**Introduction to Functional Programming**
Lent 2005
Suggested Exercises 3

1. **Sequences**
Define a function that takes a sequence (a lazy list) of integers, and produces a sequence containing the sums of adjacent pairs. So, the list $[x_1, x_2, x_3, x_4, \ldots]$ would be mapped to $[x_1 + x_2, x_3 + x_4, \ldots]$.

2. **Lazier Lists**
The following datatype provides an alternative implementation of lazy lists to the one given in the lectures. It avoids the unnecessary computation of the first element, when it is not required.

```
datatype 'a seq = Nil
                | Cons of unit -> 'a * 'a seq;
```

Modify the functions `head`, `tail`, `from`, and `get` to work with this datatype.

Repeat Question 1 for this datatype.

3. **Searching**
Suppose there are three missionaries and three cannibals who need to cross to the far side of a river using a single boat that can carry one or two people at a time. Both groups will cooperate and can paddle back and forth freely, but old habits will lead the cannibals to eat the missionaries if the missionaries are ever outnumbered on either side of the river.

```
datatype side = Near | Far
type state = side * int * int
```

An instance of `state`, e.g., (*side*, *missionaries*, *cannibals*), tells where the boat is and how many people from each group remain on the near side at the moment represented by that state. The number from each group already on the far shore is implied. Each time the state changes, `side` should alternate and one or two people should move with the boat to the opposite shore.

Write a function that takes an initial state such as `(Near,3,3)` as its argument and returns the shortest complete list of states to safely move everyone to the far side, i.e., to the state `(Far,0,0)`.