

# fll – Fortran Linked List Library

version - 2.1

## User Guide

*Adam Jirasek*

# Introduction

- Available at [github.com/libm3l/fll](https://github.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 set is 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 fl\_cp()

- **fl\_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 fl\_mv()

- **fl\_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 fl\_mk()

- **fl\_mk(name,type,ndim,nsiz,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 nsiz are automatically set to 0
- Return - pointer to newly created node

# Function fl\_locate()

- **fl\_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 (1<sup>st</sup>, 2<sup>nd</sup> 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 fl\_nnodes()

- **fl\_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 (1<sup>st</sup>, 2<sup>nd</sup> 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 fl\_getndata()

- **fl\_getndata(pnode,name, number,recursive,fpar)**
  - 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 (1<sup>st</sup>, 2<sup>nd</sup> 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 \*
  - Return – pointer to the data

# Function fl\_getndata()

- Functions are
  - Real numbers
    - fl\_getndata\_r0
    - fl\_getndata\_r1
    - fl\_getndata\_r2
  - Double numbers
    - fl\_getndata\_d0
    - fl\_getndata\_d1
    - fl\_getndata\_d2
  - Strings
    - fl\_getndata\_s0
    - fl\_getndata\_s1
    - fl\_getndata\_s2

# Subroutine fl\_rm()

- **fl\_getndata(pnode,fpar)**

- Removes data
- Input parameters
  - Pnode – list to be removed



# Subroutine fl\_cat()

- **fl\_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 fl\_deattach()

- **fl\_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 fl\_mv() function
  - Input parameters
    - Pnode – list to be printed
    - Parent – if TRUE write information about node's parent

# Subroutine fl\_write()

- **fl\_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 fl\_read()

- **fl\_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- ascii file, B – binary file
  - Returns pointer to imported fl list

# Subroutine fl\_write()

- **fl\_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 fl\_read\_ffa()

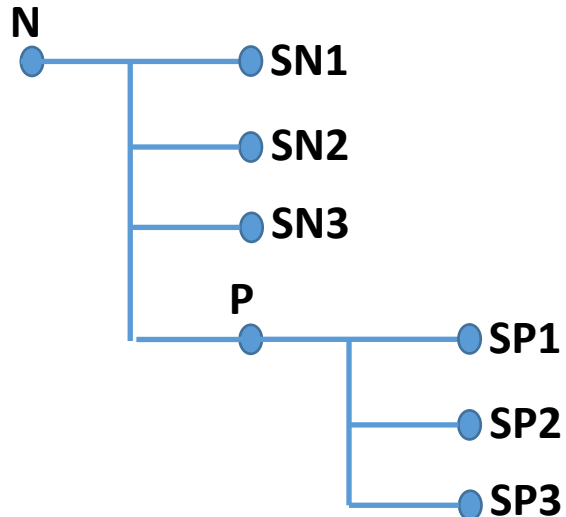
- **fl\_read\_ffa(pnode,file,iounit,fmt,fpar)**
  - Read data from FFA format file
  - Input parameters
    - Pnode – list to be printed
    - File – name of file
    - Iounit – number of file descriptor
    - Fmt – A- ascii file, B – binary file
  - Returns pointer to imported fl list

# Moving, copying nodes details

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



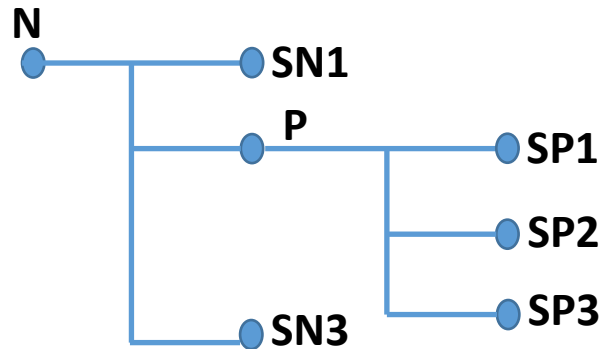
1. **fll\_mv(P,N,fpar)** will result in node **P** being moved into node **N** as a new subset



# 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 fl\_mpi\_cp\_all()

- **fl\_mpi\_cp\_all(pnode,communicator,sendpart,fpar)**
  - Copies (broadcasts) fl 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 fl list on sendpart process and pointer to new copy on all other processes

# Subroutine fl\_mpi\_cp ()

- **fl\_mpi\_cp\_all(pnode,communicator,sendpart,recpart,fpar)**
  - Copies fl list from sendpart to recpart
  - Input parameters
    - Pnode – list to be printed
    - Communicator – MPI communicator
    - Sendpart – rank of sending partition
    - Recpart – rank of receiving partition
    - Fpar – function specific structure
  - Returns pointer to pnode fl list on sendpart process and pointer to new copy on recpart process

# Subroutine fl\_mpi\_mv ()

- **fl\_mpi\_cp\_all(pnode,communicator,sendpart,recpart,fpar)**
  - Moves fl list from sendpart to recpart
  - Input parameters
    - Pnode – list to be printed
    - Communicator – MPI communicator
    - Sendpart – rank of sending partition
    - Recpart – rank of receiving partition
    - Fpar – function specific structure
  - Returns NULL on sendpart process and pointer to new copy on recpart process

# Function fl\_getnbytes()

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

# Function fl\_scan\_file()

- **fl\_scan\_file(filename, iounit, fmt, fpar)**
  - Returns fl list of with data sets contained in file
  - Input parameters
    - Filename – name of file
    - Iounit – number of IO file descriptor
    - Fmt – file format 'A' or 'B'
    - Fpar – function parameter pointer
  - Function used in connection with fl\_read\_record()

# Function fl\_read\_record()

- **fl\_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 descriptor
    - Pnode – data structure of the file, obtained from fl\_scan\_file()
      - If NULL, function reads the file and creates this list itself
      - If not NULL, function uses this list instead of scanning file
    - Name – name of node
    - Number – order of the node (1<sup>st</sup>, 2<sup>nd</sup> 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 fl\_mpi\_cp ()

- **fl\_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 fl\_mpi\_cp\_all ()

- **fl\_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 fl\_mpi\_mv ()

- **fl\_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 fl\_mpi\_sum ()

- **fl\_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