TO INSTALL THE CODE:

Use `gunzip cmhog3.0.tar.gz; tar -xvf cmhog3.0.tar`. This should create two directories:

1. **src** - contains source code
2. **exe** - directory where CMHOG executable is run

TO RUN THE CODE:

The code is located in `/src`. Each subroutine in the code is in a separate file with the `.src` extension. Common blocks are in the `.h` files. There is also a Makefile and a file called `cmhog.def`. The code uses cpp to control the different physics and problem size options. The file `cmhog.def` contains the cpp macro definitions that will tell cpp what to do during the make. Below is a sample version of 'cmhog.def'

```
#define NO_OF_I_ZONES  150        | no. of zones in x-dirn
#define NO_OF_J_ZONES  40         | no. of zones in y-dirn
#define NO_OF_K_ZONES  1          | no. of zones in z-dirn
#define MAX_OF_IJK     150        | max of above three

#define RESTART           empty    | don't change
#define GRID_GENERATOR    ggen     | don't change
#define PROBLEM_GENERATOR dmr      | subroutine name of problem gen.
#define R_SOLVER          twoshock | subroutine name of Riemann solver
#define SPECIAL           dmrbv    | optional routine called in loop

#define A_BIG_NUMBER   1.0e+10  | machine dependent big number
#define A_SMALL_NUMBER 1.0e-10  | machine dependent small number
```

To understand all the macros and their different uses and valid options requires studying the source code in detail. To start with, the most important points are:

- NO_OF_I_ZONES, etc must be 6 more than actual number of grid points used to account for ghost zones. If the equations are not to be solved in a given dimension, set the number of zones in that dimension to one. For example, for a 1D problem with 100 zones in the X-direction, set NO_OF_I_ZONES=106, NO_OF_J_ZONES=1, and NO_OF_K_ZONES=1. For a 2D problem in the YZ-plane with 64×128 zones set NO_OF_I_ZONES=1, NO_OF_J_ZONES=70, and NO_OF_K_ZONES=134.
- PROBLEM_GENERATOR is the name of the subroutine called during setup which defines the initial conditions for each dependent variable. The example here runs the Double Mach Reflection (dmr) test problem given in Woodward & Collela, JCP, 54 (1984). The problem generator is in the file `dmr.src`. Other problem generators are `advect.src`, `sedov.src`, `shkset.src`, and `twoibw.src`.
- SPECIAL is an optional routine called once every cycle. In this example, it modifies the boundary conditions along the top edge of the grid to account for the shock motion in the DMR test. The routine is in the file `dmrbv.src`. In general, if SPECIAL is not needed then just set it to the (empty) subroutine `special` (contained in the file `special.src`).

To compile the code use the Makefile in the directory `/src` using

`make PROB=`*problem_generator* `SPEC=`*special* `compile`

where *problem_generator* is the same as the macro name in `cmhog.def` (the name of the problem-generator subroutine), and SPEC is the name of the SPECIAL function. The Makefile assumes the fortran and C compilers are f77 and cc respectively. No optimization flags are used; the Makefile can be edited to add optimization. The Makefile also has several other targets, such as 'clean' (`make clean` removes all the .o files).

The code uses an input deck called `cmhogin` in the directory `/exe` to set runtime parameters. Examples are given in the `/exe` directory with the names `cmhogin_*`. To run the DMR test problem use `cmhogin_dmr`. Each line of the input deck is read by a different namelist in the routines `setup.src` or `ggen.src`; look at these routines to figure out what each parameter does and what it is called. Actually the use of namelists is not standard fortran 77 and is the biggest source of pain in porting the code. But namelists are so useful they are used anyways.

The code outputs data in two file types. The subroutine `hdfall.src` writes out all dependent variables over the whole mesh in HDF4 scientific data set format. HDF4 is a data format standard developed at NCSA. HDF4 files are binary compatible across a wide variety of machines, which makes them useful. The HDF4 libraries can be downloaded from the NCSA website. Note that the latest version of HDF is V5, and that HDF4 and HDF5 files are incompatible. Else you can always change `hdfall.src` to write the data in whatever format you prefer. Note hdf files are written every `thdf` unit of time in the calculation (`thdf` is a parameter in the namelist `iocon`). The first hdf file is called 'hdf000aa', the second 'hdf001aa', and so on throughout the calculation (each data dump written to a separate file). You can then post-process these files to make graphics, movies, etc. The code also write a history dump using the routine `hdump.src` (you can modify it as you like). Restart dumps are written into the file `res000aa` every `tdump` unit of time. The code recognizes various interrupt messages when it is running in an interactive shell, for example typing a question mark ('?') will cause diagnostic information to be output.

As for post-processing the data, a wide variety of NCSA tools to read the HDF4 files and process the data are available. In addition, there is a simple IDL script for reading the HDF4 files called `pltcmhog3.0.pro` in the directory `/exe` which can be used to get started.

SUMMARY

So finally, the steps to run the code are:

1. Edit the file `cmhog.def` in the directory `/src` to set the number of zones/problem type/etc. of the problem you want to run.

2. Use `make PROB=`*problem_generator* `SPEC=`*special* `compile`, where *problem_generator* is the same as the macro name in `cmhog.def` (for the Double Mach Reflection test problem use dmr), *special* is the same as the macro name in `cmhog.def` (for DMR test use dmrbv).

3. When the make is completed it puts the executable called `cmhog.exe` in the directory `/exe`. So go to this directory and edit the input deck `cmhogin` to ensure all parameters to the namelists are set as you would like.

4. Then run the code and generate data!!

As a first step, try running the double mach reflection test problem just to see how the process works. The `cmhog.def` and `cmhogin` files are set up to do this problem in the distribution.

Some final words. The code uses the Lagrange plus remap algorithm with UNsymmetrized directional splitting. The contact steepeners and flatteners and Lapidus viscosity are implemented but not always needed or used. The Riemann solver is the two-shock approximation run for three iterations. There are a lot of different ways to do PPM, and lots of bells and whistles in the algorithm. One needs to delve into the algorithm to really understand it to make intelligent choices. (i.e. Maybe symmetrized splitting is needed for the problem? This is easy to change.) It would be dangerous to use this code as a black box as is, because some of the choices made which are suitable for my problems may not be for yours. The code is flexible and easily changed.