```
module molecular
use ifport
use struc
implicit none

contains
function separation(ds,L)
    real :: separation
    real, intent(in) :: ds, L
    if (ds .GT. 0.5*L) then
        separation = ds - L
    else if (ds .LT. -0.5*L) then
        separation = ds + L
    else
        separation = ds
    end if
end function separation
!-----------------------------------------------------------
subroutine pbc(pos,pos2,L,N)
    real, intent(inout), dimension(:)   :: pos,pos2
    real, intent(in)                    :: L
    integer, intent(in)                 :: N
    integer     :: i
    do i=1,N
        if (pos(i) .LT. 0.0) then
            pos(i) = pos(i) + L
            pos2(i) = pos2(i) + L
        else if (pos(i) .GT. L) then
            pos(i) = pos(i) - L
            pos2(i) = pos2(i) - L
        end if
    end do
end subroutine pbc
!-----------------------------------------------------------
subroutine inctemp(xnew,xold,R)
    real, intent(inout), dimension(:)   :: xnew
    real, intent(in), dimension(:)      :: xold
    real, intent(in)                    :: R
    xnew = xold - R*(xold - xnew)
end subroutine inctemp
!-----------------------------------------------------------
subroutine init(p1,p2,p_first,config)
    type(obj), intent(out), dimension(:) :: p1, p2, p_first
    type(conf), intent(in)                   :: config
    real :: grid, ri, rj
    integer :: n
    CALL RANDOM_SEED()
    grid = config%L / int(sqrt(real(config%n_particles)))
    n=1
    ri=grid/2.0
    do while (ri<config%L)
        rj=grid/2.0
        do  while (rj<config%L)
            if (n<=config%n_particles) then
                p2(n)%x = rj + (rand()-0.5)*grid*0.25
                p2(n)%y = ri + (rand()-0.5)*grid*0.25
                p2(n)%vx = (rand()-0.5)*config%v_max*2.0
                p2(n)%vy = (rand()-0.5)*config%v_max*2.0
                p1(n)%x = p2(n)%x - p2(n)%vx*config%dt
                p1(n)%y = p2(n)%y - p2(n)%vy*config%dt
            end if
            rj=rj+grid
```

```
            n=n+1
        end do
        ri=ri+grid
    end do
    p_first = p2
end subroutine init
!-----------------------------------------------------------------
subroutine force(fx,fy,dx,dy,r,potential,config)
    real, intent(in)    :: dx,dy,r
    real, intent(out)   :: fx,fy
    real, intent(out)   :: potential
    type(conf), intent(in) :: config
    real    :: f, R1,R2,R4,R6,R8,R12,R14
    real    :: x1,y1,x2,y2
    integer :: i, j, k
    f=0.0
    fx=0.0
    fy=0.0
    potential=0.0
    if (config%Lennar.EQ.1) then
        if (r.lt.config%Cutoff) then
            R1=1.0/r
            R4=R1*R1*R1*R1
            R6=R4*R1*R1
            R8=R6*R1*R1
            R12=R8*R4
            R14=R8*R6
            f =24.0*(2.0*R14 - R8)
            fx = f*dx
            fy = f*dy
            potential = 4.0*(R12 - R6)
        end if
    else
        k=nint(config%Cutoff/config%L)
        Do i=-k,k
            x1=dx+i*config%L
            x2=x1*x1
            do j=-k,k
                y1=dy+j*config%L
                y2=y1*y1
                R2=x2+y2
                R1=sqrt(R2)
                R1=1.0/R1
                R2=1.0/R2
                f = R1*R2
                fx = fx + x1*f
                fy = fy + y1*f
                potential = potential + R1
            end do
        end do
    end if
end subroutine force
!---------------------------------------------------------
subroutine accel(p,temp,gcum,config)
    type(obj), intent(inout), dimension(:)  :: p
    type(eng), intent(out)                  :: temp
    real, dimension(:), intent(inout)       :: gcum
    type(conf), intent(in)                  :: config
    integer :: i, j, ibin
    real :: dx, dy, fx, fy, rmax, potential, r
    rmax=config%L*0.5
    p%ax=0.0
```

```
    p%ay=0.0
    do i=1,config%n_particles-1
        do j=i+1,config%n_particles
            dx=separation(p(i)%x-p(j)%x,config%L)
            dy=separation(p(i)%y-p(j)%y,config%L)
            r = sqrt(dx*dx+dy*dy)
            call force(fx,fy,dx,dy,r,potential,config)
            p(i)%ax = p(i)%ax + fx
            p(i)%ay = p(i)%ay + fy
            p(j)%ax = p(j)%ax - fx
            p(j)%ay = p(j)%ay - fy
            if (r<rmax) then
                ibin = nint(r/config%dr) + 1
                gcum(ibin) = gcum(ibin) + 2.0/(config%VV)
            end if
            temp%p = temp%p + potential
            temp%virial = temp%virial + dx*p(i)%ax + dy*p(i)%ay
        end do
    end do
end subroutine accel
!-----------------------------------------------------------
subroutine verlet(p_old,p_current,p_new,p_first,temp,gcum,config)
    type(obj), intent(inout), dimension(:)  :: p_old,p_current,p_new
    type(obj), intent(in), dimension(:)     :: p_first
    type(eng), intent(out)                  :: temp
    type(conf), intent(in)                  :: config
    real, dimension(:), intent(inout)       :: gcum
    integer :: i

    call accel(p_current,temp,gcum,config)
    p_new%x = 2.0*p_current%x - p_old%x + p_current%ax*config%dt2
    p_new%y = 2.0*p_current%y - p_old%y + p_current%ay*config%dt2

    call inctemp(p_new%x,p_current%x,config%R)
    call inctemp(p_new%y,p_current%y,config%R)

    p_current%vx=(p_new%x - p_old%x)/(config%dt*2.0)
    p_current%vy=(p_new%y - p_old%y)/(config%dt*2.0)

    call pbc(p_new%x,p_current%x,config%L,config%n_particles)
    call pbc(p_new%y,p_current%y,config%L,config%n_particles)

    call energy(p_new,p_current,p_first,temp,config%n_particles,config%L)
    p_old=p_current
    p_current=p_new
end subroutine verlet
!-----------------------------------------------------------
subroutine VVerlet(p_current,p_first,temp,gcum,config)
    type(obj), intent(inout), dimension(:)  :: p_current
    type(obj), intent(in), dimension(:)     :: p_first
    type(eng), intent(out)                  :: temp
    type(conf), intent(in)                  :: config
    type(obj), dimension(:), allocatable    :: p_old
    real, dimension(:), intent(inout)       :: gcum
    real, dimension(:), allocatable         :: xnew, ynew
    integer :: i
    allocate(p_old(config%n_particles),xnew(config%n_particles),ynew(config%n_particles
))
    p_old = p_current

    call accel(p_current,temp,gcum,config)         ! new acceleration
    xnew = p_current%x + p_current%vx*config%dt + 0.5*p_current%ax*config%dt2
```

```
    ynew = p_current%y + p_current%vy*config%dt + 0.5*p_current%ay*config%dt2

    call inctemp(xnew,p_current%x,config%R)
    call inctemp(ynew,p_current%y,config%R)

    call pbc(xnew,p_old%x,config%L,config%n_particles)
    call pbc(ynew,p_old%y,config%L,config%n_particles)

    p_current%x = xnew
    p_current%y = ynew
    p_current%vx = p_current%vx + 0.5*p_current%ax*config%dt
    p_current%vy = p_current%vy + 0.5*p_current%ay*config%dt
    call accel(p_current,temp,gcum,config)
    p_current%vx = p_current%vx + 0.5*p_current%ax*config%dt
    p_current%vy = p_current%vy + 0.5*p_current%ay*config%dt
    call energy(p_current,p_old,p_first,temp,config%n_particles,config%L)
end subroutine VVerlet
!----------------------------------------------------------------------
subroutine energy(p_new,p_current,p_first,temp,n_particles,L)
    type(obj), dimension(:), intent(in)     :: p_new,p_current,p_first
    type(eng), intent(inout)                :: temp
    integer, intent(in)                     :: n_particles
    real, intent(in)                        :: L
    integer             :: i
    temp%k = sum(p_current%vx*p_current%vx+p_current%vy*p_current%vy)*0.5
    temp%e = temp%k + temp%p
    temp%t = temp%k / n_particles
    temp%pr = (temp%k + temp%virial/n_particles)/(L*L)
    temp%dr2 = sum((p_new%x - p_current%x)**2 + (p_new%y - p_current%y)**2)/n_particles
    do i=1,n_particles
        temp%r2 = temp%r2 + separation(p_new(i)%x - p_first(i)%x,L)**2 +
separation(p_new(i)%y - p_first(i)%y,L)**2
    end do
    temp%r2=temp%r2/n_particles
end subroutine energy
!---------------------------------------------------------------------
subroutine mean(temp,temp_mean,config)
    type(eng), intent(in), dimension(:) :: temp
    type(eng), intent(out)              :: temp_mean
    type(conf), intent(in)              :: config
    temp_mean%p=sum(temp%p)/config%timebond
    temp_mean%k=sum(temp%k)/config%timebond
    temp_mean%e=sum(temp%e)/config%timebond
    temp_mean%t=sum(temp%t)/config%timebond
    temp_mean%pr=sum(temp%pr)/config%timebond
    temp_mean%virial=sum(temp%virial)/config%timebond
    temp_mean%dr2 = sum(temp%dr2)/config%timebond
    temp_mean%r2 = sum(temp%r2)/config%timebond
end subroutine mean
!-----------------------------------------------------------
end module molecular
```