Lesson 7: Procedures and Functions

In a nutshell, this lesson will cover:

- Procedures
  - The Variable Parameter
- Functions

Procedures

Procedures are just like small programs. Sometimes they are called sub-programs. They are an essential part of program and help programmers to avoid repetitions since they promote code re-use. A procedure in Pascal starts with a *begin* and ends with an *end* just like a program;. It can also have its own variables (called *local variables*). Local variables belonging to procedures cannot be used with the main-program. Variables declared in a program are called *global variables*.

To have an exact definition of a procedure, you should compare a program which makes use of code that is repeated over and over again, with another program avoiding the repeated sections by using a procedure, which is called several times. This is demonstrated in the following examples:

```
Program Lesson7_Program1a;
Uses Crt;
Var Counter : Integer;

Begin
          textcolor(green);

          GotoXy(10,5);
          For Counter := 1 to 10 do
          Begin
                    Write(chr(196));
          End;

          GotoXy(10,6);
          For Counter := 1 to 10 do
          Begin
                    Write(chr(196));
          End;

          GotoXy(10,7);
          For Counter := 1 to 10 do
          Begin
                    Write(chr(196));
          End;

          GotoXy(10,10);
          For Counter := 1 to 10 do
          Begin
                    Write(chr(196));
          End;

          Readkey;
End.
```

Now have a look at the next program which uses a procedure:

```
Program Lesson7_Program1;
Uses Crt;

Procedure DrawLine;
{This procedure helps me to avoid the rewriting the for loops}
Var Counter : Integer;

Begin
          textcolor(green);
          For Counter := 1 to 10 do
          Begin
                    Write(chr(196));
          End;
End;

Begin
          GotoXy(10,5);
          DrawLine;
          GotoXy(10,6);
          DrawLine;
          GotoXy(10,7);
          DrawLine;
          GotoXy(10,10);
          DrawLine;
          Readkey;
End.
```

There are some differences between these two programs which are very important to note. These are :

- **Size of the program**

  Not in terms of file size, but lines of code. It is very important for a program to be small and compact. The first program is much larger than the second program.

- **Neatness**

  Adopting a neat style of code writing in a program helps the programmer (and other future debuggers) to cater with future bugs. I think that the first program is cumbersome, whilst the other is not! What do you think?!

- **Repetitions**

  Repetitions in a program can cause unnecessary headaches for a programmer. So procedures are an essential way to avoid repitions in a program. Repeated code which could be transformed into procedures generate unnecessary lines of code!

- **Debugging Efficiency**

  When you are required to debug the program, bugs could be much more easier to find out as the program is sliced into smaller chunks. You may run the program and notice a mistake at a certain point and which is located in a particular procedure/function. It would be much more difficult to find a mistake in a program if it would be one whole piece of code. Do slice your program into smaller chunks, and this needs the planning and design of the whole problem in hand prior starting to code. Coding (or writing up your program) is just one section of the whole software developement process. The whole **Software Developement Lifecycle** is a critical software engineering process

which is divided into different stages facilitating the development of highly complicated software.

Using Procedures with Parameters

Returning back to program Lesson7_Program1b, note the gotoxy statement before the DrawLine procedure; this could also be eliminated so that we can avoid repeating the use of the gotoxy! An effort should always be made to eliminate duplication of code. Hence, we introduce the use of ***parameters*** in our procedure so that we can change the position of the characters within the procedure accordingly.. This is demonstrated in the following program:

```
Program Lesson7_Program2;
Uses Crt;

Procedure DrawLine(X : Integer; Y : Integer);
{ the declaration of the variables in brackets are called parameters }
Var Counter : Integer; { this is called a local variable }
Begin
          GotoXy(X,Y); {here I use the arguments of X and Y}
          textcolor(green);
          For Counter := 1 to 10 do
          Begin
                    Write(chr(196));
          End;
End;

Begin
          DrawLine(10,5);
          DrawLine(10,6);
          DrawLine(10,7);
          DrawLine(10,10);
          Readkey;
End.
```

This program includes a procedure which uses parameters. Every time it is called, the parameters can be variable, so that the position of the line could be changed. This time, we have also eliminated the *gotoxy* statement before every *DrawLine* statement. The numbers in the brackets of the *DrawLine* are the parameters which state the position of the line. When arguments are passed to a procedure, variables are declared within the brackets of the procedure, and must be separated by a semi-colon ";". The variables (known as the parameters) should be used by the procedure only.

The program has been transformed a lot from the original one - duplicated code has been completely eliminated thanks to *parameterized procedures*, code became neater and more readable.

I have written another program which prompts the user to enter his/her favourite text colour and background colour to be used to write text later on in the program.

```
Program Lesson7_Program3;
Uses Crt;
Var
    UName, USurn, UCoun, UMail : String[50];
                              { These var's are global because they are declared in the main
program }
    TxtB, TxtC, i : Integer;
    InfoCor : Boolean;

Procedure EnterUserInfo(TxtCol : SmallInt; TxtBck : SmallInt);
Begin
```

```
          textcolor(TxtCol);
          textbackground(TxtBck);
          ClrScr;
          Write('Your Name: ');
          Readln(UName);
          Write('Your Surname : ');
          Readln(USurn);
          Write('Country : ');
          Readln(UCoun);
          Write('E-Mail Address: ');
          Readln(UMail);
          Write(' Thank you for entering your personal information!!');
          Readkey;
End;

Procedure ConfirmationField(TxtCol : SmallInt; TxtBck : SmallInt);
Var
    YN : Char; { a local variable }

Begin
          textcolor(TxtCol);
          textbackground(TxtBck);
          ClrScr;
          Writeln('Your Name: ',UName);
          Writeln('Your Surname : ',USurn);
          Writeln('Country : ',UCoun);
          Writeln('E-Mail Address: ',UMail);
          Writeln;
          Writeln;
          Writeln('This is a confirmation field. Please verify that');
          Writeln('your information is correct!');
          Writeln;
          Write('Is your personal information all correct? [Y/N] ');
          Repeat
                    YN := Readkey;
                    Case YN Of
                              'N' : InfoCor := False;
                              'Y' : InfoCor := True;
                    End;
          Until (YN = 'N') OR (YN = 'Y');
End;

Begin { main program }
          InfoCor := True;
          ClrScr;
          TextBackground(cyan);
          TextColor(green);
          Write('A list of colours is being displayed...');

          For i := 1 to 16 do
          Begin
                    Case i Of
                              16 : Begin
                                        TextBackGround(white);
                              End;
                    End;
                    textcolor(i);
                    Writeln(i,': This is Colour No.',i);
          End;

          TextBackGround(black);
          TextColor(white);
          Write('Please, put into your mind your favourite colour. ');
          Write('When you are ready press any key...');
          Readkey;
          ClrScr;
          Write('Enter your favourite text colour: (only numbers) ');
          Readln(TxtC);
          Write('Enter your favourite background colour : ');
          Readln(TxtB);
          Writeln;
          Writeln;
          Write('Now, you must enter your personal information. ');
          Write('Hit any key to continue...');
          Readkey;
          ClrScr;
```

```
          EnterUserInfo(TxtC,TxtB);
          ConfirmationField(TxtC,TxtB);

          If InfoCor = False Then
          Repeat
                    Writeln;
                    Writeln('You verified that your information is, for some reason,
incorrect.');
                    Writeln('You are now going to re-enter your correct information. Hit any
key..');
                    Readkey;
                    EnterUserInfo(TxtC,TxtB);
                    ClrScr;
                    ConfirmationField(TxtC,TxtB);
          Until InfoCor = True;
End.
```

The Variable Parameter

We can opt to make some of the parameters of our procedures as reference variables. In this case, data may flow through the variable in both ways. What I am trying to say is that you can pass data and get data through the procedure using a ***variable parameter.*** Here is a declaration of a variable parameter:

Procedure <PROCEDURE_NAME(**Var** *Variable_Name* : ***Type***);>

Syntax is similar to a normal parameter. When using variable parameters, it's like extending the use of a normal parameter so that it can be used to output data.

Here is an example of how to use a variable parameter and its purpose:

```
Program VAR_PARAM_EXAMPLE;

Procedure Square(Index : Integer; Var Result : Integer);
Begin
          Result := Index * Index;
End;

Var
          Res : Integer;

Begin
          Writeln('The square of 5 is: ');
          Square(5, Res);
          Writeln(Res);
End.
```

In the example above, the `Result` variable parameter is used to store the result of the square of the `Index` parameter. After the procedure `Square` has been executed, the `Res` global variable will have the result of 25 since the procedure stores it back in the variable parameter. This way, it can be used in main program to output the result.

Functions

The second type of sub-program is called a ***function***. The only difference from the procedure is that the function always return a value at the end. Note that a procedure cannot return a value. A function start and end in a similar way to that of a procedure. If more than one value is required to be returned by a function, you should make use of the variable parameter. A function can have parameters too. If you change the sub-program from procedure to a function, of the previous program, there will be no difference in the output of the program. Just make sure which one is best when you need to implement a sub-routine. For example, if you don't need to return any values, a procedure can be the best option. On the other hand, if a value should be returned after the sub-routine is executed, a function should be used instead.

The syntax of a function is similar to that of a procedure, but a function needs to declare the return type after its declaration (after the closing bracket). You will see in the next example.

Example of a program using a function is seen below:

```
Program Lesson7_Program4;
Uses Crt;
Var SizeA, sizeB : Real;
          YN : Char;
          unitS : String[2];

Function PythagorasFunc(A: Real; B: Real) : Real; { The pythagoras theorem }
Begin
          PythagorasFunc := SQRT(A * A + B * B);
          { Output: Assign the function name to the value.
          If you forget to assign the function to the value,
          you will get a trash value from memory }
End;

Begin

          Repeat
                    Writeln;
                    Write('Enter the size of side A : ');
                    Readln(sizeA);
                    Write('Enter the size of side B : ');
                    Readln(sizeB);

                    Repeat
                              Write('metres or centimetres? Enter : [m or cm] ');
                              Readln(unitS);
                    Until (unitS = 'm') OR (unitS = 'cm');

                    Writeln(PythagorasFunc(sizeA,sizeB),' ',unitS);
                    Writeln;
                    Write('Repeat? ');
                    YN := Readkey;
          Until (YN IN ['N','n']);

End.
```

SQRT is a math function provided by common library of Pascal. In the pythagoras function above, the result is returned to the function by assigning the value to its function name. This way we can use the function in the main program to store the result of the function in a variable (in this case it has been outputted directly).

Ah..yes, we won't conclude this lesson without discussing an off-topic introduction of another Pascal programming construct. You might have noticed that the condition in this Repeat..Until loop has used an interesting construct: **Sets**. Pascal provides a powerful container values checking mechanism by using Sets. As you can see in the example above, the code keeps looping until YN becomes either 'N' or 'n'. Interesting no?!