MPL Training
Lesson 16
Black Panther

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

I was just looking over some of my code in the MTAFile program, and noticed a section that I thought I spend a little time talking about here.

While this is something that you may not use much in MPL programs, it is good to have a basic understanding of for future endeavors.

Let’s say you have a bunch of records that you would like to display the information of, onto the screen. But, you want it to be organized. You decide you’d like to sort that list of records so they are in alphabetical order. Doesn’t that sound daunting… ;)

It’s actually not too difficult, and can be done with a loop inside of a loop. But, before we start getting all loopy, let’s take a look at how we’re going to do this. I’ll make a simple graphic here. (I hope)

```
----------   ----------   ----------   ----------
|    4   |   |    2   |   |   1   |    |    3   |
|        |   |        |   |       |    |        |
----------   ----------   ----------    ---------
```

(Ok, so this is why I do the coding, and not the graphics…)

So, we have four boxes, each with a number on them. Notice how they’re not in any order. (and ugly) Let’s take a look at how we can go about putting them in numerical order.

We’ll start a loop, which will take the value of the first box, in our case 4, and compare it with the value of the second box. If the number in the second box is smaller, we will exchange the two boxes (records).

```
----------   ----------   ----------   ----------
|    2   |   |    4   |   |   1   |    |    3   |
|        |   |        |   |       |    |        |
----------   ----------   ----------    ---------
```

Then the loop will look at the value of the second box, 4, and compare it with the third box, 1. Since 1 is less than 4, it will switch them.

```
----------   ----------   ----------   ----------
|    2   |   |    1   |   |   4    |   |   3    |
|        |   |        |   |        |   |        |
----------   ----------   ----------    ---------
```

Then it will compare boxes 3 and 4, and change them if needed.

```
----------   ----------   ----------   ----------
|    2   |   |    1   |   |   3    |   |   4    |
|        |   |        |   |        |   |        |
----------   ----------   ----------    ---------
```

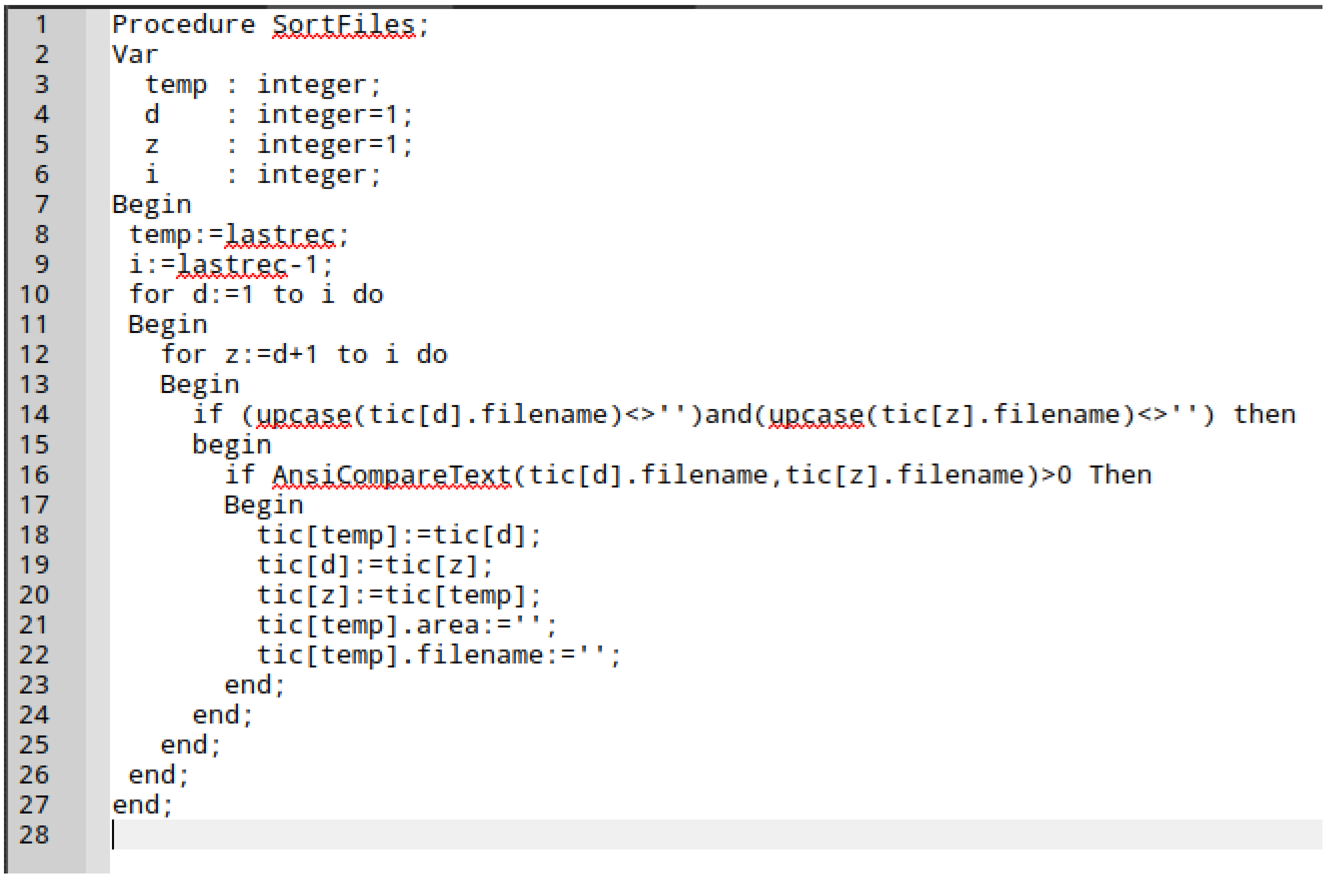
Now, we have nothing to compare the forth box to, so the loop will start the process over again. It will compare boxes 1 and 2, and change them if needed.

```
----------   ----------   ----------   ----------
|    1   |   |    2   |   |   3    |   |   4    |
|        |   |        |   |        |   |        |
----------   ----------   ----------    ---------
```

It looks like we have it sorted. But, the code doesn't know that, so it will continue to check and compare the values a total of 4 times. (due to there being 4 boxes, or records, it iterates that many times through the loops)

This type of sorting is the easiest to understand, and to program. It's referred to as a Bubble Sort. The bubble sort does a pretty good job and getting smaller lists into order. However, in larger lists, it becomes very ineffective, and should be avoided.

The code really isn't too difficult, or too long, to grasp. Here is an example of actual code used in my MTAFile program that is sorting the records by the filename.



```
1  Procedure SortFiles;
2  Var
3    temp : integer;
4    d    : integer=1;
5    z    : integer=1;
6    i    : integer;
7  Begin
8   temp:=lastrec;
9   i:=lastrec-1;
10  for d:=1 to i do
11  Begin
12    for z:=d+1 to i do
13    Begin
14      if (upcase(tic[d].filename)<>'')and(upcase(tic[z].filename)<>'') then
15      begin
16        if AnsiCompareText(tic[d].filename,tic[z].filename)>0 Then
17        Begin
18          tic[temp]:=tic[d];
19          tic[d]:=tic[z];
20          tic[z]:=tic[temp];
21          tic[temp].area:='';
22          tic[temp].filename:='';
23        end;
24      end;
25    end;
26  end;
27 end;
28
```

I’ll just do a brief run-through of what this procedure is doing. Keep in mind, that in order to exchange records when you switch them, you will need to create an extra record. If you notice on lines 18 through 20, we copy one record to temp, copy the other over the first, and then copy temp over the second. That’s the most effective way to ‘swap’ records.

First, we just set up some integers that will be doing our counting for us. I know, I shouldn’t have used just letters for this. I was relatively new to programming when I did this, and almost everything was just a letter variable name…

The ‘lastrec’ variable holds the record number for the last record. So, if we have a total of 100 records, that variable will hold the number 101. (don’t ask, it’s the way I wrote it and it works…) We then set our ‘temp’ record to that number, 101, and the variable ‘i’ is set to 100.

Our first For/Do loop will start at record 1, and go up to 100.

The second For/Do loop starts at ‘d+1’, so it will start at 2, and go up to 100.

Then there is an If/Then statement, checking to see if the filename field in the record is equal to null, or in other words, blank. (When I did duplicate checking in this program, it would blank out the filename field when a duplicate was found)

The next If/Then statement compares the text in the filename field in the two records. This ‘AnsiCompareText’ function is only for pascal, and is not in MPL, but you’ll get the idea. It will return ‘0’ if they are equal, greater than 0 if the first one is larger, and less than 0 if the second one is larger.

Now, when I refer to the text as being larger than the other, what the function does, is look at the ascii codes for all of the characters within the text field.

So, if the first filename is larger, it will need to exchange the two records to put them in order. It will move record ‘d’ into ‘temp’, ‘z’ into ‘d’, then ‘temp’ into ‘d’, thereby exchanging the two records.

The last two lines, before the ends, are just clearing the pertinent information from the temp record, so there is no possibility of having problems on the next comparison.

In the userlist MPL program that I was trying to write in MPL, I was using this type of a sort to put the list in whichever order the user wanted. It works pretty well, as I stated above, for shorter lists. I wouldn’t try using this type of sort for a list of 1000 records, as it would take forever to complete. After looking at the loops, you will be able to see why. If you had 1000 records, the outer loop would run 1000 times, and the inner loop would run 1000 times the first iteration (decreasing by 1 each subsequent iteration), per iteration of the outer loop.

If you are like me, you learn more with a visualization of how this type of sorting is being accomplished. Take a look at the following website, and click on ‘Bubble Sort’. Watch how it goes through and compares each set of two, and slowly sorts the list. Keep in mind, this list would be sorted on your computer in 100ths of a second, but it gives you an idea of what is actually happening.

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

While you’re on that website, take a look at some of the other types of sorting they have on there. You can see why some are more efficient than others. The one that I think is interesting to watch, is the Quick Sort. Take a look and you’ll see why.

As I said, this is something that you probably won’t need to do right now, but perhaps in the future you will need to. I don’t expect you to memorize all of the steps in this type of a procedure, but just get a basic understanding of how a sort works, and how relatively simple they are to program.