

```

program lid_driven_cavity
    implicit none
    integer, parameter :: N = 50      ! grid size (NxN grid)
    real :: dx, dy, dt, Re          ! grid spacing, time step, Reynolds number
    real :: u(N, N), v(N, N), p(N, N) ! velocity and pressure fields
    integer :: i, j, step
1   real :: start_time, end_time, elapsed_time
2
3   ! Parameters
4   dx = 1.0 / (N-1)      ! Grid spacing in x direction
5   dy = 1.0 / (N-1)      ! Grid spacing in y direction
6   dt = 0.001             ! Time step size
7   Re = 100               ! Reynolds number
8
9   ! Initialize arrays
10  u = 0.0
11  v = 0.0
12  p = 0.0
13
14  ! Initialize the top boundary (lid) velocity
15  u(N, :) = 1.0
16
17  ! Start timing
18  call cpu_time(start_time)
19
20  ! Main loop for time stepping
21  do step = 1, 1000
22      call compute_velocity(u, v, p, dx, dy, dt, Re)
23      call update_pressure(p, dx, dy)
24
25      ! Output or check convergence
26      if (mod(step, 100) == 0) then
27          print *, 'Step: ', step
28      end if
29  end do
30
31  ! Stop timing
32  call cpu_time(end_time)
33  elapsed_time = end_time - start_time
34  print *, 'Elapsed time for CFD simulation: ', elapsed_time, ' seconds'
35
36 contains
37
38  ! Function to update the velocity and pressure fields (simplified)
39  subroutine compute_velocity(u, v, p, dx, dy, dt, Re)
40      real, dimension(:, :, intent(inout)) :: u, v, p
41      real, intent(in) :: dx, dy, dt, Re
42      integer :: i, j
43
44      ! Simple explicit method for velocity (simplified)
45      do i = 2, N-1
46          do j = 2, N-1
47              u(i, j) = u(i, j) - dt * ( (u(i, j) * (u(i+1, j) - u(i-1, j))) / (2*dx)
48                                         + (v(i, j) * (u(i, j+1) - u(i, j-1))) / (2*dy) )
49          end do
50      end do
51
52      ! Simple velocity update for v (similar)
53      do i = 2, N-1
54          do j = 2, N-1
55              v(i, j) = v(i, j) - dt * ( (u(i, j) * (v(i+1, j) - v(i-1, j))) / (2*dx)
56                                         + (v(i, j) * (v(i, j+1) - v(i, j-1))) / (2*dy) )
57          end do
58      end do
59  end subroutine compute_velocity
60
```

```
! Function to solve for pressure (simplified Poisson equation solver)
subroutine update_pressure(p, dx, dy)
    real, dimension(:, :, ), intent(inout) :: p
    real, intent(in) :: dx, dy
    integer :: i, j
```

```
1 ! Simple pressure Poisson equation (Jacobi iteration)
2 do i = 2, N-1
3     do j = 2, N-1
4         p(i, j) = 0.25 * ( p(i+1, j) + p(i-1, j) + p(i, j+1) + p(i, j-1) )
5     end do
6 end do
7 end subroutine update_pressure
8
9 end program lidDrivenCavity
```