Operations 4.0 and 4.1

Simon Yllmark

February 8, 2024

1 Introduction

This is a report for two assignments where one is called Operation on lists and the other Higher order functions. The main idea here is to implement functions that does operations on list, for the more advanced part which is the second assignment where it can take lists and functions as arguments, it can also take in other parameters but this is like the general description for them.

2 Recursive functions

For the first assignment which is the basic assignment I implemented a bunch of recursive functions. Here is a list of those functions as well a grouping of those function into three different groups.

```
#group 1
  def length2([]) do 0 end
  def length2([_|tail]) do 1 + length(tail) end

  def prod([]) do 1 end
  def prod([head | tail]) do head * prod(tail) end

  def sum([]) do 0 end
  def sum([head|tail]) do
    head + sum(tail)
  end

  #group 2
  def even([]) do [] end
  def even([head|tail]) do
    if (rem(head,2) == 0) do
      [head|even(tail)]
    else
```

```
      even(tail)
    end
  end

  def odd([]) do [] end
  def odd([head|tail]) do
    if (rem(head,2) == 1) do
      [head|odd(tail)]
    else
      odd(tail)
    end
  end

  def div2([], _), do: []
  def div2([head | tail], divisor) when rem(head, divisor) == 0 do
    [head | div2(tail, divisor)]
  end
  def div2([_head | tail], divisor) do div2(tail, divisor) end

  #group 3
  def inc([]) do [] end
  def inc([head | tail]) do [head + 1 | inc(tail)] end

  def dec([]) do [] end
  def dec([head | tail]) do [head - 1 | dec(tail)] end

  def mul([], _) do [] end
  def mul([head|tail], value) do [head * value | mul(tail, value)] end

  def rem2([], _) do [] end
  def rem2([head | tail], divisor) do [rem(head, divisor) | rem2(tail, divisor)] enc
```

The group group 1 are functions that return an integer. The functions in group one either calculates how long a list is, the sum of all the numbers in a list or the product of all the numbers in a list.

The group group 2 are functions that returns a modified list. The functions iterate through the hole list and check if all the element have a certain property like if they are even, odd or evenly divisible by a number.

The group group 3 are functions that returns a modified list but this time it does not check for properties but rather modifies the elements. Off course in Elixir all the elements immutable so the functions return a new list. The functions can increment, decrement, multiple all the elements in a list or calculate the reminder of an element by value x.

3 Higher order functions

Then we have these higher order function that can take functions as parameters. There where three main functions that the assignment instructed me to implement and those where:

1. map()
   (a) take a list of values and a function that when applied to a value returns another value. Create a new list with the resulting values when you apply the function on each value in the original list.
2. reduce()
   (a) apply the function on the value of the list and the accumulated value. Return the final value.
3. filter()
   (a) apply the function to all values of the given list and return a list with those values for which the function returned true.

Then I had to implement another functions that "takes a list of integers and returns the sum of the square of all values less than n" and how I solved this is shown below:

```
def sum4(list, n) do
  list
  |> filter(&(&1 < n))
  |> map(&(&1 * &1))
  |> sum()
end
```

This I think is a good example of how the code that I implemented works in an example that make use of two higher order function and one recursive function and use pipeline so the function does not need to create intermediate variables.

The function takes a list of integers and an integer and filter out all the values in that are equal or greater than that value, then it squares all those values in that list and finally sum all those values. Then the function return the sum of all the squared element in that list that are lesser than value n.

4 Conclusion

This I think was one of the if not the easiest assignment so far. It is kind of nice to have assignment that are easy ones in a while instead of like assignments like the interpreter which I still do not fully understand. Well, thank you for this easey assignment.