Chapter 10
Reading in Data

Winnie-the-Pooh read the two notices very carefully, first from left to right, and afterwards, in case he had missed some of it, from right to left

A A Milne, Winnie-the-Pooh

Aims
The aims of this chapter are to introduce some of the ideas involved in reading data into a program. In particular, using the following:

- Reading from files
- Reading integer data
- Reading real data
- Skipping columns of data in a file
- Skipping lines in a file
- Reading from several files consecutively
- Reading using internal files
- Timing of formatted and unformatted reads

10.1 Reading from Files

In the examples so far we have been reading from the keyboard using what Fortran calls list directed input. In this chapter we will look at reading data from files where the data is generally in tabular form.

I. Chivers and J. Sleightholme, *Introduction to Programming with Fortran*,
https://doi.org/10.1007/978-3-319-75502-1_10

10.2 Example 1: Reading Integer Data

In this example we are interested in reading in people's heights and weights in imperial measurements (feet and inches and stones and pounds) from a file and converting to their metric equivalent (metres and kilograms). The data is taken from an undergraduate class of Mechanical Engineering students.

Here is the data.

```
6   1  13   5
5  11  11  13
6   1  13   5
5   7  14   2
5   9  10  12
5   6  13   1
5   1  10   1
5   4   8  13
5  10  10   3
5  10  11  13
```

The first two columns are the heights in feet and inches, and the second two columns are the weights in stones and pounds.

Here is the program.

```
program ch1001
  implicit none
  integer, parameter :: npeople = 10
  integer, dimension (1:npeople) :: height_feet, &
    height_inch, weight_stone, weight_pound
  real, dimension (1:npeople) :: weight_kg, &
    height_m
  integer :: i

  open (unit=10, file='ch1001.txt',status='old')
  open (unit=20, file='ch1001.out',status='new')

  do i = 1, npeople
    read (10, fmt=100) height_feet(i), &
      height_inch(i), weight_stone(i), &
      weight_pound(i)
    100 format (i2, 2x, i2, 2x, i2, 2x, i2)
    weight_kg(i) = (weight_stone(i)*14+ &
      weight_pound(i))/2.2
    height_m(i) = (height_feet(i)*12+height_inch &
      (i))*2.54/100
    write (unit=20, fmt=110) height_m(i), &
```

```
        weight_kg(i)
      110 format (1x, f5.2, 2x, f4.1)
    end do
    close (10)
    close (20)
  end program ch1001
```

Here is the output.

```
  1.85  85.0
  1.80  75.9
  1.85  85.0
  1.70  90.0
  1.75  69.1
  1.68  83.2
  1.55  64.1
  1.63  56.8
  1.78  65.0
  1.78  75.9
```

The first statements of interest are

```
open (unit=10, file='ch1001.txt',status='old')
open (unit=20, file='ch1001.out',status='new')
```

which links the Fortran unit number `10` with a file called `ch1001.txt`, and links the Fortran unit number `20` with a file called `ch1001.out`.

The next statements of interest are

```
read(10,fmt=100)height_feet(i) ,height_inch(i), &
                weight_stone(i),weight_pound(i)
100 format(i2,2x,i2,2x,i2,2x,i2)
```

which reads 4 integer values from a line with integer data in columns 1–2, 5–6, 9–10 and 13–14 with 2 spaces between each value.

At the end of the program we close the files.

```
close(10)
close(20)
```

We write out the metric versions of the height and weight with the following statement.

```
    write(unit=20,fmt=200) height_m(i),weight_kg(i)
    200 format(1x,f5.2,2x,f4.1)
```

to the file called `ch1001.out`.

We recommend that when working with formatted files you use a text editor that displays the column and line details.

Notepad under Windows has a status bar option under the View menu. Gvim under Windows has line and column information available. Under Redhat, vim and gedit both display line and column information. Under SuSe Linux kedit and vim display line and column information. There should be an editor available on your system that has this option.

10.3 Example 2: Reading Real Data

This example reads in the height and weight data created by the previous program and calculates their BMI values. BMI stands for Body Mass Index and is calculated as *Weight/Height*^{2}

Here is the program.

```
program ch1002
  implicit none
  integer, parameter :: n = 10
  real, dimension (1:n) :: h
  real, dimension (1:n) :: w
  real, dimension (1:n) :: bmi
  integer :: i

  open (unit=100, file='ch1001.out',status='old')
  do i = 1, n
    read (100, fmt='(1x,f5.2, 2x, f4.1)') h(i), &
      w(i)
  end do
  close (100)
  bmi = w/(h*h)
  do i = 1, n
    write (unit=*, fmt='(1x,f4.1)') bmi(i)
  end do
end program ch1002
```

The following statement

```
  open(unit=100,file='ch1001.out',status='old')
```

links the Fortran unit number `100` with the file `ch1001.out`.

The following statement

```
    read (100,fmt='(1x,f5.2, 2x, f4.1)') h(i), w(i)
```

reads the height and weight data from the file. We skip the first space then read the height from the next 5 columns in `f5.1` format. We skip two spaces and then read the weight from the next 4 columns in `f4.1` format.

The following statement

```
close(100)
```

closes the file.

The following statement

```
write(unit=*,fmt='(1x,f4.1)') bmi(i)
```

writes out the BMI values in `f4.1` format.

Here is the output.

```
24.8
23.4
24.8
31.1
22.6
29.5
26.7
21.4
20.5
24.0
```

10.4 Met Office Historic Station Data

The UK Met Office makes historic station data available.
Visit

```
http://www.metoffice.gov.uk/public/weather/
climate-historic/#?tab=climateHistoric
```

to see the data. The line has been broken to fit the page width.

The data consists of

- Mean daily maximum temperature (tmax)
- Mean daily minimum temperature (tmin)
- Days of air frost (af)
- Total rainfall (rain)
- Total sunshine duration (sun)

Here is a sample of the Nairn data. Nairn is a town in Scotland on the North Sea. The first seven lines have had to be formatted to fit the page width.

```
Nairn   there is a site change in 1998
Location before 1998 2869E 8568N 8m amsl
  after 1998 2912E 8573N 23 m amsl
Estimated data is marked with a * after the value.
Missing data (more than 2 days missing in month)
  is marked by  ---.
Sunshine data taken from an automatic Kipp &
  Zonen sensor marked with a #, otherwise
  sunshine data taken from a
    Campbell Stokes recorder.
 yyyy  mm   tmax    tmin      af    rain     sun
            degC    degC    days      mm   hours
 1931   1    5.0     0.6      11    78.4    43.4
 1931   2    6.7     0.7       7    48.9    63.6
 1931   3    6.2    -1.5      19    37.6   145.4
 1931   4   10.4     3.1       3    44.6   110.1
 1931   5   13.2     6.1       1    63.7   167.4
 1931   6   15.4     8.0       0    87.8   150.3
 1931   7   17.3    10.6       0   121.4   111.2
 1931   8   15.6     9.1       0    57.2   127.5
 1931   9   15.0     8.4       0    38.1   122.3
 1931  10   12.1     5.5       2    59.4    95.8
 1931  11   10.3     3.9       3    43.7    61.5
 1931  12    8.9     3.2       7    33.6    36.5
```

In the examples that follow we will be using this station's data.

10.5 Example 3: Reading One Column of Data from a File

Here is the file we will be reading the rainfall values from.

```
 1931   1    5.0     0.6      11    78.4    43.4
 1931   2    6.7     0.7       7    48.9    63.6
 1931   3    6.2    -1.5      19    37.6   145.4
 1931   4   10.4     3.1       3    44.6   110.1
 1931   5   13.2     6.1       1    63.7   167.4
 1931   6   15.4     8.0       0    87.8   150.3
 1931   7   17.3    10.6       0   121.4   111.2
 1931   8   15.6     9.1       0    57.2   127.5
 1931   9   15.0     8.4       0    38.1   122.3
 1931  10   12.1     5.5       2    59.4    95.8
 1931  11   10.3     3.9       3    43.7    61.5
 1931  12    8.9     3.2       7    33.6    36.5
12345678901234567890123456789012345678901234567890
         1         2         3         4         5
```

We have added two additional lines at the end to indicate the columns where the data is. These lines are not read by the program.

Here is the program.

```
program ch1003
  implicit none
  character *20 :: file_name = &
    'nairndata_01.txt'
  integer, parameter :: nmonths = 12
  real, dimension (1:nmonths) :: rainfall
  real :: rain_sum
  real :: rain_average
  integer :: i

  open (unit=10, file=file_name)
  do i = 1, nmonths
    read (unit=10, fmt=100) rainfall(i)
100 format (37x, f5.1)
  end do
  close (10)
  rain_sum = sum(rainfall)/25.4
  rain_average = rain_sum/nmonths
  write (unit=*, fmt=110)
    110 format (19x, 'Yearly   Monthly', /, 19x, &
    'Sum      Average')
  write (unit=*, fmt=120) rain_sum, rain_average
    120 format ('Rainfall (inches) ', f7.2, 2x, &
    f7.2)
end program ch1003
```

The data file is called `nairndata_01.txt` and we open the file at the start of the program and associate the file with unit `100`.

The following statements read the 12 monthly values from the file skipping the first 37 characters.

```
  do i=1,nmonths
    read(unit=10,fmt=100) rainfall(i)
    100 format(37x,f5.1)
  end do
```

We then close the file and calculate the rainfall sums and average and print out the results. Here is the output.

```
                    Yearly   Monthly
                    Sum      Average
Rainfall (inches)    28.13     2.34
```

The format statement 110 uses a / to move to the next line, so that the headings line up.

10.6 Example 4: Skipping Lines in a File

This program is a simple variant of the last one.

Now we are reading from the original Met Office Nairn data file, which has seven header lines.

```
program ch1004
  implicit none
  character *20 :: file_name = 'nairndata.txt'
  integer, parameter :: nmonths = 12
  real, dimension (1:nmonths) :: rainfall
  real :: rain_sum
  real :: rain_average
  integer :: i

  open (unit=10, file=file_name,status='old')
  do i = 1, 8
    read (unit=10, fmt=*)
  end do
  do i = 1, nmonths
    read (unit=10, fmt=100) rainfall(i)
100 format (37x, f5.1)
  end do
  close (100)
  rain_sum = sum(rainfall)/25.4
  rain_average = rain_sum/nmonths
  write (unit=*, fmt=110)
110 format (19x, ' Yearly   Monthly', /, 19x, &
    ' Sum      Average')
  write (unit=*, fmt=120) rain_sum, rain_average
120 format ('Rainfall  (inches) ', f7.2, 2x, &
    f7.2)
end program ch1004
```

The key statements are

```
do i=1,8
  read(unit=10,fmt=*)
end do
```

which skips the data on these lines. Fortran reads a record at a time in this example.

The output is as before.

10.7 Example 5: Reading from Several Files Consecutively

In this example we read from eight of the Met Office data files for Cardiff, Eastbourne, Lerwick, Leuchars, Nairn, Paisley, Ross On Wye and Valley.

We skip the first seven lines, then read year, month rainfall and sunshine data, skipping the other columns.

We then calculate rainfall and sunshine yearly totals and averages for these eight stations.

We use a character array to hold the station file names.

Here is the program.

```
program ch1005
  implicit none
  character *20, dimension (8) :: file_name = (/ &
    'cardiffdata.txt    ', 'eastbournedata.txt ' &
    , 'lerwickdata.txt    ', &
    'leucharsdata.txt   ', 'nairndata.txt      ' &
    , 'paisleydata.txt    ', &
    'rossonwyedata.txt  ', 'valleydata.txt     ' &
    /)

  integer, parameter :: nmonths = 12
  integer, dimension (1:nmonths) :: year, month
  real, dimension (1:nmonths) :: rainfall, &
    sunshine
  real :: rain_sum
  real :: rain_average
  real :: sun_sum
  real :: sun_average
  integer :: i, j
  character *80 :: fmt1 = '(3x,i4,2x,i2,3x,4x,4x,&
    &4x,4x,4x,3x,f5.1,3x,f5.1)'
```

```
  do j = 1, 8
    open (unit=100, file=file_name(j),status='old')
    do i = 1, 7
      read (unit=100, fmt='(a)')
    end do
    if (j==5) then
      read (unit=100, fmt='(a)')
    end if
    do i = 1, nmonths
      read (unit=100, fmt=fmt1) year(i), &
        month(i), rainfall(i), sunshine(i)
    end do
    close (100)
    rain_sum = sum(rainfall)/25.4
    sun_sum = sum(sunshine)
    rain_average = rain_sum/nmonths
    sun_average = sun_sum/nmonths
    write (unit=*, fmt='(//,"Station = ",a,/)') &
      file_name(j)
    write (unit=*, fmt= &
      '(2x,''Start '',i4,2x,i2)') year(1), &
      month(1)
    write (unit=*, fmt= &
      '(2x,''End   '',i4,2x,i2)') year(12), &
      month(12)
    write (unit=*, fmt=100)
100 format (19x, ' Yearly   Monthly', /, 19x, &
      ' Sum      Average')
    write (unit=*, fmt=110) rain_sum, &
      rain_average
110 format ('Rainfall  (inches) ', f7.2, 2x, &
      f7.2)
    write (unit=*, fmt=120) sun_sum, sun_average
120 format ('Sunshine           ', f7.2, 2x, &
      f7.2)
  end do

end program ch1005
```

Each time round the loop we open one of the data files.

```
    open(unit=100,file=file_name(j),status='old')
```

We then skip the next seven lines.

```
do i=1,8
  read(unit=100,fmt='(a)')
end do
```

We then read the data.

```
do i=1,nmonths
  read(unit=100,fmt=fmt1) &
  year(i),month(i),&
  rainfall(i),sunshine(i)
end do
```

We then close the file.

```
close(100)
```

We then do the calculations and print out the sum and average data for each site. The format statement uses // to generate a blank line.

Programs that will download the latest versions of the Met Office station data files are available on our web site. The programs are available for both Windows and Linux.

10.8 Example 6: Reading Using Array Sections

Consider the following output, which is the exam results data from an earlier chapter after scaling.

```
 50.0  47.0  70.0  89.0  30.0  46.0
 37.0  67.0  85.0  65.0  68.0  98.0
 25.0  45.0  65.0  48.0  10.0  36.0
 89.0  56.0  82.5  45.0  30.0  65.0
 68.0  78.0  95.0  76.0  98.0  65.0
```

A program to read this file using array sections is as follows:

```
program ch1006
  implicit none
  integer, parameter :: nrow = 5
  integer, parameter :: ncol = 6
  real, dimension (1:nrow, 1:ncol) :: &
    exam_results = 0.0
```

```
  real, dimension (1:nrow) :: people_average = &
    0.0
  real, dimension (1:ncol) :: subject_average = &
    0.0
  integer :: r, c

  open (unit=100, file='ch1006.txt',status='old')
  do r = 1, nrow
    read (unit=100, fmt=100) exam_results(r, &
      1:ncol)
    people_average(r) = sum(exam_results(r,1: &
      ncol))
  end do
  close (100)
  people_average = people_average/ncol
  do c = 1, ncol
    subject_average(c) = sum(exam_results(1:nrow &
      ,c))
  end do
  subject_average = subject_average/nrow
  do r = 1, nrow
    print 110, (exam_results(r,c), c=1, ncol), &
      people_average(r)
  end do
  print *, &
    '  ====  ====  ====  ====  ====  ===='
  print 120, subject_average(1:ncol)

100 format (1x, 6(1x,f5.1))
110 format (1x, 6(1x,f5.1), ' = ', f6.2)
120 format (1x, 6(1x,f5.1))
end program ch1006
```

Here is the output.

```
  50.0  47.0  70.0  89.0  30.0  46.0 =  55.33
  37.0  67.0  85.0  65.0  68.0  98.0 =  70.00
  25.0  45.0  65.0  48.0  10.0  36.0 =  38.17
  89.0  56.0  82.5  45.0  30.0  65.0 =  61.25
  68.0  78.0  95.0  76.0  98.0  65.0 =  80.00
  ====  ====  ====  ====  ====  ====
  53.8  58.6  79.5  64.6  47.2  62.0
```

10.9 Example 7: Reading Using Internal Files

Sometimes external data does not have a regular structure and it is not possible to use the standard mechanisms we have covered so far in this chapter. Fortran provides something called internal file that allow us to solve this problem. The following example is based on a problem encountered whilst working at the following site

```
http://www.shmu.sk/sk/?page=1
```

They have data that is in the following format

```
#xxxxxxxxxx yyyyyyyyyy
```

where x and y can vary between 1 and 10 digits. The key here is to read the whole line (a maximum of 22 characters) and then scan the line for the blank character between the x and y digits.

We then use the `index` intrinsic to locate the position of the blank character. We now have enough information to be able to read the x and y integer data into the variables `n1` and `n2`.

```
program ch1007
  implicit none
  integer :: ib1, ib2
  integer :: n1, n2
  character (len=22) :: buffer, buff1, buff2
! program to read a record of the form
! #xxxxxxxxxx yyyyyyyyyy
! so that integers n1 = xxxxxxxxxx n2 =
! yyyyyyyyyy
! where the number of digits varies from 1 to 10
!
! use internal files
  print *, 'input micael''s numbers'
  read (*, '(a)') buffer
  ib1 = index(buffer, ' ')
  ib2 = len_trim(buffer)
  buff1 = buffer(2:ib1-1)
  buff2 = buffer(ib1+1:ib2)
  read (buff1, '(i10)') n1
  read (buff2, '(i10)') n2
  print *, 'n1 = ', n1
  print *, 'n2 = ', n2
end program ch1007
```

The statement

```
read(buff1,'(i10)')n1
```

reads from the string `buff1` and extracts the x number into the variable `n1`, and the statement

```
read(buff2,'(i10)')n2
```

reads from the string `buff2` and extracts the y number into the variable `n2`.

This is a very powerful feature and allows you to manage quite widely varying external data formats in files. `buff1` and `buff2` are called internal files in Fortran terminology.

10.10 Example 8: Timing of Reading Formatted Files

A program to read a formatted file is shown below:

```
program ch1008
  implicit none
  integer, parameter :: n = 10000000
  integer, dimension (1:n) :: x
  real, dimension (1:n) :: y
  integer :: i
  real :: t, t1, t2, t3
  character *15 :: comment

  call cpu_time(t)
  t1 = t
  comment = ' Program starts '
  print 120, comment, t1
  open (unit=10, file='ch0913.txt', &
    status='old')
  do i = 1, n
    read (10, 100) x(i)
  end do
  call cpu_time(t)
  t2 = t - t1
  comment = ' Integer read '
  print 120, comment, t2
  do i = 1, n
    read (10, 110) y(i)
```

```
  end do
  call cpu_time(t)
  t3 = t - t1 - t2
  comment = ' Real read '
  print 120, comment, t3
  do i = 1, 10
    print 130, x(i), y(i)
  end do
100 format (1x, i10)
110 format (1x, f10.0)
120 format (1x, a, 2x, f7.3)
130 format (1x, i4, 2x, f10.7)

end program ch1008
```

Here is some sample timing.

```
  Program starts    0.016
  Integer read      2.964
  Real read         4.072
    1   1.0000000
    2   2.0000000
    ...
    ...
    9   9.0000000
   10  10.0000000
```

10.11 Example 9: Timing of Reading Unformatted Files

The following is a program to read from an unformatted file:

```
program ch1009
  implicit none
  integer, parameter :: n = 10000000
  integer, dimension (1:n) :: x
  real, dimension (1:n) :: y
  integer :: i
  real :: t, t1, t2, t3
  character *15 :: comment

  call cpu_time(t)
  t1 = t
  comment = ' Program starts '
```

```
  print 100, comment, t1
  open (unit=10, file='ch0914.dat', &
    form='unformatted',status='old')
  read (10) x
  call cpu_time(t)
  t2 = t - t1
  comment = ' Integer read '
  print 100, comment, t2
  read (10) y
  call cpu_time(t)
  t3 = t - t1 - t2
  comment = ' Real read '
  print 100, comment, t3
  do i = 1, 10
    print 110, x(i), y(i)
  end do
100 format (1x, a, 2x, f7.3)
110 format (1x, i10, 2x, f10.6)
end program ch1009
```

Here is some sample timing.

```
 Program starts    0.031
 Integer read      0.016
 Real read         0.031
         1    1.000000
         2    2.000000
         ...
         9    9.000000
        10   10.000000
```

10.12 Summary

This chapter has provided a coverage of some of the basics of reading data into a program in Fortran. We have seen examples that have

- Read integer data
- Read real data
- Skipped lines in a file
- Skipped columns of data in a file
- Read from files
- Used the open and close statements
- Associated unit numbers with files

- Read using fixed format data files
- Shown the time difference between using formatted files and unformatted files
- Used internal files

The above coverage should enable you make effective use of reading data in Fortran.

We would recommend not using edit descriptors when reading numeric data entered via the keyboard as it is difficult to see if the data matches what the edit descriptors expect.

10.13 Problems

10.1 Compile and run the examples in this chapter. Note that you will have to run ch0913.f90 and ch0914.f90 to create the data files that are needed by ch1008.f90 and ch1009.f90

10.2 Write a program to read in and write out a real number using the following:

```
format(f7.2)
```

What is the largest number that you can read in and write out with this format? What is the largest negative number that you can read in and write out with this format? What is the smallest number, other than zero, that can be read in and written out?

10.3 Rewrite two of the earlier programs that used `read,*` and `print,*` to use format statements.

10.4 Write a program to read the file created by either the temperature conversion program or the litres and pints conversion program. Make sure that the programs ignore any header and title information. This kind of problem is very common in programming (writing a program to read and possibly manipulate data created by another program).

10.5 Demonstrate that input and output formats are not symmetric — i.e., what goes in does not necessarily come out.

10.6 What happens at your computer when you enter faulty data, inappropriate for the formats specified? We will look at how we address this problem in Chap. 18.