Introduction To FORTRAN: Outline

Takis Tsoutsanis

Cranfield University
Computational Engineering Sciences

October 2024

Objectives

The objectives of the introductory sessions are to:

- Provide overview of FORTRAN language and its key elements
- Describe and provide hands-own training in key resources and tools
- Introduce FORTRAN programming via a series of tasks

Recommended Resources

- S.J. Chapman, Fortran 90/95 for Engineers & Scientists, 2nd ed. 2004
- archer2 tutorials
- On Canvas:
  - Language Reference (PDF) from INTEL & PGI
  - A number of online tutorials
  - Links to Free Compilers

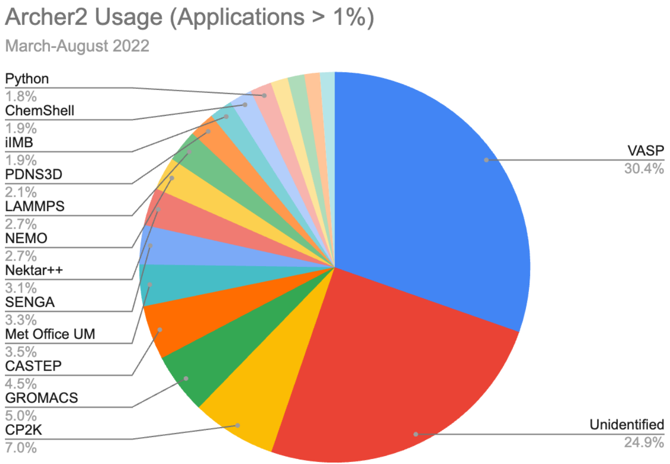
History

FORmula TRANslation

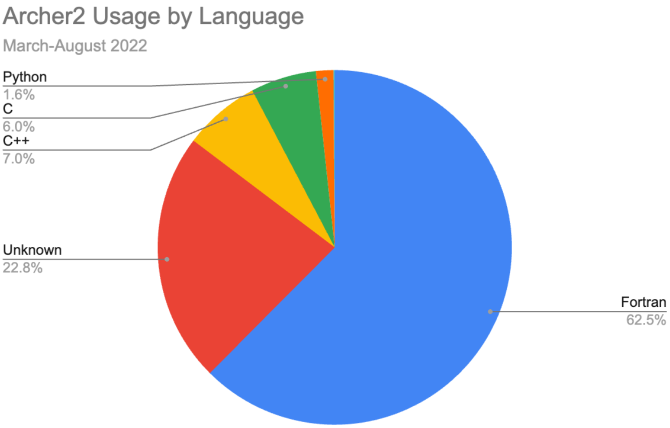
- 1957 - FORTRAN I, first attempt at a high-level computing language, fixed form
- 1958 - FORTRAN II enhancements and fixes ... until
- 1962 - FORTRAN IV - ANSI standard "FORTRAN 66", stable for 15 years until
- 1977 - FORTRAN 77 - major update, operations on character variables, IFs, etc
- 1990 - FORTRAN 90 - free-source, array operations, allocatable memory, derived data-types
- 1997 - FORTRAN 95 - minor changes in the standard, FORALL, additional intrinsic functions (ex. CPU_TIME)
- ... FORTRAN 2003 - ongoing, no complete implementation available yet

NOTE: Backwards compatibility introduces hideous 77 features into modern compilers.

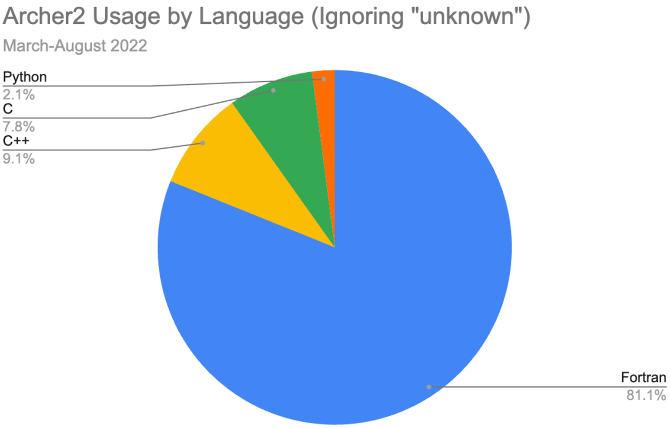
Why FORTRAN?


Archer2 Usage (Applications > 1%)
March-August 2022
Python 1.8%
ChemShell 1.9%
iIMB 1.9%
PDNS3D 2.1%
LAMMPS 2.7%
NEMO 2.7%
Nektar++ 3.1%
SENGA 3.3%
Met Office UM 3.5%
CASTEP 4.5%
GROMACS 5.0%
CP2K 7.0%
VASP 30.4%
Unidentified 24.9%

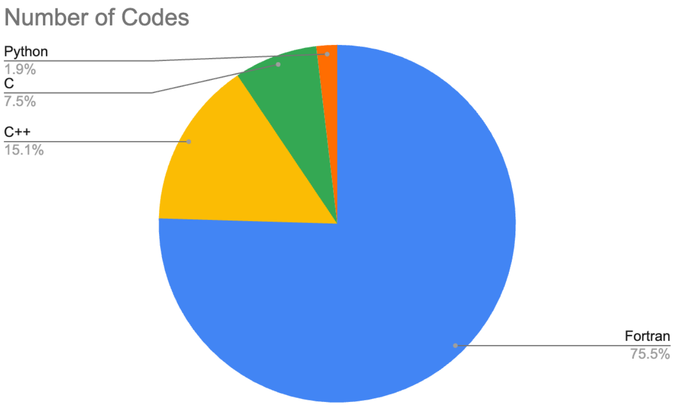
Why FORTRAN?


Archer2 Usage by Language
March-August 2022
Python
1.6%
C
6.0%
C++
7.0%
Unknown
22.8%
Fortran
62.5%

Why FORTRAN?


Archer2 Usage by Language (Ignoring "unknown")
March-August 2022
Python
2.1%
C
7.8%
C++
9.1%
Fortran
81.1%

Why FORTRAN?



Tools

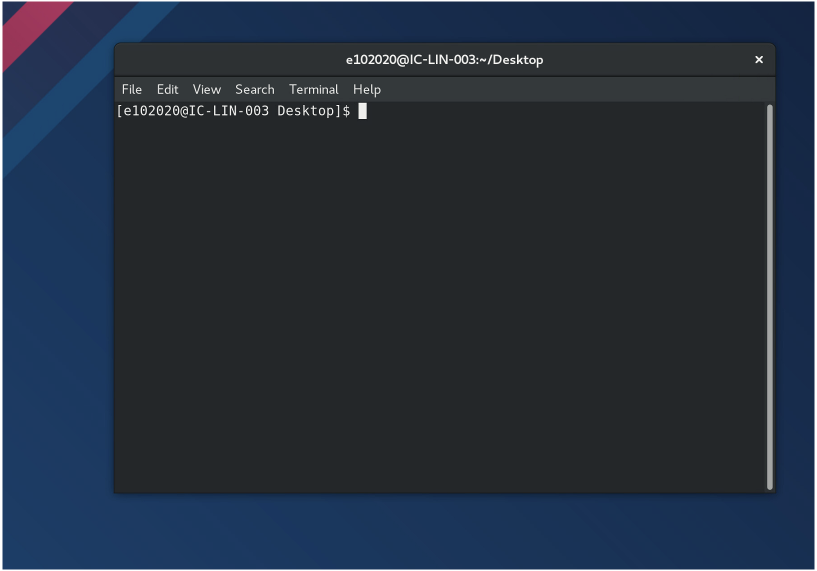
Most of our basic programming during labs will be in Linux. For simple programs we will need:

- An editor (Kate/VSCode/Gedit)
- A terminal
- A compiler (intel/gnu)
- A simple plotting utility & a not-so-simple plotting utility

Terminal 1


New Folder
Paste
Show Desktop in Files
Open in Terminal
Change Background...
Display Settings
Settings
Cranfield
University

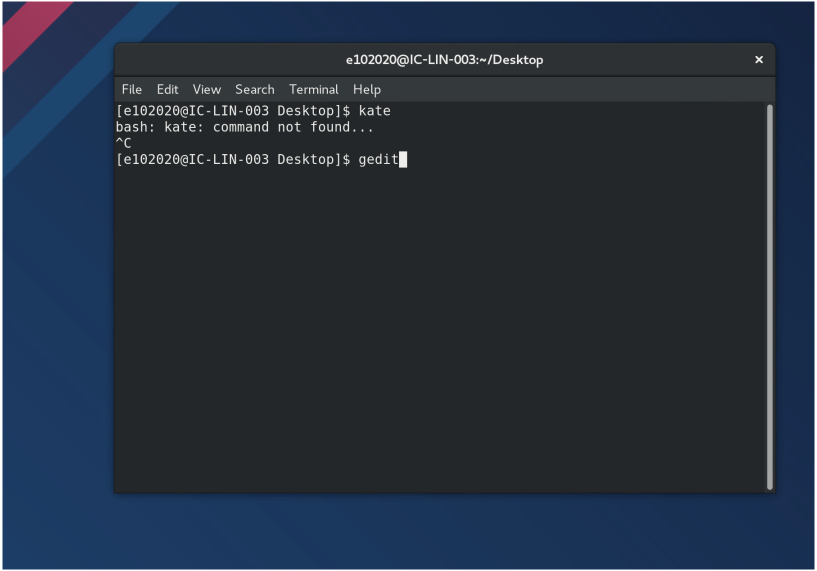
Terminal 2


e102020@IC-LIN-003:~/Desktop
File Edit View Search Terminal Help
[e102020@IC-LIN-003 Desktop]$

Terminal: Basic Commands

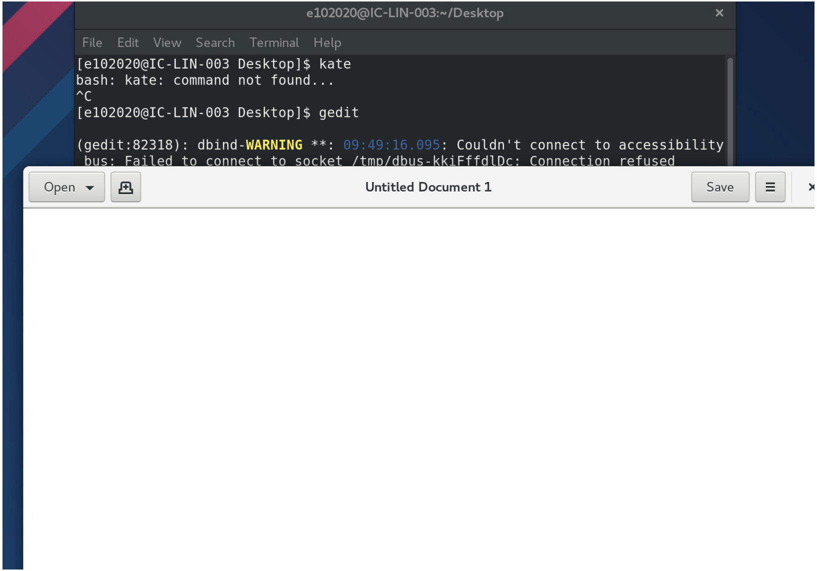
- “ls <destination>” - list the contents of the current directory
- “cd <destination>” - change the directory (“..” - indicates one leve up, Tab gives name completion)
- “cp <source> <destination>” - copy file
- “mv <source> <destination>” - move file
- man <command> - help on a command - best friend
- man -k “phrase” - search help for commands with descriptions containing the phrase

Gedit 1



e102020@IC-LIN-003:~/Desktop
File Edit View Search Terminal Help
[e102020@IC-LIN-003 Desktop]$ kate
bash: kate: command not found...
^C
[e102020@IC-LIN-003 Desktop]$ gedit

Gedit 2


e102020@IC-LIN-003:~/Desktop
File Edit View Search Terminal Help
[e102020@IC-LIN-003 Desktop]$ kate
bash: kate: command not found...
^C
[e102020@IC-LIN-003 Desktop]$ gedit
(gedit:82318): dbind-WARNING **: 09:49:16.095: Couldn't connect to accessibility
bus: Failed to connect to socket /tmp/dbus-kkiFffdlDc: Connection refused
Open
Untitled Document 1
Save

Compiler

Intel Fortran compiler is invoked by:

```
ifort <options> -o <executable name> <source file>
```

Useful options for debugging (8.1 version):

```
-warn all - enables all warning messages
-check all - enables all run-time checks
-traceback - reports the line where the run-time error occurred
-fpe0 - run-time floating-point error check
```

For example the most informative debugging compilation of ex.f90 source will be given by:

```
ifort -warn all -traceback -check all -fpe0 -o ex ./ex.f90
```

Compiler

Useful options for optimisation:

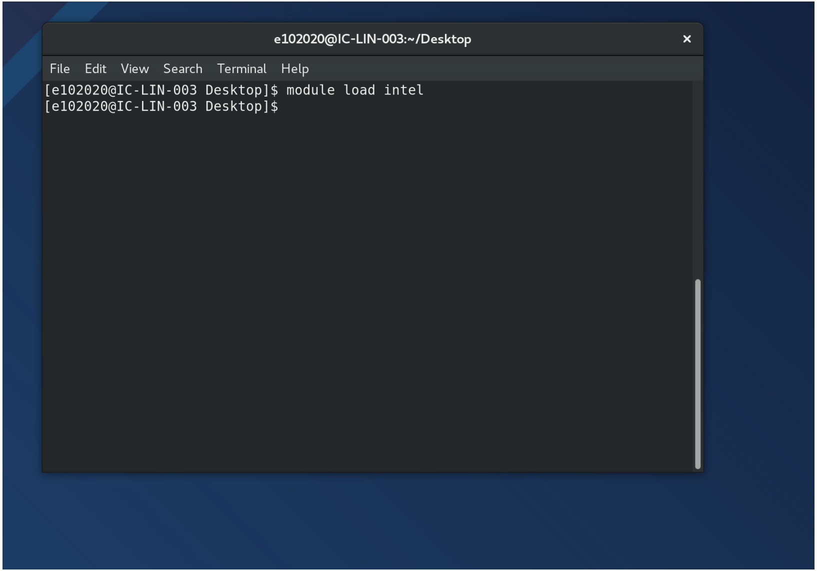
```
-O<n> - where n=1..3 optimisation level
-ip - interprocedural optimisations
-x<K,W,N,B,P>  - optimises for a specific processor
```

Other useful options:

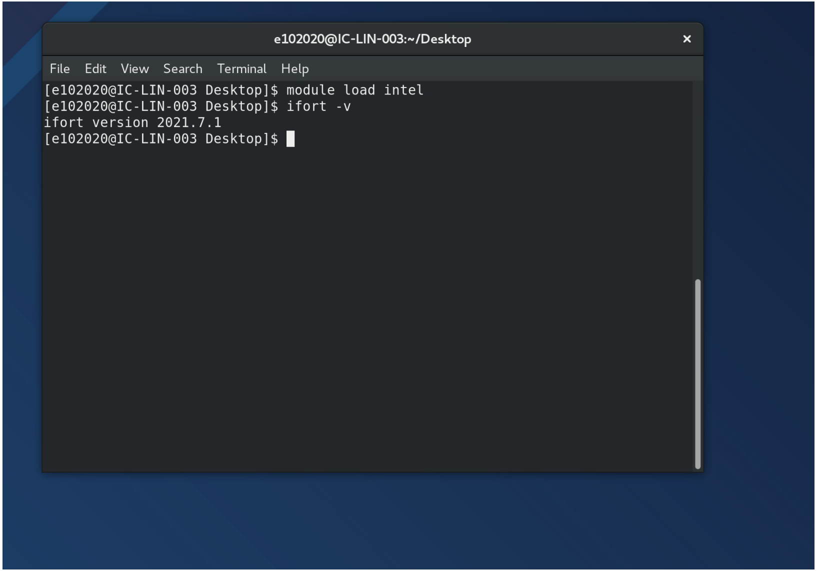
```
-r8 - default floating point numbers is double-precision
-i8 - default integer numbers is double-precision
```

For the complete list of options:

```
man ifort
```

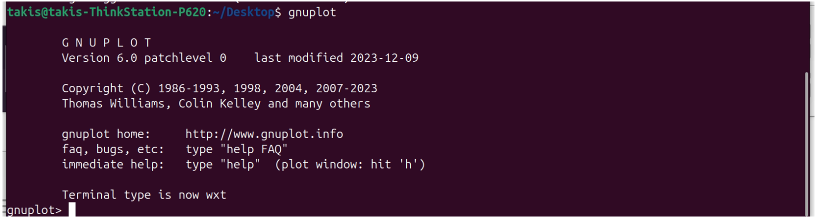

e102020@IC-LIN-003:~/Desktop
File Edit View Search Terminal Help
[e102020@IC-LIN-003 Desktop]$ module load intel
[e102020@IC-LIN-003 Desktop]$

Fortran 2


e102020@IC-LIN-003:~/Desktop
File Edit View Search Terminal Help
[e102020@IC-LIN-003 Desktop]$ module load intel
[e102020@IC-LIN-003 Desktop]$ ifort -v
ifort version 2021.7.1
[e102020@IC-LIN-003 Desktop]$

GNUPlot

GNUPlot - www.gnuplot.info, t16web.lanl.gov/Kawano/gnuplot/index-e.html



Basic GNUPlot Commands

2D Plotting, datafile in the form:

```
# Col1 Col2 Col3 ...
<num 1> <num 2> <num 3> ...
<num 1> <num 2> <num 3> ...
```

- 2D plot on screen:

```
plot "file" using <xcol>:<ycol>, "file" using <xcol>:<ycol>...
```

- 2D plot to file

```
set term post
set out "filename.ps"
plot "file" using <xcol>:<ycol>, "file" using <xcol>:<ycol>...
```

Basic GNUPlot Commands

3D Plotting, datafile in the form:

```
# Col1 Col2 Col3 ...
<X> <Y> <Z> ...
...
blank line before next row in X
<X> <Y> <Z> ...
```

- 3D plot on screen:

  `splot “filename” using 1:2:3 with lines`

- Main command!:

  `help`

Tecplot

- One of the standard post-processing package
- 1D, 2D, 3D steady and unsteady data visualisation
- Flow analysis, extraction of flow properties
- Data manipulation
- Macros/scripts

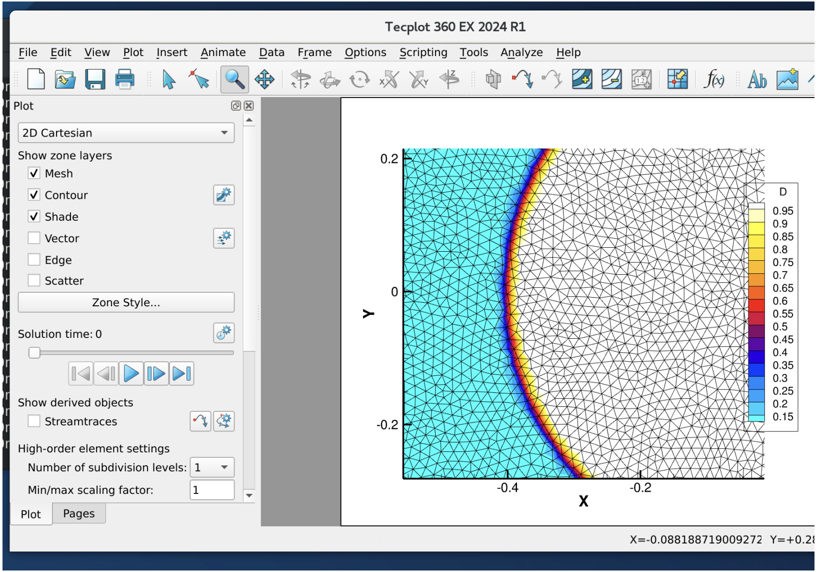
Tecplot

e102020@IC-LIN-003:~/Desktop

File Edit View Search Terminal Help

```
[e102020@IC-LIN-003 Desktop]$ module load tecplot360ex
tecplot360ex                 tecplot360ex/2022r2_linux64
tecplot360ex/2021r1_linux64  tecplot360ex/2023r1_linux64
tecplot360ex/2021r2_linux64  tecplot360ex/2023r2_linux64
tecplot360ex/2022r1_linux64  tecplot360ex/2024r1_linux64
[e102020@IC-LIN-003 Desktop]$ module load tecplot360ex/2024r1_linux64
[e102020@IC-LIN-003 Desktop]$ tec360
```

Data concepts: Data file, Data Layout



FORTRAN Data Types

The following data types are available in FORTRAN 90:

- Integer
- Floating Point (real, double)
- Logical (boolean)
- Character
- Complex

Implicit typing

By default variables with names starting with i,j,k,l,m,n are integers other variables are real. This feature should NOT be used. It is turned off by the "implicit none" statement in the beginning of the program.

Integers & Reals

Integer variable declaration with default precision

```
integer :: iIntegerVariable
```

Integer variable declaration with specified precision

```
integer((kind=)<1,2,4 or 8>) :: iIntegerVariable
```

Real variable declaration with default precision

```
real :: rRealVariable
```

Real declaration variable with specified precision

```
real((kind=)<4,8 or 16>) :: rRealVariable
```

"double precision" type is equivalent to real(8)

Logical & Character

Logical variable declaration with default precision

```
logical :: lCondition
```

Example of values

```
lCondition=.true.
```

Character variables

```
character :: cOneSymbol
character (LEN=<length>) :: sString
```

Strings are treated as arrays, for example

```
character (10) :: sSourceString, sDestString
sSourceString=''Today''
sDestString=sSourceString(3:5)
(sDestString now is ''Day'')
```

Constants

Constants are defined using “parameter “ statement, for example:

```
real, parameter :: rZero = 1.e-8
integer, parameter :: iMaxNodes=10000
```

Named constants help to avoid redundant definitions and variation of constant’s precision across the program.

Arrays

Static arrays of variables:

```
real, dimension (i1:i2,j1:j2...) :: rArray
```

Dynamic array of variables

```
real, allocatable, dimension (:,:,:....) :: rArray
```

Example

```
!Declare
real, allocatable, dimension (:,:) :: rArray
integer iError
!allocate
allocate (rArray(20,2:5),stat=iError)
deallocate(rArray)
```

Status is an integer variable 0 - for success, otherwise - an error code. Good practice: always check the error code.

Array Operations Examples

```
real, dimension (10) :: rA1, rA2, rA3
```

Definition (example)

```
rA1=(/(0.1*i,i=1,10)/)
rA2=(/(10.+0.1*i,i=1,10)/)
```

Whole array operations

```
rA3=rA1+rA2
```

Subsections

```
rA3(2:5)=rA1(3:6)
```

Array WHERE

"Where" operation allows you to iterate through the array and apply a conditional operation without using loops:

```
real, dimension (10) :: rA1, rA2, rA3
rA1=(/(0.1*i,i=0,9)/)
rA2=(/(10.+0.1*i,i=0,9)/)
where (rA1 /= 0.)
    rA3=rA2/rA1
elsewhere
    rA3=1.
end where
```

Note: Mask is evaluated once in the beninning of the complete operation only i.e. if the operation changes the mask - it will not affect the result.

Structural Units

- Program - main entry point and main flow control.
- Subroutine - a sub-programs which can take anumber of arguments.
- Functions - a subroutine which returns one value and can be used in expressions.
- Module - a container (compare with a Class in C)

Typical Program

```
! Example of a program
     Program Very_Simple_One
     implicit none
! Declare an integer variable and a floating point variable
     integer :: i
     real :: r
! read the variable from console
     write (*,*) 'Give me a number, please: '
     read (*,*) i
! do some tricky stuff
     r=sqrt(sin(abs(real(i))))
! output result
     write (*,*) 'Our trickery resulted in: ', r
! finalise and exit
     stop
     end program Very_Simple_One
```

Subroutine

Sub-program without a mandatory return value

```
! Example of a subroutine: compute a quadratic
    subroutine Quad (a,b,c,x,result)
    implicit none
    real,intent(in):: a,b,c,x
    real,intent (out) :: result
!compute the result
    result=a*x**2+b*x+c
! finalise and exit
    end subroutine Quad
```

Note, arguments must be of correct type. Using:

```
call Quad(1.,1.,.1,0.2, rResult)
```

Function

Defining

```
! Example of a function: compute a quadratic
    real function rQuad (a,b,c,x)
    implicit none
    real,intent(in):: a,b,c,x
! compute the result
    rQuad=a*x**2+b*x+c
! finalise and exit
    end function rQuad
```

Note, arguments must be of correct type. When the function is used, the calling subroutine must define variable rQuad. Using:

```
print *, "Qudratic: ", rQuad(1.,1.,.1,0.2)
```

Module

Defining

```
! Example of a module
    module ModuleExample
    implicit none
    <data declaration>
    contains
    <subroutines & functions>
    end module ModuleExample
```

Accessing a module

```
! Example of a program using module
    program Example
    use ModuleExample

    ...

    ...
    end program Example
```

Conditions

Multiple IF

```
if (a<b) then
...
else if (b<a<c) then
...
else
...
end if
```

Select case

```
select case
select case (variable)
...
case (value1)
...
case (value2)
...
case default
...
end select
```

Loops

The following loop control structures are available in FORTRAN

- Blank do: do ... end do
- Explicit while: do while (condition) ... end do
- Iterator: do <iterator> ... end do

Blank do

Do without a condition, loops untill we exit explicitly

```
do
...
if (condition) exit
...
end do
```

Explicit While

Loop with an explicit condition tested on each entrance

```
do while (condition)
...
...
end do
```

Iterator

Loop with an integer counter

```
do i=1,100
...
...
end do
```

- Naming:
  For complicated structure of nested loops it is beneficial to name the loops explicitly:

  ```
  loop1: do i=1,100
  ...
  ...
  end do loop1
  ```

- Cycle statement - continues to the next iteration of the loop
- Exit statement - exits the loop

File Input/Output

- open a file and associate it with a unit (handle)

```
open (<unit>, file=<filename>,iostat=<error variable>,
specifiers list)
for example:
open (11,file=''input.dat'',iostat=iErr,form=''formatted'',
status=''replace'')
Good practice:  always check the error code.
```

- close a file

```
close (unit)
```

Input/Output

- Unified output statement:

```
write (unit,format) <data>
```

- Unified input statement

```
read (unit, format) <data>
```

Formats

Format statements are used to format data on read or write, for example:

```
write (*,'(3I4,E15.7)') i,j,k, Velocity(i,j,k)
```

writes 3 integers in 4 characters field and one floating-point value in engineering notation and 7 digits after the decimal point in 15 characters field.

Style Rules

- Design before programming
  - Outline the inputs
  - Outline the outputs
  - Draw the algorithm chart
- Implement in incremental steps where each sub-step can be tested separately
- Comment every step
- Use meaningful variable names

Task 1: Simple Program

Create a simple program and test the following commands one at a time

- print*, 'Hello World'
- print*, 'Hello World, Pi =',4.D0*ATAN(1.D0)
- print '(a,G25,15)', 'Hello World, Pi =', 4.D0*ATAN(1.D0)
- Declare to integers I1 and I2, type
  - read*, I1,I2
  - print*, I1,I2,I1+I2
  - print'(3I3)',I1,I2,I1+I2
- How does the output change?

Task 2: Quadratic

Create a program which can read in the coefficients of a quadratic equation $ax^2+bx+c=0$ and output the roots

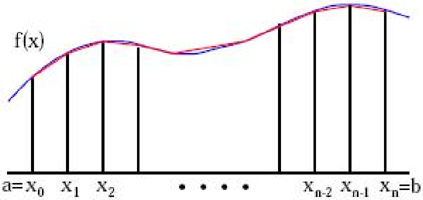
- put in a read statement to read user inputted values for a, b, and c
- solve the quadratic equation to find the roots $(=\frac{-b\pm\sqrt{b^2-4ac}}{2a})$
- also implement the formula which is more accurate when $b^2 >> 4ac$, where the roots are $\frac{2c}{-b\pm\sqrt{b^2-4ac}}$
- the most accurate formula is the one where both the square root and $-b$ are the same sign
- use the code to solve for how high a mobile telephone tower must be to transmit to the horizon at 1m, 100m, 1km and 100km away. The relationship between the height of the tower h, the distance to the horizon d and the radius of the earth R (=6350km) is $h^2+2hR=d^2$
- what is the effect of using the less accurate formula for calculating h when d is small?

Task 3: Pi Computation

- Well-known formula:

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi.$$

- Numerical integration (Trapezoidal rule):



$$\int_a^b f(x)\,dx \approx h[\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{n-1}) + \frac{1}{2}f(x_n)].$$

$x_i = a + ih, \quad h = (b-a)/n, \quad n = \#$ of subintervals.

Create a program which:

- Reads from a text parameter file the number of intervals to use for integration
- Integrates

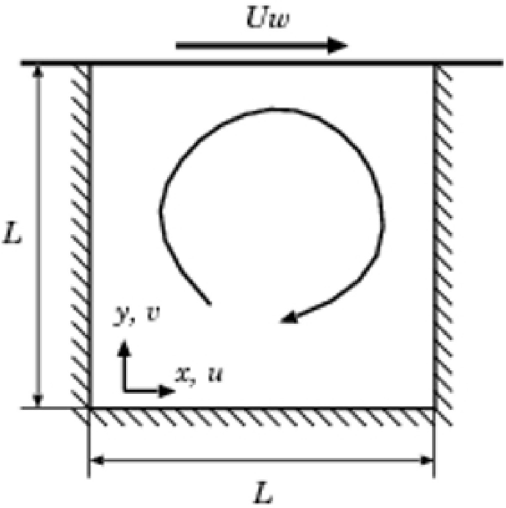
$$\int_0^1 \frac{4}{1+x^2}\,dx$$

  using piece-wise constant and piece-wise interpolation of function for:
  - constant interval size
  - cell size in $[0, 0.5]$ interval half that of cell size in interval $[0.5, 1.]$
- Computes deviation from the “correct value”

$$3.14159265358979323846$$

- Reports the computed values and deviation in % on screen.
- Reports the results of Pi computation and deviation from the correct value for the series of grids with $2, 4, 8, \ldots N$ cells where N is the number read from file.

Task 4: 2D Cavity Flow



Grid Mapping

Transformation x^{α} maps$[0,1] \rightarrow [0,1]$. However the uniform grid is clustered to 0. Why?

If our initial domain is

$$x \in [x_0, x_1]$$

then

$$\xi = \frac{x - x_0}{x_1 - x_0}$$

is from 0 to 1 and one-side biased clustering for an initial uniform grid is given by:

$$X^{*} = X_0 + (X_1 - X_0)\left(\frac{X - X_0}{X_1 - X_0}\right)^{\alpha}$$

To cluster to both sides, we need to treat two halves of the domain separately.

2D Cavity Grid

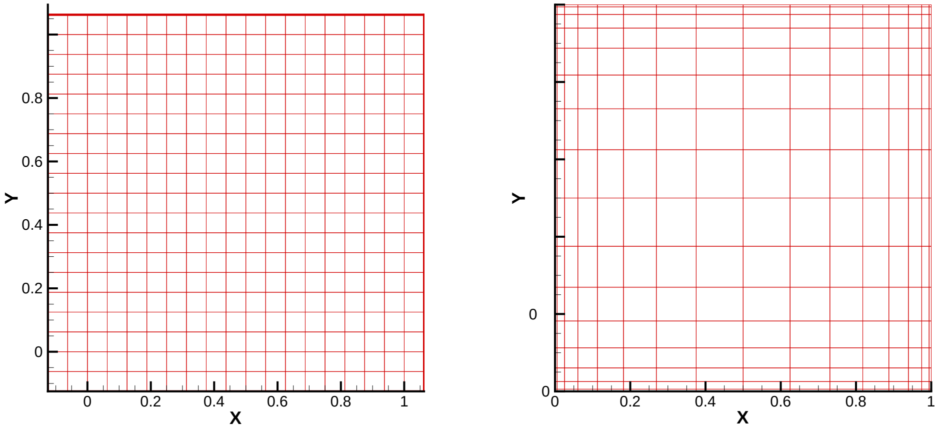
Create a program which generates a grid for a 2D cavity. The program should:

- Read from a text parameter file the number of intervals for the grid in X and Y, the aspect ratio (assume X dimension is 1) and clustering exponents for X and Y directions on the left and on the right.
- Based on this data, allocate storage for the grid and create the grid with the following options (per coordinate):
  - A uniform grid
  - A grid clustered to one or both sides using power law clustering $x \rightarrow \xi = x^{\alpha}$ with the exponents read from the input file.
- Output this grid in a format readable by Tecplot
- Save grid image to a jpeg file

Note: Clustering should be isolated from the rest of the code using either a subroutine or a function.

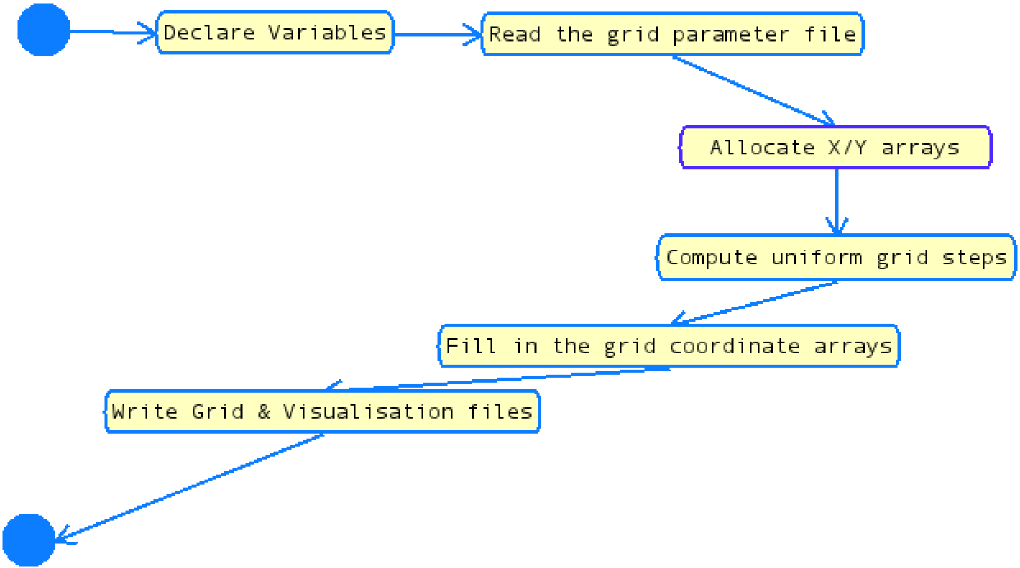
2D Cavity Grid: Clustering

The clustering for the cavity is performed by creating a uniform grid and then transforming the coordinates:



2D Cavity Grid Generator: Code Outline

Overall outline:



Grid Parameter File

Grid parameter file should include:

- Aspect Ratio
- Number of cells per direction
- Exponent for clustering per direction and per side
- Flag indicating whether to write a visualisation grid

Creating the Grid

For uniform grid, the coordinate of i, j point is given by (assuming indices start from 0):

$$(x, y)_{ij} = (x_0 + \Delta x \cdot i, y_0 + \Delta y \cdot j)$$

Note

Please keep full arrays $X(i,j)$ and $Y(i,j)$ despite the fact that X is a function of i alone and Y is the function of j alone. We might use this program to generate grids for which this is not the case later.

Mapping the Grid

In order to transform the uniform grid, a function should be created which takes the uniform coordinate and clustering parameters as inputs and returns the clustered value. For example:

$$rXClustered = Cluster\,(rX, rX0, rX1, rExpX0, rExpX1)$$

$$rYClustered = Cluster\,(rY, rY0, rY1, rExpY0, rExpY1)$$

Writing a Grid

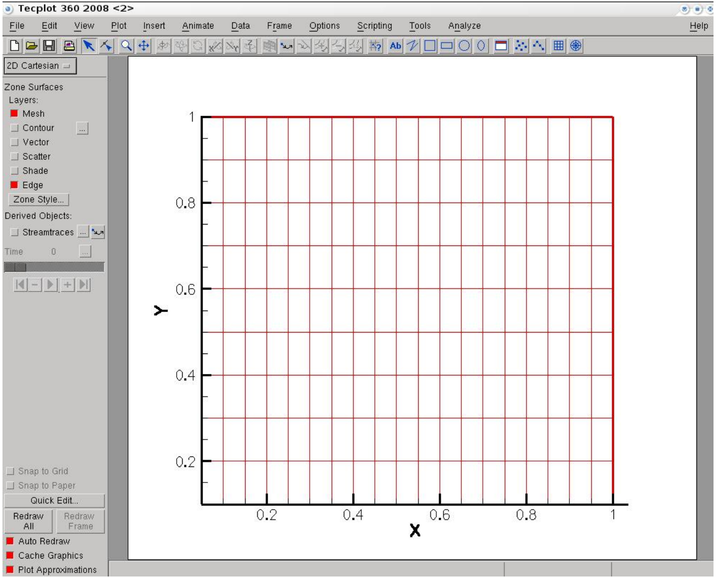
Tecplot 2D ASCII point data file format:

```
write(1,*) 'TITLE="SIMPLE GRID"'
write(1,*) 'VARIABLES="X" "Y"'
write(1,*) "ZONE I=20 J=10, F=POINT" do j=1,10
  do i=1,20
  write(1,*) 0.05*i,0.1*j
end do
end do
```

Writing a Grid


```
TITLE="SIMPLE GRID"
VARIABLES="X" "Y"
ZONE I=20 J=10, F=POINT
 5.0000001E-02  0.1000000
 0.1000000      0.1000000
 0.1500000      0.1000000
 0.2000000      0.1000000
 0.2500000      0.1000000
 0.3000000      0.1000000
 0.3500000      0.1000000
 0.4000000      0.1000000
 0.4500000      0.1000000
 0.5000000      0.1000000
 0.5500000      0.1000000
 0.6000000      0.1000000
 0.6500000      0.1000000
 0.7000000      0.1000000
 0.7500000      0.1000000
 0.8000000      0.1000000
 0.8500000      0.1000000
 0.9000000      0.1000000
 0.9500000      0.1000000
  1.000000      0.1000000
 5.0000001E-02  0.2000000
 0.1000000      0.2000000
```

Visualising the Grid



Writing an Image