CSCE 121(505-508) **Homework 5**
Due 11:59 p.m., 2/24/14 [100 points]

This Homework assignment is meant to (a) help you understand how grammars are parsed, and (b) give you experience with input/output.

All work should be turned into CSNet by the deadline. **In addition, please bring a hardcopy printout of your code to class the next day.** For programs, you need to turn in only the source code (not object or executable code). Your code will be tested using g++: you are welcome to develop in Visual Studio, but please make sure your code also runs in g++.

0) Create a text file, README, in which you:
    a. State the Aggie Honor statement, or else explain why you cannot do so.
    b. List any resources used, outside the textbook and discussions with the Instructor, TA, or Peer Teacher
    c. List any known problems with the assignments you are turning in. For example, if you know your code does not run correctly, state that. This does not need to be a long explanation.
    d. For places where you may have done some additional work, put in a brief summary detailing what you did.
1) [100 points] Below is a grammar that can be used to handle question 1b from Homework 3. You are to write a program that can:
    a. Ask a user for an input file name and an output file name
    b. From that input file, parse statements written according to the grammar described below.
        i. Note: you should follow the pattern of the calculator grammar reader.
        ii. You only need to account for the grammar below, not anything that the below grammar does not handle.
    c. You are to convert the input into a Boolean expression.
        i. Each predicate should be assigned a letter, from A to Z. There will be no more than 26 predicates.
            1. Ignore capitalization in predicates. For example, “It is raining” is the same as “it is raining” is the same as “IT IS RAINING”
        ii. You should use the ! command to indicate “not”, the && to indicate “and”, the || to indicate “or”, and the -> to indicate an if-then (the if part coming before the arrow, the then part after the arrow)
    d. You are to output to the output file the following:
        i. For each statement, output a single line giving a Boolean expression corresponding to the input.
        ii. Following that, output a blank line followed by a list of the different predicates, one per line. These should be of the form <Predicate ID>: <Predicate>
    e. If you encounter an error in the input (i.e. an invalid input), ignore the rest of the text until a period is encountered, or until the end of the file.
        i. Input is invalid if it doesn’t correspond to the grammar below. Don’t worry about whether it makes sense or obeys other rules – if it obeys the grammar, it is valid.

Here is the grammar to use:

Statement:
    Conditional “.”
    Phrase1 “.”
Conditional:
    “If” Phrase1 “then” Phrase1
Phrase1:
    Phrase1 “or” Phrase1
    Phrase2
Phrase2:
    Phrase2 “and” Phrase2
    Phrase3

Phrase3:

Words “not” Words

Words

Words:

Word Words

Word

Word:

{String of letters, except reserved words: If, then, and, or, not}

A predicate is considered to be the set of words from a Phrase3 (either the Words or the two groups of Words before and after the “not” joined together).

So, if the following is the input file:

```
If it is raining then it is cloudy.  It is not raining.          // Notice two statements on one line
It is cloudy and I am dry.
I am not dry and the
sky is blue.                                                    // Notice the broken line
If it is cloudy and I am not dry then it is raining.
It is raining it is snowing wow.          // This sounds bad but is perfectly valid
If it is raining it is snowing            // Notice the bad formatting (no “then”)
It is raining.                            // This is skipped since we go until the first period.
```

Could have the following output:

```
A -> B
! A
B && C
! C && D
B && ! C -> A
E
```

A: It is raining
B: It is cloudy
C: I am dry
D: The sky is blue
E: It is raining it is snowing wow

Notes and hints:

- Partial credit will be given, so it is OK to have a limited set of options working.
- Make sure you have one part working before moving on to the next. Don’t try to put the whole thing together at once.
- Build up slowly from the bottom/basics first. For example, make sure you can read in file names, open and read a file, and open and write a file. Then work on reading in words and outputting them. Then work on identifying words and identifying a predicate. Then work on adding in the Phrase 2 stuff, then Phrase 1, then conditional, etc.
- For predicates, you will probably want to store a vector of strings. To identify a predicate, you will need to check the existing predicates first, to determine whether it’s new or not.
- Start out handling everything in lower case (or upper case). Separately, write a function that converts one type of string to all lower (or upper) case. Then, you can handle input by always converting it first, and then know that you can compare strings easily.
- You might want to write output while you parse, rather than creating a string and then outputting.
- When implementing, don’t try to handle all the error cases at first. Make sure you have a routine that handles the “correct” case, and then add in checks for the errors.