Plan du Cours

- Allocation Dynamique - Introduction
- Introduction au concept de Liste
  - Implémentation : Classe TabListe
  - Problèmes liés à l'utilisation d'un tableau
- Listes Chaînées (Simples)
  - Introduction
  - Ajout
  - Suppression
  - Maillon Tête Factice
- Accès
- Listes Chaînées Doubles
  - Insertion
  - Suppression
- Listes Chaînées Circulaires

Accès à un Elément d'une Liste Chaînée

- Deux types d'accès
  - **par position**
    - on connaît le rang **k** de l'élément dans la liste
      - aucun intérêt pour un tableau (accès direct)
    - sur n'importe quel type de liste, triée ou non
  - **associatif**
    - on connaît la valeur **val** de l'élément et on cherche la première occurrence
    - le type de liste, triée ou non triée, aura son importance

Accès au k^{ième} Elément (Calcul du Résultat)

- while cour != None and i < k : ...
- Tableau de sortie

i = k	cour = None	résultat
vrai	vrai	trouve = faux
vrai	faux	trouve = vrai ; valk = cour._valeur
faux	vrai	trouve = faux
faux	faux	impossible

- *trouve* est vrai lorsque cour != None

Fonction accesk

Spécification

```
def accesk (self,k)
"""
 :entrée self:
 :entrée k: int
 :sortie trouve: bool
 :sortie valk: object
 :post-cond: si le kème élément existe, trouve est vrai et
    valk stocke sa valeur
 """
```

Fonction accesk

Spécification

```
def accesk (self,k)
"""
 :entrée self:
 :entrée k: int
 :sortie trouve: bool
 :sortie valk: object
 :post-cond: si le k ème élément existe, trouve est vrai et valk stocke sa valeur
 """
```

Implémentation

```
def accesk (self, k) :
    trouve = False
    valk = None
    if k < 1 :
        return trouve, valk
    else :
        cour = self._premier
        # ou cour = self._tete._suiv si implémentation avec tete factice
        i = 1
        while cour != None and i < k :
            cour = cour._suiv
            i = i+1
        if cour != None :
            trouve = True
            valk = cour._valeur
        return trouve, valk
```

Fonction accesval

Spécification

```
def accesval (self,val)
  """
  :entrée self:
  :entrée val: object
  :sortie trouve: bool
  :post-cond: si l'élément val existe, trouve est True
  """
```

Fonction accesval

Spécification

```
def accesval (self,val)
  """
  :entrée self:
  :entrée val: object
  :sortie trouve: bool
  :post-cond: si l'élément val existe, trouve est True
  """
```

Implémentation

```
def accesval (self,val) :
    cour = self._premier
    # ou cour = self._tete._suiv si implémentation avec tête factice
    trouve = False
    while cour != None and cour._valeur != val:
            cour = cour._suiv
    if cour != None:
            trouve = True
    return trouve
```

Classe LCListe : Interface

```
def __init__(self):
 """
 :sortie self:
 :post-cond: attributs premier (ou tete) et dernier
  initialisés et initialisation nb éléments
 """

def ajout(self,x):
 """
 :entrée-sortie self:
 :entrée x: object
 :pré-cond: aucune
 :post-cond: ajout de x à la liste
 """

def retourner_pos(self,x):
 """
 :entrée self:
 :entrée x: object
 :sortie i: int
 :pré-cond: l'élément x se trouve dans la liste
 :post-cond: i est sa position
 """
```

```
def set(self,id,nouvVal):
 """
 :entrée-sortie self:
 :entrée id: int
 :entrée nouvVal: object
 :sortie vieilleVal: object
 :pré-cond: l'élément à modifier est à la position id
 :post-cond: retourne l'ancien élément à la position id
 """

def suppr(self,id):
 """
 :entrée self:
 :entrée id: int
 :sortie supprVal: object
 :pré-cond: l'élément à supprimer est à la position id
 :post-cond: retourne l'élément supprimé supprVal
 """
```

Classe LCListe : Implémentation (suite)

```
def retourner_pos (self, x):
  cour = self._premier
  # ou cour = self._tete._suiv
  i = 1
  while cour._valeur != x:
    cour = cour._suiv
    i = i+1
  return i

def set (self, id, nouvVal):
  cour = self._premier
  # ou cour = self._tete._suiv
  i = 1
  while i < id:
    cour = cour._suiv
    i = i + 1
  vieilleVal = cour._valeur
  cour._valeur = nouvVal
  return vieilleVal
```

```
def suppr (self, id):
  prec = None
  cour = self._premier
  i = 1
  while i < id:
    prec = cour
    cour = cour._suiv
    i = i+1
  supprVal = cour._valeur
  if prec == None:
    self._premier = cour._suiv
  else:
    prec._suiv = cour._suiv
  if cour == self._dernier:
    self._dernier = prec
  self._nbElem = self._nbElem – 1
  return supprVal
```