CS200–Introduction to Programming

Assignment 6

Due: 16 Dec 2014

This assignment has three parts and you have ten days for these tasks.

Start early, as the assignment would take time.

The course policy about plagiarism is as follows:

1. Students must not share actual program code with other students.
2. Students must be prepared to explain any program code they submit.
3. Students must indicate any assistance they received.
4. Students cannot copy code from the Internet
5. All submissions are subject to automated plagiarism detection.

Students are strongly advised that any act of plagiarism will be reported to the Disciplinary Committee

Also anything in green refers to something in the actual code.

Part 1 : Building a Lexicon (30 points)

Write a class, which creates a lexicon of words from a file. A lexicon is the set of words used in a particular language or set of collected documents. In this case you will also store the frequency of the word occurrences as well. Your class should take a single constructor argument: a file name. Then you should read the file and create the lexicon of words in the file. The interface for the class is defined in ***Lexicon.h***, and as before, you need add internal data members to the header file, and create a ***Lexicon.cpp*** class that implements the methods described in the header file.

For example, if the input file (say *foo.txt*) has the following line:
"The fox jumped out of the basket"

And we execute "*./p3* lexicon *foo.txt"* on the command line, the lexicon for this file should contain the entries:
the, 2
fox, 1
jumped, 1
out, 1
of, 1
basket, 1

You should ignore the case of words - note that "The" and "the" are treated as the same word. You only need to break the word boundaries on whitespace. Punctuation will be considered part of the word. Feel free to use the example code as a base for your code.

Hint: Using the map template class in the STL makes implementing the Lexicon class very simple.

Part 2 (50 points): Stack-based calculation

You are to implement an interactive stack-based calculator class using the STL to store internal data. Obviously, you should leverage the stack STL class to do this. All inputs should come in from standard in (cin), and when an operator is encountered, the bottom two numbers on the stack should be combined using that operator (top of stack is the left operand), and then the result should be pushed onto the stack (lines beginning with `>' are inputs):

```
>> ./p3 stackcalc
Welcome to the Stack calculator!
> 2.5
> 4.33
> 7
> +
--> 11.33
> 0.83
> -
--> 11.00
```

```
> +
--> 13.5
> +
--> Sorry, there is only 1 entry on the stack.
<and so on>
```

You must implement the following five operations: +,-,*,/, plus an additional operator, pop, which pops the bottom-most (i.e. most recent) item o_ of the stack. Therefore, the only valid tokens are:

- *pop*
- *quit (should exit the program)*
- *show (display stack contents)*
- *One of the four arithmetic operations*
- *<non-zero number>*
- *<number>.<number>*

If the input token does not fit one of the above patterns, provide an error message and ignore the input. Likewise, if there are not enough arguments for an operator (e.g. only one item on the stack, and a + is encountered), an informative error should be printed and the operator should be ignored. Note that the value 0 is not considered valid input. All output should be directed to standard out.

Part 3: Template based Matrix (20 points)

Define matrices class in all classical numerical formats: bool, float, double, complex<float> or complex<double>. It should supports two implementations of dense matrices (array) and sparse matrices (array of link list) based on a template parameter. (Hint: declare an *enum matrixtype {dense,sparse}*).

Constructors:

Matrix(int rows, int cols)

Matrix(int rows, int cols, const T & initvalue)

Matrix(const Matrix & m)

Functions:

Matrix transpose()

Matrix minor_matrix(int row, int col)

Matrix gauss_jordan();

int rows()
int cols()

```
void swap_rows(int r1, int r2)

void remove_row(int row)

void remove_column(int  col)
operator+(const Matrix<T> &)
Matrix<T> operator*(const T &) const;
T myabs(const T &) const;  // A portable absolute value.

friend ostream& operator<< <T> (ostream &, const Matrix<T> &);
```