NCL 学习参考手册

根据 NCL Mini-Language Reference Manual Version 1.1.6 翻译

有用的链接：
NCL 主页：http://www.ncl.ucar.edu/
本手册下载：http://www.ncl.ucar.edu/Document/Manuals/
脚本例子和样本图形：http://www.ncl.ucar.edu/Applications/
http://www.ncl.ucar.edu/Document/Manuals/Getting_Started

关键字	**courier-bold**
内置函数	**courier-bold blue**
用户贡献函数	**courier-bold green**
符号	**bold**
绘图模板	**courier-bold green**
绘图资源	**courier-bold**
用户变量	*italics*
WWW 链接	underline

致谢

衷心感谢上海海洋大学海洋科学学院胡松老师的悉心指导；感谢南京信息工程大学大气科学学院海洋科学系程军老师领我入门；感谢刘畅同学耐心细致的修订工作。

本人在学习 NCL 过程中，对手册进行了翻译，仅供方便学习查找使用。由于本人水平有限，本手册还存在许多不足，对一些专业名词略有模糊，望大家指正，可发邮件至：

l_n.2006@yahoo.com.cn

崔琳琳　　2010 年 7 月 31 日　　于上海海洋大学

目录

第一章 引言

NCAR 命令语言（NCAR Command Language，NCL）是一种专门为数据使用、分析、可视化而设计的编程语言。它的许多功能与现代编程语言相同，包括类型、变量、运算符、表达式、条件语句、循环、函数和过程。可以从下面的网站免费下载二进制代码，适用于多数常见 Linux/Unix、Mac OSX 和 Cygwin（Windows）：

http://www.ncl.ucar.edu/

1.1 设置用户路径

运行 NCL 之前，必须将你的 NCARG_ROOT 环境变量设置在安装 NCL 可执行文件及其附属文件的目录下，还必须确保存放 NCL 可执行文件的目录在你的搜索路径上，这些最好在你的主目录下的.*文件中完成。如果你不确定你在哪种 shell 下，可以在主目录下输入命令“ls –a”查看有哪种以“.”开始的文件，然后查看这些文件，检查环境变量是如何设定的。

例如：

C-shell（csh）：

```
setenv NCARG_ROOT /usr/local
set path=(/usr/local/bin $PATH)
```

bash:

```
set NCARG_ROOT=/usr/local
set PATH=(/usr/local/bin $PATH)
export NCARG_ROOT
export PATH
```

1.2 运行

NCL 可以在交互式模式或批处理模式下运行。NCL 区分大小写，所有声明需要以回车结束。

交互式模式：

```
ncl <回车>
 [输入声明语句]
[record “savefile”...stop record will save]
quit
```

批处理模式：批处理文件后缀通常以“.ncl”，这只是惯例，不是硬性要求。通过下面的方法可以调用 NCL：

```
ncl foo.ncl
ncl foo.ncl>&! foo.out    [C-shell]
ncl foo.ncl>&! foo.out  & [C-shell]
ncl < foo.ncl [acceptable]
```

NCL 支持命令行参数选项。使用“ncl –h”可显示当前支持的选项。第 6 章将详细讨论命令行参数。

NCL 函数、过程以及共享对象可以通过下面方法进入：

```
load "myfoo.ncl"
load "$HOME/myscripts/funct.ncl"
external DEMO"/tmp/Path/myfoo.so"
```

第二章 语言

2.1 符号

通常使用的系统符号包括:

;	该行为注释
@	参数属性
!	引用已命名维数
&	引用坐标变量
{…}	用于加下标的坐标
$	当通过 **addfile** 来输入或输出变量时嵌入字符串
[…]	类型列表的下标变量
(/…/)	构建一个数组
:	用于数组语法
|	作为已命名维数的分隔符
\	用于转行
::	调用外部代码的分隔符
->	用于支持的数据格式的输入/输出

2.2 数据类型

数值型：双精度 double（64 位）、单精度 float（32 位）、长整型 long（32 或 64 位）、整型 integer（32 位）、短整型 short（16 位）、字节 byte（8 位），不支持复型（complex）。
非数值型：字符串 string、字符 character、图形 graphic、文件 file、逻辑型 logical、列表 list

2.3 保留的关键字

`begin, break, byte, character, continue, create, defaultapp, do, double, else, end, external, false, file, float, function, getvalues, graphic, if, integer, load, local, logical, long, new, noparent, numeric, procedure, quit, Quit, QUIT, record, return, setvalues, short, string, then, True, undef, while`，以及所有的内置函数和过程名。

2.4 表达式

括弧“（）”为最高优先级。
NCL 不会对设为_FillValue 的数组元素进行操作（参见 2.13 节）。
代数运算符：

+	加
-	减
*	乘
^	幂
%	求整
#	矩阵乘
>,<	大于，小于

逻辑运算符：

.lt.	小于
.le.	小于或等于
.gt.	大于
.ne.	不等于
.eq.	等于
.and.	和
.or.	或
.xor.	异或
.not.	非

2.5 变量

变量名必须以字母开头但是可以包含数字和字母和下划线。变量可以有从属信息（通常称为元数据）。可以通过NCL函数和语法读取、生成、修改以及删除元数据（参见2.10-2.12节）。

通过NCL的**addfile**功能输入的变量可以自动获得任何与变量相关的元数据。

2.6 循环

```
do n=start,end,optional_stride(步长)
        [语句]
end do

do while (逻辑表达式)
        [语句]
end do
```

break:跳到 **end do** 的下一条语句
continue:直接进入下一次循环

任何编程语言使用的循环都应尽可能少。通常循环可以用数组或内置函数来取代。如果使用多级循环，就需要考虑运行速度，利用Fortran或C语言编写链接码是最好的方法。（参见第七章）

2.7 语句

```
begin
        [...]
end

if (逻辑表达式) then
        [...]
end if

if(逻辑表达式) then
        [...]
else
        [...]
end if
```

逻辑表达式是从左至右运算的，因此在多重表达语句中，将最可能出错的放在左边：

```
if (z.eq.3.and.all(x.gt.0)) then
        [...]
end if
```

2.8 维数和下标

NCL中有两种类型的数组下标：标准脚注和坐标脚注，基本功能和F90、Matlab、IDL相似。另外，NCL维数可能有名称，脚注范围是0：N-1。脚注格式如下：

起始:终止:步长

省略起始值时默认为0，省略终止值时默认为N-1，默认步长为1。“:”省略起始或终止值时表示所有元素。

下面的例子中，假定变量T是三维数组(nt,ny,nx)，名称是(time,lat,lon)。

2.8.1 标准脚注：可以用于任何数组变量

```
T                ;整个数组，不需要写成 T(:,:,:)
T(0,:,::5)       ;第一个时间值，所有纬度，每第 5 个经度值
T(0,::-1,:50)    ;第一个时间值，纬度反向，第 0-50 个经度值
T(:1,45,10:20)   ;前两个时间值，第 46 个纬度值，第 10-20 个经度值
```

2.8.2 坐标脚注：用于符合NetCDF数据的维数。根据定义，坐标变量必须是单调递增或递减的一维数组，且变量名和维数名称一致。如：

```
X = T(:,{-20,20},{90:290:2})
```

（所有时间，纬度-20 至 20，经度 90 至 290，步长 2）

2.9 维数简化

当数组下标中给定常量，便发生维数简化。假定*T*(nt,nz,ny,nx)，那么：

```
T1 = T(5,:,12,:);得到 T1(nz,nx)
T2 = T(:,:,:,0) ;得到 T2(nt,nz,ny)
```

所有正确的元数据都被复制了，NCL忽略简化的维数大小。因此用户可以强制保留简化的维数：

```
T3 = T(5:5,:,12,:)      ;T3(1,nz,nx)
T2 = T(5:5,:,12:12,:)   ;T4(1,nz,1,nx)
```

2.10 命名的维数

命名的维数用来调整数组大小，坐标变量要求命名维数。!符号用来建立维数名称或者获取维数名称。维数编号从左到右，最左边的为0。假定*T*是三维数组(ntime,nlat,nlon)：

分配：

```
T!0   = “time”
T!1   = “lat”
T!2   = “lon”
```

获取：

```
LAT  = T!1       ;LAT = “lat”
```

2.10.1 维数重新排序：只有需要重新排列维数时才使用命名的维数。

```
reordered_T = T(lon|:,lat|:,time|:)
```

命名的维数不是脚注，但是，命名的维数可以同坐标脚注和标准脚注一起使用。

```
X = T({lat|-20:20}，lon|30:42，time|:)
```

（重新排序为(lat,lon,time)并选择纬度为-20:20，经度为30:42，所有时间值）

2.11 坐标变量

根据netCDF定义，坐标变量必须是单调递增或递减的一维数组，变量名和它分配的维数名称是一样的（如time(time)）。坐标变量代表了维中每个索引的数据坐标，可以用于坐标脚注。符号&用来引用和分配坐标变量。为了分配给坐标变量一个维数，维数必须有一个与之相关的名称：

```
T!0    = “lat”
T!1    = “lon”
T&lat  = (/-90.,-85.,...,85.,90./)
T&lon  = fspan(0.,355.,72)
```

2.12 属性

属性是与存在的变量相关的描述性信息。变量属性的表示方法为变量名+@+属性名：

```
T@units       = “Degrees C”
T@_FillValue  = -9999.0
T@wgt         = (/.25,.50,.25/)
W             = T@wgt
```

属性可以用于表达式和下标：

```
T = TS * TS@scale_factor + TS@add_offset
```

2.13 _FillValue

_FillValue属性为netCDF和NCL的保留属性名，表示缺少的值。如果你的数据有一个“missing_value”属性但是没有**_FillValue**属性，你可以分配：

```
x@_FillValue = x@missing_value
```

2.14 强制型转

强制型转是一种将数据由一种类型转为另一类型的隐式变换，这发生在两个不同类型的值由同一个运算符运算的情况下。例如：

```
X = 5.2 + 9
```

这里5.2是浮点型，9是整型，这时，9被强制转换为浮点型然后再相加。NCL能在信息不丢失的情况下强制转换。如果*K*是整型，*X*是浮点型（或双精度型），下面的式子将会出错：

```
K = X
```

当可能出现信息丢失时，必须使用明确的转换函数：

```
K = floattointeger(X)
```

双精度类型的变量可以用“d”格式进行显式创建：

```
X = 23431234.0d
```

其它类型的转换函数可以从这个网址找到：

http://www.ncl.ucar.edu/Document/Functions/type_convert.shtml

2.15 变量和元数据

NCL有两种分配类型，值分配和变量对变量的分配，只有后者复制元数据。

当分配的右边不是变量时进行**值分配**。右边可以是常量，表达式的值，构造数组的值（/…/）。维的名称、坐标变量或属性不分配，缺省值分配。如果表达式的右边不包含任何缺省值，那么_FillValue不分配。如：

```
a = (/1,2,3,4,5,6,7,8,9,10/)
q = w * sin(z) + 5
```

```
b = 19911231.5d   ; 双精度
```

如果左边先于分配语句被定义，那么右边的值将分配给左边。如果左边是变量的下标引用，那么右边的元素将被映射到坐标变量的适当位置。如果左边有属性、维名称、坐标变量，他们将被完全保留，因为只有数值被分配到左边变量。例如，假定*T*是变量，它有名称、坐标数组和属性。而且，*T*包含温度数据，有单位：

```
T@units = “degC”
```

表示单位是摄氏度。转换为开氏度是值分配，因为右边是一个表达式：

```
T = T + 273.15
```

T将会保留元数据包括单位。在这种情况下，用户须改变单位属性：

```
T@units = “degK”
```

坐标对坐标的分配是指所有的属性、坐标变量、维名称以及真实值都被分配。如果*y*不存在，那么 *y = x* 将*x*完全复制给*y*，包括元数据。如果之前定义了*y* ，那么*x*必须是相同类型的（或者转换成*y*的类型），而且x必须和y具有相同的维数。另外，如果*y*有元数据，那么*x*的元数据将取代*y*的元数据。需要注意的是，构造数组(/…/)能够用来将一个变量分配给另一个变量，它和值对变量的分配一样，只分配数值，而忽略属性、维数和坐标变量。如：

```
X = (/y/)
```

第三章　NCL 文件输入/输出

3.1 支持的格式

NCL支持的文件格式有：netCDF3/4, HDF4, HDF4-EOS,GRIB-1,GRIB-2和CCM History Tape格式。网上讨论区：

http://www.ncl.ucar.edu/Applications/list_ioshtml

建立文件引用：

函数**addfile**能够用来输入所有支持的文件：

```
f = addfile (fname.ext, status)
```

*f*是一个参考或者指向一个文件。*fname*是数据文件的全路径或相对路径。支持的格式有不同的文件后缀名：netCDF 文件为“.nc”、“.cdf”，HDF4文件为“.hd”、“.hdf”，HDF4-EOS文件为“.hdfeos”，GRIB-1\GRIB-2文件为“.grb”、“.grib”，CCM History Tape文件为“.ccm”。

```
status：“r”    （读：所有支持文件）
        “c”    （创建：netCDF 和 HDF4）
        “w”    （读、写：netCDG 和 HDF4）
```

如：

```
grb     =  addfile(“/my.grib/foo.grb”,”r”)
f       =  addfile(“foo.nc”,”r”)
hdf     =  addfile(“/your/hdf/foo.hdf”,”c”)
h       =  addfile(“foo.hdfeos”,”r”)
```

```
ccm      =  addfile("foo.ccm","r")
```

文件不一定要扩展名，如，如果文件为“foo.grb”，那么，**addfile**将首先查找该文件。如果文件不存在于指定的名称下，**addfile**将查找名为“foo”的文件，将它作为GRIB文件。

读变量：

如果*X*是*f*所指向文件中的变量，那么：

```
x = f->X
```

如果变量元数据（属性、坐标变量）可用，它们将会自动分配给变量*x*。

何时使用$:

两种情况下要使用$：1）文件的变量名中有非字母的字符：

```
x = f->$"ice cream+oreo-cookies…yummy!"$
```

2)指向符号->右边的项本身是字符串变量类型：

```
vars = (/"T","U","V")
x = f->$vars(n)$      ;n=0,1,2
```

打印所支持格式文件的内容：

print可用于查看文件内容。打印的信息和“*ncdump* –h foo.nc”生成的内容相似。如：

```
f = addfile ("foo.grb","r")
print(f)
```

注意：NCL命令行工具**ncl_filedump**能够输出文件内容摘要。详见：

http://www.ncl.ucar.edu/Document/Tools/ncl_filedump.shtml

输出时重新排序变量：

假定*X*是三维变量(ntime, nlat, nlon)，使数组中的纬度反向：

```
X = f->X(:,::-1,:)
```

输入字节和短整型数据：

NCL程序库中几个函数可以将短整型short和字节byte转换为浮点型float：

```
x = short2flt（f->X）    ;也可写成  x = short2flt_hdf（f->X）
y = byte2flt(f->Y)
```

为了使用这些函数，“contributed.ncl”程序库必须在使用前加载：

```
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
```

多文件处理：

函数**addfiles**（注意多出s）能够存取多个文件的数据。如：

```
fils = systemfunc("ls ann*.nc")
f    = addfiles(fils,"r")
T    = f[:]->T
```

这里合成的变量*T*将会跨度所有文件。

详见：

http://www.ncl.ucar.edu/Document/Functions/Built-in/addfiles.shtml

http://www.ncl.ucar.edu/Document/Functions/Contributed/addfiles_GetVar.shtml

改变addfile和addfiles的默认性质：

setfileoption程序能够改变**addfile**的默认性质。详见：
http://www.ncl.ucar.edu/Document/Function/Built-in/setfileoption.shtml

3.2 二进制数据文件

读取二进制文件：

fbinrecread(*path*:string,*recnum*:integer,*dims*[*]:integer,*type*:string)能够用来读取Fortran无格式的顺序文件。以0开头的记录可能有不同的类型和长度。

```
x = fbinrecread("f.ieee",0,(/64,128/),"float")
```

fbinrecread(*path*:string,*rnum*:integer,*dims*[*]:integer,*type*:string)能够用来读取固定记录长度二进制记录（直接存取）。文件中所有记录必须具有相同的维数和类型。读取第（*n*+1）个记录：

```
x = fbinrecread("f.ieee",n,(/73,144/),"float")
```

其它读取二进制数据的函数还有：**cbinread**, **fbinread**, **craybinrecread**.

写二进制文件：

fbinrecwrite(*path*:string,*recnum*:integer,*value*)能够用来写Fortran无格式的顺序文件。记录可能有不同的类型和长度。

假定有下面五个变量time(ntime), lat(nlat), lon(nlon), y(ntime,nlat,nlon), z(nlat,nlon)。注意使用-1作为记录号表示追加记录。

```
fbinrecwrite("f.ieee",-1,  time)
fbinrecwrite("f.ieee",-1,  lat)
fbinrecwrite("f.ieee",-1,  lon)
fbinrecwrite("f.ieee",-1,  y)
fbinrecwrite("f.ieee",-1,  z)
```

fbindirwrite(*path*:string,*value*)能够用来写Fortran直接存取文件。所有的记录必须具有相同的长度和类型。

```
do n=0,ntim-1
   fbindirwrite("/my/path/f.ieee',y(nt,:,:))
end do
```

其它写IEEE二进制数据的子程序有：**cbinwrite** **fbinwrite**

3.3 ASCII

读取ASCII文件:

asciiread(*filepath*:string,*dims*[*]:integer,*datatype*:string)能够使数据在输入时即可成型。复杂的ASCII文件需要通过C或Fortran语言的子程序读取。

```
z = asciiread("data.asc",(/100,13/),"float")
```

*z*是一个100行和13列的浮点型变量。

写ASCII文件：

asciiwrite(filepath:string,value)输出一列值，且用户不能控制格式。

```
asciiwrite("foo.ascii",x)
```

write_matrix(data[*][*]:mumeric,fmtf:string,option)能输出多组值，而且用户可以控制格式。

```
fmtf       = "15f7.2"   ;用 Fortran 表示法表示
opt        = True
opt@fout   = "foo.ascii'
write_matrix(x,fmtf,opt)
```

3.4 写 netCDF/HDF 文件

NCL有两种写netCDF/HDF文件的方法，一种是简单方法，另一种是传统方法。

简单方法：

直接用“.nc”替代“.hdf”就可以生成HDF文件：

```
fo = addfile("foo.nc","c")
fo->X = x
fo->Y = y
```

要生成一个无限制的维，通常是time，就要在输出值之前加上下面一行代码：

```
filedimdef(fo,"time",-1,True)
```

传统方法：

该方法要求用户在输出之前明确定义整个文件的内容。预先定义netCDF文件：

filevardef:	定义变量名
filevarttdef:	将一个变量的属性复制给一个或多个文件变量
filedimdef:	定义维
fileattdef:	将一个变量属性复制给一个文件作为全局属性
setfileoption	一些选项能显著改进性能

在下面的例子中，假定变量time,lat,lon,T存在于存储器内。当写netCDF文件时，变量T被命名为TMP.

```
fout = addfile("out.nc","c")

; create global attributes (生成全局属性)
fileAtt                  = True
fileAtt@title            = "Sample"
fileAtt@Conventions      = "None"
fileAtt@creation_date    = systemfunc("date")
setfileoption(fout,"DefineMode",True)        ; optional
fileattdef(fout,fileAtt)

; predefine coordinate variables （预先定义坐标变量）
dimNames = (/"time","lat","lon"/)
dimSizes = (/-1,nlat,nlon/) ; -1 means unspecified
dimUnlim = (/True,False,False/)

; predefine names, type, dimensions （预先定义名称，类型，维数）
; explicit dimension naming or getvardims can be used （明确维名称，
或者可以用getvardims命令）
filedimdef(fout,dimNames,dimSizes,dimUnlim)
filevardef(fout,"time",typeof(time),getvardims(time))
filevardef(fout,"lat" ,typeof(lat) ,"lat")
filevardef(fout,"lon" ,"float" ,"lon")
filevardef(fout,"TMP" ,typeof(T) , getvardims(T) )

; predefine each variable's attributes （预先定义每个变量的属性）
filevarattdef(fout,"time",time)
filevarattdef(fout,"lat" ,lat)
filevarattdef(fout,"lon" ,lon)
filevarattdef(fout,"TMP" ,T)
setfileoption(fout,"SuppressDefineMode",False) ; optional

; output values only   （输出值）      [use (/… /) to strip metadata]
fout->time          = (/time/)
fout->lat           = (/lat/)
fout->lon           = (/lon/)
fout->TMP           = (/T/) ; T in script; TMP on file
```

向netCDF写入标量：

NCL利用保留维数名称“ncl_scalar”来确定要写入netCDF文件的数值。

```
; simple method
fo                  = addfile("simple.nc","c")
con                 = 5
con!0               = "ncl_scalar"
```

```
fo->constant        = con

; traditional method
re                  = 6.37122e06
re@long_name        = "radius of earth"
re@units            = "m"

fout = addfile("traditional.nc", "c")
filevardef(fout,"re",typeof(re),"ncl_scalar")
filevarattdef(fout,"re",re)
fout->re = (/re/)
```

写好netCDF/HDF文件：

任何文件都必须有的全局文件属性包括标题、常规和原始资料。其它文件属性包括history, references等。

用命令行转换所支持的格式为netCDF文件：

ncl_convert2nc能够将GRIB-1,GRIB-2,HDF4,HDF4-EOS格式转换成netCDF格式。详见：
http://www.ncl.ucar.edu/Document/Tools/ncl_convert2nc.shtml

3.5 远程文件访问：OPeNDAP

一些数据服务器通过OPeNDAP（**Ope**n Source Project for **N**etwork **D**ata **A**ccess **P**rotocol）提供远程文件访问：

```
f     = addfile ("http://path/foo.nc", "r")
```

或

```
fils  = "http://path/" + (/ "foo1.nc", "foo2.nc", "foo3.nc"/)
f     = addfiles(fils, "r")
```

用户可以用**isfileprensent**命令检查文件的可用性。需要注意的是，如果你安装了防火墙，你也许不能通过该方法获得数据。而且，一些OPeNDAP服务器要求用户先注册。

第四章 输出

NCL的打印功能有限，在一些例子中，最好调用Fortran或C语言来控制数据格式。可用的函数和过程如下：

printVarSummary	输出变量概要，包含所有元数据
print	与**printVarSummary**相同，后接每个元素的值
sprinti, sprintf	可以控制格式

write_matrix　　以表格形式输出

4.1 printVarSummary

用法　**printVarSummary**(*u*)

Variable: u
Type: double
Total Size: bytes
　　147456 values
Number of Dimensions: 4
　　Dimensions / Sizes: [time | 1] x [lev | 18] x
　　[lat | 64] x [lon | 128]
Coordinates:
　　time: [4046..4046]
　　lev: [4.809 .. 992.5282]
　　lat: [-87.86379 .. 87.86379]
　　lon: [0. 0 .. 357.1875]
Number of Attributes: 2
　　long_name: zonal wind component
　　units: m/s

4.2 print

用法 **print(***u***)**
输出信息基本与**printVarSummary**相同，而且有每个元素的值和相关脚注：
　　(0,0,0,0)　31.7
　　(0,0,0,1)　31.4
　　(0,0,0,2)　32.3　[snip]
调用**ncl –n** foo.ncl能够避免输出脚注。

```
print(u(0,{500},:,{100}))
```

将输出变量*u*在第0时次，接近500层，经度接近100处的所有纬度。

```
print("min(u)="+min(u)+" max(u)="+max(u))
```

输出结果为如下字符串：
　　min(u)= -53.8125 max(u)=55.9736

4.3 sprintf, sprinti

```
print("min(u)="+sprintf("%5.2f",min(u)))
```

结果：

```
min(u) = -53.81
```

```
ii=(/-47,3579,24680/)
print(sprinti("%+7.5i",ii))
```

结果：

```
-00047, +03579, +24680
```

4.4 write_matrix

若有*T*(*nrow*,*ncol*)，其中*nrow* = 3，*ncol* = 5，则

```
write_matrix(T,"5f7.2",False):
```

```
 4.36   4.66    3.77 -1.66   4.06
 9.73  -5.84   0.89   8.46  10.39
 4.91   4.59  -3.09   7.55   4.56
```

注意：虽然**write_matrix**原为用于二维数组，但是可以用**ndtooned**和**onedtond**来输出多维。假定*X*(nt,nz,ny,nx)：

```
dimx = (/nt,nz*ny*nx/) 或
dimx = (/nt*nz,ny*nx/)
write_matrix(onedtond(ndtooned(X),dimx),"12f8.3",False)
```

第五章 数据分析

NCL提供不同的数据分析方法：（1）数组语法，（2）内部函数，（3）用户提供的函数，（4）调用Fortran和C语言。详见：

http://www.ncl.ucar.edu/Ducument/Mamuals/Ref_Manual/NclUsage.shtnml

5.1 数组语法

NCL中的代数运算提供数量运算和数组运算。对于数组运算，每个操作量必须有相同的维数和大小，否则会出错。而且，每个操作量的数据类型必须相同，或者将一个强制转换成与其它相同的类型。令A、B为(10,30,64,128)：

```
C = A+B        ;逐个元素相加
```

```
D = A-B        ;逐个元素相减
E = A*B        ;逐个元素相乘
```

如果C、D、E之前不存在将会自动生成。如果之前存在，那么右边操作的结果类型必须强制转换成左边的类型。

当数值存在一个复杂的数组运算的表达式中时，数值将会参加数组中每个元素的操作。如：

```
F = 2*E + 5
```

这里数组*E*中每个元素将会乘以2然后加上5，结果将会分配给*F*。

NCL中< and >的特殊用法，假定*sst*是(100,72,144)，*sice* = -1.8(一个数值)，那么：

```
sst = sst >  sice
```

表示*sst*中任何小于*sice*的值将会被*sice*代替。

所有数组表达式自动忽略与被设为**_FillValue**的值有关的操作。

5.2 数组一致性

数组表达式要求所有操作量具有相同的维数和大小。在这种情况下，数组要求一致性。数值符合任何数组的形式。假定*T*和*P*的维数是(10,30,64,128)：

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

*theta*的结果为(10,30,64,128)。**conform**和**conform_dims**能够用来生成符合其它数组的数组。假定*T*是(10,30,64,128)，*P*是(30)：

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

conform扩展了*P*，详见：

http://www.ncl.ucar.edu/Document/Functions/Built-in/conform.shtml
http://www.ncl.ucar.edu/Document/Functions/Built-in/conform_dims.shtml

5.3 数组内存分配

有两种方法可以分配数组内存：

1）利用数组构造函数（/.../）

```
a_integer   = (/1,2,3/)
a_float     = (/1.0, 2.0, 3.0/)
a_double    = (/4d0,5d-5,1d30/)
a_string    = (/"a","b","c"/)
a_logical   = (/True,False,True/)
a_2darray   = (/(/1,2/),(/5,6/),(/8,9/)/) ; 3行x2列
```

2）利用**new**(array_size,shape,type,[_FillValue])语句：这里**_FillValue**是可选项。如果它不存在，那么将会分配默认值。指明**No_FillValue**将不会分配默认**_FillValue**值。

```
                                                    ; _Fillvalue
```

```
a = new(3,float)                          ; 999.
b = new(10,float,1e20)                    ; 1e20
c = new((/5,6,7/),integer)                ; 999
d = new(dimsizes(U),double)               ; 9999.
e = new(dimsizes(ndtooned(U)),logical)    ; -1
s = new(100,string)                       ; “missing”
q = new(3,float,”No_FillValue”)           ; no _FillValue
```

函数会自动为返回变量分配内存，因此，通常不需要用**new**语句。

5.4 函数和过程

NCL既有函数也有过程，用户需要注意它们的区别：

1）函数是表达式，因为它们返回的是一个或多个值，而且可以被作为表达式的一部分。如**max**, **sin** 和 **exp** 是标准数学函数：

```
z = exp(sin(max(q))) + 12.345
```

函数不要求返回相同类型和大小的数组。

2）过程（类似于Fortran子例行程序）不能是表达式的一部分，因为它们返回的不是数值。

NCL函数和过程的参数：

参数是通过引用来传递的。这就意味着改变一个函数或过程中参数的值、属性、维名称和坐标变量将会改变主程序中的值。按照惯例，函数参数不应该被函数改变。下面的讨论将会遵从这一惯例。

参数定义：

在NCL中，函数和过程的参数被限制，需要特定类型、维数和大小，或者参数可以没有类型和维数限制。

（a） 限制参数是指参数要求具有指定的类型、大小和形式：

```
procedure ex (x[*][*]:float,y[2]:byte,\
res:logical,text:string)
```

参数*x*是一个二维数组，浮点型，参数*y*是一维数组，长度为2，*res*和*text*分别是逻辑型和字符型，维数任意。

（b） 一般类型：

```
function xy_interp(x:numeric,y:numeric)
```

（c） 类型、大小、维形式都不指定：

```
procedure foo(a,b,c)
```

（d） 组合：

```
function ex(d[*]:float,x:numeric,wks:graphic,y[2],a)
```

当用户给过程设定参数时需要注意一个很重要的问题，这就是仅仅当过程需要修改输入参数时，当输入参数必须被改成当前程序类型时，NCL不能反向强制转换，使得过程中对参数进行改变时参数不受影响，并且NCL会发出错误信息。

5.5 内部函数和过程

NCL包含许多内部函数和过程，详见：

http://www.ncl.ucar.edu/Document/Functions/

函数能够返回数组，而且没有必要预先分配数组内存。例如，G是一个四维数组(ntime,nlev,73,144),将G插入一个128个经度、64个纬度且在卷曲42有三角形截断的高斯型网格中，将会用到下面的内部函数：

```
g = f2gsh(G,(/64,128/),42)
```

f2gsh能够执行所有时间和层次上的内插。数组*g*将会自动生成且大小为(ntime,nlev,64,128)

通常内置函数不返回、生成、改变元数据。但是，许多常用的内置函数，在名为“contributed.ncl”的用户提供的函数中带有“**_Wrap**”版本，能够正确处理元数据。

http://ww.ncl.ucar.edu/Document/Functions/Contributed/

函数和过程要求维的特定参数按照指定的顺序存放。在此情况下，就要使用命名维数来重新排列维中的参数。例如**dim_***这套函数，所有这些函数均对最右边的量进行操作。假设*T*是四维变量(ntime,nlev,nlat,nlon),名称(time,lev,lat,lon)。不重新排序时，**dim_avg**将会计算纬向平均，因为它是对经度进行处理的。

```
Tzon = dim_avg(T)
```

结果为(ntime,nlev,nlat)。如果将变量重新排序，时间放在最右边，则是对时间求平均。

```
Ttim = dim_avg(T(lev|:,lat|:,lon|:,time))
```

结果为(nlev,nlat,nlon)。

通常，函数不要求明确分配内存。假定*T*包含10年的月平均资料，为了计算每月的标准偏差，就需要预先分配返回值的内存，因为计算产生了循环。

```
; preallocate array
nmos = 12
Tclm = new((/nlat,nlon,nmos/),typeof(T),T@_FillValue)
Tstd = Tclm ; trick
Ttmp = T(lat|:,lon|:,time|:)) ; reorder
ntime = dimsizes(in->time) ; get size of time

do n=0,nmos-1
Tclm(:,:,n) = dim_avg(Ttmp(:,:,n:ntime-1:nmos))
Tstd(:,:,n) = dim_stddev(Ttmp(:,:,n:ntime-1:nmos))
end do
```

给过程预先分配数组:

如果一个过程返回一个或多个参数，就需要预先分配返回变量的内存。例如**uv2sfvpg**，它读取纬向风速u、经向风速v，然后返回流函数（psi）和势函数（chi）。返回的数组必须和风速参数就有相同的大小和类型：

```
psi = new(dimsizes(u),typeof(u))
chi = new(dimsizes(u),typeof(u))
uv2sfvp(u,v,psi,chi)
```

函数嵌入：

函数本身就是表达式，因此它们能组成表达式。但是，嵌入函数的可读性很重要。如：

```
X = f2gsh(fo2fsh(fbinrecread(f,6,(/72,144/),"float")),(/nlat,lon/),42)
```

分开写能够很容易读懂：

```
G = fbinrecread(f,6,(/72,144/),"float")
X = f2gsh(fo2fsh(G),(/nlat,mlon/),42)
```

最常用的内置函数有：**all**, **any**, **conform**, **conform_dims**, **ind**, **ind_resolve**, **dimsizes**, **fspan**, **ispan**, **ndtooned**, **onedtond**,**mask**, **ismissing**, **system**, **systemfunc**, **print**, **printVarSummary**和**where**。

详见：

http://www.ncl.ucar.edu/Document/Functions/array_manip.shtml

http://www.ncl.ucar.edu/Document/Functions/array_query.shtml

5.6 contributed.ncl

NCL中包含一个使用者提供的函数库。为了使用该库，必须在脚本开头加载contributed.ncl：

load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"

5.6.1 Wrappers

“contributed.ncl”脚本为许多NCL内置函数提供Wrappers。Wrapper函数用于关联元数据和返回变量。Wrappers与对应的函数具有相同的调用序列，而且Wrapper的名称和函数的一样，只是加了后缀("_**Wrap**")。

使用dim_avg会失去T的元数据：

```
Tzon = dim_avg(T)
```

使用**dim_avg_Wrap**会保留元数据：

```
Tzon = dim_avg_Wrap(T)
```

可用的wrappers一览表：

http://www.ncl.ucar.edu/Document/Functions/Contributed/

copy_VarAtts, **copy_VarCoords**, **copy_VarMeta**, **copyatt**等函数由用户提供，用于显式复制坐标变量或属性。

5.6.2 类型转换

能够将一种类型转换为另一种类型且保留元数据的“contributed.ncl”函数有：**short2flt** , **byte2flt**, **short2flt_hdf**, **numeric2int**和**dble2flt**.

5.6.3 气候函数

contributed.ncl气候函数和距平函数，也能生成适当的元数据，例如：**clmMon*()**，**stdMon*()**，**month_to_month**等。

5.7 自定义函数

自定义函数的一般结构是：

```
undef("function_name")
function function_name(declaration_list)
local local_variable_list
begin
      [statement(s)]
return (return_value)
end

undef("procedure_name")
procedure procedure_name(declaration_list)
local local_variable_list
begin
      [statement(s)]
end
```

undef过程的作用是使之前定义的函数或过程失效。**local**语句列出当前函数或过程的变量。使用**undef**和**local**并非必需，但建议使用。
注意：所有的用户自定义的函数或过程在使用前必须下载。它们可以在同一文件中，也可以在其它地方。

```
load "./myFoo.ncl"
```

可选参数：

用户也会在过程和函数中插入可选参数。按照惯例，这要通过把属性附加到逻辑型变量来完成。高级图形函数和过程都使用这种方法选择参数。可以使用NCL的内部**is***函数查看这个参数（如**isatt**）。

```
opt         = True
opt@scale   = 0.01
opt@add     = 1000
opt@wgts     = (/0.25,0.50,0.25/)
opt@a3d    = array_3D

undef ("foo")
function foo(x:numeric,opt:logical)
local dimx,rankx,xx
begin
```

```
        dimx = dimsizes(x)
        rankx = dimsizes(dimx)
        if(typeof(x).eq."short")then
           if(opt.and.isatt(opt,"scale"))then
              xx = x*opt@scale
           else
              xx = x
           end if
        else
           return(x)
        end if
end
```

5.8 系统交互功能

用户可以通过**systemfunc**和**system**实现系统交互功能。基本上，用户创建包含Unix命令的字符串。分号可以分隔多个Unix命令。在字符串中使用单引号，可使用Unix命令参数。

Systemfunc能够执行系统命令，且将信息返回给NCL变量。

```
files_full_path = systemfunc("ls /my/data/*.nc")
files_names = systemfunc("cd /my/data ; ls *.nc")
date = systemfunc("date")
```

运用Unix的“cut”命令提取ASCII文件的14-19列，返回值是一个字符串型的一维数组，然后转换成浮点型。

```
x = stringtofloat(systemfunc("cut –c14-19 sample.txt"))
```

system允许用户执行，但是它不返回任何信息。如：

```
system("cp 10.nc /ptmp/user/") ; 复制文件
system("sed ‘s/NaN/-999./g’ "+asc_input+" > asc_output")
```

在下面的例子中，1995年的所有文件都来自NCAR的大容量存储系统，放在目录/ptmp/user/下：

```
MSS = "/USER/mss/path"
dir = "/ptmp/user/"
year = 1995

system("msrcp –n ’mss:" +MSS+nyear+ "-*.nc’" + dir + ".")
```

默认情况下，当NCL传递命令给系统时，它调用bsh。下面的例子是用bsh创建一个目录：

```
DIR = "SAMPLE"
system("if ! test -d "+DIR+" ; then mkdir "+DIR+" ; fi")
```

用户可能对其他的UNIX shell更为熟悉。下面是用csh创建相同的目录：

```
system("csh -c 'if (! -d "+DIR+") then ; mkdir "+DIR+" ; endif'")
```

为了防止bsh试图解释csh语法，命令用单引号。如果csh包含单引号，就用“\”。

第六章 命令行选项和分配

NCL提供有限的命令选项，以及简单的NCL 语句的设置和执行。详见：
http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclCLO.shtml

6.1 调用 NCL 时选项变更

下面是一些预先定义的选项：

-h 显示命令行选项的用法
-n 不列举print()中的值
-x 回显输入的NCL语句
-V 输出NCL版本及退出

*-x*选项用法的一个简单例子：

```
% ncl -x
  Copyright (C) 1995-2005 - All Rights Reserved
  University Corporation for Atmospheric Research
  NCAR Command Language Version 4.2.0.a033
  The use of this software is governed by a License Agreement.
  See http://www.ncl.ucar.edu/ for more details.
ncl 0> a = 5
+ a = 5
ncl 1> exit
+ exit
```

6.2 在命令行指定变量任务

调用NCL时在命令行创建变量能够帮助处理数据。命令行参数的一些简单例子：

```
ncl nyrStrt=1800 nyrLast=2005 foo.ncl
ncl 'f="test.nc"' p=(/850,500,200/) 'v=(/"T","Q"/)' foo.ncl
```

空格不允许使用。字符串必须用单引号。
脚本可以包含变量的默认设置：

```
ncl a=5 c=3.14d0 foo3.ncl
```

foo3.ncl脚本能够通过**isvar**函数检查命令行的变量：

```
if(.not.isvar("a")) then
    a = 10                          ; 若未定义则设为10
end if
if(.not.isvar("b") then
    b = 72.5                        ; 若未定义则设为72.5
end if
```

在Unix shell下调用命令行参数是一件麻烦事。命令行：

```
ncl 'filName="foo.nc"' test.ncl
```

在csh中应该写成：

```
#!/bin/csh –f
set a=foo.nc
eval ncl filName=\\\"$a\\\" test.ncl
```

6.3 ncl_filedump

这个命令操作将会生成一个指定文件的文本描述。文件可以是任何支持的格式：netCDF, GRIB-1, GRIB-2, HDF和HDF-EOS。默认输出是到屏幕上，也可以指定一个文件。忽略输入文件的格式，输出的文本格式和“ncdump –h”产生的类似。

```
ncl_filedump [options] fname.ext
```

函数**addfile**可以用来输入任何支持格式的文件。文件不一定要有扩展名。如文件“foo.grb”，**ncl_filedump**将开始查找文件。如果文件不存在于指定的名称下，**ncl_filedump**查找名为“foo”的文件，将它作为GRIB文件。

ncl_filedump –h可以查看选项，详见：

http://www.ncl.ucar.edu/Document/Tools/ncl_filedump.shtml

6.4 ncl_convert2nc

这个命令行参数会将GRIB-1, GRIB-2, HDF, HDF-EOS文件转换成netCDF格式。

```
ncl_convert2nc fname(s) [options]
```

注意选项指定在文件名后。**ncl_convert2nc** –h可以查看选项，详见：

http://www.ncl.ucar.edu/Document/Tools/ncl_convert2nc.shtml

第七章　使用外部代码

用C语言编写的NCL可以调用外部代码。外部C语言函数的使用详见：

http://www.ncl.ucar.edu/Document/Manuals/Ref_Manual/NclExtend.shtml

7.1 NCL/Fortran 界面

随NCL一起发行的**WRAPIT**程序促进了Fortran子函数的使用。输入“**WRAPIT –h**”可以查看可用选项。**WRAPIT**编译外部编码然后生成一个称为“共享对象”的文件，文件的扩展名为“.so”。

WRAPIT需要的唯一信息就是Fortran和NCL之间的接口，包括子程序声明语句和参数。不需要明确指

定参数类型，因为**WRAPIT**能够识别Fortran的默认类型。用户可以修改Fortran语句来重新定义默认类型。NCL使用定界符：

```
C NCLFORTSTART
C NCLEND
```

来证明接口部分。注意，定界符是f77注释形式，因此不会影响编码。**C NCLFORTSTART**在子程序之前，**C NCLEND**在参数声明之后。

```
C NCLFORTSTART
      subroutine demo(xin,xout,mlon,nlat,text)
      integer mlon, nlat
      real xin(mlon,nlat), xout(mlon,nlat)
      character*(*) text
C NCLEND
```

7.2 f77 子程序

生成和调用共享对象有四步，下面以一个例子来说明。现有一个名为foo.f的文件，这个文件可能包含一个或多个f77子程序。

1）用分界符将每个子程序分隔开

```
C NCLFORTSTART
      subroutine demo(xin,xout,mlon,nlat,text)
      integer mlon, nlat
      real xin(mlon,nlat), xout(mlon,nlat)
      character*(*) text
C NCLEND
```

其余的Fortran代码可能包含很多子程序。

2）用**WRAPIT**生成共享对象，默认生成与Fortran文件同名的.so文件（例如foo.so）：

```
WRAPIT foo.f
```

3）在NCL脚本中加入**external**语句。**external**语句包含任意标识符，NCL用它来选择和定位正确的共享对象。默认位置为当前目录。

```
external FOO "./foo.so"
```

4）从NCL中调用指定的子程序。子程序必须包括三部分（a）NCL指定的目标共享对象的名称（b）分隔符“::”（c）Fortran子程序接口。

```
FOO::demo(xin,xout,nlon,nlat,text)
```

NCL脚本示意如下：

```
external FOO "./foo.so"
begin
      [statement(s)]
      xout = new((/nlat,nlon/),typeof(xin)
   FOO::demo(xin,xout,mlon,nlat,text)
      [statement(s)]
end
```

7.3 f90 子程序

调用f90子程序的过程除了第一步外其它和f77一样。在f77中，NCL分隔符直接插入子程序。然而**WRAPIT**使用的Fortran解析器不能识别f90语法。因此，用户必须为每个子程序创建一个“stub”的分隔符。Stub文件是f90声明的重复。Stub文件可以不是完整的子程序。**WRAPIT**只关心子程序的调用及它的参数。例如以下是包含在quad.f90文件里的子程序：

```
      subroutine cquad(a,b,c,nq,x,quad)
         implicit none
         integer, intent(in)    :: nq
         real, intent(in)       :: a, b, c, x(nq)
         real, intent(out)      :: quad(nq)
         integer                :: i ! local

         quad = a*x**2 + b*x + c   ! f90 array syntax
         return
      end subroutine cquad

      subroutine prntq(x,q,nq)
         implicit none
         integer, intent(in)    :: nq
         real, intent(in)       :: x(nq),q(nq)
         integer                :: i ! local
         do i = 1,nq
            write(*,'(I5, 2F10.3)')i,x(i),q(i)
         end do
         return
      end subroutine prntq
```

1）用f77语法创建stub文件，保存在quad90.stub文件中。每个stub文件要求一套**C NCLFORTSTART**和**C NCLEND**分隔符。

```
      C  NCLFORTSTART
         subroutine cquad (a,b,c,nq,x,quad)
         dimension x(nq),quad(nq) ! ftn default
      C  NCLEND
      C  NCLFORTSTART
         subroutine prntq(x,q,nq)
         integer nq
         real x(nq),q(nq)
      C NCLEND
```

2）用**WRAPIT**生成共享对象。如果存在f90模块，应比调用它的子程序优先编译。用户必须指定所使用的编译器。**WRAPIT -h**查看命令选项。

```
WRAPIT quad90.stub quad.f90
```

3-4）与7.2相同

NCL脚本例子：

```
      external QUPR "./quad90.so"
```

```
begin
      a      = 2.5
      b      = -.5
      c      = 100.
      nx     = 10
      x      = fspan(1.,10.,10)
      q      = new(nx,float)
      QUPR::cquad(a,b,c,nx,x,q)
      QUPR::prntq(x,q,nx)
end
```

7.4 使用函数库

该过程和使用f90代码相似。假定我们利用IMSL的rcurv子程序，为了方便，运用f77语法。

1）任意创建一个名为rcurvWrap.f的wrapper程序

```
C  NCLFORTSTART
   subroutine rcurvwrap(n,x,y,nd,b,s,st,n1)
   integer n, nd, n1
   real x(n),y(n),st(10),b(n1),s(n1)
C NCLEND
   call rcurv (n,x,y,nd,b,s,st) ! IMSL name
   return
   end
```

2）利用**WRAPIT**编译wrapper子程序，并且指定IMSL函数库的位置。

```
WRAPIT -l mp -L /usr/local/lib64/r4i4 -l imsl_mp rcurvWrap.f
```

3）和7.2、7.3相同

脚本样本：

```
external IMSL "./rcurvWrap.so"
begin
      x = (/0,0,1,1,2,2,4,4,5,5,6,6,7,7/)
      y = (/508.1,498.4,568.2,577.3,651.7,657.0,\
      755.3,758.9,787.6,792.1.841.4,831.8,854.7,\
      871.4/)
      nobs = dimsizes(y)
      nd       = 2
      n1       = nd+1
      b        = new(n1,typeof(y))
      s        = new(n1,typeof(y))
      st       = new(10,typeof(y))
      IMSL::rcurvwrap(nobs,x,y,nd,b,s,st,n1)
      print(b)
      print(s)
      print(st)
```

```
end
```

7.5 WRAPIT 的作用

WRAPIT是UNIX脚本，它执行Fortran编译和链接生成共享对象的任务，它给用户提供许多参数：

```
WRAPIT -h <return>
```

WRAPIT执行下列任务:

1) 使用C语言程序**wrapit77**生成一个调用f77解析器的C wrapper程序，并生成使NCL和Fortran连接的C代码。

```
wrapit77 < foo.f >! foo_W.c
```

2）编译和生成 object模块

```
cc -c foo_W.c   ; foo_W.o
f77 -c foo.f    ; foo.o
```

3） 利用生成动态的共享对象(.so)文件

4） 清除临时文件，只保留共享对象(.so)文件。

7.6 NCL/Fortran 数组映射

在Fortran中，维最左边的列变化最快，而在NCL中最右边的列变化最快。有时这回引起混淆。当在NCL脚本中调用Fortran子程序时，很少需要对数组重新排序。因此，数组维数名称看起来同单个数组元素直接映射顺序相反。这里使用规则是：一种语言中变化最快的维映射到另一种语言变化最快的维。

NCL		Fortran
x(time,lev,lat,lon)	<=map=>	x(lon,lat,lev,time)

建立下面两个数组，其中N=2，M=3：

ncl: x(N,M) <==> x(M,N) :Fortran

x(0,0)	<==>	x(1,1)
x(0,1)	<==>	x(2,1)
x(0,2)	<==>	x(3,1)
x(1,0)	<==>	x(1,2)
x(1,1)	<==>	x(2,2)
x(1,2)	<==>	x(3,2)

7.7 unix shell 脚本中的 NCL 和 Fortran（C）

当运行正在调用一个或多个Fortran（C）共享对象的NCL脚本时，将所有单步执行语句联合成一个单独的unix shell脚本较为方便。

下面是通过csh脚本显示的：

```
#!/usr/bin/csh
# ========== Edit NCL Code ============
```

```
cat >! main.ncl << "END_NCL"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_code.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/gsn_csm.ncl"
load "$NCARG_ROOT/lib/ncarg/nclscripts/csm/contributed.ncl"
external SUB "./sub.so"
begin
      ...NCL code…
end
"END_NCL"

# ===========Edit Fortran Code ========
cat >! sub.f << "END_SUBF"
C NCLFORTSTART
      ...
C NCLEND
"END_SUBF"

# ========== Invoke WRAPIT ==========
WRAPIT sub.f

# ========== EXECUTE ========
ncl main.ncl >&! main.out
exit
```

7.8 将 NCL 作为脚本语言使用

下面使用NCL做一个循环（1）从NCAR的大容量存储系统(*msrcp*)获取文件；（2）调用netCDF操作命令（ncrcat）生成新的netCDF文件；（3）删除（rm）（1）中生成的文件。基本过程是通过**system**程序生成一个字符串，便于系统执行。**print**语句用来追踪脚本语句。

```
      mssi = "/Model/Sample/"          ; MSS path
      diri  = "/ptmp/user/"            ; dir containing input files
      fili   = "b20.007.pop.h.0"       ; prefix of input files
      diro  = "/ptmp/user/out/"        ; dir containing output files
      filo   = "b20.TEMP."             ; prefix of output files

      nyrStrt  = 300                   ; 1st year
      nyrLast  = 1000                  ; last year
      do nyear=nyrStrt,nyrLast
         print ("---- "+nyear+" ----")
                                       ; acquire 12 MSS files for year
         msscmd = "msrcp -n 'mss:" +mssi+ fili+nyear+ \
                  "-[0-1][0-9].nc' "+diri+"."
         print ("msscmd="+msscmd)
```

```
    system (msscmd)
                                ; strip off the TEMP variable
    ncocmd = "ncrcat -v TEMP " +diri+fili+"*.nc "+ \
             diro+filo+nyear+".nc"
    print ("ncocmd="+ncocmd)
    system (ncocmd)
                                ; remove the monthly files
    rmcmd = "'rm' "+diri+fili+nyear+ ".nc"
    print ("rmcmd="+rmcmd)
    system (rmcmd)
end do
```