

Sergey Yurchenko, Physics and Astronomy

Introduction to Fortran 95

Materials used:

- AC Marshal: Introductory Programming with Fortran 90, 3 Day Course, University of Liverpool, 1997
- Martin Counihan: Fortran 95, UCL press, London, 1996
- Nick Maclaren and Steve Morgan: Slides for Introduction to Modern Fortran. Computing Services Department, University of Liverpool
- An introduction to programming in Fortran 90, University of Durham Information Technology Service , version 2.0
- John Burkardt: An Introduction to Fortran Programming, <http://people.sc.fsu.edu/~jburkardt/html/fortran.html>

Exercise: What are the values of the following integer expressions, if $n = -2$?

- Ⓐ $n^{**}2-2$
- Ⓑ $n/2/2$
- Ⓒ $\text{MOD}(42, n+11)$
- Ⓓ $\text{MOD}(n, -3)$
- Ⓔ $\text{ABS}(n^{**}3-n)$
- Ⓕ $2^{**}n$
- Ⓖ $\text{sign}(1, n)$
- Ⓗ $\text{int}(5/\text{real}(n))$
- Ⓘ $5/n$
- Ⓚ $5.0/n$

FORTRAN History

- ◉ **FOR**mula **TRAN**slation invented 1954–8
by John Backus and his team at IBMVersion history
 - FORTRAN 1957
 - FORTRAN II
 - FORTRAN IV
 - FORTRAN 66 (released as ANSI standard in 1966)
 - FORTRAN 77 (ANSI standard in 1977)
 - FORTRAN 90 (ANSI standard in 1990)
 - FORTRAN 95 (latest ANSI standard version)
- ◉ Large majority of existing engineering software is coded in FORTRAN (various versions)

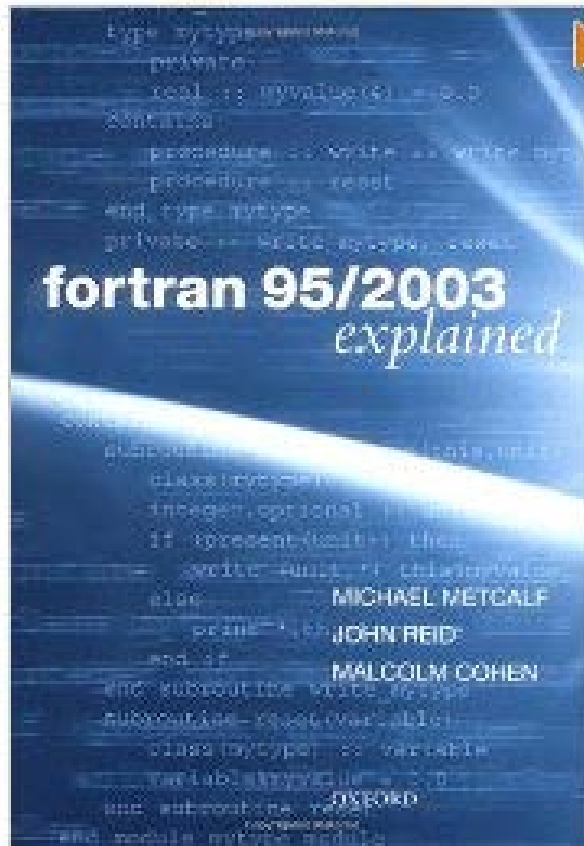
Why FORTRAN

- ⦿ FORTRAN was created to write programs to solve scientific and engineering problems
- ⦿ Introduced integer and floating point variables
- ⦿ Introduced array data types for math computations
- ⦿ Introduced subroutines and subfunctions
- ⦿ Compilers can produce highly optimized code (fast)
- ⦿ Lots of available numerical-math libraries
- ⦿ Lots of scientific programs written using Fortran

FORTRAN Today

- ⦿ FORTRAN 77 is “standard” but FORTRAN 90/95 has introduced contemporary programming constructs
- ⦿ There are proprietary compilers
 - Intel Fortran; Compaq/HP Visual Fortran; Absoft Fortran; Lahey Fortran
- ⦿ There is a free compiler in Unix-Linux systems
 - f77, g77
 - g95, gfortran
- Available scientific libraries
 - LINPACK: early effort to develop linear algebra library
 - MKL: commercial library
 - IMSL: commercial library
 - NAG: commercial library

Click to **LOOK INSIDE!**



Recommended book

Fortran 95/2003 Explained

(Numerical Mathematics and Scientific Computation)

Michael Metcalf, John Reid, Malcolm Cohen

Home - Graduate Program x


https://wiki.ucl.ac.uk/display/GradPrgCrS/Home

TV-Stream.to - Das ... Live Bus Departures ... index Raumklang Dresden... VFK - Dresden Jörg ... Willkommen bei SMV Resolve a DOI Live Bus Departures ... PaStiX Other bookmarks

UCL WIKI

UCL

Dashboard > Graduate Programming Courses > Home Browse Sergey Yurchenko Search

 Home Edit Share Add Tools

Added by [Deborah E Pollard](#), last edited by [Christian Hill](#) on Oct 01, 2012 ([view change](#)) ([show comment](#))

Graduate Programming Courses

These 3-day courses are intended for Graduate students at UCL, and will be of particular interest to First Year Ph.D. students in the Physical Sciences. Teaching takes the form of lecturer presentations interspersed with assisted exercises, and no previous programming experience is assumed.

Courses are offered in the following programming languages:

- Fortran 95
- C++
- Python

The course lecturers are Dr Christian Hill, Dr Sergey Yurchenko and Martin Uhrin.

Labels: None

3 Child Pages [Reorder Pages](#) | [Add Child Page](#)

Yurchenko - Leverhu....pdf Benchmarks.ps London_Dole_Nov20....pdf 12DiLaBr.C2H6.pdf gen.bib Show all downloads...

EN 10:45

Exercise: Which of these are legal Fortran constants?
What are their types?

- | | | |
|---------------|--------------------|----------------|
| (i) . | (ii) 3. | (iii) 3.1 |
| (iv) 31 | (v) 0. | (vi) +2 |
| (vii) -E18 | (viii) "ACHAR(61)" | (ix) 3 500 |
| (x) 4,800,000 | (xi) "X or Y" | (xii) "X"//"Y" |
| (xiii) 4.8E6 | (xiv) 5000E-3 | (xv) "VAT 69" |
| (xvi) 6.6_big | (xvii) (1, -1) | (xviii) 007 |
| (xix) 1E | (xx) -630958813365 | |

How Do I Run The Program?

- First, prepare the program using an editor to enter the program text.
 - A plain text editor such as nedit, vi, nano
 - Save the program text with the suffix .f90 (e.g. hello.f90)
- Run the FORTRAN compiler taking its input from this file and producing an executable program
- For example, run the following from the command window:

```
$ gfortran4 -static hello.f90 -o hello.x  
$ ./hello.x
```

First Fortran program

```
!  
! PURPOSE: Our first program  
!  
  program hello  
  
    implicit none  
    ! Variables  
  
    ! Body of hello  
    print *, 'Hello, World'  
  
  end program hello
```

Compile and run

```
$ gfortran -static hello.f90 -o hello.x
```

```
$ ./hello.x
```

```
Hello, World
```

Comments

```
!  
! PURPOSE: Our first program  
!  
program hello  
  
    implicit none  
    ! Variables  
  
    ! Body of hello  
    print *, 'Hello, World'  
  
end program hello
```

Not Useful comments:

! Add 1 to a

a = a + 1

• More Useful:

! Increment to account for new user login

a = a + 1

• Sometimes, Not Necessary:

NumUsersLoggedIn = NumUsersLoggedIn + 1

First Fortran program

```
!  
! PURPOSE: Our first program  
!  
  program hello  
  
    implicit none  
    ! Variables  
  
    ! Body of hello  
    ! print *, 'Hello, World'  
    write(*,*) 'Hello, World'  
  
  end program hello
```

Programming is the implementation of algorithms

Recipe CHOCOLATE CAKE

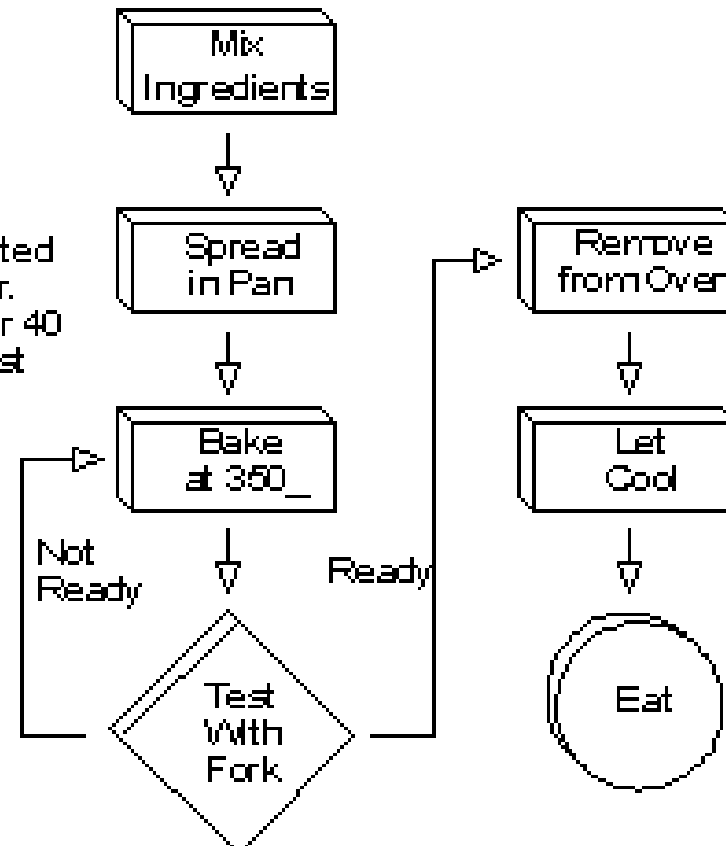
4 oz. chocolate 3 eggs
1 cup butter 1 tsp. vanilla
2 cups sugar 1 cup flour

Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

Declare variables:
chocolate eggs mix
butter vanilla
sugar flour

```
mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
  bake (mix, 350)
```



The Square Root Algorithm

To find the square root of a number, divide the number by your guess and average.

- ⦿ A is the number whose square root is desired;

$$A = 12.0$$

- ⦿ X is our estimate of its square;

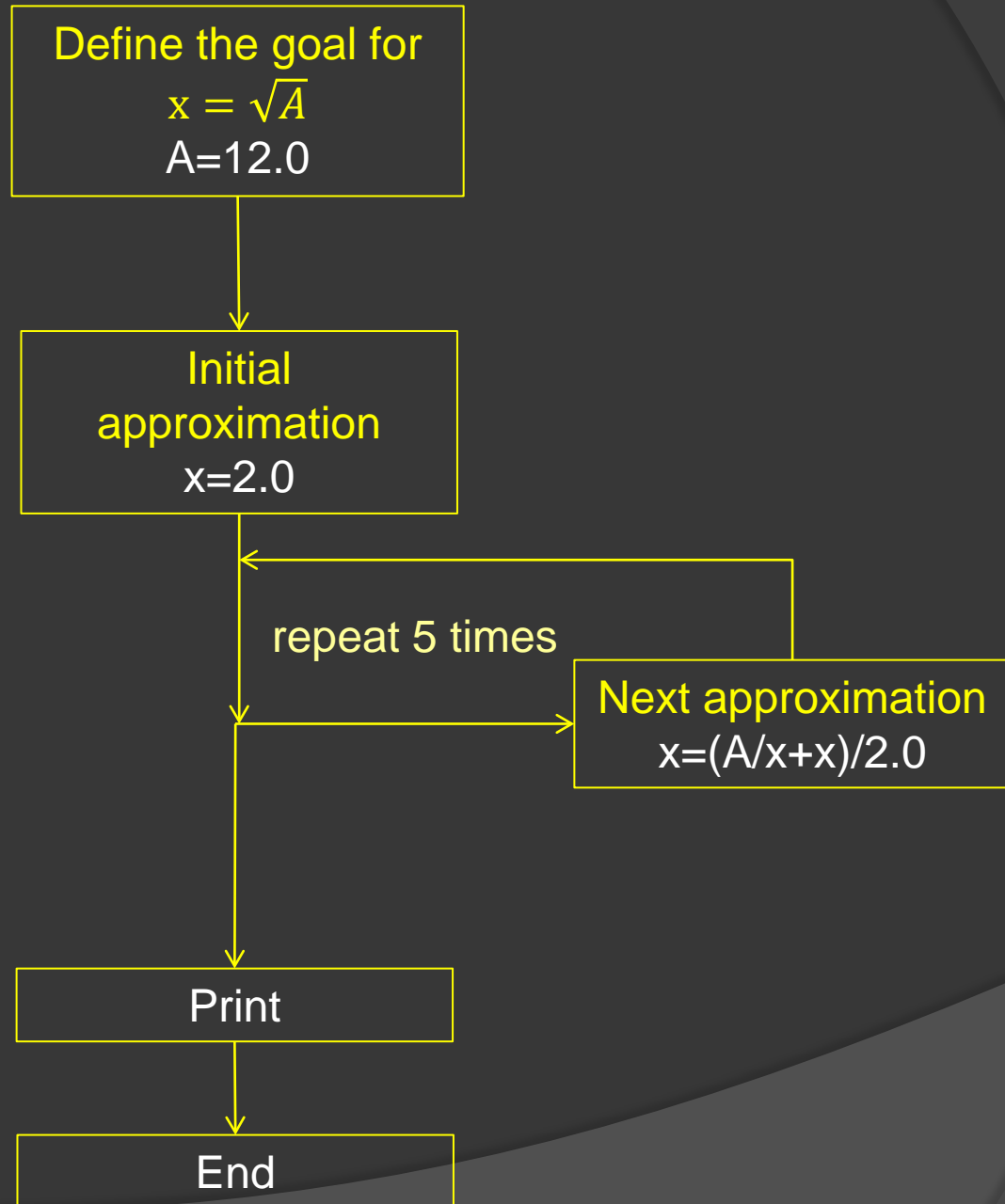
$$X = 2.0$$

- ⦿ Our goal: $X == A/X$

- ⦿ Average X and A/X to make a new estimate;

$$X = (X + A/X) / 2.0$$

- ⦿ We repeat this averaging process "several times".



Example: sqrt2.f90

```
program sqrt2
  implicit none
  integer :: i
  real    :: a,x

  a = 12.0
  x = 2.0

  print *, ''
  print *, 'Number whose square root is desired, A = ', a
  print *, 'Initial estimate X = ', x
  !
  ! Instead of repeating this loop 5 times, we could
  ! stop early if X is close enough.
  !
  do i = 1, 5
    x = ( x + a / x ) / 2.0
    print *, 'New X = ', x
  end do

  print *, ''
  print *, 'Correct square root is ', sqrt ( A )

  stop
end program sqrt2
```

Example: sqrt2.f90

```
$ gfortran sqrt2.f90 -o sqrt2.x  
$ ./sqrt2.x
```

Number whose square root is desired, A = 12.000000

Initial estimate X = 2.0000000

New X = 4.0000000

New X = 3.5000000

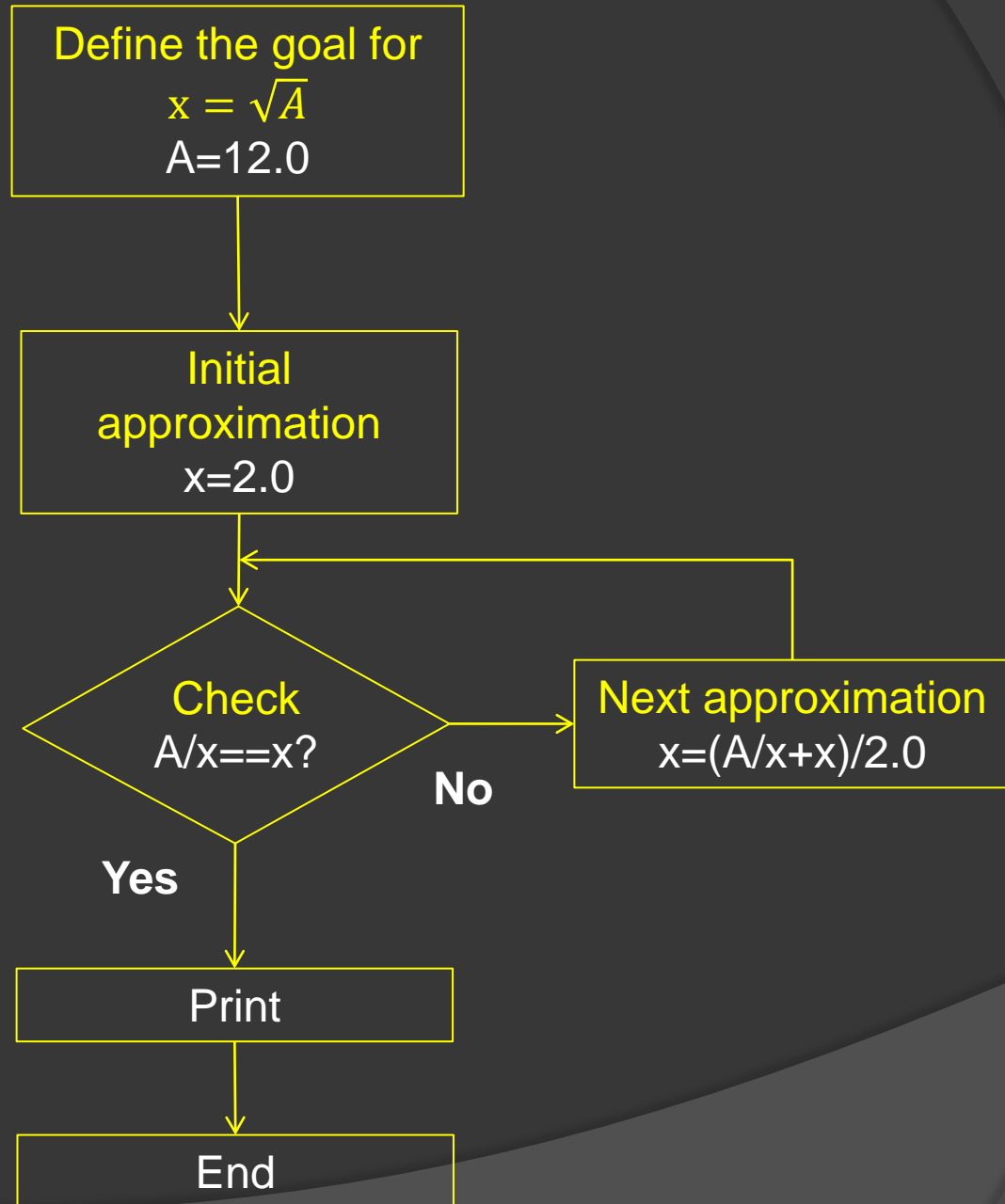
New X = 3.4642859

New X = 3.4641018

New X = 3.4641018

Correct square root is 3.4641016

! Instead of repeating this loop 5 times, we could
! stop early if X is close enough



Using IF

```
program sqrt2
  implicit none
  integer :: i
  real    :: a,x

  a = 12.0
  x = 2.0

  print *, ''
  print *, 'Number whose square root is desired, A = ', a
  print *, 'Initial estimate X = ', x
  !
  ! Instead of repeating this loop 5 times, we could
  ! stop early if X is close enough.
  !
  do
    x = ( x + a / x ) / 2.0
    print *, 'New X = ', x
    if ( abs(x-sqrt(12.0))<0.0001 ) exit
  end do

  print *, ''
  print *, 'Correct square root is ', sqrt ( A )

  stop
end program sqrt2
```

How to Write a Computer Program

There are 4 main steps:

1. specify the problem
2. analyse and break down into a series of steps towards solution
3. write the Fortran 95 code
4. compile and run(i.e test the program).

It may be necessary to iterate between steps and in order to remove any mistakes.

The testing phase is very important.

! =====

! This program implements a simple molecular dynamics simulation,

! using the velocity Verlet time integration scheme. The particles



! nsteps = number of time steps in the simulation

! mass = mass of the particles

! dt = time step

integer, parameter :: ndim=3, nparts=500, nsteps=1000

real(8) :: mass=1.0d0

real(8) :: dt=1.0e-4

! simulation variables

real(8) :: box(ndim) **! dimensions of the simulation box**

real(8) :: position(ndim,nparts), velocity(ndim,nparts)

real(8) :: force(ndim,nparts), accel(ndim,nparts)

real(8) :: potential kinetic E0 xx rij(ndim) d v dv

Conditional Statements

We can't allow a starting guess of 0!

```
if ( x == 0.0 ) then
    print *, 'Fatal error!'
    print *, ' The starting guess was X = ', x
    print *, ' but this is not allowed.'
    stop
end if
```

Exercise: Write equivalent Fortran 95 rational expression

e.g. $X > Y$ is the same as $X .Gt. Y$

1. $X .Eq. Y$

2. $X .Ne. Y$

3. $X .Lt. Y$

4. $X .Le. Y$

5. $X .Gt. Y$

6. $X .Ge. Y$

If $X < Y$, what is the value of Z ?

$Z = X .Ge. Y$

Relational Expressions

- Two expressions whose values are compared to determine whether the relation is true or false
 - may be numeric (common) or non-numeric
 - Relational operators:

Operator	Relationship
.LT. or <	less than
.LE. or <=	less than or equal to
.EQ. or ==	equal to
.NE. or /=	not equal to
.GT. or >	greater than
.GE. or >=	greater than or equal to

- Character strings can be compared
 - done character by character
 - shorter string is padded with blanks for comparison

Conditional Statements

The operators allowed to be used are the arithmetic comparisons:

`(x == y) (x > y) (x >= y)`
`(x /= y) (x < y) (x <= y)`

and the logical operators **.NOT.**, **.AND.** and **.OR.:**

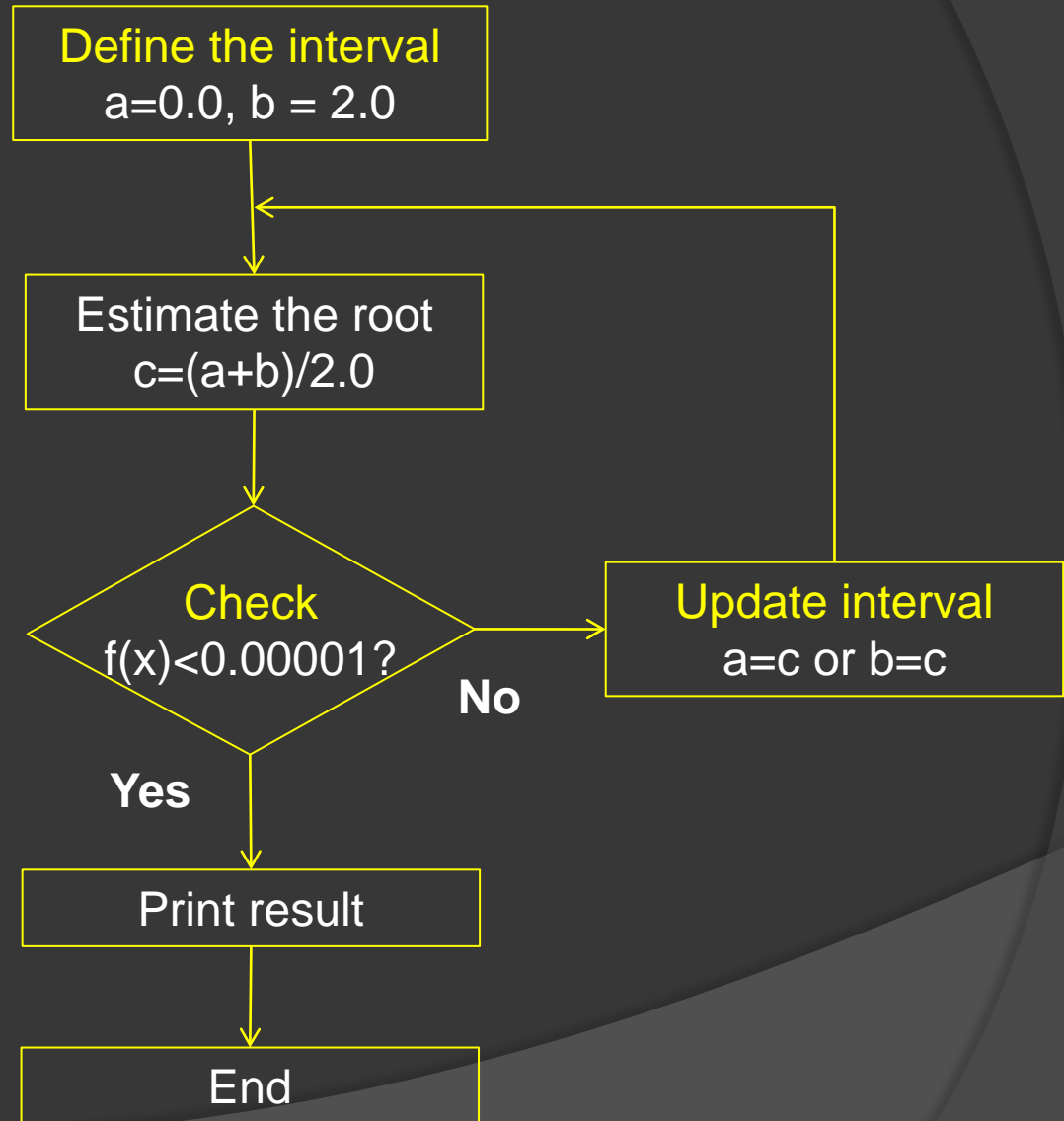
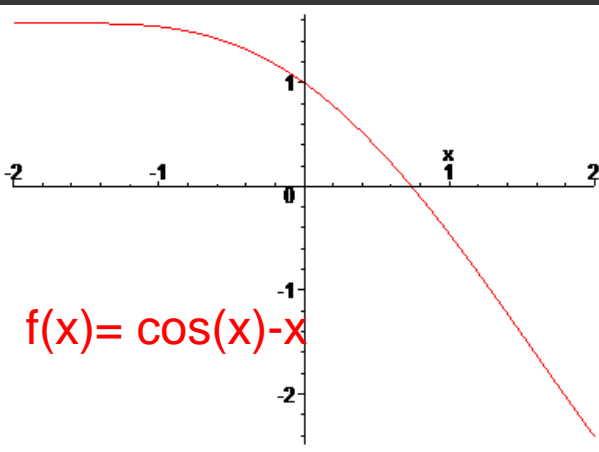
`(x < y .and. y < z)`
`(.not. (sin (x) == 0.0))`
`(i == 1 .or. i == n)`

Conditional Statements

The **IF** statement has an expanded form that includes **ELSE IF** and **ELSE** statements:

```
if ( x > 0 ) then
    abs = x
else if ( x < 0 ) then
    abs = - x
else
    abs = 0
end if
```

Solve $\cos(x)=x$ using the **bisecting** method the interval $[a,b]$



```
program bisect
```

```
implicit none
```

```
real :: a,b,c
```

```
! A and B are the endpoints of an interval to be searched.
```

```
! We require that F(A) and F(B) have opposite signs.
```

```
!
```

```
a = 0.0
```

```
b = 2.0
```

```
if ( ( f(a) > 0.0 .and. f(b) > 0.0 ) .or. &
      ( f(a) < 0.0 .and. f(b) < 0.0 ) ) then
```

```
  print *, 'Fatal error! The function does not change sign at the endpoints.'
```

```
  stop
```

```
end if
```

```
do      ! "DO forever"
```

```
  c = ( a + b ) / 2.0
```

```
  if ( f(c) == 0.0 ) then
```

```
    exit
```

```
  else if ( abs ( f(c) ) < 0.00001 ) then
```

```
    exit
```

```
  end if
```

```
  if ( sign ( 1.0, f(c) ) == sign ( 1.0, f(a) ) ) then
```

```
    a = c
```

```
  else
```

```
    b = c
```

```
  end if
```

```
end do
```

```
print *, 'Estimated root occurs at X = ', c
```

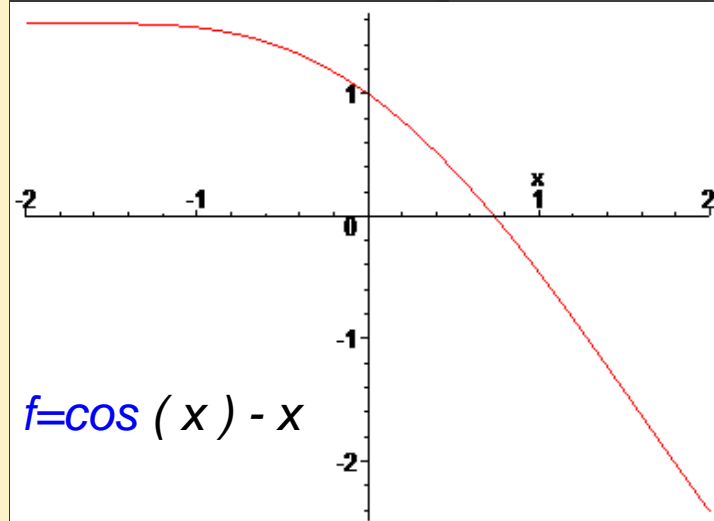
```
print *, 'Value of function is F(X) = ', f(c)
```

```
stop
```

```
end program
```

```
function f ( x )
```

```
.....
```



```
$gdortran4 bisect.f90
```

```
$/a.out
```

```
Estimated root occurs at X = 0.7390900
```

```
Value of function is F(X) = -8.1062317E-06
```

```
.....
```

```
function f ( x )
```

```
  f = cos ( x ) - x
```

```
  return
```

```
end function
```

Program Structure

PROGRAM *name*

IMPLICIT NONE

declarations

statements

END PROGRAM *name*

Program statement, and a matching End statement

A declaration statement that declares that "a,b,c" are real variables

Adding comments for this purpose is called program documentation.

```
program bisect
implicit none
```

```
real :: a,b,c
```

```
! A and B are the endpoints of an interval to be searched.
```

```
! We require that F(A) and F(B) have opposite signs
```

```
!
```

```
a = 0.0
```

```
b = 2.0
```

```
if ( ( f(a) > 0.0 .and. f(b) > 0.0 ) .or. ( f(a) < 0.0 .and. f(b) < 0.0 ) ) then
```

```
  print *, 'Fatal error! The function does not change sign at the endpoints.'
```

```
  stop
```

```
end if
```

```
do ! "DO forever"
```

```
  c = ( a + b ) / 2.0
```

```
  if ( f(c) == 0.0 ) then
```

```
    exit
```

```
  else if ( abs ( f(c) ) < 0.00001 ) then
```

```
    exit
```

```
  end if
```

```
  if ( sign ( 1.0, f(c) ) == sign ( 1.0, f(a) ) ) then
```

```
    a = c
```

```
  else
```

```
    b = c
```

```
  end if
```

```
end do
```

```
print *, 'Estimated root occurs at X = ', c
```

```
print *, 'Value of function is F(X) = ', f(c)
```

```
stop
```

```
end program ←
```

```
function f ( x )
```

```
.....
```

The arithmetic operators, which are used to do calculations, are:

+, for addition,

-, for subtraction,

*, for multiplication, as in $0.5 * X$,

/, for division, as in a / x ,

, for exponentiation, as in $x^{}2$,

: Checking a condition has the form:

If (X. Eq. Y) Then

(statements to carry out if X is equal to Y)

End If

We can also stack conditions together, as in this example:

If (X. Lt. Y) Then

(statements to carry out if X is less than Y)

Else If (X .Gt. Y) Then

(statements to carry out if X is greater than Y)

End If

Points Raised: The previous program introduces some new ideas

Comments - anything on a line following a is ignored

& - means the line is continues

IF construct - different lines are executed depending on the value of the Boolean expression

relational operators - == (is equal to) or <(less then)

procedure call - COS(x), f(x)

Bugs: Compile time Errors

- In previous program what if we accidentally typed

`d = (a + b) / 2.0`

```
$ gfortran bisect.f90 -o bisect.x  
bisect.f90:21.5:
```

```
  d = ( a + b ) / 2.0  
  1
```

```
Error: Symbol 'd' at (1) has no IMPLICIT type
```

Bugs: Compile time Errors

- In previous program what if we accidentally typed

```
c = ( a + b ) / 0.0
```

then the program would compile but we would lead to an unpredictable result (Chaos).

Declarations and specifications

```
PROGRAM seconds
```

```
! Comments start with an exclamation mark
```

```
IMPLICIT NONE
```

```
INTEGER :: hours, mins, secs, temp
```

```
PRINT *, 'Type the hours, minutes and seconds'
```

```
READ *, hours, mins, secs
```

```
temp = 60*( hours*60 + mins) + secs
```

```
WRITE (*,*) 'Time in seconds =', temp
```

```
END PROGRAM seconds
```

This program, called seconds.f90, can be compiled:

```
$ gfortran4 seconds.f90
```

and run:

```
$ a.out
```

Names

- Up to 31 letters, digits and underscores
- Names must start with a letter
- Upper and lower case are equivalent
- DEPTH, Depth and depth are the same name
- The following are valid Fortran names

A, AA, aaa, Tax, INCOME, Num1, NUM2, NUM333,
N12MO5, atmospheric_pressure, Line_Colour,
R2D2, A_21_173_5a

Invalid Names

The following are invalid names

1A does not begin with a letter

_B does not begin with a letter

Depth\$0 contains an illegal character '\$'

A-3 would be interpreted as subtract 3 from A

B.5: illegal characters '.' and ':'

A_name_made_up_of_more_than_31_letters
too long, 38 characters

Analysis of Program

The code is delimited by PROGRAM ... END PROGRAM statements.
Between these there are two distinct areas.

PROGRAM seconds

! Comments start with an exclamation mark

IMPLICIT NONE

INTEGER :: hours, mins, secs, temp

PRINT *, 'Type the hours, minutes and seconds'

READ *, hours, mins, secs

temp = 60*(hours*60 + mins) + secs

WRITE (*,*) 'Time in seconds =', temp

END PROGRAM seconds

Specification Part

specifies named memory locations (variables) for use

specifies the type of the variable,

Execution Part

reads in data

calculates the temp and
prints out results.

Implicit vs Explicit Declarations

```
PROGRAM seconds
```

```
! Comments start with an exclamation mark
```

Don't!

```
!IMPLICIT NONE
```

```
INTEGER :: ihours, jmins, ksecs, ltemp
```

```
PRINT *, 'Type the hours, minutes and seconds'
```

```
READ *, ihours, jmins, ksecs
```

```
ltemp = 60*( ihours*60 + jmins) + ksecs
```

```
WRITE (*,*) 'Time in seconds =', ltemp
```

```
END PROGRAM seconds
```

Good habit: force **explicit** type declarations

```
IMPLICIT NONE
```

User must explicitly declare all variable types

Specification part

`IMPLICIT NONE` --- this should always be present. Means all variables must be declared.

- `INTEGER :: hours, mins, secs, temp` --- declares three INTEGER (whole number) variables.

Other variable types:

```
real(8)           :: energy      ! total energy value in Hartree
integer(4)        :: igamma      ! irrep index (1,2,3,4) for A1,A2,B1,B2
character(len=2)  :: symmetry(4) = ("/A1","A2","B1","B2"/) ! an irrep of C2v
logical          :: symmetric = .true. ! the parity of the state
real(4)          :: erange(4) ! lowest energy of each irrep
type(symmetryT)  :: symmetry ! definition of the point group symmetry used
!
real(8), parameter :: pi      = 4.0_rk * atan2(1.0_rk,1.0_rk) ! PI=3.14...
real(8), parameter :: twopi = 2.0_rk * pi ! 2*PI=6.28...
real(8), parameter :: sqrt2 = sqrt(2._rk) ! \sqrt{2}

! physical constants -- All constants updated 21 March 2012 from the NIST
real(8), parameter :: planck = 6.62606957e-27_rk ! Planck constant in erg*second
real(8), parameter :: avogno = 6.02214129e+23_rk ! Avogadro constant
```


Integers

Typically ± 2147483647 (-2^{31} to $2^{31}-1$)

INTEGER, INTEGER(4)

Sometimes $\pm 9.23 \times 10^{17}$ (-2^{63} to $2^{63}-1$)

INTEGER(8)

Real Constants

- Real constants must contain a decimal point or an exponent

They can have an optional sign, just like integers

The basic fixed--point form is anything like:

123.456, -123.0, +0.0123, 123., .0123
0012.3, 0.0, 000., .000

- Optionally followed **E** or **D** and an exponent

1.0E6, 123.0D-3, .0123e+5, 123.d+06, .0e0

1E6 and 1D6 are also valid Fortran real constants

Reals

32–, 64– and 128–bit formats are:

real(4): 10^{-38} to 10^{+38} and 6–7 decimal places ; the same as **real**

double precision 10^{-308} to 10^{+308} and 15–16 decimal places

real(16): 10^{-4932} to 10^{+4932} and 33–34 decimal places

Complex Numbers

This course will generally ignore them
If you don't know what they are, don't worry

These are (**real**, **imaginary**) pairs of REALs
i.e. Cartesian notation

Constants are pairs of reals in parentheses

E.g. **(1.23, -4.56)** is **1.23-4.56i**
or **(-1.0e-3, 0.987)** is **-0.001+0.987i**

Logical Type

These can take only two values: **true** or **false**
.TRUE. and **.FALSE.**

- Their type is **LOGICAL**

LOGICAL :: red, amber, green

IF (red) THEN

PRINT *, 'Stop'

red = .False. ; amber = .True. ; green = .False.

ELSIF (red .AND. amber) THEN

...

Character Type

Used when **strings of characters** are required
Names, descriptions, headings, etc.

- Fortran's basic type is **a fixed-length string**

Unlike almost all more recent languages

- Character constants are quoted strings

PRINT *, 'This is a title'

PRINT *, "And so is this"

The characters between quotes are the value. . .

What are the substrings

(i) `a = "mulligatawny"`

(ii) `s(7:8)`

(iii) `a(1:4)`

(iv) `a(6:6)`

(v) `a(10:8)`

Giving a Variable a Value

The simplest assignment statement has the form:

Variable name = Constant

Examples:

X = 1.3

Number = 17

Failing = .True. !Assigning a value to a LOGICAL variable

Name = 'Francis' !Assigning a value to a CHARACTER variable

Exercise: Write type declaration statements to declare

- (i) Three real variables called power1, power2 and creeper;
- (ii) Two integers called kappa and kappa_prime;
- (iii) Two strings, each of four characters, called v and w.

Exercise: Which of these are legal Fortran constants?
What are their types?

- | | | |
|---------------|--------------------|----------------|
| (i) . | (ii) 3. | (iii) 3.1 |
| (iv) 31 | (v) 0. | (vi) +2 |
| (vii) -E18 | (viii) "ACHAR(61)" | (ix) 3 500 |
| (x) 4,800,000 | (xi) "X or Y" | (xii) "X"//"Y" |
| (xiii) 4.8E6 | (xiv) 5000E-3 | (xv) "VAT 69" |
| (xvi) 6.6_big | (xvii) (1, -1) | (xviii) 007 |
| (xix) 1E | (xx) -630958813365 | |

Giving a Variable a Value

The more interesting form of the assignment statement is:

Variable name = Expression

Examples:

$X = X + 1$

$Y = (Z + 3) * (Z - 4)$

$\text{Score} = (\text{Poise} + \text{Difficulty} + \text{Speed}) / 3.0$

$Z = \text{Sqrt}(X)$!This takes the square root of X

Execution: Expressions and Assignment

Program Sumup

Real Average

Real Sum

Sum = 0

Sum = Sum + 2

Sum = Sum + 16

Object → Sum = Sum + 3 ← **Expression**

Average = Sum / 3

Write (*,*) 'The sum is ', Sum

Write (*,*) 'The average value is ', Average

Stop

End

Using temporary variables

Compare the following versions of the same computation:

```
X=LENGTH**2+WIDTH*RATE*TIME/FACTOR
```

```
X = LENGTH**2 + WIDTH * RATE * TIME / FACTOR
```

```
X = Length**2 + (Width*Rate*Time)/Factor
```

```
Area1      = Length**2
```

```
Distance   = Rate*Time
```

```
Area2      = Width*Distance
```

```
Area2      = Area2/Factor
```

```
X          = Area1 + Area2
```

Operator Precedence

Fortran uses normal mathematical conventions

- Operators bind according to precedence
- And then generally, from left to right

The precedence from highest to lowest is

- ** exponentiation
- * / multiplication and division
- + - addition and subtraction

- Parentheses ('(' and ')') are used to control it

Use them whenever the order matters or it is clearer

Examples

$X + Y * Z$ is equivalent to

$X + Y / 7.0$ is equivalent to

$A - B + C$ is equivalent to

$A + B ** C$ is equivalent to

$- A ** 2$ is equivalent to

$A - (((B + C)))$ is equivalent to

$X + (Y * Z)$

$X + (Y / 7.0)$

$(A - B) + C$

$A + (B ** C)$

$-(A ** 2)$

$A - (B + C)$

- You can force any order you like

$(X + Y) * Z$

Adds X to Y and then multiplies by Z

Exercise: Evaluate the following

(i) 3^{**2**3}

(ii) $4+6/12*2$

(iii) $18/4/2^{**}(-1)$

(iv) $\text{SQRT}(4^{**3**2})$

(v) $9/8*7/6*5/4*3/2$

(vi) $9*8/7*6/5*4/3*2$

(vii) $6-4+1$

(viii) $16.0^{**3.0}/2.0$

(ix) $(-1)^{**0.5**}(-1)$

(x) $2-1^{**0.5**}(-1)$

Exercise: Evaluate the following

If a , b , c , d and e are the names of real variables, write Fortran expressions for:

- (i) The average of a , b , c , d and e ;
- (ii) The root-mean-square of a , b and c ;
- (iii) b , expressed as a percentage of the total of a , b , c , d and e ;
- (iv) The geometric mean of a and b .

Warning

$X + Y + Z$ may be evaluated as any of
 $X + (Y + Z)$ or $(X + Y) + Z$ or $Y + (X + Z)$ or . . .

They are all mathematically equivalent
But may sometimes give slightly different results

Precedence Problems

Mathematical conventions vary in some aspects

$A / B * C$ – is it $A / (B * C)$ or $(A / B) * C$?

$A ** B ** C$ – is it $A ** (B ** C)$ or $(A ** B) ** C$?

Fortran specifies that:

$A / B * C$ is equivalent to $(A / B) * C$

$A ** B ** C$ is equivalent to $A ** (B ** C)$

- Yes, ****** binds from right to left!

Mixing integers and reals

Example: mix.f90

Program Mix

In most cases, you can mix **Integer** and **Real** quantities

Integer I

Integer J

Real X

Real Y

I = 2 ; J = 3.5

X = 2 ' Y = 3.5

Write (*,*) 'I=', I, ' and X=', X

I = 2.3 ; X = 2.3

Write (*,*) 'I=', I, ' and X=', X

I = Y ; X = J

Write (*,*) 'I=', I, ' and X=', X

I = J ; X = Y

Write (*,*) 'I=', I, ' and X=', X

I = 2.3 * J * Y

X = 2.3 * J * Y

Write (*,*) 'I=', I, ' and X=', X

Stop
End

```
$ gfortran mix.f90 -o mix.x
$ ./mix.x
I=          2  and X=    2.0000000
I=          2  and X=    2.3000000
I=          3  and X=    3.0000000
I=          3  and X=    3.5000000
I=         24  and X=   24.149998
```

Integer Division

INTEGER :: K = 9, L = 5, M = 3, N

REAL :: X, Y, Z

X = K ; Y = L ; Z = M

N = (K/L)*M

N = (X/Y)*Z

Exercise: Integer Division

Predict results of the following

Real :: X

X = 5/3 !This will result in X=

X = 5.0/3.0 !This will result in X=

The same problem happens when the quantities to be divided are variables:

I = 5

J = 3

X = I/J !This will result in X =

Here, you have to ask Fortran not to use **Integer** division by requesting that it temporarily treat **I** and **J** as **Real** values:

X = Real(I) / REAL(J) !This will result in X=

Integer Division

$X = 5/3$!This will result in $X=1.0$

$X = 5.0/3.0$!This will result in $X=1.6666....$

The same problem happens when the quantities to be divided are variables:

$I = 5$

$J = 3$

$X = I/J$!This will result in $X = 1.0$

Here, you have to ask Fortran not to use **Integer** division by requesting that it temporarily treat **I** and **J** as **Real** values:

$X = \text{Real}(I) / \text{REAL}(J)$!This will result in $X=1.6666....$

Mixed Expressions

INTEGER and **REAL** is evaluated as **REAL**

Either and **COMPLEX** goes to **COMPLEX**

Be careful with this, as it can be deceptive

```
INTEGER :: K = 5
```

```
REAL    :: X = 1.3
```

```
X = X+K/2
```

That will add 2.0 to **X**, not 2.5

K/2 is still an **INTEGER** expression

Conversions

There are several ways to force conversion

- Intrinsic functions **INT**, **REAL** and **COMPLEX**

$$X = X + \text{REAL}(K)/2$$
$$N = 100 * \text{INT}(X/1.25) + 25$$

You can use appropriate constants

You can even add zero or multiply by one

$$X = X + K/2.0$$
$$X = X + (K + 0.0)/2$$

The last isn't very nice, but works well enough

DO: Repeated Execution

Do Loop

```
Do I = 1, 10  
    Write (*,*) 'Current estimate is ', Xroot  
    Xroot = (Xroot + (X/Xroot) ) / 2.0  
End Do
```

DO: Repeated Execution

count.f90

Program Count

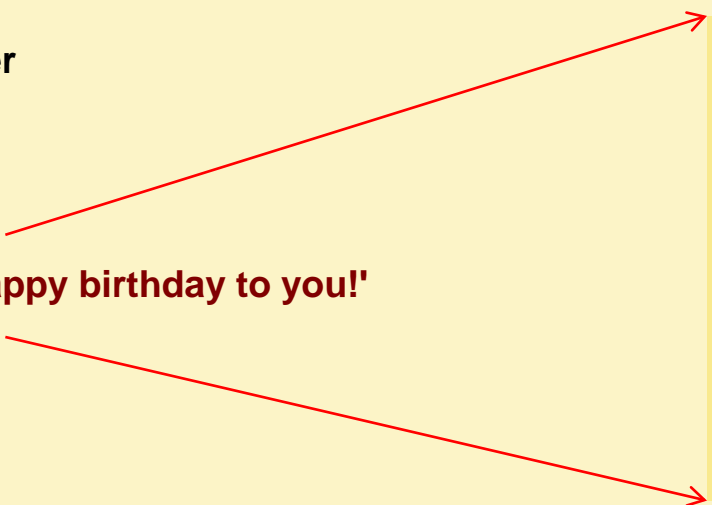
```
integer :: Age
integer :: I
integer :: J
integer :: Number
integer :: Sum
!
Age = 12
Do I = 1, Age
  Write (*,*) 'Happy birthday to you!'
End Do
!
Sum = 0
Number = 0
Do J = 1, Age
  Number = Number + 1
  Sum = Sum + Number
End Do
!
Write (*,*) 'You"ve blown out ', Sum , ' candles in your life!'
!
Stop
End Program Count
```

```
$ gfortran count.f90 -o count.x
$ ./count.x
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
Happy birthday to you!
You"ve blown out 78 candles in your life!
```

DO: Repeated Execution

Program Count

```
integer :: Age
integer :: I
integer :: J
integer :: Number
integer :: Sum
!  
Age = 12  
Do I = 1, Age  
  Write (*,*) 'Happy birthday to you!'  
End Do  
  
!  
Sum = 0  
Number = 0  
Do J = 1, Age  
  Number = Number + 1  
  Sum = Sum + Number  
End Do  
  
!  
Write (*,*) 'You"ve blown out ', Sum, ' candles in your life!'  
!  
Stop  
End Program Count
```



```
I = 1  
Write (*,*) 'Happy birthday to you!'  
I = 2  
Write (*,*) 'Happy birthday to you!'  
...  
I = Age  
Write (*,*) 'Happy birthday to you!'
```

Using MOD(A,N) function

```
If ( Mod ( Age, 2 ) .Eq. 0 ) Then  
    Write(*,*)'Age is even.'  
Else  
    Write(*,*)'Age is odd.'  
End If
```

IF: Conditional Execution

⦿ $X = \text{Sqrt} (Y)$

if (Y >= 0.0) then

X = Sqrt (Y)

end

if (Y .Ge. 0.0) Then

!

X = sqrt (Y)

write (*,*) 'The Square Root of Y is ', X

!

else

!

write (*,*) 'Your number was negative!'

write (*,*) 'We cannot take the square root of a negative number.'

!

end if

Multi-way IFs

```
IF (expr == val1) THEN
    x = 1.23
ELSE IF (expr >= val2 .AND. expr <= val3) THEN
    CONTINUE
ELSE IF (expr == val4) THEN
    x = x + 4.56
ELSE
    x = 7.89 - x
END IF
```

Very commonly, expr is always the same and all of the vals are constant expressions. Then there is another way of coding it

SELECT CASE

```
PRINT *, 'Happy Birthday'
SELECT CASE (age)
CASE(18)
    PRINT *, 'You can now vote'
CASE(40)
    PRINT *, 'And life begins again'
CASE(60)
    PRINT *, 'And free prescriptions'
CASE(100)
    PRINT *, 'And greetings from the Queen'
CASE DEFAULT
    PRINT *, 'It''s just another birthday'
END SELECT
```


Arrays in FORTRAN

- ⦿ Arrays can be multi-dimensional (up to 7) and are indexed using ():
 - **TEST(3)**
 - **FORCE(4,2)**
- ⦿ Indices are normally defined as 1...N
- ⦿ Can specify index range in declaration
 - **REAL L(2:11,5)** – L is dimensioned with rows numbered 2-11 and columns numbered 1-5
 - **INTEGER K(0:11)** – K is dimensioned from 0-11 (12 elements)
- ⦿ Arrays are stored in column order (1st column, 2nd column, etc) so accessing by incrementing row index first usually is fastest.
- ⦿ Whole array reference:
 - **K=-8** - assigns 8 to all elements in K

Exercise: Ranks, Extents etc.

Give the rank, bounds, size and shape of the arrays defined as follows

	rank	bounds	size	shape
REAL DIMENSION(1:10):: ONE				
REAL DIMENSION(2,0:2) ::TWO				
INTEGER DIMENSION(-1:1,3,2) ::THREE				
REAL DIMENSION(0:1,3) ::FOUR				

Terminology

```
REAL :: A(0:99), B(3, 6:9, 5)
```

The **size** is the total number of elements

A has size 100 and B has size 60

The **shape** is its rank and extents

A has shape (100) and B has shape (3,4,5)

Arrays are **conformable** if they share a shape

- The **bounds** do not have to be the same

Arrays: example

```
INTEGER, DIMENSION(0:99) :: arr1, arr2, arr3
```

```
INTEGER, DIMENSION(1:12) :: days_in_month
```

```
CHARACTER(LEN=10), DIMENSION(1:250) :: names
```

```
CHARACTER(LEN=3) :: months(1:12)
```

```
REAL :: box_locations(1:350)
```

```
REAL, DIMENSION(-10:10, -10:10) :: pos1, pos2
```

```
REAL :: bizarre(0:5, 1:7, 2:9, 1:4, -5:-2)
```

OPENING FILES