Example

loop to count to 10

File input/output

- Reading values from the keyboard works provided there are only a few values that are "user friendly" to enter.
- It doesn't work too well if there are thousands (or more) of values to enter.
- How can data be read from a file rather than the keyboard?

General Rules for file input

- The name of the data file can be anything provided it doesn't conflict with an existing file.
- The "layout" of data in the file must match the layout of the read statements in the program.

Program

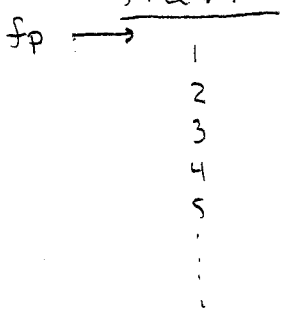
```
READ a,b,c
READ d,e
READ f,g,h
```

contents of file.in

```
1, 7, 9
4, 2
3, 7.4dØ, 9.3dØ
```

can separate by spaces or commas

- file pointer



- file pointer starts at the top of the file, just before line 1
- each new READ advances the pointer one line.

- make sure there are no blank lines in the file. This will lead to an "EOF", "End of file" error.
- Don't try to read past the end of the file. This will also lead to an EOF error (though there is a way around this that we will look at soon.
- You can have more data on a line than you are reading, but not vice versa

Program	file.in	
READ a,b,c	1,2,3,4,5	OK
READ a,b,c	1,2	EOF error

How to actually get data into the program?

- Easiest way is to use input/output redirection (doesn't require any changes to your program)
- Input redirection

```
a.exe < file.in
```

* this tells a.exe to get READ(*,*) statements from file.in instead of the keyboard.

- output redirection

```
a.exe > file.out
```

* tells a.exe to send WRITE(*,*) output to file.out instead of the screen
* file.out will be created if it doesn't exist
* if file.out does exist, its contents will be overwritten.

- output redirection with append

```
a.exe >> file.out
```

  * tells a.exe to send WRITE(*,*) output to file.out instead of screen.
  * if file.out exists, the new output will be appended to file.out

- These can be mixed

```
a.exe < file.in > file.out
a.exe < file.in >> file.out
```

OPEN/CLOSE statements

- These are an alternative to file i/o redirection
- Offer more flexibility (for example if your inputs are spread over several files or your outputs need to go to different files)
- Can have as many files open at one time as you wish (though there are practical limits from an efficiency viewpoint)
- Look at sum of integers program again

- The key to this is to use a variation on the READ statement

(assume input redirection is being used. The program using OPEN/CLOSE would be similar)

```
      INTEGER :: rvalue, total

      total = 0
loop1 DO
         READ(*,*, END=100) rvalue
         total = total + rvalue
      ENDDO
100   CONTINUE
```

NO WRITE prompt since I don't plan on using keyboard input

comments

- 100 is a line number. Any line can be numbered, but this is usually only done in cases where it is needed
- line numbers must be integers and unique.
- They don't have to be sequential, but they usually are.
  - READ(*,*, END=100) rvalue
    This means "read rvalue, but if there is an input error, jump to line # 100."
  - The loop will jump to 100 when the end of the file is reached (EOF error)
  - CONTINUE is a dummy statement. It does nothing.