```
! this is the main file for backward facing step computation code
program main

    use params
    use model_vars

    implicit none

    ! initializing the computational variables
    call initializer()
    print *,"Initialization Done..."

    ! resume computation
    call resume()

    ! entering main loop
    main_loop: do itr = istart,Nstep

        ! momentum equations coefficients computation
        call ME_coeff_compute()

        ! momentum equations Solution
        call solve_ME()

        ! pressure correction equation coefficients computation
        call PCE_coeff_compute()

        ! pressure correction equation solution
        call solve_PCE()

        ! velocity and pressure correction and divergence check
        call correct_VP_DC()

        print *,"TimeStep : ",itr

        ! writing the current timestep value to a file and checking convergence
        convergence = maxval(abs(Pressure_prev-p))
        open(unit=2, file="TimeStep.txt", status="replace")
        write(unit = 2, fmt = *) itr, convergence
        close(unit=2)

        ! writing max residual of p per timestep for post-computation plotting
        max_p_residual(itr) = convergence
        open(unit = 10, file='p_residual.txt', status = "replace")
        do i = 1,Nstep
            write(unit = 10, fmt = *) max_p_residual(i)
        end do
        close(unit = 10)

        ! export current time step staggered parameters for Resumability
        call export_staggered()

        if(convergence < 1e-9) then
            print *,"Solution Converged!!!"
            exit
        end if

        print *,"Data saved for TimeStep : ",itr

    end do main_loop

    ! primitive variables co-location computation
    call colocate()
    print *,"Colocation of variables done..."

    ! export result to a CSV FILE for PARAFOAM
    call export()
    print *,"CSV File generated..."

    print *,"Solution Done"

    print *,"Program Terminated ..."

end program main

! ------------------------------------------------------------------------------

! this file contains the initializer subroutine for backward facing step
```

```
subroutine initializer()

    use params
    use model_vars

    implicit none

    ! computing the mesh grid
    dx = length/float(nx-1); dy = width/float(ny-1)
    X = 0.0; Y = 0.0
    do i = 1,nx
    |   do j = 1,ny
    |   |   X(i,j) = float(i-1)*dx
    |   |   Y(i,j) = -width/2+float(j-1)*dy
    |   end do
    end do

    ! computing Y co-ordinates for U staggered grid
    do i = 1,nuy
    |   Yu(i) = -dy/2 + float(i-1)*dy
    end do

    ! computing molecular dynamic viscosity
    mu = rho*Vavg*width/Re

    ! computing diffusion and source-related terms
    De = mu/dx*dy; Dw = De; Dn = mu/dy*dx; Ds = Dn
    ap0 = rho*dx*dy/dt

    ! initializing primitive variables
    u = 0.0; v = 0.0; p = 0.0; pp = 0.0
    u(1,Nwend+1:nuy) = 24*Y(1,Nwend:ny)*(width/2-Y(1,Nwend:ny))
    u(1,1:Nwend+1) = 0 ! adjustments regarding parabolic velocity inlet
    us = u; vs = v; p = 0; Pressure_prev = p
    max_p_residual = 0

    ! starting point of TimeStep
    istart = 1

end subroutine initializer

! ------------------------------------------------------------------------------

! this module contains parameters required for computation
module params

    implicit none

    ! declaring standard datatypes for variables
    integer,parameter :: ikd = selected_int_kind(8)
    integer,parameter :: rkd = selected_real_kind(8,8)

    ! density and Reynolds Number of fluid, based on width
    real(kind=rkd),parameter :: rho = 1.0, Re = 800
    ! length and width of fluid domain
    real(kind=rkd),parameter :: length = 30.0, width = 1.0
    ! average inlet velocity and computational time step size
    real(kind=rkd),parameter :: Vavg = 1.0, dt = 1e-3
    ! number of nodes and timesteps
    integer(kind=ikd),parameter :: nx = 1201, ny = 41, Nstep = 200000
    ! wall step start and end nodes
    integer(kind=ikd),parameter :: Nwstart = 1, Nwend = int(ny/2)

    ! nodal arrangements for staggered grid
    integer(kind=ikd),parameter :: npx = nx+1, npy = ny+1
    integer(kind=ikd),parameter :: nux = npx-1, nuy = npy
    integer(kind=ikd),parameter :: nvx = npx, nvy = npy-1

end module params

! this module contains all the variables used in this program
module model_vars

    use params

    implicit none

    ! 2 dimensional variables
```

```
    real(kind=rkd),dimension(nux,nuy) :: u,us,apu,aeu,awu,anu,asu,bu,du,uprev
    real(kind=rkd),dimension(nvx,nvy) :: v,vs,apv,aev,awv,anv,asv,dv,bv,vprev
    real(kind=rkd),dimension(npx,npy) :: p,pp,app,aep,awp,anp,asp,pprev,Bp
    real(kind=rkd),dimension(nx,ny) :: X,Y,Uc,Vc,Pc

    ! 2 dimensional pressure variable to check for convergence
    real(kind=rkd),dimension(npx,npy) :: Pressure_prev

    ! pressure residual export variable declaration
    real(kind=rkd),dimension(Nstep) :: max_p_residual

    ! 1d variable for implementing BC
    real(kind=rkd),dimension(nuy) :: Yu

    ! scalar variables
    real(kind=rkd) :: Fe,Fw,Fn,Fs,De,Dw,Dn,Ds,con_p,con_u,convergence
    real(kind=rkd) :: Pe,Pw,Pn,Ps,con_v,dx,dy,mu,ap0,tmp
    integer(kind=ikd) :: itr,i,j,iterate_p,iterate_v,istart

end module model_vars

! ------------------------------------------------------------------------------

! this file contains following
! 1. ME_coeff_compute subroutine
! 2. solve_ME subroutine

subroutine ME_coeff_compute()

    use params
    use model_vars

    implicit none

    ! x momentum equation coefficients computation
    do i = 2,nux-1
        do j = 2,nuy-1
            Fe = rho*dy*0.5*(u(i,j)+u(i+1,j))
            Fw = rho*dy*0.5*(u(i,j)+u(i-1,j))
            Fn = rho*dx*0.5*(v(i,j)+v(i+1,j))
            Fs = rho*dx*0.5*(v(i,j-1)+v(i+1,j-1))

            Pe = Fe/De; Pw = Fw/Dw; Pn = Fn/Dn; Ps = Fs/Ds

            aeu(i,j) = De*max(0.0,(1.0-0.1*abs(Pe))**5) + max(0.0,-Fe)
            awu(i,j) = Dw*max(0.0,(1.0-0.1*abs(Pw))**5) + max(0.0, Fw)
            anu(i,j) = Dn*max(0.0,(1.0-0.1*abs(Pn))**5) + max(0.0,-Fn)
            asu(i,j) = Ds*max(0.0,(1.0-0.1*abs(Ps))**5) + max(0.0, Fs)
            apu(i,j) = aeu(i,j)+awu(i,j)+anu(i,j)+asu(i,j)+ap0
            bu(i,j) = ap0*u(i,j); du(i,j) = dy/apu(i,j)
        end do
    end do

    ! y momentum equation coefficients computation
    do i = 2,nvx-1
        do j = 2,nvy-1
            Fe = rho*dy*0.5*(u(i,j)+u(i,j+1))
            Fw = rho*dy*0.5*(u(i-1,j)+u(i-1,j+1))
            Fn = rho*dx*0.5*(v(i,j)+v(i,j+1))
            Fs = rho*dx*0.5*(v(i,j)+v(i,j-1))

            Pe = Fe/De; Pw = Fw/Dw; Pn = Fn/Dn; Ps = Fs/Ds

            aev(i,j) = De*max(0.0,(1.0-0.1*abs(Pe))**5) + max(0.0,-Fe)
            awv(i,j) = Dw*max(0.0,(1.0-0.1*abs(Pw))**5) + max(0.0, Fw)
            anv(i,j) = Dn*max(0.0,(1.0-0.1*abs(Pn))**5) + max(0.0,-Fn)
            asv(i,j) = Ds*max(0.0,(1.0-0.1*abs(Ps))**5) + max(0.0, Fs)
            apv(i,j) = aev(i,j)+awv(i,j)+anv(i,j)+asv(i,j)+ap0
            bv(i,j) = ap0*v(i,j); dv(i,j) = dx/apv(i,j)
        end do
    end do

end subroutine ME_coeff_compute

subroutine solve_ME()

    use params
    use model_vars
```

```
    implicit none

    ! momentum equations solution
    uprev = us; vprev = vs
    do  iterate_v = 1,1000
        do i = 2,nux-1
            do j = 2,nuy-1
                us(i,j) = 1/apu(i,j)*(aeu(i,j)*us(i+1,j)+awu(i,j)*us(i-1,j)+ &
                    anu(i,j)*us(i,j+1)+asu(i,j)*us(i,j-1)+bu(i,j)) + &
                    du(i,j)*(p(i,j)-p(i+1,j))
            end do
        end do
        do i = 2,nvx-1
            do j = 2,nvy-1
                vs(i,j) = 1/apv(i,j)*(aev(i,j)*vs(i+1,j)+awv(i,j)*vs(i-1,j)+ &
                    anv(i,j)*vs(i,j+1)+asv(i,j)*vs(i,j-1)+bv(i,j)) + &
                    dv(i,j)*(p(i,j)-p(i,j+1))
            end do
        end do
        us(nux,:) = us(nux-1,:)
        vs(nvx,:) = vs(nvx-1,:)

        con_u = maxval(abs(uprev-us)); uprev = us
        con_v = maxval(abs(vprev-vs)); vprev = vs

        if (con_u<1e-9 .and. con_v<1e-9) then
            exit
        end if
    end do

end subroutine solve_ME

! ------------------------------------------------------------------------------

! this file contains
! 1. PCE_coeff_compute subroutine
! 2. solve_PCE subroutine
! 3. correct_VP_DC subroutine

subroutine PCE_coeff_compute()

    use params
    use model_vars

    implicit none

    ! pressure correction equation coefficients computation
    do i = 2,npx-1
        do j = 2,npy-1
            aep(i,j) = rho*dy*du(i,j)
            awp(i,j) = rho*dy*du(i-1,j)
            anp(i,j) = rho*dx*dv(i,j)
            asp(i,j) = rho*dx*dv(i,j-1)
            app(i,j) = aep(i,j)+awp(i,j)+anp(i,j)+asp(i,j)
            Bp(i,j) = rho*(dy*(us(i-1,j)-us(i,j))+dx*(vs(i,j-1)-vs(i,j)))
        end do
    end do
    pp = 0.0; pprev = pp

end subroutine PCE_coeff_compute

subroutine solve_PCE()

    use params
    use model_vars

    implicit none

    ! pressure correction equation solution
    do iterate_p = 1,100
        do i = 2,npx-2
            do j = 2,npy-1
                pp(i,j) = 1.0/app(i,j)*(aep(i,j)*pp(i+1,j)+awp(i,j)*pp(i-1,j) &
                    +anp(i,j)*pp(i,j+1)+asp(i,j)*pp(i,j-1)+Bp(i,j))
            end do
        end do
        pp(1,:) = pp(2,:); pp(:,1) = pp(:,2); pp(:,npy) = pp(:,npy-1)
```

```
        con_p = maxval(abs(pprev-pp)); pprev = pp
        if (con_p < 1e-7) then
            exit
        end if
    end do

end subroutine solve_PCE

subroutine correct_VP_DC()

    use params
    use model_vars

    implicit none

    ! pressure correction
    p = p + 0.1*pp

    ! x velocity correction
    do i = 2,nux-1
        do j = 2,nuy-1
            u(i,j) = us(i,j) + du(i,j)*(pp(i,j)-pp(i+1,j))
        end do
    end do
    u(nux,:) = u(nux-1,:)

    ! y-velocity correction
    do i = 2,nvx-1
        do j = 2,nvy-1
            v(i,j) = vs(i,j) + dv(i,j)*(pp(i,j)-pp(i,j+1))
        end do
    end do
    v(nvx,:) = v(nvx-1,:)

    if(any(isnan(p))) then
        error stop "solution diverged"
    end if

end subroutine correct_VP_DC

! ------------------------------------------------------------------------------

! this file contains
! 1. collocate subroutine
! 2. export subroutine
! 3. export_staggered subroutine
! 4. resume subroutine
subroutine colocate

    use params
    use model_vars

    implicit none

    ! colocating x velocity
    Uc = 0.0; Uc(:,1) = u(:,1); Uc(:,ny) = u(:,nuy)
    do j = 2,ny-1
        Uc(:,j) = 0.5*(u(:,j)+u(:,j+1))
    end do

    ! colocating y velocity
    Vc = 0.0; Vc(1,:) = v(1,:); Vc(nx,:) = v(nvx,:)
    do i = 2,nx-1
        Vc(i,:) = 0.5*(v(i,:)+v(i+1,:))
    end do

    ! colocating pressure
    Pc = 0.0
    do i = 1,nx
        do j = 1,ny
            Pc(i,j) = 0.25*(p(i,j)+p(i+1,j)+p(i,j+1)+p(i+1,j+1))
        end do
    end do

end subroutine colocate

subroutine export()
```

```
    use params
    use model_vars

    open(unit=1, file="Data.csv", status="replace")

    15 format(f12.5,",",f12.5,",",f12.5,",",f12.5,",",f12.5,",",f12.5)

    write(unit=1,fmt=*) "X,Y,Z,U,V,P"

    do i = 1,nx
        do j = 1,ny
            write(unit=1,fmt=15) X(i,j),Y(i,j),0.0,Uc(i,j),Vc(i,j),Pc(i,j)
        end do
    end do

    close(unit=1)

end subroutine export

subroutine export_staggered()

    use params
    use model_vars

    implicit none

    open(unit = 3, file = "u.txt", status = "replace")
    do i = 1,nux
        do j = 1,nuy
            write(unit = 3, fmt = *) u(i,j)
        end do
    end do
    close(unit = 3)

    open(unit = 4, file = "v.txt", status = "replace")
    do i = 1,nvx
        do j = 1,nvy
            write(unit = 4, fmt = *) v(i,j)
        end do
    end do
    close(unit = 4)

    open(unit = 5, file = "p.txt", status = "replace")
    do i = 1,npx
        do j = 1,npy
            write(unit = 5, fmt = *) p(i,j)
        end do
    end do
    close(unit = 5)

end subroutine export_staggered

subroutine resume()

    use params
    use model_vars

    implicit none

    ! reading previous stopped timestep
    open(unit=11, file="TimeStep.txt")
    read(unit = 11, fmt = *) istart,tmp
    close(unit=11)

    ! reading previous pressure values
    open(unit = 13, file = "p.txt")
    do i = 1,npx
        do j = 1,npy
            read(unit = 13, fmt=*) p(i,j)
        end do
    end do
    close(unit = 13)

    ! reading previous x-velocity values
    open(unit = 21, file = "u.txt")
    do i = 1,nux
        do j = 1,nuy
```

```
        read(unit = 21, fmt = *) u(i,j)
    end do
end do
close(unit = 21)

! reading previous y-velocity values
open(unit = 22, file = "v.txt")
do i = 1,nvx
    do j = 1,nvy
        read(unit = 22, fmt = *) v(i,j)
    end do
end do
close(unit = 22)

! reading residual values of pressure
max_p_residual = 0.0
open(unit = 23, file = "p_residual.txt")
do i = 1,Nstep
    read(unit = 23, fmt = *) max_p_residual(i)
end do
close(unit = 23)

end subroutine resume

! --------------------------EOF-----------------------------------------------
```