7.2: pattern matching

Give a pattern-matching definition of a function which:

1. adds together the first two integers in a list, if a list contains at least two elements;
2. returns the head element if the list contains one
3. returns zero otherwise

7.6: and/or function

Define the functions `and2` and `or2` on a list of `Bool`s. These should return the conjuction or disjunction of a list of booleans. Note that these are already defined in the prelude, so we call them `and2` and `or2`.

Examples:
`and2 [True, False] = False`
`or2 [False, True] = TRUE`

7.8: elemNum

Using primitive recursion over lists, define a function
`elemNum :: Integer -> [Integer] -> Integer`
so that `elemNum x xs` returns the number of times that `x` occurs in the list `xs`.
Next, define `elemNum` without primitive recursion, using list comprehensions and built-in functions instead.

7.9: 7.9: unique

Define a function `unique :: [Integer] -> [Integer]` so that `unique xs` returns the list of elements of `xs` which occur exactly once.

Example: `unique [4, 2, 1, 3, 2, 3] = [4, 1]`

You might like to think of two solutions to this problem: one using list comprehensions and the other not.

7.16: modifying insertion sort

Given is the following implementation of insertion sort.

```
iSort :: [Integer] -> [Integer]
iSort [] = []
iSort (x:xs) = ins x (iSort xs)

ins :: Integer -> [Integer] -> [Integer]
ins x [] = [x]
ins x (y:ys)
  | x <= y    = x:(y:ys)
  | otherwise = y:(ins x ys)
```

By modifying the `ins` function we can change the behaviour of `iSort`.
Redefine `ins` in two different ways so that
1 : the list is sorted in descending order;
2: duplicates are removed from the list.

7.25: sublist/sequence

One list is a *sublist* of another if all the elements of the first list occur in the second, in the same order.
For example, `"ship"` is a sublist of `"Fish & Chips"`, but not of `"hippies"`.

A list is a *subsequence* of another if it occurs as a sequence of elements next to each other.
For example, `"Chip"` is a subsequence of `"Fish & Chips"`, but not of `"Chin up"`.

Define functions that decide whether a list is a sublist/sequence of another list.

7.33: palindrome

Define a function `isPalin` which tests whether a string is a palindrome.
Example of a palindrome: `"Madam I'm Adam"`

Note that punctuation and white space are ignored, and that there is no distinction between capital and small letters.

7.34: subst

Design a function `subst :: String -> String -> String -> String`
so that `subst oldSub newSub st` is the result of replacing the first occurence in `st` of the substring `oldSub` by the substring `newSub`.
For example: `subst "much " "tall " "How much is that?" = "How tall is that?"`