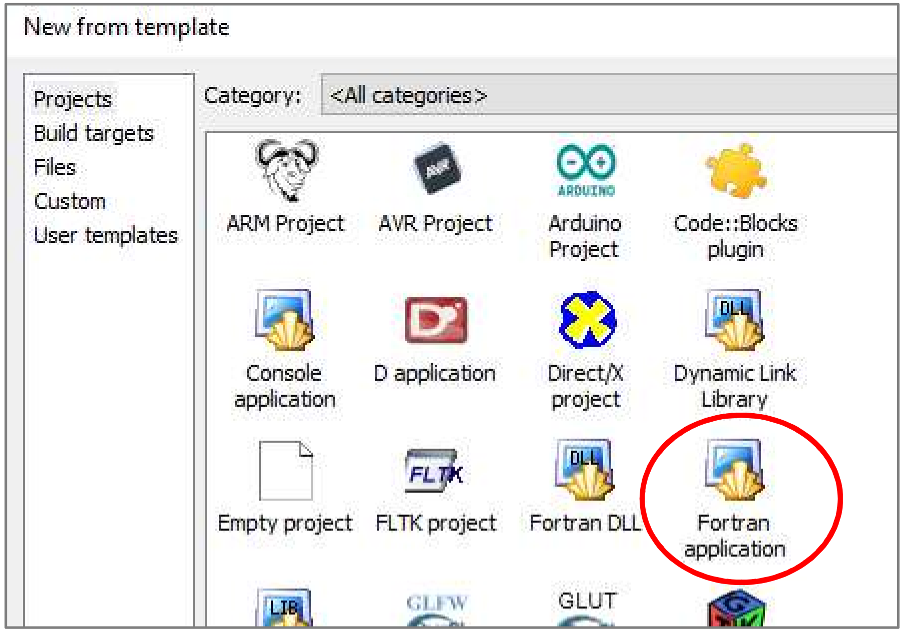
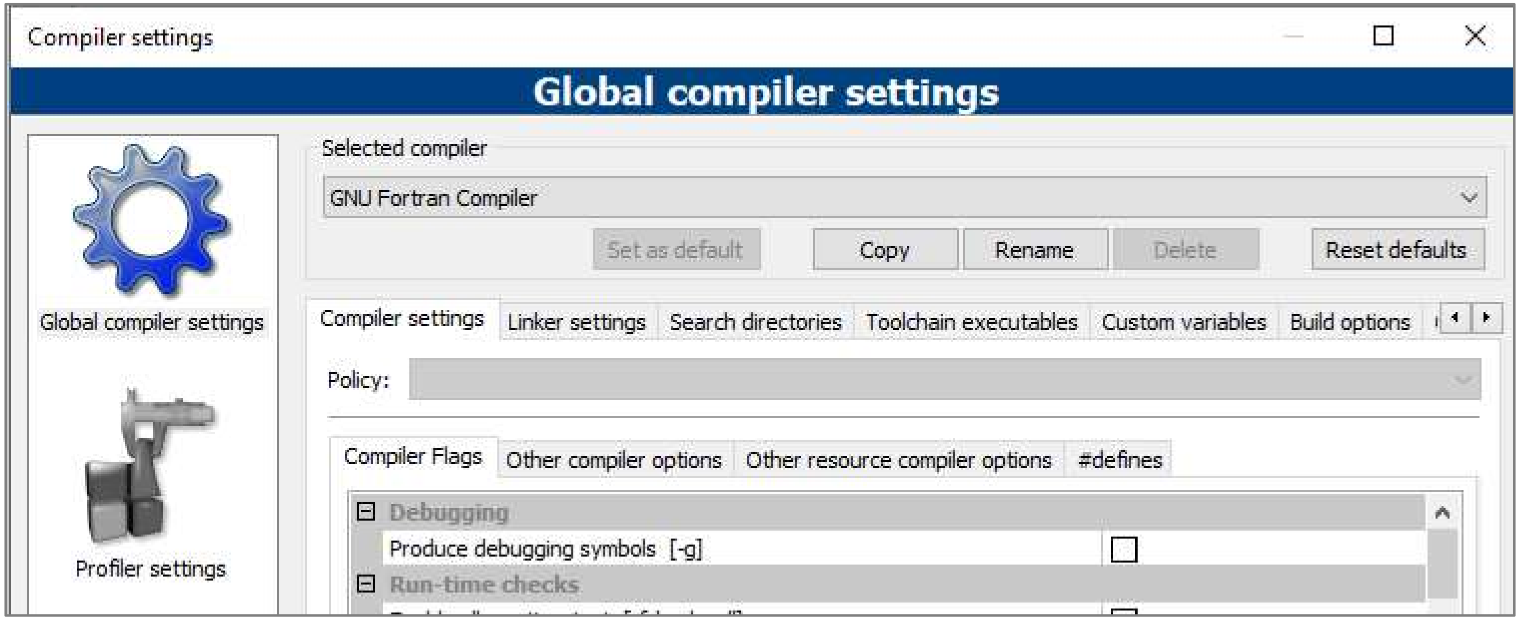
CSE 425: Concepts of Programming Languages

A brief
Introduction to Fortran

Md. Shahriar Karim
Assistant Professor
Department of Electrical and Computer Engineering
North South University
Bangladesh

Fortran95: CodeBlocks: Steps





Good programming style

- The logical structure of the program should be as easy as possible

    - Use comments for terms that are not self-explanatory. Also, avoid comments for obvious computational expressions

Similarly if you have a loop, a comment of the form below is of no help:

```
! loop from 1 to 10
do i=1,10
```

➢ But a comment of the following form, say in a program calculating a binomial might be very useful:

```
! loop to calculate nCr
do k=1,r
```

Image source: Dept. of Physics, University of Cambridge

- Indenting of the code blocks is highly appreciated, as it tends to improve the readability

Declarations/initialization of variables

Declarations:

```
! A fortran95 program for G95
! CSE 425, Md.Shahriar Karim
! Basic example 1
program main
  implicit none
  real:: a, b, total, pi, area, radius
  pi = 3.414
  a = 9
  b = 7
  total = a + b
  print *, 'The total is ', total    ! p
  write(*,*) 'The total is ', total    !
end program main    ! program main is en
!end  simple end would also be correct
```

- Turns off the implicit type definition and default naming convention initial Fortran version allowed
- :: is the separator,
  - necessary for type specification statement where the variable values are initialized
  - But is also used to define variable type without initializing the variable. For instance, both real :: a, b and real a, b are valid definition

Precision:

- KIND: This type parameter (KIND) is used to define the precision of a real, integer, complex, or logical variable.
- For example, a real number with at least 10 decimal places of precision with a range of at least -10^{34} *to* 10^{34} can be defined as:

REAL(KIND = SELECTED_REAL_KIND(10,34)) :: VARIABLE_NAME

Data Types and Variables

Declarations:

```
! A fortran95 program for G95
! CSE 425, Md.Shahriar Karim
! Basic example 1
program main
  implicit none
  real:: a, b, total, pi, area, radius
```

```
TYPE :: VARIABLE NAME
TYPE VARIABLE NAME
```

- Turns off the implicit type definition and default naming convention initial Fortran version allowed
  - Variable names starting with i, j, k, l, m, or n, are integers.
  - Variables starting with other alphabets are real variables.

Data Types:
- Character
- Real
- Integer
- Complex
- Logical

```
character (len = 20) :: test
test = 'Hello World'
```

```
integer :: number1
Integer :: number2
```

Constant:
- Definition of constant just require a 'parameter' right after the type of the variable.

```
integer, parameter :: constant1
real, parameter :: pi = 3.1428
```

Conventions: variables and operators

- Fortran77: Variable name allowed 1-6 characters length chosen from letters a-to-z and digits 1-to-10

- Fortran90: Variable name allowed 1-31 characters length chosen from letters a-to-z and digits 1-to-10. It also allows (_) underscore in variable names.
  - abc valid variable name; ABC denotes the same. So, it is not case sensitive
  - 123 invalid variable name; also, 123abc is invalid. That is, variable name must start with a character.

- Operators

Operator	**Precedence**	**Meaning**
**	1	Raise to the power of
*	2	Multiplication
/	2	Division
+	3	Addition or unary plus
−	3	Subtraction or unary minus

You can change the precedence by using brackets; sub-expressions within brackets are evaluated first.

Image source: Dept. of Physics, University of Cambridge

Program skeleton

- Start with `program main`, and it is enclosed within `end program main`. Within the enclosure, codes are `indented(recommended)`.
- Data type needs to be defined: `real :: a, b`
- Not case sensitive: does not create error if `total` is used later as `Total`

*main.f95

```fortran
! A fortran95 program for G95
! CSE 425, Md.Shahriar Karim
! Basic example 1
program main
  implicit none
  real:: a, b, total, pi, area, radius
  pi = 3.414
  a = 9
  b = 7
  total = a + b
  print *, 'The total is ', total    ! printing output
  write(*,*) 'The total is ', total  ! printing output

  ! calculate the area of the circle
  radius = 4;
  area = pi*radius**2    ! ** symbol for square of the quantity
  print *
  print *, 'The area is ', area    ! printing output

  write(*,*) 'Radius was ', radius  ! printing output

end program main   ! program main is ended here
!end  simple end would also be correct
```

Fortran Program: Input/output

- read: Used to take input from the external source.
- write: Used to outputting information from the program.

The form of the I/O statements is as follows:

```
read(stream, label [, end=end][, err=err]) list
```

and

```
write(stream, label) list
```

- *stream*: number linked to previously defined file, a character; or, * can be used to indicate default value (screen of a terminal session). If stream is a character variable, write command stores the value in that variable.

- *label*: It is the line number where the format statement. However, it can be replaced using * to use free-format.

- *list*: list of items (separated by commas) to be transferred to the output window; it can also include quoted text strings

Logical controls: if else and nested if else

- Syntax

```
if (Condition) then
statement1
else if (Condition) then
Statement2
else
statement3
end if
```

```
if (Condition) then
Statement1
else
Statement2
end if
```

.lt.	or	<	less than
.le.	or	<=	less than or equal
.eq.	or	==	equal
.ge.	or	>=	greater than or equal
.gt.	or	>	greater than
.ne.	or	/=	not equal
.not.			not
.and.			and
.or.			inclusive or

Image source: Dept. of Physics, University of Cambridge

main.f95

```
! A fortran95 program for G95
! CSE 425, Basic Example 3
program lettergrade
  implicit none
  real :: score
  print *, 'Enter CSE 425 score'
  read *, score
  if (score>=93) then
  write(*,*) "Letter grade is A"
  else if (score>=90 .and. score <93) then
    write(*,*) "Letter grade is A-"
  else if (score>=87 .and. score <90) then
    write(*,*) "Letter grade is B+"
  else if (score < 87) then
        write(*,*) "Letter grade is B"
  end if
end
```

Repetition: do loop

- Syntax

```
do variable = start_range, stop_range [,step]
statements
end do
```

- The `exit` command can terminate the loop

Factorial calculation

main.f95

```fortran
! A fortran95 program for G95 compiler
! CSE 425, Basic example 2
program factorial
   implicit none
   integer:: factorial_n, n, i
   print *,'Enter the number:'     ! user input
   read *, n      ! assigning user input to n
   factorial_n = 1
   do i = 1, n
   if (i>4) exit   ! calculate upto factorial 4
   factorial_n = factorial_n*i
   end do  !  repetition ends
   write(*,*) n, factorial_n
end program factorial    ! program main is ended here
!end  simple end would also be correct
```

Inner-Outer: Logical Controls

- In large program, logical control could be titled as `outer` and `inner` to improve the readability of the program
- Example: Finding the `square root` of a `real number` and dividing it further by another `real number`. Also, the `divided by zero` problem has been discarded.

main.f95

```fortran
! A fortran95 program for G95
! CSE 425, Basic Example 4
program sqrt_calculation
  implicit none
  real :: desired_no, divisor_no, x
  write(*,*) 'Find the square root of a number/divisor'
  print *
  write(*,*) 'Enter the divisor and target number'
  read *, divisor_no, desired_no
  outer: if (divisor_no /= 0) then
    inner: if (desired_no < 0) then
        write(*,*), 'Invalid input'
    else inner
        x = sqrt(desired_no)/divisor_no
        write(*,*) 'Divisor number is', divisor_no
        write(*,*) 'Other number is', desired_no
        write(*,*) 'sqrt(desired_no)/divisor_no =', x
    end if inner

    else outer

    write(*,*) 'Divided by zero issue'

    end if outer
end program sqrt_calculation
```

Intrinsic functions for computation

- FORTRAN provided a list of intrinsic functions that are frequently used for scientific computation.

Name	Action
`ABS(A)`	absolute value of any `A`
`ACOS(X)`	inverse cosine in the range $(0,\pi)$ in radians
`AIMAG(Z)`	imaginary part of `Z`
`AINT(X [,KIND])`	truncates fractional part towards zero, returning `real`
`ANINT(X [,KIND])`	nearest integer, returning `real`
`ASIN(X)`	inverse sine in the range $(-\pi/2,\pi/2)$ in radians
`ATAN(X)`	inverse tangent in the range $(-\pi/2,\pi/2)$ in radians
`ATAN2(Y,X)`	inverse tangent of `Y/X` in the range $(-\pi,\pi)$ in radians
`CMPLX(X [,Y][,KIND]`	converts to `complex` `X`+i`Y`; if `Y` is absent, 0 is used
`CONJG(Z)`	complex conjugate of `Z`
`COS(W)`	cosine of argument in radians
`COSH(X)`	hyperbolic cosine
`EXP(W)`	exponential function
`FLOOR(X)`	greatest `integer` less than `X`

Image source: Dept. of Physics, University of Cambridge

A few more examples

*main.f95

```fortran
! A fortran95 program for G95
! Complex Number
program complex_number
implicit none
! Define variables and constants
complex, parameter :: i = (0, 1) ! sqrt(-1)
complex :: num1, num2
num1 = (3, 4); num2 = (3, -4)
write(*,*) i * num1 * num2
end program complex_number
```

Array in Fortran

- Array of variables is set up in the declaration set up; array indexing depends on how the declaration is done
- Syntax:

```
real :: array_1(3)   ! Array of 3 values
real :: array_2(3, 3) ! Rank is 2 and it is a matrix
```

- Example

main.f95

```
! A fortran95 program for G95
! CSE 425, Basic Example 5
program main
  implicit none
  real array_1(3), dummy_1, magnitude ! array of size 3
  integer array_size, i
  write(*,*) 'Enter A_x, A_y, A_z'
  read *, array_1(1), array_1(2), array_1(3) ! indexing at 1
  array_size = size(array_1)
  dummy_1 = 0
  do i = 1, array_size
    dummy_1 = dummy_1 + array_1(i)*array_1(i)
  end do
  magnitude = sqrt(dummy_1)
  write(*,*) 'Array:', array_1(1), array_1(2), array_1(3)
  write(*,*) 'magnitude is', magnitude
end
```

Varying Array in Fortran

- Array size can be left open, and it can be assigned as needed. Also, such array could be released free once the task is completed.

- Syntax:

```
real, dimension(:), allocatable :: array_1 !
We define an array of unknown size array_1
```

- The array size later could be fixed as follows:

```
s = 100 ! array size
allocate(array_1(s))
```

- After the task is finished, the array is deallocated as follows:

```
s = 100
deallocate(array_1)
```

Varying Array in Fortran

- Mathematical computations can be performed directly on the array itself.

```
real :: array_a(5), array_b(5), array_c(5)
integer :: I

do i = 1, 10
array_c(i) = array_a(i) + array_b(i)
end do
```

Instead

```
real :: array_a(5), array_b(5), array_c(5)
array_c = array_a + array_b
```

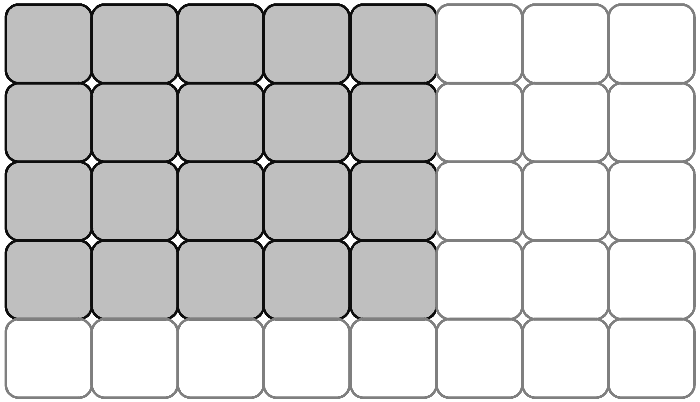
Conditional `where: in array`

- Syntax

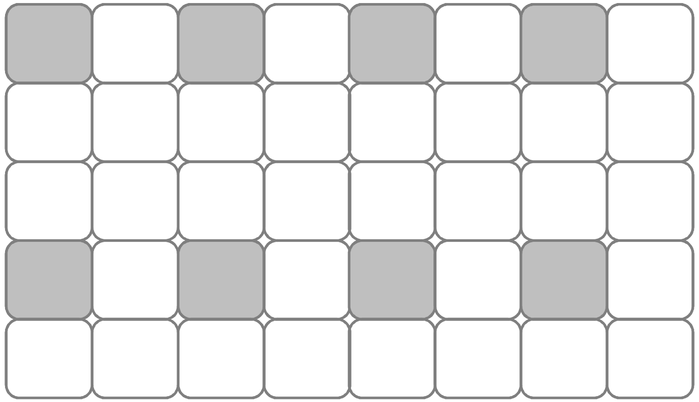
```
where (logical array statements)
        statements
elsewhere
        alternative statements
end where
```

Array structure

`real, dimension(1:4,1:5) :: array_1`



`real, dimension(1:6:3,1:7:2) :: array_1`



Dynamic Array

main.f95

```fortran
! A fortran95 program for G95
! Basic Example: Dynamic Array
program dynamic_array_example
  implicit none
  real, dimension(:,:), allocatable ::dynamic_1
  integer :: row_dim, column_dim
  integer :: i, j

  print *, 'Enter row_dim and column_dim'
  read *, row_dim, column_dim

  allocate(dynamic_1(row_dim,column_dim))
  do i = 1, column_dim
    do j = 1, row_dim
        dynamic_1(j,i) = i*j
        print *, "dynamic_1(",j,",",i,") = ", dynamic_1(j,i)
    end do
  end do
  !deallocate(dynamic_1)
end program
```

Mandatory syntax: dynamic_1 becomes dynamic

Allocating necessary memory location as per the need

Multiplication of Vector v and Identity Matrix I

```
integer name
ii=name(x,y,z)
stop
end

function name(x1,y1,z1)
integer name
name=int(x1+y1+z1)
return
end
```

main.f95

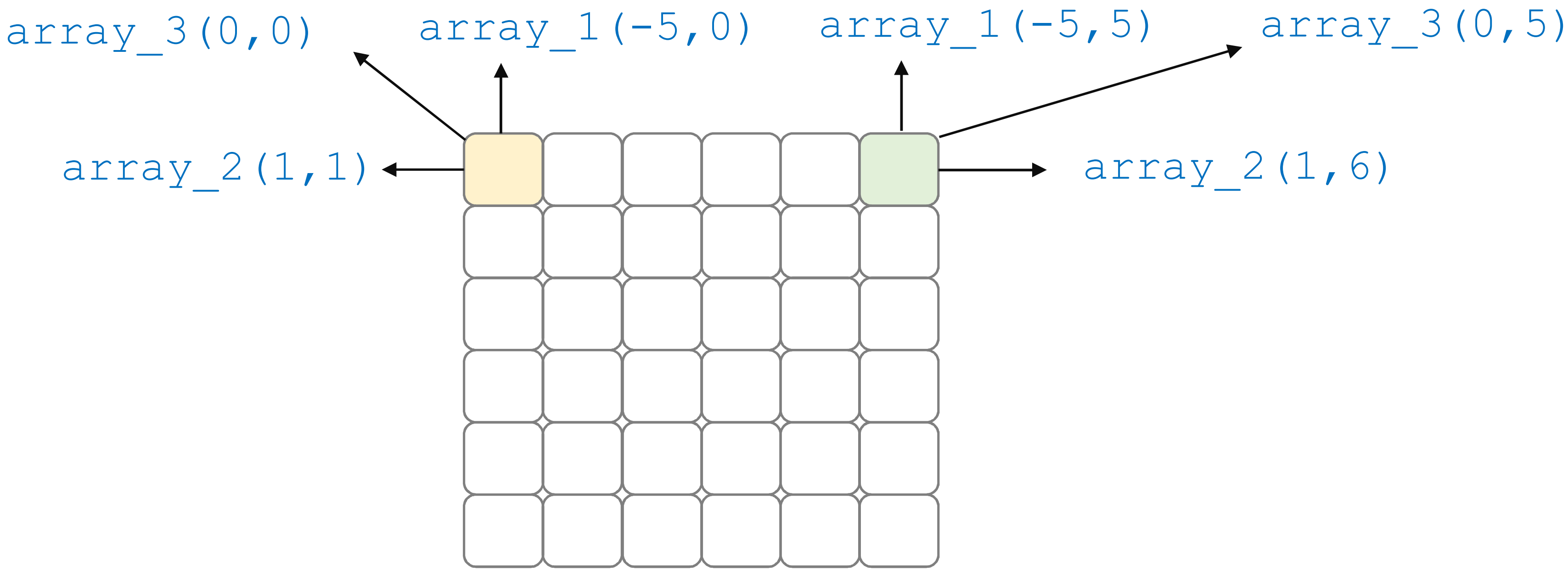
```
! A fortran95 program for G95
! Basic Array Example
program main
  implicit none
  real :: vec_1(3), vec_2(3), mtx_I(3,3)
  integer :: i, j
  vec_1(1) = 5
  vec_1(2) = 5
  vec_1(3) = 5
  ! matrix initialization I
  do i = 1, 3
    do j = 1, 3
        if(i>j .or. i<j) then
        mtx_I(i,j) = 0
        else
        mtx_I(i,j) = 1
        end if
    end do
  end do
  ! matrix-vector product
  do i = 1, 3
    vec_2(i) = 0
    do j = 1,3
        vec_2(i) = vec_2(i) + mtx_I(i,j)*vec_1(j)
    end do
  end do

  write(*,*) 'vec_1 =',vec_1
  write(*,*) 'matrix vector product: AI', vec_2
  write(*,*) ' matrix is: ', mtx_I
end
```

Conditional `where: in array`

- Negative indexing is allowed. Also, user can define the starting and ending indexes as needed.

```
real, dimension(-5:0,0:5) :: array_1
real, dimension(6,6) :: array_2
real, dimension(0:5,0:5) :: array_3
```



Functions

- Functions are generally placed after the `end` part of the main program.
- Function definition starts with the declaration of the `type of value` the defined function is expected to return. However, `type of the value` could be defined within the function definition.
- It also include the `function name`, and the `argument list` it takes as the inputs.
- All the `variables that are used` in the function, `including the arguments` of the function, must have `type declaration` in the function immediately after `the first line` of the function.
- The `function name` must be used `in an assignment statement` within the defined function.
- The defined function must finish with `end` statements

Functions: External Definition

- Calculation of the average of three input numbers using external function definition

main.f95

```fortran
! A fortran95 program for G95
! Basic Example: Function Definition
program main
  implicit none
  real avg, a, b, c, cal_avg
  write(*,*) 'Enter three numbers'
  read *, a, b, c

  avg = cal_avg(a, b, c)
  write(*,*), 'The three numbers', a, b, c
  write(*,*), 'Average is ', avg

end program main

  real function cal_avg(x, y, z)
    real x, y, z, sum
    sum = (x + y + z)
    cal_avg = sum/3
    return !obsolete in f90 & f95
  end function cal_avg
```

External function definition

Example: Function Definition

main.f95

```
! A fortran95 program for G95
! Basic Example: Function Definition
program main
  implicit none
  real avg, a, b, c, cal_avg
  write(*,*) 'Enter three numbers'
  read *, a, b, c

  avg = cal_avg(a, b, c)
  write(*,*), 'The three numbers', a, b, c
  write(*,*), 'Average is ', avg

end program main

  real function cal_avg(x, y, z)
    real x, y, z, sum
    sum = (x + y + z)
    cal_avg = sum/3
    return !obsolete in f90 & f95
  end function cal_avg
```

- Calling function
- Arguments
- Called function type definition
- Called function definition
- Type declaration of the arguments, and variables
- return is necessary in earlier Fortran, but now is obsolete
- Function name cal_avg is used in an assignment statement
- end is mandatory to complete the function definition

Functions: Internal Definition

- Calculation of cube-root of any given number using internal function definition

*main.f95

```
! A fortran95 program for G95
! For CSE 425, Fortran Basic Tutorial
! Function definition: Internal, using the CONTAINS
program main
  implicit none
  real x, r
  write(*,*) 'Enter your desired cube-root finding:'
  read*, x
  r = cube_root(x)
  write(*, *) 'Cube-root of', x, 'is', r

  contains

  real function cube_root(x)
  implicit none
  real x
  intent(in) x
  cube_root = exp(log(x)/3.0)
  end function cube_root

end program main
```

Internal function definition

Without Intent (in)

- Calculation of cube-root of any given number using internal function definition

main.f95

```fortran
! A fortran95 program for G95
! For CSE 425, Fortran Basic Tutorial
! Function definition: Internal, using the CONTAINS c
program main
  implicit none
  real x, r
  write(*,*) 'Enter your desired cube-root finding:'
  read*, x
  r = cube_root(x)
  write(*, *) 'Cube-root of', x, 'is', r

  contains

  real function cube_root(x)
  implicit none
  real x
  !intent(in) x
  x = (log(x)/3.0)
  cube_root = exp(x)
  end function cube_root

end program main
```

Intent (in) commented out

What is the value of X printed after cube-root calculation?

Subroutines

- Subroutines are similar to external functions defined in Fortran, but with an exception that they do not return value

- Instead, subroutines can modify the arguments used to call it by the program

- Swapping numbers

```
function name(arg1, arg2 …)
[declarations, including those
for the arguments]
[executable statements]
end function name
```

```
subroutine name(arg1, arg2,)
[declarations,]
[executable statements]
end subroutine name
```

Subroutines: Example

- Swapping numbers

main.f95

```
! A fortran95 program for G95
! Subroutine example
program swap_number_main
implicit none
real :: num1, num2
print *, 'Enter two numbers:'
! Read in two values
read(*,*) num1, num2
call swap_num(num1,num2)
write(*,*) num1,num2
contains   ! syntax to include subroutine
subroutine swap_num(x_first, y_second)
real :: x_first, y_second, temp
temp = x_first
x_first = y_second
y_second = temp
end subroutine swap_num
end program swap_number_main
```

Fortran code: Newton's Method

- Root finding method

Example: $f(x) = x^3 + x - 3 \qquad f'(x) = 3x^2 + 1$

Newton's Method:

$$f(x) = 0 \Rightarrow x = root \qquad x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

where $f'(x_i)$ is the first derivative calculated at x_i

t here | main.f95

```fortran
! A fortran95 program for G95
! Newton's Method:  Solve f(x) = 0
! f(x) = x^3 + x - 3, df(x) = 3*x^2 + 1
program rootfinding_newton
  implicit none  ! define all variables
  real :: x = 1
  real :: error_tol = 0.000001
  integer  i ! no colon means we cannot assign value
  integer  maxit
  integer  converg

  i = 0   ! assigning value
  maxit = 30
  converg = 0
  do while (converg == 0 .and. i < maxit)
    x = x - f_x(x)/f_deriv_x(x)
    write(*,*) x, f_x(x)
    i = i+1
    if(abs(f_x(x))<=error_tol) converg = 1
  end do

  if (converg == 1) then
    write (*,*) 'The method converged'
    else
    write(*,*) 'The method did not converge'
  end if
```

Conditional do loop

contains: ends the main program; functions and subroutines are defined right after this.

```fortran
contains !end of main, starts functions & subroutines
  function f_x(x)
    real f_x, x
    f_x = x**3 + x - 3
  end function f_x

  function f_deriv_x(x)
    real f_deriv_x, x
    f_deriv_x = 3*x**2 + 1
  end function f_deriv_x
end program rootfinding_newton
```

References

[1]. http://www.chem.ox.ac.uk/fortran/

[2]. Programming in Fortran 95, Computational Physics, University of Cambridge

[3]. https://web.stanford.edu/class/me200c

[4]. Computing with Fortran, Institute of Energy Technology, ETH, Zürich