Lesson 4: Program Flow Control (If Statement, Repeat-Until & For Loop)

In a nuthsell, this lesson will cover:

Up till now, we have based our programs on the most simple control structure: **Sequence**. This means that all of our programs so far were made up of sequential instructions executed one after the other. Basing such programming on such a simple control structure, would not be possible for complex programs to be developed. With only sequential instructions, it won't be possible to enable *decision making* in code or *code repetition*.

Thus, it's now the time to learn the most important branches of programming: the **if statements** - decision making, **for loops** - iterations and the **repeat-until loop** - iterations. These 2 general programming control structures are common in almost every programming language that exists (procedural, event-driven/object oriented programming). Make sure that when you have finished reading this lesson, you have practised them enough before continuing to learn Pascal.

The IF Statement

The **If statement** executes the subsequent statement(s) conditionally. This means that if a condition evaluates to true, then the statement(s) following the if statement are executed, otherwise these statements are skipped and will not be executed. It works like this:

If *<condition evaluates to* ***true****>* ***then*** *<execute these statements>*

OR:

If *<condition evaluates to* ***true****>* ***then*** *<execute these statements>,*
else
<execute these other statements>

Therefore, in Pascal the '**if** statement' should be written using the following syntax:

If <*conditional expression*> **then** <*one line of code*> ... ;

OR:

If <*conditional expression*> **then**
Begin

<*multiple instructions*> ...

End;

{ use **Begin .. End** if more than one instruction is required }

Conditional Expressions

Conditional expressions are used a lot in programming. They are always evaluated to a boolean expression, being either **True** or **False**. In the if statement, when a conditional expression is evaluated to True, the subsequent commands attached to the if statment are executed. A conditional expression can be a simple comparison between a variable and a value. If they are equal, then the expression evaluates to True, False otherwise.

In Pascal, in order to compare two values, you should not use an assignment statement in the **'if'** construct, otherwise the compiler will signal a **syntax error**. A single equal sign is used instead. Let us see how they are used:

Wrong:
If x := 20 **then** x := x + 1; {the underlined character - the colon, must be excluded}

Correct:
If x = 20 **then** x := x + 1; {only an equal sign is used for comparison}

A program is shown below as an example of how the 'if statement' works:

```
Program lesson4_Program1;
Uses  Crt;
Label 1;  { this is used with a goto statement }
Var Sel: String;
    N1,N2, Total : Real;
    YN : Char;  { this is a character variable type, which holds single characters ONLY }

Begin
          1:Clrscr;
          Total := 0;  { always initialise integer/real variables }
          GotoXy(4,3);
          Writeln('1.Addition');
          GotoXy(4,4);
          Writeln('2.Subtraction');
          GotoXy(4,5);
          Writeln('3.Exit');
          GotoXy(6,8);
          Write('Select: ');
          Sel := Readkey;

          If Sel = '1' {condition} Then
          Begin  {more than one statement}
```

```
                    ClrScr;
                    Write('Input No.1:');
                    Readln(N1);
                    Write('Input No.2:');
                    Readln(N2);
                    Total := N1 + N2;
                    Writeln('Addition: ',N1:2:3,' + ',N2:2:3,' = ',Total:2:3);
                    Write('Press any key to continue...');
                    Readkey;
                    Goto 1; { this returns execution back to the beginning of the program (not
recommended) }
          End; { Closing the if statement }

          If Sel = '2' Then { note that here we do not use an assignment statement }
          Begin
                    ClrScr;
                    Write('Input No.1:');
                    Readln(N1);
                    Write('Input No.2:');
                    Readln(N2);
                    Total := N1 - N2;
                    Write('Subtraction: ');
                    Write(N1:2:3,' - ',N2:2:3,' = ',Total:2:3);
                    Write('Press any key to continue...');
                    Readkey;
                    Goto 1;
          End;  { Closing the if statement }

          If Sel = '3' Then
          Begin
                    ClrScr;
                    Write('Are you sure?(Y/N)');
                    YN := Readkey;
                    If YN = 'y' Then Halt; { 1 instruction, so no need of Begin..End }
                    If YN = 'n' Then Goto 1; { the goto statement is not recommended for
frequent use }
          End;
End.
```

This program is demonstrating the use of *If statements* by asking the user to select an arithemtic operation to be applied to two numbers. The Readkey registers a single character instantly and three if statments are used to evaluate the choice of the user. Notice how the variable `Sel` is compared with corresponding values. Since variable `Sel` is a character, then a character should also be compared. The data types of both values on either side should be identical, otherwise the compiler won't allow it.

Also, note that in the above program, the **`Goto`** statement is used. Use of the **`Goto`** statement is strongly disencouraged because it promotes bad programming and goes against structured programming principles. It was used here for example purposes only..

Besides the use of conditional statements in this program, there is something else of interest. We have trimmed floating point/real numbers for the first time. Notice how the line:

```
Writeln('Addition: ',N1:2:3,' + ',N2:2:3,' = ',Total:2:3);
```

has got its Real variables been formatted. `N1:2:3` will represent the floating point value of `N1` with at lease 2 positions and 3 precisions (numbers after the decimal point). It is often very useful to use this formatting when representing Real values, otherwise they will be displayed in scientific format and that is not what you want your users to see unless specifically required.

If ..Then .. Else

As you know, the instructions which follow the if statement won't be executed if the condition does not evaluate to true. We are going to extend the If statement such that at least one branch will be executed, depending on the outcome of the conditional expression. In an **if..then..else** statement, there is at least one set of statements that will be executed.

Let's take a look at the example below:

```
Writeln('Who has discovered the land of America?');
Readln(ans);
If (ans = 'Christopher Colombus') Then
          score := score + 1 { if this does not execute, }
Else
          Writeln('Sorry, you''ve got it wrong!'); { then this executes }
```

Note that if the **'else'** keyword is included with an if statement, then there should be **no semi-colon** before the 'else' keyword; as shown in the example above.

Nested If Statements

The previous program has already shown an example of nested if statements.

```
If Sel = '3' then
Begin
          ClrScr;
          Write('Are you sure?(Y/N)');
          YN := Readkey;
          If YN = 'y' Then HALT; {Nested if statement}
          If YN = 'n' Then Goto 1; {Another Nested if statement}
End;
```

It is usually found in the form:

If <*condition*> **then** { if #1 }

If <*condition*> **then** { if #2 }

code ...

Else
code ... { if #2 }

Else
code ... { if #1 }

A nested if statement is an if statement within another if statement, as shown above. There can be multiple if statements subsequently embedded within each other. It is recommended to use the block statement enclosures (i.e. **Begin .. End**) to avoid confusions when using nested if statements.

The Repeat-Until Loop

This type of loop is used to repeat the execution of a set of instructions for at least one time. It is repeated until the conditional expression evaluates to true, that is, keeps looping while the condition remains false. Once the loop condition becomes true, the loop ends.

The following example, shows the model of the 'repeat-until' loop:

Repeat

...(code)
...(code)
...(code)

Until <*conditional expression*>;

Here's an example:

```
Uses Crt;
Var YN : String;

Begin
          Writeln('Y(YES) or N(NO)?');
          Repeat { repeat the code for at least one time }
                    YN := Readkey;
                    If YN = 'y' Then Halt; { Halt - exit }
                    If YN = 'n' Then Writeln('Why not? Exiting...');
                    Delay(1800); { wait a second plus 800 milliseconds }
          Until (YN = 'y') OR (YN = 'n');
End.
```

See? It's very simple! In the above program, there is a ***boolean operator*** in the 10th line (***or***). This will be described later on, although one can still get the meaning of that expression. The conditional expression caters for both uppercase and lowercase inputs.

The For Loop

The **For loop** executes a set of instructions for a specified number of times. In such kind of loop, a counter is always used to count the number of iterations that have been executed so far. Once they have all been executed, the for loop terminates.

The for loop is in the following form:

• When used with one statement

for <*counter variable*> * := <*initial value*> **to/downto** <*final value*> **do**

{ one line of code }

• If used for more than one statement

for <*counter variable*> * := <*initial value*> **to**/**downto** <*final value*> **do Begin**

{ code... }

{ code... }

End;

* This variable is called the '**loop counter**'.

Now, let us analyse the usage of the for loop. In order to appreciate what this loop do for us, I have created two programs which produce the same output but one uses the for loop and the other does not.

Without **for loop**:

```
Program lesson4_Program2a;
Uses Crt;
Begin
          Writeln('for loop');    { somewhat tedious writing all this! }
          Writeln('for loop');
          Writeln('for loop');
          Writeln('for loop');
          Writeln('for loop');
          Writeln('for loop');
          Writeln('for loop');
          Readln;
End.
```

With **for loop**:

```
Program lesson4_Program2b;
Uses Crt;

Var Counter : Integer; { loop counter declared as integer }

Begin
          For Counter := 1 to 7 do { it's easy and fast! }
                    Writeln('for loop');
          Readln;
End.
```

Note that the two programs above perform the same function, but which programming style is more adequate?

Suppose we have to make a program which displays a small box made of the ASCII characters. We shall obviously choose the characters which mostly resemble the edges and corners of a box.

Without the **for loop**:

```
Program Program3a_lesson4;
Uses Crt;

Begin
          Gotoxy(25,5); Writeln('+');
          Gotoxy(25,6); Writeln('I');
          GotoXy(25,7); Writeln('I');
```

```
          GotoXy(25,8); Writeln('I');
          GotoXy(25,9); Writeln('I');
          GotoXy(25,10); Writeln('I');
          GotoXy(25,11); Writeln('+');
          GotoXy(26,11); Writeln('-');
          GotoXy(27,11); Writeln('-');
          GotoXy(28,11); Writeln('-');
          GotoXy(29,11); Writeln('-');
          GotoXy(30,11); Writeln('-');
          GotoXy(31,11); Writeln('-');
          GotoXy(32,5); Writeln('+');
          Gotoxy(32,6); Writeln('I');
          GotoXy(32,7); Writeln('I');
          GotoXy(32,8); Writeln('I');
          GotoXy(32,9); Writeln('I');
          GotoXy(32,10); Writeln('I');
          GotoXy(32,5); Writeln('+');
          GotoXy(26,5); Writeln('-');
          GotoXy(27,5); Writeln('-');
          GotoXy(28,5); Writeln('-');
          GotoXy(29,5); Writeln('-');
          GotoXy(30,5); Writeln('-');
          GotoXy(31,5); Writeln('-'); { Finally.. Phew! }
          Readln; { wait for user to read }
End.
```

With **for loop**:

```
Program Program3b_lesson4;
Uses Crt;

Var
          Counter : Integer;  { will be used as a loop counter }

Begin

          For Counter := 1 to 5 do
          Begin
                    gotoxy(25, 5 + Counter);
                    Writeln('I');
          End;

          For Counter := 5 Downto 1 do
          Begin  {an example of 'downto' instead of 'to', note the 'gotoxy(_,_)'}
                    gotoxy(32, 11 - Counter);
                    Writeln('I');
          End;

          For Counter := 1 to 6 do
          Begin
                    gotoxy(25 + Counter, 11);
                    Writeln('-');
          End;

          For Counter := 6 Downto 1 do
          Begin
                    gotoxy(32 - Counter, 5);
                    Writeln('-');
          End;

          {--------------The Corners(+)---------------}
          Gotoxy(25,5);
          Writeln('+');
          GotoXy(25,11);
          Writeln('+');
          GotoXy(32,5);
          Writeln('+');
          GotoXy(32,11);
          Writeln('+');
          GotoXy(45,7);
          Write('Made with For Loops :)');
          Readln;
```

```
End.
```

Again, the two programs above perform the same function. However, one can notice the difference between the programming style used in both programs. Once you master the for loop, a lot can be achieved with it!

Nested for loops

A nested for loop is similar to that of nested if statements. A nested for loop is in the followingform:

for <*loop counter*> **:=** <*original value*> **to** <*final value*> **do Begin**

{ code if any.. 'begin' should be included when using multiple statements }

for <*loop counter*> **:=** <*original value*> **to** <*final value*> **do Begin**

{ code.. if more than one statement, include 'begin' in the second for loop }

End;

{ ... more code (if necessary) }

End;

Although nested for loops are rarely used, they can be useful in certain situations but they may cause problems if designed properly.

While-Do Loop

This type of loop is executes the enclosed code ***while*** the condition is ***true***. It is different from the 'Repeat-Until' loop since the loop might not be executed for at least one time. The code works like this:

While <*condition is true*> ***do***

instruction 1;

instruction 2;

instruction 3;

etc...

End*; { If while-do loop starts with a begin statement }*

Example Program using the While-Do loop:

```
Program Lesson4_Program4;
Uses Crt;

Var Ch : Char;

Begin
          Writeln('Press ''q'' to exit...');
          Ch := Readkey;
          While Ch <> 'q' do
          Begin
                    Writeln('Please press ''q'' to exit.');
                    Ch := Readkey;
          End;
End.
```

The program above is quite straightforward. It asks the user to hit the 'q' button to exit. If other keys are pressed it continues repeating until the 'q' key is pressed. Situations like these are often useful in complex programs and using the while loops help us avoid the use of the *goto* statement.