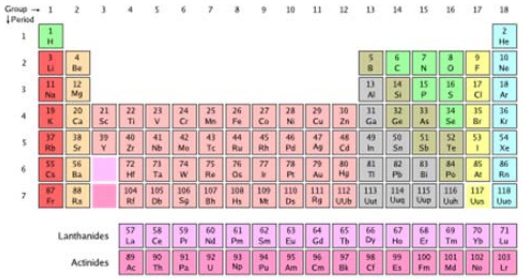
CIS 1.5 (Science Section)
Brooklyn College
Professor Langsam



Assignment #4

This assignment will classify chemical elements. Although there are 118 elements listed in the Periodic Table, the program will only work with a smaller subset of them.

A data file, *PeriodicTable.txt*, has been created in the following format:

Atomic number Name Atomic weight Group Period

You may download the file at: http://eilat.sci.brooklyn.cuny.edu/cis1_5/PeriodicTable.txt. You are to write the following functions:

checkNumber – This function will receive an *atomicNumber* as a parameter (call by value)and will check whether the *atomicNumber* is a valid number. Valid numbers are the values 1-118. If the number is valid, it should print an appropriate message and return **true**; otherwise it should print an appropriate error message and return **false**.

readData – This function receives an *atomicNumber* (call by value) and returns the corresponding *name*, *atomicWeight*, and *period* (call by reference). Each time the function is invoked it opens the file and searches for the corresponding entry in the data file.

printName – This function receives an element's *name* and prints a heading and the name of that element

isNobleGas – This function receives an integer representing the element's group (call by value) and returns ***true*** if the element is a noble gas and ***false*** otherwise. Elements in group 18 are noble gasses.

isAlkaliMetal – This function receives an integer representing the element's group (call by value) and returns ***true*** if the element is an alkali metal and ***false*** otherwise. Elements in group 1 are alkali metals.

isAlkaline – This function receives an integer representing the element's group (call by value) and returns ***true*** if the element is an alkaline earth metal and ***false*** otherwise. Elements in group 2 are alkaline earth metals.

atomicParticles – This function receives an element's *atomicNumber* and *atomicWeight* (call by value) and returns the number of *protons*, *neutrons* and *electrons* in the atom of that element (call by reference). Recall that:

number of protons = number of electrons = atomic number

number of neutrons = atomic weight – atomic number

classify – This functions will receive an *atomicNumber* (call by value) and call the function **readData** to obtain that element's relevant information. If the information is not in the *PeriodicTable.txt* data file, it should print the message: "*Element not listed yet."*

In all other cases it is to call the functions **printName**, **isNobleGas**, **IsAlkaliMetal**, **isAlkaline** and **atomicParticles** in order to print the appropriate information. Note: with the exception of **printName**, all printing is to be done by the **classify** function. Also note, that an element cannot be classified as in more than one group (e.g., it cannot be both a noble gas and also an alkali metal), so you should be using **if…else** statements. If the element is not a noble gass and is not an alkali metal and also is not an alkaline earth metal, the function should print "*Not classified yet*."

main – The main function is to use a loop to process date contained in an input file. An atomic number is to be read in. The main function should call the function **checkNumber** to check that the atomic number is valid. If the atomic number is valid it should call the function **classify** to print out all the information (as described above) for that element. However, if the element number is not valid, an appropriate message will be printed (by **checkNumber**) and go on to the next input.

When all the input has been processed the **main** function should print the total number of input elements processed, and how many inputs were valid and how many inputs were invalid.

Input data:

Create a data file that contains the following entries:

```
2
120
20
19
79
```

Make up 10 more elements that will test your program. Include some invalid data (both too small and too large) and include some elements whose names have not been listed yet as well as some whose types have not been classified yet.

Sample Output:

For the data above your program is to print:

```
Element #2: Helium
Period 1
Classification: Noble Gas
Number of Protons: 2
Number of Neutrons: 2
Number of electrons: 2


Element #120: Invalid data

Element #20: Calcium
Period 4
Classification: Alkaline Earth Metal
Number of Protons: 20
Number of Neutrons: 20
Number of electrons: 20

Element #19: Name not listed yet

Element #79: Gold
Period 6
Classification: Not classified yet
Number of Protons: 79
Number of Neutrons: 118
Number of electrons: 79
```

Style:

Each function should have a good comment explaining its role in the program and what parameters it receives. If the function calls another function, the comment should say so. Use meaningful variables and indent your program as described in class. All output is to go to a file. Note that there is no console input required, thus there is no reason to prompt the user.

Programming Note:

When a file is closed, the file pointer remains pointing to the last item that was read. In order to have the pointer move to the beginning of the file, it is necessary to use the clear() file method. Each time the function **readData** is called, it must begin searching through the *periodicTable.txt* file from the beginning. Thus your function should be structured (in part) as follows:

```
infile.open("OriginalFile.txt"); //open the file
if ( !infile.is_open() ) {       //checks to see if file was opened
      //The file could not be opened
      cout << endl << "ERROR: Unable to open file" << endl;
      system ("PAUSE");
      exit(1);
}
infile.clear();                  //REQUIRED prior to reopening file


//search through the file for the desired information


infile.close();                  //close the file
```