bonjour
J'ai un programme qui utilisent des modules enregistrés ailleurs. Je compile avec gfortran ces modules puis le code principal et je reçois le message suivant:

```
fbar(i,j,minus_i)=(1.d0-xi)*fbar(i,j,f1_i)+xi*f_star
       1
Error: Unclassifiable statement at (1)
ye2011.for:1400.41:

    &"density","rsd-density rsd-x rsd-y"
                                1
Error: Syntax error in WRITE statement at (1)
```

il a rejoint le programme et les modules . J'ai vraiment besoin d'aide. J'ai vraiment besoin d'aide. Est-ce que quelqu'un peut me faire offrir une solution. merci beaucoup. merci de votre patience et de votre soutien

```
Program LBM
        use ComPara, only: CP_max_t, CP_frame, CP_tol,
    & CP_time_step, CP_L2_err,lx, ly,gamma, Flag_BC_force
        use Node_data_D2Q9, only: macro, macro_temp
!use CellCulture
        implicit none
!integer ierr
         real*8 :: tic, toc, tic1, toc1
!real*8 t0,t1,t2
         integer time,max_t,frame,ti, min_t
         integer i, j, k, ierr
         character*40 flog
         character Flag_resume
         !-----------initialize
call CPU_time(tic)
         call init_parameters_comment
         ! call read_geometry
         call init_density
         !if(FLAG_BC_Pressure .OR. FLAG_BC_Velocity) then
         call init_BC
         !end if
         ! call init_CellCulture
         call init_FE
         call cal_FE_fluid
         call write_fluid_structure
         ! call FE_category
         ! call output_neutral
         ! stop
```

```
        ! call write_cell_structure
        call CPU_time(toc)
                write(*, '(/1x,a,e9.3,a)') "Elapsed time for initialization is ",toc-tic," seconds."

write(*,'(/1x,a)')"============================================================"

        !-----------LOOP
        call CPU_time(tic)
                 max_t=CP_max_t
                 frame=CP_frame
                 ti=0
                 fLog=".\Results\lbm.log"!!!enregistrement en .log log en min
        open(9,file=flog)
        !stop
                write(*,'(/1x,a)')"=======================MAIN
                LOOP============================"
                 write(9,'(/1x,a)')"=======================MAIN
                 LOOP============================"
        CP_L2_err=1.d0
        !cell_generation=0
        !do while(cell_no<cell_max_no .and. cell_generation<cell_max_generation)
        ! judge to resume or not
                 write(* ,'(/1x,a)')"Do you want to resume your calculation? (y) or (n)"
                 read(*,'(a)')Flag_resume
        !min_t=1
                if( Flag_resume=='y') then
        call read_cas_file(min_t)
                write(*,*)"Resume calculation from time step=", min_t
                open(9,file=fLog, STATUS = "OLD", ACTION = "WRITE", IOSTAT = ierr,
ACCESS="APPEND")
                write(9,*)"resume calculation"
                 else
                 open(9,file=fLog, STATUS = "NEW", ACTION = "WRITE", IOSTAT = ierr)
                 min_t=1
                 end if
        !main loop
                 do time=min_t,max_t+min_t
                 if(maxval(CP_L2_err)>CP_tol) then
                 call collision
                 call propagation
                 call bounceback
                 call BC_treatment
                 call cal_macro
                 if( time==min_t) then
        !call CPU_time(tic1)
                 macro_temp=macro
        !call CPU_time(toc1)
        !write(*, '(/1x,a,e9.3,a)') "Elapsed time for copy macro is ",toc1-tic1," seconds."
```

```
                    end if
                    if(mod(time,CP_time_step)==0) then

write(* ,'(/1x,a)')"=========================================================="
                    ti=ti+1
                    call cal_L2_error
                    call check(time,tic,toc,ti)
        !call write_fluid_cell_vel(cell_generation)
                    call FE_cal_vel_wss
                    call write_fluid_vel_wss(1)
                    call write_cas_file(time)
                    end if
        !write(*,*) time
                    end if
                    end do
        !initial the cell
        !if( cell_generation==0) then
        ! call init_CellCulture
        ! call write_cell_structure(cell_generation)
        !

        !end if
        call write_fluid_cell_vel(cell_generation)
        !call Cell_growth
        ! call write_cell_structure(cell_generation)
        ! CP_L2_err=1.d0
        ! write(*,*)"cell_generation=",cell_generation
        ! stop
        !end do
                    call del_mem!!!!! n'existe pas!!!
                    close(9)
                    end program LBM


!////////////////////////fin de programme principale/////////////////////////////////////////




!////////////////////////modules/////////////////////////////


!/////////////////ComPara/////////////////////

        !> define the common parameters
```

```
Module ComPara

!-----------LBM parameters-----------------



                integer :: lx, ly !<computation domain
                real*8 :: density, omega, force !< density, relexation parameter, body force
                 integer :: CP_max_t, CP_frame, CP_time_step
                 real*8 :: CP_tol
                 real*8 :: CP_L2_err(3)
            real*8 :: gamma
  ! the precondition parameters
  ! cf. I:\LBM\Reference\Optimization\PhysRevE.70.066706-1.pdf for more details




 !-----------mathematic constant------------



                real*8, parameter:: PI= 3.14159265358979323846d0 !< accurate enough :P
                real*8, parameter:: CP_Cs= 0.577350269189625764 51d0
                real*8, parameter:: CP_Cs_sq = 0.33333333333333333333d0
                real*8, parameter:: CP_bct= 0.5d0 !< for the upper and bottom boundary, esp. for
Poiseuille flow



  !----------functions------------------------



                 real*8 , external :: my_mod
                 logical, external :: fun_no_fracture
                 logical, external :: fun_is_fluid
                 integer, external :: fun_point_number
                 integer, external :: fun_4point_case
                 real*8 , external :: fun_distance
                 real*8 , external :: fun_theta
                 integer, external :: FE_node_xy
                 real*8 , external :: fun_cal_rate



  !----------BC flag--------------------



                 logical :: FLAG_Debug_Poisseuille=.True.
                 logical :: FLAG_BC_Pressure=.True.
```

```
                    logical :: FLAG_BC_Velocity=.True.
                    logical :: FLAG_BC_FH=.True.
                    logical :: FLAG_BC_force=.True.
                    real*8 :: BC_Pressure_inlet
                    real*8 :: BC_Pressure_outlet
                    end module ComPara




!///////////////FE////////////////////////////////




                    Module FE
                    implicit none
        !-----------Output: finite element---------
        !real*8 ,ALLOCATABLE, dimension(:, :) :: FE_node !< x,y coordinate for each all point
        !integer,ALLOCATABLE, dimension(:, :) :: FE_seq
        !< connectivity
                    real*8 ,ALLOCATABLE, dimension(:, :) :: FE_fluid_node !< x,y coordinate for
each fluid point
                    integer,ALLOCATABLE, dimension(:, :) :: FE_fluid_seq
        !< fluid connectivity
        !integer,ALLOCATABLE, dimension(:, :, :) :: FE_node_rate
        !integer,ALLOCATABLE, dimension(:, :) :: node_xy
                    real*8 ,ALLOCATABLE, dimension(:, :) :: FE_wss_node
                    integer,ALLOCATABLE, dimension(:, :) :: FE_wss_seq
                    real*8 ,ALLOCATABLE, dimension(:, :) :: FE_wss_area

        !< 4- sequence of fluid interface particles
        !< sequency of fluid particles

                    real*8 ,ALLOCATABLE, dimension(:, :) :: FE_wss_sort
                    integer,ALLOCATABLE, dimension(:, :) :: FE_fluid_seq_1
        !<----------flag ----------------------------
        !integer
        !----------count----------------------------
        !integer :: FE_seq_number=0!< the number of fluid meshes
                    integer :: FE_number(4)=0!< the number of fluid particles

        !< the number of fluid particles
                    integer :: FE_node_number(4)=0 !< the mount of 4-direction interface
                    integer :: FE_seq_fluid_number=0 !< the mount of total fluid nodes
```

```
                integer :: FE_total_node_number=0
                integer :: FE_wss_node_number=0
        !integer :: FE_wss_seq_number =0
                integer :: FE_wss_sort_i=10
        !functions
                end Module FE




!//////////////////////nodedata//////////////////////////



                Module Node_Data_D2Q9
                implicit none
                Type Node
        !>NodeType
        !!0 - fluid
        !!1 - solid
        !!2 - fluid, interface
        !!3 - solid, interface
                integer :: NodeType=0
        !record of queue
                integer :: rate_ij =0
        ! macroscopic density
        !real*8 :: density=0.0d0
        ! x,y-direction velocity
        !real*8 :: velocity(2)=0.0d0
                 end type Node
        !define the parameter in the node
                real*8 :: D2Q9_weight(0:8),D2Q9_c(0:8,2)
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: fbar
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: feq
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: macro
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: macro_temp
                type(Node), pointer ,dimension(:,:) :: pNode
        !integer,ALLOCATABLE, dimension(:, :) :: NodeType
                end module Node_Data_D2Q9



!//////////////////bondary condition//////////////////
```

```
! Boundary Condition module
              Module BoundaryCondition
              implicit none
              type BC_Velocity
! Zou & He (POF 1997)
! i,j -- position
! index: id
!
!1 -- east
! 2 -- north
! 3 -- west
! 4 -- south
! u_x,u_y: given velocity
               integer :: i,j,id
               real*8 :: u_x,u_y
               end type BC_Velocity
               Type BC_Pressure
! Zou & He (POF 1997)
! i,j -- position
! index: id
! 1 -- east
! 2 -- north
! 3 -- west
! 4 -- south
! u_1: given velocity along the boundary
! rou:
                   integer :: i,j,id
                    real*8 :: u_1=0.d0, rou
                    end type BC_Pressure
!> Filippova and Hanel boundary treatment
                    Type BC_FH
!!NodeType
!!0 - fluid
!!1 - solid
!!2 - fluid, interface
!!3 - solid, interface
!integer :: NodeType=0
!!Boundary condition type
!!x,y-direction distance to the node
!real*8 ::rate(8)=0.0d0
                    integer :: i,j
                    integer :: id(0:4)=0
                    real*8 :: rate(8)=0.d0
                   end type BC_FH

                   type(BC_velocity), pointer:: pBC_velocity(:)
                   type(BC_pressure), pointer:: pBC_pressure(:)
                   type(BC_FH),pointer:: pBC_FH(:)
```

```
                    end module BoundaryCondition



!//////////////////////////////cellculture//////////////////////////////



    ! Cell Culture module
                    Module CellCulture
                    implicit none
                    type cell
                    integer :: i,j
                    real*8 :: p, wss
                    end type cell
                    type(cell), pointer::pCell(:)
                    integer:: cell_no, cell_x_min, cell_x_max
                    integer:: cell_max_no, cell_max_generation, cell_generation
                    real*8 :: cell_max_wss, cell_dead_wss
                    real*8 , external :: fun_wss_pro
                    end module



!////////////////////////////// subroutines////////////////////////////////



!////////////init_parameters_comment/////////////////



    subroutine init_parameters_comment

      ! initialize the parameter
      use ComPara
      use Node_Data_D2Q9
      use FE, only: FE_wss_sort_i
      implicit none
      integer :: ierr
      real*8 :: t0
      namelist /parameter/ lx,ly, density, omega, force, &
```

```
        FLAG_Debug_Poisseuille, &
        FLAG_BC_Pressure, FLAG_BC_Velocity, FLAG_BC_FH, FLAG_BC_force, &
        CP_max_t, CP_frame, CP_time_step, CP_tol, &
        FE_wss_sort_i

write(*,'(/1x,a)')"============================================================="
        write(*,'(/1x,a)')"initize the parameter from parameter.in"
        !open(1, file='.\input\parameter.in', STATUS = "OLD", ACTION = "READ", &
        IOSTAT = ierr)
        if( ierr==0) then
        ! computation domain
        read(1,*)
        read(1,*)lx

        read(1,*)
        read(1,*)ly
        read(1,*)
        read(1,*)density
        read(1,*)
        read(1,*)omega
        read(1,*)
        read(1,*)force
        ! boundary condition flag
        !
        read(1,*)
        !
        read(1,*)BC_Pressure_inlet
        !
        read(1,*)
        ! read(1,*)BC_Pressure_outlet
        ! ! Boundary condition
        read(1,*)
        read(1,*)FLAG_Debug_Poisseuille
        read(1,*)
        read(1,*)FLAG_BC_Pressure
        read(1,*)
        read(1,*)FLAG_BC_Velocity
        read(1,*)
        read(1,*)FLAG_BC_FH
        read(1,*)
        read(1,*)FLAG_BC_force
        ! temporal parameter
        read(1,*)
        read(1,*)CP_max_t
        read(1,*)
        read(1,*)CP_frame
        read(1,*)
        read(1,*)CP_time_step
        read(1,*)
```

```
read(1,*)CP_tol
!
read(1,*)
!
read(1,*)FE_wss_sort_i
end if
close(1)
!-------report the parameter
open(1, file='.\results\parameter.out')
write(1,nml=parameter)
close(1)
!-------allocate fbar

allocate(fbar(lx,ly,0:8),stat=ierr)
if( ierr/=0) then
write(*,'(/1x,a)')"Allocate fbar FAILED."
stop
end if
!-------allocate feq
allocate(feq(lx,ly,0:8),stat=ierr)
if( ierr/=0) then
write(*,'(/1x,a)')"Allocated feq FAILED."
stop
end if
!-------allocate macro
allocate(macro(lx,ly,3),stat=ierr)
if( ierr/=0) then
write(*,'(/1x,a)')"Allocated macro FAILED."
stop
end if
!-------allocate macro_temp
allocate(macro_temp(lx,ly,3),stat=ierr)
if( ierr/=0) then
write(*,'(/1x,a)')"Allocated macro_temp FAILED."
stop
end if
!-------allocate NodeType
allocate(pNode(lx,ly),stat=ierr)
if( ierr/=0) then
write(*,'(/1x,a)')"Allocated pNode FAILED."
stop
end if
!-------init weight and lattice unit velocity
t0=1.d0/36.d0
D2Q9_weight(0)=t0*16.d0
D2Q9_weight(1:4)=t0*4.d0
D2Q9_weight(5:8)=t0
D2Q9_c((/0,2,4/),1)=0.d0
D2Q9_c((/1,5,8/),1)=1.d0
```

```
D2Q9_c((/3,6,7/),1)=-1.d0
D2Q9_c((/0,1,3/),2)=0.d0
D2Q9_c((/2,5,6/),2)=1.d0
D2Q9_c((/4,7,8/),2)=-1.d0
end subroutine init_parameters_comment




!//////////////////init_density//////////////////



!initialize the variable of each node
subroutine init_density
use ComPara, only : density, lx, ly, CP_cs_sq, BC_Pressure_inlet, gamma
use Node_Data_D2Q9
implicit none
integer i,j,k
real*8 :: u_n(8),u_x, u_y, u_squ, d_loc, u_max
write(*,'(/1x,a)') 'initializing the equilibrium distribution function feq and fbar'
macro(:,:,2)=BC_Pressure_inlet
macro(:,:,3)=0.d0
!macro(:,:,1)=dsqrt(macro(:,:,2)*macro(:,:,2)+macro(:,:,3)*macro(:,:,3))
!u_max=MAXVAL(macro(:,:,1))
!gamma=(10.d0*dsqrt(3.d0)*u_max)**2
macro(:,:,1)=density
do j=1, ly, 1
do i=1, lx, 1
d_loc= macro(i,j,1)
u_x = macro(i,j,2)
u_y = macro(i,j,3)
u_n(1) =u_x
u_n(2) =u_y
u_n(3) = - u_x
u_n(4) = - u_y
u_n(5) = u_x + u_y
u_n(6) = - u_x + u_y
u_n(7) = - u_x - u_y
u_n(8) =u_x - u_y
u_squ = u_x * u_x + u_y * u_y
!.....zero velocity density
feq(i,j,0) = D2Q9_weight(0) * d_loc * (1.d0 - u_squ / (2.d0 * CP_cs_sq))
do k=1,8
```

```
        feq(i,j,k) = D2Q9_weight(k)* d_loc * (1.d0 + u_n(k) / CP_cs_sq &+ u_n(k) ** 2.d0 / (2.d0 *
CP_cs_sq ** 2.d0) &- u_squ / (2.d0 * CP_cs_sq) )
        end do
        end do
        end do

        fbar(:,:,:)=feq(:,:,:)
        !write(1,*) 'initial density'
        end subroutine init_density




!/////////////// init_BC//////////////////////




        ! initial boundary condition from different file
        subroutine init_BC
        use ComPara, only: FLAG_BC_Velocity, FLAG_BC_Pressure, flag_BC_FH
        !use BoundaryCondition
        implicit none
        !integer i,j, ierr, p_i,v_i
        write(*,'(/1x,a)') 'Initializing Boundary Conditions...'
        if( flag_BC_Pressure==.True.) then
        call init_Pressure
        end if
        if( flag_BC_Velocity==.True.) then
        call init_Velocity
        end if
        if( flag_BC_FH==.True.) then
        call init_FH
        end if
        end subroutine init_BC
```

```
!//////////////////////init_Pressure//////////////////

! initial pressure from BC_pressure.in file
subroutine init_Pressure
use ComPara, only: ly
use BoundaryCondition
implicit none
integer ierr, p_i, i
real*8 BC_Pressure_outlet, BC_Pressure_inlet
write(*,'(/1x,a)') 'initializing pressure and velocity boundary condition'
!init pressure outlet
! Poisseuille flow
if( FLAG_Debug_Poisseuille==.True.) then
!
inlet
!p_i=2*(ly-2)
p_i=ly-2
BC_Pressure_inlet=0.8d0
BC_Pressure_outlet=0.79d0
if(FLAG_BC_Pressure) then
allocate(pBC_pressure(p_i),stat=ierr)
if(ierr /= 0) then
write(*,'(/1x,a)')"Allocated pBC_pressure FAILED."

end if
do i=1,ly-2
pBC_pressure(i)%i=lx
pBC_pressure(i)%j=i+1
pBC_pressure(i)%id=1
pBC_pressure(i)%u_1=0.0d0
pBC_pressure(i)%rou=BC_Pressure_outlet
end do
do i=ly-1,p_i
pBC_pressure(i)%i=1
pBC_pressure(i)%j=i-ly+2
pBC_pressure(i)%id=3
pBC_pressure(i)%u_1=0.0d0
pBC_pressure(i)%rou=BC_Pressure_inlet
end do
end if
else
open(1, file='.\input\BC_pressure.in', STATUS = "OLD", ACTION = "READ", &
IOSTAT = ierr)
if( ierr /= 0) then
```

```
write(*,'(/1x,a)')"Open file BC_pressure.in FAILED."
STOP
end if
read(1,*)p_i
allocate(pBC_pressure(p_i),stat==ierr)
if(ierr /= 0) then
write(*,'(/1x,a)')"Allocated pBC_pressure FAILED."
end if
!
# read comments
read(1,*)
do i=1,p_i
read(1,*) pBC_Pressure(i)%i, pBC_Pressure(i)%j,pBC_Pressure(i)%id, pBC_Pressure(i)%u_1,
pBC_Pressure(i)%rou
end do
end if
end subroutine init_Pressure
```

```
!//////////////////////init_Velocity//////////////////////
```

```
! initial velocity from BC_velocity.in file
subroutine init_Velocity
!init velocity inlet
!v_i=2*(ly-2)
implicit none

integer v_i,i
logical flag_BC_velocity_DEBUG
flag_BC_Velocity_DEBUG=.False.
If( flag_BC_Velocity_DEBUG) then
v_i=ly-2
!
if(FLAG_BC_Velocity) then
allocate(pBC_velocity(v_i),stat=ierr)
if(ierr /= 0) then
write(*,'(/1x,a)')"Allocated pBC_Velocity is unsuccessful."
end if
do i=1,ly-2
```

```
pBC_velocity(i)%i=1
pBC_velocity(i)%j=i+1
pBC_velocity(i)%id=3
pBC_velocity(i)%u_x=1.d-3
pBC_velocity(i)%u_y=0.d0
end do
!
do i=ly-1,v_i
! pBC_velocity(i)%i=lx
! pBC_velocity(i)%j=i-ly+2
! pBC_velocity(i)%id=1
! pBC_velocity(i)%u_x=1.d-3
! pBC_velocity(i)%u_y=0.d0
!
end do
else
open(1, file='.\input\BC_velocity.in', STATUS = "OLD", ACTION = "READ", &
IOSTAT = ierr)
if( ierr /= 0) then
write(*,'(/1x,a)')"Open file BC_velocity.in FAILED."
STOP
end if
read(1,*)v_i
allocate(pBC_Velocity(v_i),stat==ierr)
if(ierr /= 0) then
write(*,'(/1x,a)')"Allocated pBC_pressure FAILED."
end if
!
# read comments
read(1,*)
do i=1,v_i
read(1,*) pBC_Velocity(i)%i, pBC_Velocity(i)%j,pBC_Velocity(i)%id, pBC_Velocity(i)%u_x,
pBC_Velocity(i)%u_y
end do
end if

end subroutine init_Velocity
```

!////////////////////////////init_FH//////////////////

```
        subroutine init_FH

        !initialize the geometry
        use ComPara
        use Node_Data_D2Q9
        use BoundaryCondition
        use FE, only: FE_number, FE_node_number
        implicit none
        integer ierr
        integer i,j,k, FH_number(0:4), temp , temp1!, temp2
        write(*,'(/1x,a)')"Reading geometry from .\input\FH_indicator.dat & FH_rate_file.dat "
        !-----------open the indicator.dat file
        open(1,file='.\input\FH_indicator.dat', STATUS = "OLD", ACTION = "READ", &IOSTAT =
ierr)
        if( ierr/=0) then
        write(*,'(/1x,a)')"read file .\input\FH_indicator.dat FAILED."
        stop
        end if
        !open(2,file='.\input\FH_rate_file.dat', STATUS = "OLD", ACTION = "READ", &IOSTAT =
ierr)
        !----------read file
        read(1, *)
        read(1, *)FH_number(0)
        ! allocate node
        allocate(pBC_FH(0:FH_number(0)),stat=ierr)
        if( ierr/=0) then
        write(*,'(/1x,a)')"Allocate pBC_FH FAILED."
        stop
        end if
        ! read NodeType from indicator.dat
        read(1, *)FH_number(1)
        read(1, *)FH_number(2)
        read(1, *)FH_number(3)
        read(1, *)FH_number(4)
        ! temp1=0
        ! temp2=0

        do i=1,lx
        do j=1,ly
        !write(*,*) i,j
        read(1,*)pNode(i,ly+1-j)%NodeType
        select case(pNode(i,ly+1-j)%NodeType)
        case(0)
        FE_number(1)=FE_number(1)+1
        pNode(i,ly+1-j)%rate_ij=FH_number(3)+FH_number(4)+FE_number(1)
        case(1)
        FE_number(2)=FE_number(2)+1
        pNode(i,ly+1-j)%rate_ij=FE_number(2)+FH_number(3)+FH_number(4)+FH_number(1)
```

```
case(2)
FE_number(3)=FE_number(3)+1
pNode(i,ly+1-j)%rate_ij=FE_number(3)
pBC_FH(FE_number(3))%id(0)=FE_number(3)
read(2, *) pBC_FH(FE_number(3))%i
read(2, *) temp1
pBC_FH(FE_number(3))%j=ly+1-temp1
read(2, '(8e14.9)') pBC_FH(FE_number(3))%rate(:)
do k=1,4
if(pBC_FH(FE_number(3))%rate(k)>0.d0) then
FE_node_number(k)=FE_node_number(k)+1
pBC_FH(FE_number(3))%id(k) =FE_node_number(k)
end if
end do
case(3)
FE_number(4)=FE_number(4)+1
pNode(i,ly+1-j)%rate_ij=FE_number(4)+FH_number(3)
temp=pNode(i,ly+1-j)%rate_ij
read(2, *) pBC_FH(temp)%i
read(2, *) temp1
pBC_FH(temp)%j=ly+1-temp1
read(2, '(8e14.9)') pBC_FH(temp)%rate(:)
end select
end do
end do
!close(1)
!close(2)
!-----check the input file

if( maxval(ABS(FE_number(1:4)-FH_number(1:4)))>0) then
write(*,*)"in subroutine read_geometry, input file header is wrong"
stop
end if
!write(1,'(/1x,a)')"end of subroutine init_geo."
!write(1,'(/1x,a)')"---------------------------------------- "
!write(*,'(/1x,a)')"end of subroutine read_geo."
!write(*,'(/1x,a)')"---------------------------------------- "
end subroutine init_FH
```

```
!////////////////////////////read_cas_file(time)///////////////
```

```
subroutine read_cas_file(time)
use ComPara
use Node_data_D2Q9
implicit none
integer i,j,k
integer time
character*40 flog
write(*,'(/1x,a)') 'read feq from lbm2d.cas'
fLog=".\input\lbm2d.cas"
open(10,file=fLog,STATUS='OLD')
! write the time step
read(10,'(I10)')time
do j=1,ly,1
do i=1,lx,1
read(10,'(9e20.11)')(feq(i,j,k),k=0,8)
end do
end do
!write(9,*)feq(20,30,:)
!close(10)
fbar(:,:,:)=feq(:,:,:)
end subroutine read_cas_file
```

```
!////////////////////////////collision/////////////////////////
```

```
!collision
subroutine collision
use ComPara, only: lx, ly, omega, fun_is_fluid
use Node_data_D2Q9, only: feq, fbar

implicit none
integer i,j,k
real*8 max_tau,t1(0:8),t2(0:8),t3(0:8)
!calculate equillibium function feq
! call cal_feq
do j=1,ly,1
do i=1,lx,1
if(fun_is_fluid(i,j)) then
!do k=0,8,1
! Reference
! JFM Y Li, etc
! 2004, vol. 519, pp. 273
t1(:)=feq(i,j,:)
```

```
t2(:)=fbar(i,j,:)
t3=1.d0-t1/t2
max_tau=MAXVAL(t3)
max_tau=DMAX1(max_tau, omega)
feq(i,j,:)=(1.0d0-max_tau)*t2+max_tau*t1
!end do
end if
end do
end do
!write(1,*) 'collision finish'
!write(*,*) 'collision finish'
end subroutine collision
```

!//////////////////////propagation//////////////////////

```
!propagation
subroutine propagation
!integer time
use ComPara, only: lx, ly
use Node_data_D2Q9, only: feq, fbar
implicit none
integer i, j, k, x_e, x_w, y_n, y_s
do j=1,ly,1
do i=1,lx,1
!.........compute upper and right next neighbour nodes with regard
!
to periodic boundaries
y_n = mod(j,ly) + 1
x_e = mod(i,lx) + 1
!.........compute lower and left next neighbour nodes with regard to
!
periodic boundaries

y_s = ly - mod(ly + 1 - j, ly)
x_w = lx - mod(lx + 1 - i, lx)
!.........zero
fbar(i
,j ,0) = feq(i,j,0)
!.........east
fbar(x_e,j ,1) = feq(i,j,1)
```

```
!.........north
fbar(i
,y_n,2) = feq(i,j,2)
! .........west
fbar(x_w,j
,3) = feq(i,j,3)
! .........south
fbar(i
,y_s,4) = feq(i,j,4)
! .........north-east
fbar(x_e,y_n,5) = feq(i,j,5)
! .........north-west
fbar(x_w,y_n,6) = feq(i,j,6)
! .........south-west
fbar(x_w,y_s,7) = feq(i,j,7)
! .........south-east
fbar(x_e,y_s,8) = feq(i,j,8)
end do
end do
!write(1,*) 'propagation'
end subroutine propagation
```

```
!//////////////////////////bounceback//////////////////////
```

```
!implement bounceback scheme in the solid particles
subroutine bounceback
use ComPara, only: lx, ly, fun_is_fluid
use Node_data_D2Q9, only: feq, fbar, pNode
implicit none
integer i,j, x_e, x_w, y_n, y_s
real*8 :: temp(0:8)
do j=1,ly,1
do i=1,lx,1
!debug

!solid or solid interface
if(fun_is_fluid(i,j)==.False.) then
y_n = mod(j,ly) + 1
x_e = mod(i,lx) + 1
y_s = ly - mod(ly + 1 - j, ly)
x_w = lx - mod(lx + 1 - i, lx)
```

```
!.........east
fbar(x_e,j ,1) = fbar(i,j,3)
!.........north
fbar(i
,y_n,2) = fbar(i,j,4)
! .........west
fbar(x_w,j,3) = fbar(i,j,1)
! .........south
fbar(i,y_s,4) = fbar(i,j,2)
! .........north-east
fbar(x_e,y_n,5) = fbar(i,j,7)
! .........north-west
fbar(x_w,y_n,6) = fbar(i,j,8)
! .........south-west
fbar(x_w,y_s,7) = fbar(i,j,5)
! .........south-east
fbar(x_e,y_s,8) = fbar(i,j,6)
end if
end do
end do
end subroutine bounceback
```

```
!//////////////////////////BC_treatment//////////////////////
```

```
!implement bounceback scheme in the solid particles
subroutine BC_treatment
use ComPara, only: Flag_BC_FH, Flag_BC_force, Flag_BC_Pressure, FLAG_BC_Velocity
!use Node_data_D2Q9, only: feq, fbar, pNode
!use BoundaryCondition
implicit none
! add force
if( Flag_BC_force) then
call add_force
end if
! implement the velocity & pressure boundary
if( Flag_BC_Pressure .OR. Flag_BC_Velocity) then
call BC_treatment_vp
```

```
end if
! implement the FH boundary
if( Flag_BC_FH) then
call BC_treatment_FH
end if
end subroutine BC_treatment
```

!//////////////////////////add_force//////////////////////

```
!adding force 0.01 pascal/lu
subroutine add_force
use ComPara, only: lx, ly, force, fun_is_fluid, gamma
use Node_data_D2Q9, only: fbar, feq
implicit none
real*8 :: t1,t2
integer i,j
t1=force/3.d0 /gamma
t2=force/12.d0 /gamma
do j=1,ly,1
do i=1,lx,1
if(fun_is_fluid(i,j)) then !
if(fbar(i,j,3)>t1 .and. fbar(i,j,6) >t2 .and. fbar(i,j,7)>t2) then
fbar(i,j,1)=fbar(i,j,1)+t1
fbar(i,j,3)=fbar(i,j,3)-t1
fbar(i,j,5)=fbar(i,j,5)+t2
fbar(i,j,6)=fbar(i,j,6)-t2
fbar(i,j,7)=fbar(i,j,7)-t2
fbar(i,j,8)=fbar(i,j,8)+t2
end if
end if
end do
end do
!write(1,*) 'add force 0.01 pascal/lu'
end subroutine add_force
```

!//////////////////////////BC_treatment_vp//////////////////////

```
    !treat the velocity and pressure

    subroutine BC_treatment_vp
    use ComPara, only: Flag_BC_Pressure, FLAG_BC_Velocity
    use Node_data_D2Q9, only: fbar
    use BoundaryCondition
    implicit none
    integer :: time
    integer :: i,j,k,p_i,v_i
    !u_n -- normal velocity
    real*8 :: u_n, rou_in, ftemp(0:8),u_t

    !pressure Boundary treatment


if(FLAG_BC_Pressure) then
    p_i=size(pBC_pressure)
    do k=1,p_i
    i=pBC_pressure(k)%i
    j=pBC_pressure(k)%j
    u_t=pBC_pressure(k)%u_1
    rou_in=pBC_pressure(k)%rou


    !Zou and He pressure boundary on East side


if(pBC_pressure(k)%id==1) then
    ftemp(:)=fbar(i,j,:)
    u_n=1.d0-(2.d0*(ftemp(1)+ftemp(5)+ftemp(8))+(ftemp(0)+ftemp(2)+ftemp(4)))/rou_in
    fbar(i,j,3)=ftemp(1)+2.d0/3.d0*rou_in*u_n
    fbar(i,j,6)=ftemp(8)-.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
    fbar(i,j,7)=ftemp(5)+.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
    end if


    !Zou and He pressure boundary on West side


    if(pBC_pressure(k)%id==3) then
    ftemp(:)=fbar(i,j,:)
    u_n=1.d0-(2.d0*(ftemp(3)+ftemp(6)+ftemp(7))+(ftemp(0)+ftemp(2)+ftemp(4)))/rou_in
    fbar(i,j,1)=ftemp(3)+2.d0/3.d0*rou_in*u_n
    fbar(i,j,5)=ftemp(7)-.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
```

```
fbar(i,j,8)=ftemp(6)+.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
end if


!Zou and He pressure boundary on North side


if(pBC_pressure(k)%id==2) then
ftemp(:)=fbar(i,j,:)
u_n=1.d0-(2.d0*(ftemp(2)+ftemp(5)+ftemp(6))+(ftemp(0)+ftemp(1)+ftemp(3)))/rou_in
fbar(i,j,4)=ftemp(2)+2.d0/3.d0*rou_in*u_n
fbar(i,j,7)=ftemp(5)+.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
fbar(i,j,8)=ftemp(6)-.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
end if


!Zou and He pressure boundary on South side



if(pBC_pressure(k)%id==4) then
ftemp(:)=fbar(i,j,:)
u_n=1.d0-(2.d0*(ftemp(4)+ftemp(7)+ftemp(8))+(ftemp(0)+ftemp(1)+ftemp(3)))/rou_in
fbar(i,j,2)=ftemp(4)+2.d0/3.d0*rou_in*u_n
fbar(i,j,5)=ftemp(7)-.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
fbar(i,j,6)=ftemp(8)+.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
end if
end do
end if

!velocity Boundary treatment

if(FLAG_BC_Velocity) then
v_i=size(pBC_velocity)
do k=1,v_i
i=pBC_velocity(k)%i
j=pBC_velocity(k)%j


!Zou and He velocity boundary on East side



if(pBC_velocity(k)%id==1) then
u_n=-pBC_velocity(k)%u_x
u_t=pBC_velocity(k)%u_y
ftemp(:)=fbar(i,j,:)
rou_in=(2.d0*(ftemp(1)+ftemp(5)+ftemp(8))+(ftemp(0)+ftemp(2)+ftemp(4)))/(1.d0-u_n)
```

```
fbar(i,j,3)=ftemp(1)+2.d0/3.d0*rou_in*u_n
fbar(i,j,6)=ftemp(8)-.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
fbar(i,j,7)=ftemp(5)+.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
end if


!Zou and He velocity boundary on West side


if(pBC_velocity(k)%id==3) then
u_n=pBC_velocity(k)%u_x
u_t=pBC_velocity(k)%u_y
ftemp(:)=fbar(i,j,:)
rou_in=(2.d0*(ftemp(3)+ftemp(6)+ftemp(7))+(ftemp(0)+ftemp(2)+ftemp(4)))/(1.d0-u_n)
fbar(i,j,1)=ftemp(3)+2.d0/3.d0*rou_in*u_n
fbar(i,j,5)=ftemp(7)-.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
fbar(i,j,8)=ftemp(6)+.5d0*(ftemp(2)-ftemp(4))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
end if

!Zou and He velocity boundary on North side


if(pBC_velocity(k)%id==2) then
u_n=-pBC_velocity(k)%u_y
u_t=pBC_velocity(k)%u_x
ftemp(:)=fbar(i,j,:)
rou_in=(2.d0*(ftemp(2)+ftemp(5)+ftemp(6))+(ftemp(0)+ftemp(1)+ftemp(3)))/(1.d0-u_n)
fbar(i,j,4)=ftemp(2)+2.d0/3.d0*rou_in*u_n
fbar(i,j,7)=ftemp(5)+.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
fbar(i,j,8)=ftemp(6)-.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
end if


!Zou and He velocity boundary on South side


if(pBC_velocity(k)%id==4) then
u_n=pBC_velocity(k)%u_y
u_t=pBC_velocity(k)%u_x
ftemp(:)=fbar(i,j,:)
rou_in=(2.d0*(ftemp(4)+ftemp(7)+ftemp(8))+(ftemp(0)+ftemp(1)+ftemp(3)))/(1.d0-u_n)
fbar(i,j,2)=ftemp(4)+2.d0/3.d0*rou_in*u_n
fbar(i,j,5)=ftemp(7)-.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0+0.5d0*rou_in*u_t
fbar(i,j,6)=ftemp(8)+.5d0*(ftemp(1)-ftemp(3))+rou_in*u_n/6.d0-0.5d0*rou_in*u_t
end if
end do
end if
end subroutine BC_treatment_vp
```

```
!//////////////////////////BC_Treatment_FH//////////////////////


        !BCTreatment(time)
        subroutine BC_Treatment_FH
        use ComPara, only: lx, ly
        use Node_data_D2Q9, only: pNode
        use BoundaryCondition
        implicit none
        integer :: y_n, x_e, y_s, x_w, y_nn, x_ee, y_ss, x_ww, i, j, k
        !real*8 :: tmp, xi, u_sf, f_star, c_squ, u_f, u_squ
        integer :: f1_i, minus_i, r_ff_x, r_ff_y, r_s_x, r_s_y
        !write(1,*) 'BCTreatment'
        !BC treatment by F_H method
        !!debug
        !write(*,*) '(10,1)'
        !write(*,*) pNode(10,1)%node(0:8)
        !write(*,*) '(10,2)'
        !write(*,*) pNode(10,2)%node(0:8)
        do k=1, size(pBC_FH)-1
        i=pBC_FH(k)%i
        j=pBC_FH(k)%j
        if(pNode(i,j)%NodeType==2) then
        !debug
        !write(*,*) i,j
        !fluid interface
        !.........compute upper and right next neighbour nodes with regard
        !
        to periodic boundaries

        y_n = mod(j,ly) + 1
        y_nn= mod(y_n,ly)+1
        x_e = mod(i,lx) + 1
        x_ee= mod(x_e,lx)+1
        !.........compute lower and left next neighbour nodes with regard to
        !
        periodic boundaries
        y_s = ly - mod(ly + 1 - j, ly)
        y_ss= ly - mod(ly+1-y_s,ly)
        x_w = lx - mod(lx + 1 - i, lx)
        x_ww= lx - mod(lx+1-x_w,lx)
        !BC treatment
        !###########################################
        !# 1 rate(1)
        !###########################################
        if(pBC_FH(k)%rate(1)>0.d0 ) then
```

```
!###########################################
!deal with rate(1)
f1_i=1
!x_ff
r_ff_x=x_w
r_ff_y=j
r_s_x=x_e
r_s_y=j
minus_i=3
call cal_BC_FH(i,j, 1, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
end if
if(pBC_FH(k)%rate(3)>0.d0) then
f1_i=3
r_ff_x=x_e
r_ff_y=j
r_s_x=x_w
r_s_y=j
minus_i=1
call cal_BC_FH(i,j, 3, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
end if
!###########################################
!# 2 rate(2)

!###########################################
if(pBC_FH(k)%rate(2)>0.d0 ) then
!###########################################
!deal with rate(1)
f1_i=2
!x_ff
r_ff_x=i
r_ff_y=y_s
r_s_x=i
r_s_y=y_n
minus_i=4
call cal_BC_FH(i,j, 2, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
end if
if(pBC_FH(k)%rate(4)>0.d0) then
f1_i=4
r_ff_x=i
r_ff_y=y_n
r_s_x=i
r_s_y=y_s
minus_i=2
call cal_BC_FH(i,j, 4, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
end if
!###########################################
!# 3 rate(3)
!###########################################
if(pBC_FH(k)%rate(5)>0.d0 ) then
```

```
      !##########################################
      !deal with rate(1)
      f1_i=5
      !x_ff
      r_ff_x=x_w
      r_ff_y=y_s
      r_s_x=x_e
      r_s_y=y_n
      minus_i=7
      call cal_BC_FH(i,j, 5, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
      end if
      if(pBC_FH(k)%rate(7)>0.d0) then
      f1_i=7

      r_ff_x=x_e
      r_ff_y=y_n
      r_s_x=x_w
      r_s_y=y_s
      minus_i=5
      call cal_BC_FH(i,j, 7, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
      end if
      !##########################################
      !# 4 rate(4)
      !##########################################
      if(pBC_FH(k)%rate(6)>0.d0 ) then
      !##########################################
      !deal with rate(1)
      f1_i=6
      !x_ff
      r_ff_x=x_e
      r_ff_y=y_s
      r_s_x=x_w
      r_s_y=y_n
      minus_i=8
      call cal_BC_FH(i,j, 6, f1_i,r_ff_x, r_ff_y,r_s_x, r_s_y, minus_i)
      end if
      if(pBC_FH(k)%rate(8)>0.d0) then
      f1_i=8
      r_ff_x=x_w
      r_ff_y=y_n
      r_s_x=x_e
      r_s_y=y_s
      minus_i=6
      call cal_BC_FH(i,j, 8, f1_i,r_ff_x, r_ff_y,r_s_x,
     & r_s_y, minus_i)
      end if
      end if
      end do
      !!debugging
```

```
        !write(*,*) '(10,1)'
        !write(*,*) pNode(10,1)%node(0:8)
        !write(*,*) '(10,2)'
        !write(*,*) pNode(10,2)%node(0:8)

        !write(1,*) 'BCTreatment'
        !write(*,*) 'BCTreatment'
        end subroutine BC_Treatment_FH




!//////////////////////////cal_BC_FH//////////////////////////////




        !used for Loop BC treatment
        subroutine cal_BC_FH(i,j, rate, f1_i,r_ff_x, r_ff_y,r_s_x,
    & r_s_y, minus_i)
        use ComPara, only:
        CP_cs_sq, fun_cal_rate, omega
        use Node_data_D2Q9, only: fbar, macro, D2Q9_c, D2Q9_weight
        implicit none
        integer :: i,j, rate, f1_i, r_ff_x, r_ff_y, r_s_x, r_s_y, minus_i
        real*8 :: tmp, xi, u_sf, u_w, f_star, c_squ, u_f, u_squ
        u_w=0.d0
        c_squ = CP_cs_sq
        tmp = fun_cal_rate(i,j,rate)
        ! 0<q<0.5
        if(tmp<0.5d0 .AND. tmp>0.d0) then
        xi=omega*(2.d0*tmp-1.d0)/(1.d0-2.d0*omega)
        ! here is u_sf*c_i instead of u_sf
        ! i=1
        u_sf=(D2Q9_c(f1_i,1)*macro(r_ff_x,r_ff_y,2)
    &+D2Q9_c(f1_i,2)*macro(r_ff_x,r_ff_y,3))
        ! 0.5<=q<=1
        else if (tmp>=0.5d0 .AND. tmp<1.d0) then
        xi=2.d0*omega*(2.d0*tmp-1.d0)/(2.d0+omega)
        u_sf=(1.d0-1.5d0/tmp)*(D2Q9_c(f1_i,1)*macro(i,j,2) &
        +D2Q9_c(f1_i,2)*macro(i,j,3))+1.5d0/tmp*u_w
        else
        write(*,*) 'check pNode rate(k)',i,j,rate,tmp
        end if
        u_f=D2Q9_c(f1_i,1)*macro(i,j,2)+D2Q9_c(f1_i,2)*macro(i,j,3)
        u_squ=(macro(i,j,2) ** 2.d0+macro(i,j,3)**2.d0)
```

```
f_star=D2Q9_weight(f1_i)* macro(i,j,1) * (1.d0 + u_sf / c_squ &
+ u_f ** 2.d0 / (2.d0 * c_squ ** 2.d0) - u_squ / (2.d0 * c_squ))
fbar(i,j,minus_i)=(1.d0-xi)*fbar(i,j,f1_i)+xi*f_star
!
pNode(r_s_x,r_s_y)%node(minus_i)=(1.d0-xi)*pNode(i,j)%node(f1_i)+xi*f_star
end subroutine cal_BC_FH



!//////////////////////////cal_macro//////////////////


!calculate the macroscope variables and the equilibrium functions feq
subroutine cal_macro
use ComPara, only: lx, ly, fun_is_fluid, density, CP_cs_sq, gamma
use Node_data_D2Q9, only: fbar, feq, macro, D2Q9_weight

implicit none
integer i,j,k
real*8 :: u_n(8),u_x, u_y, u_squ, c_squ, d_loc, u_max
c_squ = CP_cs_sq
! calculate the gamma
!macro(:,:,1)=dsqrt(macro(:,:,2)*macro(:,:,2)+macro(:,:,3)*macro(:,:,3))
!u_max=MAXVAL(macro(:,:,1))
!gamma=(10.d0*dsqrt(3.d0)*u_max)**2
!write(*,*) "gamma=",gamma
do i=1,lx,1
do j=1,ly,1
!------------calculate the macroscopic variables
d_loc=sum(fbar(i,j,:))
if( d_loc>0.0d0 .and. fun_is_fluid(i,j)) then
macro(i,j,1)=d_loc
macro(i,j,2)= ((fbar(i,j,1)+fbar(i,j,5)+fbar(i,j,8)) &
-(fbar(i,j,3 ) + fbar(i,j,6 ) + fbar(i,j,7 ))) / d_loc
macro(i,j,3)=((fbar(i,j,2)+fbar(i,j,5)+fbar(i,j,6))&
-(fbar(i,j,4 ) + fbar(i,j,7 ) + fbar(i,j,8 )))/ d_loc
else
macro(i,j,1)=density
macro(i,j,2:3)=0.d0
end if
!------------upgrade the equilibrium functions feq
d_loc=macro(i,j,1)
u_x =macro(i,j,2)
u_y =macro(i,j,3)
u_n(1) =u_x
```

```
        u_n(2) =u_y
        u_n(3) = - u_x
        u_n(4) = - u_y
        u_n(5) = u_x + u_y
        u_n(6) = - u_x + u_y
        u_n(7) = - u_x - u_y
        u_n(8) =u_x - u_y
        u_squ = u_x * u_x + u_y * u_y
        feq(i,j,0) = D2Q9_weight(0) * d_loc *
    &(1.d0 - u_squ / (2.d0 * c_squ))
        do k=1,8
        feq(i,j,k)=D2Q9_weight(k) * d_loc * (1.d0 + u_n(k)
    & / c_squ &+ u_n(k) ** 2.d0 / (2.d0 * c_squ ** 2.d0)
    &- u_squ / (2.d0 * c_squ) )
        end do
        end do

        end do
        end subroutine cal_macro



!//////////////////////check//////////////////////



        !check(time)
        subroutine check(time,tic,toc,ti)
        use ComPara, only: lx, ly, CP_L2_err
        use Node_data_D2Q9, only: macro
        implicit none
        integer :: time,ti
        real*8 :: tic,toc
        real*8 :: d_loc,t
        character*13 :: char_time
        integer :: temp(4)
        call CPU_time(toc)
        d_loc=toc-tic
        t=d_loc/float(3600)!xxx.xxx hours left
        temp(1)=int(t)
        t=(t-float(temp(1)))*float(60)!xxx.xxx mins left
        temp(2)=int(t)
        t=(t-float(temp(2)))*float(60)
        temp(3)=int(t)
        t=(t-float(temp(3)))*float(100)
        temp(4)=int(t)
```

```
      write(char_time,'(I4.4,a,I2.2,a,I2.2,a,I2.2)')temp(1)
&,":",temp(2),":",temp(3),":",temp(4)
      d_loc=sum(macro(:,:,1))
      if(mod(ti,25)==1) then
      write(* ,'(/,a,4x,a,2x,a,2x,a)')"time(sec)", "iteration","
      !
      write(9
      ,'(/,a,4x,a,2x,a,2x,a)')"time(sec)",
      density","rsd-density rsd-x
      "iteration","
      rsd-y"
      end if
      write(*,'(a,i8,f12.4,3e12.3)') char_time, time, d_loc,
& CP_L2_err(1:3)
      write(9,'(a,i8,f12.4,3e12.3)') char_time, time, d_loc,
& CP_L2_err(1:3)
      end subroutine check

      density","rsd-density
      rsd-y"
      rsd-x



      !////////////////////////cal_L2_error////////////////////



      !calculate the L2 relative error
      subroutine cal_L2_error
      use ComPara, only: lx, ly, CP_L2_err, fun_is_fluid
      use Node_data_D2Q9, only: macro, macro_temp, pNode
      implicit none
      integer :: i,j
      real*8 :: err1(3), err2(3)
      !real*8 :: d_loc,t
      err1=0.d0
      err2=0.d0
      do i=1,lx
      do j=1,ly
      if(fun_is_fluid(i,j)) then
      err1(:)=err1(:)+(macro(i,j,:)-macro_temp(i,j,:))**2
      err2(:)=err2(:)+(macro(i,j,:))**2
      end if
      end do
      end do
      CP_L2_err(:)=dsqrt(err1(:)/err2(:))
      macro_temp=macro
```

```
        end subroutine cal_L2_error



!//////////////////////////////functions/////////////////////////////



!//////////////////my_mod(a,p)//////////////////


        ! function my_mod
        real*8 function my_mod(a,p)
        implicit none
        real*8 a,p !, my_mod
        my_mod=a-int(a/p)*p
        end function my_mod


!//////////////////fun_is_fluid(i,j)////////////


        ! function fun_is_fluid(i,j)
        ! determine whether point is fluid or not
        logical function fun_is_fluid(i,j)
        !use ComPara
        use Node_data_D2Q9, only: pNode
        implicit none
        integer :: i,j
        !logical fun_no_fracture
        if(pNode(i,j)%NodeType==0 .OR. pNode(i,j)%NodeType==2 .OR. pNode(i,j)%NodeType==6)
then


        fun_is_fluid=.True.
        else
        fun_is_fluid=.False.
        end if
        end function fun_is_fluid


!//////////////////fun_cal_rate(i,j,k)////////////


        ! calculate the rate(i,j,k)
        real*8 function fun_cal_rate(i,j,k)
```

```
!use ComPara
use Node_data_D2Q9, only: pNode
use BoundaryCondition, only: pBC_FH
implicit none
integer :: i,j,k
if(pNode(i,j)%NodeType==2 .OR. pNode(i,j)%NodeType==3) then
fun_cal_rate=pBC_FH(pNode(i,j)%rate_ij)%rate(k)
else
fun_cal_rate=0.d0
end if
end function fun_cal_rate


!/////////////////fun_4point_case(i,j)//////////////////////


! function fun_4point_case(i,j)
! determine whether point is fluid or not
integer function fun_4point_case(i,j)
use ComPara, only: fun_cal_rate, fun_point_number
use Node_data_D2Q9, only: pNode
implicit none
integer :: i,j
if( fun_cal_rate(i,j,5)<=0.5d0 &
.AND. fun_cal_rate(i+1,j,6)<=0.5d0 &
.AND. fun_cal_rate(i+1,j+1,7)<=0.5d0 &
.AND. fun_cal_rate(i,j+1,8)<=0.5d0 ) then
if( pNode(i,j)%NodeType==2) then
if(fun_cal_rate(i,j,5)>0.d0) then
fun_4point_case=1
else

fun_4point_case=2
end if
else if ( pNode(i,j)%NodeType==3) then
if(fun_cal_rate(i,j,5)>0.d0) then
fun_4point_case=3
else
fun_4point_case=4
end if
end if
else
open(1, file='.\results\fun_4point_case.dat',ACCESS ='append')
write(1,*) "fun_4point_case",i,j
close(1)
write(*,*) "in function fun_4point_case: rate(5:8)>0.5d0 has been found in", i,j
end if
end function fun_4point_case
```

```
!//////////////////fun_distance(x1,y1,x2,y2)//////////////////



        !function fun_distance(x1,y1,x2,y2)
        ! calculate the distance between (x1,y1) and (x2, y2)
        !
        real*8 function fun_distance(x1,y1,x2,y2)
        implicit none
        real*8 :: x1, x2, y1, y2
        fun_distance=DSQRT((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))
        end function fun_distance



 Module BoundaryCondition
                        implicit none
                        type BC_Velocity
        ! Zou & He (POF 1997)
        ! i,j -- position
        ! index: id
        !
        !1 -- east
        ! 2 -- north
        ! 3 -- west
        ! 4 -- south
        ! u_x,u_y: given velocity
                         integer :: i,j,id
                         real*8 :: u_x,u_y
                         end type BC_Velocity
                         Type BC_Pressure
        ! Zou & He (POF 1997)
        ! i,j -- position
        ! index: id
        ! 1 -- east
        ! 2 -- north
        ! 3 -- west
        ! 4 -- south
        ! u_1: given velocity along the boundary
        ! rou:
                           integer :: i,j,id
                            real*8 :: u_1=0.d0, rou
                            end type BC_Pressure
        !> Filippova and Hanel boundary treatment
```

```
                    Type BC_FH
!!NodeType
!!0 - fluid
!!1 - solid
!!2 - fluid, interface
!!3 - solid, interface
!integer :: NodeType=0
!!Boundary condition type
!!x,y-direction distance to the node
!real*8 ::rate(8)=0.0d0
                     integer :: i,j
                     integer :: id(0:4)=0
                     real*8 :: rate(8)=0.d0
                    end type BC_FH

                    type(BC_velocity), pointer:: pBC_velocity(:)
                    type(BC_pressure), pointer:: pBC_pressure(:)
                    type(BC_FH),
                    pointer:: pBC_FH(:)
                    end module BoundaryCondition



               Module CellCulture
                     implicit none
                     type cell
                     integer :: i,j
                     real*8 :: p, wss
                     end type cell
                     type(cell), pointer::pCell(:)
                     integer:: cell_no, cell_x_min, cell_x_max
                     integer:: cell_max_no, cell_max_generation, cell_generation
                     real*8 :: cell_max_wss, cell_dead_wss
                     real*8 , external :: fun_wss_pro
                     end module


          Module ComPara
!-----------LBM parameters-----------------
                integer :: lx, ly !<computation domain
                real*8 :: density, omega, force !< density, relexation parameter, body force
                 integer :: CP_max_t, CP_frame, CP_time_step
                 real*8 :: CP_tol
                 real*8 :: CP_L2_err(3)
! the precondition parameters
! cf. I:\LBM\Reference\Optimization\PhysRevE.70.066706-1.pdf for more details

               real*8 :: gamma
!-----------mathematic constant------------
```

```
                real*8, parameter:: PI= 3.14159265358979323846d0 !< accurate enough
                real*8, parameter:: CP_Cs= 0.5773502691896257645 1d0
                real*8, parameter:: CP_Cs_sq = 0.33333333333333333333d0
                real*8, parameter:: CP_bct= 0.5d0 !< for the upper and bottom boundary, esp. for
Poiseuille flow
        !----------functions------------------------
                 real*8 , external :: my_mod
                 logical, external :: fun_no_fracture
                 logical, external :: fun_is_fluid
                 integer, external :: fun_point_number
                 integer, external :: fun_4point_case
                 real*8 , external :: fun_distance
                 real*8 , external :: fun_theta
                 integer, external :: FE_node_xy
                 real*8 , external :: fun_cal_rate
        !----------BC flag--------------------
                logical :: FLAG_Debug_Poisseuille=.True.
                logical :: FLAG_BC_Pressure=.True.
                logical :: FLAG_BC_Velocity=.True.
                logical :: FLAG_BC_FH=.True.
                logical :: FLAG_BC_force=.True.
                real*8 :: BC_Pressure_inlet
                real*8 :: BC_Pressure_outlet
                end module ComPara



        Module Node_Data_D2Q9
                 implicit none
                 Type Node
        !>NodeType
        !!0 - fluid
        !!1 - solid
        !!2 - fluid, interface
        !!3 - solid, interface
                 integer :: NodeType=0
        !record of queue
                integer :: rate_ij =0
        ! macroscopic density
        !real*8 :: density=0.0d0
        ! x,y-direction velocity
        !real*8 :: velocity(2)=0.0d0
                 end type Node
        !define the parameter in the node
                real*8 :: D2Q9_weight(0:8),D2Q9_c(0:8,2)
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: fbar
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: feq
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: macro
                real*8 ,ALLOCATABLE, dimension(:, :, :) :: macro_temp
```

```
        type(Node), pointer ,dimension(:,:) :: pNode
    !integer,ALLOCATABLE, dimension(:, :) :: NodeType
        end module Node_Data_D2Q9



    Module FE
        implicit none
    !-----------Output: finite element---------
    !real*8 ,ALLOCATABLE, dimension(:, :) :: FE_node !< x,y coordinate for each all point
    !integer,ALLOCATABLE, dimension(:, :) :: FE_seq
    !< connectivity
        real*8 ,ALLOCATABLE, dimension(:, :) :: FE_fluid_node !< x,y coordinate for
each fluid point
        integer,ALLOCATABLE, dimension(:, :) :: FE_fluid_seq
    !< fluid connectivity
    !integer,ALLOCATABLE, dimension(:, :, :) :: FE_node_rate
    !integer,ALLOCATABLE, dimension(:, :) :: node_xy
        real*8 ,ALLOCATABLE, dimension(:, :) :: FE_wss_node
        integer,ALLOCATABLE, dimension(:, :) :: FE_wss_seq
        real*8 ,ALLOCATABLE, dimension(:, :) :: FE_wss_area

    !< 4- sequence of fluid interface particles
    !< sequency of fluid particles

        real*8 ,ALLOCATABLE, dimension(:, :) :: FE_wss_sort
        integer,ALLOCATABLE, dimension(:, :) :: FE_fluid_seq_1
    !<----------flag ----------------------------
    !integer
    !----------count----------------------------
    !integer :: FE_seq_number=0
        integer :: FE_number(4)=0
    !< the number of fluid meshes
    !< the number of fluid particles
        integer :: FE_node_number(4)=0 !< the mount of 4-direction interface
        integer :: FE_seq_fluid_number=0 !< the mount of total fluid nodes
        integer :: FE_total_node_number=0
        integer :: FE_wss_node_number=0
    !integer :: FE_wss_seq_number =0
        integer :: FE_wss_sort_i=10
    !functions
        end Module FE
```