Open Lecture 1. Open SciTE (black death star) not Scilab

We cover these chapters of your Fortran text:

7&8, 9&10, 11&12, 13&14, 15&16, 17&18, 19&22, 23.

Reading assignment: chapters 7, 8, 16, warnings, common errors, SciTE settings at the end of this lecture.

No partial credit for programs which don’t work.

Submit homework problems in separate emails. As before, don’t type anything in the email body. The entire body should be copy/pasted from one SciTE window, otherwise the submission is rejected. If your submitted program doesn’t work, I’ll make a few (usually three or four) corrections then return it to you for further correction and resubmission.

You lose 20% for every resubmission.

You lose 20% for each week overdue.

If you can’t install Fortran (most of you), you can use our website’s Fortran compiler or the computers in the undergraduate lounge (Keller 318). If you have a C or better, you can rent an old department laptop for $20/semester with $100 deposit.
Matlab, PHP, Javascript are interpreted languages, Fortran, Java, C are compiled languages. Running a program in an interpreted language involves two steps for each line. First the line is interpreted (translated into machine code). Then the machine code is executed (run/go). This process is repeated for each line. To run a program in a compiled language, all lines are translated into machine code in an initial compile step giving a new executable file (".exe" suffix in windows or ".app" on Macs). This code is then executed in a separate run/go step.

If a loop runs a million times, interpreted programs translate each line a million times, compiled programs translate them only once (the compile step). Compiled programs are much faster and all large computer programs are written in a compiled language. Fortran is similar to Matlab but, being compiled, is faster, more stable and has better error messages. The compiled .exe program runs on any windows computer, including those without Fortran. A Matlab program only runs on computers with Matlab installed.
In Fortran, all variables must be declared. This allows additional optimizations over Matlab’s general-purpose variables. When a variable is declared to be an integer, it is stored more compactly and handled in the faster arithmetic logic unit instead of the slower floating-point unit.

As before: imperative sentences are in green. They give instructions for classwork and homework; information on the other hand is carried in the black declarative sentences. When looking for information, skip the green.

Read **SciTE Settings**, and **Warnings** at the end of this chapter.
<table>
<thead>
<tr>
<th>Matlab</th>
<th>Fortran</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>if ... then</code></td>
<td><code>if (...) then</code></td>
<td><code>if (...) {</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>elseif (...)</code></td>
<td><code>elseif (...) then</code></td>
<td><code>elseif (...) {</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>else</code></td>
<td><code>else</code></td>
<td><code>else</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>end</code></td>
<td><code>endif</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>for i=1:10</code></td>
<td><code>do i=1,10</code></td>
<td><code>for (i=1;i&lt;=10;i++){</code></td>
</tr>
<tr>
<td><code>...</code></td>
<td><code>...</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td><code>end</code></td>
<td><code>enddo</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>while(0==0)</code></td>
<td><code>do</code></td>
<td><code>while(0==0){</code></td>
</tr>
<tr>
<td><code>return</code></td>
<td><code>return</code></td>
<td><code>return();</code></td>
</tr>
<tr>
<td><code>break</code></td>
<td><code>exit</code></td>
<td><code>break;</code></td>
</tr>
<tr>
<td><code>continue</code></td>
<td><code>cycle</code></td>
<td><code>continue;</code></td>
</tr>
<tr>
<td><code>end</code></td>
<td><code>enddo</code></td>
<td><code>}</code></td>
</tr>
<tr>
<td><code>// comment</code></td>
<td><code>! comment</code></td>
<td><code>/* comment */</code></td>
</tr>
<tr>
<td><code>test lines</code></td>
<td><code>program (required)</code></td>
<td><code>main</code></td>
</tr>
</tbody>
</table>

*then must be on the same line as the if or elseif.
elseif, endif, endprogram can be written else if, end if, ...

Indent (2 or 3 spaces or a tab) statements between flow-control commands.

Using tabs gives errors with the class website’s online compiler, giving “nonconforming tab character at ... ”.
If not already open, double-click the SciTE death star. Copy lines, File/New in SciTE, paste, File/Save as c11_1_2num_divisors.f95 Translate the red Matlab statements into Fortran.

!c11_1_2num_divisors.f95

integer function num_divisors(n) result(y)

// function y=num_divisors(n) start comments with !

integer::n,d

y=0

for d=1:n !change for to do. if statements need ( )s.
    if(modulo(n,d)==0) then !change modulo to mod
        y=y+1
    end
end

end function !test lines must be in a program

program test_num_divisors

integer::n,num_divisors

for n=3:7
    print *,n,'has',num_divisors(n),'divisors'
end

endprogram !<ctrl-F7><F5>, not <ctrl-l> don't press <F12>
Press `<ctrl-F7>` to compile or select Tools/Compile. The *console* window opens on the *right*. to put it at the bottom, select Options uncheck Vertical Split. Exit code: 0 means 0 errors (except for online compiler). Press `<F5>` to run the program or select Tools/Go. Compiling automatically saves.
Classwork 11.2(2) is\_perfect\_square  Correct red errors.

!c11_2_2is\_perfect\_square.f95

logical function is\_perfect\_square(n)
integer :: n,i

is\_perfect\_square = .false.
for i=0:n !not for
  if n==i**2 !need parentheses ( ... ), need then red
    is\_perfect\_square = .true.
    return
  end
end
endfunction

program is\_perfect\_square  !bad program name
logical :: is\_perfect\_square
integer :: n,i  !declarations come first
print *,'Enter an integer.'
read *,n
for i=0:n
  if (is\_perfect\_square(i)) then
    print *,i
  end
end
endprogram  !switch to console, enter 20, get 0, 1, 4, 9, 16.
<ctrl-F7> compiles (or Tools/Compile).
<F5> runs the program (or Tools/Go).
<ctrl-F6> (or click in console) to switch to console window.

Enter a number, say 20. Should get 0, 1, 4, 9, 16.

**CLASSWORK 11.3(4) next_perfect_square**  Write a function giving the next perfect square after \( n \). 1st do by hand. Copy lines, File/New in SciTE edit window, paste, File/Save as ...

```
!c11_3_4next_perfect_square.f95
   !copy is_perfect_square here  the function not program
integer function next_perfect_square(n)result(p)
ingteger::n    ! Not integer::n,p
logical::is_perfect_square
   !delete this line, complete the function, use an infinite do-loop
dfunction
endfunction

program test_next_square
integer :: next_perfect_square
do i=2,12,3    ! Step size = 3
   print '(i2,a,i2)',i,' -> ', next_perfect_square(i)
enddo
endprogram
```
For FTN95 compiler (maybe try gfortran instead), replace print ' (i2,a,i2)',i,' -> ', next_perfect_square(i)

with

10 format(i2,a,i2)
print 10,i,' -> ', next_perfect_square(i)

**CLASSWORK DUE END OF PERIOD**

11.1(2) num_divisors 11.2(2) is_perfect_square 11.3(4) next_perfect_square

email: dale@math.hawaii.edu  subject line: 190 c11(8)

Send all in one email. Put a couple of blank lines after each problem. Don’t add comments without “!”.

**SEND EACH HW PROBLEM IN A SEPARATE EMAIL**

If using the online compiler, don’t use tabs.
HOMEWORK 11.1(2) myfactorial

Email to: dale@math.hawaii.edu  Send problems separately.
Send with subject line: 190 h11.1(2)

//my_factorial.f95  Convert to Fortran. Correct reds.
integer function my_factorial(n)
! need result(k) in line above
integer::i,n
k=1
for i=1:n
   k=k*i
end
endfunction
endfunction

program my_factorial  !rename to test_my_factorial
for i=1:9
   print *, " n=",i,"n!=",my_factorial(i)
end
endprogram  !ans: 1,2,6,24,120,720,...
HOMEWORK 11.2(4) is_power_3

Email to: dale@math.hawaii.edu Send homework separately.
Send with subject line: 190 h11.2(4)

The powers of 3 are \(3^0, 3^1, 3^2, 3^3, \ldots = 1, 3, 9, 27, \ldots\)

Write a function is_power_3(n) which is .true. if n is a power of three, .false. if not. You may use \(3^{**i}\) but not Matlab’s \(3^i\)

After the function write a program print_powers_3 which asks for an input n and then prints powers of 3 which are \(\leq\) to n. Modify the classwork is_perfect_square function using \(3^{**i}\) instead of \(i^{**2}\). Also modify its program.

!is_power_3.f95
Write a function \texttt{next\_power\_3}(n) which returns the first power of 3 after \(n\). Include the previous function \texttt{is\_power\_3} but \textbf{not} its test program. Modify classwork \texttt{next\_perfect\_square}.

\begin{verbatim}
!next_power_3.f95
  !copy the \texttt{is\_power\_3} function here, not its program
  !copy \texttt{next\_perfect\_square} function here, modify to \texttt{next\_power\_3}
program test_next_power_3
  integer :: next_power_3
  do i=2,12,3  ! Step size = 3
    print '(i2,a,i2)',i,' -> ', next_power_3(i)
  enddo
endprogram
\end{verbatim}

\textbf{Send each HW problem in a separate email}

11.1(2) \texttt{myfactorial} 11.2(4) \texttt{is\_power\_3} 11.3(4) \texttt{next\_power\_3}
**Names** Names (of variables, programs, functions, subroutines, files) such as `x11`, `is_prime`, must start with a letter and may be followed by letters (a-z), digits (0-9) or the underscore “_”. In names, Fortran does not distinguish case; `a` is equivalent to `A`. A program can’t have the same name as a function. If the function is `divisors`, the program could be `test_divisors`. The file name `divisors.f95` can be the same. Unlike Matlab, Fortran file names can’t use “(”, “)”, or “.”. After the file name add the extension `.f95`. Not allowed: `c1.2(3)print.f95` Is allowed: `c1_2_3print.f95` The compiled program `c1_2_3print.exe` runs on any PC.

**Assignments and logical expressions**

- `x=y` sets `x` equal to `y`, this is an assignment.
- `x==y` tests if `x` equals `y`.
- `x<y`, `x<=y`, `x>y`, `x>=y`, `x/=y`, `x/=y`.
- For `<`, `<=`, `>`, `>=`, `!=`. 
**Integer arithmetic**

3+2; 3*2; 3**2; !not 3^2
max(3,5,1,4); min(3,5,1,4)
mod(13,4) !remainder of 13 divided by 4 mod not modulo
13/4 !for integers, this is the quotient, not division. 13/4 = 3
If you want division instead of the quotient, write 13./4.
Wrong: sqrt(4). Right: sqrt(4.)

**Real arithmetic**  Reals must have "."; integers must not.
3.*2.; 3./2.; sqrt(4.); abs(-3.);
max(3.,5.,1.,4.); min(3.,5.,1.,4.)
rand() !random number from [0,1)
abs(x), sqrt(x), exp(x), log(x), log10(x)
= |x| \(\sqrt{x}\) \(e^x\) \(\ln(x)\) \(\log_{10}(x)\)
sin(x), cos(x), tan(x), asin(x), acos(x), atan(x),
\(\sin^{-1}(x)\), \(\cos^{-1}(x)\), \(\tan^{-1}(x)\)
FUNCTIONS

Where Matlab has

```matlab
function y=square(n)
    y=n**2
endfunction
```

Fortran has

```fortran
integer function square(n) result(y)
    integer::n
    y=n**2
endfunction
```

Or you can use the function’s name is the output variable.

```fortran
integer function square(n)
    integer :: n
    square=n**2
end function
```
**Programs**

First line is **program** followed by the program name.

Last line is **endprogram** or **end program**.

```fortran
program print_num
    print*, 'Enter a number> ' !prints to screen,
    read *, x !reads from the keyboard
    print *, "Your entry", x !default format
    print ' ("Your entry", f7.2)', x !f=floating point
endprogram
```

**Print, Read**

```fortran
read *, n
reads a value from the keyboard into n. read can also read from files or from strings, but * indicates keyboard input.
    print *, n, 'squared is', n**2
    print '(i2,a,i3)', n, 'squared is', n**2
Both print data to the screen. * indicates default formatting. '(i2,a,i3)' formats
a 2-place integer (i2),
a string of minimum needed length (a),
then a 3-place integer (i3).
    print *, read * with spaces or print*, read* without.
```
Save to extension .f95, compile, go

Copy lines of print_num, File/New in a SciTE edit window, paste. Select Tools/Compile. What happens? Fortran programs don’t compile unless saved with extension “.f95”. The extension specifies the language (Fortran, C, Java, ...). The highlighting changes when saved as a .f95 file.

File/Save As print_num.f95
<ctrl-F7> compiles, <F5> runs the program.

The input request appears in the console window (right). Enter a number there, say 20 (type 20 then press <enter>). The program output appears on the console side.
Warning when entering input, backspace doesn’t work, try it.
**DECLARATIONS** Variables are locations in memory. You cannot use a variable until a location and space for the variable has been *allocated* by a declaration (explicit or implicit). Place all declarations at the beginning of a function or program.

```plaintext
real :: numbers with decimals, 3.5, -2., .088
real:: x, y  space is optional
integer ::m numbers without decimals, 3, -2
logical :: either .true. or .false.
character:: strings of letters, digits and other characters enclosed in single or double quotes: 'and', "x=", "It's a good day."
```

In programs (but not in functions), you may optionally assign an initial value, e.g., x=2.

```plaintext
real:: x=2., y=0.     ! reals must have a decimal point.
integer:: j=1, i=2   ! integers must not have decimal.
character(3) :: s="hat", t='cat'
```
**Implicit Declaration Convention** Unless declared otherwise, variable names starting with i, j, ..., n, are integers, all others are reals. Write *implicit none* immediately after the function/program line to prevent implicit declaration. Use this to find spelling errors.

**Long lines, multiple lines**

To put two lines on one line, separate them with a “;”.

\[
\begin{align*}
x &= x + 1; & y &= x ** 2 \\
\end{align*}
\]

When writing or copying/pasting programs, make the editor window is wide enough that lines don’t wrap.

If you have a long line which has to be continued on the next line, place a “&” at the end of the line to tell Fortran that the line continues on the next line.

\[
\begin{align*}
\text{print } *, & 'Here is a long line that needs \&}
& \text{to continue on the next line.}'
\end{align*}
\]
**Error Messages** When programs don’t compile, the console error message gives the error’s line number. If an error code reads:

```
In file x.f95:15.4
```

the error is in line 15, 4 spaces from the left. If you don’t see line numbers select *View/Line Numbers*. The “1” in the next line points to the position of the error.

Ignore all errors except the first one. After correcting the first one, recompile. Often the other errors will go away.

**Warnings and Common Errors** If SciTE becomes unresponsive, it may be in an infinite loop. [Click Tools/Stop Executing]

- Fortran files must have extensions `.f95`
- A Fortran file must have exactly one program.
- Functions and programs must have different names. If the function’s name is *square*, maybe name the program *test_square*.

If you suddenly lose color and formatting, you probably pressed `<f12>`. At the menu bar select *Language/Fortran*. 
If you get an error when entering user input on the right-hand console window, you may have pressed <backspace> to correct an error. This doesn’t work. For correct routing, subject lines must be exactly as stated. Don’t include anything which won’t compile. Put ! in front of comments.

Line wraps. Don’t let a line continue on to a second line. Widen the editor window so the line doesn’t wrap or shorten the line (delete comments) or use the continuation symbol & to continue the line on the next line.

Exponentiation is x**n, not x^n. The remainder is mod, not modulo. Begin comments with !, not //.

If compile is grayed out, check that you have saved the file with the extension .f95. Also check that your computer isn’t still trying to execute a previous program (click Tools/Stop Executing).

If you get a system cannot find the file specified error, check that the file name does not have characters such as “.”, “(” or “)”. The name 9_4_3.f95 works but 9.4(3).f95 won’t.
If the value of a variable is garbage, say 3552748, quite likely it’s value was never set in the program.

If you get an undefined error, make sure that any function which you are using in a program has been included and that it has been declared and that you haven’t misspelled the function name.

Each file must have exactly one program. If you get a Two main PROGRAM units error, your file has two programs. If you don’t have any program, you’ll get an undefined reference error.

If you get a make: *** No targets... error, you selected Tools/Build instead of Tools/Go or pressed <F7> instead of <F5>.

If you get a conversion error... message, you probably didn’t declare the variable. Use implicit none to prevent this.

For the online compiler and maybe the FTN95 compiler, your SciTE file must not have tabs. It will complain about nonconforming tab character at ... .
**SciTE settings**

Let’s open a previously saved program. Select File/Open .... If you don’t see your Fortran programs, in the bottom-right Files of type box, select Fortran in the drop-down menu.

When programs don’t compile, the console’s error message gives the error’s line number. If an error code reads:

```
In file x.f95:15
```

the error is in line 15. To see the line numbers select View/Line Numbers. SciTE will also indicate the position of the error in the line by printing a “1” below the error but you need a monospace font to see the position. To do this, open Options then others.properties. Find style.errorlist.32 and modify style.errorlist.32=$(font.small) to style.errorlist.32=$(font.monospace)

To permanently make Fortran files and line numbers visible, select Options/Open Global Options File. Find #line.margin.visible=1 and delete the “#”. Find source.files=*.asm;*.c;... and add *.f95;*.txt; to get
Find font.small and change font.small=... to font.small=font:Courier New,size:8
If the size is too small, try 9 instead of 8.

**Textbook**  As with Matlab, everything on the final will be from the lectures not the Fortran textbook. If you are in the upper half of the class, the lecture notes should suffice. If you are in the lower half of the class, there will be things you don’t understand even after studying a lecture. In this case you will need to study the reading assignment given for that lecture. Don’t just memorize steps, if you don’t understand how programs operate, you won’t be able to pass the final.