This is a recap of everything covered so far in these lessons. Seems like a lot of ground has been covered but a lot remains. James Michener once wrote about the settlers who had surmounted the mountains on the east coast (no easy feat back then with covered wagons) and looked forward to going further west...California! Over the Rockies and then the pacific. He stated that had the settlers known at the time the height of the rockies that they would not have made the trek, not so much due to the height (which dwarfs the east coast mountains) but because of the associated difficult experience of surmounting the rapid incline of the east coast mountains (the appalachians I believe). Fortunately they didnt know the height, the upshot of his meandering on the subject was that while the heighth of the rockies is much greater the incline (gradient?) is less steep!

Whats all this about? Your over the first set of mountains! These lessons will go on to explore vast territory. The learning curve is not a short one...but it is fun. Ahead are such things as graphics (and associated statements), procedures and functions(what they are and their usefulness),units in the toolbox (treasure chest or pandoras box?),the format and sequence of a program, event handling, menus, windows, resources, etc. A wonderland of sorts but for now let's recap what we have covered.

1) **write** - this statement tells the mac to write something to your screen.

2) **writeln** - this statement tells the mac to write something to your screen and provide a <cr>, where <cr> means carriage return. This command also provides a linefeed.

3) **read** - this statement tells the mac to display on the screen the keyboard input by the user of the program.

4) **readln** - this statement tells the mac to display on the screen the keyboard input by the user of the program and do a <CR> and linefeed when the RETURN key is pressed.

5) The program name line.

PROGRAM Guess_the_number; {name of program)

Pascal program names do not allows for spaces. Use underlines when necessary (___).

6) Comments- The curly brackets **{ }**in a program have special significance they let you comment on any line without worrying about whether the compiler will mess you up by trying to understand what is inside of them. A word about comments, use them, often. By doing so you can glance quickly at a program and refresh your memory about what you were doing. With only 6-10 lines of code this may not seem important but as our programs get bigger it will.

7) End of statement lines- At the end of most lines in a pascal program there is a semicolon. The semicolon is the character that identifies the end of a line. (There are circumstances where no semicolon is necessary)

8) **VAR** line- alerts the compiler that identification/definition of variables will follow.

The following line after VAR identifies the VARiables that will be used in this program and what kind of variable they are.

```
number, guess   : integer;
```

Both number and guess are the variables that are used. they are both integers. (If you are like me you might not remember that math term. An INTEGER is a whole number without a fraction. 35 is a INTEGER, so is 52 but 6.14 is not an INTEGER. Get the idea. Don't worry I read up on it. By the way 6.14 is a REAL number.)

That could also have been written

VAR

```
number   :integer;                    {note the colon separation}
guess    :integer;
```

9) CONSTANTS- are values which will never change within the program.

CONST **{this would have preceded the VAR section. Constants are identified first}**

days_in_year = 365; **{the number of days in the year is now set. It will not change through out the program}**

10) The begin statement

BEGIN {note that the begin statement doesn't have a semi colon}

The next line is

number:= random mod 5; {as commented this generates a random number from 1 to 5. Note the colon and equal sign after number. This **INITIALIZES** the variable.}

11) Compiler- A **compiler** is a program used to convert your program text (code) into a free standing program. In Turbo Pascal you can check the syntax (the correctness of your program) without compiling it. You can also run it (which compiles the program and then presents you with what the program would look like and do if you had double clicked it from the desktop. Finally you can COMPILE it to disk. Doing this will generate a stand alone program that will show up on the desktop with a generic icon (the one with a hand writing on a page).

12) **end.** The 'end. ' statement tells the program that its over. The end statement can and is used often within a program. Just as you can begin a procedure or a loop of some kind, so too must you end it. Hence, you may have many begins and ends within a program.

13) Apostrophes in text- When you want to use an apostrophe or single quote sign (') in the context of a **write** or **writeln** statement you have to use two apostrophes ('). Don't confuse that with double quotes ("). Here is an example:

writeln (' What's your name?'); {invalid statement}

writeln (' What''s your name?'); {valid statement}

14) **Variables** have a few rules about them.

• Variables must be initialized before we can use them. In essence, we must give them their first value.

•Variables can take the form of a number (**integer** or **real**), a **character**, or a string of characters (**str [93]**) .

• Variables can be assigned the value of other variables (if they are the same type).

•Variables can also be a value calculated by other constants or variables (assuming they are numeric).

15) the **while**_____**do** expression is used when we want something to happen and can only express the condition with an inequality such as **while x < 10 do**

16) the **if,then,else** expression lets us state explicit acitivities to be done under specific conditions.'**if,then,else**' condition.
You can 'nest' conditions within conditions. Here is an example:
if this is true **then**
(here is where you describe what to do if true)
else
(here is what you describe what to do if it is not true)

17) the **case** _____ **of** expression allows for a series of activities based on values that will be encountered.

18) **Initializing**- The initialization operator ':=' must follow immediately after the variable...no spaces allowed. All variables must not only be identified and defined but also initialized before they can be used. All that initializing does is give the program an initial value of a variable to work with: it can change often after that.

19) **spaces for numbers**- to indicate how many spaces are to be allowed for numbers displayed by your program use **writeln (x:2)** or some derivation of this. This says write the value of x and give it 2 spaces then do a <cr>. This keeps a column of numbers 'justified' all the unit digits will display in the unit column and the tens digits in the tens columns.

20) **for___do**- the for to do statement is very similar to the **while____do** statement. Remember that the **while____ do** statement dealt with an unequality, that is, one value was identified as either greater than or less than the other and as long as (while) that condition remained certain things happened. **For___do** is the same thing except it is for an equality...that is, it will set things in motion until the two variables are equal (it uses a built in counter).

21) A **boolean** is a variable type (remember integer , char, string..those are types too). The boolean variable deals with **true** conditions and **false** condtions.It can only assume one of these two values. It is either true.....or it is false. Only one of these need actually be defined, that is if I tell the program what will represent a true condition, then all other conditions will be false, and vice versa if I tell the program what will be a false condition ..then all other conditions will be false. Just associate it with true statements and false statements.

22) **repeat,until.** This statement tells the program **repeat** doing something **until** a certain condition is met.

Now we can make some wonderful things happen but only as far as text is involved. *ANY* computer can do that. Macs can do more. Like graphics. And thats where we are headed next.
If you don't feel comfortable with any if the above items covered so far re-review them...and let me know. I can modify the lessons to include things that i may be subconsciously assuming or overlooking.