Question 1: List Filtering Function

Write a function called `filterEven` with the following type spec:

```
fun filterEven: int list -> int list
```

The function should take a list of integers and return a new list containing only the even numbers.

Example Runs:

```
filterEven [1, 2, 3, 4, 5];  (* Expected output: [2,4] *)
filterEven [7, 9, 11];       (* Expected output: [] *)
```

Question 2: List Transformation with Mapping

Write a function called `tripleAndSubtract` with the following type spec:

```
fun tripleAndSubtract: int list -> int list
```

This function should triple each element of the list and then subtract 1 from the result.

Example Runs:

```
tripleAndSubtract [1, 2, 3];   (* Expected output: [2,5,8] *)
tripleAndSubtract [0, -2, 4];  (* Expected output: [-1,-7,11] *)
```

Question 3: Tree Maximum Value

Given the following data type for a binary tree:

```
datatype result = NONE | SOME of int;

datatype iTree = EMPTY | NODE of int * iTree * iTree;
```

Write a function called `maxValue` with the type specification:

```
fun maxValue: iTree -> result
```

This function should return the maximum integer value in the tree. If the tree is empty, it should return NONE.

Example Runs:

```
maxValue (NODE(10, NODE(5, EMPTY, EMPTY), NODE(20, EMPTY, EMPTY)));
(* Expected output: SOME 20 *)
maxValue EMPTY;  (* Expected output: NONE *)
```

Question 4: Frequency Counting in a Custom Tree

Consider a datatype representing a tree that holds string values and a frequency count:

```
datatype cTree = CEMPTY | CNODE of string * int * cTree * cTree;
```

Write a function updateCount with the following type spec:
`fun updateCount: string * cTree -> cTree`

This function should insert a string into the tree in order. If the string is already present, increment its frequency count by 1. If not, insert it in the correct in-order position.

Example Runs:

```
(* Given an initial tree: *)
val tree = CNODE("mango", 1, CNODE("apple", 1, CEMPTY, CEMPTY),
CNODE("peach", 1, CEMPTY, CEMPTY));
updateCount ("apple", tree);   (* Expected result: "apple" count
becomes 2 *)
updateCount ("banana", tree);  (* Expected result: "banana" inserted
between "apple" and "mango" *)
```