Assignments

Assignment 2

Attached Files:
- data2eval.txt (478 B)
- trees4data2eval.png (128.764 KB)
- commandwindow4data2eval.png (65.732 KB)
- TreeDisplay.java (5.435 KB)
- UseTreeDisplay.java (1.143 KB)
- data.txt (67 B)
- tress4data.png (84.898 KB)

In this project you will build a system to represent, evaluate, and normalize boolean expressions. It is intended to give you practice using references, binary trees, and recursion.

Binary expressions are defined by the grammar:

- `Expression -> "(" + Expression + " " + BinaryOp + " " + Expression + ")"`
- `Expression -> "(" + UnaryOp + " " + Expression + ")"`
- `Expression -> Atom`
- `BinaryOp -> "^" or "v"`
- `UnaryOp -> "!"`
- `Atom -> "A" or "B" or "C" or ... or "Z"`

For example:

```
(! ((! (A ^ B)) v (! C)))
```

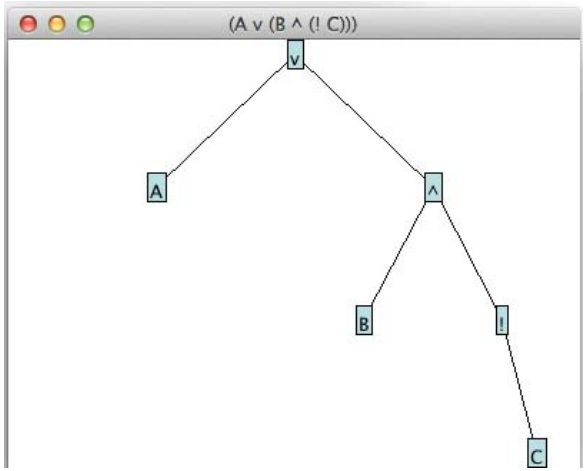
Note that expressions must be fully parenthesized, so the following is illegal:

```
(A v B v C)
```

Each expression will be represented by its parse tree, with operators at the nodes and atoms at the leaves. For instance, the expression:

```
(A v (B ^ (! C)))
```

will represented by the tree:



Notice that binary operators, unary operators, and atoms have different numbers of children. You will use the TreeDisplay class to display the trees you construct. See UseTreeDisplay.java for a simple tutorial on use.

Boolean expressions are evaluated according to the usual rules:

- An atom has whatever value was assigned to it.
- A conjunction is True if both of its conjuncts are True, otherwise it is False.
- A disjunction is True if at least one of its disjuncts is True, otherwise it is False.
- A negation is True if its constituent is False, otherwise it is False.

For instance, if the atoms A, B, and C have values True, True, and False, respectively, we have the following:

- `(A ^ B)` evaluates to true
- `((A v B) ^ C)` evaluates to false
- `(! ((! (A ^ B)) ^ (! C)))` evaluates to true

See data4eval.txt, the data file for evaluating boolean expressions. The companion file trees4data2eval.png shows the expected output for ExpressionEvaluator.

An expression is in *Disjunctive Normal Form* if it is a disjunction of conjunctions of atoms or negations of atoms. The parse tree of an expression in disjunctive normal form has the following properties:

- There are no conjunctions over disjunctions.

- There are no negations over conjunctions or disjunctions.
- There are no double negations.

An expression can be put into normal form by first pushing negations down the tree using DeMorgan's laws:

```
(! (A v B))  --->  ((! A) ^ (! B))
(! (A ^ B))  --->  ((! A) v (! B))
```

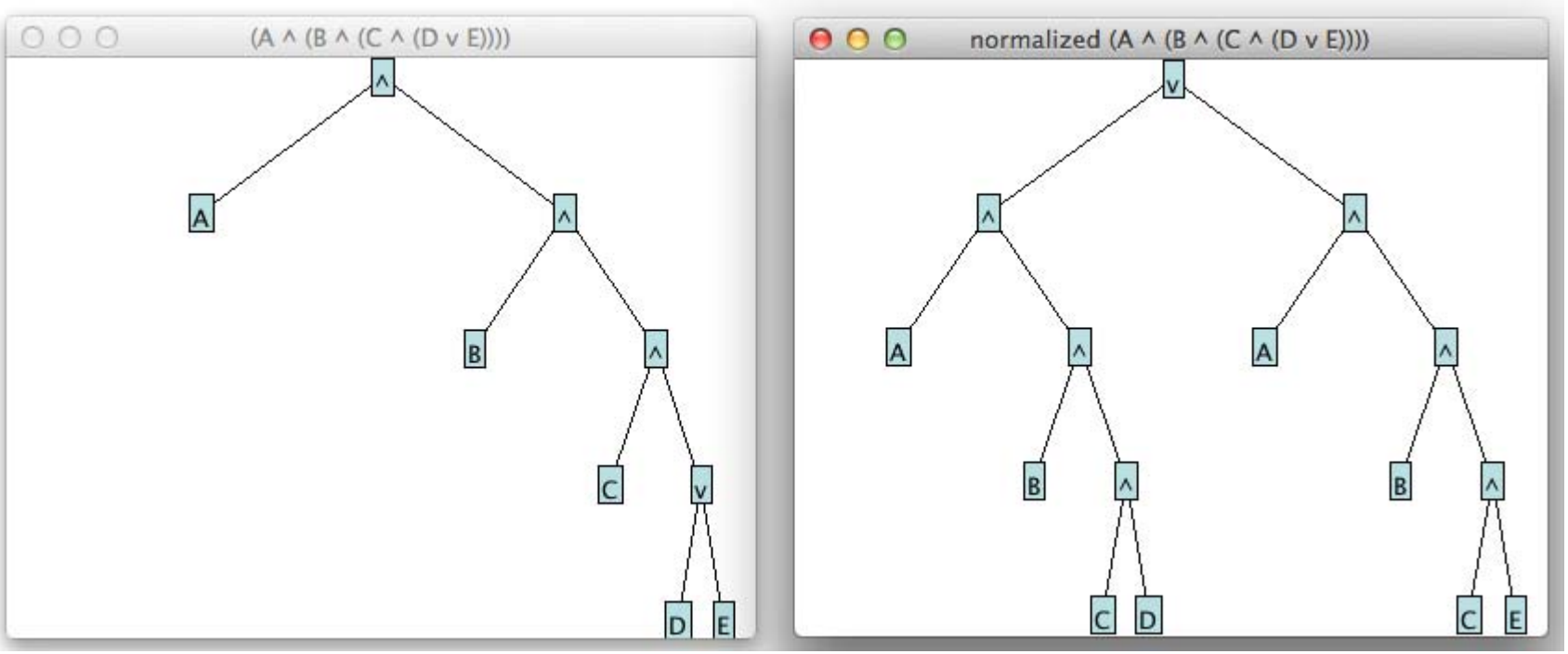
then distributing conjunctions over disjunctions:

```
(A ^ (B v C))  --->  ((A ^ B) v (A ^ C))
((A v B) ^ C)  --->  ((A ^ C) v (B ^ C))
```

and finally eliminating double negations with:

```
(! (! A))
```

Note that `A`, `B`, and `C` in these rules are *Metavariables* that range over expressions. These rules may need to be applied several times until the expression is in normal form.



See data.txt, the data file for displaying expression tree. The companion file trees4data.png shows the expected output for ExpressionDisplayer.

Requirements

You need to submit five files:

- `Expression.java`
- `ExpressionDisplayer.java` (driver for data2display.txt)

  This program and the next are almost identical.

- `ExpressionEvaluator.java` (driver for data2eval.txt)

- `ParseError.java`
- `README.txt`

Most of your work will be in the class definition `Expression` in `Expression.java`. It must contain the methods:

- `public Expression(String s)` creates a new Expression object and its tree and displays the tree.
- `public static void setAtom(String atom, String value)` sets the value of an atom.
- `public boolean evaluate()` evaluates this expression expression.
- `public Expression copy()` makes a deep copy of this expression.
- `public void normalize()` converts this expression's tree to normal form.
- `public void displayNormalized()`displays the normalized tree.
- `public String toString()` returns the print form of an expression.

Your `README.txt` file should:

- Document the usage of your system.
- Describe the major data structures and algorithms you used.
- Be written at a high level.
- Describe known problems with your system.
- Be short and clear.
- Be written in plain text (for portability).

Your documentation should be written for people who will use your system. You should use examples and avoid a detailed description of you code.

To submit this assignment:

- Bundle the files with `zip` into a file named `HW2-NAME.zip` where `NAME` is your full name (first and last) with no spaces or punctuation marks.
- Upload your file into the Assignment tool in CourseWeb.

Each of your files must have a header with your name, email address, last four digits of your PeopleSoft number, and date.

You will be graded according to the following criteria:

- **30 Points:** Constructor works and builds appropriate tree.
- **25 Points:** Methods `setAtom()` and `evaluate()` work.
- **30 Points:** Method `normalize()` and displayNormalized() works.
- **5 Points:** Method `toString()` works.
- **10 Points:** Documentation is clearly written.
- **25 Points:** Correct implementation and output for ExpressionDisplayer.
- **25 Points:** Correct output for ExpressionEvaluator.

This assignment is due 11:59 pm on Tuesday September 24.
