So far most of our programming skills have dealt with controlling execution of a program. We now learn a little about how to write programs which use a lot of data.

There are many mechanisms in Pascal for storing data, we will only study **arrays**, leaving the other more advanced structures for another course.

### 1 Arrays... a first look.

Conceptually it is easiest to think of an array as a table of data in memory. The entries to the table, are called the *elements* of the array and they can be accessed by giving their position, known as the *offset* or *array index*.

E.g. If we wanted to use the third element of a table to store the value 3 we’d write something like

```pascal
Table[3] := 2;
```

Later on if we needed to get the value of the third element back for some calculation we could say something like:

```pascal
Writeln(Table[3]);
```

An array is declared in one of two ways. The basic way is as follows:

```pascal
VAR ArrayName:ARRAY[Low .. High] OF SomeType;
```

To declare an array of 5 integers called `Table` we write:

```pascal
VAR Table:ARRAY[1 .. 5] OF Integer;
```

We can easily set the values of elements of the array `Table` by assigning values to their locations. E.g.

```pascal
Table[1] := 31
Table[2] := 1
Table[3] := 41
Table[4] := 5
Table[5] := 9
```

This makes our array look like

**Table:**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>1</td>
<td>41</td>
<td>5</td>
<td>9</td>
<td>Value</td>
</tr>
</tbody>
</table>

We can use the value stored in an array by accessing it using it’s index in the array. For example if we wanted to print the third value we could write:

```pascal
Writeln(Table[3]);
```

The second way to declare array is a little more complicated, but it’s worth learning about. If we need to create more than one array that looks identical to another array in structure we can declare a new **type**.

For example we could define arrays of integers of five elements in the following way.

```pascal
TYPE IntegerArray = ARRAY[1 .. 5] OF Integer;
```
This says, “make the name IntegerArray mean an ARRAY of five integers.” We use the keyword TYPE to create a new type variable for use when declaring variables. Now we can declare a series of arrays of length five and type integer using IntegerArray. e.g.

```pascal
VAR Table1, Table2, Table3 : IntegerArray;
```

makes Table1, Table2, and Table3 arrays of integers five elements long.

## 2 Reading values into an array.

Let’s use the definition of an array given above.

```pascal
TYPE IntegerArray = ARRAY[1..5] OF Integer;
VAR Table: IntegerArray;
```

We can read values into the Table in many ways, perhaps the most natural way is to use a FOR-DO loop.

```pascal
FOR I := 1 TO 5 DO
    read(Table[I]);
```

This reads in an integer from the keyboard five times, assigning the value to the entry Table[I].

If we type in, 23 45 43 323 56 then on pass one through the loop, I := 1 and read sets the value of Table[1], then on the second pass I := 2 and read sets the value of Table[2] and so on...

## 3 Using the values in an array.

We can write out the values from an array using a FOR-DO loop as well,

```pascal
FOR I := 1 TO 5 DO
    Write(Table[I]);
```

which will print out the values at each location of I in turn.

We could even print the numbers out backwards.

```pascal
FOR I := 5 DOWNTO 1 DO
    Write(Table[I]);
```

We could change the values before we print them, this will double the value of every number in the array.

```pascal
FOR I := 1 TO 5 DO
    Table[I] := 2 * Table[I];
```

## 3.1 Averaging your grades.

I have 61 students in my class, suppose I wanted to write a program to work out relative grades for every student.

I could read in the grade for the class, and then work out the average grade. With that average I could work out whether a student is doing badly or doing well.

```pascal
PROGRAM Grader(input, output);
```
CONST Students = 61;

TYPE ClassGrades: Array [1 .. 61] OF Integer;
VAR GradeArray, RelativeGrades: ClassGrades;
  Average: Integer;

PROCEDURE ReadGrades;
VAR Counter: integer;
BEGIN
  FOR Counter := 1 TO Students DO
    BEGIN
      Write('Enter a grade ');
      Readln(GradeArray[Counter]);
    END;
END {ReadGrades}

PROCEDURE AverageGrades(VAR Average: Real);
VAR Counter: integer;
BEGIN
  Sum, Avg: real;
  Sum := 0.0; Avg := 0.0;
  FOR Counter := 1 TO Students DO
    Sum := Sum + GradeArray[Counter];
  Avg := Sum / Students;
END

PROCEDURE SetRelatives(Average: Real);
VAR Counter: integer;
BEGIN
  FOR Counter := 1 TO Students DO
    RelativeGrades[Counter] := GradeArray[Counter] - Average;
END

BEGIN
  ReadGrades;
  (* Works out the average *)
  AverageGrades(Average);
  (* Calculate Relative Grades *)
  SetRelativeGrades(Average);
END;

We should note that we’re using two arrays to perform our calculations here.

4 Arrays of Character values.

We can write a program that reads in characters into an array and then reverses their order.

PROGRAM Reverser(input, output);
CONST Length = 8;
VAR Count;
  MyBuff: Array [1 .. Length] OF Char;
BEGIN
  FOR Count := 1 TO Length DO
    Read(MyBuff[Count]);
  FOR Count := Length DOWNTO 1 DO
    Write(MyBuff[Count]);
END.
5 Different indices.

We can use different indices for arrays, the only restriction is that we use sequences of ordinals so that we can reference elements in the most natural way.

We can define an array using characters as the indices.

```
VAR CharIndexed : Array['A' .. 'Z'] OF Integer;
```

Why would we want to do this... Suppose we wanted to count the frequency of letters in an array, we can write:

```
PROGRAM Frequencies(input, output);
CONST Length = 80;
VAR CharIndexed : Array['A' .. 'Z'] OF Integer;
    Count : integer;
    Ch : Char;
BEGIN
    FOR Count := 1 TO Length DO
    BEGIN
        Read(Ch);
        CASE Ch OF
            'a','A': CharIndexed['A'] := CharIndexed['A'] + 1;
            'b','B': CharIndexed['B'] := CharIndexed['B'] + 1;
            .
            .
        END;
    END;
    FOR Ch := 'A' TO 'Z' DO
        Writeln('There are ', CharIndexed[Ch], Ch,'s');
END.
```

Other ways are to choose different ranges. e.g.

```
VAR TABLE : ARRAY [10 .. 20] OF INTEGER;
```

says the table starts at location 10 and ends at 20. To use it we'd need to pick indices between 10 and 20.