I programmed this code in eclipse in Ubuntu, which is quite similar to Visual Studio in Windows. The makefile is included in this package. To generate the executable file, just type make in the terminal.

(1)Defined Rules for Expression of Equations

In the original description of the problem, there is such an example equation x^2 + 3.5xy + y = y^2 - xy + y, where the variables are denoted by single character and the multiplication symbol is omitted. This is fine for equation where the variable only consists of single character. However, it will cause ambiguous meanings for equations where the variables are denoted by more than one characters; for example: x1, x2, Sx, Sy, abc. They can be single variable or the multiplication of several variables, or the multiplication between a number constant and a variable. To have clear meaning for expression of equations, here we define some rules for the expression of equations:

(1) All the variables should be enclosed by a pair of round brackets, for example, (a), (x1), (x2), (abc), etc.

(2) Multiplication is denoted by ‘*”, and cannot be omitted, for example:
(a)*2, (a)*(b), (x)*(y), (x1)*(abc)*3.5….

(3) For exponentiation, if the exponent is negative, the exponent should be enclosed by a pair of round brackets since the precedence of “^” is larger than “-”; This is the same for multiplication.

(4) An example of expression is :
(x)^2 + 3.5*(x)*(y) + (y) = (y)^2 – (x)*(y) + (y),
which corresponds to the original expression of : x^2 + 3.5xy + y = y^2 - xy + y.

(2) Some Terminologies

For better expression, we define several terminologies here.
(1) Variable: a single variable that consists of base and exponent such as (X)^N, where X is the name of the variable, it can be of any length larger than zero, but must start from a character, N is the exponent, which can be any integer value. If N equals one, it can be omitted. In case where N equals zero, there is no need to include this variable.

(2) Term: A term consist of the multiplication of Variables and a numerical constant such as 10.0*(X1)^N*(X2)^M, or just a numerical constant like 10.0, 20.0, etc.

(3) Expression: an expression is a series of terms that are combined by exponentiation, plus, minus, multiplication operators such as:
(10*(X1)^n*(X2)^m + 20)^k. .
In addition, operators such plus, minus, multiplication, exponentiation, and brackets are also treated as expression as well for easy programming.
Please note that if there are more than one terms in an expression, the exponent must be larger

than zero, for non-positive exponent operation is not supported in this program. Actually, I spent a lot of time on this part, it turns out more complicated to define rules of simplifying these expressions. I gave up at the end.

(3) The structure of this program

Four classes are designed, which represent different level of abstraction. Class Var is to present the previously defined variable, Class Term is to represent the term, Class Expression is to represent expression; and Class Parser is to encapsulate the process of simplification. The operation in the level of Expression is transferred into Term level, and the operation in the term level is transferred into Var level.

The algorithm is: first convert the expression into a vector of Expression, then convert the expression of equation from infix into postfix by employing stack, then evaluate the whole expression. During the evaluation process, the expression is ordered lexicographically, which can add "like terms" and therefore save much space and time complexity.

(4) How to Type Inputs

Once running this program, one can input equations in two ways: one by typing the equation ended by ENTER one line by one line in the terminal or console, or by type f("filename") and ENTER to read equations in the file, where "filename" is the name of the file, the program then output the results in the file named by filename.out. One can repeatedly type equations in terminal or have the program read file data until type CTRL + C

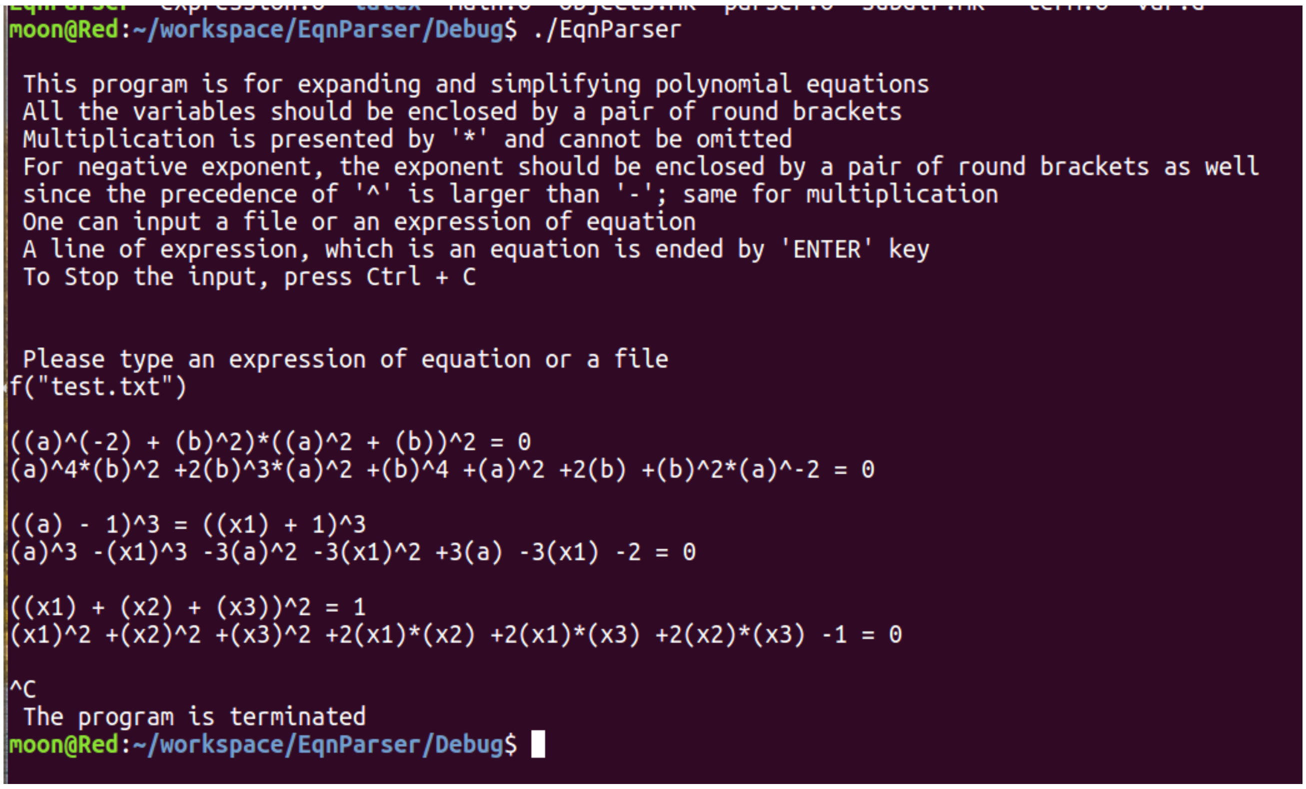
(5) Test Examples

in the file, test.txt, we have following equations, and the results in stored in results.out

```
(((a)^2)*(b) + (b)^2)^3 = 0
((a)^(-2) + 2)^(2) = (x)^(-2)
2.5*(a)^2 + (b)^(-2) = 0.5*(a)^2
((a) + (b)^2)*((x) + (y) ) = 0
(x2)^3 + (y2)^3 + (z2)^3 = 2*(x2)^3
(((x1)^2)*(x2)^2 + (x3)^3)^2 = 0
((a) + 1)^4 = 0
((a)^(-2) + 1)*((a)^2 + 2) = 0
```

We can see they are all correct.

here is a screen-print of how to implement this program in Linux terminal



```
moon@Red:~/workspace/EqnParser/Debug$ ./EqnParser

 This program is for expanding and simplifying polynomial equations
 All the variables should be enclosed by a pair of round brackets
 Multiplication is presented by '*' and cannot be omitted
 For negative exponent, the exponent should be enclosed by a pair of round brackets as well
 since the precedence of '^' is larger than '-'; same for multiplication
 One can input a file or an expression of equation
 A line of expression, which is an equation is ended by 'ENTER' key
 To Stop the input, press Ctrl + C


 Please type an expression of equation or a file
f("test.txt")

((a)^(-2) + (b)^2)*((a)^2 + (b))^2 = 0
(a)^4*(b)^2 +2(b)^3*(a)^2 +(b)^4 +(a)^2 +2(b) +(b)^2*(a)^-2 = 0

((a) - 1)^3 = ((x1) + 1)^3
(a)^3 -(x1)^3 -3(a)^2 -3(x1)^2 +3(a) -3(x1) -2 = 0

((x1) + (x2) + (x3))^2 = 1
(x1)^2 +(x2)^2 +(x3)^2 +2(x1)*(x2) +2(x1)*(x3) +2(x2)*(x3) -1 = 0

^C
 The program is terminated
moon@Red:~/workspace/EqnParser/Debug$ 
```

(6)
Efficiency Test
you can type
((a) + (b))^100 = 0, the results can be obtained within seconds, the efficiency is impressive.