Giving Change

Problem: given a set of coins (infinite supply of each denomination), produce

- exact change for a given amount
- involving a low (but not necessarily minimal) number of coins.

This may not always be possible

return 7c using 5c coins and 3c coins

(though it is always possible if we have 1c coins.)

- We would like not to test all combinations

Greedy Strategy: return as many as possible from highest denomination, then as many as possible from second-highest denomination, etc.

- this is not always optimal:

  return 8c using 5c,4c,1c

- but for US coin set {25,10,5,1} it is optimal (though not trivial to prove)

Lab #2

- Find a near-minimal set of coins that equal the given amount.
- Return the result as an option type
  - If change is possible: `SOME [...]`
  - If change isn't possible: `NONE`
- Note that you will need to backtrack in some cases
  - Your exception handler should be in the same function as your recursive algorithm
  - The handler will have access to the "old" state

Lab #2

The function takes a list of coin amounts, and the amount of change required:

```
val Lab2 = fn : int list * int ->
  int list option
```

Include the following test cases:

```
Lab2([25,10,5,1],48); (* Normal *)
Lab2([5,2],16); (* Backtracking *)
Lab2([4],7); (* Impossible *)
```

Which should have the following outputs:

```
val it = SOME [25,10,10,1,1,1] :
  int list option
val it = SOME [5,5,2,2,2] :
  int list option
val it = NONE : int list option
```