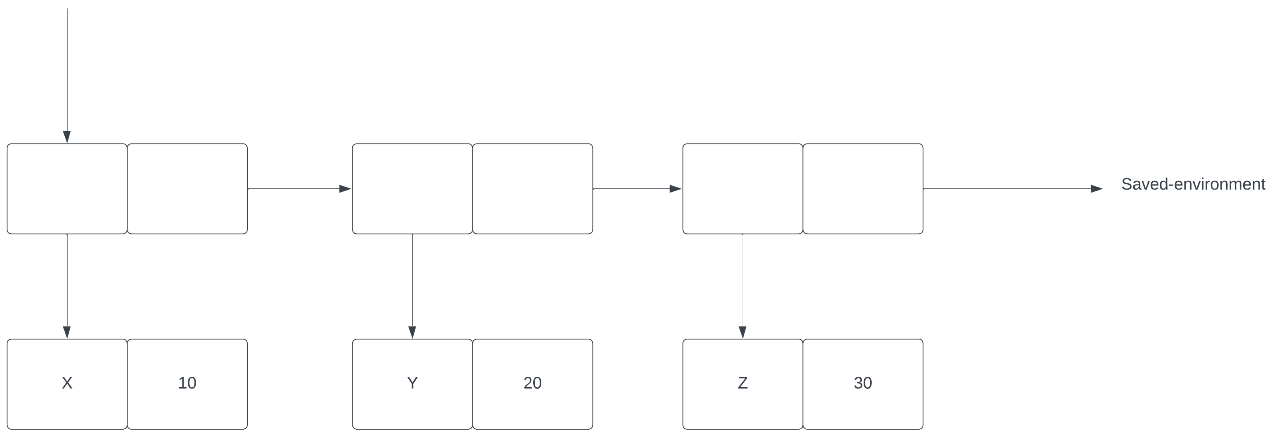
Part A

1) Create an initial environment that contains 3 different variables (x, y, and z).

```
> (define e (empty-env))
>> ('empty-env)
> (extend-env 'z (num-val 30) e)
>> ('z 30 (empty-env))
> (extend-env 'y (num-val 20) e)
>> ('y 20 ('z 30 (empty-env)))
> (extend-env 'x (num-val 10) e)
>> ('x 10 ('y 20 ('z 30 (empty-env))))
> (extend-env 'v (num-val 3) e)
>> ('v 3 ('x 10 ('y 20 ('z 30 (empty-env)))))
```

2) Using the environment abbreviation shown in the lectures, write how the environment changes at each variable addition.



Part B

Specify expressed and denoted values for MyProc language.

```
ExpVal = Int + Bool + Proc + Stack
DenVal = Int + Bool + Proc + Stack
```

Part C

All of the functionalities are implemented and all test-cases are being passed.

Part D

Assuming lists are implemented in an efficient, low-level way in Scheme, we don't really need to implement stacks in a low-level. Because lists in Scheme are essentially stacks. However, if we wanted to lets say implement a low-level, native array list in scheme, we would probably need a way to store the data of this structure first. We would need to store the integers in memory and a way to keep track of their place in the memory. So perhaps we could accomplish this with pointers?