Programowanie współbieżne
Lista 1

1. Napisz funkcję reverse: [A](xs: List[A]) List[A], odwracającą zadaną listę,
   np. reverse(List("Ala", "ma", "kota")) == List(kota, ma, Ala)
   Nie wolno używać żadnych funkcji bibliotecznych (oprócz konstruktorów list).

2. Napisz funkcję exists: [A] (xs: List[A]) (p: A => Boolean) Boolean.
   exists (xs) (p) ma wartość logiczną zdania „$\exists x \in xs.p(x)$"
   np. exists (List(5,1,2,3)) (_ == 2) == true

   Należy napisać trzy wersje tej funkcji:
   a) z wykorzystaniem dopasowania do wzorca i rekursji,
   b) z wykorzystaniem metody List.foldLeft,
   c) z wykorzystaniem metody List.foldRight.

Na wykładzie zostały zdefiniowane drzewa binarne:

```
sealed trait BT[+A]
case object Empty extends BT[Nothing]
case class Node[+A](elem:A, left:BT[A], right:BT[A]) extends BT[A]

val t = Node(1, Node(2, Empty, Node(3, Empty, Empty)), Empty)
```

3. Napisz funkcję sumBT: (bt: BT[Int])Int, która oblicza sumę liczb całkowitych, przechowywanych w węzłach drzewa.
   np. sumBT(t) == 6

4. Napisz funkcjonał
   foldBT: [A, B](f: A => (B, B) => B)(acc: B)(bt: BT[A])B
uogólniający funkcję sumowania wartości z węzłów drzewa binarnego tak, jak funkcjonał foldRight uogólnia funkcję sumowania elementów listy.

5. Wykorzystaj foldBT do zdefiniowania:
   a) sumy liczb całkowitych sumBTfold: (bt: BT[Int])Int
      np. sumBTfold(t) == 6
   b) listy wartości pamiętanych w węzłach drzewa (w obejściu infiksowym).
      np. inorderBTfold(t) == List(2, 3, 1)

6. Wykorzystując foldBT zdefiniuj funkcjonał
   mapBT: [A, B](f: A => B)(tree: BT[A])BT[B]
   aplikujący daną funkcję do wartości we wszystkich węzłach drzewa.
   np. mapBT((v: Int) => 2 * v)(t: BT[Int]) == Node(2,Node(4,Empty,Node(6,Empty,Empty)),Empty)