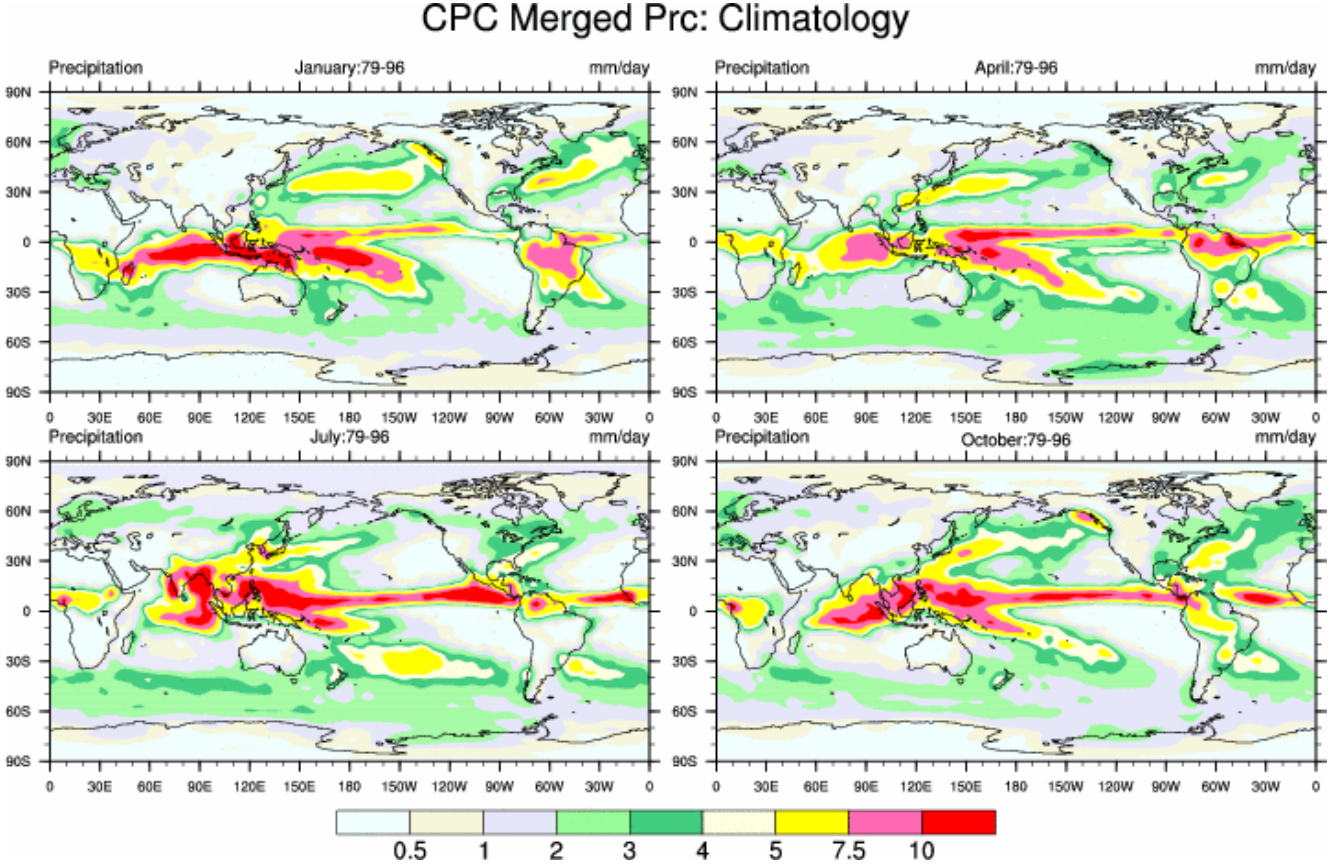
NCL Data Processing



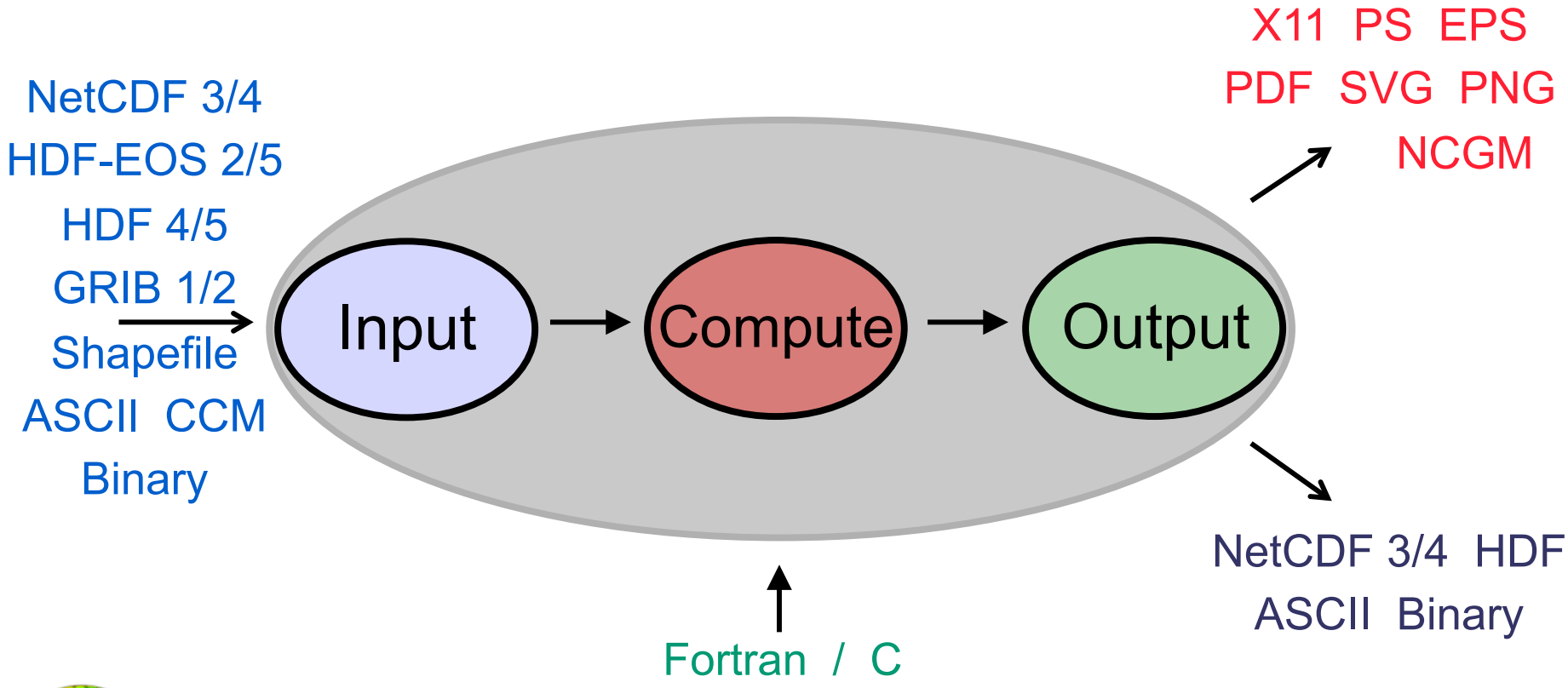
Dennis Shea

National Center for Atmospheric Research

NCAR is sponsored by the National Science Foundation

NCAR Command Language

An Integrated Processing Environment



NCL

Data Processing: *Coding*

1. Clear and simple code is best
2. Indent code blocks
3. Use descriptive variable names
4. Comment code segments and overall objectives
5. Use built-in functions: efficiency
6. Create functions to perform repetitive tasks
7. Use parameters in place of hard-coded numbers
8. Test code segments (unit testing)

Data Processing: *Coding*

1. Keep code clear and simple; 7. Use parameters

Clear code is better than slick code

Slick code:

```
F = f2gsh( fo2fsh( fbinrecread(f,6,(/9,18,72,144/)
          , "float")),(/nlat,mlon/),42)
```

Clear code:

```
nr    = 6     ; record to read (NCL 0-based)
trunc= 42     ; triangular truncation
nlat = 64     ; number of gaussian latitudes
mlon = 128    ; number of longitudes

data = fbinrecread(f,nr,(/9,18,72,144/), "float")
F    = f2gsh( fo2fsh(data),(/nlat,mlon/),trunc)
```

Data Processing: *Coding*

2. Indent code blocks; 4. Comment code

Unindented, Uncommented code:

```
do nyr=yrStrt,yrLast
 nmoStrt=0
 nmoLast=NMOS-1
 if(yr.eq.rStrt)then
 nmoStrt =month(0)-1
end if
if(yr.eq.yrLast)then
nmoLast=month(ntim)-1
end if
do nmo=nmoStrt,nmoLast
ii=ind(yr.eq.year.and.(nmo+1).eq.month)
  :
end do
end do
```

Data Processing: *Coding*

2. Indent code blocks; 4. Comment code

Indented, commented code:

```
;--- Loop over years and available month; calculate PV

do nyr=yrStrt,yrLast
     nmoStrt = 0                        ; index to start (default)
     nmoLast = NMOS-1                   ; index to end  (default)
     if (yr.eq.yrStrt) then
        nmoStrt = month(0)-1            ; possible partial first year
     end if
     if (yr.eq.yrLast) then
        nmoLast = month(ntim)-1         ; possible partial last year
     end if
   do nmo=nmoStrt,nmoLast               ; loop over available months for 'nyr'
        :
   end do                               ; nmo end
 end do                                 ; nyr end
```

Data Processing: *Coding*

3. Use descriptive variable names

Undescriptive:

```
qq = foo*(1000/z1)^0.286
```

Descriptive:

```
theta = tmp*(1000/prs)^0.286
```

Data Processing: *Know Your Data*

1. Examine file contents prior to using data
2. Variable information: names, types, sizes, shapes
3. What attributes are associated with the variables
4. If NetCDF, is some convention being used
5. NCL code: use printVarSummary often

Data Processing: *Know Your Data*

1. Examine file contents prior to use

   ncl_filedump: NetCDF-3/4, HDF-4/5, GRIB-1/2

   ncdump –h: NetCDF-3/4 (Unidata)

Usage from command line:

```
ncl_filedump MYD10CM.A2011001.hdf | less
ncdump –h b40.cam2.h0.1982-01.nc | less
```

Data Processing: *Know Your Data*

Sample file examination: ncl_filedump, ncdump –h

```
global attributes:
        Conventions = “CF-1.0”
dimensions:
        time = UNLIMITED ; // (3704 currently)
double time(time)                                                   ; 1D
        time:units = "days since 1850-01-01 00:00:00”  ; cd_calendar
        time:calendar = "noleap”                                   ; models
float VT(time,lev,lat,lon)                                          ; 4D
        VT: units = “K m/s”
        VT: long_name = “Meridional heat transport”
short olr(time, lat, lon)                                           ; 3D
        olr:add_offset = 327.65f                    ; must be unpacked
        olr:scale_factor = 0.01f                    ; short2flt
        olr: long_name = “outgoing long wave radiation”
```

Data Processing: *Know Your Data*

Sample variable examination: printVarSummary

```
Variable: prc                          ; NCL variable (data object; structure)
Type: float                            ; printed form of NetCDF variable
Number of Dimensions: 3
Dimensions and sizes:   [time | 352] x [lat | 72] x [lon | 144]
Coordinates:
            time: [101902..112585]     ; coordinate variable; {...} syntax
            lat: [88.75..-88.75]       ;       “           “
            lon: [1.25..358.75]        ;       “           “
Number Of Attributes: 15
  long_name :    Average Monthly Rate of Precipitation
  units : mm/day
_FillValue :        -9.96921e+36
```

NCL Processing Outline

- Algebraic and Logical operators
- **if** statements; **do** / **do while** loops; **where**
- Common error messages

Logical Relational (Boolean) Operators

Same as fortran-77

.le.	less than or equal
.lt.	less than
.ge.	greater than or equal to
.gt.	greater than
.ne.	not equal
.eq.	equal
.and.	and
.not.	not
.or.	or

.and. .not. .or. combine logical expressions

Algebraic Operators

All support scalar and array operations

-	Subtraction / Negation
+	Addition / String concatenation
*	Multiplication
/	Divide
%	Modulus (integers only)
>	Greater than selection
<	Less-than selection
#	Matrix multiply
^	Exponentiation

Algebraic Operators

+ is an overloaded operator
(…) allows you to circumvent precedence rules

algebraic operator:

```
x = 5.3 + 7.95          ➔  x = 13.25
```

concatenate string:

```
str = "pine" + "apple"   ➔  str = "pineapple"
```

algebraic operator and string concatenator:

```
x = "alpha" + "_" + (5.3 + 7)
   ➔ x = "alpha_12.3"
```

Array Syntax/Operators

- Similar to array languages like: f90/f95, Matlab, IDL
- Arrays must conform: same size and shape
- Scalars automatically conform to all array sizes
- Non-conforming arrays: use built-in conform function
- All array operations automatically ignore _FillValue
- Use of array syntax is essential for efficiency

Array Syntax/Operators

Example: clipping arrays

Let foo = (/ 1, 9, 14, -9.3, 0 /)

FOO = foo **>** -1.8 ; (/ 1, 9, 14, -1.8, 0 /)

Let SST be (100,72,144) and SICE = -1.8 (scalar)

```
SST = SST > SICE
```

Fortran 90 equivalent:

```
where (SST.lt.SICE) SST = SICE
```

NCL: **where** used more commonly

```
SST = where(SST.lt.SICE, SICE, SST)
```

Array Syntax/Operators

Example: arrays conform

Let T(30,30,64,128), P(30,30,64,128) => arrays conform

```
THETA = T*(1000/P)^0.286 ; THETA(30,30,64,128)
```

Example: arrays do not conform

Let T be (30,**30**,64,128), P be (**30**).

(0, **1**, 2, 3) **<=** dimension numbers

```
theta = T*(1000/conform(T,P,1))^0.286
```

```
theta(30,30,64,128)
```

Conditional/Repetitive Execution

- **if** : conditional execution of one or more statements
- **do** : loops; fixed repetitions; **for** other languages; fortran
- **do while** : until some condition is met
- **where** : conditional/repetitive execution

if blocks (1)

if-then-end if (note: end if has space)

```
if ( all(a.gt.0) ) then  ; then is optional
  . . .statements. . .
end if                   ; space is required
```

if-then-else-end if

```
if ( any(ismissing(a)) ) then
  . . .statements. . .
else
  . . .statements. . .
end if
```

lazy expression evaluation [left-to-right]

```
if ( any(b.lt.0) .and. all(a.gt.0) ) then
    . . .statement. . .
end if
```

if blocks (2)

- No 'else if' statement
- However, 'else' and 'if' can be grouped on same line
- Every 'if' / 'else if' must have corresponding 'end if'

```
str = "MAR"

if (str.eq."JAN") then
  print("January")
  else if (str.eq."FEB") then
    print("February")
    else if (str.eq."MAR") then
    print("March")
      else if (str.eq."APR") then
        print("April")
      else
      print("Enough of this!")
      end if                          ; end if
    end if                            ; must be grouped
  end if                              ; at the end
end if
```

do loops(1)

- **do** : code segments repeatedly executed ‘n’ times
- Use of multiple embedded **do** loops should be minimized

do loops(2)

do-end do (note: end do has space)

```
do  i=scalar_start_exp, scalar_end_exp [, scalar_stride_exp]
      . . .statements. . .
end do
```

stride always positive; default is one (1)

```
do  n=nStrt, nLast  [,stride]     ; all scalars;     stride always positive
    . . .statements. . .
end do

do  n=nLast, nStrt, 5             ; nLast>nStrt decreases each iteration
    . . .statements. . .
end do
```

do loops(3)

- Sequential loop execution my be altered

break: based on some condition **exit** current loop

```
do  i=iStrt, iLast
      . . .statements. . .
      if (foo.gt 1000) then
         dum = 3*sqrt(foo)     ; optional . . .statements. . .
         break                 ; go to statement after end do
      end if
      . . .statements. . .
end do
. . .statements. . .           ; first statement after end do
```

do loops(4)

- Sequential loop execution my be altered

continue: based on some condition go to **next** iteration

```
do  i=iStrt, iLast
      . . .statements. . .
      if (foo.gt 1000) then
         continue              ; go to end do and next iteration
      end if
      . . .statements. . .
end do
```

do: Tips and Errors

- NCL array subscripting (indexing) starts at **0** not **1**. Let x(**ntim**,…)

```
do nt=0,ntim-1                NOT  =>   do nt=1,ntim
   foo = func(x(nt,…)..)
end do
```

- Use **:=** syntax when arrays may change size within a loop

```
do yyyy=nyrStrt, nyrLast   ; loop over daily files (leap years+1 file)
   fili := systemfunc("ls 3B42_daily."+yyyy+".nc")  ; (365 or 366)
     q  := addfiles(fili, "r")
     p  := q[:]->rain                  ; (365 or 366, nlat,mlon)
end do
```

Else. if you had used the standard assignment **=** you would get the dreaded

```
fatal:Dimension sizes of left hand side and right hand
side of assignment do not match
```

- Prior to v6.1.1, variables had to be explicitly deleted

  delete(**[/** fili, q, p **/]**)

where

- **where** : based on conditional(s); return **merged** results
- Very useful; clean code

result = **where**(conditional(s), …True…, …False…)

x = **where**(x.gt.0, x, x+256)

z = **where**(oro.eq.1 .and. q.ne.0, a+273.15, 1.8*b+32)

Error messages(1)

```
fatal:Subscript out of range, error in subscript #0

    x = (/9,4,3,7,1/)      ; indices 0,1,2,3,4
    print(x(5))
```

fatal: Number of subscripts on right-hand-side do not match number of dimensions of variable: (4), Subscripts used: (3)

```
    x = random_uniform(-50,50,(/ntim,nlat,mlon/))
    y = x(0,:,:,:)
```

Warning: Assignment type mismatch, right hand side can't be **coerced** to type of left hand side

```
    i = (/ 2, 7, 33, 4 /)              ; integer
    f = (/ 3.7, 88.4, -12.3, 1.0/)     ; float
    i = f              ; trying to assign float to integer
    i:= f              ; reassign operator (i <= float <=f)
    i = toint( f ) ; i remains integer =>(/3,88,-12,1/)
```

Error messages(2)

```
fatal:Dimension sizes of left hand side and right hand
side of assignment do not match
```

fatal: ["Execute.c":8565]: Execute: Error occurred at or near line 258

```
        . . .statements. . .
   (258) x = funcfoo(…)
```

```
Use printVarSummary (print) to debug
Place before line 258
```

```
        printVarSummary(x)      ; ← reveal size, shape
        debug = func_foo(…)     ; temporary
        printVarSummary(debug)  ; ← reveal size, shape
        x = funcfoo(…)
```

Common Error Messages

http://www.ncl.ucar.edu/Document/Language/error_messages.shtml

Available under “Popular Links” and “Support”