CAPES MATHS OPTION INFORMATIQUE

LISTE CHAINEES, PILES ET FILES

Classe Liste chaînée

Construire une classe implémentant les listes doublement chaînées non circulaires incluant :

- un constructeur
- deux attributs premier et dernier qui sont des cellules
- des méthodes estVide, nbElements, iemeElement, modifierIemeElement, afficher, ajouterEnTete, ajouterEnQueue, supprimerTete, rechercherElement et insererElement

Vous définirez pour cela une classe-structure Cellule composée de trois champs : info, suivant et precedent.

```
class Cellule:
    def __init__(self, info, suivant, precedent):
        self.info = info
        self.suivant = suivant
        self.precedent = precedent

class ListeChainee:
    """ Liste doublement chaînée non circulaire"""

    def __init__(self):
        self.premier = None
        self.dernier = None

    def estVide(self):
        return self.premier == None

    def nbElements(self):
        nb = 0
        elt = self.premier
        while elt:
            nb += 1
            elt = elt.suivant
        return nb

    def iemeElement(self, indice):
        elt = self.premier
        for i in range(indice):
            elt = elt.suivant
            if elt == None:
                print("Erreur iemeElement : indice trop grand")
        return elt.info

    def modifierIemeElement(self, indice, infoElement):
        elt = self.premier
        for i in range(indice):
            elt = elt.suivant
            if elt == None:
                print("Erreur modifierIemeElement : indice trop grand")
        elt.info = infoElement

    def afficher(self):
        elt = self.premier
        while elt:
            print(elt.info, end=' ')
            elt = elt.suivant
        print()
```

```
    def ajouterEnTete(self, infoElement):
        elt = Cellule(infoElement, self.premier, None)
        if self.estVide():
            self.dernier = elt
        else:
            self.premier.precedent = elt
        self.premier = elt

    def supprimerTete(self):
        elt = self.premier
        self.premier = elt.suivant
        if self.premier:
            self.premier.precedent = None
        else:
            last = None

    def ajouterEnQueue(self, infoElement):
        if self.estVide():
            self.ajouterEnTete(infoElement)
        else:
            elt = Cellule(infoElement, None, self.dernier)
            self.dernier.suivant = elt
            self.dernier = elt

    def rechercheElement(self, infoElement):
        elt = self.premier
        trouve = False
        pos = 0
        while elt and not trouve:
            if infoElement == elt.info:
                trouve = True
            else:
                elt = elt.suivant
                pos += 1
        if trouve:
            return pos
        else:
            return -1

    def insererElement(self, indice, infoElement):
        if self.estVide() or indice == 0:
            self.ajouterEnTete(infoElement)
        elif indice == self.nbElements():
            self.ajouterEnQueue(infoElement)
        else:
            elt = self.premier
            for _ in range(indice): elt = elt.suivant
            nvelt = Cellule(infoElement, elt, elt.precedent)
            elt.precedent.suivant = nvelt
            elt.precedent = nvelt
```

Classe Pile

Construire une classe implémentant les piles (sans utiliser le module queue de Python), incluant :

- un constructeur de paramètres self et une liste L
- un attribut liste
- la surcharge de la méthode d'affichage
- des méthodes empiler, depiler, sommet, estVide, hauteur et pop

```
class Pile(object) :

   def __init__(self,L) :
      self.liste = L

   def __repr__(self) : #ou __str__
      s = 'pile: '
      for e in self.liste :
         s += str(e) + ' '
      return s

   def empiler (self,e) :
      self.liste.insert(0,e)

   def depiler(self) :
      del self.liste[0]

   def sommet (self) :
      return self.liste[0]

   def estVide (self) :
      return len(self.liste) == 0

   def hauteur (self) :
      return len(self.liste)

   def pop (self) :
      e = self.liste[0]
      del self.liste[0]
      return e
```

Classe File

Construire une classe implémentant les files (sans utiliser le module queue de Python), incluant :

- un constructeur de paramètres self et une liste L
- un attribut liste
- la surcharge de la méthode d'affichage
- des méthodes pop, push, estVide et longueur

```
class File(object) :

   def __init__(self,L) :
      self.liste = L

   def __repr__(self) : #ou __str__
      s = 'file: '
      for e in self.liste :
         s += str(e) + ' '
      return s

   def pop (self) :
      e = self.liste[-1]
      del self.liste[-1]
      return e

   def push (self,e) :
      self.liste.insert(0,e)

   def estVide (self) :
      return len(self.liste) == 0
```

```
   def longueur (self) :
      return len(self.liste)
```

Inversion d'une File en utilisant une Pile

Le but de cet exercice est d'écrire en Python une procédure qui inverse une file d'entiers qui lui est passée en paramètre. On demande de ne pas utiliser de tableau ou de liste de travail pour effectuer l'inversion, mais d'utiliser plutôt une pile. Il existe en effet une méthode très simple pour inverser une file en utilisant une pile.

```
def inverserFile (file) :
   pile = Pile()
   while not file.estVide() :
      pile.empiler(file.pop())
   while not pile.estVide() :
      file.push(pile.pop())
```

Validité du parenthésage d'une expression

Un problème fréquent pour les compilateurs et les traitements de textes est de déterminer si les parenthèses d'une chaîne de caractères sont équilibrées et proprement incluses les unes dans les autres. On désire donc écrire une fonction qui teste la validité du parenthésage d'une expression :

– on considère que les expressions suivantes sont valides : "()", "[([bonjour+]essai)7plus-];"

– alors que les suivantes ne le sont pas : "(", ")(", "4(essai]".

Notre but est donc d'évaluer la validité d'une expression en ne considérant que ses parenthèses et ses crochets. On suppose que l'expression à tester est dans une chaîne de caractères, dont on peut ignorer tous les caractères autres que '(', '[', ']' et ')'. Écrire en Python la fonction `valide` qui renvoie vrai si l'expression passée en paramètre est valide, faux sinon.

```
def valide (ch) :
   pile = Pile()
   i = 0
   for car in ch :
      if car == '(' or car == '[' :
         pile.empiler(car)
      elif car == ')' :
         if not pile.estVide() :
             sommet = pile.pop()
             if sommet != '(' :
                return False
         else :
            return False
      elif car == ']' :
         if not pile.estVide() :
            sommet = pile.pop()
            if sommet != '[' :
               return False
         else :
             return False
   return pile.estVide()
```