

- ฟังก์ชันต่าง ๆ ให้เขียนแบบ Recursive เท่านั้น ห้ามใช้ loop ถ้าไม่เขียนด้วย recursion จะได้ 0 คะแนนในข้อนี้ ๆ
- อนุญาตให้ใช้ เมธอดของลิสต์ได้แค่ isEmpty, length, head, tail, ::, ++ เท่านั้น ใครใช้เกินมา จะได้ 0 คะแนนในข้อนี้ ๆ
- เขียนเมธอดใหม่เองจากเมธอดพื้นฐานที่อนุญาตข้างต้นได้
- ให้แยกหนึ่งข้อต่อหนึ่งไฟล์ ตั้งชื่อไฟล์ตามข้อ เช่น Question01.scala ให้เป็นของฟังก์ชัน insertLast
- ในแต่ละข้อให้เขียน main เพื่อทดสอบได้ตามใจ อาจารย์จะตรวจโดยใช้ main ของอาจารย์เอง
- การส่ง ส่งวันที่ 20 มีนาคม โดย zip ทุกไฟล์รวมกัน แล้วตั้งชื่อ zip file เป็น ID_scalaHW01 เช่น 6332011421_scalaHW01

1. จงเขียนฟังก์ชัน `def insertLast(x:Any, l:List[Any]) : List[Any] = {` ซึ่งได้คำตอบเป็นลิสต์ที่เกิดจากการเอา x ไปใส่ไว้ท้าย l
2. จงเขียนฟังก์ชัน `def insertInOrder(x:Int, list:List[Int]):List[Int] = {` ซึ่งเกิดจากการเอา list ที่ sort จากน้อยไปมากมาใส่ x ลงไป โดย ลิสต์ที่รีเทิร์นมาต้องยังมีการเรียงจากน้อยไปมากอยู่
3. จงเขียนฟังก์ชัน `def subList(l1:List[Any], l2:List[Any]):Boolean = {` ซึ่งฟังก์ชันนี้รีเทิร์น true เมื่อ สมาชิกทั้งหมดของ l1 อยู่ใน l2 ข้อมูลซ้ำกันในลิสต์เดียวกันไม่เป็นไร ไม่สนจำนวน (ลิสต์ว่างเป็น sublist ของทุกลิสต์นะ)
4. จงเขียน `def mergesort(list: List[Int]):List[Int] = {` ฟังก์ชันนี้ทำการ merge sort ของในลิสต์ รีเทิร์นลิสต์ที่เรียงจากน้อยไปมากออกมา
5. จงเขียน `def palindrome(list: List[Any]):Boolean = {` ฟังก์ชันนี้ทดสอบว่าลิสต์นั้นเป็นพาลินโดรมหรือไม่
6. จงเขียน `def myMap(f:Int => Int) (list:List[Int]) :List[Int] = {` ฟังก์ชันนี้รับพารามิเตอร์สองชุด ชุดแรกเป็นฟังก์ชันที่แมป จำนวนเต็มไปจำนวนเต็ม อีกชุดหนึ่งเป็นลิสต์ ฟังก์ชันนี้รีเทิร์น ลิสต์ที่เกิดจากการทำ ฟังก์ชัน f ที่สมาชิกทุกตัวของ list (ยกเว้น list เป็น ลิสต์ว่าง จะรีเทิร์นลิสต์ว่าง)

ตัวอย่าง `myMap(x => x*2) (List(1,2,3,4,5))` จะได้ `List(2,4,6,8,10)`

`myMap(square) (List(1,2,3,4,5))` จะได้ `List(1,4,9,16,25)` ถ้ามีการเขียนฟังก์ชัน square ไว้แล้ว

7. จงเขียน `def myFilter(f:Int => Boolean) (list:List[Int]) :List[Int] = {` ฟังก์ชันนี้รับพารามิเตอร์สองชุด ชุดแรกเป็นฟังก์ชันที่รับ Int แล้วรีเทิร์น Boolean ชุดที่สองเป็น list ของจำนวนเต็ม ฟังก์ชันนี้รีเทิร์นลิสต์ใหม่ ที่มีเฉพาะสมาชิกจาก list ที่ apply f แล้วเป็นจริงเท่านั้น ส่วนลิสต์ว่าง จะได้ลิสต์ว่างรีเทิร์นมา
ตัวอย่าง `myFilter(x => x%2==0) (List(1,2,3,4,5))` จะได้ `List(2,4)`
`myFilter(isLessThan3) (List(1,2,3,4,5))` จะได้ `List(1,2)` ถ้ามีการเขียน `def isLessThan3(x:Int) = x<3`

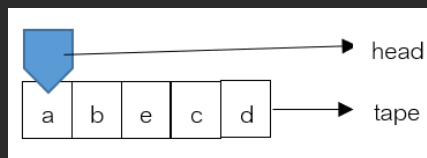
- * 8. จงเขียน `def sumAll(lists:List[List[Int]]) :List[Int] = {` ฟังก์ชันนี้รับ ลิสต์ของลิสต์ แล้วสร้างลิสต์ใหม่ ที่มีสมาชิกตัวที่ i เป็นผลรวมของสมาชิกตัวที่ i จากทุกลิสต์ ตัวอย่างการรันและ output เป็นดังนี้

```
println(sumAll(List()))
println(sumAll(List(List())))
println(sumAll(List(List(0,2,3,4), List(), List(0,5), List(0,2,0,4,5,6))))
println(sumAll(List(List(3,4), List(1,2,3,4,5,6), List(1,2,3,4))))
println(sumAll(List(List(1,2,3,4,5,6), List(1,2,3,4), List(1,2), List(0,0,0,0,0,0,0,9))))
```

ได้เข้าที่พูดดังนี้

```
List() } **
List() }
List(0, 9, 6, 8, 5, 6)
List(5, 8, 6, 8, 5, 6)
List(3, 6, 6, 8, 5, 6, 0, 0, 9)
```

9. ทัวริงแมชชีน มีรูปร่างดังรูป โดยมีเทปที่เก็บ character ไว้



ให้เขียนฟังก์ชัน

`def turingStep(f:Char => Char, tape:List[Char], n:Int): List[Char] = {`
 ฟังก์ชันนี้ ทำการ apply f กับตัว character n ตัวแรกใน tape (เป็นการจำลองการทำงานและเลื่อนหัว n ครั้ง
 ของหัวอ่าน)

ตัวอย่าง ถ้า f1 นิยามเป็น ฟังก์ชันที่เปลี่ยน character ให้เป็น lower case และ `tape = List('C', 'H', 'A', 'R')`

ผลการรันของ

`println(turingStep(f1, tape, 2))`
`println(turingStep(f1, tape, 3))`
`println(turingStep(f1, tape, 0))`
`println(turingStep(f1, tape, 5))` ตัวเลขเกินได้ แต่ไม่มีกรณีที่เป็นลบ

2 ตัวแรก
 3 ตัวแรก
 5 ตัวแรก

จะได้เป็น

`List(c, h, A, R)`
`List(c, h, a, R)`
`List(C, H, A, R)`
`List(c, h, a, r)`

10. จงเขียนฟังก์ชัน

`def alternate(f1: (Int,Int) => Int, f2: (Int,Int) => Int, list:List[Int]):Int`
`= {`

ฟังก์ชันนี้ รับ ฟังก์ชัน f1 กับ f2 และ ลิสต์ list โดย f1 กับ f2 นั้นเป็นฟังก์ชันอะไรก็ได้ ที่รับพารามิเตอร์สองตัว แล้วให้เลขจำนวนเต็มออกมา
 ซึ่งถ้า f1 เป็นฟังก์ชัน + และ f2 เป็นฟังก์ชัน ลบ - จะได้ภาพจำลองการรัน (ไม่ใช่โค้ดจริง แต่พยายามเขียนให้เห็นภาพ) ว่า

`alternate(+, -, [])` ได้ผลลัพธ์เป็น 0
`alternate(+, -, [55])` ได้ผลลัพธ์ 55
`alternate(+, -, [1, 2])` ได้ผลลัพธ์ = $1+2 = 3$
`alternate(+, -, [1, 2, 3])` ได้ผลลัพธ์ = $1+2-3 = 0$
`alternate(+, -, [1, 2, 3, 4])` ได้ผลลัพธ์ = $1+2-3+4 = 4$

+
 -
 +