The **WHILE** statement.

In looping so far we’ve only considered situations where we know exactly how many times we need to pass through a loop. This is known as *definite* looping and is implemented using the **FOR-DO** loop statement in PASCAL.

We now consider a new form of looping where we needn’t know how many times to pass through a loop before we execute it. This kind of looping is known as *indefinite looping* and is implemented in PASCAL using either the **WHILE-DO** statement or the **REPEAT-UNTIL** statement.

We will consider the **WHILE** statement which uses conditions, like we had in the **IF-THEN-ELSE** statements, to control the amount of times a loop body is iterated through.

1  **The **WHILE** statement.**

A **WHILE** loop consists of a **WHILE-DO** and the statement or compound statement following it.

More precisely the syntax is:

```pascal
WHILE ConditionalExpression DO 
  Statement (or CompoundStatement);
```

We read the statement above as “**WHILE** Some condition is true **DO** the body of the loop”. Before we re-enter the body of the loop we must do the test again and see if the condition re-evaluates to true.

Here is an example of a piece of code that uses a **WHILE-DO** loop to count the number of characters in a sentence that ends with a period.

```
Period = '.';
Ch:char;
Count:integer;
WHILE Ch <> Period DO 
  BEGIN
    Count := Count + 1;
    read(Ch);
  END;
```

This is different from the **FOR-DO** loop in that there is no limit on how long we can keep looping, the only thing that stops the program from looping is the appearance of a **Period** in the input.

Let’s examine this. If the input sentence were,

```
Stop here. This will be ignored.
```

Now what happens it the loop will execute for each character read in until the period is encountered. When the period is encountered the test in the **WHILE-DO** fails and we go onto the statement that follows the body of the loop, this is called *dropping out of the loop*.

2  **The census problem, again!**

If we go back to our census problem from the first chapter we can finally, in the last weeks of the semester, write a general purpose census program. It looks something like:

```
PROGRAM Census(input,output);
VAR Region:Char;
  N,E,W,S:Integer;
```
BEGIN
N := 0; E := 0; W := 0; S := 0;
Region := 'a' (* Just to get in the loop *)

WHILE Region <> 'Q' DO
BEGIN
Write('Please type in where you’re from:');
Write('N,E,W,S, OR Q to Quit!');
Readln(Region);
CASE Region OF
  'N': N := N + 1;
  'E': E := E + 1;
  'W': W := W + 1;
  'S': S := S + 1;
END;
END;
Write('The results are: ');
Writeln('N =', N, ' E=', E, ' W=', W, ' S=', S);
END.

So in this program everytime the loop gets executed we increment the appropriate counter. If we ever encounter a 'Q' in the input the program drops out of the WHILE-DO loop and executes the next statement in the program.

A subtle point to notice is that we need to have a way to get into the loop for the first time. In the FOR-DO loops we entered the loop because the loop conditions allowed us to do so.

In WHILE-DO loops we need to guarantee that we pass the first test to be able to enter the loop.

3 The REPEAT loop.

Another alternative to the WHILE-DO loop is the REPEAT-UNTIL. REPEAT-UNTIL is essentially the same as the WHILE-DO loop except that the test comes after the body of the loop is executed.

The code above required that we needed some way to guarantee a first pass through the loop to get to the Readln(Region) statement. We did this above using the following trick,

Region := 'a' (* Just to get in the loop *)

but it’s a trick, and is used only to force our program to behave in a certain way.

In general we need to use the constructs that are most natural to the desired flow of control. To overcome situations where we need to execute the body of the loop first we use the REPEAT-UNTIL statement.

The syntax of REPEAT-UNTIL is

REPEAT
  Statement (or CompoundStatement);
UNTIL ConditionalExpression

So we could translate the program above into:

PROGRAM Census(input, output);
VAR Region: Char;
  N, E, W, S: Integer;
BEGIN
  N := 0; E := 0; W := 0; S := 0;
REPEAT
  Write('Please type in where you’re from:');
  Write('N,E,W,S, OR Q to Quit!');
  Readln(Region);

  CASE Region OF
    'N': N := N + 1;
    'E': E := E + 1;
    'W': W := W + 1;
    'S': S := S + 1;
  END;
UNTIL Region = 'Q'

  Write('The results are: ');
  Writeln('N =',N,' E=',E,' W=',W,' S=',S)
END.

Here the REPEAT-UNTIL loop executes the body of the loop at least once, thereby eliminating the unnecessary initialization of the variable used in the test above.

NOTE: For compound statements we don’t need the BEGIN-END statements in a REPEAT-UNTIL loop.

4 Keeping Count in a WHILE-DO loop

We often need to loop indefinitely but would like a way to keep track of how many times we’ve been through the loop. This is easily done, in fact, it’s nothing new to you. All we need to do is initialize a new variable to accumulate the number of times we’ve been through the loop. e.g.

```pascal
Count:integer
Count := 0;
WHILE Condition DO
  BEGIN
    .
    .
    Count := Count + 1;
  END
```

Here as we pass through the loop the Count is incremented. We could put the accumulating line just after the BEGIN.

Here’s a program that averages a set of positive numbers. If we read in a negative number program stops iterating.

```pascal
PROGRAM Average(input,output);
VAR Input, Count, Sum, Average:Real;
BEGIN
  Count := 0.0; Sum := 0.0;
  Read(Input);
  WHILE (Input >= 0.0) DO
    BEGIN
      Count := Count + 1.0;
      Sum := Sum + Input;
      (* Get the input for the next iteration *)
      Write('Type a number ');
      Read(Input);
    END;
  Average := Sum / Count;
  Writeln('Average =', Average)
END
```
There are some things you should notice about this version of the program.

1. We’re using Reals instead of integers, this is convenient, since averages are usually real numbers. The Count is also a Real, if we’d used a FOR-DO loop the count that we usually get from the control variables would have had to be of an ordinal type.

2. We might have been tempted to use a REPEAT-UNTIL loop in our program but that would have caused a problem. We always want to do the test, the first number may well have been a negative number!

3. Because of the need to test that the first number is not a negative number we need to put an additional Read(Input) before the loop so that we can enter the loop if we have legal input. Notice that once inside the loop we need to use the old value of Input before we read in a new value!

5 WHILE-DO is a generalization of FOR-DO

It should be obvious to you by now that WHILE-DO is more general than FOR-DO. In fact we can emulate the behaviour of FOR-DO loops using a WHILE-DO.

The loop,

\[
\text{Count:integer;}
\]
\[
\text{FOR Count := Min TO Max DO}
\]
\[
\text{BEGIN}
\]
\[
\text{Writeln(Count);}
\]
\[
\text{END}
\]

has the same effect as writing,

\[
\text{Count, Min, Max:integer;}
\]
\[
\text{Count := Min;}
\]
\[
\text{WHILE Count <= Max DO}
\]
\[
\text{BEGIN}
\]
\[
\text{Writeln(Count);}
\]
\[
\text{Count := Count + 1;}
\]
\[
\text{END}
\]

and we can make this even more general,

\[
\text{Count, Min, Max, Step:Real;}
\]
\[
\text{Count := Min;}
\]
\[
\text{WHILE Count <= Max DO}
\]
\[
\text{BEGIN}
\]
\[
\text{Writeln(Count);}
\]
\[
\text{Count := Count + Step;}
\]
\[
\text{END}
\]

which will make this loop increment itself by Step each time we pass through the loop.