Testing Modernization of Legacy Fortran

Aly Ammar
CISL Intern

July 28, 2019

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

Why Modernize?

- Adhere to language standards
- Portability
- Better reliability
- Improved features allow better functionality

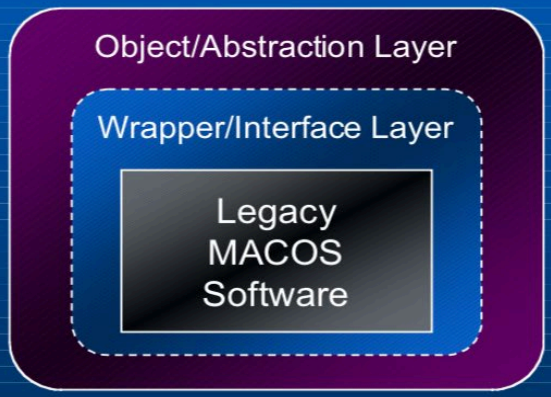
UCLA – NASA Jet Propulsion Lab

- UCLA and NASA Jet Propulsion Lab collaborated to modernize NASA code.
- Modernize legacy code using wrappers.
- Implement modules, interfaces, modern arrays etc.

UCLA – NASA Jet Propulsion Lab

Development

- **Efficient interaction among MACOS, interfaces, abstraction layer, and user I/O**



Allows safe interaction with a legacy code

http://exodus.physics.ucla.edu/high%20performance%20computing/modernization.pdf

UCLA – NASA Jet Propulsion Lab

Pros:

- Retains the original F77 code.
- Code does not have to be taken offline during the process.

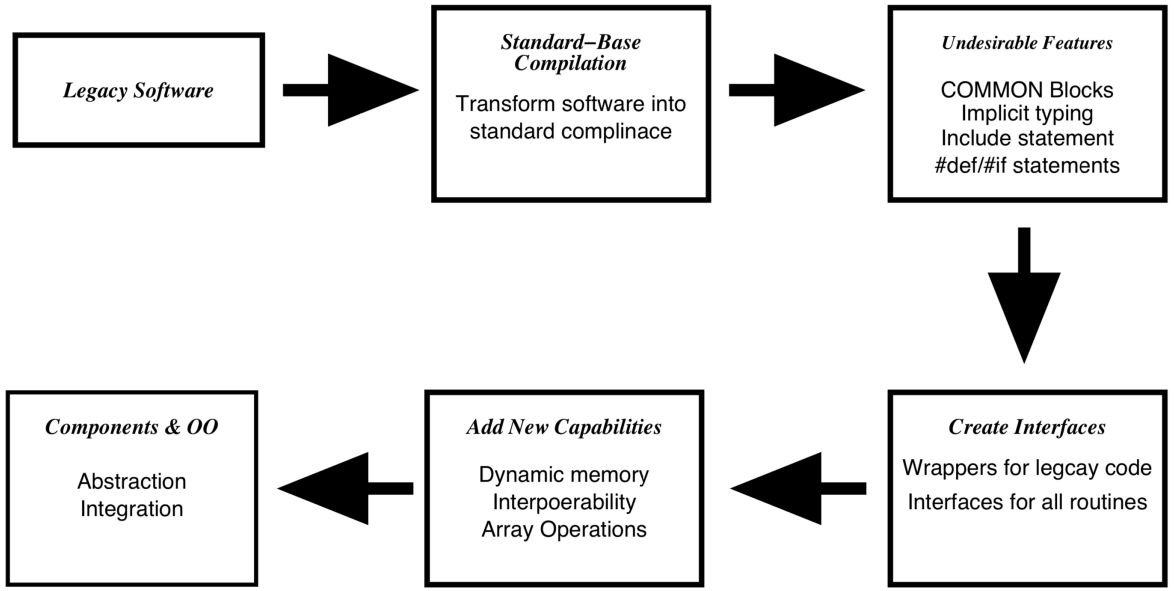
Cons:

- The implementation is specific to the code.
- Difficult to implement for a lot of code.

Rutherford Appleton Laboratory

- Researchers analyze ways to modernize legacy code, mostly Fortran
- Evaluate different stages in modernizing the code
- Consider solutions to pitfalls and common problems
- Suggest tools that can be useful in the process

Rutherford Appleton Laboratory



Greenough C., Worth DJ., 2006, The Transformation of Legacy Software: Some Tools and a Process Version 3, CLRC, 1358-6254

Rutherford Appleton Laboratory

Pros:

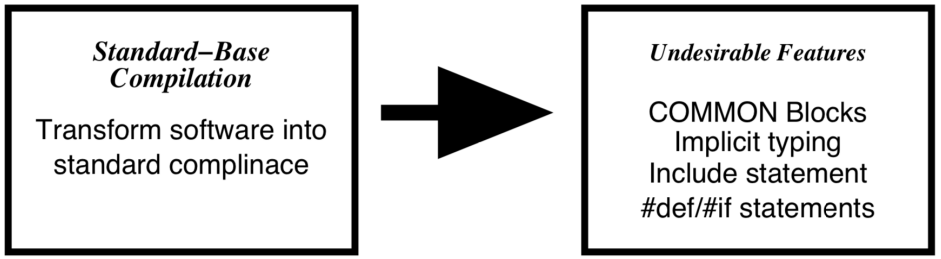
- Objectives and stages similar to the project.
- Provides suggestions for useful tools

Cons:

- Most tools require expensive licenses.
- No all in one solution.

Objectives in Modernization

- Focus on first two stages
  - Fixed form to free form
  - Common blocks
  - Include statements



Greenough C., Worth DJ., 2006, The Transformation of Legacy Software: Some Tools and a Process Version 3, CLRC, 1358-6254

Fixed Form to Free Form

- Fixed form is usually older f66/f77 code
- From the good old punch card days
- Specific number of columns for code, comments etc.

- Free form is similar to traditional code.
- Still has limited number of columns,

- ftools module contains fixed to free form tools
- Some are very powerful
- Some code formatters and pretty printers.

Fixed to Free Form Tools

- To_f90:
  - Very quick.
  - Easy to use.
  - No formatting options
  - Code may have formatting errors
- fconvert:
  - Allows formatting options
  - Still quick
  - Valid fixed and free form output
  - Very basic conversion

Fixed to Free Form Tools

- f90ppr:
  - Lots of directives and macro options
  - Allows complex conversions.
  - Slower compared to other tools.
  - Requires modification of source file
- f2f:
  - Easiest to use
  - Fast
  - Basic conversion
  - No styling options

Pre-Conversion/Post-Conversion with convert

```
      ARG=2.*4.*ATAN(1.0)/5.
      TR11=COS(ARG)
      TI11=SIN(ARG)
      TR12=COS(2.*ARG)
      TI12=SIN(2.*ARG)
      DO 101 K=1,L1
         CH(1,1,K,1) = CC(1,1,1,K)+2.*CC(1,IDO,2,K)+2.*CC(1,IDO,4,K)
         CH(1,1,K,2) = (CC(1,1,1,K)+TR11*2.*CC(1,IDO,2,K)
     1   +TR12*2.*CC(1,IDO,4,K))-(TI11*2.*CC(1,1,3,K)
     1   +TI12*2.*CC(1,1,5,K))
         CH(1,1,K,3) = (CC(1,1,1,K)+TR12*2.*CC(1,IDO,2,K)
     1   +TR11*2.*CC(1,IDO,4,K))-(TI12*2.*CC(1,1,3,K)
     1   -TI11*2.*CC(1,1,5,K))
         CH(1,1,K,4) = (CC(1,1,1,K)+TR12*2.*CC(1,ID
     1   +TR11*2.*CC(1,IDO,4,K))+(TI12*2.*CC(1,1,3,
     1   -TI11*2.*CC(1,1,5,K))
         CH(1,1,K,5) = (CC(1,1,1,K)+TR11*2.*CC(1,ID
     1   +TR12*2.*CC(1,IDO,4,K))+(TI11*2.*CC(1,1,3,
     1   +TI12*2.*CC(1,1,5,K))
101   CONTINUE
```

<- Original Code:

Notice the ‘1’s indicating the continuing line marker.

The continue statement indicates the end of loop

Converted Code: ->

The continuation indicators are now ‘’&’ symbols.

The continue statement is not changed to End Do statement.

```
      ARG = 2. * 4. * ATAN (1.0) / 5.
      TR11 = COS (ARG)
      TI11 = SIN (ARG)
      TR12 = COS (2. * ARG)
      TI12 = SIN (2. * ARG)
      DO 101 K = 1, L1
         CH (1, 1, K, 1) = CC (1, 1, 1, K) + 2. * CC (1, IDO, 2, K)     &
         + 2. * CC (1, IDO, 4, K)
         CH (1, 1, K, 2) = (CC (1, 1, 1, K) + TR11 * 2. * CC (1, IDO, 2,&
         K) + TR12 * 2. * CC (1, IDO, 4, K) ) - (TI11 * 2. * CC (1, 1,  &
         3, K) + TI12 * 2. * CC (1, 1, 5, K) )
         CH (1, 1, K, 3) = (CC (1, 1, 1, K) + TR12 * 2. * CC (1, IDO, 2,&
         K) + TR11 * 2. * CC (1, IDO, 4, K) ) - (TI12 * 2. * CC (1, 1,  &
         3, K) - TI11 * 2. * CC (1, 1, 5, K) )
         CH (1, 1, K, 4) = (CC (1, 1, 1, K) + TR12 * 2. * CC (1, IDO, 2,&
         K) + TR11 * 2. * CC (1, IDO, 4, K) ) + (TI12 * 2. * CC (1, 1,  &
         3, K) - TI11 * 2. * CC (1, 1, 5, K) )
         CH (1, 1, K, 5) = (CC (1, 1, 1, K) + TR11 * 2. * CC (1, IDO, 2,&
         K) + TR12 * 2. * CC (1, IDO, 4, K) ) + (TI11 * 2. * CC (1, 1,  &
         3, K) + TI12 * 2. * CC (1, 1, 5, K) )
101   END DO
```

Common Blocks

- Possibly the most undesirable feature
- Was the only way to implement global variables
- Misused too often

- Post 90/95 Fortran allow USE statements.
- Recommend modules for global variables
- Modules are safer and more versatile.

ctm.py

- Created a new tool to convert common blocks to modules
- Parses code to find common blocks
- Create a module file for the common block
- Replace common statement with “USE modname”
- Ensure common statement is the same size in the entire code.

- Still needs work to be fully functional

Demo script

```
c
      subroutine muh2(iparm,fparm,wk,iwk,coef,bndyc,rhs,phi,mgopt,
     +                ierror)
      implicit none
      integer iparm(17),mgopt(4),iwk(*),ierror
      integer intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nfx,nfy,iguess,
     +             maxcy,method,nwork,lwork,itero,ngrid,klevel,kcur,
     +             kcycle,iprer,ipost,intpol,kps
      real fparm(6),xa,xb,yc,yd,tolmax,relmax
      integer kpbgn,kcbgn,ktxbgn,ktybgn,nxk,nyk,isx,jsy
      integer int,k,kb,nx,ny,ic,itx,ity,iw
      real wk(*),phi(*),rhs(*)
      common/imud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nfx,nfy,iguess,
     +             maxcy,method,nwork,lwork,itero,ngrid,klevel,kcur,
     +             kcycle,iprer,ipost,intpol,kps
      common/fmud2/xa,xb,yc,yd,tolmax,relmax
      common/mud2c/kpbgn(50),kcbgn(50),ktxbgn(50),ktybgn(50),
     +nxk(50),nyk(50),isx,jsy
      integer ibeta,ialfa,izmat,idmat
      common/muh2c/ibeta,ialfa,izmat,idmat
      external coef,bndyc
```

```
cuh2.f    Makefile        mod_mud2c.f90   mud24.f
cuh34.f   mod_fmud2.f90   mod_muh2c.f90   mud24sp.f
cuh3.f    mod_imud2.f90   mud24cr.f       mud2cr.f
```

```
c     muacom.r
c
      subroutine muh2(iparm,fparm,wk,iwk,coef,bndyc,rhs,phi,mgopt,
     +                ierror)
use imud2_mod
use fmud2_mod
use mud2c_mod
use muh2c_mod
      implicit none
      integer iparm(17),mgopt(4),iwk(*),ierror
      integer intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nfx,nfy,iguess,
     +             maxcy,method,nwork,lwork,itero,ngrid,klevel,kcur,
     +             kcycle,iprer,ipost,intpol,kps
      real fparm(6),xa,xb,yc,yd,tolmax,relmax
      integer kpbgn,kcbgn,ktxbgn,ktybgn,nxk,nyk,isx,jsy
      integer int,k,kb,nx,ny,ic,itx,ity,iw
      real wk(*),phi(*),rhs(*)
!      common/imud2/intl,nxa,nxb,nyc,nyd,ixp,jyq,iex,jey,nfx,nfy,iguess,
```

```
subroutine fmud2_mod
!       common/fmud2/xa,xb,yc,yd,tolmax,relmax
end module fmud2_mod
```

Moving Forward

- Explore ways to replace deleted or obsolescent features
- Finish development of the ctm.py tool
- Evaluate application of co-arrays for parallelization.
- Eat a cookie!

Acknowledgements

- Mentors Dan Nagle and Davide Del Vento
- CISL, NCAR and SIParCS staff