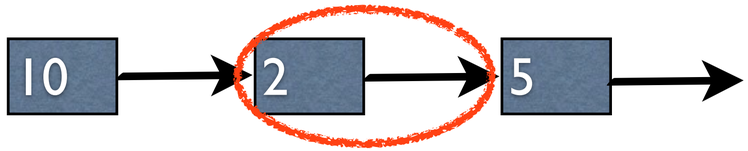
Concaténer les deux listes

Ajout en tête

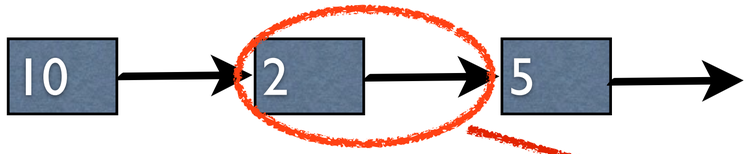
Supprimer en tête

Codage

Codage

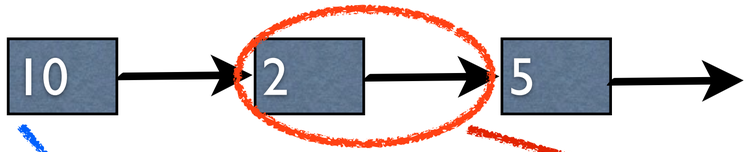


Codage




```
typedef struct Cellule Cellule;

struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

Codage




```
typedef struct Liste Liste;
struct Liste
{
    Cellule *tete;
};
```

```
typedef struct Cellule Cellule;
struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

Initialisation et ajout

```
Liste *initialisation()
{
    Liste *liste = malloc(sizeof(*liste));
    Cellule *cellule = malloc(sizeof(*cellule));
    if (liste == NULL || cellule  == NULL)
    {
        exit(EXIT_FAILURE);
    }
    cellule ->contenu = 0;
cellule ->suivante = NULL;
liste->tete = cellule      return
liste;
}
```

Initialisation et ajout

```c
typedef struct Cellule Cellule;
struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```c
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout

```c
typedef struct Cellule Cellule;
struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```c
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout

```c
typedef struct Cellule Cellule;
struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```c
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout

```c
typedef struct Cellule Cellule;
struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```c
void main(){
Liste l;
insertion(l, 10): }
```

```c
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout

```c
typedef struct Cellule Cellule;

struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```c
void main(){
Liste l;
insertion(l, 10): }
```

```c
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout


```
typedef struct Cellule Cellule;

struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```
void main(){
Liste l;
insertion(l, 10): }
```

```
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

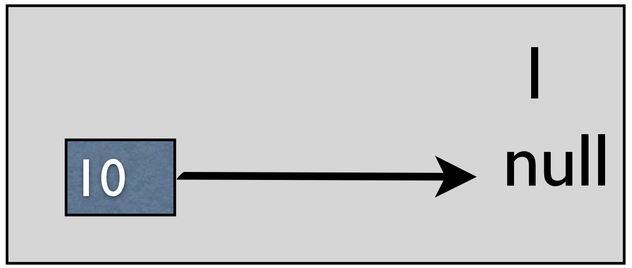
Initialisation et ajout


```
typedef struct Cellule Cellule;

struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```
void main(){
Liste l;
insertion(l, 10): }
```

```
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout

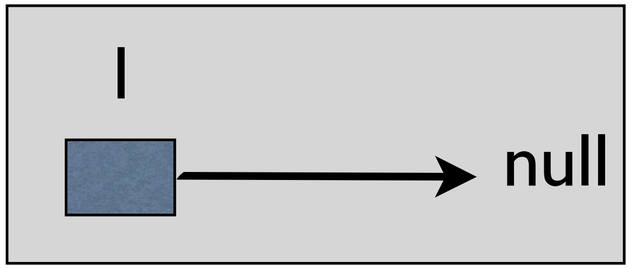



```
typedef struct Cellule Cellule;

struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

```
void main(){
Liste l;
insertion(l, 10): }
```

```
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

Initialisation et ajout




```c
typedef struct Cellule Cellule;

struct Cellule
{
    int contenu;
    Cellule *suivante;
};
```

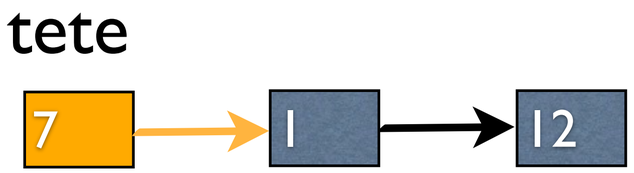
```c
void main(){
Liste l;
insertion(l, 10): }
```

```c
void insertion(Liste *liste, int nvNombre)
{
    /* Création d'une nouvelle cellule */
    Cellule *nouvelle= malloc(sizeof(*nouvelle));
    if (liste == NULL || nouvelle == NULL)
    {
        exit(EXIT_FAILURE);
    }
    nouvelle->contenu = nvNombre;
    /* Insertion de l'élément au début de la liste */
    nouvelle->suivante = liste->tete;
    liste->tete = nouvelle;
}
```

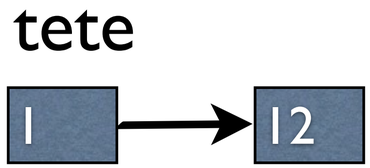
Suppression en tête

tete

Suppression en tête



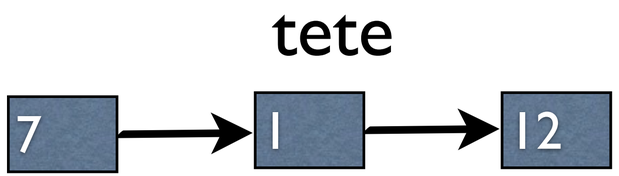
Suppression en tête



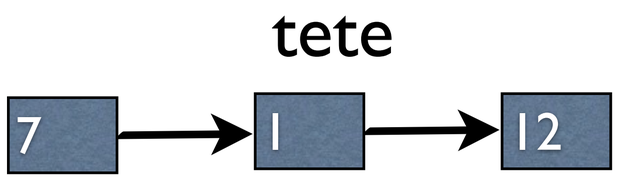
Suppression en tête

tete

Suppression en tête

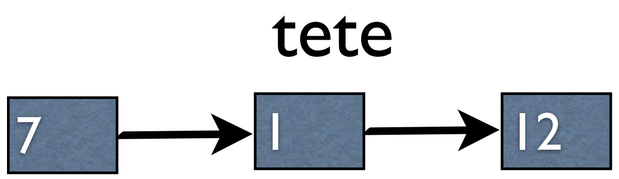


Suppression en tête



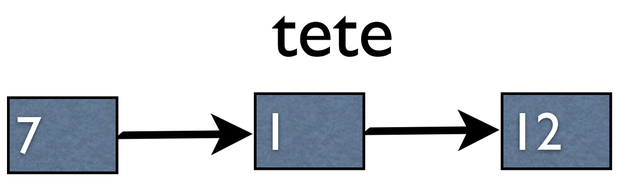

```
void suppression(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
  if (liste ->tete != NULL)
    {
        Cellule *aSupprimer = liste->tete;
        liste->tete = liste->tete->suivante;
        free(aSupprimer);
    }
  }
```

Suppression en tête




```
void suppression(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
  if (liste ->tete != NULL)
    {
        Cellule *aSupprimer = liste->tete;
        liste->tete = liste->tete->suivante;
        free(aSupprimer);
    }
  }
```

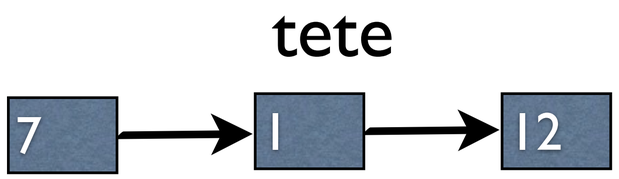
Suppression en tête




```
tete = tete.suivante;
```

```
void suppression(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
  if (liste ->tete != NULL)
    {
        Cellule *aSupprimer = liste->tete;
        liste->tete = liste->tete->suivante;
        free(aSupprimer);
    }
}
```

Suppression en tête




```
tete = null.suivante;
```

```
void suppression(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
  if (liste ->tete != NULL)
    {
        Cellule *aSupprimer = liste->tete;
        liste->tete = liste->tete->suivante;
        free(aSupprimer);
    }
}
```

Afficher la liste

Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
  }
```

Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```

Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```

Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```

sb

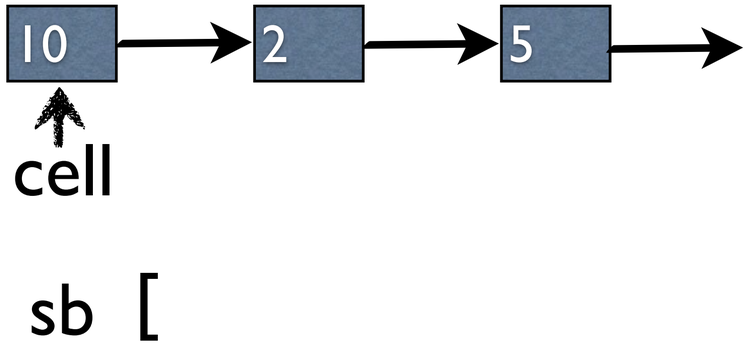
Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```

sb [

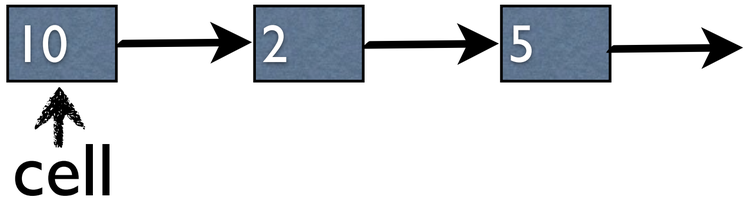
Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



Afficher la liste

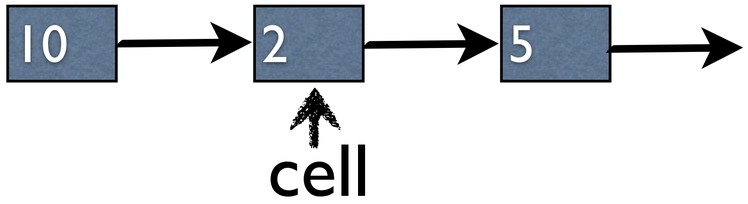
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10

Afficher la liste

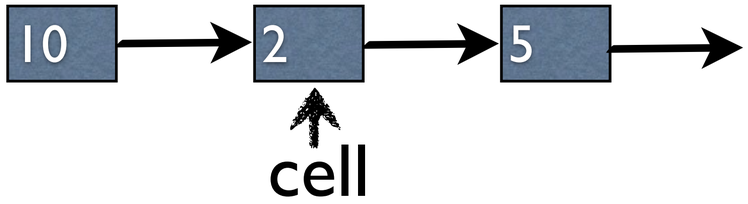
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10

Afficher la liste

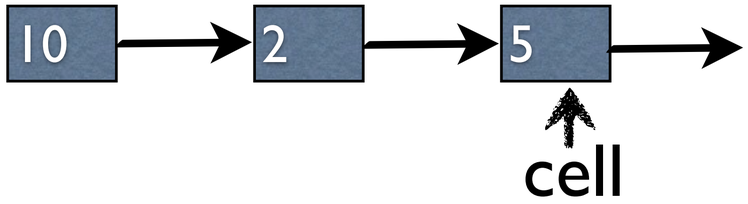
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10 2

Afficher la liste

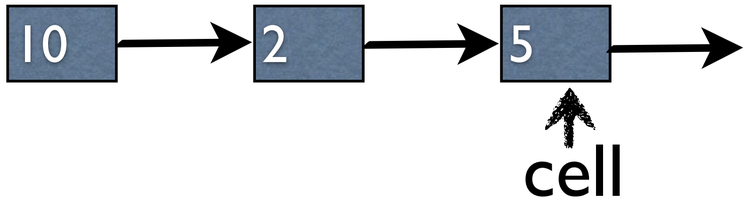
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10 2

Afficher la liste

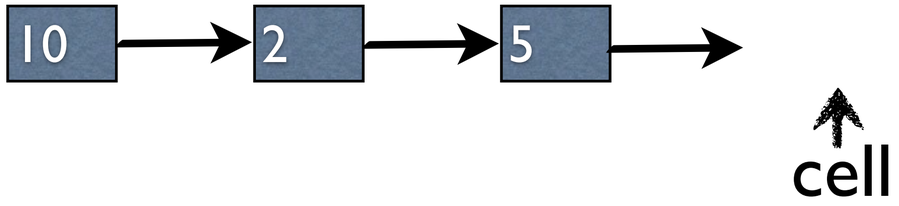
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10 2 5

Afficher la liste

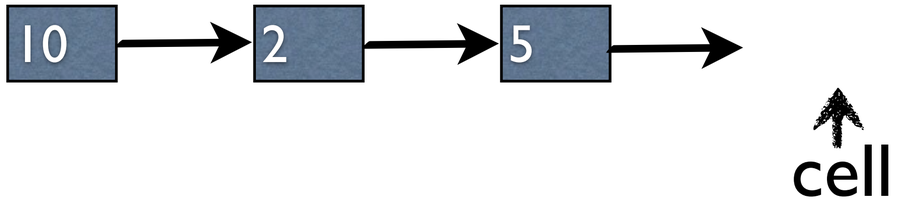
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10 2 5

Afficher la liste

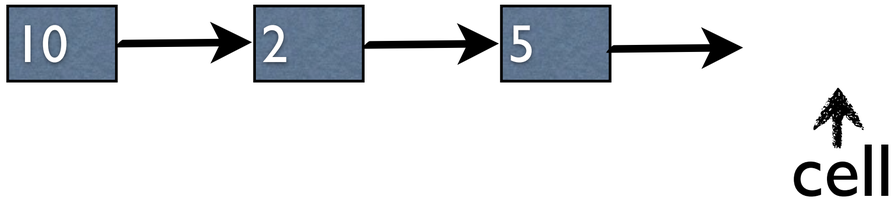
```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10 2 5

Afficher la liste

```
void afficherListe(Liste *liste)
{
    if (liste == NULL)
    {
        exit(EXIT_FAILURE);
    }
   Cellule *cell = liste->tete;
  while (cell != NULL)
    {
         printf("%d -> ", cell->contenu);
        cell = cell->suivant;
    }
}
```



sb [10 2 5]

Les piles et les files

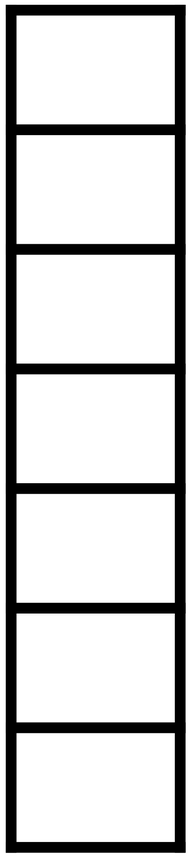
Ce sont deux variantes un peu particulières des listes chaînées. Elles permettent de contrôler la manière dont sont ajoutés les nouveaux éléments. Cette fois, on ne va plus insérer de nouveaux éléments au milieu de la liste mais seulement au début ou à la fin.

Les piles et les files sont très utiles pour des programmes qui doivent traiter des données qui arrivent au fur et à mesure.

Implémentation d'une pile

```c
typedef struct Element Element;
struct Element
{
    int nombre;
    Element *suivant;
};
```

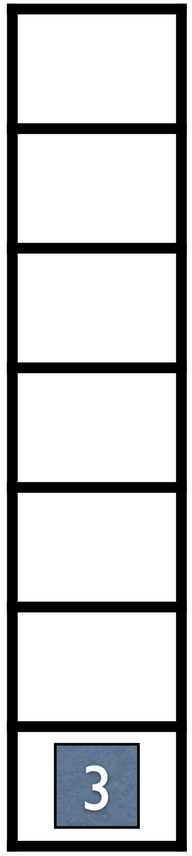
```c
typedef struct Pile Pile;
struct Pile
{
    Element *premier;
};
```



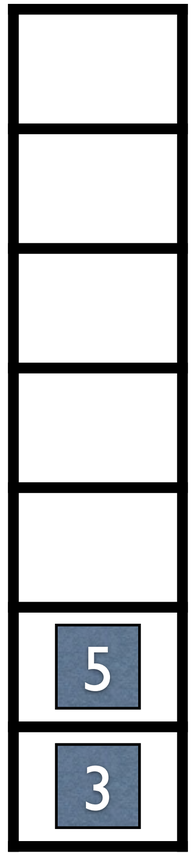
Implémentation d'une pile

Empilage

```c
void empiler(Pile *pile, int nvNombre)
{
    Element *nouveau = malloc(sizeof(*nouveau));
    if (pile == NULL || nouveau == NULL)
    {
        exit(EXIT_FAILURE);
    }

    nouveau->nombre = nvNombre;
    nouveau->suivant = pile->premier;
    pile->premier = nouveau;
}
```



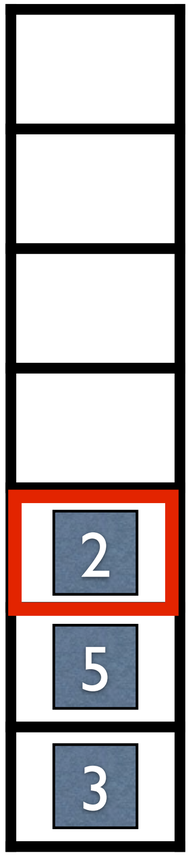
Implémentation d'une pile



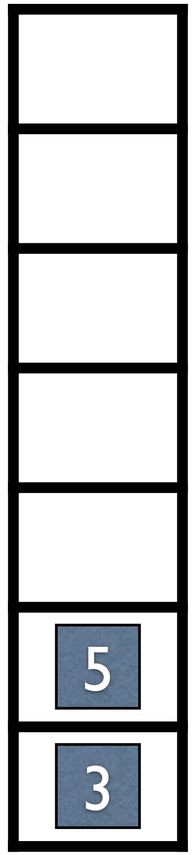
Implémentation d'une pile

Dépilage

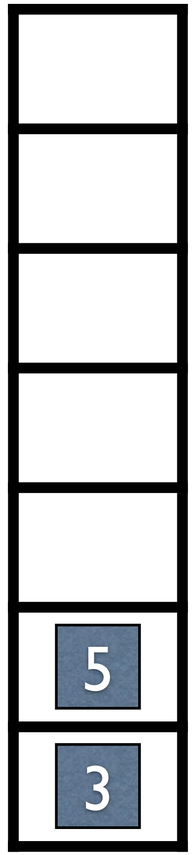
```c
int depiler(Pile *pile)
{
    if (pile == NULL)
    {
        exit(EXIT_FAILURE);
    }

    int nombreDepile = 0;
    Element *elementDepile = pile->premier;

    if (pile != NULL && pile->premier != NULL)
    {
        nombreDepile = elementDepile->nombre;
        pile->premier = elementDepile->suivant;
        free(elementDepile);
    }

    return nombreDepile;
}
```



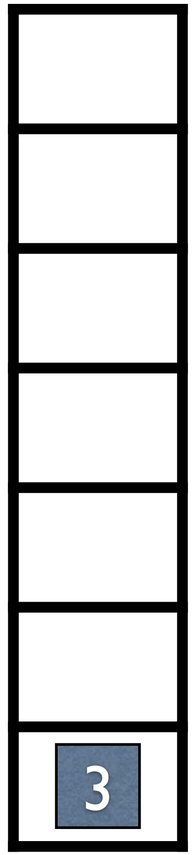
Implémentation d'une pile



Implémentation d'une pile



Implémentation d'une pile



Implémentation d'une file

Implémentation d'une file

```c
typedef struct Element Element;
struct Element
{
    int nombre;
    Element *suivant;
};

typedef struct File File;
struct File
{
    Element *premier;
};
```

Implémentation d'une file

Enfilage

```c
int defiler(File *file)
{
    if (file == NULL)
    {
        exit(EXIT_FAILURE);
    }

    int nombreDefile = 0;

    /* On vérifie s'il y a quelque chose à défiler */
    if (file->premier != NULL)
    {
        Element *elementDefile = file->premier;

        nombreDefile = elementDefile->nombre;
        file->premier = elementDefile->suivant;
        free(elementDefile);
    }

    return nombreDefile;
}
```

Implémentation d'une file

Défilage

```c
void enfiler(File *file, int nvNombre)
{
    Element *nouveau = malloc(sizeof(*nouveau));
    if (file == NULL || nouveau == NULL)
    {
        exit(EXIT_FAILURE);
    }

    nouveau->nombre = nvNombre;
    nouveau->suivant = NULL;

    if (file->premier != NULL) /* La file n'est pas vide */
    {
        /* On se positionne à la fin de la file */
        Element *elementActuel = file->premier;
        while (elementActuel->suivant != NULL)
        {
            elementActuel = elementActuel->suivant;
        }
        elementActuel->suivant = nouveau;
    }
    else /* La file est vide, notre élément est le premier */
    {
        file->premier = nouveau;
    }
}
```