Monad P3 : IORef Mutable Variable (2C)

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using LibreOffice/OpenOffice.

Based on

Haskell in 5 steps
https://wiki.haskell.org/Haskell_in_5_steps

Reading / writing updatable variable

every **name** in Haskell is <u>bound</u> to one **fixed** (**immutable**) **value**.

Sometimes it is easy to program, if **updatable variables** are used

> the **value** associated with a **variable**,
> can be <u>different</u> at <u>different</u> execution times,
> so reading its value can't be
> considered as a **pure function**

https://wiki.haskell.org/IO_inside#IO_actions_as_values

Problems with updatable variable

```
main = do let a0 = readVariable varA
               _  = writeVariable varA 1
              a1 = readVariable varA
          print (a0, a1)
```

Problems:

the two calls to '**readVariable**' look the same,
so the compiler reuses the result of the first call.

the result of the '**writeVariable**' call is not used
so the compiler omits this call completely.

these three calls may be rearranged in any order
because they appear to be **independent** of each other.

https://wiki.haskell.org/IO_inside#IO_actions_as_values

Use IO actions

Using IO actions guarantees that:

the **result** of the "same" **action** will not be reused

each action will have to be **executed**

the **execution order** will be retained as written

https://wiki.haskell.org/IO_inside#IO_actions_as_values

Solution – using IORef

```
import Data.IORef
main = do varA <- newIORef 0  -- Create and initialize a new variable
          a0 <- readIORef varA
          writeIORef varA 1
          a1 <- readIORef varA
          print (a0, a1)
```

https://wiki.haskell.org/IO_inside#IO_actions_as_values

IORef

varA has the type "**IORef Int**"

varA :: **IORef Int**

a **variable** (reference) in the **IO monad** holding a value of type **Int**

newIORef creates a new **variable** (**reference**) and returns it,
and then read/write actions use this **reference**.

The **value** returned by the **readIORef varA** action
depends not only on the **variable** involved
but also on the **time** this operation is performed
so it can return **different** values on **each call (not pure)**

```
import Data.IORef
main = do varA <- newIORef 0
          a0 <- readIORef varA
          writeIORef varA 1
          a1 <- readIORef varA
          print (a0, a1)
```

https://wiki.haskell.org/IO_inside#IO_actions_as_values

liftM

```
liftM :: (a -> b) -> (IO a -> IO b)

liftM f action = do x <- action
                    return (f x)
```

https://wiki.haskell.org/IO_inside#IO_actions_as_values

IO ( )

put :: s -> State s ( )

put :: s -> (State s) ( )

one value input type **s**

the effect-monad **State s**

the value output type **( )**

the operation is used *only for its effect*;

the *value* delivered is *uninteresting*

putStr :: String -> IO ()

delivers a string to stdout but does not return anything exciting.

https://stackoverflow.com/questions/16892570/what-is-in-haskell-exactly

IORef Definition (1)

```
newtype IORef a = IORef (STRef RealWorld a)

data    STRef s a = STRef (MutVar# s a)

data MutVar# s a
```

A MutVar# **behaves like a single-element mutable array.**

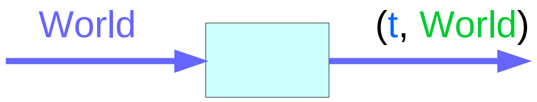
http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

IO (IORef a)

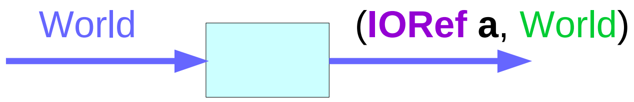
type IO t = World -> (t, World)

IO (IORef a)

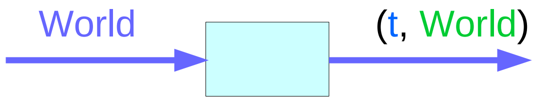
World -> (t, World)



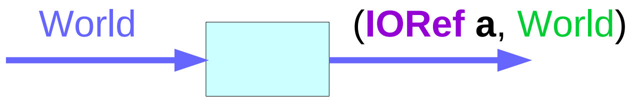
World -> (IORef a, World)



IO t type view



IO (IORef a) type view



https://www.cs.hmc.edu/~adavidso/monads.pdf

IORef Methods

data IORef a A mutable variable in the IO monad

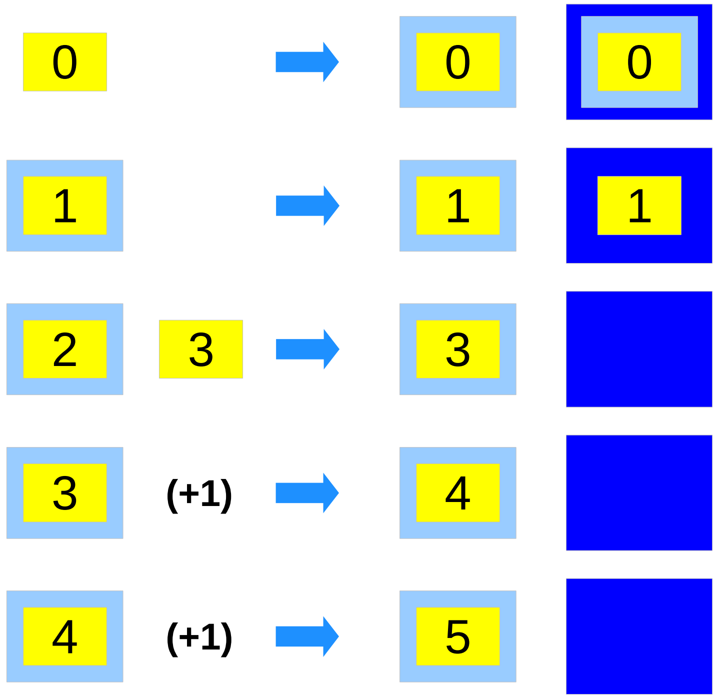
newIORef :: a -> IO (IORef a)
Build a new **IORef**

readIORef :: IORef a -> IO a
Read the value of an **IORef**

writeIORef :: IORef a -> a -> IO ()
Write a new value into an **IORef**

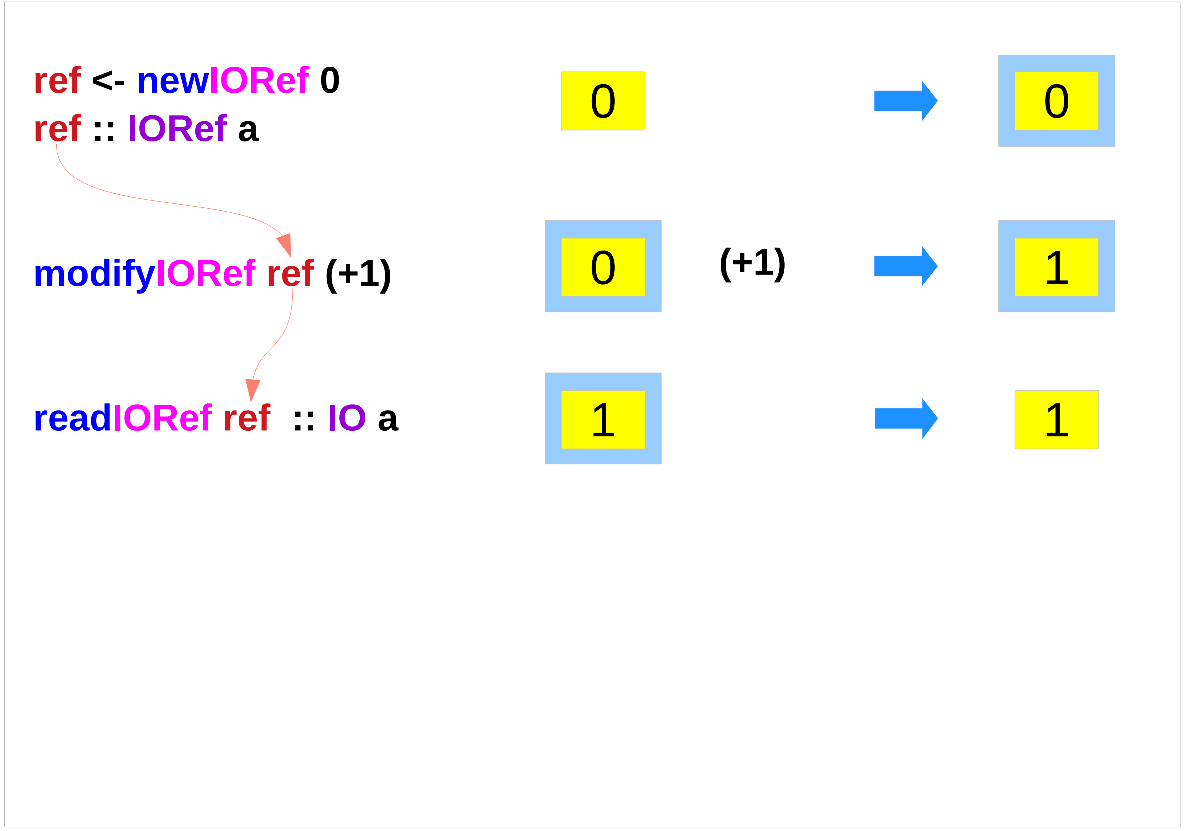
modifyIORef :: IORef a -> (a -> a) -> IO ()
Mutate the contents of an **IORef**.

modifyIORef‘ :: IORef a -> (a -> a) -> IO ()
Strict version of **modifyIORef**



http://hackage.haskell.org/package/base-4.12.0.0/docs/Data-IORef.html

IORef Usage



ref <- newIORef 0
ref :: IORef a

modifyIORef ref (+1)

readIORef ref :: IO a

http://hackage.haskell.org/package/base-4.12.0.0/docs/Data-IORef.html

IORef Example

```
newIORef :: a -> IO (IORef a)
newIORef 0 :: IO (IORef a)
ref <- newIORef 0
ref :: IORef a

(+1) :: (a -> a)
modifyIORef :: IORef a -> (a -> a) -> IO ()
modifyIORef ref (+1) :: IO ()

readIORef :: IORef a -> IO a
readIORef ref :: IO a


ref <- newIORef 0
replicateM_ 1000000 $ modifyIORef ref (+1)
readIORef ref >>= print
```

```
data IORef a

newIORef      :: a -> IO (IORef a)
readIORef     :: IORef a -> IO a
writeIORef    :: IORef a -> a -> IO ()
modifyIORef   :: IORef a -> (a -> a) -> IO ()
modifyIORef‘  :: IORef a -> (a -> a) -> IO ()
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/Data-IORef.html

IORef modifyIORef’

```
data IORef a

modifyIORef :: IORef a -> (a -> a) -> IO ()
```

Warning: **modifyIORef** does not apply the function strictly.
This means if the program calls **modifyIORef** many times,
but seldomly uses the value,
thunks will pile up in memory resulting in a **space leak**.
This is a common mistake made when using an **IORef** as a **counter**.
For example, the following will likely produce a **stack overflow**:

```
ref <- newIORef 0
replicateM_ 1000000 $ modifyIORef ref (+1)
readIORef ref >>= print
```

To avoid this problem, use **modifyIORef**' instead.

http://hackage.haskell.org/package/base-4.12.0.0/docs/Data-IORef.html

Global Variable Access Examples

```
import Data.IORef

type Counter = Int -> IO Int

makeCounter :: IO Counter
makeCounter = do
    r <- newIORef 0
    return (\i -> do modifyIORef r (+i)
                     readIORef r          )
```

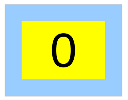
```
testCounter :: Counter -> IO ()
testCounter counter = do
        b <- counter 1
        c <- counter 1
        d <- counter 1
        print [b,c,d]

main = do
  counter <- makeCounter
  testCounter counter
  testCounter counter
```

https://stackoverflow.com/questions/16811376/simulate-global-variable

makeCounter

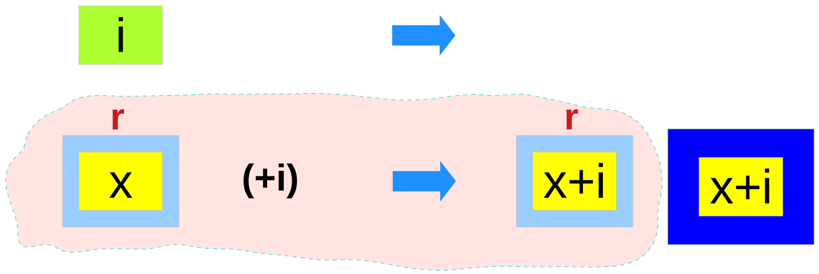
```
type Counter = Int -> IO Int

makeCounter :: IO v
makeCounter = do
   r <- newIORef 0
   return (\i -> do modifyIORef r (+i)
                    readIORef r          )
```

1. **create** r::IORef *once*

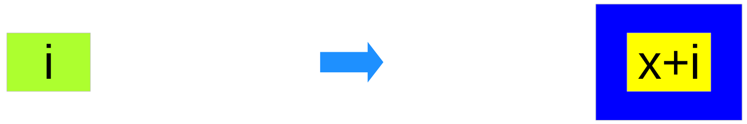


r <- newIORef 0

2. **returns** a function which <u>takes</u> an input **i** and <u>updates</u> r and <u>outputs</u> its modified value



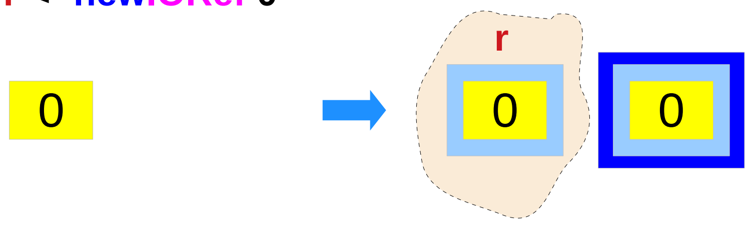
the underlying operation
using a global variable like r



https://stackoverflow.com/questions/16811376/simulate-global-variable

makeCounter creates a **r IORef** value

r <- newIORef 0



creation of **r::IORef** *once*

https://stackoverflow.com/questions/16811376/simulate-global-variable

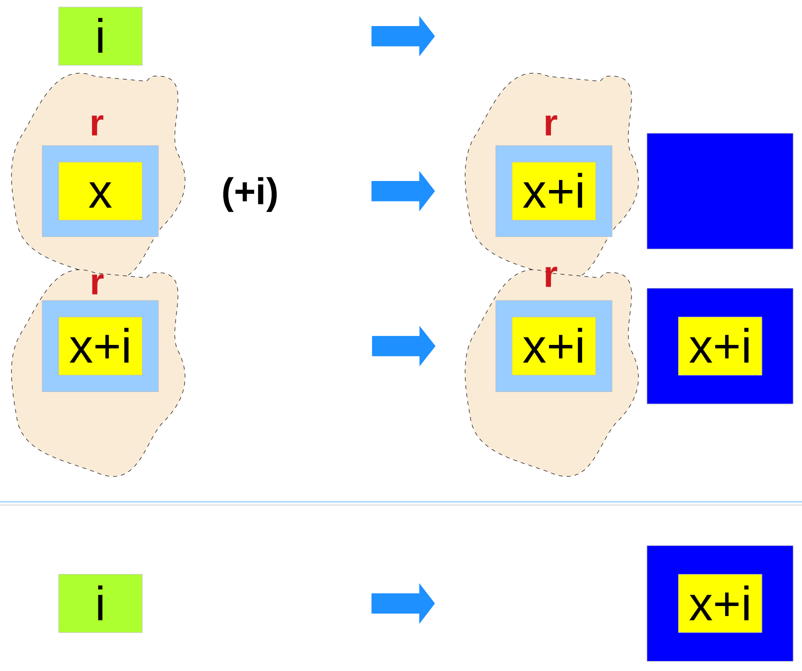
makeCounter returns a function

A function that is returned by **return**

\i -> do modifyIORef r (+i)

readIORef r

r refers the same **IORef** data value created by
r <- newIORef 0



modifyIORef r (+i)

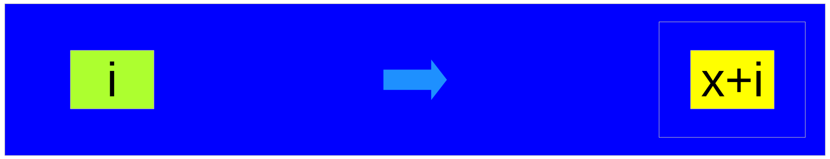
readIORef r

type signature of the returned function

https://stackoverflow.com/questions/16811376/simulate-global-variable

makeCoutner type signature

```
type Counter = Int -> IO Int


makeCounter :: IO Counter
makeCounter :: IO (Int -> IO Int)


makeCounter = do
    r <- newIORef 0
    return (\i -> do modifyIORef r (+i)
                     readIORef r          )


    return (a -> IO a)
    return (Int -> IO Int)
```

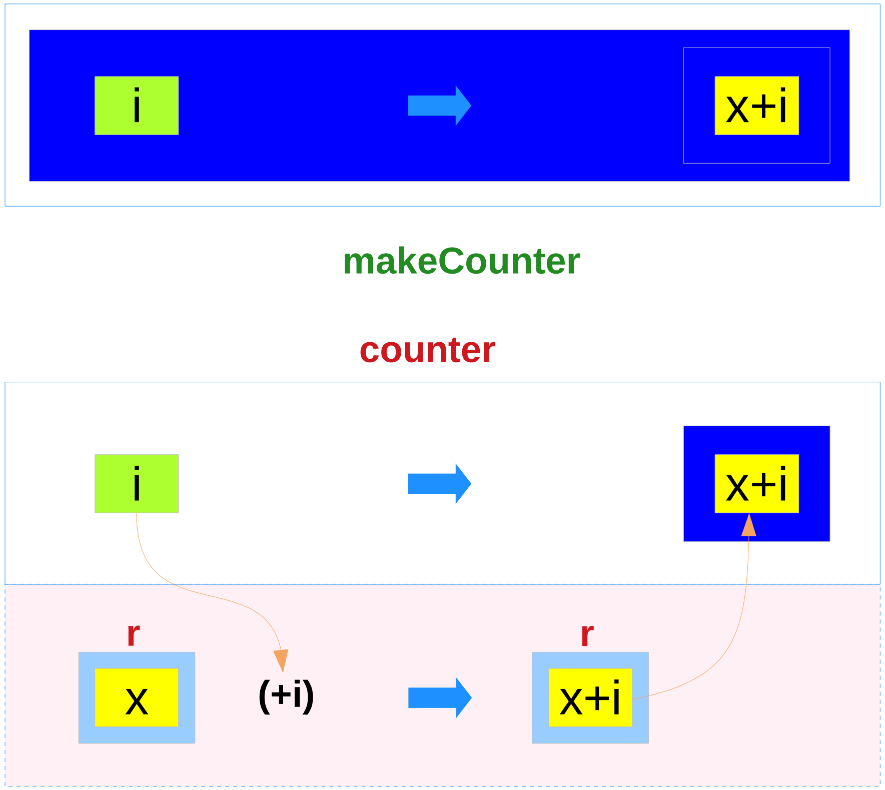


```
newIORef :: a -> IO (IORef a)
newIORef 0 :: IO (IORef a)
r <- newIORef 0
r :: IORef a

modifyIORef :: IORef a -> (a -> a) -> IO ()
modifyIORef r (+i) :: IO ()

readIORef :: IORef a -> IO a
readIORef r :: IO a
```

https://stackoverflow.com/questions/16811376/simulate-global-variable

counter function

```
type Counter = Int -> IO Int

makeCounter :: IO Counter
makeCounter :: IO (Int -> IO Int)

counter <- makeCounter
counter :: Int -> IO Int
```

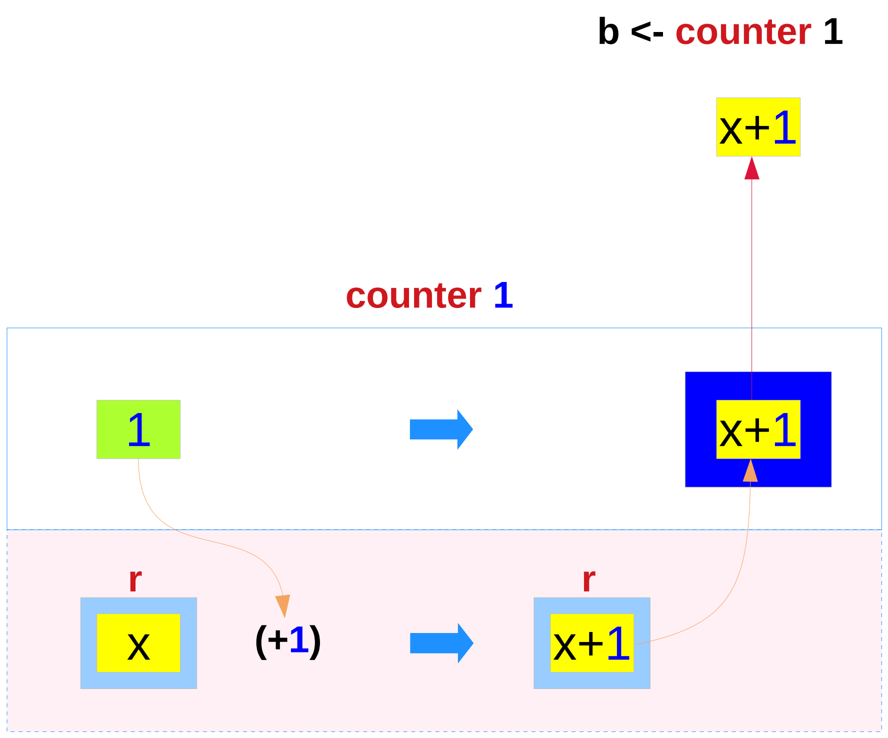


the underlying operation
using a global variable like r

https://stackoverflow.com/questions/16811376/simulate-global-variable

counter function application

```
type Counter = Int -> IO Int

testCounter :: Counter -> IO ()
testCounter :: Int -> IO Int -> IO ()
testCounter counter :: IO ()

counter :: Counter
counter :: Int -> IO Int
counter 1 :: IO Int
b <- counter 1
b :: Int
```

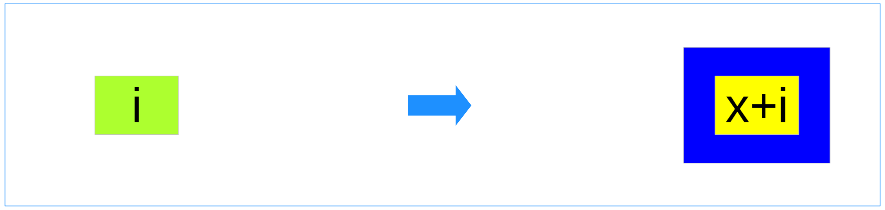


https://stackoverflow.com/questions/16811376/simulate-global-variable

testCounter

```
type Counter = Int -> IO Int

testCounter :: Counter -> IO ()
testCounter :: Int -> IO Int -> IO ()
testCounter counter :: IO ()

counter :: Counter
counter :: Int -> IO Int
counter 1 :: IO Int
b <- counter 1
b :: Int
```

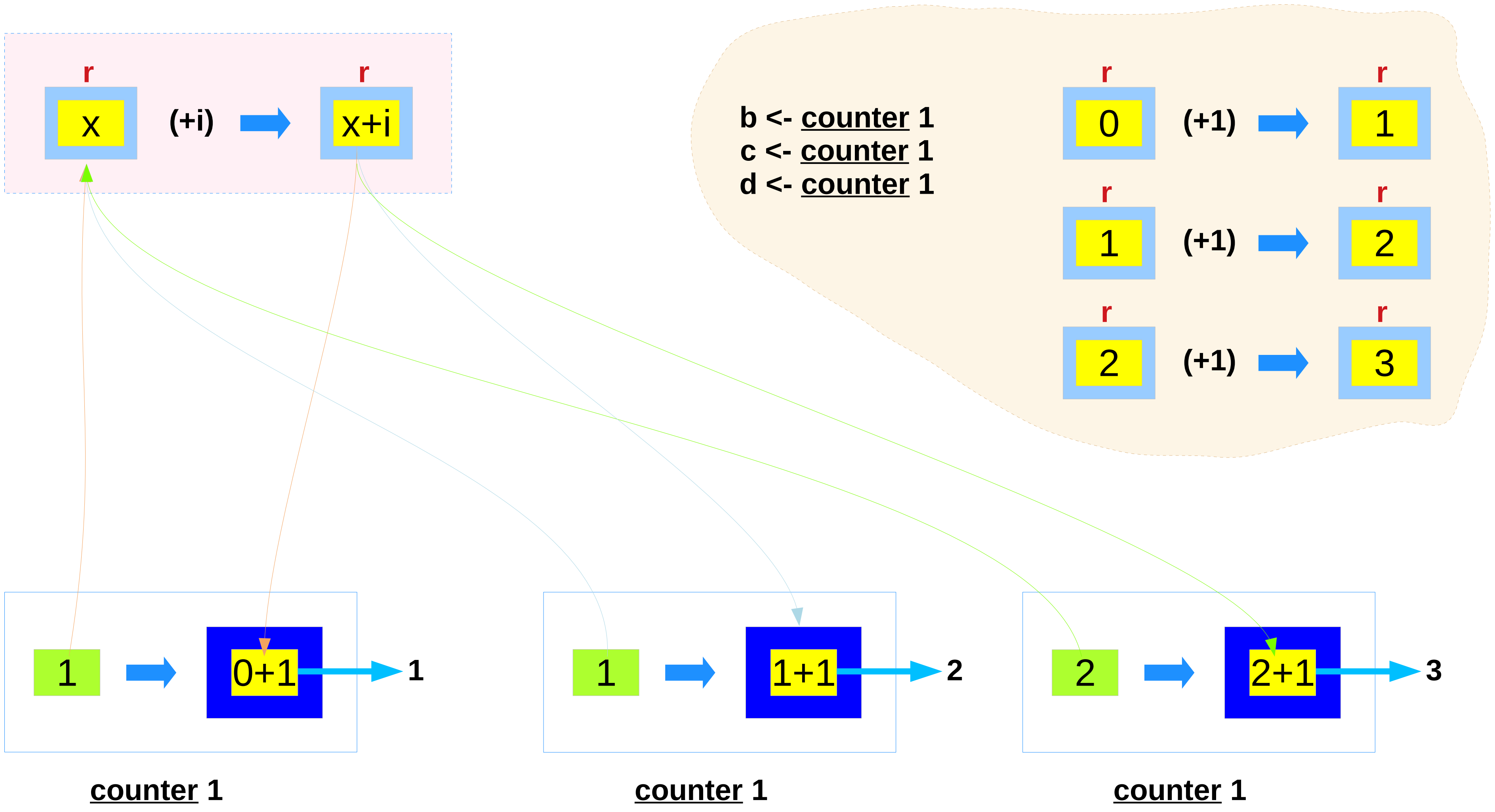
```
testCounter :: Counter -> IO ()
testCounter counter = do
        b <- counter 1                  :: Int
        c <- counter 1                  :: Int
        d <- counter 1                  :: Int
        print [b,c,d]
```



counter

https://stackoverflow.com/questions/16811376/simulate-global-variable

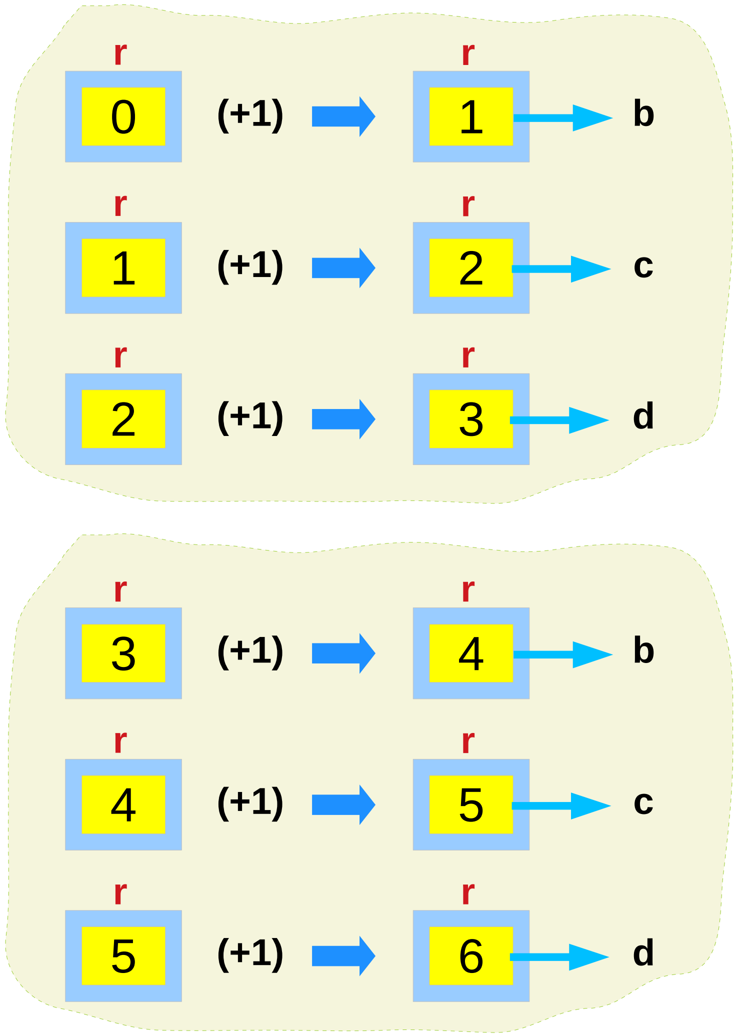
testCounter applies the **counter** function successively



https://stackoverflow.com/questions/16811376/simulate-global-variable

main

```
main = do
  counter <- makeCounter              :: Int -> IO Int
  testCounter counter                 :: IO ()
  testCounter counter                 :: IO ()

testCounter :: Counter -> IO ()
testCounter counter = do
        b <- counter 1
        c <- counter 1
        d <- counter 1
        print [b,c,d]
```



https://stackoverflow.com/questions/16811376/simulate-global-variable

IO ( )

put :: s -> State s ( )

put :: s -> (State s) ( )

one value input type **s**

the effect-monad **State s**

the value output type **( )**

the operation is used *only for its effect*;

the *value* delivered is *uninteresting*

putStr :: String -> IO ()

delivers a string to stdout but does not return anything exciting.

https://stackoverflow.com/questions/16892570/what-is-in-haskell-exactly

IORef Definition (1)

```
- |A mutable variable in the 'IO' monad
newtype IORef a = IORef (STRef RealWorld a)
  deriving Eq -- ^ @since 4.2.0.0
  -- ^ Pointer equality.
  --
  -- @since 4.1.0.0

-- |Build a new 'IORef'
newIORef    :: a -> IO (IORef a)
newIORef v = stToIO (newSTRef v) >>= \var -> return (IORef var)
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

IORef Definition (2)

```
-- |Read the value of an 'IORef'
readIORef   :: IORef a -> IO a
readIORef  (IORef var) = stToIO (readSTRef var)

-- |Write a new value into an 'IORef'
writeIORef  :: IORef a -> a -> IO ()
writeIORef (IORef var) v = stToIO (writeSTRef var v)

atomicModifyIORef :: IORef a -> (a -> (a,b)) -> IO b
atomicModifyIORef (IORef (STRef r#)) f = IO $ \s -> atomicModifyMutVar# r# f s
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

newIORef Method (1)

```
newIORef    :: a -> IO (IORef a)
newIORef v = stToIO (newSTRef v) >>= \var -> return (IORef var)
```

```
newtype IORef a = IORef (STRef RealWorld a)
stToIO :: ST RealWorld a -> IO a
stToIO (ST m) = IO m
newSTRef :: a -> ST s (STRef s a)
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

newIORef Method (2)

```
newIORef    :: a -> IO (IORef a)
newIORef v = stToIO (newSTRef v) >>= \var -> return (IORef var)



newSTRef :: a -> ST s (STRef s a)
newSTRef v :: ST s (STRef s a)
stToIO :: ST RealWorld a -> IO a                    a … STRef s a;   s … RealWorld

                ST RealWorld (STRef s a) -> IO (STRef s a)
                ST RealWorld (STRef RealWorlds a) -> IO (STRef RealWorld a)

stToIO (newSTRef v) :: IO (STRef RealWorld a)
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

newIORef Method (3)

```
newIORef    :: a -> IO (IORef a)
newIORef v = stToIO (newSTRef v) >>= \var -> return (IORef var)


stToIO (newSTRef v) :: IO (STRef RealWorld a)
stToIO (newSTRef v) >>= \var -> return (IORef var)
                var :: STRef RealWorld a


newtype IORef a = IORef (STRef RealWorld a)
                IORef var = IORef (STRef RealWorld a)
                IORef var :: IORef a
                return (IORef var) :: IO (IORef a)
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

readIORef (1)

```
readIORef   :: IORef a -> IO a
readIORef  (IORef var) = stToIO (readSTRef var)
```

```
newtype IORef a = IORef (STRef RealWorld a)
stToIO :: ST RealWorld a -> IO a
stToIO (ST m) = IO m
readSTRef :: STRef s a -> ST s a
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

readIORef (2)

```
readIORef   :: IORef a -> IO a
readIORef  (IORef var) = stToIO (readSTRef var)
        IORef var :: IORef a

newtype IORef a = IORef (STRef RealWorld a)
        var :: STRef RealWorld a

readSTRef :: STRef s a -> ST s a
readSTRef var :: ST RealWorld a
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

readIORef (3)

```
readIORef   :: IORef a -> IO a
readIORef  (IORef var) = stToIO (readSTRef var)


readSTRef var :: ST RealWorld a
stToIO :: ST RealWorld a -> IO a
stToIO (readSTRef var) :: IO a
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

writeIORef (1)

```
writeIORef  :: IORef a -> a -> IO ()
writeIORef (IORef var) v = stToIO (writeSTRef var v)
```

```
newtype IORef a = IORef (STRef RealWorld a)
stToIO :: ST RealWorld a -> IO a
stToIO (ST m) = IO m
writeSTRef :: STRef s a -> a -> ST s ()
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

writeIORef (2)

```
writeIORef  :: IORef a -> a -> IO ()
writeIORef (IORef var) v = stToIO (writeSTRef var v)
        IORef var :: IORef a
        v :: a

newtype IORef a = IORef (STRef RealWorld a)
        var :: STRef RealWorld a

writeSTRef :: STRef s a -> a -> ST s ()
writeSTRef var v :: ST RealWorld ()
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

writeIORef (3)

```
writeIORef  :: IORef a -> a -> IO ()
writeIORef (IORef var) v = stToIO (writeSTRef var v)

writeSTRef var v :: ST RealWorld ()
stToIO :: ST RealWorld a -> IO a
stToIO (readSTRef var) :: IO ()
```

http://hackage.haskell.org/package/base-4.12.0.0/docs/src/GHC.IORef.html#IORef

IORef

```
-- | A
```

https://osa1.net/posts/2016-07-25-IORef-STRef-exposed.html

References

[1] ftp://ftp.geoinfo.tuwien.ac.at/navratil/HaskellTutorial.pdf

[2] https://www.umiacs.umd.edu/~hal/docs/daume02yaht.pdf