TÖL212M Rökstudd Forritun - Einstaklingsverkefni 9

Andri Fannar Kristjánsson

17. mars 2025

Einstaklingsverkefni 9

1

Sækið skrána `E9-skeleton.java` í Canvas og vistið hana sem `E9.java`. Klárið að forrita klasann í skránni. Munið að allar lykkjur þurfa skýra og fullnægjandi fastayrðingu. Vandið ykkur vel í að skrifa fastayrðingarnar.

1.1 Svar:

Hér fyrir neðan má sjá kóðann þar sem föllin í klasanum hafa verið forrituð. Einnig er hægt að sjá skrána hér: https://tinyurl.com/3yes9269.

```
// Author: Snorri Agnarsson, snorri@hi.is

// Author of Solution: Andri Fannar Kristjánsson, afk6@hi.is

// Java lists without side effects.

public class E9 {
    // Instances of Link are immutable links with a
    // head that is an integer and a tail that is a
    // finite chain of links. Note that there is no
    // possibility to change the tail and therefore
    // al chains are finite. An empty chain is denoted
    // by null.
    public static class Link {
        private final int head;
        private final Link tail;

        // Usage: E9.Link x = new E9.Link(head, tail);
        // Pre: head is an int, tail is an E9.Link (may be null).
        // Post: x refers to a new E9.Link with the given head and
        // tail.
        public Link(int head, Link tail) {
            this.head = head;
            this.tail = tail;
        }

        // Usage: int h = link.head();
        // Pre: link refers to an E9.Link object.
        // Post: h contains the head of link.
        public int head() {
            return head;
        }

        // Usage: E9.Link t = link.tail();
        // Pre: link refers to an E9.Link object.
        // Post: t contains the tail of link.
        public Link tail() {
            return tail;
        }
    }

    // Usage: E9.Link x = E9.cons(head,tail);
    // Pre: head is an int, tail is an E9.Link (may be null).
    // Post: x refers to a new E9.Link with the given head and
    // tail.
    public static Link cons(int h, Link t) {
        return new Link(h, t);
    }

    // Usage: int h = E9.head(x);
    // Pre: link refers to an E9.Link object.
    // Post: h contains the head of x.
    public static int head(Link x) {
        return x.head();
    }

    // Usage: E9.Link t = tail(x);
    // Pre: x refers to an E9.Link object.
    // Post: t contains the tail of x.
    public static Link tail(Link x) {
        return x.tail();
    }

    // Usage: int n = E9.length(x);
    // Pre: x is an E9.Link, may be null.
    // Eftir: n is the number of links in the chain x.
    public static int length(E9.Link x) {
        // ... use a loop to implement this body
        if (x == null) {
            return 0;
        }
        E9.Link current = x;
        int count = 0;
        while (current.tail() != null)
        // Loop Invariant:
        // 0 <= count <= |x|, where |x| is the total number of
        // links in the chain x.
        // current is the count-th link in the chain x.
        // The total length of x is count plus the number of links
        // in the chain starting at current
        {
            count++;
            current = current.tail();
        }
        return count + 1;
    }

    // Usage: int i = E9.nth(x,n);
    // Pre: n>=0, x is a chain with at least n+1 links.
    // Post: i is the head of the n-th link in the chain
    // where the 0-th link is the first link.
    public static int nth(E9.Link x, int n) {
        // ... use a loop to implement this body
        E9.Link current = x;
        for (int i = 0; i < n; i++)
        // Loop Invariant:
        // 0 <= i <= n, where n is the index of the link in the chain x.
        // current is the i-th link in the chain x, current == x[i].
        {
            current = current.tail();
        }
        return current.head();
    }

    // Usage: E9.Link x = makeChain(a);
    // Pre: a refers to an int[]. Must not be null,
    // but may be empty.
    // Post: x is a chain that contains the values in a
    // such that for i=0,...,a.length-1 we have
    // E9.nth(x,i) == a[i].
    public static Link makeChain(int[] a) {
        // ... use a loop to implement this body
        E9.Link x = null;
        for (int i = a.length - 1; i >= 0; i--)
        // Loop Invariant:
        // 0 <= i <= a.length, where a.length is the total number
        // of elements in the array a.
        // x is the chain corresponding to the subarray
        // a[i+1 ... a.length-1].
        {
            x = cons(a[i], x);
        }
        return x;
    }

    // Usage: int i = E9.last(x);
    // Pre: x refers to a E9.Link, must not be null.
    // Post: i is the value in (the head of) the last
    // link in x.
    public static int last(Link x) {
        // ... use a loop to implement this body
        E9.Link current = x;
        while (current.tail() != null)
        // Loop Invariant:
        // current points to the i-th link in the chain x and 0 <= i < |x|.
        // 0 <= i <= |x|, where |x| is the total number of links in the chain x.
        {
            current = current.tail();
        }
        return current.head();
    }

    // Usage: E9.Link z = E9.removeLast(x);
    // Pre: x refers to a E9.Link, must not be null.
    // Post: z is a chain of new links such that
    // E9.length(z) == E9.length(x)-1
    // and for i=0,...,E9.length(z)-1 we have
    // E9.nth(z,i) == E9.nth(x,i).
    public static Link removeLast(Link x) {
        // ... use a loop to implement this body
        E9.Link current = x;
        E9.Link newLink = null;
        while (current.tail() != null)
        // Loop Invariant:
        // Let i be the number of elements processed so far, 0 <= i < |x|.
        // newLink is the reverse of the chain x[0 ... i-1],
        // newLink = reverse(x[0 ... i-1]).
        // current points to the i-th link in the chain x.
        // Concatinating reverse(newLink) with the chain starting
        // at current will result in the original chain x.
        {
            newLink = cons(current.head(), newLink);
            current = current.tail();
        }

        return reverse(newLink);
    }

    // Usage: E9.Link r = E9.reverse(x);
    // Pre: x is a chain, may be empty.
    // Post: z is a new chain of equal length to x, such
    // that for i=0,...,E9.length(x)-1 we have
    // E9.nth(x,i) == E9.nth(r,E9.length(x)-i-1).
    public static Link reverse(Link x) {
        // ... use a loop to implement this body
        E9.Link current = x;
        E9.Link newLink = null;
        while (current != null)
        // Loop Invariant:
        // Let i be the number of elements processed, 0 <= i <= |x|.
        // newLink is the reverse of the chain x[0 ... i-1],
        // newLink = reverse(x[0 ... i-1]).
        // current points to the i-th link in the chain x.
        // Concatinating reverse(newLink) with the chain starting
        // at current will in the original chain x.
        {
            newLink = cons(current.head(), newLink);
            current = current.tail();
        }
        return newLink;
    }

    // Run the command
    // java E9 1 2 3 4
    // and show what the program writes
    public static void main(String[] args) {
        E9.Link x = null;
        for (int i = 0; i != args.length; i++)
            x = E9.cons(Integer.parseInt(args[i]), x);
        while (x != null) {
            E9.Link z = reverse(x);
            x = z;
            while (z != null) {
                System.out.print(z.head);
                System.out.print(" ");
                z = z.tail;
            }
            x = removeLast(x);
            System.out.println();
        }
    }
}
```