2 Making Decisions

2.1 Aims
By the end of this worksheet, you will be able to:
- Do arithmetic
- Start to use FORTRAN intrinsic functions
- Begin to understand program flow and logic
- Know how to test for zero – important!
- Learn more about good programming style

2.2 Assignment
When we start programming, the similarity between mathematical equations and FORTRAN statements can be confusing. Consider the following FORTRAN statements:

\[ \begin{align*}
  x &= 2 & \text{Store the value 2 in memory location x} \\
  y &= 3 & \text{Store the value 3 in memory location y} \\
  z &= x + y & \text{Add the values stored in memory location x and y and store the result in memory location z}
\end{align*} \]

In mathematics, “\(x = 2\)” means that the variable \(x\) is equal to 2. In FORTRAN it means “store the value 2 in the memory location that we have given the name \(x\).”

The significance of this is made clearer by the following equation in mathematics:

\[ x + y = z \]

In mathematics, this means that the left hand side of the equation is equal to the right hand side. In FORTRAN, this expression is meaningless: there is no memory location “\(x+y\)” and so it would lead to a compiler error.

Rule – there can only ever be ONE variable name on the left hand side of an equals sign

Exercise 2.1
Write a program which reads in two numbers \(a\) and \(b\). Get the program to swap the values around so that the value that was in \(a\) is now in \(b\), and print out the result. Hint you need to declare a third variable for intermediate storage of the data. (Check your program by examining program swap.f95 at http://fortrantutorial.com/fortrantutorial-example-programs/)

2.3 Arithmetic
The arithmetic operators are
- \(+, -\) plus and minus
- \(*, /\) multiply and divide
- \(\star\) exponentiation (raise to the power)
- \(()\) brackets
- The order of precedence in FORTRAN is identical to that of mathematics.
- Unlike algebra, the operator must always be present \(xy\) is NOT the same as \(x\star y\)
- Where operations are of equal precedence they are evaluated left to right
- Consecutive exponentiations are evaluated right to left
- We can override the order of evaluation by use of brackets
Exercise 2.2

The following program is an example of the use of arithmetic.

```fortran
program calculate
implicit none
! a simple calculator
real :: x, y, z, answer
x=1.5
y=2.5
z=3.5
answer=x+y/z
print *, 'result is ', answer
end program calculate
```

Explore the use of arithmetic operators by modifying program `calculate`. Use it to calculate the values:

1. $\frac{x + y}{x + z}$
2. $xyz$
3. $x^2$

2.4 Intrinsic Functions

FORTRAN is especially useful for mathematical computation because of its rich library of inbuilt functions (intrinsic functions). We shall mention a few briefly here:

<table>
<thead>
<tr>
<th>function name</th>
<th>type of argument</th>
<th>type of result</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin(x)</td>
<td>real</td>
<td>real</td>
<td>sine</td>
</tr>
<tr>
<td>cos(x)</td>
<td>real</td>
<td>real</td>
<td>cosine</td>
</tr>
<tr>
<td>tan(x)</td>
<td>real</td>
<td>real</td>
<td>tangent</td>
</tr>
<tr>
<td>atan(x)</td>
<td>real</td>
<td>real</td>
<td>arctangent</td>
</tr>
<tr>
<td>abs(x)</td>
<td>real/integer</td>
<td>real/integer</td>
<td>absolute value</td>
</tr>
<tr>
<td>sqrt(x)</td>
<td>real</td>
<td>real</td>
<td>square root</td>
</tr>
<tr>
<td>exp(x)</td>
<td>real</td>
<td>real</td>
<td>$e^x$</td>
</tr>
<tr>
<td>log(x)</td>
<td>real</td>
<td>real</td>
<td>$\log_{10}x$</td>
</tr>
</tbody>
</table>

Trigonometric functions are calculated in radians (1 radian = 180/Pi degrees).

There are, of course, many more, and this list doesn't cover all FORTRAN variable types. The following example shows the use of some of the inbuilt functions.

```fortran
program trig
implicit none
real :: a, pi
print *, 'Enter an angle between 0 and 90'
read *, a
pi=4.0*atan(1.0)
print *, 'the sine of ', a, ' is ', sin(a*pi/180)
end program trig
```
2.5 Making Decisions

So far, our programs have worked as little more than basic calculators. The power of programming comes in when we have to make decisions. Copy the example program, test.f95, to your own file space. See if you can understand what is going on.

```fortran
program test
  implicit none
  !use of a simple menu
  real :: x,y,answer
  integer :: choice
  !set up the menu – the user may enter 1, 2 or 3
  print *, 'Choose an option'
  print *, '1    Multiply'
  print *, '2    Divide'
  print *, '3    Add'
  read *,choice
  x=3.4
  y=2.9
  !the following line has 2 consecutive
  !equals signs – (no spaces in between)
  if (choice == 1) then
    answer=x*y
    print *, 'result = ',answer
  end if
  if (choice == 2) then
    answer=x/y
    print *, 'result = ',answer
  end if
  if (choice == 3) then
    answer=x+y
    print *, 'result = ',answer
  end if
end program test
```

The bolded lines in the above program are called if...end if statements. They work like this:

```
if (condition is true) then
  execute this line
  and this
  and so on until we get to ...
end if
```

It follows that if the condition is NOT true then the code 'jumps' to the next statement following the 'end if'. The statements between the if and the end if are deliberately indented, this makes the program easier to read.

We use two consecutive equals signs (no space in the middle) to test for equality. Compare

```fortran
if (choice == 3) then
  test
choice = 3
  assignment
```
Exercise 2.3

Examine program test above. The line

\[ \text{print *,'result = ',answer} \]

is repeated several times. Is this a good idea? Can you modify the program to make it more efficient?

2.6 Program Style

A good program:

- Uses comments \textit{appropriately} to explain what is happening.
- Uses indentation to make the program easier to read.
- Uses \textit{meaningful} variable names.
- Uses sensible prompts to let the user know what is going on.
- Uses \textit{implicit none} at the start of every program.
- Is efficient!

If you want to get maximum marks for your assignments keep the above points firmly in mind. It is \textit{not enough just to get a program to work}!

2.7 More on decision making

In our test.f95 above, there was a problem if the user entered a value that wasn't catered for by the program.

What happens if the user doesn't enter one of the values 1, 2 or 3?

We are going to look at a new structure, called \texttt{if, else, endif} that handles this situation. Examine the following code snippet:

\[
\begin{array}{ll}
\text{if (choice } = 1) \text{ then} & \text{do something} \\
\text{else if (choice } = 2) \text{ then} & \text{do something else} \\
\text{else} & \text{do this if nothing else satisfies the conditions} \\
\text{endif}
\end{array}
\]

2.8 Other logical operators

So far, all our tests have been for \textit{equality}. There are several tests we can make:

\[
\begin{array}{ll}
= &= \text{equal to } \text{(there is no space between the equals signs)} \\
/ &= \text{not equal to} \\
< &= \text{less than} \\
\leq &= \text{less than or equal to} \\
> &= \text{greater than} \\
\geq &= \text{greater than or equal to}
\end{array}
\]
2.9 Multiple Conditions

Suppose we need to test if x is greater than y and y is greater than z. There are different ways of doing this:

```fortran
   if (x > y) then
      if (y > z) then
         do something
      end if
   end if
```

This can also be handled by the following:

```fortran
   if (x > y .and. y > z) then
      do something
   end if
```

If we wanted to check whether a number were less than a given value or greater than a given value we could write:

```fortran
   if (x < 10 .or. x > 20) then
      do something
   end if
```

Exercise 2.4

Write a program that reads a number from the keyboard. Get the program to decide whether:

- the value of the number is greater than 0 but less than 1
- or is greater than 1 but less than 10
- or is outside of both these ranges

Print out a suitable message to inform the user.

2.10 The simple if statement

There is a simpler, one line form of the if statement. Say we just wanted to print out a simple message such as

```fortran
   print *, 'enter a positive number'
   read *, num
   if (num < 0) stop
   if (num < 10) print *, 'less than 10'
   if (num > 10) print *, 'greater than 10'
   print *, 'It is a positive number'
```

This snippet also introduces a useful, simple statement stop – it simply stops the program.
2.11 Important note – testing for zero

Suppose that you wish to test whether a real variable is zero. The test

    if (x = = 0) then ...

is not a satisfactory test. Although integer numbers are held exactly by the computer, real numbers are not.

The way around this is to test if the absolute value of the variable is less than some small predefined value. For example:

    if (abs(x) < .000001) then
        print *,’No zero values! Please enter another number’
        read *, x
    end if