Operations on Lists

Ayda Ghalkhanbaz

Spring Term 2024

Introduction

The aim of this task is to implement a set of functions using recursive programming. These functions operate on lists such as modifying the elements in the list or calculating an attribute of the list. Implementation of this task may not be the hardest but it has a primary purpose that will be clear in the next (higher grade) task.

Method

The implementation of this program includes defining three groups of functions that operate on lists. Although the functions aren't grouped in the assignment, they can be categorised for simplicity. Those functions in the same group have an enormous similarity in their code therefore, we will discuss implementation of each group and presenting each group by showing the code snippet of one function from each group. Note that this categorisation is not the same as the one in the higher grade assignment.

First Group

The first group takes one argument and returns an integer indicating an attribute of the list, that is, sum/product of elements in the list or the length of the list. This is done by calling a helper function that takes two arguments; a list and the result accumulator (which can be an integer or a list depending on the function). This function utilises recursion and calculates the sum/product of elements or the length of the list using addition and multiplication. The result is then returned to the original function. The code snippet below demonstrates the `prod/1` function along its helper function `prod/2`. This function multiplies the elements of the list and returns the result.

```
# prod of the elements of an empty list is 1
def prod([]) do 1 end
```

```
def prod(list) do prod(list, 1) end

def prod([], res) do res end
def prod([head| tail], res) do
res = head * res
prod(tail, res)
end
```

Second Group

Next group's functions accept a list as an argument and return another list containing only even or odd numbers. These functions call a helper function that takes 2 arguments (the original list and an empty list), then utilises if-statements to compare and decide whether the current element in the list is even or odd. They will then add the target element to the accumulator list and call the function recursively to traverse the entire list. If the current element isn't the desired one, then they skip that element and continue with the rest of the list. Below, you'll find the function `even/1` together with `even/2` as an example of this group.

```
def divz([], _) do [] end
def divz(_, 0) do IO.puts("Error: Division with zero") end
def divz(list, number) do
  Enum.reverse(divz(list, number, []))
end
def divz([], _, divList) do divList end
def divz([head|tail], number, divList) do
  if rem(head,number) == 0 do
    divz(tail, number, [head | divList])
  else
    divz(tail, number, divList)
  end
end
```

Third Group

The third and last group has a similar structure as the second group. However, the functions in this group don't contain any comparison, instead they have to traverse the entire list and perform the desired operation on each element (such as addition, multiplication, division, etc.), add the modified element to the accumulator list and return it as the final result. The following

code snippet illustrates `inc/2` and its helper function `inc/3`, representing the third group's logic and structure.

```
  def inc([], _) do [] end
  def inc(list, 0) do list end
  def inc(list, number) do
    Enum.reverse(inc(list, number, []))
  end
  def inc([], _, incrList) do incrList end
  def inc([head|tail], number, incrList) do
    inc(tail, number, [head+number | incrList])
  end
```

Result

The run tests below illustrates the functionality of a function as each group's representation.

```
iex(6)> Reduce1.test()

The List: [1, 2, 3, 4, 5, 6]

-------- Product ----------
The Product of Elements in the list: 720

-------- Division ----------
The Division of Elements in the list with 3: [3, 6]

-------- Incremention ----------
The Incremention of Elements in the list with 2: [3, 4, 5, 6, 7, 8]
:ok
```