CS115 – Lab 4: Making Decisions

Spring 2020

Question 1: Subsets

Exercise

Create three functions named `in-subset-1?`, `in-subset-2?`, and `in-subset-3?`. Each consumes a `Num` and returns `#true` if the number is in the subset and `#false` otherwise:

1. $2 < x \leq 40$
2. $0 < x < 4$ or $8 < x < 31$
3. the numbers outside of $2 \leq x \leq 5$

Question 2: Group Pricing

Ticket costs at the Berlin Zoo are as follows:

- Adults pay 15.50 € to visit just the zoo, or 21.00 € for the zoo and aquarium.
- Children aged 4 to 15 pay 8.00 € to visit just the zoo, or 10.50 € for the zoo and aquarium.
- Infants under 4 are free.

Exercise

Create a function `(price ages aquarium?)`. `ages` is a `(listof Nat)` representing the ages of all the guests. `aquarium?` is a `Bool` indicating if the group will see the aquarium.
The function returns the total price for this group. `(price (list 35 8 2) #false) => 23.50`
`(price (list 38 11 5 2 2 2) #true) => 42.00`

Question 3: Keep Big Values

Exercise

In this question you will add up a list of values, but only counting the values that are "big" compared to the total seen so far.
Write a function `(keep-big L)` that consumes a `(listof Nat)`. Start with a total of zero, then starting at the right end of the list, and moving toward the left, add the next value *only if it is greater than the total so far*.

For example, consider `(keep-big (list 16 9 5 4 3))`:

- $3 > 0$, so add 3 to 0, giving 3;
- $4 > 3$, so add 4 to 3, giving 7;
- $5 \not> 7$, so ignore the 5, stay at 7;
- $9 > 7$, so add 9 to 7, giving 16;
- $16 \not> 16$, so ignore the new 16, stay at 16.

Hint

Use `foldr`.

Question 4: Sorting

Sorting algorithms are often studied in Computer Science classes because the problem is easy to describe, and there is a wide range of interesting algorithms.

In real life, however, you should almost certainly use the built-in sort function.

The function `sort` consumes a list `L`, and a predicate function `P`. `P` consumes two arguments; if it returns `#true`, the first argument appears in the final list to the left of the second argument.

For example:
`(sort (list 1 3 2 4) <) => (list 1 2 3 4)` since `(< 1 2)`, `(< 2 3)`, etc.
`(sort (list 1 3 2 4) >) => (list 4 3 2 1)` since `(> 4 3)`, `(> 3 2)`, etc.

To be able to sort items in order, we need to be able to tell if a pair of values is in order or not.
Here we will use the following sorting rule:

- even numbers come first, with smaller even numbers coming before larger even numbers;
- odd numbers come later, with larger odd numbers coming before smaller odd numbers.

Exercise

Write a predicate `(weird-order? a b)` that returns `#true` if `a` comes before `b`, and `#false` otherwise.
`(weird-order? 2 6) => #true (weird-order? 5 1) => #true (weird-order? 5 8) => #false`

This is all the built-in `sort` function requires to put a list in order:
`(sort (list 11 7 1 3 10 6 5 2 8) weird-order?) => (list 2 6 8 10 11 7 5 3 1)`

Exercise

Using `sort`, write a function `(sort-weird L)` that consumes a `(listof Int)` and sorts it according to the rule.
`(sort-weird (list 11 7 1 3 10 6 5 2 8)) => (list 2 6 8 10 11 7 5 3 1)`