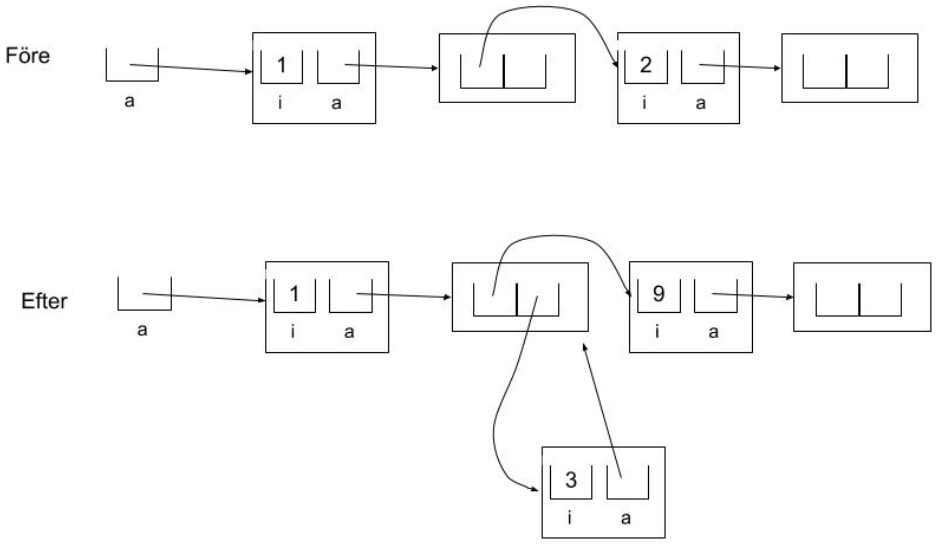
```
/*
 *   Suggested solutions exam DAT043 2020-03-14
 *
 */

   // ------------------------- 1 -------------------- (4p)

   void pyramid() {
      Scanner sc = new Scanner(in);
      out.print("Number of blocks: ");
      int totalBlocks = sc.nextInt();
      int nblocks = 1;
      int maxHeight = 0;
      int i = 3;
      while (totalBlocks >= nblocks) {
         totalBlocks -= nblocks;
         nblocks = i;
         nblocks *= nblocks;
         i += 2;
         maxHeight++;
      }
      out.print("Max height: " + maxHeight);
   }

   // ------------------------- 2 -------------------- (2p)

   // No such question

   // ---------------- 3 ---------------------- (7p)

   int[] toArray(int n) {
      int[] ds = new int[nDigits(n)];
      for (int i = ds.length - 1; i >= 0; i--) {
         ds[i] = n % 10;
         n = n / 10;
      }
      return ds;
   }

   int nDigits(int n) {   // Helper
      int count = 0;
      while (n > 0) {
         n = n / 10;
         count++;
      }
      return count;
   }
```

```
// ---------------- 4 ---------------------- (13p)

 // Assume square matrix
boolean contains(int[][] m, int[] a) {
    for (int row = 0; row < m.length; row++) {
        if (isSubArray(m[row], a)) {
            return true;
        }
        if (isSubArray(getCol(m, row), a)) {
            return true;
        }
    }
    return false;
}

int[] getCol(int[][] m, int colIndex) {
    int[] col = new int[m.length];
    for (int r = 0; r < col.length; r++) {
        col[r] = m[r][colIndex];
    }
    return col;
}

boolean isSubArray(int[] arr, int[] sub) {
    // Two index to traverse the arrays in parallel
    int i = 0, j = 0;
    while (i < arr.length && j < sub.length) {
        if (arr[i] == sub[j]) {
            i++;
            j++;
            if (j == sub.length) {
                return true;
            }
        } else {
            i = i - j + 1;  // Restart
            j = 0;
        }
    }
    return false;
}


// ------------------------ 5 ------------------ (8p)

 String expandString(String str) {
    StringBuilder sb = new StringBuilder();
    while (str.length() > 0) {
        int num = Character.getNumericValue(str.charAt(0));
```

```
        int open = str.indexOf("(");
        int close = str.indexOf(")");
        String group = str.substring(open + 1, close);
        for (int j = 0; j < num; j++) {
            sb.append(group);
        }
        str = str.substring(close + 1);
    }
    return sb.toString();
}

// ---------- 6 ------------------------------ (8p)
```



```
// -------------------- 7 ------------------------- (10p)

  public class Mail {                          // 2p
     private final String receiverAddress;
     private final  String senderAddress;
     private final String text;

     public Mail(String receiverAddress, String senderAddress, String text) {
        this.receiverAddress = receiverAddress;
        this.senderAddress = senderAddress;
        this.text = text;
     }
      // Setter/getter, equals, hashCode omitted but present
  }

  public class Account {                       // 3p
     private final Person owner;
     private final String address;
     private final List<Mail> inbox = new ArrayList();
     private final List<Mail> outbox = new ArrayList();

     public Account(Person owner, String address) {
        this.owner = owner;
        this.address = address;
     }
     // Getter/setter etc.
  }

  public class MailServer {                    // 5p
     private final List<Account> accounts = new ArrayList<>();

     public boolean forward(Account sender, Account receiver, Mail mail) {
        if ( !accounts.contains(sender) || !accounts.contains(receiver)){
           return false;
        }
        if( !sender.getInbox().contains(mail)){
           return false;
        }
        int i = sender.getInbox().indexOf(mail);
        Mail m = sender.getInbox().remove(i);
        sender.getOutbox().add(m);
        receiver.getInbox().add(m);
        return true;
     }
  }
```

```
// --------------------- 8 -------------------------- (8p)

    void countDigitalMultiplicativeRoots() {
        Scanner sc = new Scanner(in);
        out.print("From ");
        int from = sc.nextInt();
        out.print("To ");
        int to = sc.nextInt();

        int n = 0;
        for (int at = from; at <= to; at++) {
            if (digital_root(at) == multiplicative_root(at)) {
                n++;
            }

        }
        out.print("Number of equals " + n);
    }


    int digital_root(int tal) {
        int sum = 0;
        while (tal > 0) {
            sum += tal % 10;
            tal /= 10;
        }
        if (sum >= 10)
            return digital_root(sum);
        else
            return sum;
    }

    int multiplicative_root(int tal) {
        int prod = 1;
        while (tal > 0) {
            prod *= tal % 10;
            tal /= 10;
        }
        if (prod >= 10)
            return multiplicative_root(prod);
        else
            return prod;
    }
```