Iteration – the DO Loop

In this exercise you will use the DO statement to repeat the same section of a program over and over again.

1. Problem:

On large cargo or passenger aircraft there may be many weight and balance stations, and the way in which the information is recorded and fed to the computer may be different. For example, as a cargo plane is loaded the loadmaster might fill in the amount of weight added to a certain station on a pre-printed form created just for that purpose. The data would be entered into the computer from that form later on, perhaps by an assistant. In such cases it may be more useful to read in the weight and balance data for all of the stations (a fixed and known number) rather than waiting for some special signal to end input. To do this it is easiest to use a DO loop to repeat the input step as many times as needed.

You should therefore modify your program from the previous exercise (or write a new one) so that it reads in the number of stations, followed by the arm and weight information for exactly that many stations, using a DO loop. After all of the data have been read, compute and display the position of the center of gravity and the total weight. Use IF statements, as in the previous exercise, to print out a warning if the center of gravity or weight are not within acceptable limits.

2. Input:

The first thing you read should be the number of stations for which data follows. After this, read the station arm and weight (in that order, as a pair) for exactly that many stations, no more and no less. DO NOT terminate input if both numbers are zero.

Don't forget to echo the values of all variables when they are read in.

In this exercise negative weights will not be allowed, since the program is not meant to be interactive. If a negative weight is entered your program should treat the weight as zero and print an error message. Do not go back to read data for this station again.

3. Output:

Print the gross weight of the loaded aircraft and the position of the center of gravity, as before. Also print the number of stations from which these computations were obtained, for verification. You may print this either before or after the other information. Try to organize your output into a nice, easy to read summary report.

If the weight or center of gravity are not within acceptable limits then print an appropriate warning message. This includes being above the diagonal line at the left of the C.G. limits shown in the figure in the last exercise (this was optional then; now it's not). Do not print such messages until all of the data have been read.

4. Style Notes – Indenting

Although Fortran statements can begin in any column between 7 and 72, you can make your programs much more easy to read and understand if you follow some simple guidelines for indenting. In the case of the `DO` loop you can easily set off the parts of your program which are executed repeatedly if you indent all of the repeated code by a couple of spaces.

It is considered good style to end the loop with a `CONTINUE` statement. This statement may or may not be indented, as you choose. It is better to terminate a loop with a numbered `CONTINUE` statement then it is to use an unlabeled `ENDDO` statement, because a labeled statement is easier to find when you are looking through a long code listing.

5. Optional Improvements:

You do not have to make these improvements to your program, but you can do so if you really want to make it work nicely:

- Print the station number each time before you read in the arm and weight for that station.
- Print the total weight (and C.G. location) after each set of data points has been entered.
- A weight of zero is acceptable, but print a warning if both the weight and arm are zero, since this may be an attempt to end the input, or it may be an indication that the wrong data are being used.

Some design advice: get your basic program working first, then add the optional improvements, one by one. It is much easier to build from a working program and fix any small problems which occur as you make small changes. In contrast, if you make many changes at once you will then have to try to figure out which of these introduced whatever problems might arise.

6. Emacs Features – program compilation

Emacs has some useful features which can make the task of entering and debugging a program much easier. Instead of saving your file and exiting emacs, then compiling your program and noting which lines contain errors, then editing your file again to find these lines and correct your errors, there is a much easier way. The `.emacs` file you initially copied to your home directory contains bindings of two useful commands for compiling and finding errors:

- If you type C-c c (control-C followed by C) then emacs will create another window, by splitting the screen, and run a compilation in that window. You do not have to save your file and exit or suspend emacs. Emacs will ask you if you want to save your file (press the "y" key or the space-bar to say yes). Then you will then be prompted for the compilation command. You should erase the default command (‘`make -k`‘) and enter your own Fortran compilation command (*e.g.*, ‘`f77 mavassar09.f -o mavassar09`‘). The next time you press C-c c this same command will be used, at least until you finally exit emacs.

- After the compilation, if there are errors, you can type C-c n (control-C followed by N) to go to the "next" error. You don't need to know which line the error occurred on, as emacs will figure that out for you. Pressing C-c n again takes you to the next error, and so on. You can start over simply by typing C-u C-c n.

The commands which make these functions possible are kept in your `.emacs` file. If you want to use them on another computer which does not have these "key bindings" you can add them by adding the following lines to your `.emacs` file:

```
(global-set-key "\C-cc" 'compile)       ; ^C-c is 'compile'
(global-set-key "\C-cn" 'next-error)    ; ^C-n is 'next-error'
```

You can bind other useful functions to keystrokes in emacs. Take a look at your `.emacs` file for more examples, and see the emacs manual for more information.

7. Reading

For this exercise you should read about iteration and looping (or "repetition") using the `DO` statement (also known as `DO` loops).

8. Model of a DO loop:

Iteration is used so much that Fortran includes the DO loop as a specialized construct for looping. But if it did not, you could still do the same thing with conditional execution, using just the `IF` and `GOTO` statements.

As an example, consider a loop that will add all the *even* numbers between 2 and 100. With a `DO` loop this is easily done as:

```
            ISUM=0
            DO 47 N=2,100,2
              SUM=SUM+N
    47        CONTINUE
            PRINT *,'The sum is ',ISUM
```

When this program is compiled, the result is essentially the same as the following:

```
            ISUM=0
            N=2
    29        CONTINUE
              SUM=SUM+N
              N=N+2
    47        IF(N.LE.100) GOTO 29
            PRINT *,'The sum is ',ISUM
```

One thing to notice is that in each of these examples the loop counter variable `N` is one step beyond the terminating loop value. In these examples `N` will have the value 102 after exiting the loop. If you don't believe it, try both examples and print out the value of `N` along with the sum.

"WARNING"

The Fortran convention that blanks are ignored and that undeclared variables have a default type can lead to unexpected and undesirable results. On July, 22, 1963, the U.S. Mariner I rocket was launched at Cape Canaveral on the first mission to Venus. It started veering toward the earth and was destroyed at a cost of $18.5 million. The reason was due to a period being mistakenly typed in place of a comma in a DO statement."

– "*The Essentials of FORTRAN*"
Rev. Dennis C. Smolarski, S.J., Ph.D