**Juan Bavaresco**

**FUNCTIONAL CALCULATOR**

The algorithm starts with the creation of 1 class called Formula, in charge of doing the equations calculations. It contains a vector of string called “elements” (which will save every number, sign, arithmetic operator or parenthesis found on the string), an “Initialization” method and a couple of methods in charge of calculations.

The Main method of the program starts by declaration of a variable of type ifstream, which will contain the data coming from the textfile (in this case, found on the program’s folder and called “data.txt”. The program terminates if there is an error on the file.

Then, as long as there are lines on the file, a series of commands are executed. First, we ignore lines that start with the character ‘#’. Then we check if the string line has any syntax errors. This is done with the “HasErrors” function.

This function starts counting the amount of open parenthesis and closed parenthesis on the equation string. If they are different, then there is a syntax error.

If they are equal, then the function follows with the declaration of a regular expression (a pattern to identify matches in a text stream) which will be used to identify syntax errors in the string. This regular expression has the following structure:

[^-.+\*()\/0-9] | [-.+][-+] | [\*.\/][\*\/] | [.]{2,} | [0-9.][(] | [-.+\*\/][)] | [(][)\/\*] | [)][0-9.(]

[^-.+\*()\/0-9]: matches a character different (^) than a number or than the characters - . + \* ( ) /

[-.+][-+]: matches if a character between - . + is followed by a character between - +

[\*.\/][\*\/]: matches if a character between \* . / is followed by a character between \* /

[.]{2,}: there are 2 or more dots together

[0-9.][(]: there is a number or a dot before an opening parenthesis

[-.+\*\/][)]: there is a - . + \* / before a closing parenthesis

[(][)\/\*]: there is an opening parenthesis followed by a ) / \*

[)][0-9.(]: there is a closing parenthesis followed by a number, a dot or (

Then, a regex\_iterator goes with this expression through the whole equation string to get all the respective matches. If the iterator has any matches, then there is a syntax error on the string, which is printed to the console and the string is discarded.

On the other hand, if the equation string has no errors, then we instantiate a new Formula object with this string using the Formula constructor, that uses the “Initialization” method.

The “Initialization” method works similarly to the “HasErrors” function.

It starts with the declaration of a regular expression which will be used to separate the string into numbers, +, -, \*, / and parenthesis. The expression has the following structure:

^[-+](?:[0-9]\*[.])?[0-9]+ | (?:[0-9]\*[.])?[0-9]+ | [-+\*\/()]

^[-+](?:[0-9]\*[.])?[0-9]+: at the start of the string, there is a + - sign, followed by a float or integer value

(?:[0-9]\*[.])?[0-9]+: there is a float or integer value alone

[-+\*\/()]: there is a - + \* / ( ) character

Again, a regex\_iterator goes with this expression through the whole equation, which has already been tested that has no syntax errors, to get all the respective matches. If the iterator finds matches in the string, each match is inserted into the “elements” vector that is a data member of the object. This is the following way: the regex\_iterator, if successful on finding a match, becomes a dereferenceable iterator pointing to a valid [match\_results](http://www.cplusplus.com/match_results) object. Then, by dereferencing the iterator, we obtain a match value, and after inserting the match value into the “elements” vector, the iterator is incremented to point to the following match.

Then, after the object has been instantiated and initialized, the calculation is done using Formula methods and the result is printed on the console. This process is repeated for every line in the file.

The first method to be explained is the “Plus\_Minus\_Sign” method, which discovers if a + - sign is for positive/negative and not for sum/subtraction. First, a vector<string>::iterator searches for the first + or – inside the “elements” vector, and if found, the iterator points to that element in the vector. Then, we search if the + - sign is preceded by an open parenthesis, a multiplication sign or a division sign. If false, the + - sign is for sum/subtraction and a following + - sign is searched. If true, we now find if the + - sign is followed by a number. If false, then once again the + - sign is for sum/subtraction and a following + - sign is searched. If true, then we now the + - sign is for positive/negative, and the following number is modified accordingly. Then, a following + - sign is searched.

This method has been created because, since C++ uses ECMAScript's regex syntax, it doesn’t support look-behind constructs. Why is this important? Because if look-behind could be used, then the regex used to separate the string into numbers, +, -, \*, / and parenthesis could also identify if the + - sign is for positive/negative, in the following way:

(?<=\(|\/|\\*)[+-](?:[0-9]\*[.])?[0-9]+: there is a + - sign, not preceded by a ( / \*, and followed by a float or integer value.

The next one is the “Operator” method. This method does operations with an specific arithmetic operator between the four valid. First, a vector<string>::iterator searches for the first arithmetic operator in a range between two elements of the previously filled “elements” vector, and if found, the iterator points to that element in the vector and operations are made. The values at the “elements” vector between the operator (pointed by the iterator) are saved as doubles (we know they are numbers as we already found out that the equation has no syntax errors) and the operation is made according to the operator. Then, the 3 elements on the “elements” vector that took part on the operation (the two values and the operator) are substituted by the operation result, and the iterator searches for another operator of the same kind, and the same process is made until there are no more.

We also have the “Calculations” method, which creates a vector<char> filled with the four operators to be used (' / ', ' \*' , ' + ', ' - ') in order of importance. Then a for loop is created to iterate through this vector<char>, to be able to use the “Operator” method on the “elements” vector for each arithmetic operator, and therefore obtaining a final calculation value.

Finally we have the “All\_Calculations” method. This method basically counts initially how many parenthesis the “elements” vector has. Then, it modifies the equation by calling the “Plus\_Minus\_Sign” method, and then it simplifies the equation by doing calculations starting from the inner-most pair of parenthesis by using the “Calculations” methods, until there are no more inside-parenthesis terms to solve. Then, the simplified equation is finally solved and returned, in order to be printed to the console.