TD Si2 FIP - Groupe IE4 (A/B)

Documents de travail pour les mini-groupes

Constituez des groupes de 5 ou 6 etudiants svp

1. Beaumois - Bellay - Celton - Kesteman - Jean-Louis (2 places restantes) https://codimd.math.cnrs.fr/s/fprRIB_lX#
2. Gédéon - Balitrand - Blavec - Artus (2 places restantes)https://codimd.math.cnrs.fr/s/UFBNClrJC
3. Ballanger - Orhan - Calba - Hermenier - Guegan - Comte (complet) : https://codimd.math.cnrs.fr/s/HqmycdOov
4. El Mellali - Tourang - Mahamoud - Zhang - Adam (complet) : https://codimd.math.cnrs.fr/s/ktpQOR5cK
5. Mimouni - Souchet - Lechevalier - Kerguiduff - Hamila - Ginguenet (complet) : https://codimd.math.cnrs.fr/s/dUjPrSrVz
6. Bayo - Aoutin - Godec Guihard Ileri Berthet https://codimd.math.cnrs.fr/s/Bk9r3SzSP

Exo 1

```
/**
 * @param l un list d'entier
 * @return somme des elements dans la list
 */
def somme(l: List[Int]): Int = {
  l match {
    case Nil => 0
    case a::b => a + somme(b)
  }
}

val list1 :List[Int] = Nil
val list2 : List[Int] = 1::2::3::Nil

println(somme(list1)) // affiche 0
println(somme(list2)) // affiche 6
assertEquals(0 , somme(list1)) // affiche rien si assertion est verifiee,
exception si assertion est non verifiee
```

Exo 2 - filtrerSup

```
  /**
    *@param l une liste d'entier
    *@param n un entier
    *@return la liste composée des éléments de l > n
    */
    def filtrerSup(l: List[Int], n: Int): List[Int] = {
        l match{
            case Nil => Nil
            case e::rest => if(e > n) e::filtrerSup(rest, n)
                            else filtrerSup(rest, n)
        }
    }

    val l: List[Int] = 3::5::1::8::4::9::Nil
    filtrerSup(l) // vaut 5::8::9::Nil
    assertEquals(5::8::9::Nil,filterSup(l))
```

Exo 3

```
ici le code
```

```
val exo3 : List[Int] = 2 :: 5 :: 1 :: 7 :: 9 :: Nil

/**
 * @param l une liste d'entiers non vide
 * @return le plus grand élément de la liste l
 */
def maxListe(l : List[Int]): Int ={
  l match{
    case n1 :: Nil => n1
    case n1 :: l1 => {
        if (n1 > maxListe(l1)) { n1 }
        else { maxListe(l1)}

    }
  }
}
println(maxListe(exo3))
//on attend l'entier 9 en résultat
```

```
   /**
    * @param l une liste d'entier non vide
    * @return le plus grand élément de l
    */
   def maxListe(l: List[Int]): Int = {
       l match {
           case v1 :: Nil => v1
           case v1 :: rest => {
               val maxrest: Int = maxListe(rest)
               if (v1 > maxrest) v1
               else maxrest
           }
       }
   }

   val ListTest: List[Int] = List(1, 0, -2, 4, 2)

   print(maxListe(ListTest))
   //Retourne l'entier 24
```

Exo 4

```
  /**
    *@param l une liste de paires de String
    *@return la liste des deuxième éléments de chaque paire de l
    */
   def projection(l: List[(String,String)]): List[String] = {
       l match {
           case Nil => Nil
           case (_,s) :: rest => s :: projection(rest)
       }
   }

     val ls: List[(String,String)] = ("ford","mercedes")::
("peugeot","ferrari")::("fiat","audi")::Nil
     projection(ls) // vaut "mercedes"::"ferrari"::"audi"::Nil
     assertEquals("mercedes"::"ferrari"::"audi"::Nil, projection(ls))
```

Exercice 9

```
    val arbre: Abin =
        Noeud("RE",
              Noeud("VOLTE",
                    Noeud("ES", Vide, Vide),
                    Vide
                   ),
              Noeud("VE", Vide, Vide))


   val es: Abin = Noeud("ES", Vide, Vide)
   val ve: Abin = Noeud("VE", Vide, Vide)

   val volte: Abin = Noeud("Volte", es, Vide)

   val a: Abin = Noeud("RE", volte, ve)
```

Exercice 10

```
/**
 * @param a un arbre binaire
 * @return la hauteur de a
 */
def hauteur(a: Abin) : Int =
   a match {

      case Vide => 0

      case Noeud(_, g, d) =>  {
          val hg: Int = hauteur(g)
          val hd: Int = hauteur(d)
          1 + max(hg, hd)
      }



   }
```