```fortran
!  ======================================================================
!               2-Dimensional Finite Element Analysis
!                  of Steady Ideal Flows Governed by Laplace Eq.
!
!               * Galerkin F.E.M.
!               * Gaussian Elimination used to solve the Linear Systems
!
!                                                  2006. 4.22. coded by S.Tanaka
!                                                  ( Fortran90 Version )
!  ======================================================================
!-- MAIN PROGRAM -
program Laplace
 implicit none
 !Mesh
  integer                         :: node, nelm        ! Total number of nodes and elments
  integer,           allocatable :: nc(:,:)            ! Node connection data in elment
  double precision, allocatable :: xx(:), yy(:)      ! X, Y in Cartecian coordinate planes
  double precision, allocatable :: area(:)           ! Area of Element
  double precision, allocatable :: dx(:,:), dy(:,:)  ! \frac{\partial N}{\partial x}
 !Boundary Condition
  integer                         :: ipbc              ! Total number of nodes on \Gamma_{1}
  integer,           allocatable :: npbc(:)           ! Number of node on \Gamma_{1}
  double precision, allocatable :: fpbc(:)           ! \hat{\phi} on \Gamma_{1}
 !Physical values
  double precision               :: kx, ky            ! Coefficient of permeability
  double precision, allocatable :: phi(:)            ! Potential values on node
  double precision, allocatable :: ea(:,:)           ! L.H.S. Matrix
  double precision, allocatable :: ue(:), ve(:)      ! Velocity ( on Element )

  integer :: i, j, n, m

  character(50) :: mesfile, dbcfile, outfile
!
! FILE I/O
  open(9, file = 'file.dat', status = 'unknown' )
   read(9,*) kx, ky
   read(9,'(a)')   mesfile
   read(9,'(a)')   dbcfile
   read(9,'(a)')   outfile
  close(9)
  open(10, file = mesfile, status = 'old')
  open(11, file = dbcfile, status = 'old')
  open(60, file = outfile, status = 'replace')

! READ MESH
      read(10,*) node, nelm
      allocate( xx(node), yy(node), nc(3,nelm) )
      do i = 1, node
        read(10,*) n, xx(n), yy(n)
      enddo
      do j = 1, nelm
        read(10,*)  m, ( nc(i,m), i = 1, 3 )
      enddo
      close(10)

! READ BOUNDARY CONDITION
      read(11,*) ipbc
      allocate( npbc(ipbc), fpbc(ipbc) )
      do i = 1, ipbc
        read(11,*) n, npbc(n), fpbc(n)
      enddo
      close(11)

! Dynamic Memory Allocation
         allocate( phi(node), ea(node,node), ue(nelm), ve(nelm), &
                   area(nelm), dx(3,nelm), dy(3,nelm) )
!=====================================================================
   call makeSF   &   ! Make Shape Function
       ( node,     nelm,     xx,       yy,       nc,       &
         area,     dx,       dy )
   call makLHS   &   ! Make L.H.S. Matrix
       ( node,     nelm,     nc,       dx,       dy,       area,     ea, &
         kx,       ky )
   call boundc   &   ! Impose Boundary Condition
```

```fortran
       ( node,     ea,       ipbc,     npbc,     fpbc,     phi )
   call sweep    &   ! Solver
       ( node,     ea,       phi )
   call calvel   &   ! Calculate Velocity
       ( node,     nelm,     nc,       dx,       dy,       phi,   &
         ue,       ve,       kx,       ky )
!===============================================================================
  call output    &   ! Result output
       ( node,     nelm,     phi,      ue,       ve )
end program Laplace
!-- MAIN PROGRAM --
!
!-- SUBROUTINES --
! -------------------------------------------------------------------------------
   subroutine makeSF   &   ! Make Shape Function
       ( node,     nelm,     xx,       yy,       nc,       &
         area,     dx,       dy )
! -------------------------------------------------------------------------------
      implicit none
      integer, intent(in) :: nelm, node, nc(3,nelm)
      double precision, intent(in) :: xx(node), yy(node)
      double precision, intent(out) :: area(nelm), dx(3,nelm), dy(3,nelm)
      integer :: m, n1, n2, n3
      double precision :: x1, x2, x3, y1, y2, y3, a02, a02i
!
      do m = 1, nelm
         n1 = nc(1,m)
         n2 = nc(2,m)
         n3 = nc(3,m)
         x1 = xx(n1)
         x2 = xx(n2)
         x3 = xx(n3)
         y1 = yy(n1)
         y2 = yy(n2)
         y3 = yy(n3)
         a02 = (x1 - x2) * (y1 - y3) - (x1 - x3) * (y1 - y2)
          if( a02 <= 0.0d0 ) then
             write(6,*) ' AREA FAILURE  ', m
          endif
         a02i = 1.d0 / a02
         area(m) = 0.5d0 * a02
         dx(1,m) = (y2 - y3) * a02i
         dx(2,m) = (y3 - y1) * a02i
         dx(3,m) = (y1 - y2) * a02i
         dy(1,m) = (x3 - x2) * a02i
         dy(2,m) = (x1 - x3) * a02i
         dy(3,m) = (x2 - x1) * a02i
      enddo
!
   end subroutine makeSF
!
! -------------------------------------------------------------------------------
   subroutine makLHS  & ! Make L.H.S. Matrix
       ( node,     nelm,     nc,       dx,       dy,       area,     ea, &
         kx,       ky )
! -------------------------------------------------------------------------------
      implicit none
      integer, intent(in) :: node, nelm, nc(3,nelm)
      double precision, intent(in) :: dx(3,nelm), dy(3,nelm), area(nelm), kx, ky
      double precision, intent(out) :: ea(node,node)
      double precision :: a01, dx1, dx2, dx3, dy1, dy2, dy3
      double precision :: ea11, ea12, ea13, ea22, ea23, ea33
      integer :: m, n1, n2, n3

      ea(:,:) = 0.0d0
      do m = 1, nelm
         n1 = nc(1,m)
         n2 = nc(2,m)
         n3 = nc(3,m)
         dx1 = dx(1,m)
         dx2 = dx(2,m)
         dx3 = dx(3,m)
         dy1 = dy(1,m)
         dy2 = dy(2,m)
```

```
         dy3 = dy(3,m)
         a01 = area(m)
          ea11 = (dx1 * dx1 * kx + dy1 * dy1 * ky) * a01
          ea12 = (dx1 * dx2 * kx + dy1 * dy2 * ky) * a01
          ea13 = (dx1 * dx3 * kx + dy1 * dy3 * ky) * a01
          ea22 = (dx2 * dx2 * kx + dy2 * dy2 * ky) * a01
          ea23 = (dx2 * dx3 * kx + dy2 * dy3 * ky) * a01
          ea33 = (dx3 * dx3 * kx + dy3 * dy3 * ky) * a01
          ea(n1,n1)  = ea(n1,n1) + ???????
          ea(n1,n2)  = ea(n1,n2) + ???????
          ea(n1,n3)  = ea(n1,n3) + ???????
          ea(n2,n1)  = ea(n2,n1) + ???????
          ea(n2,n2)  = ea(n2,n2) + ???????
          ea(n2,n3)  = ea(n2,n3) + ???????
          ea(n3,n1)  = ea(n3,n1) + ???????
          ea(n3,n2)  = ea(n3,n2) + ???????
          ea(n3,n3)  = ea(n3,n3) + ???????
      enddo
!
   end subroutine makLHS
!
! ----------------------------------------------------------------------------
   subroutine boundc   &   ! Impose Boundary Condition
       ( node,     ea,       ibc,      nbc,      fbc,      xx )
! ----------------------------------------------------------------------------
      implicit none
      integer, intent(in) :: node, ibc, nbc(ibc)
      double precision, intent(in) :: fbc(ibc)
      double precision, intent(out) :: xx(node)
      double precision, intent(inout) :: ea(node,node)
      integer :: n, ib, in
      double precision :: f, ea_nn
!
      xx = 0.0d0
       do ib = 1, ibc
          n = nbc(ib)
          f = fbc(ib)
          ea_nn = ea(n,n)
          do in = 1, node
             xx(in) = xx(in) - ea(in,n) * f
             ea(n, in) = 0.0d0
             ea(in, n) = 0.0d0
          enddo
          ea(n,n) = 1.0d0
          xx(n) = f
       enddo
!
   end subroutine boundc
!
! ---------------------------------------------------------------------------
   subroutine sweep (ndof, a, x)   ! Solve Linear-Systems
! ---------------------------------------------------------------------------
    implicit none
    integer, intent(in) :: ndof
    double precision, intent(inout) :: a(ndof,ndof), x(ndof)
!
    integer :: i, j, k, i1, l
    double precision :: ai, cc
!
     do i = 1, ndof
        if( dabs(a(i,i)) <= 1.0d0-10 ) then
           write(6,*) 'Diagonal Value',i,'is ZERO!'
           stop
        endif
!
        ai = 1.0d0 / a(i,i)
        x(i) = x(i) * ai
        do j = 1, ndof
           a(i,j) = a(i,j) * ai
        enddo
!
        if( i /= ndof ) then
            i1 = i + 1
             do k = i1, ndof
```

```fortran
               cc = a(k,i)
                do j = i1, ndof
                   a(k,j) = a(k,j) - cc * a(i,j)
                enddo
               x(k) = x(k) - cc * x(i)
             enddo
        endif
     enddo
!
     do i = 1, ndof-1
       j = ndof - i
        do k = 1, i
           l = ndof+1-k
           x(j) = x(j) - a(j,l) * x(l)
        enddo
     enddo
!
   end subroutine sweep
!
! --------------------------------------------------------------------------------
   subroutine calvel   &   ! Calculate Velocity
       ( node,     nelm,     nc,       dx,       dy,       phi,   &
         ue,       ve,       kx,       ky )
! --------------------------------------------------------------------------------
    implicit none
    integer, intent(in) :: node, nelm, nc(3,nelm)
    double precision, intent(in) :: dx(3,nelm), dy(3,nelm), phi(node),kx,ky
    double precision, intent(out) :: ue(nelm), ve(nelm)
    integer :: i, m
!
    ue(:) = 0.0d0
    ve(:) = 0.0d0
      do m = 1, nelm
         do i = 1, 3
            ue(m) = ue(m) - ?????????
            ve(m) = ve(m) - ?????????
         enddo
      enddo
   end subroutine calvel
!
! ---------------------------------------------------------------------------------
      subroutine output    &   ! Result output
       ( node,     nelm,     phi,      ue,       ve )
! ---------------------------------------------------------------------------------
      implicit none
      integer, intent(in) :: node, nelm
      double precision, intent(in) :: phi(node), ue(nelm), ve(nelm)
      integer :: n
!
      write(60,600) ( n, phi(n), n =1, node )
      write(60,601) ( n, ue(n), ve(n), n =1, nelm )
      close(60)
  600 format(i7, d15.6)
  601 format(i7,2d15.6)
!
      end subroutine output
```