

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

```
1      v(i, j) = v(i, j) - dt * ( (u(i, j) * (v(i+1, j) - v(i-1, j))) / (2*dx)
2                                (v(i, j) * (v(i, j+1) - v(i, j-1))) / (2*dy)
3      end do
4  end do
5  end subroutine compute_velocity
6
7  ! Function to solve for pressure (simplified Poisson equation solver)
8  subroutine update_pressure(p, dx, dy)
9      real, dimension(:, :), intent(inout) :: p
10     real, intent(in) :: dx, dy
11     integer :: i, j
12
13     ! Simple pressure Poisson equation (Jacobi iteration)
14     do i = 2, N-1
15         do j = 2, N-1
16             p(i, j) = 0.25 * ( p(i+1, j) + p(i-1, j) + p(i, j+1) + p(i, j-1) )
17         end do
18     end do
19 end subroutine update_pressure
20
21 end program lid_driven_cavity
```