



TEXAS ADVANCED COMPUTING CENTER

WWW.TACC.UTEXAS.EDU



TEXAS

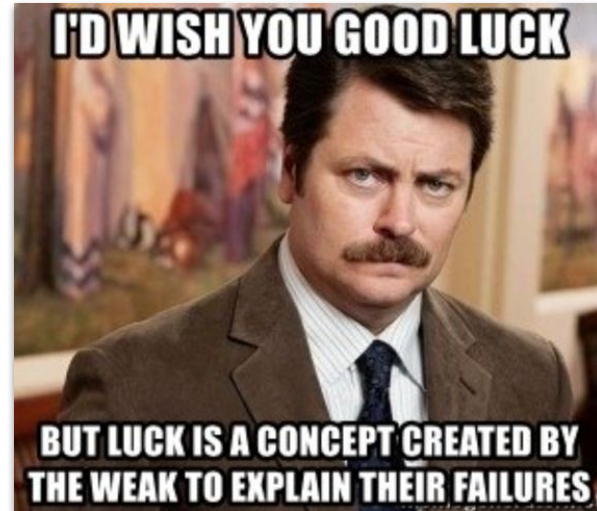
The University of Texas at Austin

Fortran, Formatted IO Implicit Loops

Standard In/Out, Implicit Loops, File Input/Output
Kevin Schmidt, Susan Lindsey, Charlie Dey

Objectives

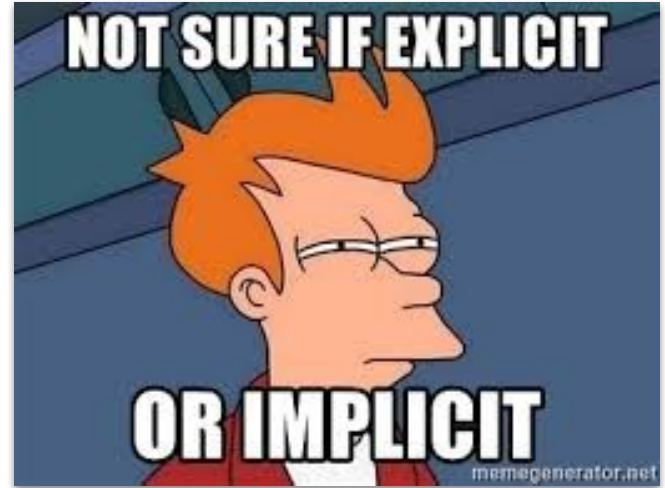
- Format Fortran Output Explicitly
- Use Implicit Loops



Terminology

Implicit - Compiler assumes what you really meant to code.

Explicit - The meaning of a programmatic statement is fully defined in the source code.



Input/ Output (I/O)

Background

- **Can be tricky**
- All languages provide more or less the same functionalities
- Two (three) choices are to be made:
 - Reading from/writing to file, keyboard/screen
 - Formatted or Unformatted
 - Sequential or Direct Access/Streams
- Formatted: Human readable text (machines read that too, but slower)
- Unformatted: Native computer format (hard to read by humans)
- Sequential: Read/write a file one record or line after another
- Carriage return ends a record or line
- No skipping
- Direct Access: Fast forward/rewind to "any" position in file (Restrictions may apply)

I/O

Formatted vs. Unformatted

- Formatted I/O is most often combined with sequential access
 - Read/write from/to files
 - Write to screen
- Unformatted I/O is often combined with direct access/streams
 - easier to setup; no need to figure the format
 - direct access allows to freely choose the read position

Formatted Output

An example

```
program array1
implicit none
integer :: i
integer, dimension(5) :: A = (/ 1, 2, 3, 4, 5
/)

do i=1,5
    print *, A(i)
end do

end program array1
```

This is an example of an *implicit* format, we going to now use an explicit format

*more on dimension later when we talk about Arrays

Formatted Output

An example

```
program array1
implicit none
integer :: i
integer, dimension(5) :: A = (/ 1, 2, 3, 4, 5
/)

do i=1,5
  print '(i4)', A(i)
end do

end program array1
```

explicit format, integer
telling the computer that we're going to be printing an
integer that is 4 characters wide.

Formatted Output

Adding a format to the print statement for *explicit* formatting

- The format string replaces the star (*)
- Most common formats are:

a<w> : Character (Number <w> is optional)

i<w> : Integer

f<w.d> : Floating point number

es<w.d> : Floating point number (scientific notation)

l<w> : Logical (Number <w> is optional)

x : blank space

<w> is the width (may be optional)

<d> is the number of decimal places (if needed)

Formats may be repeated with a preceding number: <r>format

Formatted Output

Character formatting

```
character(len=8) :: n = 'John Doe'
```

```
print '(a1,a8,a1)', '>', n, '<' ! >John Doe< ! Explicit width
print '(a, a, a)', '>', n, '<' ! >John Doe< ! Implicit width
print '(a, a4,a)', '>', n, '<' ! >John< ! String truncated
print '(a, a5,a)', '>', n, '<' ! >John < ! String truncated
print '(a, a6,a)', '>', n, '<' ! >John D< ! String truncated
print '(a, a8,a)', '>', n, '<' ! >John Doe< ! String padded
print '(a, a9,a)', '>', n, '<' ! > John Doe< ! String padded
```

a<w> :: optional w is the width (number of characters)

Formatted Output

Integer formatting

```
integer :: i = 1234,    &  
         j = 12345678, &  
         n = -12345678
```

```
print '(a,i4,a)', 'i=', i, '<' ! i=1234<  
print '(a,i6,a)', 'i=', i, '<' ! i= 1234<  Padding with blanks  
print '(a,i8,a)', 'i=', i, '<' ! i=   1234<  
print '(a,i8,a)', 'j=', j, '<' ! j=12345678<  
  
print '(a,i4,a)', 'j=', j, '<' ! j=****<    Number has 8 digits  
  
print '(a,i2,a)', 'i=', i, '<' ! i=**<      Format holds only 4  
                                         4 digits, 2 spaces  
  
print '(a,i8,a)', 'n=', n, '<' ! n=*****< Account for sign  
print '(a,i9,a)', 'n=', n, '<' ! n=-12345678<
```

i<w> :: w is the width (number of characters)

Formatted Output

Real formatting

```
real          :: pi = 3.14159, &
                p2 = 314.159, &
                pm = -314.159

print '(a,f7.5,a)', 'pi=',pi,'<' ! pi=3.14159<  1 digit before dot
print '(a,f6.4,a)', 'pi=',pi,'<' ! pi=3.1416<   Rounded
print '(a,f7.3,a)', 'p2=',p2,'<' ! p2=314.159<  3 digits before dot
print '(a,f7.5,a)', 'p2=',p2,'<' ! p2=*****<
print '(a,f9.5,a)', 'p2=',p2,'<' ! p2=314.15900< 3 digits before dot
print '(a,f7.3,a)', 'pm=',pm,'<' ! pm=*****<  3 digits before dot
print '(a,f8.3,a)', 'pm=',pm,'<' ! pm=-314.159<  4 digits before dot
```

f<w.d> :: w is the width (number of characters),
d is the number of decimal places

Formatted Output

Real formatting, scientific notation

```
real :: xp = 123456., &  
      xn = -123456., c = 2.99e8  
  
print '(a,es11.5,a)', 'xp=',xp,'<' ! xp=1.23456E+05<  
print '(a,es12.5,a)', 'xn=',xn,'<' ! xn=-1.23456E+05<  
print '(a,es11.5,a)', 'xn=',xn,'<' ! xn=*****<  
print '(a,es12.5,a)', 'xp=',xp,'<' ! xp= 1.23456E+05<  
print '(a,es15.5,a)', 'xp=',xp,'<' ! xp=    1.23456E+05<  
  
print '(a,es11.4,a)', 'xp=',xp,'<' ! xp= 1.2346E+05<  
print '(a,es11.4,a)', 'xn=',xn,'<' ! xn=-1.2346E+05<  
print '(a,es10.3,a)', 'c=',c,'<' ! c= 2.990E+08<
```

`es<w.d>` :: w is the width (number of characters),
d is the number of decimal places

Positive numbers: $w \geq d + 6$
Negative numbers: $w \geq d + 7$

Formatted Output

Spaces

```
real          :: age  = 17.2
character(len=8) :: name = 'John Doe'
integer       :: eid  = 1705

print '(a,a, a,f4.1, a,i4)',
      'Name is', name, 'Age is', age, 'eid is', eid

print '(a,1x,a, 2x,a,1x,f4.1, 2x,a,1x,i4)',
      'Name is', name, 'Age is', age, 'eid is', eid
```

x for spaces:
'1x': 1 space
'2x': 2 spaces

...

(I hope you guys see the pattern)

Formatted Output

Spaces

```
real          :: age  = 17.2
character(len=8) :: name = 'John Doe'
integer       :: eid  = 1705

print '(a,a, a,f4.1, a,i4)',
      'Name is', name, 'Age is', age, 'eid is', eid

print '(a,1x,a, 2x,a,1x,f4.1, 2x,a,1x,i4)',
      'Name is', name, 'Age is', age, 'eid is', eid
```

x for spaces:
'1x': 1 space
'2x': 2 spaces

...

(I hope you guys see the pattern)

output:

```
Name isJohn DoeAge is17.2eid is 1705
Name is John Doe Age is 17.2 eid is 1705
```

Formatted Output

repetition

```
real, dimension(3) :: x = [ 3.3, 5.5, 7.7 ]  
integer, dimension(3) :: l = [ 3, 5, 7 ]
```

```
print '(3f5.2)', x ! 3.30 5.50 7.70  
print '(3i5)', l ! 3 5 7
```

```
print '(3(i4,1x,f4.2,2x))', l(1),x(1), l(2),x(2), l(3),x(3)
```

```
3f7.2 == f7.2, f7.2, f7.2
```

```
3(i5,f7.2) == i5,f7.2, i5,f7.2, i5,f7.2
```

3(format) repeats the format statement 3 times

*more on dimension later when we talk about Arrays

Formatted Output

implicit loops - this is cool [Note: “This is cool” was noted by Charlie That is all]

```
real, dimension(3)    :: x = [ 3.3, 5.5, 7.7 ]
integer, dimension(3) :: l = [ 3,   5,   7   ]

print '(3f5.2)', x    ! 3.30 5.50 7.70
print '(3i5)',   l    !    3    5    7

print '(3(i4,1x,f4.2,2x))', l(1),x(1), l(2),x(2), l(3),x(3)

! You can also do something like this:
print '(3(i4,1x,f4.2,2x))', (l(i), x(i), i=1, 3)
```

Or... we can do this, an *Implicit Loop*

Formatted Output

implicit loops

```
program implicitLoop
implicit none

integer :: i, j, k
real, dimension(4) :: x, y, z
real, dimension(10,5) :: w

call random_number(x)

print *, "X:"
print '(4f13.10,x)', (x(i), i=1, 4)

y = [1., 2., 3., 4.]
z(1:4) = [ (sqrt(y(i)), i=1, 4) ]
print *, "Z, Y:"
print '(4(f16.10,x), 4(f3.1,x))', (z(i), i=1, 4), (y(i),
i=1, 4)

call random_number(w)
print *, "W:"
print '(50(f16.10,x))', ((w(i,j), i=1, 10), j=1, 5)

end program
```

More on *Implicit* Loops

Formatted Output

implicit loops

```
program implicitLoop
implicit none

integer :: i, j, k
real, dimension(4) :: x, y, z
real, dimension(10,5) :: w

call random_number(x)

print *, "X:"
print '(4f13.10,x)', (x(i), i=1, 4)

y = [1., 2., 3., 4.]
z(1:4) = [ (sqrt(y(i)), i=1, 4) ]
print *, "Z, Y:"
print '(4(f16.10,x), 4(f3.1,x))', (z(i), i=1, 4), (y(i), i=1, 4)

call random_number(w)
print *, "W:"
print '(50(f16.10,x))', ((w(i,j), i=1, 10), j=1, 5)

end program
```

More on *Implicit* Loops

Here is an
implicit loop!

Exercise

Write a program that creates an array of 100 random numbers. Use an implicit loop and a formatted output to make it look *nice with only 2 numbers after the decimal point and use implicit loop*

Bonus: Output as 10 X 10 Matrix

```
// this code generates an array of 100 random numbers
program random_number_array
implicit none
real, dimension(100) :: r
    call random_number(r)
    print *, r
end program
```