Problem 1: Book Exercises

- Exercise 8.1
- Exercise 8.2
- Exercise 8.4
- Exercise 8.5

Problem 2: Continuation Passing Style

1. Here is a simple recursive definition of a power function

```
fun pow n 0 = 1
  | pow n m = n * (pow n (m-1))
```

   Turn this definition into one that uses continuation passing style with the type:

```
powk : int -> int -> (int -> a) -> a
```

2. Here is a simple recursive definition that computes the product of a list

```
fun prod [] = 1
  | prod (x::xs) = x * (product xs)
```

   Turn this definition into one that uses continuation passing style with the type:

```
prodk : int list -> (int -> a) -> a
```

3. You are given the CPS version of map whose type and definition are provided as follows:

```
mapk : (a -> (b -> r) -> r) -> a list -> (b list -> r) -> r
mapk f [] k = k []
mapk f (x :: xs) k = mapk f xs (fn vs => f x (fn v => k (v :: vs)))
```

   Define a function add_{1k}, with the appropriate type, such that the invokation

```
mapk add_1k elemList id
```

   adds one to every element of the list *elemList* (you can assume $elemList : int\ list$)