

LCC

1.0.0.

Generated by Doxygen 1.8.17

1 LCC DOCUMENTATION	1
2 LCC	3
3 Input file choices	7
4 LCC	11
5 LCC DOCUMENTATION	15
6 Building regular shapes	17
7 Todo List	19
8 Namespace Index	21
8.1 Namespace List	21
9 Class Index	23
9.1 Class List	23
10 Namespace Documentation	25
10.1 lcc_allocation_mod Module Reference	25
10.1.1 Detailed Description	25
10.1.2 Function/Subroutine Documentation	25
10.1.2.1 lcc_reallocate_char2vect()	25
10.1.2.2 lcc_reallocate_char3vect()	26
10.1.2.3 lcc_reallocate_intmat()	26
10.1.2.4 lcc_reallocate_intvect()	26
10.1.2.5 lcc_reallocate_realmat()	27
10.1.2.6 lcc_reallocate_realvect()	27
10.2 lcc_aux_mod Module Reference	27
10.2.1 Detailed Description	28
10.2.2 Function/Subroutine Documentation	28
10.2.2.1 inv()	28
10.2.2.2 lcc_canonical_basis()	28
10.2.2.3 lcc_center_at_box()	29
10.2.2.4 lcc_center_at_origin()	29
10.2.2.5 lcc_get_coordination()	30
10.2.2.6 lcc_get_reticular_density()	30
10.2.2.7 lcc_parameters_to_vectors()	30
10.2.2.8 lcc_vectors_to_parameters()	31
10.3 lcc_build_mod Module Reference	31
10.3.1 Detailed Description	31
10.3.2 Function/Subroutine Documentation	32
10.3.2.1 lcc_add_randomness_to_coordinates()	32
10.3.2.2 lcc_bravais_growth()	32

10.3.2.3 lcc_build_slab()	33
10.3.2.4 lcc_plane_cut()	33
10.4 lcc_check_mod Module Reference	34
10.4.1 Detailed Description	34
10.4.2 Function/Subroutine Documentation	34
10.4.2.1 lcc_check_periodicity()	34
10.5 lcc_constants_mod Module Reference	34
10.5.1 Detailed Description	35
10.6 lcc_lattice_mod Module Reference	35
10.6.1 Detailed Description	35
10.6.2 Function/Subroutine Documentation	35
10.6.2.1 lcc_add_base_to_cluster()	36
10.6.2.2 lcc_add_randomness()	36
10.6.2.3 lcc_check_basis()	36
10.6.2.4 lcc_fcc()	37
10.6.2.5 lcc_make_lattice()	37
10.6.2.6 lcc_minimize_from()	38
10.6.2.7 lcc_read_base()	38
10.6.2.8 lcc_sc()	38
10.6.2.9 lcc_set_atom_type()	39
10.6.2.10 lcc_triclinic()	39
10.7 lcc_lib Module Reference	40
10.7.1 Detailed Description	40
10.8 lcc_mc_mod Module Reference	40
10.8.1 Detailed Description	41
10.9 lcc_message_mod Module Reference	41
10.9.1 Detailed Description	41
10.9.2 Function/Subroutine Documentation	41
10.9.2.1 lcc_print_error()	41
10.9.2.2 lcc_print_intval()	42
10.9.2.3 lcc_print_message()	42
10.9.2.4 lcc_print_realmat()	42
10.9.2.5 lcc_print_realval()	43
10.9.2.6 lcc_print_realvect()	43
10.9.2.7 lcc_print_warning()	43
10.10 lcc_parser_mod Module Reference	44
10.10.1 Detailed Description	44
10.10.2 Function/Subroutine Documentation	44
10.10.2.1 lcc_parse()	44
10.10.2.2 lcc_write_coords()	45
10.11 lcc_regular_mod Module Reference	45
10.11.1 Detailed Description	45

10.11.2 Function/Subroutine Documentation	45
10.11.2.1 lcc_spheroid()	46
10.12 lcc_string_mod Module Reference	46
10.12.1 Detailed Description	46
10.12.2 Function/Subroutine Documentation	46
10.12.2.1 lcc_get_word()	46
10.12.2.2 lcc_split_string()	47
10.13 lcc_structs_mod Module Reference	47
10.13.1 Detailed Description	47
11 Class Documentation	49
11.1 lcc_structs_mod::build_type Type Reference	49
11.1.1 Detailed Description	51
11.2 lcc_structs_mod::lattice_type Type Reference	51
11.2.1 Detailed Description	52
Index	53

Chapter 1

LCC DOCUMENTATION

The folder (`src/docs`) contains all the documentation relevant to both users and developers.

Prerequisites

- **pdf`latex`**
Latex GNU compiler. pdfTeX is an extension of TeX which can produce PDF directly from TeX source, as well as original DVI files. pdfTeX incorporates the e-TeX extensions.
- **doxygen**
Doxygen is a documentation system for C++, C, Java, Objective-C, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D.
- **sphinx**
Sphinx is a documentation generator or a tool that translates a set of plain text source files into various output formats, automatically producing cross-references, indices, etc. That is, if you have a directory containing a bunch of reStructuredText or Markdown documents, Sphinx can generate a series of HTML files, a PDF file (via LaTeX), man pages and much more.
- Any pdf viewer.
- Any web browser.

These programs can be installed as follows:

```
sudo apt-get install pdflatex
sudo apt-get install doxygen
sudo apt-get install dot2tex
sudo apt-get install dot2tex
sudo apt-get install python3-sphinx
pip3 install PSphinxTheme
pip3 install recommonmark
```

Build the full documentation

This will build all three types of docs (Sphinx, Doxygen, and latex)

```
make
```

The documentation that is build with Sphinx can be tested as follows:

```
firefox ccl.html
```

The file can be explored using any web browser.

One can also build any of the documentations separatly. For example, to build the Sphinx documentation, we can do:

```
make sphinx
```

Documenting

In order to add a documentation using Sphinx follow these steps: 1) make a file with a proper name under `./sphinx-src/source/`. For example: `MYPAGE.md`. 2) Add the documentation inside the file using "mark-down" syntax. 3) Modify the file in `./sphinx-src/source/index.txt` to include the documentation just as shown in the following example:

```
.. toctree::
    :maxdepth: 2
    :caption: Contents:
    ./README-main
    ./README
    ./MYPAGE
```

After modifying this file, recompile Sphinx by typing `make sphinx`.

Chapter 2

LCC

About

Los Alamos Crystal Cut (LCC) is simple crystal builder. It is an easy-to-use and easy-to-develop code to make crystal solid/shape and slabs from a crystal lattice. Provided you have a '.pdb' file containing your lattice basis you can create a solid or slab from command line. The core developer of this code is Christian Negre (cnegre@lanl.gov).

License

© 2022. Triad National Security, LLC. All rights reserved. This program was produced under U.S. Government contract 89233218CNA000001 for Los Alamos National Laboratory (LANL), which is operated by Triad National Security, LLC for the U.S. Department of Energy/National Nuclear Security Administration. All rights in the program are reserved by Triad National Security, LLC, and the U.S. Department of Energy/National Nuclear Security Administration. The Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this material to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

This program is open source under the BSD-3 License.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Requirements

In order to follow this tutorial, we will assume that the reader have a `LINUX` or `MAC` operative system with the following packages properly installed:

- The `git` program for cloning the codes.
- A `C/C++` compiler (`gcc` and `g++` for example)
- A `Fortran` compiler (`gfortran` for example)
- The `LAPACK` and `BLAS` libraries (GNU `libblas` and `liblapack` for example)
- The `python` interpreter (not essential).
- The `pkgconfig` and `cmake` programs (not essential).

On an `x86_64` GNU/Linux Ubuntu 16.04 distribution the commands to be typed are the following:

```
$ sudo apt-get update
$ sudo apt-get --yes --force-yes install gfortran gcc g++
$ sudo apt-get --yes --force-yes install libblas-dev liblapack-dev
$ sudo apt-get --yes --force-yes install cmake pkg-config cmake-data
$ sudo apt-get --yes --force-yes install git python
```

NOTE: Through the course of this tutorial we will assume that the follower will work and install the programs in the home directory (`$HOME`).

Download and installation

We will need to clone the repository as follows:

```
$ cd; git@github.com:cnegre/ClusterGen.git
```

Compiling PROGRESS and BML libraries

The LCC code needs to be compiled with both `PROGRESS` and `BML` libraries. In this section we will explain how to install both of these libraries and link the code against them.

Scripts for quick installations can be found in the main folder. In principle one should be able to install everything by typing:

```
$ ./clone_libs.sh
$ ./build_bml.sh
$ ./build_progress.sh
$ ./build.sh
```

Which will also build LCC with its binary file in `./src/lcc_main`.

Step-by-step install

Clone the BML library (in your home directory) by doing^[^1]:

```
$ cd
$ git clone git@github.com:lanl/bml.git
```

Take a look at the `./scripts/example_build.sh` file which has a set of instructions for configuring. Configure the installation by copying the script into the main folder and run it:

```
$ cp ./scripts/example_build.sh .
$ sh example_build.sh
```

The `build.sh` script is called and the installation is configured by creating the `build` directory. Go into the build directory and type:

```
$ cd build
$ make -j
$ make install
```

To ensure bml is installed correctly type `$ make tests` or `$ make test ARGS="-V"` to see details of the output. Series of tests results should follow.

After BML is installed, return to your home folder and “clone” the PROGRESS repository. To do this type:

```
$ cd
$ git clone git@github.com:lanl/progress.git
```

Once the folder is cloned, cd into that folder and use the `example_build.sh` file to configure the installation by following the same steps as for the bml library.

```
$ sh example_build.sh
$ cd build
$ make; make install
```

You can test the installation by typing `$ make tests` in the same way as it is done for BML.

LCC

Open the `Makefile` file in the `lcc/src` folder make sure the path to both bml and progress libs are set correctly. NOTE: Sometimes, depending on the architecture the libraries are installed in `/lib64` instead of `/lib`. After the aforementioned changes are done to the `Makefile` file proceed compiling with the “make” command.

Contributors

Christian Negre, email: cnegre@lanl.gov

Andrew Alvarado, email: aalvarado@lanl.gov

^[^1]: In order to have access to the repository you should have a github account and make sure to add your public ssh key is added in the configuration windows of github account.

Contributing

Formally request to be added as a collaborator to the project by sending an email to cnegre@lanl.gov. After being added to the project do the followig:

- Create a new branch with a proper name that can identify the new feature (`git checkout -b "my_new_branch"`)
- Make the changes or add your contributions to the new branch (`git add newFile.F90 modifiedFile.F90`)
- Make sure the tests are passing (`cd tests ; ./run_test.sh`)
- Commit the changes with proper commit messages (`git commit -m "Adding a my new contribution"`)
- Push the new branch to the repository (`git push`)
- Go to repository on the github website and click on "create pull request"

Chapter 3

Input file choices

In this section we will describe the input file keywords. Every valid keyword will use "cammel" syntax and will have and = sign right next to it. For example, the following is a valid keyword syntax `JobName= MyJob`. Comments need to have a # (hash) sign right next to the phrase we want to comment. Example comment could be something like: `#My comment`.

JobName=

This variable will indicate the name of the job we are running. It is just a tag to distinguish different outputs. As we mentioned before and example use could be: `JobName= MyJob`

Verbose=

Controls the verbosity level of the output. If set to 0 no output is printed out. If set to 1, only basic messages of the current execution point of the code will be printed. If set to 2, information about basic quantities are also printed. If set to 3, all relevant possible info is printed.

CoordsOutFile=

This will store the name of the output coordinates files. Basically if `CoordsOutFile= coords` two output files will be created: `coords.xyz` and `coords.pdb`.

PrintCml=

By setting `PrintCml= T` will also print create `coords.cml` which can be read by avogadro. In order to have this option working one needs to install **openbabel**. In order to read a cml file one needs to have **avogadro** installed. On gnu linux:

```
sudo apt-get install avogadro
sudo apt-get install openbabel
```

ClusterType=

This variable will define the type of shape/cluster/slab we want to construct. There are many options including Bulk, Planes, Bravais and Spheroid. We will explain all these in the following section.

ClusterType= Bulk

This will just cut a "piece of bulk" by indicating how many lattice point we want. For example, the following will create a bulk/lattice with 50 points on each a,b,c direction.

```
ClusterType= Bulk
LatticePoints= 50
```

The following, instead, will create a bulk/lattice with 100 lattice points in the x direction and 50 on the rest.

```
LatticePointsX1= 1
LatticePointsX2= 100
LatticePointsY1= 1
LatticePointsY2= 50
LatticePointsZ1= 1
LatticePointsZ2= 50
```

ClusterType= Spheroid

This will produce a "spheroid" center at the origin. And example follows:

```
ClusterType= Spherid
LatticePoints= 50 #This is necessary to construct the initial bulk
AAxis= 1.0 #Radius in direction x
BAxis= 2.0 #Radius in direction y
CAxis= 2.0 #Radius in direction z
```

See section REGULAR to see another example.

ClusterType= Planes

This will cut a shape using Miller indice. This is an important tool to construct a slab to study a surface. The cut does not guarantee periodicity. In order to have a periodic structure different plane boundaries need to be tried and the structures needs to be checked using a molecular visualizer. An example is given as follows:

```
NumberOfPlanes= 6
Planes[
  0 1 1 2.5
  0 -1 -1 1.5
  0 -1 1 4.5
  0 1 -1 3.5
  1 0 0 4.5
  -1 0 0 3.5
]
```

Three first number on each row indicate the Miller indices. The fourth number indicates how many Miller planes from the origin will be cut out. If the number of planes is 6, then the system tries to get the slab periodicity vectors since if the Miller planes are orthogonal to each other, the shape will be a "Parallelepiped". If instead the number different than 6, then the periodicity vectors are given by the "Boundaries" of the minimal box that contains the shape.

CenterAtBox=

If set to T, the shape will be centered at the box (the periodicity vectors of the shape/cluster)

Reorient=

If set to `T` this, will reorient the shape, such that vector "a" will be aligned with the x direction. This is important when making slabs needed to study a surface.

AtomType=

This will set the atom symbol if the lattice basis is not read from file.

TypeOfLattice=

This will set the Lattice unit cell. If set to `SC` or `FCC` either a simple cubic or face centered cubic lattice is built provided we set `LatticeConstanta=` to the lattice constant value. For general unit cell we can set `TypeOfLattice=` `Triclinic`, and provide the lattice parameters as in the following example:

```
LatticeConstanta= 6.5329400000000000
LatticeConstantb= 11.0221000000000000
LatticeConstantc= 7.3568800000000003
LatticeAngleAlpha= 90.0000000000000000
LatticeAngleBeta= 102.6520000000000000
LatticeAngleGamma= 90.0000000000000000
```

RandomSeed=

To generate random positions in the lattice. This will need to be used in conjunction with `RCoeff=` which controls the degree of deviation from the lattice positions.

PrimitiveFormat=

This will indicate if the lattice needs to be constructed out of a,b,c and angle parameter or primitive lattice vectors. If `PrimitiveFormat=` `Angles` (default), then the lattice parameters will need to be passed as in the following example:

```
LatticeConstanta= 6.5329400000000000
LatticeConstantb= 11.0221000000000000
LatticeConstantc= 7.3568800000000003
LatticeAngleAlpha= 90.0000000000000000
LatticeAngleBeta= 102.6520000000000000
LatticeAngleGamma= 90.0000000000000000
```

If instead, `PrimitiveFormat=` `Vectors` then the primitive vectors will need to be passed as in the following example:

```
LatticeVectors[
  2.0 0 0 #First lattice vector
  0.0 2.0 0
  0.0 2.0 2.0
]
```

UseLatticeBase=

This is an important tool that allows us to "dress" every lattice point with a basis of choice. The basis is defined to be the minimal set of coordinates and atom types needed to define a crystal system lattice point. The basis here will be read from file by providing the latticebase `LatticeBaseFile=` which will contain our atom types and coordinates. If `ReadLatticeFromFile=` is set to T, then, the lattice parameters will be read from the lattice basis file. If it is set to F, the the lattice parameters will need to be passed as explained before. Another important keyword is the `BaseFormat=`. If this is set to `abc`, then the basis coordinates stored in the file are assumed to be given in fractional coordinates of the lattice parameters. If it is set to `xyz`, then it will be assumed to be given in cartesian coordinates.

SymmetryOperations=

If the basis needs to be constricted from symmetry operation, then one needs to pass all these operation to the code as follows:

```
SymmetryOperations= T
NumberOfOperations= 4
Translations[
  0 0 0 0.0
  1 1 1 0.5
  1 1 0 1.0
  -0.5 1.5 0.5 1.0
]
Symmetries[
  0 0 0
  -1 1 -1
  -1 -1 -1
  1 -1 1
]
```

The first block indicates the "translations" within the unit cell. The first three rows indicating the directions of the translation and the fourth indicating the intensity. The second block indicates the symmetry of operations. For example, if an operation is indicated as $(-x + 1/2, -y, -z)$ then there will be a translation `0.5 0 0 1.0` and a symmetry `-1 0 0`.

```
NumberOfOperations= 0 MaxCoordination= 1 NumberOfIterations= 1 Truncation= 1.0000000000000000E+040
RCut= 20.000000000000000
RTol= 1.0000000000000000E-002 CutAfterAddingBase=F
SeedFile=seed.pdb
```


Chapter 4

LCC

About

Los Alamos Crystal Cut (LCC) is simple crystal builder. It is an easy-to-use and easy-to-develop code to make crystal solid/shape and slabs from a crystal lattice. Provided you have a '.pdb' file containing your lattice basis you can create a solid or slab from command line. The core developer of this code is Christian Negre (cnegre@lanl.gov).

License

© 2022. Triad National Security, LLC. All rights reserved. This program was produced under U.S. Government contract 89233218CNA000001 for Los Alamos National Laboratory (LANL), which is operated by Triad National Security, LLC for the U.S. Department of Energy/National Nuclear Security Administration. All rights in the program are reserved by Triad National Security, LLC, and the U.S. Department of Energy/National Nuclear Security Administration. The Government is granted for itself and others acting on its behalf a nonexclusive, paid-up, irrevocable worldwide license in this material to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so.

This program is open source under the BSD-3 License.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Requirements

In order to follow this tutorial, we will assume that the reader have a LINUX or MAC operative system with the following packages properly installed:

- The `git` program for cloning the codes.
- A C/C++ compiler (`gcc` and `g++` for example)
- A Fortran compiler (`gfortran` for example)
- The LAPACK and BLAS libraries (GNU `libblas` and `liblapack` for example)
- The `python` interpreter (not essential).
- The `pkgconfig` and `cmake` programs (not essential).

On an `x86_64` GNU/Linux Ubuntu 16.04 distribution the commands to be typed are the following:

```
$ sudo apt-get update
$ sudo apt-get --yes --force-yes install gfortran gcc g++
$ sudo apt-get --yes --force-yes install libblas-dev liblapack-dev
$ sudo apt-get --yes --force-yes install cmake pkg-config cmake-data
$ sudo apt-get --yes --force-yes install git python
```

NOTE: Through the course of this tutorial we will assume that the follower will work and install the programs in the home directory (`$HOME`).

Download and installation

We will need to clone the repository as follows:

```
$ cd; git@github.com:cnegre/ClusterGen.git
```

Compiling PROGRESS and BML libraries

The LCC code needs to be compiled with both `PROGRESS` and `BML` libraries. In this section we will explain how to install both of these libraries and link the code against them.

Scripts for quick installations can be found in the main folder. In principle one should be able to install everything by typing:

```
$ ./clone_libs.sh
$ ./build_bml.sh
$ ./build_progress.sh
$ ./build.sh
```

Which will also build LCC with its binary file in `./src/lcc_main`.

Step-by-step install

Clone the BML library (in your home directory) by doing^[^1]:

```
$ cd
$ git clone git@github.com:lanl/bml.git
```

Take a look at the `./scripts/example_build.sh` file which has a set of instructions for configuring. Configure the installation by copying the script into the main folder and run it:

```
$ cp ./scripts/example_build.sh .
$ sh example_build.sh
```

The `build.sh` script is called and the installation is configured by creating the `build` directory. Go into the build directory and type:

```
$ cd build
$ make -j
$ make install
```

To ensure bml is installed correctly type `$ make tests` or `$ make test ARGS="-V"` to see details of the output. Series of tests results should follow.

After BML is installed, return to your home folder and “clone” the PROGRESS repository. To do this type:

```
$ cd
$ git clone git@github.com:lanl/progress.git
```

Once the folder is cloned, cd into that folder and use the `example_build.sh` file to configure the installation by following the same steps as for the bml library.

```
$ sh example_build.sh
$ cd build
$ make; make install
```

You can test the installation by typing `$ make tests` in the same way as it is done for BML.

LCC

Open the `Makefile` file in the `lcc/src` folder make sure the path to both bml and progress libs are set correctly. NOTE: Sometimes, depending on the architecture the libraries are installed in `/lib64` instead of `/lib`. After the aforementioned changes are done to the `Makefile` file proceed compiling with the “make” command.

Contributors

Christian Negre, email: cnegre@lanl.gov

Andrew Alvarado, email: aalvarado@lanl.gov

^[^1]: In order to have access to the repository you should have a github account and make sure to add your public ssh key is added in the configuration windows of github account.

Contributing

Formally request to be added as a collaborator to the project by sending an email to cnegre@lanl.gov. After being added to the project do the followig:

- Create a new branch with a proper name that can identify the new feature (`git checkout -b "my_new_branch"`)
- Make the changes or add your contributions to the new branch (`git add newFile.F90 modifiedFile.F90`)
- Make sure the tests are passing (`cd tests ; ./run_test.sh`)
- Commit the changes with proper commit messages (`git commit -m "Adding a my new contribution"`)
- Push the new branch to the repository (`git push`)
- Go to repository on the github website and click on "create pull request"

Chapter 5

LCC DOCUMENTATION

The folder (`src/docs`) contains all the documentation relevant to both users and developers.

Prerequisites

- **pdf`latex`**
Latex GNU compiler. pdfTeX is an extension of TeX which can produce PDF directly from TeX source, as well as original DVI files. pdfTeX incorporates the e-TeX extensions.
- **doxygen**
Doxygen is a documentation system for C++, C, Java, Objective-C, IDL (Corba and Microsoft flavors) and to some extent PHP, C#, and D.
- **sphinx**
Sphinx is a documentation generator or a tool that translates a set of plain text source files into various output formats, automatically producing cross-references, indices, etc. That is, if you have a directory containing a bunch of reStructuredText or Markdown documents, Sphinx can generate a series of HTML files, a PDF file (via LaTeX), man pages and much more.
- Any pdf viewer.
- Any web browser.

These programs can be installed as follows:

```
sudo apt-get install pdflatex
sudo apt-get install doxygen
sudo apt-get install dot2tex
sudo apt-get install dot2tex
sudo apt-get install python3-sphinx
pip3 install PSphinxTheme
pip3 install recommonmark
```

Build the full documentation

This will build all three types of docs (Sphinx, Doxygen, and latex)

```
make
```

The documentation that is build with Sphinx can be tested as follows:

```
firefox ccl.html
```

The file can be explored using any web browser.

One can also build any of the documentations separatly. For example, to build the Sphinx documentation, we can do:

```
make sphinx
```

Documenting

In order to add a documentation using Sphinx follow these steps: 1) make a file with a proper name under `./sphinx-src/source/`. For example: `MYPAGE.md`. 2) Add the documentation inside the file using "mark-down" syntax. 3) Modify the file in `./sphinx-src/source/index.txt` to include the documentation just as shown in the following example:

```
.. toctree::
    :maxdepth: 2
    :caption: Contents:
    ./README-main
    ./README
    ./MYPAGE
```

After modifying this file, recompile Sphinx by typing `make sphinx`.

Chapter 6

Building regular shapes

One can also build regular shapes, such as for example a "spheroid." The parameters to do this can be entered as follows:

```
#Lcc input file.
LCC{
  JobName=                AgSpheroid          #Or any other name
  TypeOfLattice=          FCC
  LatticePoints=          50                  #Number of point in each direction
  LatticeConstanta=       4.08
  AtomType=               Ag
  ClusterType=            Spheroid
  AAxis=                  10                  #Radius in Ang for x direction
  BAxis=                  10                  #Radius in Ang for y direction
  CAxis=                  5                   #Radius in Ang for z direction
}
```

This will produce the following "oblate":

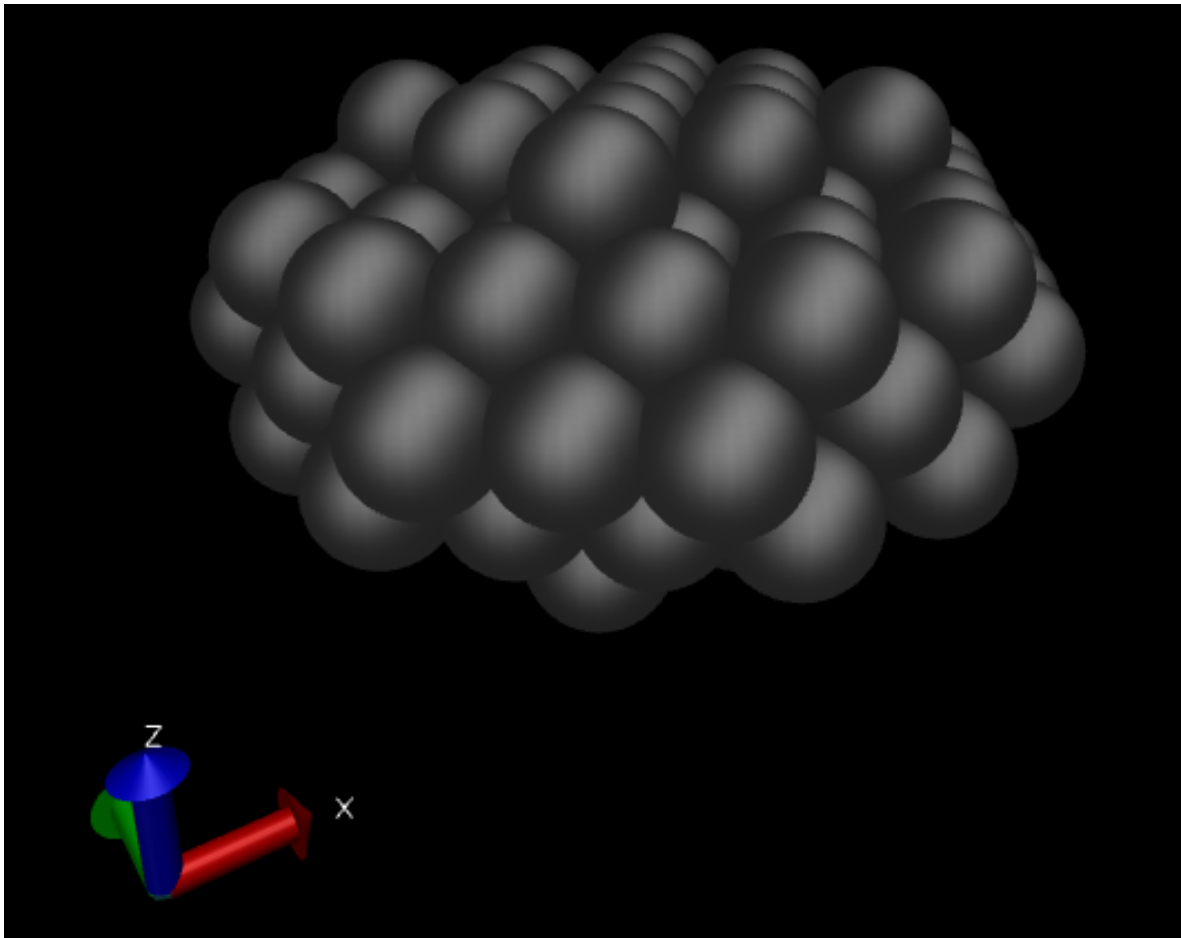


Figure 6.1 Oblate shape

Chapter 7

Todo List

Subprogram `lcc_build_mod::lcc_bravais_growth` (nCycles, dTol, dTo, tCoordination, seed_file, r_inout)

Optimize the routine.

Subprogram `lcc_lattice_mod::lcc_triclinic` (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, lattice_vectors, supra_lattice_vectors, r_sy, verbose)

A angles_to_vectors transformation will be available.

Chapter 8

Namespace Index

8.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

lcc_allocation_mod	Module for allocation operations	25
lcc_aux_mod	Module for auxiliary operations routines	27
lcc_build_mod	Module for generating the shapes after lattice is constructed	31
lcc_check_mod	Module for checking operations routines	34
lcc_constants_mod	A module to handle the constants needed by the code	34
lcc_lattice_mod	Module to hold routines for handling the lattice and lattice base	35
lcc_lib	Library module	40
lcc_mc_mod	Module for Monte Carlo related routines	40
lcc_message_mod	Module for printing through the code	41
lcc_parser_mod	This module controls the initialization of the variables	44
lcc_regular_mod	Module for generating regular shapes after lattice is constructed	45
lcc_string_mod	Module for manipulating strings	46
lcc_structs_mod	A module to handle the structures needed by the code	47

Chapter 9

Class Index

9.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

lcc_structs_mod::build_type	
Build type	49
lcc_structs_mod::lattice_type	
Lattice type to be read and extended	51

Chapter 10

Namespace Documentation

10.1 lcc_allocation_mod Module Reference

Module for allocation operations.

Functions/Subroutines

- subroutine, public [lcc_reallocate_realvect](#) (vect, ndim)
To reallocate a real vector.
- subroutine, public [lcc_reallocate_realmat](#) (mat, mdim, ndim)
To reallocate a real mxn matrix.
- subroutine, public [lcc_reallocate_intvect](#) (vect, ndim)
To reallocate a real vector.
- subroutine, public [lcc_reallocate_intmat](#) (mat, mdim, ndim)
To reallocate an integer mxn matrix.
- subroutine, public [lcc_reallocate_char2vect](#) (vect, ndim)
To reallocate a character vector.
- subroutine, public [lcc_reallocate_char3vect](#) (vect, ndim)
To reallocate a character vector.

10.1.1 Detailed Description

Module for allocation operations.

10.1.2 Function/Subroutine Documentation

10.1.2.1 lcc_reallocate_char2vect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_char2vect (  
    character(2), dimension(:), intent(inout), allocatable vect,  
    integer, intent(in) ndim )
```

To reallocate a character vector.

This will reallocate a character len=2 vector If it is already allocated, a deallocation will first happen.

Parameters

<i>vect</i>	Character(2) 1D array.
<i>ndim</i>	Dimension to reallocate the vector to.

10.1.2.2 lcc_reallocate_char3vect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_char3vect (
    character(3), dimension(:), intent(inout), allocatable vect,
    integer, intent(in) ndim )
```

To reallocate a character vector.

This will reallocate a character len=3 vector. If it is already allocated, a deallocation will first happen.

Parameters

<i>vect</i>	Character(3) 1D array.
<i>ndim</i>	Dimension to reallocate the vector to.

10.1.2.3 lcc_reallocate_intmat()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_intmat (
    integer, dimension(:,:), intent(inout), allocatable mat,
    integer, intent(in) mdim,
    integer, intent(in) ndim )
```

To reallocate an integer mxn matrix.

This will reallocate a matrix. If it is already allocated, a deallocation will first happen.

Parameters

<i>mat</i>	Integer 2D array.
<i>mnim</i>	First dimension to reallocate the matrix to.
<i>ndim</i>	Second dimension to reallocate the matrix to.

10.1.2.4 lcc_reallocate_intvect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_intvect (
    integer, dimension(:), intent(inout), allocatable vect,
    integer, intent(in) ndim )
```


To reallocate a real vector.

This will reallocate a vector If it is already allocated, a deallocation will first happen.

Parameters

<i>vect</i>	Integer 1D array.
<i>ndim</i>	Dimension to reallocate the vector to.

10.1.2.5 lcc_reallocate_realmat()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_realmat (
    real(dp), dimension(:, :), intent(inout), allocatable mat,
    integer, intent(in) mdim,
    integer, intent(in) ndim )
```

To reallocate a real mxn matrix.

This will reallocate a matrix If it is already allocated, a deallocation will first happen.

Parameters

<i>mat</i>	Real 2D array.
<i>mnim</i>	First dimension to reallocate the matrix to.
<i>ndim</i>	Second dimension to reallocate the matrix to.

10.1.2.6 lcc_reallocate_realvect()

```
subroutine, public lcc_allocation_mod::lcc_reallocate_realvect (
    real(dp), dimension(:), intent(inout), allocatable vect,
    integer, intent(in) ndim )
```

To reallocate a real vector.

This will reallocate a vector If it is already allocated, a deallocation will first happen.

Parameters

<i>vect</i>	Real 1D array.
<i>ndim</i>	Dimension to reallocate the vector to.

10.2 lcc_aux_mod Module Reference

Module for auxiliary operations routines.

Functions/Subroutines

- subroutine, public [lcc_vectors_to_parameters](#) (lattice_vector, abc_angles, verbose)
Transforms the lattice vectors into lattice parameters.
- subroutine, public [lcc_parameters_to_vectors](#) (abc_angles, lattice_vector, verbose)
Transforms the lattice parameters into lattice vectors.
- subroutine, public [lcc_get_coordination](#) (r_at, r_env, thresh, cnum)
Get the coordination of an atom.
- subroutine, public [lcc_canonical_basis](#) (lattice_vectors, r_inout, verbose)
To "canonical base" transformation.
- subroutine, public [lcc_center_at_box](#) (lattice_vectors, r_inout, verbose)
Cetering the system inside the lattice box.
- subroutine, public [lcc_center_at_origin](#) (r_inout, verbose)
Cetering the system at the origin.
- real(dp) function, dimension(:,:), allocatable [inv](#) (A)
Computes the inverse of a matrix using an LU decomposition.
- subroutine, public [lcc_get_reticular_density](#) (lattice_vectors, hkl_in, density)
Get the reticular density of a particular hkl face: This soubroutine computes:
- real(dp) function, dimension(:), allocatable **crossprod** (r1, r2)

10.2.1 Detailed Description

Module for auxiliary operations routines.

10.2.2 Function/Subroutine Documentation

10.2.2.1 [inv\(\)](#)

```
real(dp) function, dimension(:,:), allocatable lcc_aux_mod::inv (
    real(dp), dimension(:,:), intent(in) A )
```

Computes the inverse of a matrix using an LU decomposition.

Parameters

<i>A</i>	nxn Matrix to be inverted.
<i>Ainv</i>	Inverse of matrix A

10.2.2.2 [lcc_canonical_basis\(\)](#)

```
subroutine, public lcc_aux_mod::lcc_canonical_basis (
    real(dp), dimension(:,:), intent(inout), allocatable lattice_vectors,
```

```

real(dp), dimension(:,:), intent(inout), allocatable r_inout,
integer, intent(in) verbose )

```

To "canonical base" transformation.

This will reorient the shape/slab so that the first translation vector is aligned with x.

Parameters

<i>lattice_vectors</i>	Translation vectors for the shape/slab.
<i>r_inout</i>	Coordinates to be transformed.
<i>verbose</i>	Verbosity level.

10.2.2.3 lcc_center_at_box()

```

subroutine, public lcc_aux_mod::lcc_center_at_box (
    real(dp), dimension(:,:), intent(in), allocatable lattice_vectors,
    real(dp), dimension(:,:), intent(inout), allocatable r_inout,
    integer, intent(in) verbose )

```

Cetering the system inside the lattice box.

This will move the coordinates so that the geometric center of the system is at the center of the box.

Parameters

<i>lattice_vectors</i>	Translation vectors for the shape/slab.
<i>r_inout</i>	Coordinates to be transform.
<i>verbose</i>	Verbosity level.

10.2.2.4 lcc_center_at_origin()

```

subroutine, public lcc_aux_mod::lcc_center_at_origin (
    real(dp), dimension(:,:), allocatable r_inout,
    integer, intent(in) verbose )

```

Cetering the system at the origin.

This will move the coordinates so that the geometric center of the system is at (0,0,0).

Parameters

<i>r_inout</i>	Coordinates to be transform.
<i>verbose</i>	Verbosity level.

10.2.2.5 lcc_get_coordination()

```
subroutine, public lcc_aux_mod::lcc_get_coordination (
    real(dp), dimension(3), intent(in) r_at,
    real(dp), dimension(:, :), intent(in), allocatable r_env,
    real(dp), intent(in) thresh,
    integer, intent(inout) cnum )
```

Get the coordination of an atom.

Will count how many atoms are around a particular atom (coordination number) given a set radius.

Parameters

<i>r_at</i>	Coordinates of the atom for which we need the coordination.
<i>r_env</i>	Coordinated of the environment sorounding atom at <i>r_at</i> .
<i>thres</i>	Threshod distance to find coordinations.
<i>cnum</i>	Coordination number (output).

10.2.2.6 lcc_get_reticular_density()

```
subroutine, public lcc_aux_mod::lcc_get_reticular_density (
    real(dp), dimension(:, :), intent(in) lattice_vectors,
    real(dp), dimension(:), intent(in) hkl_in,
    real(dp), intent(out) density )
```

Get the reticular density of a particular hkl face: This soubroutine computes:

Parameters

<i>lattice_vectors</i>	Lattice vectors for the system.
<i>hkl_in</i>	Vector containing h, k, and l.
<i>density</i>	Reticular density.

10.2.2.7 lcc_parameters_to_vectors()

```
subroutine, public lcc_aux_mod::lcc_parameters_to_vectors (
    real(dp), dimension(2,3), intent(in) abc_angles,
    real(dp), dimension(3,3), intent(out) lattice_vector,
    integer, intent(in) verbose )
```

Transforms the lattice parameters into lattice vectors.

Parameters

<i>abc_angles</i>	2x3 array containing the lattice parameters. $abc_angles(1,1) = a$, $abc_angles(1,2) = b$, and $abc_angles(1,3) = c$ $abc_angles(2,1) = \alpha$, $abc_angles(2,2) = \beta$ and $abc_angles(2,3) = \gamma$
<i>lattice_vector</i>	3x3 array containing the lattice vectors. $lattice_vector(1,:) = \vec{a}$
<i>verbose</i>	Verbosity level.

10.2.2.8 lcc_vectors_to_parameters()

```

subroutine, public lcc_aux_mod::lcc_vectors_to_parameters (
    real(dp), dimension(3,3), intent(in) lattice_vector,
    real(dp), dimension(2,3), intent(out) abc_angles,
    integer, intent(in) verbose )

```

Transforms the lattice vectors into lattice parameters.

Parameters

<i>lattice_vector</i>	3x3 array containing the lattice vectors. $lattice_vector(1,:) = \vec{a}$
<i>abc_angles</i>	2x3 array containing the lattice parameters. $abc_angles(1,1) = a$, $abc_angles(1,2) = b$ and $abc_angles(1,3) = c$ $abc_angles(2,1) = \alpha$, $abc_angles(2,2) = \beta$, and $abc_angles(2,3) = \gamma$.
<i>verbose</i>	Verbosity level.

10.3 lcc_build_mod Module Reference

Module for generating the shapes after lattice is constructed.

Functions/Subroutines

- subroutine, public [lcc_bravais_growth](#) (nCycles, dTol, dTo, tCoordination, seed_file, r_inout)
For "growing" a crystal shape using Bravias type of growth teory.
- subroutine, public [lcc_plane_cut](#) (planes, ploads, interPlanarDistances, lattice_vectors, cluster_lattice_vectors, resindex, r_inout, verbose)
Cutting a shape based on Miller planes.
- subroutine [lcc_build_slab](#) (slab, loads, lattice_vectors, cluster_lattice_vectors, resindex, r_inout, verbose)
Cutting a shape based on PBC vectors.
- subroutine, public [lcc_add_randomness_to_coordinates](#) (r_inout, seed, rcoeff)
Will add randomness to the system.

10.3.1 Detailed Description

Module for generating the shapes after lattice is constructed.

10.3.2 Function/Subroutine Documentation

10.3.2.1 lcc_add_randomness_to_coordinates()

```
subroutine, public lcc_build_mod::lcc_add_randomness_to_coordinates (
    real(dp), dimension(:,:), intent(inout), allocatable r_inout,
    integer, intent(in) seed,
    real(dp), intent(in) rcoeff )
```

Will add randomness to the system.

Parameters

<i>r_inout</i>	System coordinates.
<i>lattice_vectors</i>	Lattice vectors.
<i>seed</i>	Random seed. rcoeff Coefficient for randomness.

10.3.2.2 lcc_bravais_growth()

```
subroutine, public lcc_build_mod::lcc_bravais_growth (
    integer, intent(in) nCycles,
    real(dp), intent(in) dTol,
    real(dp), intent(in) dTo,
    integer, intent(in) tCoordination,
    character(len=*), intent(in) seed_file,
    real(dp), dimension(:,:), intent(inout), allocatable r_inout )
```

For "growing" a crystal shape using Bravias type of growth teory.

Parameters

<i>nCycles</i>	Number of shells to add.
<i>dTol</i>	Tolerance for distinguishing the coordinates from the seed to the coodinates from the bulk.
<i>dTo</i>	Parameter to determine the coordination the incoming atom.
<i>tCoordination</i>	Target coordination. If coodination is larger than the target, the atom will be picked.
<i>seed_file</i>	Name of the file containing the seed.
<i>r_inout</i>	Input: Bulk lattice, Output: Crystal shape.

Todo Optimize the routine.

10.3.2.3 lcc_build_slab()

```
subroutine lcc_build_mod::lcc_build_slab (
    real(dp), dimension(:,:), intent(in), allocatable slab,
    real(dp), dimension(:), intent(in), allocatable sloads,
    real(dp), dimension(:,:), intent(inout), allocatable lattice_vectors,
    real(dp), dimension(:,:), intent(inout), allocatable cluster_lattice_vectors,
    integer, dimension(:), allocatable resindex,
    real(dp), dimension(:,:), intent(inout), allocatable r_inout,
    integer, intent(in) verbose )
```

Cutting a shape based on PBC vectors.

A set of PBC vectors and distances is provided.

Parameters

<i>planes</i>	List of planes to cut the shape with.
<i>ploads</i>	Distance from the origin to locate the plane.
<i>interPlanarDistance</i>	Use "interplanar distances" as measure for the cut.
<i>lattice_vectors</i>	Lattice vectors.
<i>cluster_lattice_vectors</i>	Lattice vectors of the shape. Note: this only makes sense if the planes make a parallelepiped.
<i>r_inout</i>	Coordinates in and out.
<i>verbose</i>	Verbosity level.

10.3.2.4 lcc_plane_cut()

```
subroutine, public lcc_build_mod::lcc_plane_cut (
    real(dp), dimension(:,:), intent(in), allocatable planes,
    real(dp), dimension(:), intent(in), allocatable ploads,
    logical, intent(in) interPlanarDistances,
    real(dp), dimension(:,:), intent(inout), allocatable lattice_vectors,
    real(dp), dimension(:,:), intent(inout), allocatable cluster_lattice_vectors,
    integer, dimension(:), allocatable resindex,
    real(dp), dimension(:,:), intent(inout), allocatable r_inout,
    integer, intent(in) verbose )
```

Cutting a shape based on Miller planes.

A set of panes and distances is provided.

Parameters

<i>planes</i>	List of planes to cut the shape with.
<i>ploads</i>	Distance from the origin to locate the plane.
<i>interPlanarDistance</i>	Use "interplanar distances" as measure for the cut.
<i>lattice_vectors</i>	Lattice vectors.
<i>cluster_lattice_vectors</i>	Lattice vectors of the shape. Note: this only makes sense if the planes make a parallelepiped.
<i>r_inout</i>	Coordinates in and out.
<i>verbose</i>	Verbosity level.

10.4 lcc_check_mod Module Reference

Module for checking operations routines.

Functions/Subroutines

- subroutine, public [lcc_check_periodicity](#) (*r_in*, *lattice_vectors*, *r_ref*, *tol*, *verbose*)
Check the periodicity.

10.4.1 Detailed Description

Module for checking operations routines.

10.4.2 Function/Subroutine Documentation

10.4.2.1 lcc_check_periodicity()

```
subroutine, public lcc_check_mod::lcc_check_periodicity (
    real(dp), dimension(:, :), intent(in), allocatable r_in,
    real(dp), dimension(:, :), intent(in), allocatable lattice_vectors,
    real(dp), dimension(:, :), intent(in), allocatable r_ref,
    real(dp), intent(in) tol,
    integer, intent(in) verbose )
```

Check the periodicity.

Will use a "brute force" approach to check periodidity.

Parameters

<i>r_in</i>	Input coordinates.
<i>lattice_vectors</i>	Translation vectors for the slab.
<i>r_ref</i>	Reference or "bulk structure from where the shape was cut.
<i>verbose</i>	Verbosity level.

10.5 lcc_constants_mod Module Reference

A module to handle the constants needed by the code.

Variables

- integer, parameter, public *dp* = kind(1.0d0)

Precision used throughout the code.

- `real(dp)`, parameter `pi` = 3.14159265358979323846264338327950_dp
Pi number.

10.5.1 Detailed Description

A module to handle the constants needed by the code.

This module will be used to store the constants needed in the code

10.6 lcc_lattice_mod Module Reference

Module to hold routines for handling the lattice and lattice base.

Functions/Subroutines

- subroutine, public `lcc_make_lattice` (bld, ltt, check, sy)
Make a lattice depending on the input parameter.
- subroutine `lcc_read_base` (bld, ltt, check, verbose)
Reading the basis from an input file.
- subroutine `lcc_check_basis` (base_format, r_base, lattice_vectors, verbose)
Routine to check for atom repetitions in basis \bnrief It will do all possible translations searching for atoms that could be repeated.
- subroutine `lcc_add_base_to_cluster` (ltt, sy)
Add a basis to the lattice.
- subroutine `lcc_sc` (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, h_lattice_a, supra_lattice_vectors, r_sy)
Simple cubic (SC) lattice construction.
- subroutine `lcc_fcc` (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, h_lattice_a, supra_lattice_vectors, r_sy, verbose)
Face center cubic (FCC) lattice construction.
- subroutine `lcc_triclinic` (Nx1, Nx2, Ny1, Ny2, Nz1, Nz2, lattice_vectors, supra_lattice_vectors, r_sy, verbose)
Triclinic lattice construction.
- subroutine, public `lcc_set_atom_type` (a_type, atom_symbol, atom_name, nats)
Sets the atom type.
- subroutine `lcc_add_randomness` (r_inout, lattice_vectors, seed, rcoeff)
Will add randomness to the system.
- subroutine `lcc_minimize_from` (xVar, i, ai, nats, trs, verbose)
To get the best translation that minimizes the distance to any previous fragment.

10.6.1 Detailed Description

Module to hold routines for handling the lattice and lattice base.

10.6.2 Function/Subroutine Documentation

10.6.2.1 lcc_add_base_to_cluster()

```
subroutine lcc_lattice_mod::lcc_add_base_to_cluster (
    type(lattice_type), intent(in) ltt,
    type(system_type), intent(inout) sy )
```

Add a basis to the lattice.

This routine will add the basis to the system points previously cut from the lattice. This is the last step of the solid/shape/slab creation.

Parameters

<i>ltt</i>	lattice_type See lcc_structs_mod
<i>sy</i>	system_type See progress library

10.6.2.2 lcc_add_randomness()

```
subroutine lcc_lattice_mod::lcc_add_randomness (
    real(dp), dimension(:, :), intent(inout), allocatable r_inout,
    real(dp), dimension(:, :), intent(in), allocatable lattice_vectors,
    integer, intent(in) seed,
    real(dp), intent(in) rcoeff )
```

Will add randomness to the system.

Parameters

<i>r_inout</i>	System coordinates.
<i>lattice_vectors</i>	Lattice vectors.
<i>seed</i>	Random seed. rcoeff Coefficient for randomness.

10.6.2.3 lcc_check_basis()

```
subroutine lcc_lattice_mod::lcc_check_basis (
    character(len=*), intent(in) base_format,
    real(dp), dimension(:, :), intent(in), allocatable r_base,
    real(dp), dimension(:, :), intent(in), allocatable lattice_vectors,
    integer, intent(in) verbose )
```

Routine to check for atom repetitions in basis \brief It will do all possible translations searching for atoms that could be repeated.

Parameters

<i>base_format</i>	Basis format, if xyz of abc
<i>r_base</i>	Coordinates of the basis. <i>r_base</i> (1,7) means coordinate x of atom 7
<i>lattice_vectors</i>	Lattice vectors. WARNING, in this case <i>lattice_vector</i> (1,3) means the coordinate 3=z of vector 1.

10.6.2.4 lcc_fcc()

```

subroutine lcc_lattice_mod::lcc_fcc (
    integer, intent(in) Nx1,
    integer, intent(in) Nx2,
    integer, intent(in) Ny1,
    integer, intent(in) Ny2,
    integer, intent(in) Nz1,
    integer, intent(in) Nz2,
    real(dp), intent(in) h_lattice_a,
    real(dp), dimension(:, :), intent(inout), allocatable supra_lattice_vectors,
    real(dp), dimension(:, :), intent(inout), allocatable r_sy,
    integer, intent(in) verbose )

```

Face center cubic (FCC) lattice construction.

Constructs a "bulk" of Face center cubic lattice.

Parameters

<i>Nx1</i>	Initial x lattice point.
<i>Nx2</i>	Final x lattice point.
<i>Ny1</i>	Initial y lattice point.
<i>Ny2</i>	Final y lattice point.
<i>Nz1</i>	Initial z lattice point.
<i>Nz2</i>	Final z lattice point.
<i>h_lattice_a</i>	Lattice parameter.
<i>supra_lattice_vectors</i>	Lattice unit vectors of the resulting slab.
<i>r_sy</i>	Output system coordinates.

10.6.2.5 lcc_make_lattice()

```

subroutine, public lcc_lattice_mod::lcc_make_lattice (
    type(build_type), intent(inout) bld,
    type(lattice_type), intent(inout) ltt,
    logical, intent(in) check,
    type(system_type), intent(inout) sy )

```

Make a lattice depending on the input parameter.

This will make one of the following lattices: SC: Simple cubic, FCC: Face center cubic, or Triclinic.

Parameters

<i>bld</i>	Building structure (see lcc_structures_mod)
<i>ltt</i>	Lattice structure (see lcc_sctructures_mod)
<i>check</i>	If we want to check the basis for atom repetition. Note that checks can be expensive.

10.6.2.6 lcc_minimize_from()

```
subroutine lcc_lattice_mod::lcc_minimize_from (
    real(dp), dimension(:, :), intent(in) xVar,
    integer, intent(in) i,
    integer, intent(in) ai,
    integer, intent(in) nats,
    real(dp), dimension(3), intent(inout) trs,
    integer, intent(in) verbose )
```

To get the best translation that minimizes the distance to any previous fragment.

Parameters

<i>xVar</i>	Coordinates of the full basis (including symmetry operations).
<i>i</i>	Fragment being added at the "i" operation.
<i>ai</i>	Atom index to translate and get the optimal translation.
<i>nats</i>	Number of atoms in the fragment.
<i>trs</i>	Optimal translation.

10.6.2.7 lcc_read_base()

```
subroutine lcc_lattice_mod::lcc_read_base (
    type(build_type), intent(inout) bld,
    type(lattice_type), intent(inout) ltt,
    logical, intent(in) check,
    integer, intent(in) verbose )
```

Reading the basis from an input file.

This will read the coordinates for the basis from an input file. If information about the lattice is contained, it will also be read.

Parameters

<i>bld</i>	Building structure (see lcc_structures_mod).
<i>ltt</i>	Lattice structure (see lcc_structures_mod).
<i>check</i>	If we want to check the basis for atom repetition.
<i>verbose</i>	Verbose level. Note that checks can be expensive.

10.6.2.8 lcc_sc()

```
subroutine lcc_lattice_mod::lcc_sc (
    integer, intent(in) Nx1,
```

```

integer, intent(in) Nx2,
integer, intent(in) Ny1,
integer, intent(in) Ny2,
integer, intent(in) Nz1,
integer, intent(in) Nz2,
real(dp), intent(in) h_lattice_a,
real(dp), dimension(:, :), intent(inout), allocatable supra_lattice_vectors,
real(dp), dimension(:, :), intent(inout), allocatable r_sy )

```

Simple cubic (SC) lattice construction.

Constructs a "bulk" of Simple Cubic lattice.

Parameters

<i>Nx1</i>	Initial x lattice point.
<i>Nx2</i>	Final x lattice point.
<i>Ny1</i>	Initial y lattice point.
<i>Ny2</i>	Final y lattice point.
<i>Nz1</i>	Initial z lattice point.
<i>Nz2</i>	Final z lattice point.
<i>h_lattice_a</i>	Lattice parameter.
<i>supra_lattice_vectors</i>	Lattice unit vectors of the resulting slab.
<i>r_sy</i>	Output system coordinates.

10.6.2.9 lcc_set_atom_type()

```

subroutine, public lcc_lattice_mod::lcc_set_atom_type (
  character(len=2), intent(in) a_type,
  character(len=2), dimension(:), intent(inout), allocatable atom_symbol,
  character(len=3), dimension(:), intent(inout), allocatable atom_name,
  integer, intent(in) nats )

```

Sets the atom type.

Sets the atom "symbol/type/name."

Parameters

<i>a_type</i>	Atom symbol character.
<i>atom_symbol</i>	Atom symbols.
<i>atom_name</i>	Atom name. Note: Atom name is a tag that can distinguish atoms with same symbol.

10.6.2.10 lcc_triclinic()

```

subroutine lcc_lattice_mod::lcc_triclinic (
  integer, intent(in) Nx1,

```

```

integer, intent(in) Nx2,
integer, intent(in) Ny1,
integer, intent(in) Ny2,
integer, intent(in) Nz1,
integer, intent(in) Nz2,
real(dp), dimension(:, :), intent(in), allocatable lattice_vectors,
real(dp), dimension(:, :), intent(inout), allocatable supra_lattice_vectors,
real(dp), dimension(:, :), intent(inout), allocatable r_sy,
integer, intent(in) verbose )

```

Triclinic lattice construction.

Constructs a "bulk" of triclinic lattice.

Parameters

<i>Nx1</i>	Initial x lattice point.
<i>Nx2</i>	Final x lattice point.
<i>Ny1</i>	Initial y lattice point.
<i>Ny2</i>	Final y lattice point.
<i>Nz1</i>	Initial z lattice point.
<i>Nz2</i>	Final z lattice point.
<i>lattice_vectors</i>	Lattice vectors.
<i>supra_lattice_vectors</i>	Lattice unit vectors of the resulting slab.
<i>r_sy</i>	Output system coordinates. Note: Unit cell representation has to be transformed from edges and angles to vetors before calling this routine.

Todo A angles_to_vectors transformation will be available.

10.7 lcc_lib Module Reference

Library module.

Functions/Subroutines

- subroutine, public **lcc** (readInputFile, inputFileName, syOut, writeOut, ciType, planeIn)

10.7.1 Detailed Description

Library module.

10.8 lcc_mc_mod Module Reference

Module for Monte Carlo related routines.

Functions/Subroutines

- subroutine [lcc_check_system](#) (r, iter, temp, cost, cost0)
Maximize: This checks the acceptance.

10.8.1 Detailed Description

Module for Monte Carlo related routines.

10.9 lcc_message_mod Module Reference

Module for printing through the code.

Functions/Subroutines

- subroutine, public [lcc_print_usage](#) ()
For printing the instructions on how to execute the code.
- subroutine, public [lcc_print_message](#) (message, verbose)
Print a simple message.
- subroutine, public [lcc_print_warning](#) (at, message, verbose)
Print a Warning (will not stop execution).
- subroutine, public [lcc_print_error](#) (at, message)
Print error (will stop execution).
- subroutine, public [lcc_print_intval](#) (name, value, units, verbose)
Print integer magnitude.
- subroutine, public [lcc_print_realval](#) (name, value, units, verbose)
Print real magnitude.
- subroutine, public [lcc_print_realvect](#) (name, vect, units, verbose)
Print real vector.
- subroutine [lcc_print_realmat](#) (name, mat, units, verbose)
Print real vector.
- subroutine [lcc_help](#) ()

10.9.1 Detailed Description

Module for printing through the code.

10.9.2 Function/Subroutine Documentation

10.9.2.1 lcc_print_error()

```
subroutine, public lcc_message_mod::lcc_print_error (
    character(len=*), intent(in) at,
    character(len=*), intent(in) message )
```

Print error (will stop execution).

Parameters

<i>at</i>	Name of the routine.
<i>message</i>	Message to print.

10.9.2.2 lcc_print_intval()

```
subroutine, public lcc_message_mod::lcc_print_intval (
    character(len=*), intent(in) name,
    integer, intent(in) value,
    character(len=*), intent(in) units,
    integer, intent(in) verbose )
```

Print integer magnitude.

Parameters

<i>name</i>	Name of the magnitude.
<i>value</i>	Value to print.
<i>units</i>	Units of the magnitude.

10.9.2.3 lcc_print_message()

```
subroutine, public lcc_message_mod::lcc_print_message (
    character(len=*), intent(in) message,
    integer, intent(in) verbose )
```

Print a simple message.

Parameters

<i>message</i>	Message to print.
<i>verbose</i>	Verbosity level.

10.9.2.4 lcc_print_realmat()

```
subroutine lcc_message_mod::lcc_print_realmat (
    character(len=*), intent(in) name,
    real(dp), dimension(:, :), intent(in), allocatable mat,
    character(len=*), intent(in) units,
    integer, intent(in) verbose )
```

Print real vector.

Parameters

<i>name</i>	Name of the quantities.
<i>mat</i>	Matrix to print.
<i>units</i>	Units of the quantities.
<i>verbose</i>	Verbosity level.

10.9.2.5 lcc_print_realval()

```
subroutine, public lcc_message_mod::lcc_print_realval (
    character(len=*), intent(in) name,
    real(dp), intent(in) value,
    character(len=*), intent(in) units,
    integer, intent(in) verbose )
```

Print real magnitude.

Parameters

<i>name</i>	Name of the magnitude.
<i>value</i>	Value to print.
<i>units</i>	Units of the magnitude.

10.9.2.6 lcc_print_realvect()

```
subroutine, public lcc_message_mod::lcc_print_realvect (
    character(len=*), intent(in) name,
    real(dp), dimension(:), intent(in), allocatable vect,
    character(len=*), intent(in) units,
    integer, intent(in) verbose )
```

Print real vector.

Parameters

<i>name</i>	Name of the quantities.
<i>vect</i>	Vector to print.
<i>units</i>	Units of the quantities.
<i>verbose</i>	Verbosity level.

10.9.2.7 lcc_print_warning()

```
subroutine, public lcc_message_mod::lcc_print_warning (
```

```
character(len=*), intent(in) at,
character(len=*), intent(in) message,
integer, intent(in) verbose )
```

Print a Warning (will not stop execution).

Parameters

<i>at</i>	Name of the routine.
<i>message</i>	Message to print.
<i>verbose</i>	Verbosity level.

10.10 lcc_parser_mod Module Reference

This module controls the initialization of the variables.

Functions/Subroutines

- subroutine, public [lcc_parse](#) (filename, bld, ltt)
Clustergen parser.
- subroutine, public [lcc_make_sample_input](#) ()
Make a sample inputfile sample_input.in.
- subroutine, public [lcc_write_coords](#) (sy, bld, coordsout_file, verbose)
Writes the coordinates to a file (coordsandbase.pdb)

10.10.1 Detailed Description

This module controls the initialization of the variables.

10.10.2 Function/Subroutine Documentation

10.10.2.1 lcc_parse()

```
subroutine, public lcc_parser_mod::lcc_parse (
    character(len=*), intent(in) filename,
    type(build\_type), intent(inout) bld,
    type(lattice\_type), intent(inout) ltt )
```

Clustergen parser.

This module is used to parse all the input variables for this program. Adding a new input keyword to the parser:

- If the variable is real, we have to increase nkey_re.
- Add the keyword (character type) in the keyvector_re vector.
- Add a default value (real type) in the valvector_re.
- Define a new variable and pass the value through valvector_re(num) where num is the position of the new keyword in the vector.

Parameters

<i>filename</i>	File name for the input.
<i>bld</i>	Build type.
<i>lft</i>	Lattice type.

10.10.2.2 lcc_write_coords()

```

subroutine, public lcc_parser_mod::lcc_write_coords (
    type(system_type) sy,
    type(build_type) bld,
    character(len=*) coordsout_file,
    integer, intent(in) verbose )

```

Writes the coordinates to a file (coordsandbase.pdb)

Parameters

<i>sy</i>	System type.
<i>bld</i>	Build type.
<i>coordsout_file</i>	File name to write the coordinates to.
<i>verbose</i>	Verbosity level.

10.11 lcc_regular_mod Module Reference

Module for generating regular shapes after lattice is constructed.

Functions/Subroutines

- subroutine, public [lcc_spheroid](#) (a_axis, b_axis, c_axis, r_inout)
For building spheroidal shapes out of a bulk lattice.

10.11.1 Detailed Description

Module for generating regular shapes after lattice is constructed.

10.11.2 Function/Subroutine Documentation

10.11.2.1 lcc_spheroid()

```
subroutine, public lcc_regular_mod::lcc_spheroid (
    real(dp) a_axis,
    real(dp) b_axis,
    real(dp) c_axis,
    real(dp), dimension(:, :), allocatable r_inout )
```

For building spheroidal shapes out of a bulk lattice.

Parameters

<i>a_axis</i>	Lenght in the x direction.
<i>b_axis</i>	Lenght in the y direction.
<i>c_axis</i>	Lenght in the z direction.
<i>r_inout</i>	Input and output coordinates.

10.12 lcc_string_mod Module Reference

Module for manipulating strings.

Functions/Subroutines

- subroutine, public [lcc_get_word](#) (string, posh, post, word)
Cut a word from string.
- subroutine, public [lcc_split_string](#) (string, delimit, head, tail)
Split a string in two words uning a delimiter.

10.12.1 Detailed Description

Module for manipulating strings.

10.12.2 Function/Subroutine Documentation

10.12.2.1 lcc_get_word()

```
subroutine, public lcc_string_mod::lcc_get_word (
    character(len=*), intent(in) string,
    integer, intent(in) posh,
    integer, intent(in) post,
    character(20), intent(inout) word )
```

Cut a word from string.

Parameters

<i>string</i>	Full string.
<i>posh</i>	Cut from position.
<i>post</i>	Cut to position.
<i>word</i>	Extracted word.

10.12.2.2 lcc_split_string()

```
subroutine, public lcc_string_mod::lcc_split_string (
    character(len=*), intent(in) string,
    character(1), intent(in) delimit,
    character(20), intent(inout) head,
    character(20), intent(inout) tail )
```

Split a string in two words using a delimiter.

Parameters

<i>string</i>	Full string.
<i>delimit</i>	Delimiter.
<i>head</i>	First word.
<i>tail</i>	Last word.

10.13 lcc_structs_mod Module Reference

A module to handle the structures needed by the code.

Data Types

- type [build_type](#)
Build type.
- type [lattice_type](#)
Lattice type to be read and extended.

10.13.1 Detailed Description

A module to handle the structures needed by the code.

This module will be used to build and handle structures in the code.

Chapter 11

Class Documentation

11.1 Icc_structs_mod::build_type Type Reference

Build type.

Public Attributes

- character(len=20) [job_name](#)
Job name.
- character(len=20) [output_file_name](#)
Output file name.
- character(len=60), public [coordsout_file](#)
Output file name for coordinates.
- character(len=60), public [latticebase_file](#)
Lattice base file name.
- character(len=1) [cut_by_planes](#)
Cut lattice using planes.
- character(len=1) [cut_with_base](#)
Cut lattice after base is added.
- character(len=1) [read_lattice_from_file](#)
Read lattice from file.
- character(len=1) [use_lattice_base](#)
Use lattice base.
- character(len=60) [cl_type](#)
Cluster (or solid) shape to be constructed.
- character(len=60) [planes_type](#)
Type of planes used for the cut.
- character(len=60) [seed_file](#)
File name for the seed used to grow a cluster.
- integer [n](#)
Number of atoms.
- integer [nplanes](#)
Number of planes to use in the cut.
- integer [nx1](#)
Number of lattice points in +-(x, y, and z) directions.

- integer **nx2**
- integer **ny1**
- integer **ny2**
- integer **nz1**
- integer **nz2**
- integer **seed**
Random seed.
- integer **cl_number**
Cluster number (if it is a solid with "magic" numbers)
- real(dp) **a_axis**
Axis length if cluster is a spheroid.
- real(dp) **b_axis**
- real(dp) **c_axis**
- real(dp) **rcoeff**
Coefficient used with random seed to create noise in coordinates.
- real(dp) **r_cut**
Cutoff radius to build spheroids.
- real(dp) **trunc**
Truncation for solids.
- character(2) **a_type**
Atom type (if specified on the input file)
- real(dp), dimension(:,:), allocatable **planes**
Planes for the cut.
- real(dp), dimension(:), allocatable **ploads**
Planes weight factors.
- type(system_type) **syseed**
System seed to be grow on top.
- integer **ncluster**
Number of atoms in cluster/slab.
- character(2), dimension(:), allocatable **atom_in**
Atoms in the cluster/slab.
- character(2), dimension(:), allocatable **atomname_in**
- integer, dimension(:), allocatable **resindex_in**
- character(2), dimension(:), allocatable **resname_in**
- real(dp), dimension(:,:), allocatable **r_cluster**
Coordinates of the resulting cluster/slab.
- integer **maxcoordination**
Max coordination number.
- real(dp) **rtol**
Distance tolerance for distinguishing coordinates.
- integer **niter**
Number of iterations.
- integer **verbose**
Verbose level.
- logical **center**
Center at box.
- logical **reorient**
Reorient first lattice vector toward x direction.
- logical **writectl**
Reorient first lattice vector toward x direction.
- logical **checkperiod**

- *To check periodicity.*
character(5) [rdfpair](#)
- *To compute RDFs.*
logical [writelmp](#)
- *Write LAMMPS input coordinates.*
logical [interplanardistances](#)
- *Use "number of interplanar distances" as unit of measurement for plane cut.*
real(dp), dimension(:,:), allocatable [slab](#)
- *To build a slab out of regular vectors.*
real(dp), dimension(:), allocatable **sloads**
- logical [randomcoordinates](#)
To add randomness to coordinates.

11.1.1 Detailed Description

Build type.

The documentation for this type was generated from the following file:

- /home/cnegre/LCC/src/lcc_structs_mod.F90

11.2 lcc_structs_mod::lattice_type Type Reference

Lattice type to be read and extended.

Public Attributes

- character(len=3) [base_format](#)
Lattice basis.
- character(len=60) [primitive_format](#)
The lattice primitive format (Angles of Vectors)
- character(len=60) [type_of_lattice](#)
Type of lattice (sc, bcc, fcc, and triclinic)
- real(dp) [angle_alpha](#)
Angles for triclinic lattice.
- real(dp) [angle_beta](#)
- real(dp) [angle_gamma](#)
- real(dp) [h_lattice_a](#)
abc parameters for lattice
- real(dp) [h_lattice_b](#)
- real(dp) [h_lattice_c](#)
- real(dp), dimension(:,:), allocatable [lattice_vectors](#)
Lattice vectors.
- real(dp) [volr](#)
Volume of the cell.
- real(dp), dimension(:,:), allocatable [recip_vectors](#)
Lattice reciprocal vectors.

- real(dp) [volk](#)
Volume of the reciprocal cell.
- integer [nbase](#)
Number of atoms in the basis.
- character(2), dimension(:), allocatable [base_atom](#)
Basis atoms.
- real(dp), dimension(:, :), allocatable [r_base](#)
Basis coordinates.
- type(system_type) [sybase](#)
System for the basis.
- logical [bsopl](#)
If there are symmetry operations to be performed.
- integer [nop](#)
Number of Symmetry operations.
- real(dp), dimension(:, :), allocatable [bstr](#)
Translations to be performed.
- real(dp), dimension(:), allocatable [bsopload](#)
Scaling factors (load) for the translation.
- real(dp), dimension(:, :), allocatable [bssym](#)
Symmetry operation (diagonal)
- integer, dimension(:), allocatable [spindex](#)
Species index.
- real(dp), dimension(:), allocatable [base_mass](#)
System basis masses.
- integer, dimension(:), allocatable [resindex](#)
Residue index.
- real(dp), dimension(:, :), allocatable [bulk](#)
To save the "bulk" positions.
- logical [check](#)
Check lattice.
- logical [getopttrs](#)
Get optimal translations at symmetry operations.
- logical [randomlattice](#)
To add randomness to each lattice position.

11.2.1 Detailed Description

Lattice type to be read and extended.

The type of lattice read from input.

The documentation for this type was generated from the following file:

- /home/cnegr/LCC/src/lcc_structs_mod.F90

Index

inv
 lcc_aux_mod, 28

lcc_add_base_to_cluster
 lcc_lattice_mod, 35

lcc_add_randomness
 lcc_lattice_mod, 36

lcc_add_randomness_to_coordinates
 lcc_build_mod, 32

lcc_allocation_mod, 25
 lcc_reallocate_char2vect, 25
 lcc_reallocate_char3vect, 26
 lcc_reallocate_intmat, 26
 lcc_reallocate_intvect, 26
 lcc_reallocate_realmat, 27
 lcc_reallocate_realvect, 27

lcc_aux_mod, 27
 inv, 28
 lcc_canonical_basis, 28
 lcc_center_at_box, 29
 lcc_center_at_origin, 29
 lcc_get_coordination, 29
 lcc_get_reticular_density, 30
 lcc_parameters_to_vectors, 30
 lcc_vectors_to_parameters, 31

lcc_bravais_growth
 lcc_build_mod, 32

lcc_build_mod, 31
 lcc_add_randomness_to_coordinates, 32
 lcc_bravais_growth, 32
 lcc_build_slab, 32
 lcc_plane_cut, 33

lcc_build_slab
 lcc_build_mod, 32

lcc_canonical_basis
 lcc_aux_mod, 28

lcc_center_at_box
 lcc_aux_mod, 29

lcc_center_at_origin
 lcc_aux_mod, 29

lcc_check_basis
 lcc_lattice_mod, 36

lcc_check_mod, 34
 lcc_check_periodicity, 34

lcc_check_periodicity
 lcc_check_mod, 34

lcc_constants_mod, 34

lcc_fcc
 lcc_lattice_mod, 37

lcc_get_coordination
 lcc_aux_mod, 29

lcc_get_reticular_density
 lcc_aux_mod, 30

lcc_get_word
 lcc_string_mod, 46

lcc_lattice_mod, 35
 lcc_add_base_to_cluster, 35
 lcc_add_randomness, 36
 lcc_check_basis, 36
 lcc_fcc, 37
 lcc_make_lattice, 37
 lcc_minimize_from, 38
 lcc_read_base, 38
 lcc_sc, 38
 lcc_set_atom_type, 39
 lcc_triclinic, 39

lcc_lib, 40

lcc_make_lattice
 lcc_lattice_mod, 37

lcc_mc_mod, 40

lcc_message_mod, 41
 lcc_print_error, 41
 lcc_print_intval, 42
 lcc_print_message, 42
 lcc_print_realmat, 42
 lcc_print_realval, 43
 lcc_print_realvect, 43
 lcc_print_warning, 43

lcc_minimize_from
 lcc_lattice_mod, 38

lcc_parameters_to_vectors
 lcc_aux_mod, 30

lcc_parse
 lcc_parser_mod, 44

lcc_parser_mod, 44
 lcc_parse, 44
 lcc_write_coords, 45

lcc_plane_cut
 lcc_build_mod, 33

lcc_print_error
 lcc_message_mod, 41

lcc_print_intval
 lcc_message_mod, 42

lcc_print_message
 lcc_message_mod, 42

lcc_print_realmat
 lcc_message_mod, 42

lcc_print_realval
 lcc_message_mod, 43

- lcc_print_realvect
 - lcc_message_mod, [43](#)
- lcc_print_warning
 - lcc_message_mod, [43](#)
- lcc_read_base
 - lcc_lattice_mod, [38](#)
- lcc_reallocate_char2vect
 - lcc_allocation_mod, [25](#)
- lcc_reallocate_char3vect
 - lcc_allocation_mod, [26](#)
- lcc_reallocate_intmat
 - lcc_allocation_mod, [26](#)
- lcc_reallocate_intvect
 - lcc_allocation_mod, [26](#)
- lcc_reallocate_realmat
 - lcc_allocation_mod, [27](#)
- lcc_reallocate_realvect
 - lcc_allocation_mod, [27](#)
- lcc_regular_mod, [45](#)
 - lcc_spheroid, [45](#)
- lcc_sc
 - lcc_lattice_mod, [38](#)
- lcc_set_atom_type
 - lcc_lattice_mod, [39](#)
- lcc_spheroid
 - lcc_regular_mod, [45](#)
- lcc_split_string
 - lcc_string_mod, [47](#)
- lcc_string_mod, [46](#)
 - lcc_get_word, [46](#)
 - lcc_split_string, [47](#)
- lcc_structs_mod, [47](#)
- lcc_structs_mod::build_type, [49](#)
- lcc_structs_mod::lattice_type, [51](#)
- lcc_triclinic
 - lcc_lattice_mod, [39](#)
- lcc_vectors_to_parameters
 - lcc_aux_mod, [31](#)
- lcc_write_coords
 - lcc_parser_mod, [45](#)