```
module My
using Parameters
using Random

@with_kw mutable struct Config
  char = (skip='?',less='>',more='<',num='$',klass='!')
  str  = (skip="?",)
  some = (max=512,step=.5, cohen=.3, trivial=1.05)
  seed = 1
end

THE = Config()
Random.seed!(THE.seed)

# ---------------------------------------
same(s) = s
int(x)  = floor(Int,x)
any(a)  = a[ int(size(a) * rand()) + 1 ]

function say(i)
  s,pre="$(typeof(i))(","
  for f in sort![x for x in fieldnames(typeof(i))
                 if !("$x"[1] == '_')]
    g = getfield(i,f)
    s = s * pre * "$f=$g"
    pre=", "
  end
  print(s * ")")
end

# ---------------------------------------

adds!(init=[],i=Some) = incs!(i(),init, 1)
subs!(init=[],i=Some) = incs!(i(),init,-1)
add!(i,x)             = inc!( i  ,   x, 1)
sub!(i,x)             = inc!( i  ,   x,-1)

# my cols can do:
#    incs!, inc!, statel, fresh, mid, var
# my cols know about:
#    w,pos,txt,w,key,n
incs!(i,init=[],w=1) = begin [inc!(i,x,w) for x in init]; i end

function inc!(i,x,w=1)
  y=i.key(x)
  if y != THE.str.skip
    stale(i)
    i.n += w
    inc!(i, y,w) end
end

# ---------------------------------------
@with_kw mutable struct Num
  pos=0; txt=""; w=1; key=same; n=0;
  lo=10^32; hi=-1*10^32; mu=0; m2=0; sd=nothing end

mid(i::Num)   = i.mu
stale(i::Num) = i.sd = nothing

function var(i::Num)
  if i.sd == nothing
    i.sd = i.n < 2 ? 0 : (i.m2 / (i.n - 1 + 10^-32))^0.5 end
  i.sd
end

function inc1!(i::Num,x)
  i.lo  = min(i.lo, x)
  i.hi  = max(i.hi, x)
  d     = x - i.mu
  i.mu += d / i.n
  i.m2 += d * (x - i.mu)
end

# ---------------------------------------

@with_kw mutable struct Some
  pos=0; txt=""; w=1; key=same; n=0;
  all=[]; max=THE.some.max ;tidy=false end

p(i::Some,n)   = begin fresh(i); i.all[int(n*length(i.all))+1]
       end
stale(i::Some) = i.tidy=false

function fresh(i::Some)
  if !i.tidy
    sort!(i.all)
    i.tidy=true end  end

has(i::Some,n) = begin fresh(i); i.all[n] end
mid(i::Some, lo=1, hi=length(i.all)) = has(i,int(lo+(hi-lo)*.5))

function var(i::Some,lo=1,hi=length(i.all))
  fresh(i)
  n10 = int(lo+(hi-lo)*.1) + 1
  n90 = int(lo+(hi-lo)*.9) + 1
  (i.all[n90] - i.all[n10])/2.7
end

function inc1!(i::Some, x,w=1)
  m = length(i.all)
  if m < i.max
    push!(i.all,x)
  elseif rand() < m/i.n
    i.all[ int(m*rand()) + 1 ] = x end
end

"If i.all is broken at the points listed in `a`
between `lo` and `hi`, what is the expected value?"
function xpect(i::Some,a,lo=1,hi=length(i.all))
  e1(x,y) = (y-x+1)/(hi-lo+1)*var(i,x,y)
  e,m = 0,lo
  for n in a
    e += e1(m,n)
    m = n+1
  end
  e + e1(m,hi)
end

div(i::Some) = begin fresh(i); div(i.all,i.key) end

# ---------------------------------------
@with_kw mutable struct Range
  lo=0; hi=0; _all=[]; start=0; stop=0; w=0; _kids=[] end

Base.show(io::IO, i::Range) = say(i)

"assumes lst is sorted"
function div(lst::Array,key=same)
  the = THE.some
  x(z)            = key(lst[int(z)])
  val(y,z,p=0.5) = x(y+(z-y)*p)
  var(y,z)        = (val(y,z,0.9) - val(y,z,0.1))/2.7
  function xchop(lo,hi,out=nothing)
    best = var(lo,hi)
    for j = lo+step:hi-step
      now, after = x(j), x(j+1)
      if now != after
        if after - start > epsilon
      if stop - now > epsilon
            if abs(val(lo,j) - val(j+1,hi)) > epsilon
          n1,n2 = j-lo+1, hi-j
              here  = (var(lo,j)*n1 + var(j+1,hi)*n2)/(n1+n2)
              if here*the.trivial < best
                best,out = here,j end end end end end end
    return out
  end
  function xchops(lo,hi,ranges, cut = chop(lo,hi))
    if cut == nothing
      push!(ranges, Range(lo=x(lo), hi=x(hi),
                          _all=lst[lo:hi],start=lo,stop=hi))
    else
      xchops(lo,    cut, ranges)
      xchops(cut+1, hi,  ranges) end
  end
  #---------------------------------------
  n                = length(lst)
  epsilon          = var(1,n) * the.cohen
  step, start, stop = int(n^the.step)-1, x(1), x(n)
  xchops(1,n,[])
end

function chops(lo,hi,ranges,chop)
    cut = chop(lo,hi)
    if cut == nothing
      push!(ranges, Range(lo=x(lo), hi=x(hi),
                          _all=lst[lo:hi],start=lo,stop=hi))
    else
      ychops(lo,    cut, ranges)
      ychops(cut+1, hi,  ranges) end
  end
end
#
#function unite(rs, y=same,better= <, yis=Num)
#  the = THE.some
#  all(x=yis(key=y),a=[])= begin [incs!(x,r._all) for r in a]; x
        end
#  function ychop(lo,hi,best,rs,out=nothing)
#    left = yis(key=y)
#    for j in lo:hi-1
#      l= all(x=left,[rs[j]])
#      rall(a=rs[j+1:hi])
#      now = (var(l)*l.n + var(r)*r.n)/(l.n + r.n)
#      if better(now*the.trivial, best)
#        best,out = now,j end end
#    out
#  end
#  f = (start,stop) -> ychop(start,stopr,)
#  chop(1,length(rs),[], var(all(ranges)))

#end
#
# ---------------------------------------
@with_kw mutable struct Sym
  pos=0; txt=""; w=1; key=same; n=0;
  seen=Dict();  mode=nothing; ent=nothing;  end

mid(i::Sym) = begin i.fresh(); i.mode end
var(i::Sym) = begin i.fresh(); i.ent  end

stale(i::Sym) = i.mode,i.ent = nothing,nothing
function fresh(i::Sym)
  if i.mode == nothing
    i.ent, most = 0,0
    for (k,n) in i.seen
      p = n/i.n
      i.ent -= p*log(2,p)
      if n > most most,i.mode = n.k end end end
 end

function inc1!(i::Sym,x,w=1)
  new = w + (haskey(i.seen, x) ? i.seen[x] : 0)
  i.seen[x] = max(new,0)
end

# ---------------------------------------
norm(i::Sym, x) = x
norm(i::Some,x) = begin fresh(i); (x-i.all[1])/(i.all[end]-i.all
       [1]) end

difference(i::Sym, x,y) = x==THE.string.skip ? 1 : x == y
function difference(i::Some,x,y, no = THE.string.skip)
  d(a,b) = begin a= norm(i,a); b= a<0.5 ? 1 : 0; abs(a-b) end
  if     x==no && y==no 1
  elseif x==no          d(y,x)
  elseif y==no          d(x,y)
  else                  abs(norm(i,x) - norm(i,y)) end
end


# ---------------------------------------
@with_kw struct Lines file; src=open(file) end

"Define an iterator that returns a comma-seperated file, one
 record at a time without loading the whole file into memory."
function Base.iterate(it::Lines, (n,want)=(1,[]))
  "Split on comma, coerce strings to numbers or strings, as
          approriate."
  coerce(s)  = map(coerce1, split(s,","))
  coerce1(s) = ((x = tryparse(Float64,s))==nothing) ? s : x

  "Coerce strings. If first row, check what columns we should
         use.
   Only return those columns."
  function cols(a)
    if n == 1
      want = [i for (i,s) in enumerate(a) if !('?' in s)] end
    [a[i] for i in want]
  end

  "Delete comments and whitespace. Lines ending in
   ',' are joined to the next. Skip empty lines."
  function row(txt="")
    while true
      if eof(it.src) return txt end
      new = readline(it.src)
      new = replace(new, r"([ \t\n]|#.*)"=>"")
      if sizeof(new) != 0
        txt *= new
        if txt[end] != ','
      return txt end end end end

  new = row()
  if sizeof(new) > 0
    (n, cols(coerce(new))) , (n+1,want) end
 end

#---------------------------------------
id=0

@with_kw mutable struct Tbl
  rows=[]; cols=Cols() end

@with_kw mutable struct Row
  cells=[]; cooked=[]; id=global id+= 1
end

say(Row())
say(Row())
@with_kw mutable struct Cols
  x = (all=[], nums=[], syms=[])
  y = (all=[], nums=[], syms=[], goals=[])
  klass=""
  all  = []; nums = []; syms = []; end


function table(file::String)
  t=Tbl()
  for (n,a) in Lines(file=file)
    n==1 ? head!(t,a) : row!(t,a) end
  t
end

function row!(i::Tbl,a)
  [add!(c,a[c.pos]) for c in i.cols.all]
  push!(i.rows, Row(cells=a) )
end

head!(i::Tbl,a) = [head!(i.cols,n,x) for (n,x) in enumerate(a)]

function head!(i::Cols, n,txt)
  the = THE.char
  goalp()  = the.less in txt || the.more in txt
  nump()   = the.num  in txt || goalp()
  yp()     = klassp() || goalp()
  klassp() = the.klass in txt
  x = nump() ? Some : Sym
  y = x(pos=n, txt=txt)
  if klassp() i.klass = y end
  if goalp()  push!(i.y.goals, y) end
  if nump()
    push!(i.nums,y); push!(yp() ? i.y.nums : i.x.nums, y)
  else
    push!(i.syms,y); push!(yp() ? i.y.syms : i.x.syms, y)
  end
  push!(yp()  ? i.y.all : i.x.all, y)
  push!(i.all, y)
end

#---------------------------------------

function tbl1(f="data/auto.csv")
  t = table(f)
  println("n ",length(t.rows))
  for col in t.cols.x.nums
   println(div(col)) #println(var(col)," ",col.all)
  end
end

function nums(f="data/auto.csv")
  t = table(f)
  #println(t.rows[end].cells)
  for num in t.cols.x.nums
    d=div(num)
    println(num.txt, " ",length(d))
    println(d)
  end
end

function sym1()
  s=Sym()
  [add!(s,x) for x in "aaaabbc"]
end

function Lines1(f="data/weather.csv")
  m=1
  print(m)
  for (n,tmp) in Lines(file= f)
     m += sizeof(tmp)  #println(n," ",tmp)
     if mod(n,1000) == 0 println(n,":",m) end
  end
  print(m)
end

function num1(x)
  if x<0.3 return 0.1 end
  if x<0.7 return 0.8 end
  return 0.9
 end

function numbers1(s=Some())
  [add!(s,num1(rand())) for i in 1:100]
  println([has(s,i) for i in div(s)])
end

function numbers2(n=2, s=Some())
  [add!(s,rand()^0.5) for i in 1:10^n]
  println([(i,has(s,i)) for i in div(s)])
end

#some1()
#sym1()
#@time tbl1("data/xomo10000.csv")
@time tbl1("data/weather.csv")
#@time nums("data/xomo10000.csv")
#numbers1()
#@time numbers2(3)
end
```