Reading, Writing and Buffering.

This is a small aside on how to make your program's output look better. It is important in that you need to make your programs usable, and be able to present the results in a clear way. Reading a spreadsheet would be impossible if all the data were all on one line.

We call this massaging of output into something legible formatting.

1 Fields and Field Width.

We start with two definitions.

1) A group of columns in which results appear is called a field.

2) The number of columns of text allowed in a field is called the field width.

<table>
<thead>
<tr>
<th>Field1</th>
<th>Field 2</th>
<th>Field 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>--------</td>
<td>-------</td>
</tr>
<tr>
<td>3.1</td>
<td>41345661</td>
<td>Orinoco</td>
</tr>
<tr>
<td>5.1</td>
<td></td>
<td>Ganges</td>
</tr>
<tr>
<td>2.3</td>
<td>34568</td>
<td>Thames</td>
</tr>
<tr>
<td>8.1</td>
<td>90</td>
<td>Hudson</td>
</tr>
</tbody>
</table>

Width=3 Width=8 Width = 12

NOTE: The _ denote spaces that are unused in this example.

2 Formatting Integer Output.

We need to pay attention to field width when talking about formatting. Look at the behaviour of this program.

The first two Writeln()’s give us a frame of reference on the screen, their only purpose is to illustrate what the third Writeln() does.

```pascal
PROGRAM PrintTwo(output);
CONST A = 123, B = 4567;
BEGIN
  Writeln('0 1 2 3');
  Writeln('012345678901234567890123456789012345');
  Writeln(A,B) (* uses fields of width 16 *)
END.
```

This gives the output

```
0123
123 4567
```

3 Formatting Real Output.

For Reals Write() and Writeln() has the form: Writeln(ValueToPrint) where ValueToPrint can either be

- SomeRealValue in which case we use the default field width,
- SomeRealValue:Width where we specify the width in number of columns,
- SomeRealValue:Width:Precision where we specify the width in number of columns, and the number of places after the decimal point that we want to print.

NOTE: The decimal point occupies one column as well, so if we specify the field to be w characters wide and we want d characters after the decimal point then there are only w - d - 1 places left before the decimal point.

```
PROGRAM PrintTwo(output);
CONST A = 123, B = 4567;
BEGIN
  Writeln('0 1 2 3');
  Writeln('012345678901234567890123456789012345');
  Writeln(A,10,B:10) (* uses fields of width 16 *)
  Writeln(A,B) (* uses fields of width 16 *)
END.
```

This gives the output

```
0     1     2     3
012345678901234567890123456789012345
123   4567
```

we will get

```
0     1     2     3
012345678901234567890123456789012345
123   4567
```
This program demonstrates what happens when a real value is output.

```
PROGRAM Field2(output)
(* Introduction to formatted output *)
VAR Sample:Real
BEGIN
  Sample := 123.65;
  Writeln('012345678901234567890123456789012345');
  Writeln(Sample);
  (* Produces 012345678901234567890123456789012345 *)
  Writeln('012345678901234567890123456789012345');
  Writeln(Sample:10:3);
  (* To get the non exponential representation of 123.65 we specify the column and then using a second colon the amount of precision needed. *)
  Writeln(Sample:10:2);
  (* Similarly for two decimal places we set the precision field to 2. *)
  Writeln(Sample:12);
  (* If the field isn’t quite large enough you may see the mantissa truncated and rounded accordingly. *)
  Writeln(Sample:2:1)
  (* What is the correct representation of 123.65 when you only have one place after the decimal point? *)
  Writeln('012345678901234567890123456789012345');
  Writeln(Sample:10);"
```

4 Formatting Strings and Characters.

For characters and strings Write() and Writeln() has the form: Write/StringToPrint where StringToPrint can either be

- `SomeCharacterValue` the string is just printed starting from the first available column.
- `SomeCharacterValue:Width` where we specify the width in number of columns. Here the string is printed right aligned in the field.

Characters are treated like special cases of strings of length = 1. Look at the following Pascal code.

```
PROGRAM StringDemo(output);
VAR Ch: Char;
BEGIN"
```
5 Formatting Booleans.

Consider the following program:

```pascal
PROGRAM Bool(output)
(* Investigates boolean output *)
VAR First, Second: boolean;
BEGIN
  Writeln(‘12345678901234567890’)
  First := True;
  Second := False;
  Writeln(First,Second,TRUE,FALSE);
  Writeln(First:10,Second:10);
  Writeln(First:1,Second:1)
END.
```

This produces the following output

```
12345678901234567890
TRUEFALSETRUEFALSE
TF
```

The compiler will print booleans as strings TRUE and FALSE according to the rules it uses for printing strings. What is important to notice here is that true = TRUE = tRuE and False = FalSE = false = FALSE. ie the compiler ignores case on booleans.

6 Reading Numerical Values.

We read in numerical values with the `Readln()` procedure. `Readln()` works as follows.

1. You specify the values which are to be read in, and the compiler expects to see legal values as input.
2. The values being entered must be of the same type as the values expected.
3. The values must be separated by white space.

```pascal
PROGRAM Area(input,output);
(* Investigates Reading in numbers *)
VAR Length, Width: integer;
      Cost, Price:real;
BEGIN
  Writeln(‘Type in Lenght, Width and Price`
Readln(Length, Width, Price);
Write('Length = ', Length, ' Width = ', Width, ' Price = ', Price:5:2);
(* Now calculate the cost *)
Cost := Length * Width * Price;
Write('Cost := ', Cost:5:2)
END.

When prompted by

Type in Length, Width and Price

we can type in

22 4 1.0

or

22 4 1.0

both of these are legal because we have white spaces separating the text.

The program will ignore leading white spaces too.

If we’d typed in

22 4 1.0

Our program will read in 22 and then look to it’s right for more input data. It will keep reading in data until all the variables have been input.

Suppose we had too much data, ie. our Readln() statement was replaced by:

readln(Length, Width);
readln(Cost);

What would happen if we keyed in the following?

22 4 1.0 [CR]

The program will read in the value for Length, ie. 22, then it’ll read in the value for Width, ie. 4. But now it will ignore any values until the new line, then it’ll accept a value for cost.

So with our input as shown above the program will only have set Length and Width, it will ignore 1.0 and then after a new line it will expect more input, to set the variable cost.

7 Read()

There is also a standard procedure called Read() which we can use to read in input from the keyboard. The statement

readln(Length, Width, Price);
readln(Length, Width, Price);
readln(Cost:5:2);

could easily have been replaced by

read(Length, Width);
read(Price);

Read keeps the pointer on the same line, so the program expects the price to be before any carriage return.

Now it would be all right to have:

22 4 1.0 [CR]

and

22 4 [CR]

1.0

8 Reading Characters.

Reading variables that are characters is simply a matter of using Readln() or Read() with character variables.

The only difference here is we don’t need to separate characters by white space because a character can only be a single symbol long.

PROGRAM ReverseChars(input, output);
(* Simple program to reverse characters *)
VAR A, B, C, D, E, F, G: char;
BEGIN
   Readln('Type 6 characters');
   Read(a);
   Read(b);
   Read(c);
   Read(d);
   Read(e);
   Read(f);
   Read(g);
   Write('The letters in reverse order are', g, f, e, d, c, b, a);
END.

So on input,

reverse

we’d get the output

esrever

Don’t forget that the blanks, tabs and [CR] are characters too so if we had typed the following input
reverse

we’d get

esre

because B, D, and E are assigned blanks; A, C, E, and F are assigned the values e, s, r, and e respectively.

So what happens with the following input?

rev[CR]
erse

The answer is

sre[CR]
ver

because D got the value [CR] and the last e in reverse is ignored because the program doesn’t call for it as input.

9 Reading Characters and Numbers Together.

We can use Readln() and Read() for Characters and Numbers at the same time. But we need to be careful.

Consider this program;

```pascal
PROGRAM Tricky(input,output);
VAR A, B, C:char;
Digit1, Digit2:integer;
BEGIN
  Writeln('Enter two integers and two characters');
  Readln(Digit1, Digit2, A, B);
  Writeln('Digit1=',Digit1,' Digit2=', Digit2,' A=', A,' B=',B);
END.
```

with the following input:

```
12 34 cd
```

What is the output to the above program? The output is

```
Digit1=12 Digit2=34 A= B=c
```

this is because the delimiter between numbers, integers or reals is a space, but there is none for a character. So after the program reads the first number it ignores the space because it knows [space] can be no part of a number. The space after 34 however can be a character and is read in as one. A is then set to [space] and B is set to c.

10 Buffering.

A buffer on a computer is a temporary store for a stream of data. Anything that we type on the keyboard, whether it is used or not is entered into a buffer called the input buffer. We have something called an output buffer too, that stores whatever we’re going to send to the screen.

Typing ”123 2.34554 3445 mary had a little lamb” at the keyboard puts

```
123 2.34554 3445 mary had a little lamb
```

into the buffer.

Every character is read into the buffer until the program can decide what to do with it.

When reading input from the keyboard we use predefined procedures such as Read() and Readln() which both utilize the input buffer in different ways..

10.1 Read() and the Buffer.

To see how Read() modifies the buffer consider the following program.

```pascal
PROGRAM IntroBuff(input,output);
VAR A, B, C:integer;
BEGIN
  Writeln('Type 3 integers on a line, then hit CR');
  Read(A);
  Writeln('A=',A:2);
  Read(B);
  Writeln('B=',B:2);
  Read(C);
  Writeln('C=',C:2)
END.
```

When we run the program and type

```
23 45 67 [CR]
```

The data is stored in the buffer and ends with the end of line marker as follows:

```
|2|3| 4|5| 6|7|~|
```

An indicator called a pointer, shown as a ^, points to the value in the first location of the buffer. Initially, after we’ve keyed in our values and hit [CR] the pointer is at the beginning of the buffer.

```
|2|3| 4|5| 6|7|~|
```

^
After the CR is pressed the first input statement 23 is read and the value is recorded in variable A using the `Read(A)` statement. The pointer is advanced to the beginning of the next input.

```
|2|3|4|5|6|7|
```

The `Writeln('A=',A:2)` is executed and A=23 is printed to the screen.

Next, the input is read starting from where the pointer is positioned and the value is recorded in variable B using the `Read(B)` statement. The pointer is advanced to the beginning of the next input.

```
|2|3|4|5|6|7|
```

The `Writeln('B=',B:2)` is executed and B=45 is printed to the screen.

The process can continue until we reach the END OF LINE marker.

```
|2|3|4|5|6|7|
```

10.2 Readln() and the Buffer.

To see how `Readln()` modifies the buffer consider the following program.

```pascal
PROGRAM IntroBuff2(input,output);
VAR A, B, C:integer;
BEGIN
  Writeln('Type 3 integers on a line, then hit CR');
  Readln(A);
  Writeln('A=',A:2);
  Readln(B);
  Writeln('B=',B:2);
  Readln(C);
  Writeln('C=',C:2)
END.
```

If start with the same input as before the buffer looks like,

```
|2|3|4|5|6|7|
```

Now instead of `Read(A)` we have `Readln(A)`. This time when we hit [CR] the first number 23 is copied into location A. The pointer is then advanced to the next variable but because `Readln()` has only one variable on its input list it copies only one variable in and then clears the buffer and resets the pointer to the beginning, ready for new input.

This means that now at the second step our input buffer is

```
| | | | | | | |
```

Now the program will write the value of A and wait, ever so patiently, for a new value on the next line.

If we’d had `Readln(A,B)`; instead of `Readln(A)`; the buffer would be cleared after the value of B was read in.

11 Summary.

From this lecture you should know:

- How to format Integers, Characters, Strings, Reals, and Booleans.
- How to read in Numbers, such as Real and Integer values, and how to read in Characters.
- What the buffer does and how `Read()` and `Readln()` interact with the input buffer.