Assignment 8 (Exceptions + CPS)

Problem 1: Book Exercises

- Exercise 8.1

  1. *Solution:* The answer is 0. **pred** raises an exception the second time it is called and the exception is handled by **twice.** □
  2. *Solution:* The answer is 1. The first call to **dumb** is handled by **twice**. □
  3. *Solution:* The answer is 1. The first call to **smart** handles the exception raised by its call to **pred** □

- Exercise 8.2

  Solution: ML evaluates from left to right. So in the case of **g(nil)**, **Hd(l)** is thrown first and handled. □

- Exercise 8.4

  Solution:

```
call f(7)
call f(3)
call f(1)
raise Odd
pop f(1)
pop f(3)
handle Odd in f(5)
return -5
```

□

- Exercise 8.5

  Solution: We cannot do tail call optimization because the recursive call is wrapped in an exception handler. However, since the handler doesn't actually bind any variable, we could rework the function to get it to tail call. □

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
```

Solution:

```
fun powk n 0 k = k 1
 |  powk n m k = powk n (m - 1) (fn v => k (n * v))
```

□

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
```

Solution:

```
fun prodk [] k       = k 1
  | prodk (x::xs) k  = prodk xs (fn v => k (x * v))
```

□

3. *Solution:*

```
add_1k : int -> (int -> r) -> r
add_1k x k = k (x + 1)
```

□