UNIVERSITY of HOUSTON

CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

Introduction to Fortran 90 programming, part I

Martín Huarte-Espinosa
mhuartee@central.uh.edu

UNIVERSITY of HOUSTON

CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

VSCSE Summer Training Courses

CACDS to host two summer school classses on Harness the Power of GPU's: Introduction of GPGPU Programming on June 16th - 20th, 2014 and Data Intensive Summer Scool on June 30th - July 2nd 2014. Registration is open!

Continue Reading »

Summer Mini-Series in HPC

CACDS and the HPCTools group will host a block of presentations by the Supercomputing Program Commitee between June 13-20, 2014, at the University of Houston Philip Guthrie Hoffman Hall, Building 547, Room 232, to attract interest from researchers across campus who are engaged in HPC.

Continue Reading »

Argonne Training Program

This two-week program provides intensive hands-on training and offers renowned scientists, HPS experts, and leaders who serve as lecturers and guide hands-on laboratory sessions.

Continue Reading »

cacds.uh.edu/?q=training

UNIVERSITY of HOUSTON
CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

ABOUT | RESEARCH | EDUCATION | RESOURCES | NEWS

TRAINING COURSES
EVENTS
LECTURES
WORKSHOPS
TECHNICAL SEMINARS

Training Courses

EDUCATION

Training Courses>>
- Events
- Lectures
- Workshops
- Technical Seminars

CACDS has launched several HPC training courses for members of the UH research community.
Registration is required to take the courses. Please click the date you would like to attend under each course to register.

CURRENT TRAINING

Upcoming training events at CACDS

UNIVERSITY of HOUSTON

CENTER FOR ADVANCED COMPUTING & DATA SYSTEMS

Course Name	Dates
Introduction Fortran 90 Programming II	February 24, 2015 10:00 AM - 12:30 PM
Introduction to C++ Programming I	February 27, 2015 10:00 AM - 12:30 PM
Introduction to C++ Programming II	March 3, 2015 10:00 AM - 12:30 PM
Intel Xeon Phi Programming I	February 24, 2015 2:00 - 4:00 PM
Intel Xeon Phi Programming II	February 27, 2015 2:00 - 4:00 PM

Introduction to **MPI programming with Fortran 90**, this March.

Getting started

1. Login: hpc_user{1-47}, follow your seat's order

   Password=cacds2014

2. Use your web browser to see the slides

   File > Open … > /share/apps/tutorials/intro2f90_1.pdf

3. Go to *Applications > Systems tools > Terminal* and type the following 3 commands to get the tutorial examples:

include period

```
cp /share/apps/tutorials/intro2f90.zip .
unzip  intro2f90.zip
cd  intro2f90
```

Inside intro2f90

- Files with extension .f90 are programs for you to see.

- Files with extension .solution.f90 are solutions to the program with the same name prefix.

- Files with extension .dat are data files to be used along with some of the .f90 programs.

- intro2f90_1.pdf are these slides.

- Some programs will be studied in the workshop "Introduction to Fortran 90 part II".

Overview

- Short intro, Fortran vs. C++
- Variable names, types & declarations
- Program structure
- Math
- IO
- Logical statements,
- IF
- Do Loops
- Arrays

Fortran - Historical highlights

- **FOR**mula **TRAN**slating system,
- Development started in the 50s by IBM for scientific and engineering applications,
- Many large scientific codes & communities use it,

e.g. National Labs, NASA, CERN, Universities, etc.,

- There are several versions, FORTRAN 66, 77, Fortran 90, 2003, etc.,
- **We focus on Fortran 90;** it has many important improvements from previous distributions, beneficial for scientific programming. E.g. the array-syntax notation,
- Backward compatible with Fortran 77.

Fortran 90 vs. C++

-It depends,

-Fortran has strict aliasing semantics (memory data location accessed through different symbolic names) compared to C++, so **Fortran is very good with arrays**,

-C++ focuses heavily on **sophisticated data structures; Fortran is weaker** in that regard (but see next slide),

-**Fortran is case insensitive**; MyVariable=myvariaBLE,

-Row vs column order is different and may affect speed:

E.g. 2d array of *j* columns and *i* rows stored in memory

Fortran90: $x_{11},\ x_{21},\ x_{31},\ \ldots\ x_{ij}$

C++ : $x_{11},\ x_{12},\ x_{13},\ \ldots\ x_{ij}.$

Fortran 90 vs. C++

-It depends

-Fortran has strict aliasing semantics (memory data location accessed through different symbolic names) compared to C++, so **Fortran is very good with arrays**

-C++ focuses heavily on **sophisticated data structures; Fortran is weaker** in that regard (but see next slide)

-**Fortran is case insensitive**; MyVariable=myvariaBLE

-Row vs column order is different and may affect speed:

E.g. 2d array of *j* columns and *i* rows stored in memory

Fortran90: $x_{11}, x_{21}, x_{31}, \ldots x_{ij}$

C++ : $x_{11}, x_{12}, x_{13}, \ldots x_{ij}$

It's latin and means for example. I use it a

Fortran 90 vs. C++

- Not a full object-oriented language,
- BUT it supports abstract data types, encapsulation, function overloading, and classes,
- Inheritance and dynamic dispatching are not supported directly, but can be emulated (http://www.cs.rpi.edu/~szymansk/oof90.html),

We'll see some more Fotran 90 vs. C++ differences,
which I highlight in green.

The very basics

Use a terminal and a linux text editor to open the following small fortran program. If you are unfamiliar with linux editors use **nano** (see handout):

```
nano writeIntro.f90
```

This program prints a message on the screen.

```
!This is my first Fortran 90 program
PROGRAM writeIntro
IMPLICIT   NONE
write(*,*) 'Introduction to Fortran 90'
END PROGRAM writeIntro
```

Creating the executable (binary) from the source code

1. Code is saved into a .f90 file.

2. Compile your code, type:

```
f95  write.f90  -o  write.exe
```

3. Run the program, type:

```
./write.exe
```

NOTES:

No #include... needed, unlike C++

```
!This is my first Fortran 90 program
```
Comments start with !

```
PROGRAM writeIntro
```
always present, can be any name

```
IMPLICIT   NONE
```
see next slide

```
write(*,*) 'Introduction to Fortran 90'
```
writes what's in between ' ' as standard out (*)

```
END PROGRAM writeIntro
```
always present

Note also that no **;** is needed at the end any line, unlike C++

IMPLICIT NONE

- By default all variables starting with *i, j, k, l, m* and *n*, if not declared, are set of the *INTEGER* type. Handy but may lead to confusion.

- *IMPLICIT NONE* disables this; all names must be declared and there is no implicitly assumed *INTEGER* type.

Variable names

- Must be unique

- 1 to 31 letters followed by any combination of letters, numbers or underscores.

E.g. *temperature, t, t1, temp_hot, kelvin275*

- **Incorrect:**

1day:	first character must be a letter,
_system:	same as above,
R2-D2:	this is interpreted as, *R2 minus D2,*
M.I.T.:	dots cannot be used, unlike in C++ or python,

- Unlike Java & C++, Fortran 90 does not have reserved words.

Variable data types

All variables and their data types are declared **together at the beginning of programs only**.

DATA TYPES:

INTEGER

REAL

LOGICAL

CHARACTER

COMPLEX, we won’t cover this in this class

Data types & declarations -- example

to declare

```
INTEGER :: zip=77003
```

declare several variables using commas

```
REAL :: decimal=3.1415926535, exponential=1e3
```

Only 2 options

```
LOGICAL :: HPC_rocks=.true.,   parking_tickets_rule=.false.
```

These dots are needed

Good practice to initialize variables at declaration,

but you can do so latter

```
CHARACTER (LEN=3  ) :: string
CHARACTER (LEN=21) :: string2

   string  ="HPC"
   string2="University of Houston"
```

Quiz -- find the 5 syntax errors

```
PROGRAM MySecondProgram
IMPLICIT NONE

INTEGER zip=77003.0
REAL :: decimal
LOGICAL :: HPC_rocks
CHARACTER(LEN=2) :: name

decimal=3.1415926
HPC_rocks=true
name=”Maxwell”

END PROGRAM Program
```

Quiz -- answers

```
PROGRAM MySecondProgram
IMPLICIT NONE

INTEGER       zip=77003.0
REAL :: decimal
LOGICAL :: HPC_rocks
CHARACTER(LEN=2) :: name

decimal=3.1415926
HPC_rocks=true
name="Maxwell"

END PROGRAM Program
```

Missing colons to declare

Remove ".0", because zip was defined as an integer

Missing dots; It must be .true.

Maxell is 6 characters long, but name declared for 2 only

Program unit structure

```
PROGRAM name
IMPLICIT NONE
...

INTEGER :: zip=77003
REAL :: pi
LOGICAL :: HPC_rocks
CHARACTER(LEN=2) :: name
...

pi=3.14159265
HPC_rocks=.true.
name="Ok"
...

END PROGRAM name
```

Preamble

Declarations

Executable statements

Program unit structure*

```
PROGRAM name
IMPLICIT NONE
...

INTEGER :: zip=77003
REAL :: pi
LOGICAL :: HPC_rocks
CHARACTER(LEN=2) :: name
...

pi=3.14159265
HPC_rocks=.true.
name="Ok"
...

END PROGRAM name
```

Preamble

Declarations

Executable statements

*see "Intro to Fortran 90 part II" workshop.

Math!

Operations

+ add
- substract
* multiply
** power (x**2)
/ divide

Make double precision by adding a “D” as a prefix, e.g. DSQRT().

No CALL, unlike in C++

Some intrinsic functions

`ACOS( X )` - arccosine.

`ASIN( X )` - arcsine.

`ATAN( X )` - arctan.

`ATAN2( X, Y )` - arctan.

`COS( X )` - cosine.

`COSH( X )` - hyperbolic cosine.

`EXP( X )` - exponential.

`LOG( X )` - natural logarithm.

`LOG10( X )` - base 10 logarithm.

`SIN( X )` - sine.

`SINH( X )` - hyperbolic sine.

`SQRT( X )` - square root.

`TAN( X )` - tan.

`TANH( X )` - hyperbolic tan.

IO

Stands for Input/Output; communication.

2 commands.

```
READ

WRITE
```

IO

Stands for Input/Output; communication.

2 commands. General form:

```
READ  ( [UNIT=]unit, [FMT=]format ) variable list, ...
WRITE( [UNIT=]unit, [FMT=]format ) variable list, ...
```

unit, integer associated with a file,

format describes how the data should look,

Popular notation: variables inside [ ] are optional.

I'm also stressing this by using gray colors for optional commands.

IO

Stands for Input/Output; communication.

2 commands. General form:

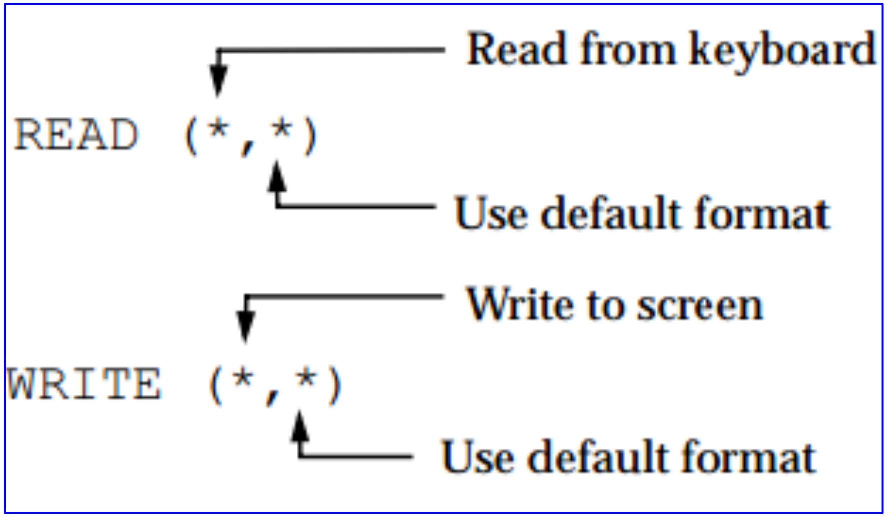
```
READ  ( [UNIT=]unit, [FMT=]format ) variable list, …
WRITE( [UNIT=]unit, [FMT=]format ) variable list, ...
```

unit, integer associated with a file,

format describes how the data should look,

E.g.:

```
READ(*,*) radius
area=3.1416*radius**2
WRITE(*,*) radius, area
```



IO -- format examples

```
INTEGER :: itest=1234567              !number to write
...
WRITE(*,*) itest                      ! 1234567
WRITE(*,'(I6)') itest                 !******
WRITE(*,'(I10)') itest                !   1234567
WRITE(*,'(I10.9)') itest              ! 001234567
WRITE(*,'(2I7)') itest, 7654321       !12345677654321
WRITE(*,'(2I8)') itest, 7654321       ! 1234567 7654321
```

```
REAL :: itest=123.4567               !number to write
...
WRITE(*,*) itest                      ! 1.2345670E+02 -not F format
WRITE(*,'(F8.0)') itest               !    123.
WRITE(*,'(F10.4)') itest              !  123.4567
WRITE(*,'(F10.5)') itest              ! 123.45670
WRITE(*,'(F10.9)') itest              !**********
WRITE(*,'(2F8.4)') itest, 7654321     !123.4567765.4321
WRITE(*,'(2F10.4)') itest, 7654321 !  123.4567  765.4321
```

From http://www.uv.es/dogarcar/man/IntrFortran90.pdf

IO -- more format examples

```
REAL :: itest=123.45*1000000    !number to write times 1 million
...
WRITE(*,*) itest                  !   1.2345670E+02
WRITE(*,'(E10.4)') itest          !0.1234E+09
WRITE(*,'(E10.5)') itest          !.12345E+09
WRITE(*,'(E10.4E3)') itest        !.1234E+009
WRITE(*,'(E10.9)') itest          !**********
WRITE(*,'(2E12.4)') itest, 7654321 ! 0.12345E+09 0.76543E+04
WRITE(*,'(2E10.4)') itest, 7654321 !0.1234E+090.7654E+04
```

```
CHARACTER(LEN=8) :: long='Bookshop'
CHARACTER(LEN=1) :: short='B'
...
WRITE(*,*) long                 ! Bookshop
WRITE(*,'(A)') long             !Bookshop
WRITE(*,'(A8)') long            !Bookshop
WRITE(*,'(A5)') long            !Books
WRITE(*,'(A10)') long           !  Bookshop
WRITE(*,'(A)') short            !B
WRITE(*,'(2A) short, long       !BBookshop
WRITE(*,'(2A3) short, long      !  BBoo
```

From http://www.uv.es/dogarcar/man/IntrFortran90.pdf

IO -- format labels

They save time. Use a specific formats several times.

- Define labels anywhere in programs

- Refer to labels in `READ(*,label)` or `WRITE(*,label)`

General form:

```
label FORMAT (format list,)
```

label is a 3 digit integer identifier,

format list is the list of edit descriptors (previous slides).

IO -- format labels example

Open heron1.f90.

It computes and prints the area of a triangle using Heron’s formula.

Note the formal labels relation:

```
    ...
    READ(*,501) A,B,C
        ...
    WRITE(*,601) A,B,C,AREA
    ...
    501 FORMAT(3f4.1)
    601 FORMAT(" A= ",f4.1,"  B= ",f4.1,"  C= ",f9.2,"  AREA= ",
f9.2,"UNITS")
    ...
```

IO -- data files

General form:

```
open(unit,file)
   … other statements, including IO ...
close(unit)
```

unit is a 2 digit integer identifier

file is a valid filename, e.g. file='output.test'.

IO -- exercise

Copy heron1.f90 into heron2.f90, type:

```
cp heron1.f90 heron2.f90
```

Then change the code to write the result into a file:
/home/hpc_userXX/intro2f90/area.dat

where XX corresponds to your session's user name.

Hints:

Let's add the open command here, and substitute the first * for a label

```
    WRITE(*,*) 'A     B      C      AREA'
    WRITE(*,601) A,B,C,AREA
```

Let'a add the close(label) command here

```
  501 FORMAT(3f4.1)
  601 FORMAT(" A= ",f4.1,"  B= ",f4.1,"  C= ",f4.1,"  AREA=
",f6.1,"UNITS")
  ...
```

The solution is in heron2.solution.f90

IO -- OPEN

OPEN takes other optional arguments that control file IO with precision.

E.g.:

```
O
PEN(UNIT = 15, FILE = 'input.dat', STATUS = "OLD", &
    ACTION = "READWRITE", IOSTAT = Open_Status)
    IF (Open_Status > 0) STOP ["-----Error, File not
opened properly------"]
```

Notes:

- Break long lines using &,
- ***STOP*** terminates the program completely.

Logical & comparison

```
LOGICAL :: result
INTEGER :: age, myAge
CHARACTER(LEN=7) :: who
...
result = 6 < 7          !True
result = 6 > 7          !False
result = 6 == 7         !False
result = 6 /= 7         !True
result = 6 <= 7         !True
result = 6 >= 7         !False
...
result = age > 34                 !variable vs. constant
result = age /= myAge             !two variables compared
result = 45 == myAge              !variables appear in any side
result = who == 'Maxwell'         !characters allowed
result = (age*3) /= myAge         !expressions allowed
```

Logical & comparison

```
LOGICAL :: result
INTEGER :: age, myAge
CHARACTER(LEN=7) :: who
...
result = 6 < 7          !True
result = 6 > 7          !False
result = 6 == 7         !False
result = 6 /= 7         !True
result = 6 <= 7         !True
result = 6 >= 7         !False
...
result = age > 34               !variable vs. constant
result = age /= myAge           !two variables compared
result = 45 == myAge            !variables appear in any side
result = who == 'Maxwell'       !characters allowed
result = (age*3) /= myAge       !expressions allowed
```

Also:

```
result = 6 .LT. 7
result = 6 .GT. 7
result = 6 .EQ. 7
result = 6 .NE. 7
result = 6 .LE. 7
result = 6 .GE. 7
```

Control -- IF

General form:

```
IF          (logical          arguments)          statement
```

statement executes only when *logical arguments* are *.true.*.

E.g.

```
IF (1.0>0.0) write (*,*) 'Yes, 1>0'

UHstudent=.true.
IF (UHstudent) write(*,*) 'Go cougars'
IF (.not. UHstudent) statement    !statement not executed
IF (.false.) statement            !statement not executed
                  !the last one useful when debugging
```

Control -- nested IF

General form:

```
IF (logical-expression-1) THEN
   statements-1
ELSE IF (logical-expression-2) THEN
   statements-2
[ELSE IF (...) THEN]
   [...]
[ELSE]
   [statements-ELSE]
END IF
```

Control -- nested IF

E.g.

```
REAL :: cost, discount
INTEGER :: n                                 !number of items

WRITE(*,*) 'How many items to buy?'
READ(*,*) n
IF ( n>10 ) THEN                             !25% on 11 or more
   discount = 0.25
ELSE IF ( n>5 .AND. n<=10 ) THEN             !15% on 6-10 items
   discount = 0.15
ELSE IF ( n>1 .AND. n<=5 ) THEN              !15% on 2-5 items
   discount = 0.1
ELSE                                         !no for 1
   discount = 0.0
END IF

cost = cost-(cost*discount)
WRITE(*,*) 'Invoice for ', cost
…
```

From http://www.uv.es/dogarcar/man/IntrFortran90.pdf

Control -- nested IF

E.g.

```
REAL :: cost, discount
INTEGER :: n                                    !number of items

WRITE(*,*) 'How many items to buy?'
READ(*,*) n
IF ( n>10 ) THEN
   discount = 0.25
ELSE IF ( n>5 .AND. n<=10 ) THEN
   discount = 0.15
ELSE IF ( n>1 .AND. n<=5 ) THEN
   discount = 0.1
ELSE
   discount = 0.0
END IF

cost = cost-(cost*discount)
WRITE(*,*) 'Invoice for ', cost
…
```

Logical connectors:

.AND.	logical intersection,
.OR.	logical union,
.NOT.	logical negation,
.EQV.	logical equivalence,
.NEQV.	logical non-equivalence.

You can use as many as you want.

From http://www.uv.es/dogarcar/man/IntrFortran90.pdf

IF - exercise

- Open a new file called grades.f90, type:
  `nano grades.f90.`

- Use a nested IF, ELSE IF,... END IF construct

- Write a little program that transforms grades from numbers to letters in the following way:
  0-5=F; 5-6=D; 6-7=C; 7-8=B; 8-10=A.

IF - exercise solution

```
INTEGER :: x
CHARACTER(LEN=1) :: Grade

IF (x < 50) THEN
   Grade = 'F'
ELSE IF (x < 60) THEN
   Grade = 'D'
ELSE IF (x < 70) THEN
   Grade = 'C'
ELSE IF (x < 80) THEN
   Grade = 'B'
ELSE
   Grade = 'A'
END IF
```

From http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/chap03/else-if.html.

Control -- CASE

General form:

```
[name:] SELECT CASE( expression )
   CASE( value ) [name]
   block
   ...
   [CASE DEFAULT
   block]
END SELECT [name]
```

Very useful in programs that can do a lot of exclusive computations, e.g. 2D or 3D computations; statistics; IO; plot.

CASE -- example & exercise

Open the file season.f90, type:

```
nano season.f90
```

Read it and make sure you understand the algorithm.
Then complete it to works for all seasons.

```
PROGRAM p2
IMPLICIT NONE

INTEGER :: month

write(*,*) 'Please write the month 1-12'
read (*,*) month
season:SELECT CASE( month ) !note that "season" is
optional; helps clarity.
   CASE(4,5)
      WRITE(*,*) 'Spring'
   CASE(6,7)
      WRITE(*,*) 'Summer'
...
```

Loops

General form:

```
[name:] DO count = start, stop [,step]
   statements
END DO [name]
```

where count, start, stop and step are integers.

Number of iterations = (stop + step – start)/step

E.g.:

```
INTEGER :: i, k
   DO i=1,10                   !write 1, 2… 10
      WRITE(*,*) i
   END DO

   even: DO k=10,2,-2          !write 10,8,6,4,2
      WRITE(*,*) k
   END DO even
```

Loops

General form:

```
[name:] DO count = start, stop [,step]
    statements
END DO [name]
```

where count, start, stop and step are integers.

Number of iterations = (stop + step – start) ⁄ step

E.g.:

```
INTEGER :: i, k
   DO i=1,10
      WRITE(*,*) i
   END DO

   even: DO k=10,2,-2
      WRITE(*,*) k
   END DO even
```

There are no ++ or -- in Fortran

Fortran:	*i=i+1*	*i=1+i*	*i=i-1*	*i=1-i*
C++:	*i++*	*++i*	*i--*	*--i*

Loops -- control transfer

EXIT

- transfers control to outside the loop before
  - the *END DO* is reached or
  - the final iteration is completed

```
 INTEGER :: value=0,
total=0
   ...
   DO
```

note this is blank

```
   READ(*,*) value
     IF (value==0) EXIT
     total = total + value
   END DO
```

When is it useful?

Loops -- control transfer

EXIT

- transfers control to outside the loop before
  - the *END DO* is reached or
  - the final iteration is completed

```
 INTEGER :: value=0,
total=0
  ...
  DO
  READ(*,*) value
    IF (value==0) EXIT
    total = total + value
  END DO
```

CYCLE

- skips the rest of the current iteration
- transfers control back to the beginning of the loop to allow the next iteration to begin

```
  INTEGER :: int
    ...
    DO
      READ(*,*) int     !read in
          IF (int<0) CYCLE !if
           !negative, read
           !another
      ...
    ENDDO
```

Loops -- control transfer

EXIT

- transfers control to outside the loop before
  - the *END DO* is reached or
  - the final iteration is completed

```
 INTEGER :: value=0,
total=0
  ...
  DO
  READ(*,*) value
    IF (value==0) EXIT
    total = total + value
  END DO
```

CYCLE

- skips the rest of the current iteration
- transfers control back to the beginning of the loop to allow the next iteration to begin

When is it useful?

```
I
  ...
  DO
    READ(*,*) int     !read in
       IF (int<0) CYCLE !if
         !negative, read
         !another
    ...
  ENDDO
```

Loops -- control transfer

EXIT

- transfers control to outside the loop before
  - the *END DO* is reached or
  - the final iteration is completed

```
 INTEGER :: value=0,
total=0
  ...
  DO
  READ(*,*) value
    IF (value==0) EXIT
    total = total + value
  END DO
```

STOP [message]

Terminates the program immediately.

CYCLE

- skips the rest of the current iteration
- transfers control back to the beginning of the loop to allow the next iteration to begin

```
INTEGER :: int
  ...
  DO
    READ(*,*) int     !read in
        IF (int<0) CYCLE !if
         !negative, read
         !another
    ...
  ENDDO
```

Exercise

- Open multiply.f90, type

  `nano multiply.f90`
- It reads real numbers and multiples them sequentially,
- It writes the total for every entered number.

Edit whenever you see a ? sign, so that:

- It does nothing when 0 is entered, but keeps going,
- It ends only when a negative number is entered.

Exercise -- solution

- Use: `if, do, exit, cycle, read & write,`
- Reads real numbers and multiples them sequentially,
- It writes the total for every entered number,
- It does nothing when 0 is entered, but keeps going,
- It ends only when a negative number is entered.

multiply.solutionf.90

Optional exercise

Our Heron formula program has a computational problem. E.g. try a triangle with sides 1.2 5.6 9.8.

Edit heron3.f90, type

```
nano heron3.f90
```

- It offers a solution to this problem.
- Also, use a DO loop and the IF statement to write either the result for cases when the triangle sides do not yield a numerical problem, or otherwise ask the user to try again.

Arrays

```
REAL, DIMENSION (5) :: A, B    !arrays of 5 elements, with
                                   !indices 1,2…, 5.
REAL, DIMENSION (16) :: B  !arrays of 16 elements, with
                                !indices 1,2..., 16

A = (/ 2, 4, 6, 8, 10 /)    !note the (/ … /)

B = (/     1, 2, 3, 4, 5, 6, 7, 8, &
       2, 4, 6, 8, 10, 12, 14, 16 /)
```

Very important for STEM applications.

Arrays

```
REAL, DIMENSION (5) :: A, B      !2 arrays of 5 elements, with
                                 !indices 1,2...5.
REAL, DIMENSION (0:4) :: c       !1 array of 5 elements, index
                                 !0,1...4

A = (/ 2, 4, 6, 8, 10/) or           !note the (/ … /)
A = (/  (2*I, I = 1,5) /) or         !I must be declared as INTEGER
A = (/ 2, (2*I, I = 2, 4), 10 /)

!Operations may be applied to arrays of equal dimensions:
B=A            !Array syntax, powerful feature of Fortran 90
C=5*C+A
WHERE (A > 6)     !This gives B = 0,0,0,1,1
      B = 1.
ELSEWHERE
      B = 0.
END WHERE
```

Arrays -- multi dimensional

```
REAL, DIMENSION(3,4) :: y              !Rank 2, 3x4 elements
REAL, DIMENSION(0:2,0:3) :: x          !Same
INTEGER, DIMENSION(12,22,4) :: z       !Rank 3
                                   !Max rank per array = 7
z=0 !constant.

!First index is the row. Second one is the column
x(0,0)=1. ; x(0,1)=1. ; x(0,2)=1. ;  x(0,3)=1.
x(1,0)=2. ; x(1,1)=2. ; x(1,2)=2. ;  x(1,3)=2.
x(2,0)=3. ; x(2,1)=3. ; x(2,2)=3. ;  x(3,3)=3.
                     !; used for many commands in one line
DO i = 0, 3          !Transpose a matrix
   DO j = i+1, 3
      d = x(i,j)     !d defined REAL earlier
      x(i,j) = x(j,i)
      x(j,i) = d
END DO ; END DO
```

Arrays -- multi dimensional

The construct

```
x(0,0)=1. ; x(0,1)=1. ; x(0,2)=1. ;  x(0,3)=1.
x(1,0)=2. ; x(1,1)=2. ; x(1,2)=2. ;  x(1,3)=2.
x(2,0)=3. ; x(2,1)=3. ; x(2,2)=3. ;  x(3,3)=3.
```

can also be written as: `x(0,:)=1. ; x(1,:)=2. ; x(2,:)=3.`,
where the colon, ***:***, means "all elements of that dimension".

Exercise -- Arrays 1

Just do the algorithm (no need to compile, etc.).

Declare an integer array `iarray`, which contains 3 rows and 4 columns. Initialize the first row with integer values 1 to 4 (from left to right), the second row with integers from 5 to 8, and fill the last row with -2. Then print the `iarray`, row by row so that each output line contains the elements of one row at a time.

From http://napsu.karmitsa.fi/courses/supercomputing/lssc/fortran_ex.txt

Exercise -- Arrays 1, solution

Just do the algorithm.

Declare an integer array `iarray`, which contains 3 rows and 4 columns. Initialize the first row with integer values 1 to 4 (from left to right), the second row with integers from 5 to 8, and fill the last row with -2. Then print the `iarray`, row by row so that each output line contains the elements of one row at a time.

```
...
INTEGER, DIMENSION(3,4) :: iarray
INTEGER i,j

iarray(1,1:4) = (/ (j, j=1,4) /)
iarray(2,1:4) = (/ (j, j=5,8) /)
iarray(3,:) = -2
DO i=1,3
    WRITE(*,*) iarray(i,:)
END DO
```

From http://napsu.karmitsa.fi/courses/supercomputing/lssc/fortran_ex.txt

Exercise -- Arrays 2

Continue. Just the algorithm.

Build a new array; 3-by-8 integer array `bigarray`, where the 4 first columns are identical to the `iarray`, and the 4 last columns are obtained from the columns of the array `iarray` by multiplying them with the number 3 and adding 5. **Use array syntax**.

From http://napsu.karmitsa.fi/courses/supercomputing/lssc/fortran_ex.txt

Exercise -- Arrays 2, solution

Continue. Just the algorithm.

Build a new array; 3-by-8 integer array `bigarray`, where the 4 first columns are identical to the `iarray`, and the 4 last columns are obtained from the columns of the array `iarray` by multiplying them with the number 3 and adding 5. **Use array syntax**.

```
...
INTEGER, DIMENSION(3,8) :: bigarray
INTEGER, DIMENSION(3,4) :: iarray
INTEGER i,j
iarray(1,1:4) = (/ (j, j=1,4) /)
iarray(2,1:4) = (/ (j, j=5,8) /)
iarray(3, : ) = -2
bigarray(:,1:4) = iarray(:,:)
bigarray(:,5:8) = iarray(:,1:4) * 3 + 5
```

From http://napsu.karmitsa.fi/courses/supercomputing/lssc/fortran_ex.txt

Arrays -- intrinsic functions

DOT_PRODUCT(A, B)	
MAXVAL(A)	maximum value in A
MAXLOC(A)	one-element 1D array whose value is the location of the first occurrence of the maximum value in A
PRODUCT(A)	product of the elements of A
SUM(A)	sum of the elements of A
MAXVAL(A, D)	array of one less dimension than A; the A maximum values along dimension D
MAXLOC(A)	one-element 1D array whose value is the location of the first occurrence of the maximum value in A
SUM(A, D)	array of one less dimension than A; sums of A elements along dimension D (if D omitted, returns entire array elements sum)
MATMUL(A, B)	matrix product of A and B
TRANSPOSE(A)	

Arrays -- dynamic allocation

Assign memory for arrays during execution.

Scientific application: adaptive-mesh simulations.

E.g.

```
INTEGER :: i,j,AllocateStatus
REAL, DIMENSION(:,:), ALLOCATABLE :: A
...
READ (*,*)  i,j  !read array dimension from keyboard
ALLOCATE ( A(i,j), STAT = AllocateStatus)
   [IF (AllocateStatus /= 0) STOP "Insufficient A memory"]
```

Arrays -- dynamic allocation

Assign memory for arrays during execution.
Scientific application: adaptive-mesh simulations.

E.g.

```
INTEGER :: i,j,AllocateStatus
REAL, DIMENSION(:,:), ALLOCATABLE :: A
…
READ (*,*)  i,j  !read array dimension from keyboard
ALLOCATE ( A(i,j), STAT = AllocateStatus)
   [IF (AllocateStatus /= 0) STOP "Insufficient A memory"]
```

The allocated memory has to be released once it it's not used:

```
DEALLOCATE (A, STAT = DeAllocateStatus)
   [IF (AllocateStatus /= 0) STOP "Memory deallocation
error"]
```

Arrays dynamic allocation -- examples

readArrayDynamically.f90 is a little program illustrating how to read an array of numbers, of unknown size, from a file. It makes use of array dynamic allocation.

1. Open the file with your editor.
2. Compile, and run.

Do you understand what's happening?

Exercise

Modify it to read a 10x10 array.

Solution: read2dArrayDynamically.f90, **DON'T CHEAT!**

How can you modify it so that the write command is more reader friendly? How can you improve these programs?

Application problem

The program **taylor.f90*** uses a Taylor expansion to estimate the value of the exponential function evaluated at 1. It also compares the intrinsic Fortran exponential function vs. the Taylor estimation. This is a good exercise on mathematical error tracking and how computers and calculators actually calculate well-known math functions such as the exponential, log(x) and the trigonometric ones.

1. Read the source code. Do not modify it. Compile & run. Make sure you understand what’s happening. Note the KIND specification.

2. What would you do differently?

3. If there is time, or for homework: do a similar program but for the SIN(x) function.

*From http://faculty.washington.edu/rjl/uwamath583s11/sphinx/notes/html/fortran_taylor.html#fortran-taylor

References

- Many books and online sites available
  - E.g. "Modern Fortran in Practice", published by Cambridge University Press.
- Specify “...fortran **90**…” in your searches; you may get info on older distributions (e.g. 77 is popular for legacy code).
- Some useful links:
  - *http://fortranwiki.org/fortran/show/Fortran+Wiki*
  - Reference card, **PRINT IT**!,

http://www.pa.msu.edu/~duxbury/courses/phy480/fortran90_refcard.pdf

  - The NAG libraries have hundreds of routines (may be commercial?) for science and engineering applications

    http://www.nag.com/numeric/fl/FLdescription.asp
  - *http://www.lahey.com/other.htm,* many fortran links
  - *http://flibs.sourceforge.net/**examples_modern_fortran**.html*

Thank you

- Please fill the course assessment forms, *just click the home icon in your browser*

- Email yourselves the **slides** from
  /share/apps/tutorials/intro2f90.pdf

- Upper menu > System > **Log Out** hpc_user...

Upcoming training events at CACDS

Introduction Fortran 90 Programming II	February 24, 2015	10:00 AM - 12:30 PM
Introduction to C++ Programming I	February 27, 2015	10:00 AM - 12:30 PM
Introduction to C++ Programming II	March 3, 2015	10:00 AM - 12:30 PM
Intel Xeon Phi Programming I	February 24, 2015	2:00 - 4:00 PM
Intel Xeon Phi Programming II	February 27, 2015	2:00 - 4:00 PM

Introduction to **MPI programming with Fortran 90**, this March.

The Nano Text Editor

^X means ``hold down the CTRL key and press the x key''. Commands listed at the bottom of your screen.

To edit a file called *filename*, type `nano` *filename*.

^O	save contents without exiting (you will be prompted for a file to save to)
^X	exit nano (you will be prompted to save your file if you haven't)
^T	when saving a file, opens a browser that allows you to select a file name from a list of files and directories

^A	move to beginning of line
^E	move to end of line
^Y	move down a page
^V	move up a page
^_	move to a specific line (**^_^V** moves to the top of the file, **^_^Y** to the bottom)
^C	find out what line the cursor is currently on
^W	search for some text.

When searching, you will be prompted for the text to search for. It searches from the current cursor position, wrapping back up to the top if necessary.

^D	delete character currently under the cursor
BackSpace	delete character currently in front of the cursor
^K	delete entire line
^	search for (and replace) a string of characters

Getting started

1. Login: hpc_user{1-47}, follow your seat's order
   Password=cacds2014

2. Use your web browser to see the slides
   File > Open … > /share/apps/tutorials/intro2f90_1.pdf

3. Go to *Applications > Systems tools > Terminal*
   and type the following 3 commands to get the tutorial examples:

include period

```
cp /share/apps/tutorials/intro2f90.zip .
unzip  intro2f90.zip
cd  intro2f90
```