1 Calculator

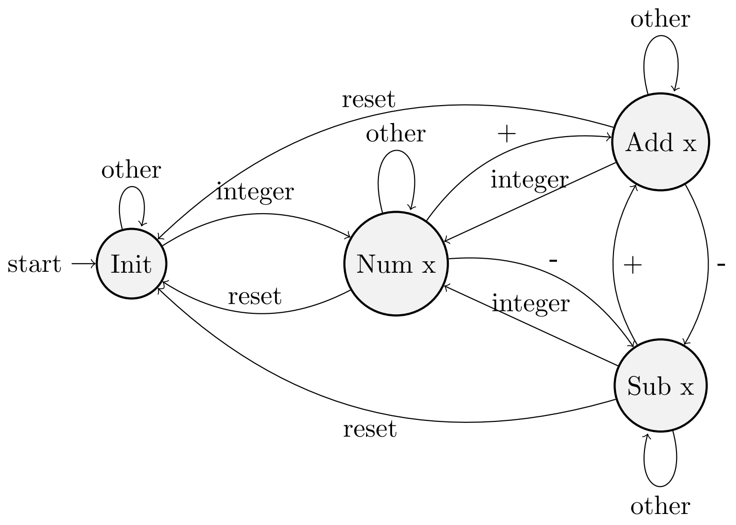
In this exercise we will implement a very simple calculator. We will use IO to simulate its inputs and outputs. After each input the intermediate result of the computation is shown. For example:

```
> 123
123
> +
123 +
> 34
157
> -
157 -
> 158
-1
...
```

The input operations this calculator will support are:

input	example string(s)
integer	`"42"`, `"-420"`, ...
plus	`"+"`
minus	`"-"`
reset	`"reset"`
exit	`"exit"`

The calculator does not receive the full expression we wish to calculate at once. The input is given line by line (e.g. `"123"` then `"+"` then `"321"`). So the calculator should somehow keep track of previous input. One way to achieve this is by modelling the calculator as some state machine:



Where `Init` represents the initial state, `Num x` represents the state where the calculator has a last known result `x`. `Sub x` and `Add x` represent that the last valid input was the minus (`"-"`) or the plus (`"+"`) operation and `x` is the last known result. The edges on the state machine represent the next state to go to given a certain input. For example if you receive an `integer`, e.g. `"26"`, in the `Init` state, you transition to the `Num x` state, where `x` is the input integer `26`.

Assignment 1: Implement `repr::State -> String` which returns a string representation of each state:

```
> repr Init

> repr (Num 42)
42
> repr (Add 42)
42 +
> repr (Sub 42)
42 -
```

Assignment 2: Implement `isInt::String -> Bool` which returns `True` if the given string represents an integer. The function shouldn't consider strings with brackets or decimals as integers. It should be able to recognize negative integer strings, e.g. `"-88"`.

```
> isInt "123"
True
> isInt "Hello"
False
> isInt "\n"
False
> isInt "-123"
True
> isInt "--123"
False
> isInt "42.0"
False
```

Assignment 3: Implement `transition::State -> String -> State` which performs a transition as shown in the above figure. When receiving invalid input, the calculator remains in it's current state. On the figure invalid input is categorised as *other*.

```
> transition Init "123"
Num 123
> transition Init "Hello"
Init
> transition (Add 1) "\n"
Add 1
```

```
> transition (Num 42) "-123"
Num (-123)
> transition (Num 42) "+"
Add 42
> transition (Num 42) "-"
Sub 42
> transition (Num 12) "hello"
Num 12
> transition (Add 42) "32"
Num 72
> transition (Add 10) "-88"
Num (-78)
> transition (Add 42) "-"
Sub 42
> transition (Num 42) "reset"
Init
> transition (Add 42) "reset"
Init
```

Assignment 4: Implement `calculator::IO ()` which reads user input line by line and displays the intermediate results until the user types `exit`, then the program termintates. Example:

```
> calculator

> 42
42
> +
42 +
> -
42 -
> 12
30
> "hello"
30
>
30
> reset

> -777
-777
> +
-777 +
> -223
-1000
> exit
Goodbye :)
```