```
class Maillon:
                _(self, v, s): \# modification énoncé
   def __init_
        self.valeur = v
        self.suivant = s
                               # modification énoncé
class File:
    def __init__(self):
        self.dernier_file = None
    def enfile(self, element):
    nouveau_maillon = Maillon(element , self.dernier_file)
        self.dernier_file = nouveau_maillon
    def est_vide(self):
        return self.dernier_file == None
    def affiche(self):
        maillon = self.dernier_file
        while maillon != None:
            print (maillon.valeur)
            maillon = maillon.suivant
    def defile(self):
        if not self.est_vide():
            if self.dernier_file.suivant == None:
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = self.dernier_file
            while maillon.suivant.suivant != None:
                maillon = maillon.suivant
            resultat = maillon.suivant.valeur
            maillon.suivant = None
            return resultat
        return None
# def validation():
      tests = [] # valide si tous les éléments sont True
#
      F = File()
      # test_1 : tester une file vide
      tests.append(F.est_vide() == True)
      F.enfile(2)
      # print_1 : '2'
      F.affiche()
      # test_2 : file non vide
      tests.append(F.est_vide() == False)
      F.enfile(5)
      F.enfile(7)
      # print_2 : '7\n5\n2'
      F.affiche()
      # test_3 : defiler
      tests.append(F.defile() == 2)
      # test_4 : defiler
      tests.append(F.defile() == 5)
      # print_3 : affichage final -> [True, True, True, True]
      print(tests)
# validation()
import unittest
class correction(unittest.TestCase):
```

```
def test_creation(self):
        F = \overline{File}()
        self.assertEqual(F.est_vide(), True)
    def test_enfile(self):
        F = File()
        F.enfile(2)
        self.assertEqual(F.est_vide(), False)
    def test_defile(self):
        F = \overline{File}()
        F.enfile(2)
        F.enfile(5)
        F.enfile(7)
        self.assertEqual(F.defile(), 2)
        self.assertEqual(F.defile(), 5)
self.assertEqual(F.defile(), 7)
        self.assertEqual(F.est_vide(), True)
    def test_alea(self):
         from random import randint
        n = 1_{000}
        tab = [randint(-1000, 1000)] for _ in range(n)]
        F = File()
        for x in tab:
             F.enfile(x)
             self.assertEqual(F.est_vide(), False)
         for x in tab:
             self.assertEqual(x, F.defile())
         self.assertEqual(F.est_vide(), True)
if __name__ == '__main__':
    unittest.main()
```