```
module datas
    real(kind=8),parameter :: PI = 4.0*atan(1.0)
    real(kind=8),parameter :: SMV = 1.0E-20
    real(kind=8),parameter :: a= 0
    real(kind=8),allocatable,dimension(:,:) :: w !solution variable
    real(kind=8),allocatable,dimension(:,:) :: flux !flux
    real(kind=8),allocatable,dimension(:) :: x
    real(kind=8),allocatable,dimension(:,:) :: u0
    real(kind=8),allocatable,dimension(:,:) :: u1
    real(kind=8) :: dx !spacing in x-direction
    real(kind=8) :: dt !time step
    real(kind=8) :: cfl !cfl number
    real(kind=8) :: lambda
    real(kind=8) :: t
    integer :: iter !iterations
end module datas

module solver
    contains
    function Riemann(ul,ur)
    real(kind=8),dimension(2) ::ul
    real(kind=8),dimension(2) ::ur
    real(kind=8) ::a=1.0
    real(kind=8) ::rho0=1.0
    real(kind=8) ::alpha2
    real(kind=8) ::beta1
    real(kind=8) ::u1
    real(kind=8) ::u2
    real(kind=8) ,dimension(2) ::Riemann

    alpha2=(a*ul(1)+rho0*ul(2))/(2*a*rho0)
    beta1=(a*ur(1)-rho0*ur(2))/(2*a*rho0)

    u1=beta1*rho0+alpha2*rho0
    u2=beta1*(-a)+alpha2*a
    Riemann(1)=rho0*u2
    Riemann(2)=a*a/(rho0)*u1
end function Riemann
end module solver

program main
    use datas
    use solver
    integer :: i,itmax,t_F,bc,flg
    real :: xmin,xmax
    integer :: m,n

    cfl = 0.7

    !geometry
    bc=2
    n =1000
    flg=1
    itmax=100
```

```
    m=1
    xmax=0.5
    xmin=-0.5
    t_F=1;bc=2;flg=1
    dx=(xmax-xmin)/(n-1)


    allocate(x(n))
    allocate(u0(n+2*m,2))
    allocate(u1(n+2*m,2))

    ! I.C.
open(unit=10,file="out.dat")
if (flg==1) then

     do i=1,n

        x(i)=-0.5+(i-1)*dx
      if (x(i)<-0.1) then
        u0(m+i,1)=0
        u0(m+i,2)=0
        elseif (x(i)<=0.1) then
            u0(m+i,1)=0
            u0(m+i,1)=0.10  !! wanring!
            u0(m+i,2)=0

        else
            u0(m+i,1)=0
            u0(m+i,2)=0
        endif
        !write(10,*) i, u0(m+i,1),u0(m+i,2)
        !write(*,*) "i,u0(m+i,1),u1(m+i,2)", i, u0(m+i,1),u0(m+i,2)
    end do
endif

! B.C.
if (bc==2) then
do i=1,m

    u0(i,1)=u0(2*m+1-i,1)
    u0(i,2)=u0(2*m+1-i,2)

    u0(m+n+i,1)= u0(m+n+1-i,1);
    u0(m+n+i,2)=-u0(m+n+1-i,2);

    enddo
endif

t=0;
dt=0.001
lambda=dt/dx    ! for constant dt,put it outside the loop

do while(t+dt<=t_F .or. it<=itmax)
```

```
      t=t+dt

    do i=1,n
        u1(m+i,:)=u0(m+i,:)-lambda*(Riemann(u0(m+i,:),u0(m+i+1,:))-Rieman
        n(u0(m+i-1,:),u0(m+i,:)))

        !write(10,*) i, u1(m+i,1),u1(m+i,2)

    enddo

if (bc==2) then
do i=1,m

    u1(i,1)=u1(2*m+1-i,1)
    u1(i,2)=u1(2*m+1-i,2)

    u1(m+n+i,1)= u1(m+n+1-i,1);
    u1(m+n+i,2)=-u1(m+n+1-i,2);

    enddo
endif

u0(:,:)=u1(:,:)

enddo

end program main


!subroutine timestep()
!    use datas
!    integer :: i
!    real(kind=8) :: umax
!
!    umax = 0.0
!    do i=1,num
!        umax=max(umax,w(i))
!    end do
!    dt = cfl*dx/umax
!end subroutine timestep
!
!subroutine calc_flux()
!    use datas
!    integer :: i
!
!    !boundary
!    flux(1) = 0.5*w(i)**2
!    flux(num+1) = 0.5*w(num)**2
!
!    !inner
!    do i=2,num
!        if (w(i-1)>=w(i)) then !form a shock
!            if (0<0.5*(w(i-1)+w(i))) then
!                flux(i) = 0.5*w(i-1)**2
```

```
!            else
!                flux(i) = 0.5*w(i)**2
!            end if
!        else !form a rarefaction wave
!            if (0<w(i-1)) then
!                flux(i) = 0.5*w(i-1)**2
!            else if (0>w(i)) then
!                flux(i) = 0.5*w(i)**2
!            else
!                flux(i) = 0.0
!            end if
!        end if
!    end do
!end subroutine calc_flux
!
!subroutine update()
!    use datas
!    integer :: i
!
!    do i=1,num
!        w(i) = w(i)+(flux(i)-flux(i+1))*dt/dx
!    end do
!end subroutine update
!
!subroutine writeout()
!    use datas
!    integer :: i
!
!    open(unit=10,file="out.dat")
!    do i=1,num
!        xpos = (-1.0+dx/2.0)+(4.0-dx)*(i-1.0)/(num-1.0+SMV)
!        write(10,*) xpos,w(i)
!    end do
!end subroutine writeout
```