Project 1 – CS222, Fall 2007 – Proposition Parsing

Description

You will write a program to compute the truth value of simple propositions involving up to three statement variables. You will be providing the proposition and the truth values as input to the program and displaying the result.

Suppose you wanted to write a program to calculate the truth value of a particular proposition like $A \land (\neg B \lor C)$, given truth values for $A$, $B$ and $C$. Not too tough. It isn’t even too bad to write a program that prints out the truth table for this proposition.

Now suppose you want to write a program that can calculate a truth value for any proposition, not knowing what the proposition will be ahead of time. That’s a whole other kettle of fish. Yet, Java compilers deal with this every time you write an `if` statement or a `while` loop, and spreadsheets handle it for every formula they calculate. How do they do it? There are a few well known techniques that we’ll take advantage of to simplify this problem.

Requirements

1. Your program will request that the user enter a proposition. To keep things simple:
   1. Only statement variables $A$, $B$ and $C$ will be used in the propositions.
   2. The strings “AND,” “OR,” and “NOT” will be used rather than special symbols for the operators.
   3. The expressions will be in postfix format to simplify processing. Postfix format is unambiguous, so no parentheses are needed. It also allows a stack to be used in a natural way for evaluating the expressions. Some examples:

<table>
<thead>
<tr>
<th>Proposition – infix notation</th>
<th>Proposition – postfix notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \land B \land \neg C$</td>
<td>$A \ B \ A \ C \ A \ B \ A \ C$</td>
</tr>
<tr>
<td>$(A \lor B) \land C$</td>
<td>$A \ B \ C \ A \ C \ A \ B \ C \ A \ C$</td>
</tr>
<tr>
<td>$\neg A \lor B \lor C$</td>
<td>$A \ A \ B \ C \ A \ B \ C \ A \ C \ A \ B \ C$</td>
</tr>
<tr>
<td>$A \land (B \lor C)$</td>
<td>$A \ B \ C \ A \ C \ A \ B \ C \ A \ C \ A \ B \ C$</td>
</tr>
</tbody>
</table>

2. Your program will request the truth values for the three statement variables, $A$, $B$ and $C$. You should expect the truth values to be entered as “T” or “F”.

3. Your program should evaluate the expression and then produce a message indicating its truth value.

4. All input should be checked for validity.
   1. For the truth values, your program should loop until either “T” or “F” is entered.
2. For the proposition, if your evaluation code finds an unrecognized token (i.e. anything other than A, B, C, AND, OR or NOT), it should throw an exception. Here is the exception class I used in my implementation:

```java
public class UnexpectedTokenException extends Exception {
    private String message;

    public UnexpectedTokenException (String msg){
        message = msg;
    }

    public String toString(){
        return ("Found unexpected token: " + message);
    }
}
```

You should be able to drop this code into your program as an inner class.

5. Your code should display good object oriented design. For a program such as this that gathers some input, performs some computation, and the produces output, it is acceptable to collect the input and produce the output from the main method. The computation should be encapsulated in an appropriate class. I will be happy to discuss your program design with you.

**Parsing postfix expressions**

There is a particularly simple algorithm for parsing postfix expressions. In the following explanation, the term “token” simply means a whitespace delimited word from the input.

While there are tokens left in the expression
    Examine the next token
    If it is one of the statement variables,
        then push the truth value of that variable onto a stack
    else
        determine which operator the token represents
        pop the appropriate number of values from the stack
        apply the operation to the popped values
        push the result back onto the stack

When all symbols are exhausted, pop the answer is the value at the top of the stack. Pop it off and display it.

**Implementation suggestions**

The `JOptionPane.showInputDialog` method is handy for collecting input values. `JOptionPane.showMessageDialog` can be used to display output. If you use these classes, you must import the `javax.swing` library at the beginning of your program.
Once you have read in your proposition, you must break it into individual tokens. The traditional way to do this is with a `StringTokenizer` object, but the `Scanner` class is somewhat less clunky, and you can create a `Scanner` that operates on a `String` object that has already been read in – for example:

```java
String input = JOptionPane.showInputDialog("Enter a proposition");
Scanner s = new Scanner(input);
String firstWord = s.next();
```

Java provides a reasonable generic `Stack` class in the `java.util` library, which you are welcome to use. If you are not familiar with the use of the generic collection classes, here is an example of creating a `Stack`:

```java
Stack<Boolean> s;
s = new Stack<Boolean>();
```

While you are most likely to be working with `boolean` primitive type values, newer versions of Java (1.5 and higher) handle conversion between primitive types and their associated wrapper classes automatically.

**Write-Up**

With this project, you must supply a well-written logical discussion of your program with sufficient detail so that – by reading this discussion but *not* looking at your program – another student in the class could construct a correct program and understand the theory on which it is based. At a minimum, the write-up must include discussion of the algorithm(s) you used in your implementation and any associated underlying mathematics. Also include the results of your experiment outline above.

A sample project and associated write-up are provided on the course assignments web page to give you an idea of what I expect here.

**Submission**

Email to lbarnett@richmond.edu as attachments your Java source code files by 5:00pm, Wednesday, September 26. Please remember that things can go wrong with computers. Your goal should be to have the project turned in at least 24 hours before the deadline to leave yourself time to deal with any unexpected problems. Also recall that the write up will take you some time and will have significant weight in the grading.

Consistent with the guidelines provided in the course syllabus,

- programs that do not compile will receive no credit;
- programs that do not execute and/or terminate without crashing will receive no credit;
- any submission not received by the deadline will not be accepted.

Your work on this and all projects in this course is subject to the conditions of the Honor Code as described in the course syllabus.