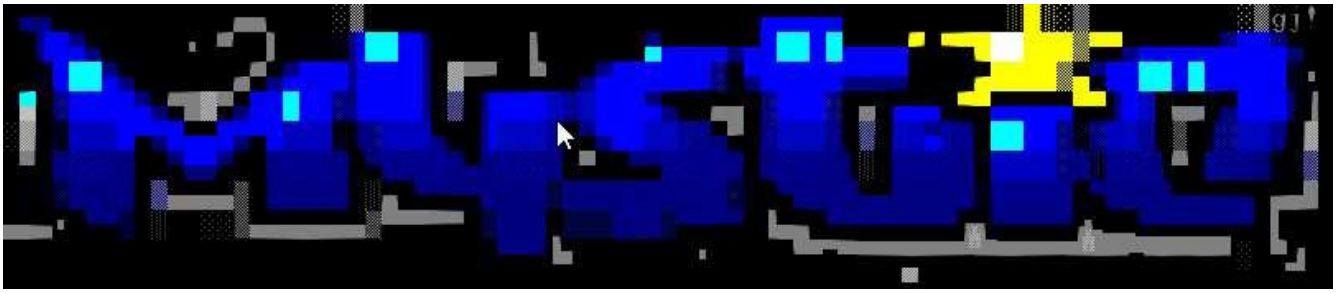

MPL Training
Lesson 15
Black Panther

Some Program Setup

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

This is something I probably should have gone over in an earlier lesson, but to be honest with you, I didn't think about it. It's become second nature for me.

When writing an MPL program, or a lot of other programming languages which follow the same rules, there is a need to have things in a particular order. For example, the first think the MPL compiler will look for in your code, is the 'Uses' statement. It needs to know what information needs to be 'included' with your program. If you will be working with user information, you will need to add 'Users', etc. We've already covered that.

The next thing the compiler looks for, is any 'Types' you will have. These are going to be any records you will be using. There are some other types of types, but you probably won't need to worry about those yet. I'm not even sure if the MPL compiler supports them…

Then you will have any global variables for your program. Just a reminder, keep these to a minimum, as most variables are only needed for one, maybe two, procedures/functions in your program. If that's the case, keep them local.

It's after this where things can get a bit tricky. I'll try to explain this as best I can. If you have any questions about this, please ask.

First of all, we all know the, usually, small procedure at the bottom of the code. It's the one that doesn't have either procedure or function listed, or a name. This is referred to as your 'Main' procedure. This is important to know, as there may be a test later. :) Just kidding, I wouldn't do that to you. But, this is the most important procedure in your program. We'll find you why as we go through this lesson.

When your program is compiling, it will read everything starting from the beginning, top, of your code. If, while it's compiling, it finds a call to a procedure that it hasn't seen yet, it will give you an error.

Here is a snippet of code to give you an example:

```
 1  Uses Cfg
 2
 3  Function addnumbers(a,b:Integer):Integer
 4  Begin
 5    addnumbers:=a+b
 6  End
 7
 8  Function subtractnumbers(a,b:Integer):Integer
 9  Begin
10    subtractnumbers:=a-b
11  End
12
13  Function multiplynumbers(a,b:Integer):Integer
14  Begin
15    multiplynumbers:=a*b
16  End
17
18  Begin
19    ClrScr
20    WriteLn('1 + 1 = '+Int2Str(addnumbers(1,1)))
21    WriteLn('2 - 1 = '+Int2Str(subtractnumbers(2,1)))
22    WriteLn('2 x 2 = '+Int2Str(multiplynumbers(2,2)))
23  End
24
```

Remember this program? If you notice the procedure at the bottom does not have the word ‘Procedure’, ‘Function’, or a name listed. It just has a ‘Begin’ and ‘End’. That is your Main procedure.

Our compiler will read everything until it gets to this procedure, and then start putting your program together. Of course this is overly simplified, as the compiler really is doing a lot more than that, but once it gets to the Main procedure, it does not expect to find anything else afterwords. The Main procedure **NEEDS** to be at the bottom of your code.

Now, you’ll notice there are other functions in this program. Let me change the code a little, and we’ll see what happens.

```
 1  Uses Cfg
 2
 3  Function addandsubtract(a,b:Integer):Integer
 4  Begin
 5    addandsubtract:=addnumbers(a,b)*subtractnumbers(a,b)
 6  End
 7
 8  Function addnumbers(a,b:Integer):Integer
 9  Begin
10    addnumbers:=a+b
11  End
12
13  Function subtractnumbers(a,b:Integer):Integer
14  Begin
15    subtractnumbers:=a-b
16  End
17
18  Function multiplynumbers(a,b:Integer):Integer
19  Begin
20    multiplynumbers:=a*b
21  End
22
23  Begin
24    ClrScr
25    WriteLn('1 + 1 = '+Int2Str(addnumbers(1,1)))
26    WriteLn('2 - 1 = '+Int2Str(subtractnumbers(2,1)))
27    WriteLn('2 x 2 = '+Int2Str(multiplynumbers(2,2)))
28  End
29
```

Alright, what I did is added another function to the top of the code, that will take the two numbers, add them together, then subtract those two numbers, and multiply them together. I know, not a well needed function, as I don't know why you would want to do that, but it works for this example.

The question is, will this code compile?

```
[dan@dan-home mystic]$ ./mplc themes/default/scripts/tutorial.mps

Mystic MPL Compiler v1.12 A47 (Linux/64 2020/12/04 01:37:07)
Copyright (C) 1997-2020 By James Coyle.  All Rights Reserved.

Compiling: themes/default/scripts/tutorial.mps                          10%

Error in themes/default/scripts/tutorial.mps (Line 5, Col 29): Unknown identifie
r: addnumbers

ERRORS:
  Error in themes/default/scripts/tutorial.mps (Line 5, Col 29): Unknown identi
  fier: addnumbers
Results: 1 files, 1 errors, 1 directories, 0.00 seconds

[dan@dan-home mystic]$
```

Uh oh… What does that mean? ‘Unknown identifier: addnumbers’?

Well, if you notice the ‘addnumbers’ function is below our new ‘addandsubtract’ function. While the compiler was reading our code, it noticed the call to the ‘addnumbers’ function, but it hadn’t seen anything about that function in our code yet.

For this reason, we need to be very careful which order we put out procedures and functions in our code. In order for this to compile, we would need to move our new ‘addandsubtract’ function so it is below both the ‘addnumbers’ and the ‘subtractnumbers’ functions.

```
 1  Uses Cfg
 2
 3  Function addnumbers(a,b:Integer):Integer
 4  Begin
 5    addnumbers:=a+b
 6  End
 7
 8  Function subtractnumbers(a,b:Integer):Integer
 9  Begin
10    subtractnumbers:=a-b
11  End
12
13  Function addandsubtract(a,b:Integer):Integer
14  Begin
15    addandsubtract:=addnumbers(a,b)*subtractnumbers(a,b)
16  End
17
18  Function multiplynumbers(a,b:Integer):Integer
19  Begin
20    multiplynumbers:=a*b
21  End
22
23  Begin
24    ClrScr
25    WriteLn('1 + 1 = '+Int2Str(addnumbers(1,1)))
26    WriteLn('2 - 1 = '+Int2Str(subtractnumbers(2,1)))
27    WriteLn('2 x 2 = '+Int2Str(multiplynumbers(2,2)))
28  End
29
```

There we go. Now our new function is listed underneath the other two. It should compile now.

```
[dan@dan-home mystic]$ ./mplc themes/default/scripts/tutorial.mps

Mystic MPL Compiler v1.12 A47 (Linux/64 2020/12/04 01:37:07)
Copyright (C) 1997-2020 By James Coyle.  All Rights Reserved.

Compiling: themes/default/scripts/tutorial.mps                    [OK]

Results: 1 files, 0 errors, 1 directories, 0.00 seconds

[dan@dan-home mystic]$
```

That looks better.

In most of the MPL programs that we will be working on, this won't be a huge issue. If you look at a large program, where some functions are used many times, and are also dependent on other functions, it can be a pain.

I'll reference my TDTA MPL game that I wrote awhile back. There are a lot of functions that are used throughout the game, and are needed at different times. Something like the function which writes data to a log file for example. These functions/procedures need to be near the top of the code, so they are listed before any calls to that function/procedure.

In summary, if you start reading your code from the top, you should always find the functions and procedures listed before any calls to them. There are ways around this in Pascal, but I'm not sure if those have been implemented in the MPL compiler, so I'm not even going to mention them here.

I stated earlier this is something that also happens in other languages. I know that in C and C++ this is also the case. Again, there are ways around it, but it still needs to be in the forefront of your mind as your typing in your code.

As this is something that I struggled with early on, I wanted to share some explanation as to how things should be set up, and why. I hope I explained this so you could understand it. It's one of those things that is easy to understand, but can be difficult to explain… Let me know if you have any questions.