TÖL212M Rökstudd Forritun - Hópverkefni 9

Andri Fannar Kristjánsson

18. mars 2025

Hópverkefni 9

1

Sækið skrána `H9-skeleton.java` í Canvas og vistið hana sem `H9.java`. Klárið að útfæra 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/4475ehdy.

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

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

// Mutable lists in Java.

public class H9 {
    // Instances of Link are mutable links with a
    // head that is an int and a tail that is a
    // finite chain of links. An empty chain is
    // denoted by null. It is possible to create
    // circular chains and it is possible to change
    // both the head and the tail.
    public static class Link {
        public int head;
        public Link tail;
    }

    // Usage: H9.Link x = H9.cons(head,tail);
    // Pre: head is an int, tail is an E9.Link (may be null).
    // Post: x refers to a new H9.Link with the given head and
    // tail.
    public static Link cons(int h, Link t) {
        Link newLink = new Link();
        newLink.head = h;
        newLink.tail = t;
        return newLink;
    }

    // Usage: int n = H9.length(x);
    // Pre: x is an H9.Link, may be null,
    // and must not refer to a circular chain.
    // Eftir: n is the number of links in the chain x.
    public static int length(H9.Link x) {
        // ... use a loop to implement this body
```

```
        if (x == null) {
            return 0;
        }
        H9.Link current = x;
        int count = 0;
        while (current.tail != null)
        // Loop Invariant:
        //   0 <= count <= |x|, where |x| is the total number of links in x.
        //   current is the count-th link in x.
        //   The total length of x is count plus the length of
        //   the chain starting at current.
        {
            count++;
            current = current.tail;
        }
        return count + 1;
    }

    // Usage: int i = H9.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(H9.Link x, int n) {
        // ... use a loop to implement this body
        H9.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: H9.Link x = H9.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
    // H9.nth(x,i) == a[i].
    public static Link makeChain(int[] a) {
        // ... use a loop to implement this body
        H9.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 = H9.last(x);
    // Pre: x refers to a H9.Link, must not be null,
    // and must not refer to a circular chain.
```

```
// 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
    H9.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: H9.Link z = H9.destructiveRemoveLast(x);
// Pre: x refers to an H9.Link, must not be null
// and must not be circular.
// Post: z is a chain that contains the same links
// in the same order as x, except that the
// link that was last in x has been removed.
// The link in x that was in front of that
// last link (if any) now has a tail that is
// null.
public static Link destructiveRemoveLast(Link x) {
    if (x == null || x.tail == null) {
        return null;
    }

    // ... use a loop to implement this body
    H9.Link current = x;
    while (current.tail.tail != null)
    // Loop Invariant:
    // Let i be the number of elements processed so far,
    // 0 <= i < |x| - 1.
    // current points to the i-th link in the chain x,
    // current.tail is not null.
    // The links from the head of x up to current are unchanged.
    // When current.tail.tail is null, current is the penultimate
    // link in the chain x.
    {
        current = current.tail;
    }

    current.tail = null;

    return x;
}

// Usage: H9.Link r = H9.destructiveReverse(x);
// Pre: x is a chain, may be empty (i.e. null).
// Post: z is a chain containing the same links as
// x, but the order of the links has been
// reversed. The int values in the links are
// unchanged.
public static Link destructiveReverse(Link x) {
```

```
        Link y = null;
        while (x != null)
        // y contains a chain of zero or more links that
        // have been removed from the front of x.
        // The order of the links in y are the reverse of
        // their order in x. The values in the links are
        // unchanged except for the tails.
        // ... Program the body of this loop
        {
            Link z = x.tail;
            x.tail = y;
            y = x;
            x = z;
        }
        return y;
    }

    // Run the command
    // java H9 1 2 3 4
    // and show what the program writes
    public static void main(String[] args) {
        H9.Link x = null;
        for (int i = 0; i != args.length; i++)
            x = H9.cons(Integer.parseInt(args[i]), x);
        while (x != null) {
            H9.Link z = H9.destructiveReverse(x);
            x = z;
            while (z != null) {
                System.out.print(z.head);
                System.out.print(" ");
                z = z.tail;
            }
            x = H9.destructiveRemoveLast(x);
            System.out.println();
        }
    }
}
```