Question 1: Determine the types of the following functions

Determine the types of the following functions:

a) `fun triple x = 3 * x;`
b) `fun compose f g x = f (g x);`
c) `fun filterEven [] = [] | filterEven (x::xs) = if x mod 2 = 0 then x::(filterEven xs) else filterEven xs;`
d) `fun pairwise f (a, b) = (f a, f b);`
e) `fun map2 f [] [] = [] | map2 f (x::xs) (y::ys) = (f x y)::(map2 f xs ys);`

Question 2: Implement a Stack Signature

signature STACK = sig

(* A type for a stack with elements of type 'a *)

```
type 'a stack
```

(* Creates an empty stack *)

```
val empty: unit -> 'a stack
```

(* Pushes an element onto the stack *)

```
val push: 'a * 'a stack -> 'a stack
```

(* Removes the top element from the stack. Raises Empty if the stack is empty. *)

```
val pop: 'a stack -> 'a stack
```

(* Returns the top element of the stack without removing it. Raises Empty if the stack is empty. *)

```
val top: 'a stack -> 'a
```

(* Checks if the stack is empty *)

```
val isEmpty: 'a stack -> bool
```

(* Applies a function to each element of the stack, from top to bottom, and returns a new stack with the results *)

```
val map: ('a -> 'b) -> 'a stack -> 'b stack
```

end;

Implementation Task:

1. Define a structure that matches the `STACK` signature.
2. Implement the stack functionality using a list to store elements.
3. Ensure your implementation correctly handles empty stack cases.

Test Cases:

After implementing the `STACK` structure, create test cases that cover each of the following scenarios:

- Creating an empty stack and checking if it is indeed empty.
- Pushing one element, then multiple elements, and verifying the state of the stack after each operation.
- Popping elements from the stack until it is empty, checking the correctness of each removed element, and ensuring the `Empty` exception is raised when attempting to pop from an empty stack.
- Checking the top element of the stack without removing it and ensuring the `Empty` exception is raised when checking an empty stack.
- Testing the `isEmpty` function on both empty and non-empty stacks.
- Applying a `map` function to a stack, such as squaring numbers in a stack of integers, and verifying the output stack reflects the applied function without altering the original stack's state.

Submit your code including all the test cases.