

```
1 package replist
2
3 import scala.collection.immutable.Stream.Empty
4
5
6 def prepend[A](rl: RepList[A], value: A): RepList[A] =
7   Cons(value, rl)
8
9 def headCopied[A](rl: RepList[A], n: Long): RepList[A
10  ] = rl match {
11    case Empty =>
12      throw new NoSuchElementException("Empty list has no
13      head to copy")
14    case Cons(h, tail) => Repeat(n, Cons(h, tail))
15    case Repeat(k, Cons(h, tail)) => Repeat(n + k, Cons(h
16    , tail))
17  }
18
19 def head[A](rl: RepList[A]): A = rl match {
20   case Empty =>
21     throw new NoSuchElementException("Empty list does
22     not have head")
23   case Cons(h, _) =>
24     h
25   case Repeat(_, Cons(h, _)) =>
26     h
27 }
28
29 def tail[A](rl: RepList[A]): RepList[A] = rl match {
30   case Empty =>
31     throw new NoSuchElementException("Empty list does
32     not have tail")
33   case Cons(h, tail) =>
34     tail
35   case Repeat(n, Cons(h, tail)) =>
36     tail
37 }
```

```

34 @tailrec
35 def last[A](rl: RepList[A]): A = rl match {
36   case Empty =>
37     throw new NoSuchElementException("Empty list does
not have last element")
38   case Cons(h, Empty) =>
39     h // base case
40   case Cons(h, tail) =>
41     last(tail) // recursive case on non repeat list
42   case Repeat(n, Cons(h, Empty)) =>
43     h // base case on repeat list
44   case Repeat(n, Cons(h, tail)) =>
45     last(tail) // recursive case on repeat list
46 }
47
48 def length[A](rl: RepList[A]): Long = {
49
50 @tailrec
51 def loop(rl: RepList[A], sum: Long): Long = rl match {
52   case Empty => sum
53   case Cons(h, tail) => loop(tail, sum + 1)
54   case Repeat(n, Cons(h, tail)) => loop(tail, sum + n)
55 }
56
57 loop(rl, 0)
58 }
59
60 def take[A](rl: RepList[A], n: Long): RepList[A] = (n,
rl) match {
61   case (n, l) if n >= length(l) =>
62     rl //short cut if n > length(rl)
63   case (n, tail) if n <= 0 =>
64     Empty //base case, returns Empty to recursive case
once n == 0
65   case (_, Empty) =>
66     Empty //should be un-reachable but just in case
67   case (n, Cons(h, tail)) =>
68     Cons(h, take(tail, n - 1)) //recursive case for non

```

```

68 repeat list
69   case (n, Repeat(k, Cons(h, tail))) if n <= k =>
70     Repeat(n, Cons(h, Empty)) //recursive case for a
    repeat list when n < number of repeats
71   case (n, Repeat(k, Cons(h, tail))) =>
72     Repeat(k, Cons(h, take(tail, n - k))) //recursive
    case for a repeat list with less than n repeats
73 }
74
75 def drop[A](rl: RepList[A], n: Long): RepList[A] = (n
, rl) match {
76   case (n, l) if n >= length(l) =>
77     Empty //short cut if n > length(rl)
78   case (n, l) if n <= 0 =>
79     l //base case, returns remaining list to recursive
    case once n == 0
80   case (n, Empty) =>
81     Empty //should be un-reachable but just in case
82   case (n, Cons(h, tail)) =>
83     drop(tail, n - 1) //recursive case for non repeat
    list
84   case (n, Repeat(k, Cons(h, tail))) if k == 2 =>
85     drop(Cons(h, tail), n - 1) //recursive case for a
    repeat list that will become a cons
86   case (n, Repeat(k, Cons(h, tail))) =>
87     drop(Repeat(k - 1, (h, tail)), n - 1) //recursive
    case for a repeat list that will still be a repeat
    list
88 }
89
90 def isEmpty[A](rl: RepList[A]): Boolean = rl match {
91   case Empty => true
92   case _ => false
93 }
94
95 def nonEmpty[A](rl: RepList[A]): Boolean = {
96   !isEmpty(rl)
97 }

```

```

98
99 @tailrec
100 def contains[A](rl: RepList[A], target: A): Boolean =
    rl match {
101     case Empty => false
102     case Cons(h, _) if h == target => true
103     case Cons(h, l) if h != target => contains(l, target
    )
104     case Repeat(_, Cons(h, l)) => contains(Cons(h, l),
        target)
105 }
106
107 def count[A](rl: RepList[A], target: A): Long = {
108     @tailrec
109     def loop(rl: RepList[A], target: A, count: Long):
        Long = rl match {
110         case Empty => count
111         case Cons(h, tail) if h == target => loop(tail,
            target, count + 1)
112         case Cons(h, tail) if h != target => loop(tail,
            target, count)
113         case Repeat(n, Cons(h, tail)) if h == target =>
            loop(tail, target, count + n)
114         case Repeat(n, Cons(h, tail)) if h != target =>
            loop(tail, target, count)
115     }
116
117     loop(rl, target, 0)
118 }
119 def removeAll[A](rl: RepList[A], target: A): RepList[A
    ] = rl match {
120     case Empty => Empty
121     case (Cons(h, tail)) if h == target => removeAll(
        drop(rl, 1), target)
122     case (Cons(h, tail)) if h != target => Cons(h,
        removeAll(tail, target))
123     case Repeat(n, Cons(h, tail)) if h == target =>
        removeAll(drop(rl, n), target)

```

```

124   case Repeat(n, Cons(h, tail)) if h != target =>
      Repeat(n, (h, removeAll(tail, target)))
125 }
126
127 @tailrec
128 def getAt[A](rl: RepList[A], i: Long): A = rl match {
129   case Empty =>
130     throw new NoSuchElementException("Empty list has
      no entry at index $i")
131   case Cons(h, tail) if i == 0 => h
132   case Cons(h, tail) if i > 0 => getAt(tail, i - 1)
133   case Repeat(n, Cons(h, tail)) if i < n => h
134   case Repeat(n, Cons(h, tail)) if i >= n => getAt(
      tail, i - n)
135 }
136
137 def toList[A](rl: RepList[A]): List[A] = rl match {
138   case Empty => Nil
139   case Cons(h, tail) => h :: toList(tail)
140   case Repeat(n, Cons(h, tail)) => h :: toList(Repeat(
      n - 1, Cons(h, tail)))
141 }
142
143 def toSet[A](rl: RepList[A]): Set[A] = rl match {
144   case Empty => Set.empty[A]
145   case Cons(h, tail) => toSet(tail) + h
146   case Repeat(n, Cons(h, tail)) => toSet(tail) + h
147 }
148
149 def concat[A](rl1: RepList[A], rl2: RepList[A]):
      RepList[A] = rl1 match {
150   case Empty => rl2
151   case Cons(h, tail) => Cons(h, concat(tail, rl2))
152   case Repeat(n, Cons(h, tail)) => Repeat(n, Cons(h,
      concat(tail, rl2)))
153 }
154
155 @tailrec

```

```
156 def reverse[A](rl: RepList[A]): RepList[A] = {
157   def loop(rl: RepList[A], revrs: RepList[A]): RepList
    [A] = rl match {
158     case Empty => revrs
159     case Cons(h, tail) => loop(tail, prepend(revrs, h
    ))
160     case Repeat(n, Cons(h, tail)) => loop(tail, Repeat
    (n, Cons(h, revrs)))
161   }
162
163   loop(rl, Empty)
164 }
165
166 def sort[A: Ordering](rl: RepList[A]): RepList[A
    ] = ???
167
```