# Opgaveløsninger (sæt 2)

## Opgave 1a (3.2)

```
void movenexttofront(Node t) {
    Node u = t.next;
    t.next = u.next;
    u.next = head.next;
    head = u;
}
```

## Opgave 1b (3.3)

```
void exchange(Node t, Node u) {
    Node v = u.next;
    u.next = t.next;
    t.next = v;
    t = v.next;
    v.next = u.next.next;
    u.next.next = t;
}
```

## Opgave 2 (3.5)

Opgaven er løst ved anvendelse af en klasse, Link, der repræsenterer listens elementer. Felterne `PRED` og `SUC` refererer til henholdsvis forgængeren og efterfølgeren for et listeelement. En tom liste indeholder kun et listehoved, `head`, hvor `head.PRED == head.SUC == head`.

Metoden `follow` kan benyttes til indsættelse af et element. Elementet indsættes efter elementet refereret ved pegeren `ptr`. Metoden `out` kan benyttes til at fjerne et element fra listen. NB. Metoderne tager ikke højde for misbrug!

```
class Link {
    Link PRED, SUC;

    void follow(Link ptr) {
        PRED = ptr;
        SUC = ptr.SUC;
        SUC.PRED = ptr.SUC = this;
    }

    void out() {
        SUC.PRED = PRED;
        PRED.SUC = SUC;
        SUC = PRED = null;
    }
}
```

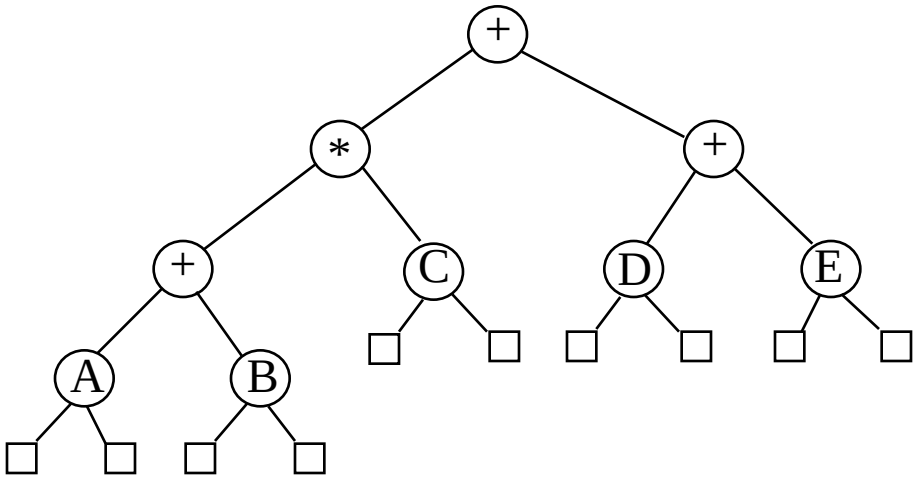
Nedenfor ses en generel pakke til håndtering af dobbelthægtede lister. Pakken er inspireret af SIMULA's indbyggede klasse SIMSET.

```
package simset;

class Linkage {
    public Link pred()
        { return PRED instanceof Link ? (Link) PRED : null; }
    public Link suc()
        { return SUC instanceof Link ? (Link) SUC : null; }
    public Linkage prev() { return PRED; }
    protected Linkage PRED, SUC;
}

public class Link extends Linkage {
    public void out() {
        if (SUC != null) {
            SUC.PRED = PRED;
            PRED.SUC = SUC;
            SUC = PRED = null;
        }
    }
    public void follow(Linkage ptr) {
        out();
        if (ptr != null && ptr.SUC != null) {
            PRED = ptr;
            SUC = ptr.SUC;
            SUC.PRED = ptr.SUC = this;
        }
    }
    public void precede(Linkage ptr) {
        out();
        if (ptr != null && ptr.SUC != null) {
            SUC = ptr;
            PRED = ptr.PRED;
            PRED.SUC = ptr.PRED = this;
        }
    }
    public void into(Head s) { precede(s); }
}

public class Head extends Linkage {
    public Head() { PRED = SUC = this; }
    public Link first() { return suc(); }
    public Link last() { return pred(); }
    public boolean empty() { return SUC == this; }
    public int cardinal() {
        int i = 0;
        for (Link ptr = first(); ptr != null; ptr = ptr.suc())
            i++;
        return i;
    }
    public void clear()
        { while (first() != null) first().out(); }
}
```

**Opgave 3a (4.1)**

| | |
|---|---|
| Preorder: | EO ACPMLTET ERE |
| Inorder: | A COMPLETE TREE |
| Postorder: | AC MLPOT EREETE |
| Level order: | EOT PEEACMLT RE |

**Opgave 3b (4.3)**



## Opgave 4

(a) En mulig implementering ses nedenfor:

```
import java.util.Vector;
import java.util.EmptyStackException;

public class Queue extends Vector {
    public void put(Object obj) { addElement(obj); }

    public Object get() {
        Object obj  = peek();
        removeElementAt(0);
        return obj;
    }

    public Object peek() {
        if (size() == 0)
            throw new EmptyStackException();
        return elementAt(0);
    }

    public boolean empty() { return size() == 0; }
}
```

(b) Metoden `put` tager konstant tid (hvis der ses bort fra ekspandering). Derimod er tidsforbruget af `get` proportional med køens længde. Metoden `removeElement` parallelforskyder nemlig de resterende elementer i vektoren.

Hvis metoderne derimod implementeres som vist på side 31 i lærebogen, tager de begge konstant tid.

Nedenfor ses et eksempelprogram, der benytter klassen `Queue`. Programmet indsætter heltallene fra 1 til 10 i en kø, hvorefter tallene udtages fra køen og udskrives i denne rækkefølge.

```
import IO.*;

public class Program {
    public static void main(String args[]) {
        Queue q = new Queue();
        for (int i = 1; i <= 10; i++)
            q.put(new Integer(i));
        while (!q.isEmpty())
            IO.println((Integer) (q.get()));

    }
}
```