The far-off country of Micomicona has a calendar with no months or numbered dates. Instead, they have one holiday at the end of the year (called Holi Day), and every other day of the year is an eve of some other day of the year. We can model this using the following ML datatype:

```
datatype day = HoliDay | Eve of day;
```

a. Write a function `comesBefore` that takes two days and determines whether the first parameter comes before the second in the course of the year. For example, `comesBefore(Eve(Eve(Eve(Eve(HoliDay)))), Eve(Eve(HoliDay)))` would return `true`, and `comesBefore(Eve(HoliDay), Eve(HoliDay))` would return `false`.

b. The Micomicona National Committee on Calendar Revision meets for lunch every three days, ending with two days before Holi Day. That is, they meet two days before Holi Day, five days before, eight days before, eleven days before, and on any day for which the number of days before Holi Day mod 3 is 2. Write a function `isLunchDay` that takes a day and determines whether the commitee meets for lunch that day. For example, `isLunchDay(Eve(Eve(Eve(Eve(Eve(HoliDay))))))` would return `true`, and `isLunchDay(Eve(Eve(Eve(HoliDay))))` would return `false`.

c. By law all rental agreements terminate on Holi Day and include a one-time charge and a daily rate. Write a function `rentalCost` that takes a day when the agreement starts and int amounts for the one-time charge and daily rate (in Micimiconan Ducats) and calculates the total cost. For example, `rentalCost(Eve(Eve(HoliDay)), 5, 3)` would return 11 (that is, $5 + 2 \cdot 3$).

d. All term papers at Micomicona University are due on Holi Day. To get an extension, students must request that the paper be *assigned earlier* rather than due later. Write a function `doubleDate` which takes a day and returns another day that is twice as many days before Holi Day. For example, `doubleDate(Eve(Eve(HoliDay)))` would return `Eve(Eve(Eve(Eve(HoliDay))))`. (Note that no extension is possible for papers assigned on Holi Day itself.)

Prove that $I(n)$ is a loop invariant for `bbb`. (14 points.)

$$I(n) = \quad \text{after } n \text{ iterations, } x = 50 + i$$

```
fun bbb(m) =
  let
    val x = ref 50;
    val y = ref 50;
    val i = ref 0;
  in
   (while !i < m do
     (x := !x + 1;
      y := !y - 1;
      i := !i + 1);
    !x + !y)
  end;
```

Write a function `findExtreme` that takes a function (with type int $\times$ int $\rightarrow$ bool) and a list of integers and uses the function to select the extreme element (least, greatest, etc) of the list. Specifically, the function that `findExtreme` takes as a parameter defines a way to order int, that is, it compares two ints (say *a* and *b*) and returns `true` if *a* comes before *b* and `false` otherwise (mathematically, this function is a *total order*). Thus `findExtreme` is a generalization of `findGreatest`. For example, `findExtreme(fn (a, b) => a > b, [6, 4, 18, 9, 2])` would return 18. (This problem is *not* naturally solved using `map` or `filter`.)