Liste simplement chainée

Définition

- Une liste est un containeur tel que le nombre d'objets (dimension ou taille) est variable. Les objets sont ordonnés.
- L'accès aux objets se fait indirectement par le contenu d'un curseur dont le type dépend de l'implémentation
- Chaque élément de la liste contient une information permettant d'atteindre l'élément suivant.
- Le dernier élément de la liste n'a pas de suivant c'est ce qui le caractérise.

Implémentation : exemple dynamique

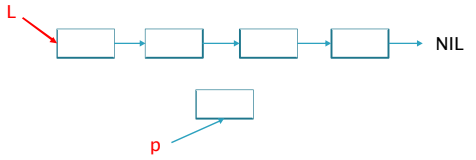
```
Le curseur est un pointeur vers un objet de la
liste.La liste est un curseur
```

Je dessine

- La représentation de listes chaînées à l'aide du diagramme avec une flèche vers le suivant a été proposée Newell & Shaw dans l'article "Programming the Logic Theory Machine" Proc. WJCC February1957

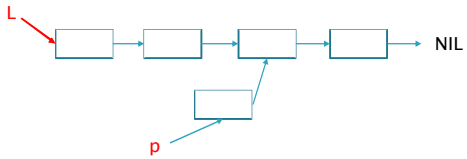
Je dessine

- Ajouter un élément



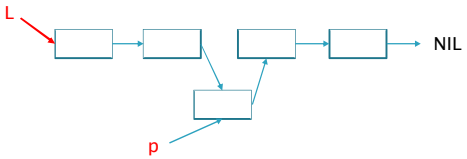
Je dessine

- Ajouter un élément



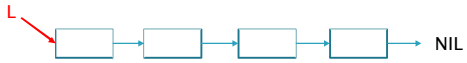
Je dessine

- Ajouter un élément



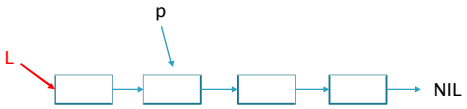
Je dessine

- supprimer un élément



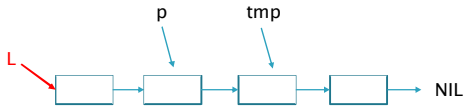
Je dessine

- supprimer un élément



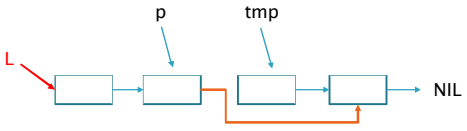
Je dessine

- supprimer un élément



Je dessine

- supprimer un élément



Je dessine

- supprimer un élément

Définition du type abstrait listeSC

Une liste est dite simplement chainée si les opérations suivantes s'effectuent en O(1).

Accès

```
fonction premier(L: Liste):curseur;
fonction suivant(L: Liste;  P:curseur):curseur;
fonction listeVide(L:Liste):booléen;
Fonction valeur(L: liste):objet;
```

Modification

```
fonction creerliste(L:Liste):vide;
fonction insererApres(L:Liste;  x:objet; P:curseur):vide;
fonction insererEnTete(L:Liste;  x:objet):vide;
fonction supprimerApres(L:Liste; P:curseur):vide;
fonction supprimerEnTete(L:Liste):vide;
```

Quelques algorithmes

Chercher un élément dans une liste

```
fonction chercher(L: Liste; E:objet):curseur;
  var p: curseur;
  début
    si listeVide(L) alors
       retourner(NIL)
    sinon
      p=premier(L);
      tant que non(suivant(L,p)!=Nil) et (valeur(p)!=e) faire
             p=suivant(L,p);
      fintantque
      si (valeur(p)!=e) alors
        retourner(NIL)
      sinon
        retourner(p)
      finsi
    finsi
  fin
```

Complexité: O(n).

Quelques algorithmes

Chercher un élément dans une liste

```
fonction chercher(L: Liste; E:objet):curseur;
  var p: curseur;
  début
    si listeVide(L) alors
       retourner(NIL)
    si valeur(L)==E alors
       retourner(L)
    chercher(suivant(L,premier(L)),E)
```

Complexité: O(n).

Quelques algorithmes

Trouver le dernier élément

```
fonction trouverDernier(L:liste):curseur;
  var p: curseur;
  debut
    si listeVide(L) alors
      retourner(NIL)
    sinon
      p=premier(L);
      tant que non(suivant(L,p)!=Nil) faire
          p=suivant(L,p);
      fintantque
      retourner(p)
    finsi
  fin
```

Complexité: O(n).

Quelques algorithmes

Trouver le dernier élément

```
fonction trouverDernier(L:liste):curseur;
  var p: curseur;
  debut
    si listeVide(L) alors
      retourner(NIL)
    finsi
    p=premier(L);
    si(suivant(L,p)==Nil)
      retourner(p)
    finsi
    trouverDernier(suivant(L,p))
  fin
```

Complexité: O(n).

Quelques algorithmes

Calculer la taille d'une liste

```
fonction taille( L:Liste):entier;
  var p;curseur;
  var t:entier;
  debut
    si listeVide(L) alors
      retourner(0)
    sinon
      retourner(1+taille(suivant(L,premier(L))))
    finsi
  fin
finfonction
```

Complexité: O(n).

Quelques algorithmes

Insérer dans une liste triée
On suppose la liste triée dans l'ordre croissant

```
fonction insertionTrie(L:liste; e: objet):vide;
  var p:curseur;
  début
    si listeVide(L) alors
      insererEnTete(L,e)
    sinon
      si valeur(premier(L))>e alors
        insererEnTete(L,e)
      sinon
        insererTrie(suivant(L,premier(L)),e)
      finsi
    finsi
  fin
```

Complexité: O(n).