We cover the chapters of your Fortran text listed in the syllabus two at a time: 7&8, 9&10, 11&12, 13&14, 15&16, 17&18, 19&22, 23. Reading assignment: chapters 8, 9, 10, 11, 16, 22.

**Vectors and Arrays**

\[
\text{real::v(4) means } v = [v_1, v_2, v_3, v_4]
\]

\[
\text{real::v(1:4) same thing } v = [v_1, v_2, v_3, v_4]
\]

\[
\text{real::v(0:3) means } v = [v_0, v_1, v_2, v_3]
\]

\[
\text{real::a(2,3) means } a = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}
\]

\[
\text{real::b(0:1,0:2) means } b = \begin{bmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{bmatrix}
\]

\[
\text{size(v) = the length of } v \quad \text{size([v_0, v_1, v_2]) = 3}
\]

\[
\text{lbound(v) = lowest index } \quad \text{lbound([v_0, v_1, v_2]) = 0}
\]

\[
\text{ubound(v) = largest index } \quad \text{ubound([v_0, v_1, v_2]) = 2}
\]

The following are equivalent:

```
print *, a(i,1),a(i,2),a(i,3)
print *, a(i,1:3)
print *,(a(i,j),j=1,3)
```
For compilers other than g95 or gfortran, i.e., online, FTN95, vectors are assigned as

\[ \mathbf{v} = (/3, 2, 1/) \] instead of

\[ \mathbf{v} = [3, 2, 1] \quad \text{but not} \quad \mathbf{v} = [3 2 1] \]

Commas are required in the edit window ([2,1] not [2 1]); spaces allowed in console window inputs.
Recall array convention:

$n$ is the number of rows, $m$ is the number of columns,
i ranges over row indices, $j$ ranges over column indices.

**Matrix functions**

The intrinsic array functions are

- $a+b, a*b =$ componentwise addition and multiplication.
- $\maxval(a)$ = the maximum value.
- $\minval(a)$ = the minimum value.
- $\maxloc(a)$ = the index of maximum value.
- $\minloc(a)$ = the index of minimum value.
- $\sum(a)$ = the sum of the elements.
- $\text{product}(a)$ = the product of the elements.
- $\text{matmul}(a,b)$ = the matrix product.
- $\text{dot_product}(a,b)$ = the dot product of vectors $a, b$.
- $\text{transpose}(a)$ = the transpose.
- $\text{size}(a,1)$ = number of rows (1st dimension)
- $\text{size}(a,2)$ = number of columns (2nd dimension)

In Fortran, $a*b$ is componentwise multiplication not ".*", $\text{matmul}(a,b)$ is matrix multiplication not "*".
**Example vectors.f95**

```fortran
program vectors1
integer :: v(3) = (/ 7, 8, 9 /)  ! initializes vector
integer :: w(0:4)
integer :: a(3,3) = 0  ! initializes everything to 0
print *, 'v=', v
print *(a,5i2,a),'w=',w
w(0) = -1; w(1) = -1; w(2:4) = v(1:3)
print '(a,5i2,a)','w=[',w ,']'
a(2,:) = [3, 2, 1];  ! online change to (/ 3, 2, 1 /)
a(3,:) = v
print '(3i3)', (a(i,:),i=1,3)
endprogram
```

Code vectors need commas, console input vectors do not. Delete the previous program. Copy/paste this, compile <ctrl-F7>, go <F5>. **Warning, don’t backspace** on console inputs. If you do, press <F5> to start again.

```fortran
program vectors2
integer :: u(3)
print *, 'Enter 3 comma-separated numbers'
read *, u; print *, u
print *, 'Enter 3 space-separated numbers'
read *, u; print *, u
endprogram
```
**CLASSWORK 14.1(2) printer.f95** Write an external subroutine which prints real matrices with 2-place decimals in 7-place fields.

!c14_1_2printer.f95

subroutine printer(b,n,m)
real::b(n,m) !n = # rows, m = # columns
!delete this, write the subroutine
endsubroutine !print decimals with two places

program printer_test
real::a(2,2)
a(1,:)=(/10.25,200./)
a(2,:)=(/3.,4.5/)
!delete this, call printer to print the matrix a
endprogram
Write an internal subroutine `row_sum(a, b)` which given an array `a` of reals, extends and fills a larger array `b` which has an additional column of row totals on the right. If \( n = 2, \ m = 3, \) and

\[
\begin{array}{ccc}
1 & 1 & 1 \\
2 & 2 & 2 \\
\end{array}
\]

then `b` =

\[
\begin{array}{cccc}
1 & 1 & 1 & 3 \\
2 & 2 & 2 & 6 \\
\end{array}
\]

Write a program which declares `a` and `b` and

1. initializes the array `a` to the \( 2 \times 3 \) matrix above,
2. calls the subroutine `row_sum` which fills `b`,
3. calls above external `printer` to print `b`.

```fortran
!c14_2_4row_sum.f95
!delete this, copy the printer subroutine here
program row_sum_test
real::a(2,3),b(2,4)
a(1,:)=(/1.,1.,1./); a(2,:)=(/2.,2.,2./)
call row_sum(a,b)
call printer(b,n,m+1)
contains
!delete this, write the subroutine row_sum
endprogram
```
ALLOCATION

If a real vector has length 3 and a matrix has size 3x3, declare them with
real::v(3), a(3,3)

If the sizes are not known at the beginning of a program, declare them to be allocatable with
real, allocatable:: v(:,), a(:, :) 

When the dimensions are known, allocate space with say
allocate(v(3), a(3,3))

!allocate.f95
program allocate
integer, allocatable:: v(:)
integer :: i, n

print *, 'How many numbers are there?'
read *, n
allocate(v(n))

print *, 'Enter', n, 'numbers.'
read *, v

print *, (v(i), i=n, 1, -1) ! goes backwards, step-size -1
endprogram
In a do-loop, \texttt{do i=n,1,-1} means start at \texttt{n}, stop at \texttt{1}, step-size = \texttt{-1}.

As an alternative to allocating space for a vector of unknown length, declare the vector to be of long enough (length=200) to handle any likely situation.

See \texttt{ENTERING LONG FORMULAS, COMMON ERRORS} at this lecture’s end.

For \( a = [a_1, a_2, \ldots, a_n] \), magnitude = \( \sqrt{a_1^2 + a_2^2 + \ldots + a_n^2} \)

\textbf{CLASSWORK 14.3(3) \texttt{magnitude.f95}} Write a real function \texttt{magnitude(v,n)} which given a vector \texttt{v} of \texttt{n} real numbers, returns its magnitude: \( \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2} \).

Write a program \texttt{test_magnitude} using this function.

It declares \texttt{v} to be allocatable.

It asks for and reads the number \texttt{n} of values.

It allocates space for vector \texttt{v}.

\texttt{allocate(v(n))}

It asks for and reads the \texttt{n} values of vector \texttt{v} on one line.

It prints the magnitude.
!c14_3_3magnitude.f95

real function magnitude(v,n)
integer::n; real::v(n)
magnitude = ______  !write the formula, use ** not ^
endfunction

program test_magnitude
integer::n;  real::magnitude
______________________________  !declare v allocatable
print *,'How many numbers are there?'
   !add assignment here, once running, comment out, <ctrl-q>
read *,n  !<ctrl-q> , use assignment, then uncomment <ctrl-q>
_________  !allocate space for v(n)
print *,'Enter',n,'numbers.';
read *,v  !comment out , replace with assignment, uncomment
print'(a,200(f5.2))','The vector ',v
print*,'has magnitude',magnitude(v,n)
endprogram

Classwork: 14.1(2)printer 14.2(4)row_sum 14.3(3)magnitude
Email: dale@math.hawaii.edu  Subject: 190  c14(9)
Add a couple of blank lines at the bottom of your email.
Rewrite the CLASSWORK 14.1(2) printer subroutine printer as an internal subroutine with the two lines

```fortran
subroutine printer(b,n,m)
real::b(n,m) !n = # rows, m = # columns
```

replaced with

```fortran
subroutine printer(b)
real::b(:,:); integer::n,m
```

Also appropriately change the call printer statement. Remember to include the contains statement.
Write an **external** subroutine `col_sum(a,b,n,m)` which given an array `a` of reals, extends and fills a larger array `b` which has an additional row of column totals at the bottom.

If \( n = 2 \), \( m = 3 \), and `a` =

\[
\begin{array}{ccc}
1 & 1 & 1 \\
2 & 2 & 2 \\
\end{array}
\]

then `b` =

\[
\begin{array}{ccc}
1 & 2 & 3 \\
1 & 2 & 3 \\
2 & 4 & 6 \\
\end{array}
\]

Write a program which declares `a` and `b` and

1. initializes the array `a` to the \( 2 \times 3 \) matrix above,
2. calls the subroutine `col_sum` which fills `b`,
3. calls the above internal `printer` to print `b`.

Suggestion, first modify the Classwork 14.2(4) `row_sum` problem to a `col_sum` problem with `col_sum(a,b)` internal as in the `row_sum` classwork problem problem. Then move the internal subroutine to an external subroutine
col_sum(a,b,n,m) with the dimensions n,m which an external routine needs and without the line
    n=size(a,1);m=size(a,2)
which is not valid for external routines.

!col_sum.f95
!delete this, write the subroutine col_sum
program col_sum_test
real::a(2,3),b(3,3)
a(1,:)=(/1.,2.,3./)
a(2,:)=(/1.,2.,3./)
call col_sum(a,b,2,3)
call printer(b)
contains
!delete this, copy the printer subroutine here
endprogram
Write a real function \texttt{distance}(v,w,n) which given vectors \( v, w \) of \( n \) real numbers, returns their distance:

\[
\sqrt{(v_1 - w_1)^2 + (v_2 - w_2)^2 + \cdots + (v_n - w_n)^2}.
\]

Write a program \texttt{test\_distance} using this function.

It declares \( v \) and \( w \) to be allocatable.

It asks for and reads the number \( n \) of values.

It allocates space for vectors \( v \) and \( w \).

\[
\text{allocate}(v(n),w(n))
\]

It asks for and reads the \( n \) values of vector \( v \) on one line.

It asks for and reads the \( n \) values of vector \( w \) in the next.

It prints “The distance = ” followed by the distance calculated by \texttt{distance}(v,w,n).

For \( n = 3 \). \( v = [1, 2, 3] \), \( w = [1, 1, 1] \) the printout is:

The distance = 2.236.

\textbf{Common errors}

Failure to declare the function in the program that uses it.

If the value of variable is garbage, say 3552748, quite likely it’s value was never set in the program.
Before writing code, do the steps by hand.
For hard problems, start with a simplified version and evolve it one step and test at a time. Do the steps by hand at first. Typically, do the printing part first (you can’t see what is happening without printing) and do the user input part last (testing takes longer when you have to supply input). If you are spending more than an hour on a program, maybe you should start over and do it one-step-and-test at a time.

**ENTERING LONG FORMULAS**

For complicated formulas, define them inside out to assure the parentheses are correct.

\[
\sqrt{\frac{1}{a_1^2} + \frac{1}{a_2^2} + \ldots + \frac{1}{a_n^2}}
\]

Start with the vector \( \mathbf{a} = [a_1, a_2, \ldots, a_n] \)
Square it: \( \mathbf{a}^{**2} \)
Take the reciprocal: \( 1/(\mathbf{a}^{**2}) \)
Sum it: \( \text{sum}(1/(\mathbf{a}^{**2})) \)
Take the square root: \( \text{sqrt}(\text{sum}(1/(\mathbf{a}^{**2}))) \)

Since we added a matching pair of ( )’s at each step, the number of left (’s will equal the number of right )’s.
**DEBUGGING (CORRECTING PROGRAMS)**

Don’t write more than 10 lines without testing.

First check that there are no grammar errors:

Once grammar mistakes are corrected, compilation gives **Exit code: 0**. If it runs but gives the wrong answer, you have a logical mistake, a *bug*. Here is a faulty function for determining primes along with a test program.

```fortran
logical function prime(n)
    integer::n,d
    do d=2,n-1
        print *, "A ", "d=", d, "n=", n, '. '
        if (mod(n,d)==0) then
            prime = .FALSE.
        !~print *, "B ", "d=", d, "n=", n, 'prime=', prime, '. '
        else
            prime = .TRUE.
        !~print *, "C ", "d=", d, "n=", n, 'prime=', prime, '. '
        endif
    enddo
endfunction
program test
logical::prime
```
To see what is happening inside your function `prime`, add debugging statements (green) to your program to print the values of your variables at each step of your program. The statements in green are the debugging statements. Once the program is corrected, delete them. <ctrl-q> uncomments.