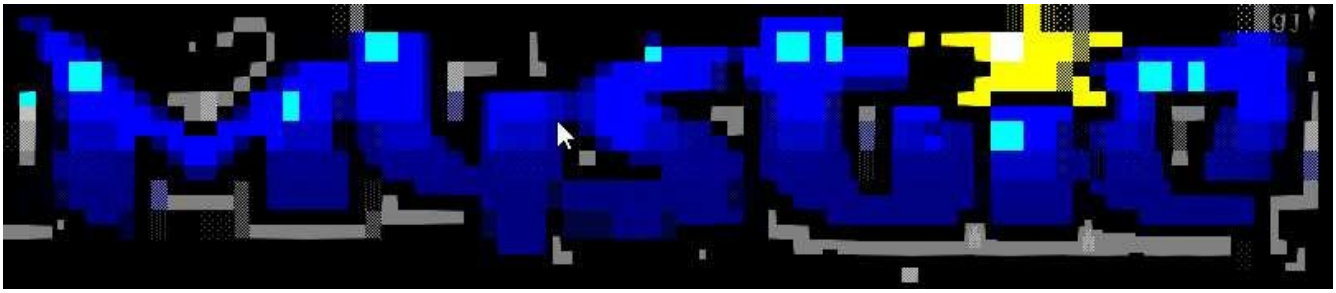

MPL Training
Lesson 17
Black Panther

-=-=-=-=-=-=-=-=-

It has been suggested that we take a closer look at the syntax of MPL, and how to properly format your code, so that it both compiles and is readable by others.

Let’s first take a look at how the current documentation shows the different functions available to us.

Random(Max:Word):Word

The first word indicates the name of the function, in this case ‘Random’. What do you think this function gives us? ;)

Inside the parentheses, are the parameters this function needs. In this case, it requires one parameter, which is a word. As we know from previous lessons, a word is a numerical variable from 0 to 65535.

After the parentheses, there is a colon, followed by ‘Word’. This is telling us that this function takes a parameter of a ‘Word’, and will give us a ‘Word’ back. So, in this case, we tell it the max number to generate a random number, and it will return the random number to us. Here’s an example:

```
RandNumber:=Random(100):Word
WriteLn(RandNumber)
```

If we run this, the function would return a random number between 0 and 100, and store it in the variable ‘RandNumber’. The ‘WriteLn’ will print the random number to the screen.

```
Var
  RanNum : Word=100
WriteLn(Random(RanNum))
```

This call to the function would do the same thing.

Let’s take a look at another one that seems to trip people up. If you look in the MPL documentation for Input, you will see a whole long line of parameters which can be a bit confusing.

Function Input(Field: Byte, Max: Byte, Mode, Byte, Default: String): String

Let’s break these down and take a closer look at each one.

Field: This is the visible input field the user will see on the screen. When you first install Mystic, you will notice the input fields for the username and password are blue. The Field length is where the length of that input box is entered.

Max: This is the maximum number of characters the user is allowed to enter at this prompt. This number should be the same, or slightly smaller than your variable definition. If your variable is set as Var name[30], the max for the input should either be 30 or 29. If this number is higher than the Field length, the users input will scroll as they type it in.

Mode: This is where you’re telling your MPL, and Mystic, what type of input to expect. Here is the list of different modes:

1 : Standard input. All characters allowed.
2 : Upper case input. Allows all characters, but will convert any lower case letters into upper case.
3 : Proper input. Allows all characters, but will convert the first letter in each word to an upper case letter.
4 : Phone input. Allows only numbers and will pre-format them using the USA-style phone numbers. IE: XXX-XXX-XXXX. Note that the length of this input should always be 12, as that is the length of the USA phone number format.
5 : Date input. Allows only numbers and will pre-format them using the date format (ie XX/XX/XX) that is currently selected by the user. NOTE: The date input will always return the date in the MM/DD/YY format, regardless of what format the user has selected. For example, if the user has selected the DD/MM/YY format, Input will expect the user to enter the date in that format, but will then convert it to MM/DD/YY when it returns the date back to the MPE program.
6 : Password input. Allows all characters, but will convert any lower case letters into upper case. The character that is typed is NOT echoed to the screen. Instead, it is replaced by the * character so that what they have entered will not be shown on the screen.
7: Lowercase Allows characters but will be lowercase.
8: User Defined Input (from system config)
9: Standard Input with no CR
10: Number Input numbers only and +-

This allows you a lot of control over what the user can enter into the prompt. If you only want numbers entered, you would use 10.

Lastly, we have the Default string. This is text the user will see in the input field before they start entering anything. Once they start typing, this text will disappear. So, if you have a form for the user to fill out, as in the new-user application, you could just use the Input function, and indicate in the default text what each of the input fields are for.

Here’s a couple of examples of how this function is used.

```
Var
  Str : String

Write(‘Enter some text’)
Str:=Input(30,30,1,’’)
```

This would display ‘Enter some text’ on the screen, and then give the user a 30 character field to enter their text. As it is mode ‘1’, it will accept any characters the user types in. There is also no default text, so the field will appear blank to the user.

```
Var
  Str : String

Str:=Input(20,20,3,’Enter your name’)
```

This will display an input field for the user, with the text ‘Enter your name’ in it. Once the user starts typing their name, that text will disappear. This is also using mode ‘3’, which means it will have proper capitalization. If the user types in ‘jon smith’, it will appear in the input box, and saved as, ‘Jon Smith’.

-=-=-=-=-=-=-=-=-

When writing lines of code, keep in mind that you can have multiple function calls within the same statement. For example:

```
   AppendText(CallText,PadRt(UserSave.Handle,20,' ')+PadRt(Int2Str(UserSave.Node),5,' ')
+PadRt(StripMCI(UserSave.City),25,' ')+PadRt(DateStr(UserSave.LastCall,1),10,' ')
+PadRt(TimeStr(UserSave.LastCall,False),10,' ')+PadRt(UserSave.CServerName,7,' '))
```

This is an actual line from the new Who’s Online MPL that I’ve been working on. Don’t be intimidated by the length of this line, as it’s not as bad as it looks. Let’s break this line down to more manageable code.

AppendText – This function will append the text given to it, into an existing text file. Anything within the parentheses after it will be included.

PadRt – This will take the given text, and display it within the number of characters indicated in the parameters. If you notice the first ‘PadRt’ call:

```
PadRt(UserSave.Handle,20,’ ‘)
```

What this is doing, is printing the value stored in the record variable UserSave.Handle. It will use 20 spaces for this field, and if needed, will append spaces, ‘ ‘, to the right of the text in order to fill the 20 spaces.

The next PadRt function call has a bit more going on in it.

PadRt(Int2Str(UserSave.Node),5,' ')

Now, when Pascal, or MPL, is compiling this line, it will start from the innermost set of parentheses, and work its way outward. So, it starts with the record variable 'UserSave.Node'. There is a function to the left of the parentheses, 'Int2Str', that is called with the record variable as a parameter. (The Int2Str will typecast the Integer to a String, as we can only directly write strings to the screen or file) Once that is complete, MPL will see this as a regular PadRt function call, with the parameters listed.

One more example from this line before we move on.

PadRt(StripMCI(UserSave.City),25,' ')

This is similar to the previous 'PadRt' function call. In this one we are calling the StripMCI function, which will remove any MCI codes contained within the record variable 'UserSave.City'.

So, as you can see, nesting function calls can be very useful when writing your code. It can also make things look more complicated than it really is. Just break these lines down, and look at each function call on their own, and you will be able to see what is happening.

This lesson will be a bit shorter than normal, as the next one will probably be longer than normal.