CLASSWORK 17.1(2)  print_title.f95  Change things in pink to print a nice title for n homework scores. Change print 11 to make the title print on one line. Change a1, i1 to get (items should be one space apart).  Result should be like this:

Hw1  Hw2  Hw3  Hw4  Hw5  Hw6  Hw7  Hw8  Grade
Hw1  Hw2  Hw3  Hw4  Hw5  Hw6  Hw7  Hw8  Hw9  Grade

!c17_1_2print_title.f95
subroutine  print_title(n)
integer:: n;
11  format(a1, i1)  !change this
do  i=1, n
    print 11, 'Hw', i
    !replace print 11, with  write(*,11,advance='no')
enddo
print '(a6)', 'Grade'  !change as above if using  FTN95
endsubroutine
program  test_print_title
do  i=1, 9
   call  print_title(i)
enddo
endprogram
Write a program `print_title2` which asks the user for the number `n` of exams and then prints, **on one line**, the heading **Exam 1, Exam 2, ..., Exam n, Final**

If the user enters 4, the printed heading should be **Exam 1, Exam 2, Exam 3, Exam 4, Final**

The heading **must include the commas**. Use a `write` statement with `advance='no'`. See previous program `print_title`.

With FTN95, format statements are written the old way, e.g., instead of `print '(a9)', 'Grade'` write `write 10 format(a9)`

print 10,'Grade'
Printing or writing `abcd` to a 10-space field gives "      abcd" -- it is *right-justified* when trimmed.

```plaintext
character(10)::s10='abcd'
character(2)::s2='abcd'
```

assign the 4-letter string `abcd` to 10-spaces and to 2-spaces.

Assigning or reading `abcd` to a 10-space field gives "abcd" -- it is *left-justified*.

Assigning or reading `abcd` to a 2-space field gives "ab" -- it is *truncated*, the tail is lost.

**Example** conversions.f95  New window, save, run.

```fortran
!conversions.f95
program character_conversions

character(10)::s10='abcd'  !6 spaces added on the right
character(2) ::s2 = 'abcd'  !'abcd' truncated to 'ab'
print *,'/',s10,"/ s10='abcd';print*,s10"
print *,'/',trim(s10),'/       print*,trim(s10)'  
print '(a,a10,a,a)'," /",trim(s10), &
     "/ print '(a10)',trim(s10)"
print *,'/',s2,"/       s2='abcd';print*,s2"
print*;  endprogram
```
What went wrong? Correct the two pinks. Add two spaces.

Want  Hello John Doe!
not   HelloJohn Doe!
not   HelloJohnDoe!

**COMMON ERROR** If you don’t see say the last !, it might be off screen due to untrimmed blank spaces.

**Note:**
read *,s   !reads one word from keyboard to s,
read '(a)',s !reads one line from keyboard to s.
program name2
character(20) :: fullname
print *,'Enter first name, space, last name'
read *, fullname
print *,'Aloha ',trim(fullname),''!
endprogram

Correct the pink line to read in all of the fullname.
Want  Aloha John Doe!
not   Aloha John
CHARACTER STRING FUNCTIONS AND OPERATIONS

character(5)::r,s,t

12345 ! positions in strings start with 1
s='abcde'

s(2:2)= 'b' ! s(2:2), not s(2)

s(2:3)= 'bc'

s(:3)= 'abc'

s(3::)= 'cde'

s//t is the concatenation of s and t = s followed by t.

s='abc'; t='xyz'

s//t = 'abcxyz' ? No, s//t in fact is
s//t = 'abc  xyz '.

To get 'abcxyz' write
trim(s)//trim(t)= 'abcxyz'

trim(s)//'--'//trim(t)= 'abc--xyz'

index('abcde','cd')= 3
index('abcde','cx')= 0

read(s,*) r, t ! reads strings r, t from string s
write(s,*) r, t ! writes strings r, t to string s
**Common error.** The concatenation of three strings $r, s, t$ is $r//s//t$, not $r, s, t$, not $r+s+t$. If words are too far apart, trim them, e.g., `trim(r)///trim(s)`, if too close, add a space, e.g., `trim(r)//' '/trim(s)`. To separate with a comma and a space, write `trim(r)//', ', '/trim(s)`. Remember this for the next classwork.
CLASSWORK 17.4(2) name3.f95  Write a program which reads a first name first, a last name last, then writes the full name = last name, comma, space, first name. to a strings full, full2.

E.g. full= Smith, Tom

!c17_4_2name3.f95
program name3
character(20)::first,last,full,full2
first='John'; last='Doe'
full = trim(last) // _______
!Want: Doe, John
print *, full
write(full2,'(3a)') trim(last), _______
print*,full2  !The line above is an alternate to concatenation.
endprogram
Write a program which, reads a full name `full` such as `full="Smith, John"` consisting of a last name, comma, space, first name, then prints the first name, a space, then the last name (with no commas).

Hint: use the substring functions, e.g., `s(:3), full(3:)`, and the index function `ind=index(full,'','').

Change the 4 pink lines.

```fortran
!c17_5_4name4.f95
program name4
character(20):: first, last, full
full='Doe, John'
!~print *,'Enter lastname, firstname with ','
!~read '(A)',full
i=index(________) ! the index of ',' in full.
first = ________ ! full(i)? full(i): ?
last = ________ ! full(i)? full(i): ?
print *,first,last !fix this:
!Want: John Doe no comma
endprogram
```
COMMON ERROR  Don’t set first=index(full, ',', '). first is a character string, i=index(...) is an integer. Set first = to some substring of full, maybe full(i:)? -- actually not.

HOMEWORK 17.2(3)  name5.f95
dale@math.hawaii.edu  subject line: 190 h17.2(3)
Write a program which reads a line from the user consisting of three words separated by spaces, e.g., the user might enter "one two three". Use
read *,first,second,third
to read the user’s line into three strings, first, second, third. It then writes (not prints) these three words to a line line consisting of the same three words but with the words separated by a comma and a space, e.g., line="one, two, three". Then print line.

!h17_2_3name4.f95
program name4
character(200)::first,second,third,line
  ... delete this line, finish the program
print*,trim(line)
endprogram
Review: If line is Good day.
read(line,*) s reads only the first word Good
read(line,'(a)') s reads the whole line Good day.
trim(line) trims off trailing blanks from line

Example read_string.f95

program read_two_words
character(20)::s1,s2
print*, 'Enter two words.'
read *,s1,s2 !reads one word at a time
print*,trim(s1),',',trim(s2)
endprogram
program read_two_lines
character(200)::t1,t2
print*,'Enter two lines of several words'
read '(a)', t1,t2 !reads whole lines at a time
print*,' Printed on one line'
print'(3a)',trim(t1),' ',trim(t2)
print*,' Printed on separate lines'
print'(a)',trim(t1),trim(t2) !prints whole lines
endprogram
Reading from and writing to files

A file is a sequence of lines stored on the hard drive. Each line is character string whose length is usually 200 or less. A file of \( n \) lines can be stored in a vector `lines` of \( n \) lines declared with `character(200)::lines(n)`.

Before reading or writing to a file, open it with an `open` statement which assigns it a file number.

```fortran
open(3, file='out.txt')
```

You read from and write to a file using its file number instead of the `*` used in keyboard reads and screen writes. Close the file after using.

```fortran
close(3)
```

File/New, copy/paste these lines, save as `in1.txt`.

```
1 1
1 2
1 3
1 4
```

File/New, copy/paste these lines, save as `in2.txt`.

```
2 1
2 2
2 3
2 4
2 5
```
program file
character(200):: line  !most file lines have < 200 letters.
open(1, file='in1.txt')  !open file in1.txt, out.txt,
open(3, file='out.txt')  !from SciTE default directory
read(1,'(a)') line       !read from in1.txt
write(*,'(a)') trim(line) !write to screen
write(3,'(a)') trim(line) !write to file out.txt
close(1); close(3)       !close both opened files
endprogram

line has 200 characters. It is not likely you want to see this many blank spaces. trim(line) deletes any trailing blanks. Write and print trim(line) instead line.

To write to a different folder, include the path.
open(1, file='C:\fortran\out.txt')

Open out.txt in SciTE; check that it has a line from the input file in1.txt. If you don’t see text files, select Files/Open... and in Files of type: replace all source with text from the drop-down menu.
The open statement creates a file if it doesn’t already exist. Writing to the file `out.txt` while it is in an open SciTE window can cause problems, close the `out.txt` SciTE window before running the program. (The online compiler doesn’t write to files, the Keller 318 computers do.)

**Change the example so that it:** Reads 4 lines from `in1.txt`; writes them to `out.txt`. (When writing to a file, also write to the screen in order to monitor the program’s progress.) After reading and writing, close all files.

**Change the example so that:** All reads from the input file are done first with the lines read into a vector `line(200)` of 200 lines and then all writes to the output file are done later.

Put both reads in a single **internal** subroutine `reader(filenum, numlines)`. Put the writes in an **internal** subroutine `writer(filenum, numlines)`. In addition, it is to read 5 lines from `in2.txt` and write them to `out.txt`. 
Put the reads and writes in external subroutines 
reader(filenum,line,n), writer(filenum,line,n).
Delete the contains statement. As with the classwork, the program is to read all lines from in1.txt and write them to file out.txt and then read all lines from in1.txt and write them to file out.txt.

**INPUT/OUTPUT ERROR STATUS - IOSTAT**
Check status of input/output errors with iostat.
- iostat =0 if a read is successful.
- iostat <0 if one is at the end of the file (eof).
- iostat >0 if the input is in the wrong format.
**Example iostat.f95**

```fortran
!iostat.f95
program input_output_status
integer:: i, stats; character(200):: frm
print*, 'Enter character, Enter real, Enter integer.'
12 format(i3)
do
  i = iostat(stats)
  print*, 'I/O sta = ', stats
  if(stats==0) exit
  print*, 'Not an integer, try again.'
enddo
print*, 'Yes, integer = ', i
endprogram
```

Enter these three lines in the console window:

```
abc !a character string instead of an integer
10. !a real number instead of an integer
10 !an integer as is wanted, stats = 0 = no errors.
```
To read all lines of a file with an unknown number of lines, use the `iostat` variable

```
read(1, '(a)', iostat=eof) line
```

to determine when a read was successful. If the read was unsuccessful, it is $<0$, and you are past the end of the file and the previous reads read all lines. The program below reads lines one-at-a-time from the file `in1.txt`. It uses the `iostat` variable `eof` to determine when the end of the file has been passed.
Example files2.f95

!file2.f95

program files2
  integer:: eof
  character(200):: line
  open(1,file='in1.txt')
  open(3,file='out.txt')
  do
    read(1,'(a)',iostat=eof) line
    if(eof<0)then; exit; endif
      !iostat is < 0 after the end of a file
    write(*,'(a)') trim(line)
    write(3,'(a)') trim(line)
  enddo
  close(1); close(3); endprogram
The following `num_lines` subroutine reads a previously opened file with file number `file_num`. It reads the lines into a vector `lines` of `numlines` lines. `eof` records the status of file reads. If `eof` is < 0, the end of the file has been reached. Before closing, it rewinds the file so that the next time the file is used, the first line of the file will be read not the last.
subroutine file_reader (filenum, maxlines, lines, numlines)

maxlines = maximum number of lines in file
numlines = actual number of lines in file
lines = vector of lines in file

integer:: filenum, eof
character(200):: lines(maxlines)
numlines=0

do i=1,300
   read(____, '(a)', iostat=eof) lines(i)
   if(eof<0) exit  ! iostat is < 0 after the end of a file
   numlines=____
enddo

rewind(filenum)  ! rewinds to the files first line

end subroutine

program test_file_reader

integer:: numlines
character(200):: lines(400)
open(1, file='in1.txt')
call file_reader(1,400, lines, numlines)
print'(a,i4)','Lines in in1.txt:', numlines
do i=1,numlines; print*, lines(i); enddo
end program
The following `line_reader` subroutine reads words from a string `line` rather than from a file. It reads the words into a vector `words` of `numwords` words. `eof` records the status of file reads and `eof` is $< 0$ if the read failed.
subroutine line_reader (line, words, numwords)
!
!counts the number of words in a string str
!reads the words into a vector of words.
!
character (200)::: line, words (200)
integer::: eof, numwords
!
do n=1, 200
!	read( _____ , *, iostat=eof) (words (i), i=1, n)	!	if (eof<0) exit ! iostat is < 0 if read fails
enddo
!	numwords= _____
end subroutine

program test_line_reader
!
character (200)::: words (200), line
integer::: eof, numwords
!
line="aa bb cc dd ee"
!
call line_reader (line, words, numwords)
!
print*, line
!
print'(a, i0, a)', "has these ", numwords, " words"
!
do i=1, numwords; print*, words (i); enddo
end program