```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

#
# fichier: polynome.py
#    date: 2011/05/03
#
# (tous les symboles non internationaux sont volontairement omis)
#

from string import *

from entier import *
from monome import *
from noeud import *
from rationnel import *

class polynome(object):
  """ classe pour un polynome (structure d'arbre binaire de recherche) """

  def __init__(self, valide =True):
    """ constructeur (polynome nul) """
    self.__valide = valide
    self.__tete = noeud(monome())



  def __str__(self):
    """ representation en chaine de caracteres """
    if self.__valide:
      return str(self.__tete)
    else:
      return "(polynome invalide)"



  def est_valide(self):
    """ indique l'etat de validite """
    return self.__valide



  def valider(self):
    """ valider l'objet """
    self.__valide = True



  def invalider(self):
    """ invalider l'objet """
    self.__valide = False



  def chercher(self, monome):
    """ rechercher un monome """
    if self.__tete:
      return self.__tete.chercher(monome)
    else:
      return False



  def inserer(self, monome):
    """ inserer un nouveau monome """
    self.__valide = self.__valide and monome.est_valide()
    if self.__valide:
      if self.__tete:
```

```
        self.__tete.inserer(monome)
      else:
        self.__tete = noeud(monome)



  def supprimer(self, monome):
    """ supprimer un monome """
    if self.chercher(monome):
      if self.__tete.get_monome() == monome:
        if self.__tete.fils_gauche():
          fils = self.__tete.fils_gauche()
        else:
          fils = self.__tete.fils_droite()
        if fils is None:
          self.__tete = None
        else:
          if self.__tete.fils_gauche():
            x = fils.plus_grand()
          else:
            x = fils.plus_petit()
          self.__tete.supprimer(x)
          self.__tete.set_monome(x)
      else:
        self.__tete.supprimer(monome)
    else:
      print "(monome inexistant)"



  def retrouver(self, monome):
    """ retrouver un monome (existant) """
    if self.chercher(monome):
      return self.__tete.retrouver(monome)
    else:
      return None



  def degre(self):
    """ donne le degre du polynome """
    if self.__tete:
      return len(self.__tete.plus_petit().get_indeterminee())
    else:
      """ concession a la definition mathematique du degre du polynome nul """
      return (-1)



  def valuation(self):
    """ donne la valuation du polynome """
    if self.__tete:
      return self.__tete.plus_grand().get_coefficient()
    else:
      return rationnel()



  def ajouter_monome(self, m):
    """ adjoindre un nouveau monome au polynome """
    if m.get_coefficient().est_nul():
      return
    if self.chercher(m) == True:
      x = self.retrouver(m)
      a = x.get_coefficient()
      b = m.get_coefficient()
      self.supprimer(x)
      r = a + b
```

```
      if not r.est_nul():
        self.inserer(monome(r, m.get_indeterminee()))
    else:
      self.inserer(m)
    """ suppression du monome nul (si besoin) """
    if self.valuation().est_nul() and self.degre() > 0:
      self.supprimer(monome())



  def monome_plus_grand_degre(self):
    """ donne le monome de plus haut degre """
    if self.__tete:
      """ plus petit == plus a gauche dans l'ABR """
      return self.__tete.plus_petit()
    else:
      return None



  def monome_plus_petit_degre(self):
    """ donne le monome de plus petit degre """
    if self.__tete:
      """ plus grand == plus a droite dans l'ABR """
      return self.__tete.plus_grand()
    else:
      return None



  def liste_decroissante(self):
    """ liste des monomes en ordre decroissant """
    if self.__tete:
      return self.__tete.liste_decroissante()
    else:
      return []



  def nombre_monomes(self):
    """ nombre de monomes du polynome """
    if self.__tete:
      return self.__tete.nombre_monomes()
    else:
      return 0



  def __add__(self, autre):
    """ addition """
    if self.__valide and autre.__valide:
      p = polynome()
      a = self.liste_decroissante()
      for m in a:
        p.ajouter_monome(m)
      b = autre.liste_decroissante()
      for m in b:
        p.ajouter_monome(m)
      return p
    else:
      return polynome(False)



  def __neg__(self):
    """ oppose du polynome """
    if self.__valide:
      p = polynome()
```

```
      a = self.liste_decroissante()
      for m in a:
        p.ajouter_monome(-m)
      return p
    else:
      return polynome(False)



  def __sub__(self, autre):
    """ soustraction """
    return self.__add__(autre.__neg__())



  def __mul__(self, autre):
    """ produit de deux polynomes """
    if self.__valide and autre.__valide:
      p = polynome()
      a = self.liste_decroissante()
      b = autre.liste_decroissante()
      for m in a:
        for n in b:
          p.ajouter_monome(monome.produit(m, n))
      return p
    else:
      return polynome(False)



  def __pow__(self, autre):
    """ exponentiation d'un polynome (exposant entier naturel) """
    if not(self.__valide and autre.__valide):
      return polynome(False)

    if autre.degre() < 0:
      return polynome(False)

    v = autre.valuation()
    if not v.est_entier():
      return polynome(False)

    n = v.get_num().valeur()
    if self.valuation().est_nul():
      if n < 0:
        return polynome(False)
      else:
        p = polynome()
        p.ajouter_monome(monome(rationnel(1)))
        return p

    if n < 0:
      return polynome(False)

    p = polynome()
    a = self
    p.ajouter_monome(monome(rationnel(1)))
    while n > 0:
      if n % 2 == 1:
        p *= a
      n //= 2
      a *= a
    return p



  def est_nul(self):
    """ le polynome est-il nul ? """
```

```
    if self.__valide:
      return (self.degre() == 0) and (self.valuation().est_nul())
    else:
      return False



  def est_unite(self):
    """ le polynome est-il egal a 1 ? """
    if self.__valide:
      return (self.degre() == 0) and self.valuation().est_unite()
    else:
      return False



  def pgcd_numerateurs(self):
    """ pgcd des numerateurs des coefficients des monomes """
    l = []
    if self.__valide:
      for m in self.liste_decroissante():
        e = abs(m.get_coefficient().get_num().valeur())
        if not (e in l):
          l.append(e)
    return pgcd_liste(l)



  def ppcm_denominateurs(self):
    """ ppcm des denominateurs des coefficients des monomes """
    l = []
    n = 1
    if self.__valide:
      for m in self.liste_decroissante():
        """ les denominateurs sont positifs """
        e = m.get_coefficient().get_denom().valeur()
        if not (e in l):
          l.append(e)
        n *= e
    return n / pgcd_liste(l)



def polynome_nul():
  """ le polynome nul """
  p = polynome()
  return p



def polynome_un():
  """ le polynome unite """
  p = polynome()
  p.ajouter_monome(monome(rationnel(1)))
  return p



if __name__ == "__main__":

  print "polynome a (invalide)"
  a = polynome()
  print a
  a.ajouter_monome(monome(rationnel(1), "xx"))
  t = monome(rationnel(4, -6, False), "x")
  a.ajouter_monome(t)
  a.ajouter_monome(monome(rationnel(-5)))
  print a
```

```
  print "\n"

  print "polynome a"
  a = polynome()
  print a
  a.ajouter_monome(monome(rationnel(1), "xx"))
  t = monome(rationnel(4, -6), "x")
  a.ajouter_monome(t)
  a.ajouter_monome(monome(rationnel(-5)))
  print a
  m = monome(rationnel(2), "x")
  a.ajouter_monome(m)
  print a
  a.ajouter_monome(monome(rationnel(4, -3), "x"))
  print a
  print a.degre()
  print a.valuation()
  print "\n"

  print "polynome b"
  b = polynome()
  b.ajouter_monome(monome(rationnel(1), "a"))
  print b
  b.ajouter_monome(monome(rationnel(7, -5), "a"))
  print b
  b.ajouter_monome(monome(rationnel(-5)))
  print b
  print "\n"

  print "polynome s (somme)"
  s = a + b
  print s
  print "\n"

  print "polynome d (difference)"
  d = a - b
  print d
  print "\n"

  print "polynome p (produit)"
  p = a * b
  print p
  print "\n"

# print p.monome_plus_grand_degre()
# print p.monome_plus_petit_degre()

# for c in p.liste_coefficients():
#   print c

# for c in p.liste_denominateurs():
#   print c
# print

# print p.ppcm_denominateurs()
# print

  print "polynome a"
  a = polynome()
  a.ajouter_monome(monome(rationnel(560), "x"))
  a.ajouter_monome(monome(rationnel(120)))
  print a
  print "polynome n"
  n = polynome()
  n.ajouter_monome(monome(rationnel(4)))
  print n
  print "polynome p (puissance)"
  p = a ** n
```

```
  print p
  print "\n"

# print "polynome n"
# n = polynome()
# n.ajouter_monome(monome(rationnel(-4)))
# print n
# print "polynome p (puissance)"
# p = a ** n
# print p
# print "\n"

# print "polynome n"
# n = polynome()
# print n
# print "polynome p (puissance)"
# p = a ** n
# print p
# print "\n"
```