fll – Fortran Linked List Library

version - 2.1

User Guide

Adam Jirasek

Introduction

- Available at gihub.com/libm3l/fll
- LGPL OSS license

- Multi level, doubly linked list
- Fortran language
- Most of functions similar names to Unix/Linux

- MPI functionality

Introduction

- List consists of nodes
  - type of directory “DIR” or “N”
    - Contains other nodes type of directory or file
  - Type of file (R,D,I,L,S etc….)
    - R – real number
    - D - double real number
    - I – integer
    - L – long integer
    - S – fixed length string

Introduction

- Example

```
Main_List DIR 3
   Subdir DIR 2
      pressure D 5  1
          1 2 3 4 5
       density D 1 5
           3 4 5 6 7
     Subdir   DIR 1
         volumes D 5 1
           1.5 2.5 3.5 4.5 5.5
     Index   L 5 2
        3 5 7 9 10 4 5 6 7 8
```

Introduction

- Above list starts with MainDir which contains three data sets
  - Two subdirectories
  - And one data set

  - The first subdirectory contains two data sets
    - **Pressure**, type double, array is 1D and, length is five and contains values 1 2 3 4 5
    - **Density**, type double, 1D array, length 5, contains 3 4 5 6 7
    - NOTE: both arrays are 1D and will be stored in 1D array even though the index of the second suggest the array has 5 columns
  - The second subdirectory contains one data set
    - **Volumes**, 1D array, type long integer, length 5, contains values 1.5 2.5 3.5 4.5 5.5
  - The third data se it a 2D array of long integers

Introduction

- Available functions
  - **fll_mv** - move node
  - **fll_cp** - copy node
  - **fll_mklist** - make node
  - **fll_locate** - locate node
  - **fll_nnodes** – get number of nodes
  - **fll_getndata** – get data of node
  - **fll_getnbytes** – get size of dataset
  - **fll_rm** - remove node
  - **fll_cat** - print node
  - **fll_read** - read list from a file
  - **fll_write** - write list to a file
  - **fll_read_ffa** - read list from FFA format file
  - **fll_write_ffa** - write list to FFA format file
  - **fll_deattach** – detaches node from list

Introduction

- Available functions
  - **fll_read_record** - reads record from fll file instead of reading entire file
  - **fll_scan_file** - reads fll file in fast mode – scans for data sets names and types

  - Each function or subroutine has fpar

Introduction

- Available MPI functions
  - **fll_mpi_cp_all** - copies fll list from one process to all processes
  - **fll_mpi_cp** - copies fll list from one process to another
  - **fll_mpi_mv** - moves fll list from one process to another

Function fll_cp()

- **fll_cp(pwhat, pwhere, fpar)**
  - Copies pwhat node to pwhere
    - If pwhere = NULL(), the function duplicates pwhat node

  - Return value – pointer to a new copy, if failed, returns NULL

Function fll_mv()

- **fll_mv(pwhat, pwhere, fpar)**
  - Moves pwhat node to pwhere
  - Return value - logical value, return value can be true or false depending on if the move operation was successful

  - Returns TRUE if function was successful or FALSE if function failed

Function fll_mk()

- **fll_mk(name,type,ndim,nsize,fpar)**
  - Makes a new node of list
  - Input – name of node, type of node, first and second dimensions
    - If type of node is DIR, ndim and nsize are automatically set to 0
  - Return - pointer to newly created node

Function fll_mk()

- Note on fll_mk
  - fll_mk will decide what to allocate based on ndim and nsize
  - If ndim = 1 **and** nsize = 1   the access variable is static and accessed with index 0
    - Example tmp%L0
  - If ndim = 1 **or** nsize = 1   the acces variable is one dimensional array accessed with index1
    - Example tmp%L1
  - If ndim > 1 **and** nsize > 1   the acces variable is two dimensional array accessed with index 2
    - Example tmp%L2

Function fll_mk()

- If user wants to force the shape of array, user must specify the index during fll_mk
  - Example

  Ptmp => fll_mk('node_name', 'L2' , ndim, nsize, fpar)

  - This will allocate two dimensional array accessed by tmp%l2 independent of values of ndim and nsize
  - The arrays are then located by using function fll_getndata with specified type and additional parameter force_type = 'y'
  - Example: the array above would be accessed

    Array => fll_getndata_l2(pointer, 'node_name', 1_lint, fpar, force_type='y')

Function fll_locate()

- **fll_locate (pnode,name, type,dim, number,recursive,fpar)**
  - Locates node
  - Input parameters
    - Pnode – list where to search
    - Name – name of node
    - Number – order of the node (1st, 2nd etc…) if more nodes of the same name
    - Type – type of node
    - Dim – dimensions of arrays in the node, can be 0,1,2, if any other number the dimensions is not considered
    - Recursive – search list recursively, if so, number == 1
    - Both name and type can be set to *
  - Return – pointer to located node

Function fll_nnodes()

- **fll_nnodes(pnode,name,type,dim, number,recursive,fpar)**
  - Return number of nodes pnode list
  - Input parameters
    - Pnode – list where to search
    - Name – name of node
    - Number – order of the node (1st, 2nd etc…) if more nodes of the same name
    - Type – type of node
    - Dim – dimensions of arrays in the node, can be 0,1,2, if any other number the dimensions is not considered
    - Recursive – search list recursively, if so, number == 1
    - Both name and type can be set to *
  - Return – number of nodes

Function fll_getndata()

- **fll_getndata(pnode,name, number,recursive,fpar,force_type)**
  - Returns data in nodes which are not type of DIR
  - Input parameters
    - Pnode – list where to search
    - Name – name of node
    - Number – order of the node (1st, 2nd etc…) if more nodes of the same name
    - Dim – dimensions of arrays in the node, can be 0,1,2, if any other number the dimensions is not considered
    - Recursive – search list recursively, if so, number == 1
    - Both name and type can be set to *
    - If force_type specified as ‘y’ array dimensions will be forced by index in name of function – see fll_mk()
  - Return – pointer to the data

Function fll_getndata()

- Functions are
  - Real numbers
    - **fll_getndata_r0**
    - **fll_getndata_r1**
    - **fll_getndata_r2**
  - Double numbers
    - **fll_getndata_d0**
    - **fll_getndata_d1**
    - **fll_getndata_d2**
  - Strings
    - **fll_getndata_s0**
    - **fll_getndata_s1**
    - **fll_getndata_s2**

Subroutine fll_rm()

- **fll_getndata(pnode,fpar)**
  - Removes data
  - Input parameters
    - Pnode – list to be removed

Subroutine fll_cat()

- **fll_cat(pnode,iounit,parent,fpar)**
  - Prints data to iounit
  - Input parameters
    - Pnode – list to be printed
    - Iounit – number of file descriptor
    - Parent – if TRUE write information about node’s parent

Function fll_deattach()

- **fll_deattach(pnode,fpar)**
  - Detaches PNODE from list
    - After being detached from list, the node parent and siblings are NULL
    - The node is removed from the original list
    - The function is an opposite to fll_mv() function
  - Input parameters
    - Pnode – list to be printed
    - Parent – if TRUE write information about node's parent

Subroutine fll_write()

- **fll_write(pnode,file,iounit,fmt,fpar)**
  - Write data to FLL native format file
  - Input parameters
    - Pnode – list to be printed
    - File – name of file
    - Iounit - number of file descriptor
    - Fmt – A- asci file, B – binary file

Function fll_read()

- **fll_read(pnode,file,iounit,fmt,fpar)**
  - Read data from FLL native format file
  - Input parameters
    - Pnode – list to be printed
    - File – name of file
    - Iounit – number of file descriptor
    - Fmt – A- asci file, B – binary file

    - Returns pointer to imported fll list

Subroutine fll_write()

- **fll_write_ffa(pnode,file,iounit,fmt,fpar)**
  - Write data to FFA format file
  - Input parameters
    - Pnode – list to be printed
    - File – name of file
    - Iounit - number of file descriptor
    - Fmt – A- asci file, B – binary file

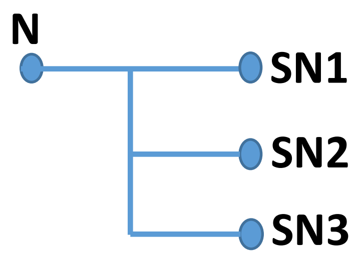
Function fll_read_ffa()

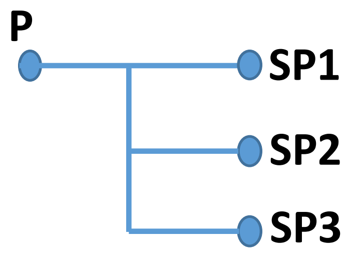
- **fll_read_ffa(pnode,file,iounit,fmt,fpar)**
  - Read data from FFA format file
  - Input parameters
    - Pnode – list to be printed
    - File – name of file
    - Iounit – number of file descriptor
    - Fmt – A- asci file, B – binary file

    - Returns pointer to imported fll list

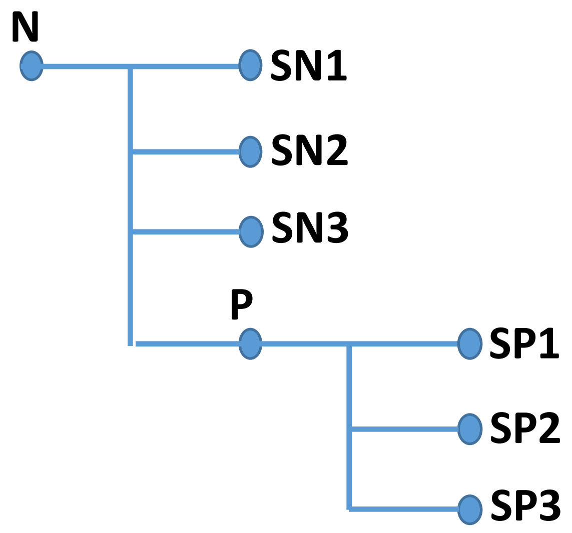
Moving, copying nodes details

- **N** node is a DIR type of node, **SN1, SN2, SN3** are data type of nodes

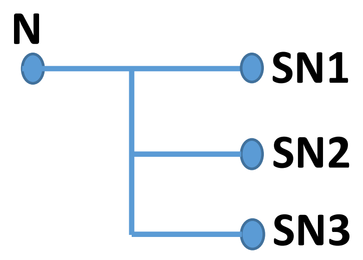


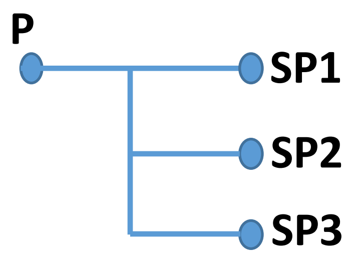


1. **fll_mv(P,N,fpar)** will result in node **P** being moved into node **N** as a new subset

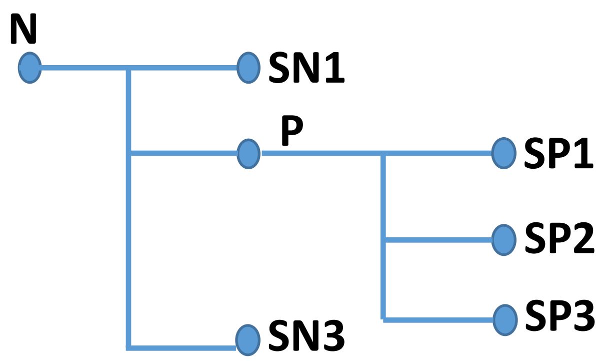


Moving, copying nodes details





1. **fll_mv(P,SN2,fpar)** will result in node **SN2** being overwritten by node **P**, original node **SN2** and its data will be removed



Subroutine fll_mpi_cp_all()

- **fll_mpi_cp_all(pnode,communicator,sendpart,fpar)**
  - Copies (broadcasts) fll list from sendpart to all other partitions in communicator
  - Input parameters
    - Pnode – list to be printed
    - Communicator – MPI communicator
    - Sendpart – rank of sending partition
    - Fpar – fuction specific structure

    - Returns pointer to pnode fll list on sendpart process and pointer to new copy on all other processes

Subroutine fll_mpi_cp ()

- **fll_mpi_cp_all(pnode,communicator,sendpart,recpart,fpar)**
  - Copies fll list from sendpart to recpart
  - Input parameters
    - Pnode – list to be printed
    - Communicator – MPI communicator
    - Sendpart – rank of sending partition
    - Recpart – rank of sending partition
    - Fpar – fuction specific structure

    - Returns pointer to pnode fll list on sendpart process and pointer to new copy on recpart process

Subroutine fll_mpi_mv ()

- **fll_mpi_cp_all(pnode,communicator,sendpart,recpart,fpar)**
  - Moves fll list from sendpart to recpart
  - Input parameters
    - Pnode – list to be printed
    - Communicator – MPI communicator
    - Sendpart – rank of sending partition
    - Recpart – rank of sending partition
    - Fpar – fuction specific structure

    - Returns NULL on sendpart process and pointer to new copy on recpart process

Function fll_getnbytes()

- **fll_getnbytes(pnode,fpar)**
  - Returns size of pnode data set in bytes
  - Input parameters
    - Pnode – pointer to data set

Function fll_scan_file()

- **fll_scan_file(filename, iounit, fmt,fpar)**
  - Returns fll list of with data sets contained in file
  - Input parameters
    - Filename – name of file
    - Iounit – number of IO file descriptior
    - Fmt - file format ‘A’ or ‘B’
    - Fpar – function parameter pointer

    - Function used in connection with fll_read_record()

Function fll_read_record()

- **fll_read_record(filename, iounit, pnode,name, type,dim, number,recursive,fpar)**
  - Returns data set from file
  - Input parameters
    - Filename – name of file
    - Iounit – number of IO file descriptior
    - Pnode – data structure of the file, obtained from fll_scan_file()
        - If NULL, function reads the file and crates this list itself
        - If not NULL, function uses this list instead of scanning file
    - Name – name of node
    - Number – order of the node (1st, 2nd etc…) if more nodes of the same name
    - Type – type of node
    - Dim – dimensions of arrays in the node, can be 0,1,2, if any other number the dimensions is not considered
    - Recursive – search list recursively, if so, number == 1
    - Both name and type can be set to *
    - Fpar – function parameter pointer

MPI I/O functions

- Available functions
  - **fll_mpi_cp** - copy data set from one process to another
  - **fll_mpi_cp_all** - copy data set from one partition to all partitions
  - **fll_mpi_mv** - move data set from one process to another
  - **fll_mpi_read** - read file in MPI mode
  - **fll_mpi_write** - write file in MPI node (N-1) model
  - **fll_mpi_write_nm** - write file in MPI node (N-M) model
  - **fll_mpi_write_snm** - write file in MPI node (S-N-M) model
  - **fll_mpi_proc_struct** - creates structures needed for MPI IO
  - **fll_nmio_struct** - creates structures needed for MPI IO
  - **fll_snmio_struct** - creates structures needed for MPI IO

Function fll_mpi_cp ()

- **fll_mpi_cp(pnode,communicator, sendpart,recpart,fpar)**
  - Copy data set from sendpart to recpart
  - Returns pointer to copied data set on recpart processor
  - Input parameters
    - Pnode – pointer to data set
    - Communicator – MPI communicator
    - Sendpart – world number of sending process
    - Recpart – world number of receiving process

Function fll_mpi_cp_all ()

- **fll_mpi_cp_all(pnode,communicator, sendpart,fpar)**
  - Copy data set from sendpart process to all other processes
  - Returns pointer to copied data set in receiving processes
  - Input parameters
    - Pnode – pointer to data set
    - Communicator – MPI communicator
    - Sendpart – world number of sending process

Function fll_mpi_mv ()

- **fll_mpi_mv(pnode,communicator, sendpart,recpart,fpar)**
  - Moves data set from sendpart to recpart
  - Returns pointer to copied data set on recpart processor
  - Input parameters
    - Pnode – pointer to data set
    - Communicator – MPI communicator
    - Sendpart – world number of sending process
    - Recpart – world number of receiving process

Function fll_mpi_sum ()

- **fll_mpi_sum(communicator, ndim, Array, fpar)**
  - Mpi_sum function
  - Input parameters
    - Communicator – MPI communicator
    - Dimension of Array, if 1 Array is scalar
    - Array is array where to mpi_sum data
      - It is an optional parameter
      - Call as:

L1 = Array	- long integer 1D array
I1 = Array	- integer 1D array
D1 = Array	- double 1D array
R1 = Array	- real 1D array
L = Array	- long integer scalar
I = Array	- integer scalar
D = Array	- double scalar
R = Array	- real scalar