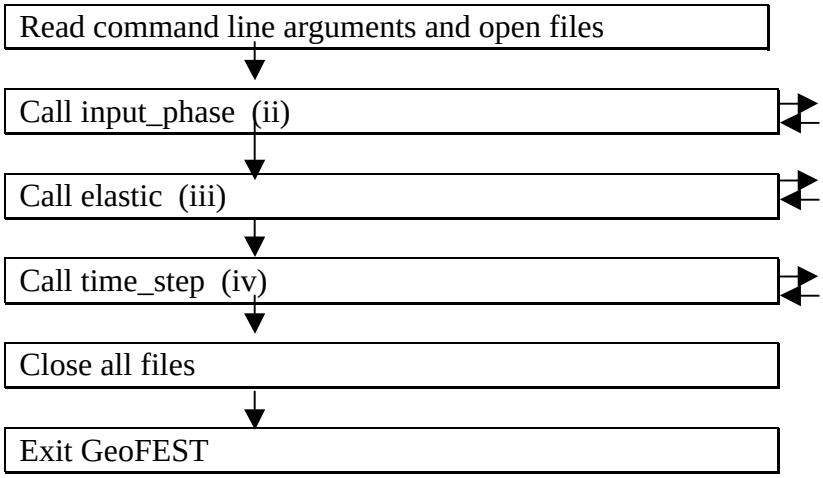
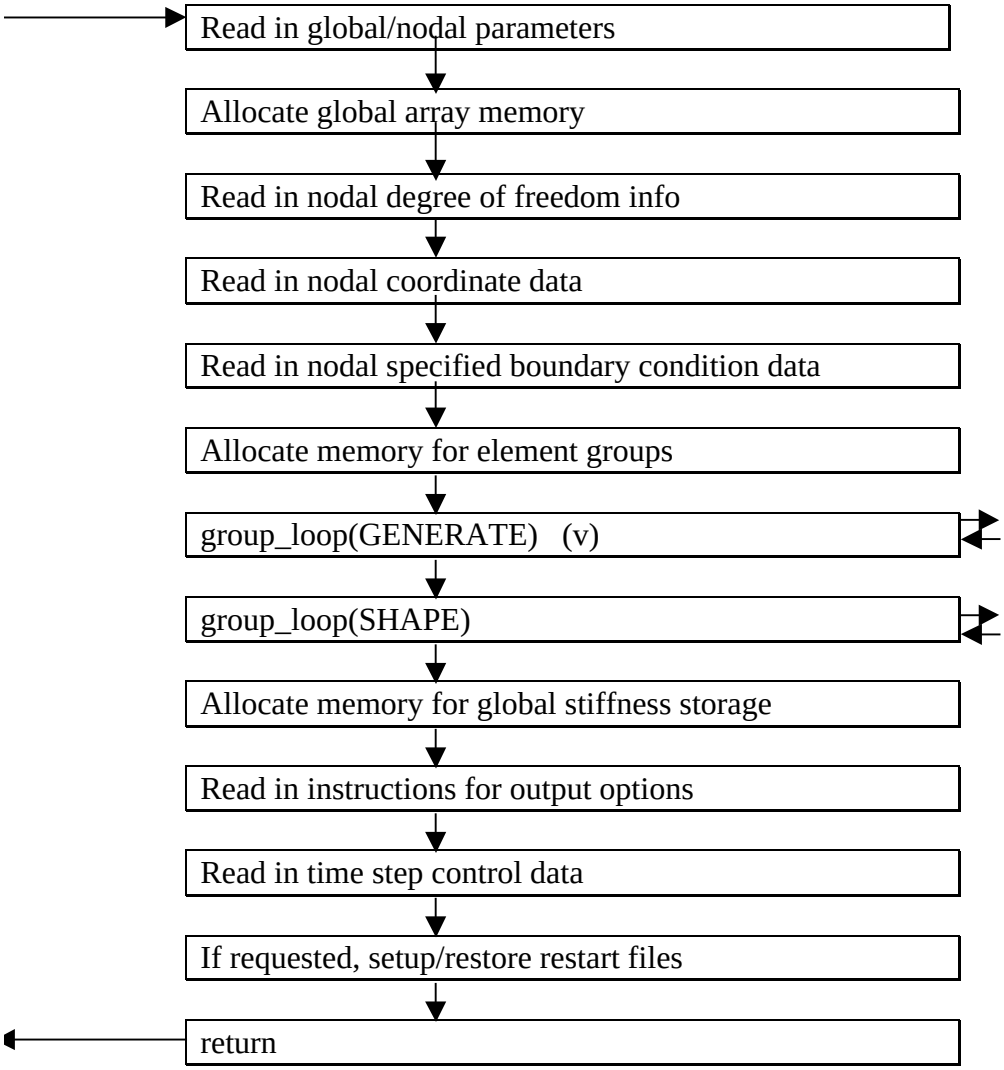
The following linked flow charts describe the basic execution flow and organization of GeoFEST, identifying the principal functional tasks and processes.

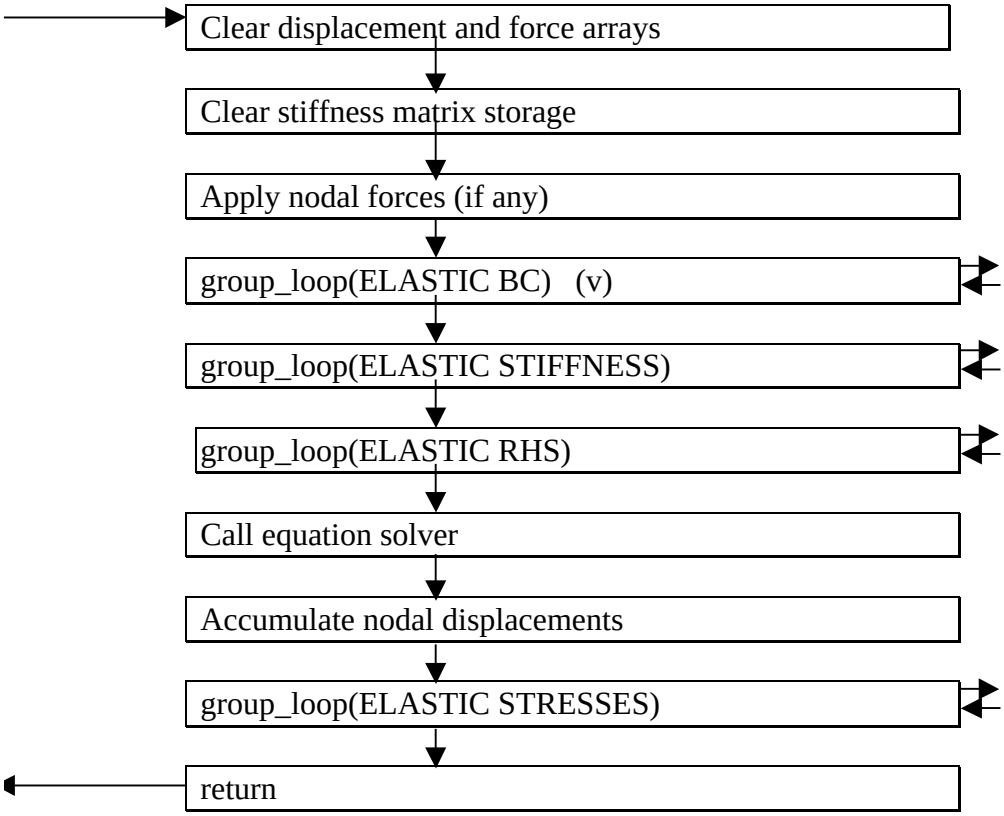
(i) GeoFEST main() entry point



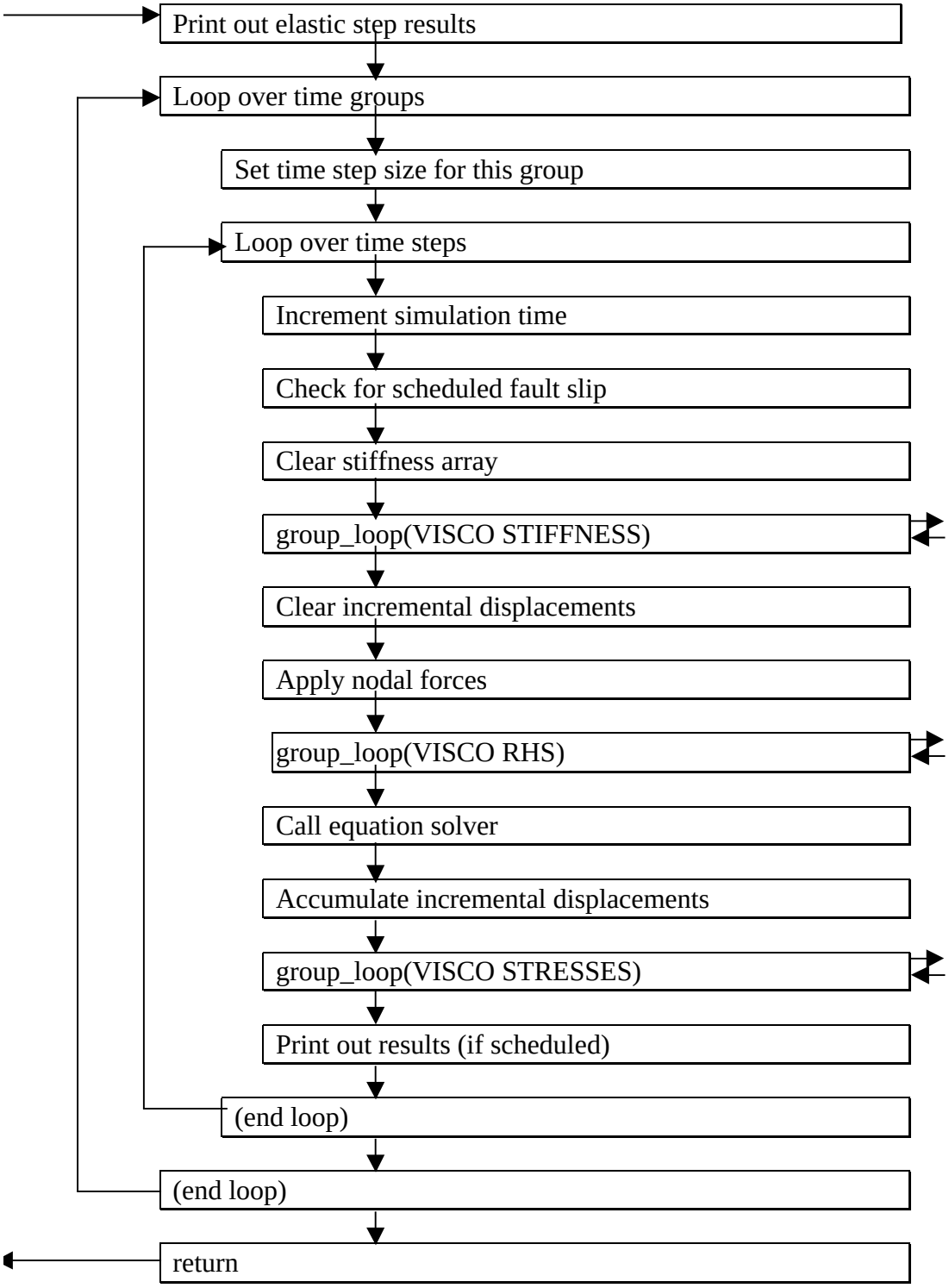
(ii) input_phase routine



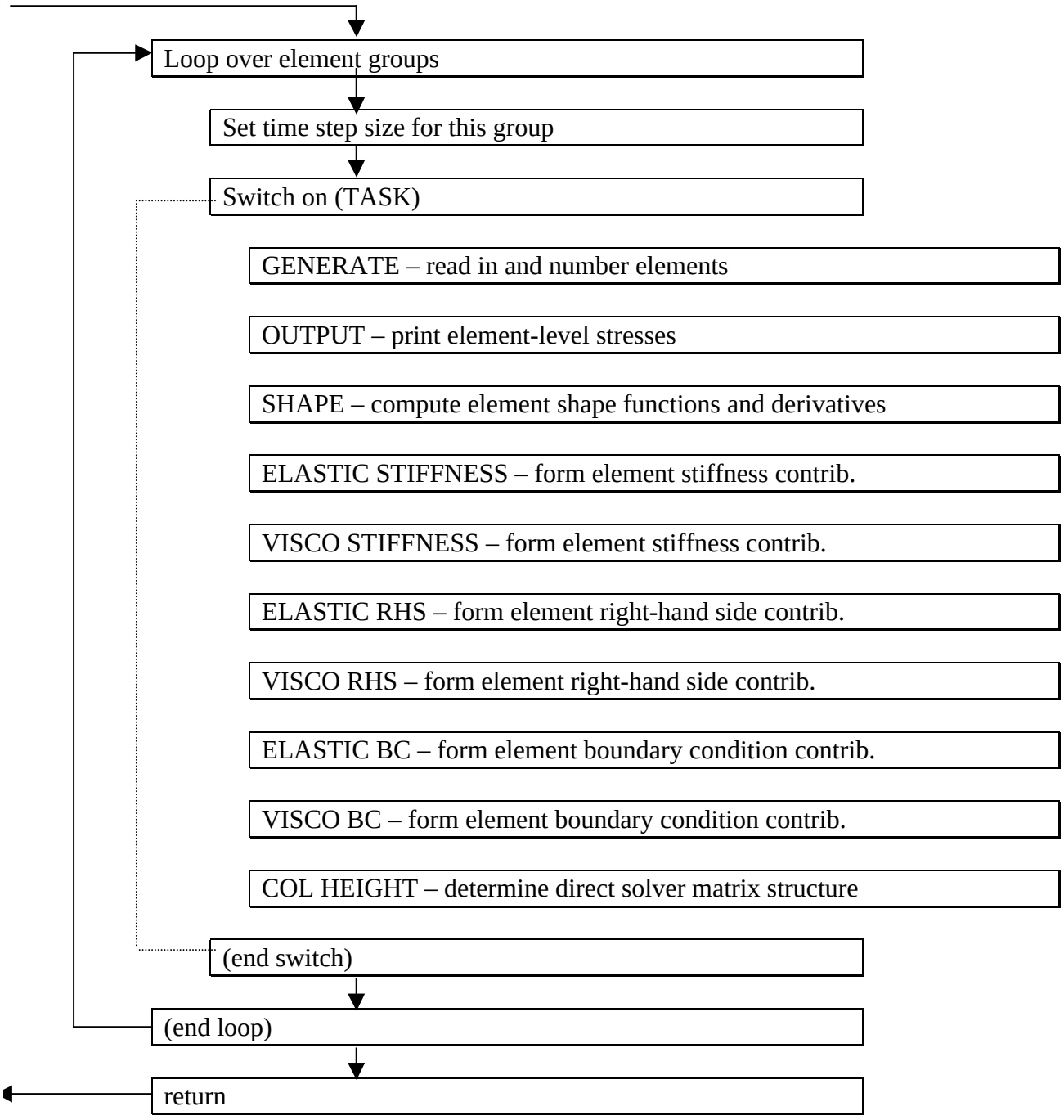
(iii) elastic routine



(iv) time_step routine



(v) group_loop routine



The following is a listing of all GeoFEST functional routines and the header comments describing their usage and purpose.

```
/*
***                          File main.c                          ***
***                       GeoFEST version 4.2                     ***
*** Copyright (c) 2002, California Institute of Technology        ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged ***
***
*** This file contains the program main entry point, the main
*** task driver, and modules for driving high-level functions
*** and interactions with the operator:
***
***          == main ==
***          == elastic ==
***          == time_step ==
***          == clear_stiff ==
***          == elgrp_loop ==
***          == node_load ==
***          == accumulate ==
***          == completion ==
***          == wrt_save ==
***          == vrestart ==
***          == el_save ==
***          == check_monitor ==
***/

/*-------------------------------------------------------------
------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   main   ***************
*/

/*  INTERFACE:   */

     main(
          int argc ,      /* number of input arguments */
          char *argv[]    /* input argument strings */
         )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine main
** main prints a banner,
** interprets up to two args as input and output files,
** scans two comment lines in the input,
** and divides (and times) the processing
** into input, elastic solution, time-stepping, and output.
**/
/*   EOP   */
/*-------------------------------------------------------------
------------------*/

/*-------------------------------------------------------------
------------------*/
```

```
/*   BOP   */
/*
   *************   ROUTINE:   elastic   ***************
*/

/*  INTERFACE:   */

     elastic(
            )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine elastic
** elastic performs an elastic solution of the
** finite element problem.
**/
/*   EOP   */
/*-------------------------------------------------------------
------------------*/

/*-------------------------------------------------------------
------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   time_step   ***************
*/

/*  INTERFACE:   */

     time_step(
              )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine time_step
** time_step computes the visco-elastic time stepping
solutions
** to the finite element problem.
**/
/*   EOP   */
/*-------------------------------------------------------------
------------------*/

/*-------------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   clear_stiff   ***************
*/

/*  INTERFACE:   */

     clear_stiff(
                 )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
```

```
/*
** Routine clear_stiff
** clear_stiff nulls the stiffness matrix memory.
**/
/*   EOP   */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   elgrp_loop   ***************
*/

/*  INTERFACE:   */

     elgrp_loop(
                int  task      /* which element-level task to perform */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine elgrp_loop
** elgrp_loop invokes a particular task that relies on
** element structures and affects all the elements in
** every element group.
**
** Supported tasks are
** GENERATE: read or generate element data, fill ien, lm arrays
** OUTPUT: compute and write out element centroid stresses
** SHAPE: compute parent-space shape functions and gradients
** FORMS_ELAS: compute elastic stiffness matrix
** FORMS_STEP: compute VE single-step stiffness matrix
** RHS_ELAS: compute elastic right-hand side vector for FE problem
** RHS_STEP: compute VE right-hand side vector for FE problem
** BC_ELAS:  compute boundary-condition terms for elastic FE problem
** BC_STEP: compute boundary-condition terms for VE FE single step
** COL_HT: compute the column heights for profile stiffness storage
** E_STRESS: compute the stresses from elastic solution
** V_STRESS: compute the stresses from a VE step solution
** DUMP: dump information needed for restarts
** RESTORE: read and restore solution vector and info for a restart
** FAIL_CHECK: check for stress-driven fault failure
** SMOOTH_BEGIN: perform pressure smoothing set-up
** SMOOTH_END: apply pressure-smoothing function from coefficients
** REORDER: analyze adjacency, find profile-optimizing node permutation
**/
/*   EOP   */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   node_load   ***************
*/

/*  INTERFACE:   */
```

```
     node_load(
               real  time ,   /* current simulation time */
               int reform     /* flag indicating stiffness reform status */
              )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine node_load
** node_load applies node-based forces that may accumulate
** with time.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   accumulate   ***************
*/

/*  INTERFACE:   */

     accumulate(
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine accumulate
** accumulate updates the total displacement based on the
** single-step increment.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   completion   ***************
*/

/*  INTERFACE:   */

     completion(
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine completion
** completion syncronizes multiple processors.
** Parallel vestige; keep for now.
** Keeps track of incurred errors in each processor.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/
```

```
rt_save   ***************
*/

/*  INTERFACE:   */

     wrt_save(
             )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine wrt_save
** wrt_save dumps state to disk for restarts.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   vrestart   ***************
*/

/*  INTERFACE:   */

     vrestart(
             )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** vrestart restores state from previous dump and
** continues simulation.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

l_save   ***************
*/

/*  INTERFACE:   */

     el_save(
             GROUP   *grp_ptr ,       /* pointer to current element group */
             int     code             /* code specifying which action */
            )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine el_save
** el_save dumps or restores element data.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
```

```
/*
   *************   ROUTINE:   check_monitor   ***************
*/

/*  INTERFACE:   */

     check_monitor(
                  )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine check_monitor
** check_monitor allows user run intervention via strings stored
** in the file monitor.fem.
**
** Supported options:
** OK: close the dump file (?)
** KILL: close files and abort
** SAVE_STOP: write state, then abort
** SAVE_GO:  write dumps as requested in input, and continue
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*
***                          File inphase.c                             ***
***                       GeoFEST version 4.2                           ***
*** Copyright (c) 2002, California Institute of Technology              ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged ***
***
*** This file contains subroutines which handle input,
*** equation numbering, and memory allocation.
***
***          == input_phase ==
***          == matrix_alloc ==
***          == gen_number ==
***          == output_phase ==
***          == el_output ==
***          == locate_pt ==
***/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   input_phase   ***************
*/

/*  INTERFACE:   */

         input_phase()

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine input_phase
** input_phase is the top-level input file reading function.
** Most input-file records are read here or in routines
```

```
** in generat.c that are called by this function.
** A few items are read by main.c as well.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   matrix_alloc   ***************
*/

/*  INTERFACE:   */

         matrix_alloc()

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine matrix_alloc
** matrix allocates space for the sparse stiffness matrix _or_
** the PCG solver arrays
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   gen_number   ***************
*/

/*  INTERFACE:   */

         gen_number()

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine gen_number
** gen_number assigns equation numbers based on nodes and ndof
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   output_phase   ***************
*/

/*  INTERFACE:   */

   output_phase(
                int    quake   /* flag to indicate quake or not */
               )

/*  RETURN VALUE:  -none- */
```

```
/*  DESCRIPTION:   */
/*
** Routine output_phase
** output_phase writes out requested data at a quake
** or scheduled time.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   el_output   ***************
*/

/*  INTERFACE:   */

      el_output(
                GROUP   *grp_ptr    /* pointer to current element group */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine el_output
** el_output prints out element stress information.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   locate_pt   ***************
*/

/*  INTERFACE:   */

      locate_pt(
                ELEMENT_DATA  *el_pt  ,  /* ptr to current element struct */
                ELEMENT_INFO  *info ,   /* ptr to element info struct */
                real   xpt[3][10]     /* return array of computed coords */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine locate_pt
** locate_pt maps the integration points into physical space.
** Global coordinates are placed in xpt[j]
** Called by el_output element stress output function (above).
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*
***                          File generat.c                          ***
***                       GeoFEST version 4.2                        ***
```

```
*** Copyright (c) 2002, California Institute of Technology        ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged ***
***
*** This file contains modules for generating and reading
*** node and element data and related isoparametric tasks
***
***          == gen_element ==
***          == next_element ==
***          == read_surf ==
***          == read_slip ==
***          == load_element ==
***          == gen_map ==
***          == gen_real ==
***          == gen_isopar_grid ==
***          == gen_shape_grid ==
***          == dot_sh ==
***
***/
/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   gen_element   ***************
*/

/*  INTERFACE:   */

    gen_element(
                GROUP   *grp_ptr    /* pointer to current element group */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine gen_element
** gen_element generates element storage and ien array
** It reads in element group information, individual elements,
** and surface traction records.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   next_element   ***************
*/

/*  INTERFACE:   */

   int  next_element(
                     int   *node_pointer , /* pointer to hold node */
                     int   *gen_pointer   /* pointer to gen parameter */
                    )

/*  RETURN VALUE:  number of the next node or object read in */
/*  DESCRIPTION:   */
/*
```

```
** Function next_element
** next_element reads the first part of a record in a double
** null-terminated list, so the calling function may
** catch the termination.
**
** The use of this function is not limited to finite elements, but is
** also employed for other lists, such as nodes and tractions.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   read_surf   ***************
*/

/*  INTERFACE:   */

     read_surf(
               int  *tally ,  /* counter for records read in */
               ELEMENT_DATA  *el_pt ,  /* ptr to current element struct */
               int  *temp_list ,  /* storage for surf location data */
               real  *temp_trac ,  /* storage for surf traction data */
               int  type ,    /* element type number */
               int  numel ,  /* number of elements */
               int  ndof    /* number of disp degrees of freedom */
              )

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
** Routine read_surf
** read_surf reads in surface traction records.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   read_slip   ***************
*/

/*  INTERFACE:   */

     read_slip(
               int  *tally ,  /* counter for records read in */
               ELEMENT_DATA  *el_data ,  /* ptr to current element struct */
               int  *temp_list ,  /* storage for split node location data */
               real  *temp_val ,  /* storage for split node slip data */
               int  type ,    /* element type number */
               int  numel ,  /* number of elements */
               int  ndof    /* number of disp degrees of freedom */
              )

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
```

```
** Routine read_slip
** read_slip reads split-node records for symmetric fault
** slip accumulation.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   load_element   ***************
*/

/*  INTERFACE:   */

  load_element(
               ELEMENT_DATA  *el_pt  ,  /* ptr to current element struct */
               ELEMENT_INFO  *info ,   /* ptr to element info struct */
               int      nel_glob ,      /* one-based global element number */
               int      ien[] ,     /* holding for ien assignments */
               int      mat       /* one-based material index number */
              )

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
** Routine load_element
** load_element loads element arrays with ien and equation data.
** The information for the element has been previously read into a
** temporary array or generated.
**
** This function checks for degeneracies of 2D elements,
** right-hand rule order of tet nodes (swapping when necessary),
** and fills in the ien and lm arrays for the element.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   gen_map   ***************
*/

/*  INTERFACE:   */

    gen_map()

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
** Routine gen_map
** gen_map fills the nodal activity map array.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   gen_real   ***************
*/

/*  INTERFACE:   */

    gen_real(
             real      *array ,  /* array of generated values */
             int        dim      /* dimensionality of grid */
            )

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
** Routine gen_real
** gen_real reads or generates floating point global array.
** Used to read or generate nodes, displacement bcs, or
** velocity bcs.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   gen_isopar_grid   ***************
*/

/*  INTERFACE:   */

 gen_isopar_grid(
                real      *array ,  /* array of generated values */
                int        dim      /* dimensionality of grid */
               )

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
** Routine gen_isopar_grid
** isopar_grid generates real data via isoparametric interpolation
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   gen_shape_grid   ***************
*/

/*  INTERFACE:   */

  gen_shape_grid(
                real      r  ,  /* isopar coordinate */
                real      s  ,  /* isopar coordinate */
                real      t  ,  /* isopar coordinate */
```

```
                real      *shape  ,  /* shape function storage */
                int       option      /* generation option flag */
               )

/*  RETURN VALUE:   -none- */
/*  DESCRIPTION:   */
/*
** Routine gen_shape_grid
** gen_shape_grid finds shape function for isoparametric generation.
** It appears to JWP to use bilinear and trilinear interpolation.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   dot_sh   ***************
*/

/*  INTERFACE:   */

    real  dot_sh(
                real      *shape  ,     /* shape function storage */
                real      *variable  ,  /* interpolate variable storage */
                int        n            /* number to sum over */
               )

/*  RETURN VALUE:   dot product value */
/*  DESCRIPTION:   */
/*
** Function dot_sh
** dot_sh performs the dot product of the vector of shape functions
** (may be gradients of shape functions)
** with the vector "variable"
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*
***                          File stiff.c                           ***
***                        GeoFEST version 4.2                      ***
*** Copyright (c) 2002, California Institute of Technology         ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged ***
***
*** This file contains modules which construct the finite
*** element stiffness matrix and the right-hand-side "force" vector.
***
***          == form_stiff ==
***          == form_rhs ==
***          == form_bc ==
***          == form_slip ==
***          == adjust_bc ==
***          == lame_form ==
***          == force_form ==
***          == surf_form ==
***          == shape ==
***          == dotsh ==
```

```
***          == adfldp ==
***          == p_shape ==
***          == form_smooth == - disabled -
***          == addsmooth == - disabled -
***          == apply_smooth == - disabled -
***/

/*--------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   form_stiff   ***************
*/

/*  INTERFACE:   */

     form_stiff(
                GROUP   *grp_ptr ,   /* pointer to current element group */
                int     code         /* code specifying elastic or visco */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_stiff
** form_stiff computes the element-wise stiffness array for the elt group.
** "code" indicates if this is for an elastic problem or a VE step.
**/
/*   EOP   */
/*--------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   form_rhs   ***************
*/

/*  INTERFACE:   */

     form_rhs(
              GROUP   *grp_ptr ,   /* pointer to current element group */
              int     code         /* code specifying elastic or visco */
             )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_rhs
** form_rhs computes the right-hand-side vector for the FE problem.
** "code" indicates if this is for the elastic problem or a VE step.
**/
/*   EOP   */
/*--------------------------------------------------------------------------*/
/*--------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   form_bc   ***************
*/
```

```
/*  INTERFACE:   */

     form_bc(
             GROUP   *grp_ptr ,   /* pointer to current element group */
             int     code         /* code specifying elastic or visco */
            )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_bc
** form_bc computes the boundary condition terms for stiffness and rhs
** due to imposed conditions
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/


/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   form_slip   ***************
*/

/*  INTERFACE:   */

     form_slip(
               int  code ,     /* code specifying elastic or visco */
               real  *rhs ,   /* temp array to hold element rhs contrib */
               real  *stiff , /* temp array to hold el stiffness contrib */
               ELEMENT_INFO  *info ,   /* ptr to element info struct */
               int  nel ,       /* current element number */
               int  node ,      /* current node number */
               real  *slip ,   /* array containing fault slip amplitudes */
               ELEMENT_DATA  *el_pt ,  /* ptr to current element struct */
               ELEMENT_MAT   *mat_pt  /* ptr to el material prop struct */
              )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_slip
** form_slip computes the split-nodes fault offsets and their influence
** on the finite element stiffness and RHS.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   adjust_bc   ***************
*/

/*  INTERFACE:   */

     adjust_bc(
               real  *disp ,   /* array containing fault slip amplitudes */
```

```
              real  *rhs ,   /* temp array to hold element rhs contrib */
              real  *es ,    /* temp array to hold el stiffness contrib */
              int   nee        /* number of element equations */
             )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine adjust_bc
** adjust_bc computes the necessary terms that modify the right-hand-side
** due to imposed displacements.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   lame_form   ***************
*/

/*  INTERFACE:   */

     lame_form(
              int  code ,     /* code specifying elastic or visco */
              real  *es ,    /* temp array to hold el stiffness contrib */
              ELEMENT_INFO  *info ,   /* ptr to element info struct */
              ELEMENT_MAT   *mat_pt ,  /* ptr to el material prop struct */
              ELEMENT_DATA  *el_pt    /* ptr to current element struct */
             )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine lame_form
** lame_form computes the volumetric stiffness term for one element,
** that is based on the constitutive volume constants.
** Results are stored in the element stiffness array.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   force_form   ***************
*/

/*  INTERFACE:   */

    force_form(
              int  code ,     /* code specifying elastic or visco */
              real  *rhs ,   /* temp array to hold element rhs contrib */
              ELEMENT_INFO  *info ,   /* ptr to element info struct */
              ELEMENT_MAT   *mat_pt ,  /* ptr to el material prop struct */
              ELEMENT_DATA  *el_pt    /* ptr to current element struct */
             )
```

```
/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine force_form
** force_form computes the "dumb" gravity contribution to the element
** right-hand side term for the elastic case, and the strain-rate
** term for the VE case.
**/
/*   EOP   */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   surf_form   ***************
*/

/*  INTERFACE:   */

     surf_form(
               int  code ,     /* code specifying elastic or visco */
               real  *rhs ,   /* temp array to hold element rhs contrib */
               ELEMENT_INFO  *info ,   /* ptr to element info struct */
               int  side ,    /* index number of the element side */
               real  *trac ,   /* array containg traction components */
               ELEMENT_DATA  *el_pt    /* ptr to current element struct */
              )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine surf_form
** surf_form computes the surface traction forcing term for the
** right-hand-side in the finite element problem.
**/
/*   EOP   */
/*------------------------------------------------------------------------*/

/*------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   shape   ***************
*/

/*  INTERFACE:   */

     shape(
           ELEMENT_DATA  *el_pt  ,  /* ptr to current element struct */
           ELEMENT_INFO  *info ,   /* ptr to element info struct */
           real  *sh_pt ,   /* el storage to hold shape functions */
           real  *det_pt ,   /* el storage to hold determinant */
           int  type ,    /* element type number */
           int  nen ,    /* number of element nodes */
           int  nint ,   /* number of Gauss integration points */
           int  all     /* flag indicating we need derivatives also */
          )

/*  RETURN VALUE:  -none- */
```

```
/*  DESCRIPTION:   */
/*
** Routine shape
** shape computes the shape functions and gradients for a single element.
**/
/*   EOP   */
/*------------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   dotsh   ***************
*/

/*  INTERFACE:   */

real dotsh(
           real  *array ,  /* array of nodal quantities to interpolate */
           int  i ,     /* derivative component index number */
           int  j ,     /* dof index number of quantity component */
           real  *sh ,   /* el storage containing shape functions */
           ELEMENT_DATA  *el_pt  ,  /* ptr to current element struct */
           ELEMENT_INFO  *info ,   /* ptr to element info struct */
           int  nen ,    /* number of element nodes */
           int  ndim ,    /* number of dimensions in array */
           int  st_flag      /* flag to invoke split node adjustment */
          )

/*  RETURN VALUE:  the element-interpolated sum value */
/*  DESCRIPTION:   */
/*
** Function dotsh
** dotsh computes the dot product of a set of shape-functions (or gradients)
** with a vector of values, hence computing an interpolated coordinate
** or function. If st_flag, x is assumed to represent displacement, and
** a correction for split-nodes is applied prior to the dot product.
**/
/*   EOP   */
/*------------------------------------------------------------------------------*/

/*------------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   adfldp   ***************
*/

/*  INTERFACE:   */

real adfldp(
            int  j ,     /* dof index number of slip component */
            int  nel ,    /* number of current element */
            int  node ,    /* number of current node */
            ELEMENT_INFO  *info    /* ptr to element info struct */
           )

/*  RETURN VALUE:  the required additional nodal displacement */
/*  DESCRIPTION:   */
/*
```

```
** Function adfldp
** adfldp returns to dotsh() the added nodal displacements
** needed to account for split node fault slip in calculating strain
**/
/*   EOP   */
/*--------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   p_shape   ***************
*/

/*  INTERFACE:   */

     p_shape(
             GROUP   *grp_ptr     /* pointer to current element group */
            )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine p_shape
** p_shape computes the parent-space shape function for common element types.
**/
/*   EOP   */
/*--------------------------------------------------------------------------*/

/*
***                          File solver.c                               ***
***                         GeoFEST version 4.2                          ***
*** Copyright (c) 2002, California Institute of Technology               ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged        ***
***
*** This file contains modules for the sparse matrix storage,
*** substructure reduction or other solvers of the matrix system:
***
***          == solver ==
***          == addstiff ==
***          == addfor ==
***          == estiffprod ==
***          == reorder ==
***          == genrcm ==
***          == rcm ==
***          == degree ==
***          == fnroot ==
***          == rootls ==
***          == colht ==
***          == profile_diag ==
***          == factor ==
***          == full_back ==
***          == pcg_loop ==
***          == converged ==
***          == put_soln ==
***/

/*--------------------------------------------------------------------------*/
/*   BOP   */
```

```
/*
   *************   ROUTINE:   solver   ***************
*/

/*  INTERFACE:   */

     solver(
            int  code     /* which solver option to use */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine solver
** solver performs single right-hand-side matrix solution,
** currently using factorization and back substitution on
** single processor OR using PCG iteration.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   addstiff   ***************
*/

/*  INTERFACE:   */

   addstiff(
            PROFILE  *a ,  /* ptr to assembled profile stiffness storage */
            real  *es ,        /* ptr to element stiffness storage */
            ELEMENT_INFO  *info ,   /* ptr to element info struct */
            ELEMENT_DATA  *el_pt    /* ptr to current element struct */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine addstiff
** addstiff adds element stiffness to global profile array,
** or to element storage if using PCG solver
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   addfor   ***************
*/

/*  INTERFACE:   */

     addfor(
            real  *dest ,  /* destination assembled rhs vector */
            real  *rhs ,        /* source element rhs vector */
            ELEMENT_INFO  *info ,   /* ptr to element info struct */
```

```
            ELEMENT_DATA  *el_pt    /* ptr to current element struct */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine addfor
** addfor assembles the global r.h.s. vector.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   estiffprod   ***************
*/

/*  INTERFACE:   */

 estiffprod(
            GROUP   *grp_ptr    /* pointer to current element group */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine estiffprod
** calculates the product of a given vector with stiffness in element storage
**    t = A * d
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   reorder   ***************
*/

/*  INTERFACE:   */

    reorder(
            GROUP   *grp_ptr    /* pointer to current element group */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine reorder
** reorder uses ien information to build adjacency information and call
** permutation optimizing routines for minimizing matrix profile.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/

/*-----------------------------------------------------------------------------*/
/*   BOP   */
```

```
/*
   *************   ROUTINE:   genrcm   ***************
*/

/*  INTERFACE:   */

     genrcm(
            int  neqns ,   /* NUMBER OF EQUATIONS */
            int  *xadj ,  /* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *adjncy ,/* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *perm ,   /* VECTOR THAT CONTAINS THE RCM ORDERING */
            int  *mask ,   /* MARK VARS NUMBERED IN ORDERING PROCESS */
            int  *xls   /* INDEX VECTOR FOR A LEVEL STRUCTURE */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine genrcm
** genrcm computes the reverse cuthill mckee ordering for a general
** adjacency graph such as a sparse finite element matrix.
**
** Original FORTRAN comments: (transcribed into C by JWP from
** p_reorder.f of PHOEBUS3_T3D):
*****************************************************************
*****************************************************************
**********   GENRCM ..... GENERAL REVERSE CUTHILL MCKEE   *******
*****************************************************************
*****************************************************************
**
**     PURPOSE - GENRCM FINDS THE REVERSE CUTHILL-MCKEE
**        ORDERING FOR A GENERAL GRAPH. FOR EACH CONNECTED
**        COMPONENT IN THE GRAPH, GENRCM OBTAINS THE ORDERING
**        BY CALLING THE SUBROUTINE RCM.
**
**     INPUT PARAMETERS -
**        NEQNS - NUMBER OF EQUATIONS
**        (XADJ, ADJNCY) - ARRAY PAIR CONTAINING THE ADJACENCY
**               STRUCTURE OF THE GRAPH OF THE MATRIX.
**
**     OUTPUT PARAMETER -
**        PERM - VECTOR THAT CONTAINS THE RCM ORDERING.
**
**     WORKING PARAMETERS -
**        MASK - IS USED TO MARK VARIABLES THAT HAVE BEEN
**               NUMBERED DURING THE ORDERING PROCESS. IT IS
**               INITIALIZED TO 1, AND SET TO ZERO AS EACH NODE
**               IS NUMBERED.
**        XLS - THE INDEX VECTOR FOR A LEVEL STRUCTURE.  THE
**               LEVEL STRUCTURE IS STORED IN THE CURRENTLY
**               UNUSED SPACES IN THE PERMUTATION VECTOR PERM.
**
**     PROGRAM SUBROUTINES -
**        FNROOT, RCM.
**
*****************************************************************
**/
/*   EOP   */
```

```
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   rcm   ***************
*/

/*  INTERFACE:   */

        rcm(
            int  root ,   /* NODE THAT DEFINES THE CONNECTED COMPONENT */
            int  *xadj ,  /* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *adjncy ,/* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *mask ,   /* MARK VARS NUMBERED IN ORDERING PROCESS */
            int  *perm ,   /* VECTOR THAT CONTAINS THE RCM ORDERING */
            int  num ,   /*  */
            int  *ccsize ,   /* SIZE OF THE CONNECTED COMPONENT */
            int  *deg   /* TEMPORARY VECTOR USED TO HOLD THE DEGREE */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** rcm does connectivity analysis (jwp interpretation; see below).
** Original comments from Fortran code:
************************************************************
************************************************************
**********     RCM ..... REVERSE CUTHILL-MCKEE ORDERING   *******
************************************************************
************************************************************
**
**     PURPOSE - RCM NUMBERS A CONNECTED COMPONENT SPECIFIED BY            7.
**        MASK AND ROOT, USING THE RCM ALGORITHM.                          8.
**        THE NUMBERING IS TO BE STARTED AT THE NODE ROOT.                 9.
**
**     INPUT PARAMETERS -                                                 11.
**        ROOT - IS THE NODE THAT DEFINES THE CONNECTED                   12.
**               COMPONENT AND IT IS USED AS THE STARTING                 13.
**               NODE FOR THE RCM ORDERING.                               14.
**        (XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR FOR                    15.
**               THE GRAPH.                                               16.
**
**     UPDATED PARAMETERS -                                               18.
**        MASK - ONLY THOSE NODES WITH NONZERO INPUT MASK                  19.
**               VALUES ARE CONSIDERED BY THE ROUTINE.  THE               20.
**               NODES NUMBERED BY RCM WILL HAVE THEIR                    21.
**               MASK VALUES SET TO ZERO.                                 22.
**
**     OUTPUT PARAMETERS -                                                24.
**        PERM - WILL CONTAIN THE RCM ORDERING.                           25.
**        CCSIZE - IS THE SIZE OF THE CONNECTED COMPONENT                 26.
**               THAT HAS BEEN NUMBERED BY RCM.                           27.
**
**     WORKING PARAMETER -                                                29.
**        DEG - IS A TEMPORARY VECTOR USED TO HOLD THE DEGREE             30.
**               OF THE NODES IN THE SECTION GRAPH SPECIFIED               31.
**               BY MASK AND ROOT.                                        32.
```

```
**
**     PROGRAM SUBROUTINES -                                                  34.
**        DEGREE.                                                             35.
**
*************************************************************
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   degree   ***************
*/

/*  INTERFACE:   */

     degree(
            int  root ,   /* NODE THAT DEFINES THE CONNECTED COMPONENT */
            int  *xadj ,  /* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *adjncy ,/* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *mask ,   /* SPECIFIES A SECTION SUBGRAPH */
            int  *deg  , /* ARRAY CONTAINING THE DEGREES OF THE NODES */
            int  *ccsize ,   /* SIZE OF THE COMPONENT SPECIFED */
            int  *ls ,   /* TEMPORARY VECTOR USED TO STORE THE NODES */
            int  num     /*  */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine degree
** degree computes the graph steps to each node in the subgraph.
** Original Comments from Fortran code:
*************************************************************
*************************************************************
**********     DEGREE ..... DEGREE IN MASKED COMPONENT   ********
*************************************************************
*************************************************************
**
**     PURPOSE - THIS ROUTINE COMPUTES THE DEGREES OF THE NODES             7.
**        IN THE CONNECTED COMPONENT SPECIFIED BY MASK AND ROOT.             8.
**        NODES FOR WHICH MASK IS ZERO ARE IGNORED.                          9.
**
**     INPUT PARAMETER -                                                   11.
**        ROOT - IS THE INPUT NODE THAT DEFINES THE COMPONENT.             12.
**        (XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR.                       13.
**        MASK - SPECIFIES A SECTION SUBGRAPH.                             14.
**
**     OUTPUT PARAMETERS -                                                 16.
**        DEG - ARRAY CONTAINING THE DEGREES OF THE NODES IN               17.
**              THE COMPONENT.                                             18.
**        CCSIZE-SIZE OF THE COMPONENT SPECIFED BY MASK AND ROOT           19.
**
**     WORKING PARAMETER -                                                 21.
**        LS - A TEMPORARY VECTOR USED TO STORE THE NODES OF THE           22.
**               COMPONENT LEVEL BY LEVEL.                                 23.
**
```

```
************************************************************
**/
/*   EOP   */
/*--------------------------------------------------------------------------*/

/*--------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   fnroot   ***************
*/

/*  INTERFACE:   */

     fnroot(
            int  *root ,   /* DEFINES THE COMP FOR PSEUDO-PERIPH NODE */
            int  *xadj ,  /* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *adjncy ,/* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *mask ,   /* SPECIFIES A SECTION SUBGRAPH */
            int  *nlvl ,  /* NUMBER OF LEVELS IN THE LEVEL STRUCTURE */
            int  *xls ,   /* LEVEL STRUCTURE ARRAY PAIR */
            int  *ls ,   /* LEVEL STRUCTURE ARRAY PAIR */
            int  num     /*  */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine fnroot
** fnroot finds the pseudo-peripheral node for a given subgraph.
** Original comments from Fortran:
************************************************************
************************************************************
*********     FNROOT ..... FIND PSEUDO-PERIPHERAL NODE    *******
************************************************************
************************************************************
**
**    PURPOSE - FNROOT IMPLEMENTS A MODIFIED VERSION OF THE                7.
**       SCHEME BY GIBBS, POOLE, AND STOCKMEYER TO FIND PSEUDO-            8.
**       PERIPHERAL NODES.  IT DETERMINES SUCH A NODE FOR THE              9.
**       SECTION SUBGRAPH SPECIFIED BY MASK AND ROOT.                     10.
**
**    INPUT PARAMETERS -                                                  12.
**       (XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR FOR THE GRAPH.         13.
**       MASK - SPECIFIES A SECTION SUBGRAPH. NODES FOR WHICH             14.
**              MASK IS ZERO ARE IGNORED BY FNROOT.                       15.
**
**    UPDATED PARAMETER -                                                 17.
**       ROOT - ON INPUT, IT (ALONG WITH MASK) DEFINES THE                18.
**              COMPONENT FOR WHICH A PSEUDO-PERIPHERAL NODE IS           19.
**              TO BE FOUND. ON OUTPUT, IT IS THE NODE OBTAINED.          20.
**
**    OUTPUT PARAMETERS -                                                 22.
**       NLVL - IS THE NUMBER OF LEVELS IN THE LEVEL STRUCTURE            23.
**              ROOTED AT THE NODE ROOT.                                  24.
**       (XLS,LS) - THE LEVEL STRUCTURE ARRAY PAIR CONTAINING             25.
**                  THE LEVEL STRUCTURE FOUND.                            26.
**
**    PROGRAM SUBROUTINES -                                               28.
```

```
**       ROOTLS.                                                          29.
**
*************************************************************
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   rootls   ***************
*/

/*  INTERFACE:   */

     rootls(
            int  root ,   /* NODE AT WHICH LEVEL STRUCTURE IS ROOTED */
            int  *xadj ,  /* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *adjncy ,/* ADJACENCY STRUCTURE OF THE GRAPH OF THE MATRIX */
            int  *mask ,   /* SPECIFIES A SECTION SUBGRAPH */
            int  *nlvl ,  /* NUMBER OF LEVELS IN THE LEVEL STRUCTURE */
            int  *xls ,   /* LEVEL STRUCTURE ARRAY PAIR */
            int  *ls ,   /* LEVEL STRUCTURE ARRAY PAIR */
            int  num     /*  */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine rootls
** rootls generates the level structure corresponding to "root".
** Original Fortran comments:
*************************************************************
*************************************************************
**********     ROOTLS ..... ROOTED LEVEL STRUCTURE      *********
*************************************************************
*************************************************************
**
**     PURPOSE - ROOTLS GENERATES THE LEVEL STRUCTURE ROOTED
**        AT THE INPUT NODE CALLED ROOT. ONLY THOSE NODES FOR
**        WHICH MASK IS NONZERO WILL BE CONSIDERED.
**
**     INPUT PARAMETERS -
**        ROOT - THE NODE AT WHICH THE LEVEL STRUCTURE IS TO
**               BE ROOTED.
**        (XADJ, ADJNCY) - ADJACENCY STRUCTURE PAIR FOR THE
**               GIVEN GRAPH.
**        MASK - IS USED TO SPECIFY A SECTION SUBGRAPH. NODES
**               WITH MASK(I)=0 ARE IGNORED.
**
**     OUTPUT PARAMETERS -
**        NLVL - IS THE NUMBER OF LEVELS IN THE LEVEL STRUCTURE.
**        (XLS, LS) - ARRAY PAIR FOR THE ROOTED LEVEL STRUCTURE.
**
*************************************************************
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/
```

```
/*---------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   colht   ***************
*/

/*  INTERFACE:   */

      colht(
            GROUP    *grp_ptr  ,  /* pointer to current element group */
            PROFILE  *a    /* ptr to assembled profile stiffness storage */
           )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine colht
** colht computes column heights of global array.
**/
/*   EOP   */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   profile_diag   ***************
*/

/*  INTERFACE:   */

 profile_diag(
              PROFILE  *a  ,  /* ptr to assembled profile stiffness storage */
              int   profile_case    /* old flag for substructuring... */
             )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine profile_diag
** profile diag computes the diagonal addresses.
**/
/*   EOP   */
/*---------------------------------------------------------------------------*/

/*---------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   factor   ***************
*/

/*  INTERFACE:   */

       factor(
              real  *stiff ,         /* assembled profile matrix */
              int   *diag ,           /* array of diagonal addresses */
              int   number_of_eqs     /* number of equations */
             )
```

```
/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine factor
** factor performs profile-based factorization:
**                          t
**                     a=(u) * d * u (crout)
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   full_back   ***************
*/

/*  INTERFACE:   */

    full_back(
              real  *stiff ,         /* factored profile matrix */
              real  *rhs ,         /* assembled rhs vector */
              int   *diag ,           /* array of diagonal addresses */
              int   number_of_eqs     /* number of equations */
             )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine full_back
** full_back performs all three steps of backsubstitution.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   pcg_loop   ***************
*/

/*  INTERFACE:   */

     pcg_loop()

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine pcg_loop
** pcg_loop performs the preconditioned conjugate gradient iteration
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
```

```
   *************   ROUTINE:   converged   ***************
*/

/*  INTERFACE:   */

  int converged(
               int   k   /* iteration number */
              )

/*  RETURN VALUE:  flag indicating successful convergence (or not) */
/*  DESCRIPTION:   */
/*
** function converged()
** monitors convergence of CG solver from magnitude and history of
** the residual norm
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   put_soln   ***************
*/

/*  INTERFACE:   */

       put_soln(
                RHS_DATA  *solve , /* structure containing rhs arrays */
                int       *id      /* id array of eq numbers */
               )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine put_soln
** put_soln transfers solution to nodal storage.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*
***                          File strain.c                          ***
***                        GeoFEST version 4.2                      ***
*** Copyright (c) 2002, California Institute of Technology         ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged ***
***
*** This file contains modules which perform calculations
*** related to element stresses and strains:
***
***          == form_stress ==
***          == form_beta ==
***          == form_dbar ==
***/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
```

```
   *************   ROUTINE:   form_stress   ***************
*/

/*  INTERFACE:   */

     form_stress(
                 GROUP   *grp_ptr ,   /* pointer to current element group */
                 int     code         /* code specifying elastic or visco */
                )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_stress
** form_stress computes the stress or time-step stress increment
** based on the FE solution.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   form_beta   ***************
*/

/*  INTERFACE:   */

     form_beta(
                ELEMENT_DATA  *el_pt  , /* ptr to current element struct */
                ELEMENT_MAT   *mat_pt , /* ptr to el material prop struct */
                ELEMENT_INFO  *info     /* ptr to element info struct */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_beta
** form_beta computes beta, the viscoplastic strain rate for
** this element and time step.
**/
/*   EOP   */
/*-----------------------------------------------------------------------------*/


/*-----------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   form_dbar   ***************
*/

/*  INTERFACE:   */

     form_dbar(
                ELEMENT_DATA  *el_pt ,  /* ptr to current element struct */
                ELEMENT_MAT   *mat_pt , /* ptr to el material prop struct */
                ELEMENT_INFO  *info     /* ptr to element info struct */
               )
```

```
/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine form_dbar
** form_dbar computes dbar, the VE single-step
** constitutive matrix:
**
** dbar === (S + alpha delta_t beta')^-1
**
** but we compute the factored form, not the explicit inverse.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*
***                          File utility.c                             ***
***                       GeoFEST version 4.2                           ***
*** Copyright (c) 2002, California Institute of Technology              ***
*** U.S.Sponsorship under NASA Contract NAS7-1260 is acknowledged ***
***
*** This file contains miscellaneous utility routines used throughout
*** the finite element program:
***
***          == move_real ==
***          == clear_real ==
***          == dot_real ==
***          == vadd ==
***          == vouter ==
***/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   move_real   ***************
*/

/*  INTERFACE:   */

      move_real(
                real   *from , /* source array */
                real   *to ,   /* destination array */
                int    n       /* number of entries */
               )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine move_real
** move_real copies data to a new location.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   clear_real   ***************
*/
```

```
/*  INTERFACE:   */

     clear_real(
                real   *array , /* array to be zapped */
                int    n       /* number of entries */
               )

/*  RETURN VALUE:  - none - */
/*  DESCRIPTION:   */
/*
** Routine clear_real
** clear_real nulls a data area.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   dot_real   ***************
*/

/*  INTERFACE:   */

  real dot_real(
                real   *vect_1 , /* multiplicand array */
                real   *vect_2 , /* multiplicand array */
                int    n       /* number of entries */
               )

/*  RETURN VALUE:  value of the scalar product */
/*  DESCRIPTION:   */
/*
** Function dot_real
** dot_real forms dot product of two vectors.
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   vadd   ***************
*/

/*  INTERFACE:   */

           vadd(
                real   *v1 , /* first array */
                real   mult , /* scalar multiplier for second array */
                real   *v2 , /* second array */
                real   *dest , /* destination array */
                int    neq       /* number of entries */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
```

```
/*
** Routine vadd
** calculates the linear combination of vectors
**    dest = v1 + mult*v2
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------*/
/*   BOP   */
/*
   *************   ROUTINE:   vouter   ***************
*/

/*  INTERFACE:   */

         vouter(
                real   *v1 , /* first array */
                real   *v2 , /* second array */
                real   *dest , /* destination array */
                int    neq       /* number of entries */
               )

/*  RETURN VALUE:  -none- */
/*  DESCRIPTION:   */
/*
** Routine vouter
** calculates the outer product of two vectors
** actually a misnomer -- it's really the element-by-element product
**/
/*   EOP   */
/*-------------------------------------------------------------------------*/
```