

```

1 program lid_driven_cavity
2   implicit none
3   integer, parameter :: N = 50 ! grid size (N x N grid)
4   real :: dx, dy, dt, Re ! grid spacing, time step, Reynolds number
5   real :: u(1:N, 1:N), v(1:N, 1:N) ! velocity and pressure fields
6   integer :: 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(1, :) = 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 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) + v(i, j+1)) - (u(i, j-1) * v(i+1, j-1))) / (2 * dx) + (v(i, j) * (u(i+1, j) + v(i, j+1)) - (u(i, j-1) * v(i+1, j-1))) / (2 * dy))
54     end do
55   end do
56
57   ! Simple velocity update for v (similar)
58   do i = 2, N - 1
59     do j = 2, N - 1
60
1412THE

```

```

1   v(i, j) = v(i, j) - dt * ((u(i, j) * (v(i+1, j) - v(i-1, j)) / (2 * dx)) - (v(i, j+1) - v(i, j-1)) / (2 * dy))
2   end do
3   end do
4   end subroutine compute_velocity
5
6   ! Function to solve for pressure (simplified Poisson equation solver)
7   subroutine update_pressure(p, dx, dy)
8     real, dimension(:, :), intent(inout) :: p
9     real, intent(in), :: dx, dy
10    integer :: i, j
11
12   ! Simple pressure Poisson equation (Jacobi iteration)
13   do i = 2, N-1
14     do j = 2, N-1
15       p(i, j) = 0.25 * (p(i+1, j) + p(i-1, j) + p(i, j+1) + p(i, j-1))
16     end do
17   end do
18 end do
19 end subroutine update_pressure
20
21 end program lid_driven_cavity
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80

```