Lesson 1: The First Few Steps in Pascal Programming

In a program, you must always obey the rules of the language, in our case, the Pascal language. A natural language has its own grammar rules, spelling and sentence construction. The Pascal programming language is a high level language that has its own syntax rules and grammar rules. As you go along with the lessons, you must note what you can do and what you cannot do in writing a Pascal program. A very simple program is shown below:

```
Program Lesson1_Program1;
Begin
         Write('Hello World. Prepare to learn PASCAL!!');
         Readln;
End.
```

The program is written only to display the message : 'Hello World. Prepare to learn PASCAL!!' - an introductory message that is displayed to you whenever you are going to learn a new programming language. This is simply shown on the screen. So, to display any message on the screen, you should use 'write' (or 'writeln'). The 'readln' procedure, here is used to *pause* the program and waits until the user hits the return key. If 'readln' is removed from that line, then the message is displayed on the screen without giving any chance for the user to read it and exits!

Try running this program with and without the 'readln' procedure and notice the difference.

Now, look at this:

```
Program Lesson1_Program2;begin
Write('Hello World. Prepare to learn PASCAL!!');Readln;End.
```

This program also runs perfectly as the previous one. The only difference is: *neatness* and *friendliness*. This first program is, what is commonly referred to in programming, as 'indented'. Indentation is a must in writing programs as it makes it easier to read ie. neater. Indentation also helps with debugging and code presentation. You will note how I indent programs. A program in Pascal always starts by the reserved word 'Program' following the title of the program. There are various restrictions on how to write this statement. Below is a simple example of a small program. (Remember: you can copy and paste the program in a text file, save the text file as *filename.pas* and open it with your compiler (such as

Free Pascal). The .pas extension is required.) In the following program, the computer must prompt the user to enter a number, then the latter is added to the second number input by the user.

```
Program Lesson1_Program3;
Var
    Num1, Num2, Sum : Integer;

Begin {no semicolon}
         Write('Input number 1:');
         Readln(Num1);
         Writeln('Input number 2:');
         Readln(Num2);
         Sum := Num1 + Num2; {addition}
         Writeln(Sum);
         Readln;
End.
```

Now we must take a look at the program. A program in Pascal starts with the reserved word '**Program**' (although it is not explicitly required) and ends with '**End**', following a full stop (this is required though). A full-stop is never used within the program, except when dealing with records (later topics) and at the end of the program as seen in the example above.

The '**Var**' keyword is used to introduce variables in a program to be used later on as temporary data storage elements. The variable names 'Num1', 'Num2' and 'Sum' in the program are data placeholders which will store whole numbers, not real/floating-point numbers (in fact, during the execution of the program, a runtime error may occur if a decimal number is input). As you can see in the example above, these variables are declared as **Integers**. The data type 'Integer' means any whole number, i.e. a number which is not a decimal number but can be either a positive or a negative number. The Pascal Integer data type ranges from -32768 to 32767. So values which are not within the specified range cannot be stored by an integer type. There are other types which are wider in range, but for now the integer type is enough to hold up our values. The variables 'Num1', 'Num2' and 'Sum' are identifiers which are not reserved words, but can be used as our variables in the program to store data in them. They could be changed more than once. Moreover, we could have used 'number1', 'number2' and 'totalsum' (note that there must be no spaces within the variables), instead of 'Num1', 'Num2' and 'Sum', respectively. As you can see, it is much better to shorten the variables than writing long words, such as 'variable_number1' but we still need give them a meaningful name to remind us of its storage purpose.

After declaring all the variables which are required to be used later in the program, the main program always starts with the reserved word '**Begin**'. Without this word, the compiler will display a diagnostic (error message). In the program above, both of the two types of 'write' are used. These are '**write**' and '**writeln**'. Both has the same function, except that the '**write**' function, does not proceed to the following line when writing a statement. If you run this program, you will notice the difference between them. When using these two terms, any message that will be typed in between the **brackets** and the **inverted commas** '**(' ')**', is **displayed on the screen**. However, if a variable is used instead of a message, **without** using the inverted commas, the CPU will display the stored variable in the memory, on the screen. In line 9, the CPU will **not** display 'Sum' on the screen, but the stored number in the memory. Another important thing which must be noticed is the **semi-colon** (**;**). The semicolon is used after each statement in the program, except those that you will learn later. However, in the example above, there isn't a semicolon after a '**begin**' statement. This is because the flow of the program has just started and must not be stopped by a ';'.

The text in between the braces ({ }) are called **comments** or **in-line documentation**. I guess you consider the comments to be 'unnecessary': This is wrong! Comments are very useful in describing complicated tasks and functions. In my experiences, I have encountered many problems, like for instance when having a break from writing a program for a long time, and then resuming again after a long period! Practically, I've spent a long time trying to understand what I have done previously (understand my own code, let alone other programmers try to understand my code). Comments within the braces are **not** read or compiled by the compiler/interpreter. I will also be using lots of comments along the lessons to explain my code to you!

The '**readln**' procedure enables the user to input numbers or text only i.e.: using the keyboard. But in our case '**readln**' is used to input numbers only (*letters are still accepted during the program but in this case it will cause a run-time error because it is not the type of input we want*) and store them in the variables '**Num1**' and '**Num2**'. This is because both variables are assigned to as integers, and integer variables do not store strings. A run-time error is detected by the **OS (Operating System; ex. Windows** or **Linux)** if something goes wrong with the input. Later in this tutorial, you will also learn how to catch input and output exceptions - unexpected runtime errors.

One last note on errors: there are 2 main types of errors, namely - **Runtime Errors** and **Compilation Errors**. Runtime errors are those which occur unexpectedly during the execution of the program, whereas a Compilation error is one which is detected during the compilation process. Note that a decimal number is also considered as a wrong input; a decimal number must not be input, since it is a real number (more on this later).

After the prompts and inputs by the user, an addition follows. i.e.

```
Sum := Num1 + Num2;
```

The result of the above statement is the addition of the values stored in variables '**Num1**' and '**Num2**'. The important thing that you should know is that one cannot make the same statement as follows:

```
Num1 + Num2 := Sum;
```

This is another **syntax error** and would not be allowed by the compiler. Remember that transfer of information is from right to left and not from left to right. So keep in mind not to make such a mistake. The '**:=**' symbol is used in an **assignment statement** and will be discussed later on.