```
                                t.Logf("Removed %d.  Expected %d\n", v, removed)
                                t.Fail()
                        }
                        removed++
                }
        }
}
```

```
// Unbounded buffer, where underlyin5 values are arbitrary values

package bufi

import (
        "errors"
)

// Linked list element
type BufEle struct {
        val interface{}
        next *BufEle
}

type Buf struct {
        head *BufEle         // Oldest element
        tail *BufEle         // Most recently inserted  re 4500(1)] *B.e New    ro     uf struc
 w(   )d  re 4500(1)] *B.e (bp     ro Iecent((        val inro f struct {)’ 11 TL ed  := &   ta
   .(    = ed  struct {)’ } struct {
    = ed  stre 4500(1)] *B.e (bp     ro Front ro      val in f struct {
    == n   e return (   } struct {)’ return bp.(   .(   stre 4500(1)] *B.e (bp     ro Remove ro
  ,     struct {

    = e.(    struct {)’ ’ 22    become unempty/terface{}
```

```
// Testing code for buffer
```

```
                        case int:
                                iv = v
                        case []byte:
                                iv = b2i(v)
                        defauof3i(v)
                        defauof3icvr2i(v)
                 ("Invalid data\n"fauof3i(v)
                        Fail(fauof3i(v)
                }uof3i(v)
                if    != remov]TJ{  defauof3icvr2i(v)
                 ("Remov]TJ%d.  Expect]TJ%d\n",   , remov]Tfauof3i(v)
                        Fail(fauof3i(v)
                }uof3i(v)
                remov]T++uof3i(v)
        }uof3i(v)
```

```
// Implementation of a UDP proxy

package main

import (
        "flag"
        "fmt"
        "log"
        "net"
        "os"
        "strings"
        "sync"
)

// Information maintained for each client/server connection
type Connection struct {
        ClientAddr *net.UDPAddr // Address of the client
        return true
}

func dlock() {
        dmutex.Lock()
}

func dunlock() {
        dmutex.Unlock()
}

// Go routine which manages connection from server to single client
func RunConnection(conn *Connection) {
        var buffer [1500]byte
        for {
                // Read from server
                n, err := conn.ServerConn.Read(buffer[0:])
                if checkreport(1, err) {
                        continue
                }
                // Relay it to client
                _, err = ProxyConn.WriteToUDP(buffer[0:n], conn.ClientAddr)
                if checkreport(1, err) {
                        continue
                }
                Vlogf(3, "Relayed ’%s’ from server to %s.\n",
                        string(buffer[0:n]), conn.ClientAddr.String())
        }
}

// Routine to handle inputs to Proxy port
func RunProxy() {
        var buffer [1500]byte
        for {
                n, cliaddr, err := ProxyConn.ReadFromUDP(buffer[0:])
                if checkreport(1, err) {
                        continue
                }
                Vlogf(3, "Read ’%s’ from client %s\n",
                        string(buffer[0:n]), cliaddr.String())
                saddr), ca3t Ra ) {
                if checkreporn RelNewom server toom serA
 dr.Stria3t Ra ) {
                if checkre     saddunr), ca3t Ra ) {
1, err) {
                                contiiiiiiiii{   fou Rao1ntr = Read f               contiiiiiii

2   Creat   new)
}

}


                if checkrenue

}
  (            ]), ca3t Ra ) {)’ 11 TLkre      ifdunr), ca3t Ra ) {

  outi     for {
 , err := ProxyConn.ReadFromUDP(buffer[0:])
                if checkreport(1, err) {
```

```
var verbosity int = 6

// Log result if verbosity level high enough
```