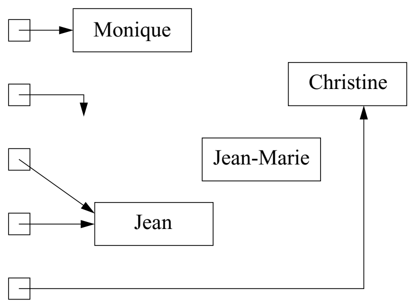
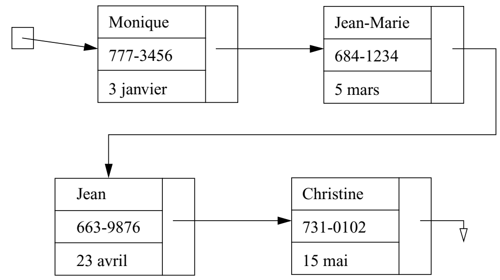
Listes chaînées.

Bornes fixes et le problème de débordement.

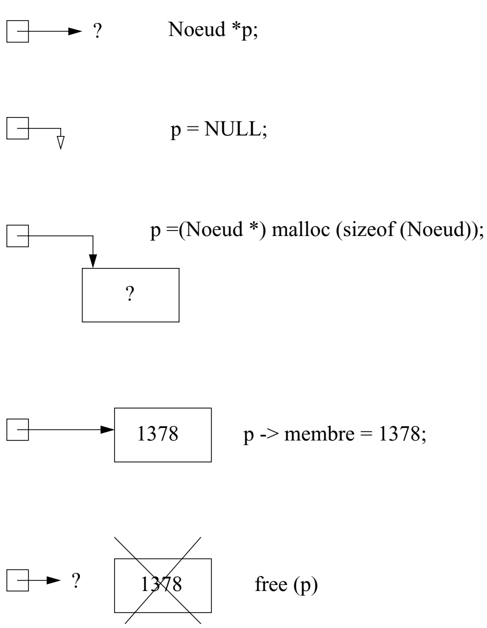
Pointeurs



Liste chaînée.



Exemple.



Déclaration d'un élément d'une liste chaînée.

```
typedef char Item;

typedef struct etiquette_noeud {
   Item info;
   struct etiquette_noeud * suiv;
} Noeud;

Noeud * tete;
```

Création d'un élément d'une liste.

```
typedef char Item;

typedef struct etiquette_noeud {
   Item info;
   struct etiquette_noeud * suiv;
} Noeud;

Noeud * tete;


Noeud * CreerNoeud (Item valeur)
{
   Noeud * p;

   if ((p = (Noeud *) malloc (sizeof (Noeud))) == NULL)
      Erreur ("Manque de memoire");
   else {
      p->info = valeur;
      p->suiv = NULL;
   return p;
}
```


Insertion d'un élément au début d'une liste chaînée.

```
q = CreerNoeud (val);
q->suiv = tete;
tete = q;
```

Création d'une liste chaînée.

```
tete = NULL;

q = CreerNoeud ('a');
q->suiv = tete;
tete = q;


q = CreerNoeud ('b');
q->suiv = tete;
tete = q;


q = CreerNoeud ('c');
q->suiv = tete;
tete = q;
```



```
Item val;
tete = NULL;

while ((val = getchar()) != '\n') {
   q = CreerNoeud (val);
   q->suiv = tete;
   tete = q;
}
```


Recherche d'une valeur dans une liste.

```
Noeud * Recherche (   Noeud * tete, /* pointeur de la tete de liste */
                      Item valeur) /* valeur cherchee */
{
   Noeud * courant; /* pointeur au noeud courant */

   for ( courant = tete;
         courant != NULL && courant->info != valeur;
         courant = courant->suiv) {}
   return (courant);
}
```

Parcours d'une liste chaînée.

```
    Noeud * courant; /* pointeur au noeud courant */

    for ( courant = tete;
            courant != NULL;
            courant = courant->suiv) {
        . . .
        /* traiter le noeud courant */
        . . .
    }
}
```

Insertion après un noeud.

```
/* InsererNoeud: inserer le noeud q apres le noeud p. */
void InsererNoeud (Noeud * p, Noeud * q)
{
    if (p == NULL || q == NULL)
        Error ("au moins un parmi les noeuds p et q n'existe pas");
    else {
        q->suiv = p->suiv;
        p->suiv = q;
    }
}
```


Insertion après un noeud contenant une valeur donnée.

```
void Inserer(        Noeud * tete, /* pointeur de la tete de liste */
                     Noeud * elem, /* element a inserer */
                     Item valeur) /* valeur après laquelle on insere */
{
    Noeud * courant; /* pointeur au noeud courant */

    for ( courant = tete;
            courant != NULL;
            courant = courant->suiv) {
        if (courant->info == valeur) {
            elem->suiv = courant->suiv;
            courant->suiv = elem;
            return;
        }
    };
    if (courant == NULL)
        Error ("pas d'element ayant la valeur cherchee");
    }
}
```


Insertion avant un noeud contenant une valeur donnée.

```
void Inserer(        Noeud * tete, /* pointeur de la tete de liste */
                     Noeud * elem, /* element a inserer */
                     Item valeur) /* valeur avant laquelle on insere */
{
    Noeud * courant; /* pointeur au noeud courant */

    if (tete->info == valeur) {
        elem->suiv = tete;
        tete = elem;
        return;
    }

    for ( courant = tete;
            courant->suiv != NULL;
            courant = courant->suiv) {
        if (courant->suiv != NULL && courant->suiv->info == valeur) {
            elem->suiv = courant->suiv;
            courant->suiv = elem;
            return;
        };
    }
    if (courant == NULL)
        Error ("pas d'element ayant la valeur cherchee");
}
```


Insertion avant un noeud contenant une valeur donnée (version pleine).

```
void Inserer(
        Noeud * * adrtete, /* pointeur au pointeur de la tete de liste */
        Noeud * elem, /* element a inserer */
        Item valeur) /* valeur après laquelle on insere */
{
    Noeud * courant; /* pointeur au noeud courant */

    if ((*adrtete)->info == valeur) {
        elem->suiv = *adrtete;
        *adrtete = elem;
        return;
    }

    for ( courant = *adrtete;
            courant->suiv != NULL;
            courant = courant->suiv) {
        if (courant->suiv->info == valeur) {
            elem->suiv = courant->suiv;
            courant->suiv = elem;
            return;
        };
    }
    if (courant == NULL)
        Erreur ("pas d'element ayant la valeur cherchee");
}
```

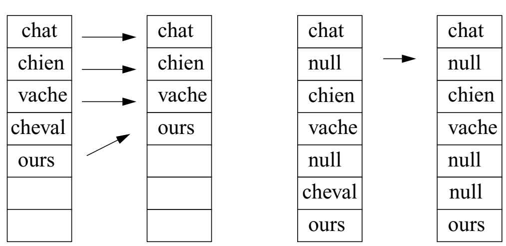
Suppression d'un élément d'une liste.

```
/* Supprimer: supprime le noeud p de la liste; r doit preceder p */
void Supprimer (Noeud * r, Noeud * p)
{
   if (r == NULL || p == NULL)
      Erreur ("Au moins un noeud parmi p et r n'existe pas");
   else if (r->suiv != p)
      Erreur (" Les noeuds r et p ne se suivent pas");
   else
      r->suiv = p->suiv;
}
```


Listes comme des types abstraits de données.

```
status creer_liste (Noeud * * adrtete);
bool liste_vide (Noeud * tete);
status ajouter(Noeud * * adrtete, Item valeur);
status inserer (Noeud * * adrtete, Item valeur);
status supprimer (Noeud * * adrtete, Item valeur);
```


Implémentation à l'aide de tableaux.



compte=5 compte=4

1-ère implémentation 2-ème implémentation

Suppression dans une liste