Lesson 10: Strings & String Manipulation

In a nutshell, this lesson will cover:

Introduction to Strings

Maybe you already know what a `String` variable is from previous program examples that you have read and tried, but you don't know what operations and what functions one can apply to them and how can they be manipulated in order to obtain another form of string.

In this lesson we will cover some important functions that the Pascal programming language has got to offer in order to cater for string operations far more easier to use than you think.

Let's jump into the strings staff straight away. In order to understand strings, one has to keep in mind that a string is made up of an array of characters. The string data type is an in-built data type that is an array of 256 characters (`Type String = Packed Array[0..255] of Char`). When stored in memory, the processor should know where the string starts and where it finishes. In order to know where the string finishes, in Pascal, the 0th element of a string is defined as the length of the string. So, if you try to access character 0 of a string, the number of characters stored in that array is returned, thus letting the processor to know where the string finishes.

The following example shows some simple string operations.

```
Var
    myString : String;

Begin
         myString := 'Hey! How are you?';
         Writeln('The length of the string is ', byte(myString[0]));
         Write(myString[byte(myString[0])]);
         Write(' is the last character.');
End.
```

Note that in the previous program I have used an automatic data-type conversion, normally referred to as data ***type-casting***. When type-casting from one data type to another, all that is happening is simply a conversion from one data type to another based on the data type in subject and the wrapping data type to which the old data type is being converted. In our case, the array variable `myString` has a special 0th element storing the number of characters in that array in string format. This value is an ascii value, so trying to display it without converting it into a number, the alternative ascii character is displayed. So you have to change the character value to an ordinary number by wrapping the variable by another data type, in our case a **byte** data type (why use **integer**? - its all waste of memory and memory consumption).

Note that `myString[byte(myString[0])]` without having data-type casting around `myString[0]` i.e `byte(..)` will lead to a compiler error because `myString[0]` is a character and not an integer.

Note that to retrieve the length of a string (ie. the number of characters in a string) there is no need to use the method shown in the above example because its more complicated than necessary. Instead, there is the simpler function, **`Length(..)`** which will return the number of characters of the string in subject. Eg. `numOfChars := Length(myString);`

Useful String Functions

There are some basic Pascal functions that has to do with string operations and so there is no need to write them yourself, and they are of course quite useful. I will explain all the string functions by describing what they do and a simple example of each.

Function Pos(SubString : **String**; S : **String**) : **Byte**;

Description

This function will search for the string ***SubString*** within the string ***S***. If the sub-string is not found, then the function would return 0. If on the other hand the sub-string is found, then the index integer value of the first character of the main string that matches the character of the sub-string is returned.

```
Var
        S : String;

Begin
        S := 'Hey there! How are you?';
        Write('The word "How" is found at char index ');
        Writeln(Pos('How', S));
        If Pos('Why', S) <= 0 Then
                Writeln('"Why" is not found.');
End.
```

Function Copy(S : **String**; Index : **Integer**; Count : **Integer**) : **String**;

Description

This function will copy some characters from string ***S*** starting from character

index ***Index*** and copies as much as ***Count***. The copied string is then returned.

```
Var
          S : String;

Begin
          S := 'Hey there! How are you?';
          S := Copy(S, 5, 6); { 'there!' }
          Write(S);
End.
```

Procedure Delete(**var** S : **String**; Index : **Integer**; Count : **Integer**);

Description

Deletes a specified number of characters from the string ***S***. The starting position of deletion is from character index ***Index.*** The number of characters that will be deleted is specified through ***Count***. The new string is passed back through the variable parameter ***S***.

```
Var
          S : String;

Begin
          S := 'Hey Max! How are you?';
          Delete(S, 4, 4); { 'Hey! How are you?' }
          Write(S);
End.
```

Procedure Insert(**var** S : **String**; Source : **String**; Index : **Integer**);

Description

This function will insert a string of any length into a source string starting from an index character. The string ***S*** will be inserted into string ***Source*** starting from the index character ***Index***. No characters will be deleted from the string except if the resulting string is longer than 255 characters. In this case, the trailing characters will be truncated to fit a 255-character string.

```
Var
          S : String;

Begin
          S := 'Hey! How are you?';
          Insert(' Max', S, 4);
          Write(S);
  { 'Hey Max! How are you?' }
End.
```

Function Concat(s1 [, s2, s3...sn] : **String**) : **String**;

Description

Concatenates 2 or more strings depending how long is the argument expression. Try to make sure not to exceed the limit of 255 characters when concatening strings as it will result in truncation. This function can also be obtained by using the plus (+) operator between strings that need to be concatenated.

```
Var
          S1, S2 : String;

Begin
          S1 := 'Hey!';
          S2 := ' How are you?';
          Write(Concat(S1, S2)); { 'Hey! How are you?' }
End.
```

is the same

```
Var
          S1, S2 : String;

Begin
          S1 := 'Hey!';
          S2 := ' How are you?';
          Write(S1 + S2); { 'Hey! How are you?' }
End.
```

Function UpCase(C : **Char**) : **Char**; OR **Function** UpCase(S : **String**) : **String**;

Description

Converts the character ***C*** to uppercase and returned. If the character is already in uppercase form or the character is not within the range of the lower case alphabet, then it is left as is.

```
Var
   S : String;
   i : Integer;

Begin
          S := 'Hey! How are you?';
          For i := 1 to length(S) do
                    S[i] := UpCase(S[i]);
          Write(S); { 'HEY! HOW ARE YOU?' }
End.
```

Although we have used a for loop to convert all characters in a String to upper-case, the function UpCase() can also accept a String and will do the tough job for you i.e. converts all characters to upper-case at once.

```
Var
   S : String;

Begin
          S := 'Hey! How are you?';
          Write(UpCase(S)); { 'HEY! HOW ARE YOU?' }
End.
```

The opposite of `UpCase(..)` is `Lowercase(S)` which does exactly the opposite of the `UpCase(..)` - converts upper case characters to lowercase in a String.

Procedure Str(Val : **Integer / LongInt / Real**; **var** S : **String**);

Description

Converts an integer or a decimal value to a string. The value parameter ***Val*** is converted into a string and passed through the variable paramter ***S***.

```
Var
          S : String;
          i : Real;

Begin
          i := -0.563;
          Str(i, S);
          Write(S);
End.
```

Procedure Val(S : **String**; **var** Val; Code : **Integer**);

Description

Converts a string to its corresponding numeric value. The string paramater ***S*** is converted into a numeric value and passed back through the variable paramter ***Val***. If the string to be converted is not a correct numeric value, an error occurs and is returned via ***Code***. If the conversion is correct then Code is 0.

```
Var
          S     : String;
          error : Integer;
          R     : Real;

Begin
          S := '-0.563';
          Val(S, R, error);
          If error > 0 Then
          Write('Error in conversion.')
          Else
                    Write(R);
End.
```