New Mexico Tech
Department of Computer Science

Extra Credit Assignment

CS122 Algorithms and Data Structures

Due 11:00AM, Wednesday, Nov. 24th, 2004

Specifications:

1. For this assignment, you should create a program that is capable of evaluating an arbitrary Boolean expression and returning the proper result. Specifically:

    - Your program should prompt for an expression and then read it in from stdin. It should then evaluate the expression and print the result to stdout. This process should repeat until the user indicates that they want to stop.

    - In your expressions, you should assume that a value of '0' is 'false', while any non-zero value is 'true'. Your calculator should be able to evaluate the following cases:

        A value by itself (i.e. '(1)')

        A simple expression containing two values and a binary operator (i.e. '(1 && 0)'). The binary operators that your program must support are '&&' (logical AND), '||' (logical OR), '==' (equals operator), '!=' (not-equals operator), '>' (greater than), and '<' (less than).

        A simple expression containing one value and a unary operator (i.e. '(!0)'). The only unary operator you need to worry about it '!' (unary NOT).

        A properly parenthesized compound expression containing any combination of the above expressions (i.e. (((3 < 4) && (4 > 3)) || 0).

    - A good technique for solving this problem is to have your program manage three stacks, one for operands, one for operators, and one for keeping track of parenthesis. You may implement your solution however you wish, though you may find the following high-level algorithm helpful:

1. While the user has more data to enter, display a prompt and read in the next expression (use cin.getline() to read the whole line at once).

2. For each character in the input string, from beginning to end:

   2a. Examine the character, and if it is a left parenthesis, push it onto the parenthesis stack, If it is a value, then push it onto the operand stack (also called the "value" stack…you will need to add some additional processing to handle values that are more than 1 character long). If it is an operator, push it onto the operator stack (you will need to add some processing to differentiate operators that are represented in 1 character from those represented in 2 characters). If it is a right parenthesis, then pop the parenthesis stack, otherwise continue from the next input character.

   2b. If the pop of the parenthesis stack does not return a left parenthesis, the input is malformed and this is an error condition. Otherwise pop the operator stack, and if you get no operator, then check to make sure the value stack is not empty. If it is, then you were given an empty expression (i.e. something like '()') and this can be considered an error condition, otherwise if it is not empty, continue from the next character in the input.

   2c. If the pop of the operator stack *did* return something, examine the operator you received, and pop one item off the value stack in the case of a unary operator, or two items in the case of a binary operator. Apply the appropriate operation to the value(s), and then push the result back onto the value stack. If at any point the value stack is empty when you try to pop from it, then your input was invalid and this is an error condition. Assuming no errors have occurred, continue from the next character in the string.

3. Once you have iterated across the entire input string, the result of the expression should be at the top of the value stack, and the operator and parenthesis stacks

should be empty (if they are not you have an error condition). Pop this value off the stack and display it as your result. Continue from step 1 as long as the user has more input.

- If you follow the suggested implementation guidelines, you may use whatever stack code you wish (you may do so as well if you do not follow the suggested implementation, assuming that your implementation still uses stacks), including the stack that is part of the C++ STL, the stack that you created for the previous programming assignment, or the stack code that is distributed with this assignment.

- Sample program output is included at the end of this document.

2. As always, coding and documentation guidelines should be followed, and a README file should be submitted along with your source code. At a minimum, the README file should include **compilation instructions**, and **instructions on how to use your program**.

3. Note that the user-friendliness of your program, which includes things like displaying clear, concise instructions to the user detailing what the program is and how it should be used when your program starts, and the formatting applied to input prompts and output data, **is** important and **does** count. This program relies heavily on user input, so user-friendliness is especially important in this case.

Submission Guidelines:

Use whatever archiving software you prefer to archive your source and README files together. The archive should be called '<your name>.Extra_Credit.<ext>'. You may include a directory structure in your archive if desired, but it is not required. E-mail your archive to aroth@nmt.edu with the subject line of "[CS 122] Extra Credit" by the due-date posted on the website. In the body of your e-mail, be sure to include your name and student-id number

Grading Criteria:

All points for this assignment are pure extra-credit points, and can in no way hurt your overall course grade. The points are allocated as follows, and there are no extensions to this project:

A correct implementation of the Boolean calculator is worth 70 points.

A useful and descriptive README file is worth 15 points.

Adhering to coding style and documentation guidelines is worth 15 points.

Sample Input/Output:

```
This program will evaluate boolean expressions and return
a value of 'true' if the expression is true, or 'false' if it
if false.  Your boolean expressions must be fully parenthesized.
Enter a blank line at the prompt to quit the program.


Enter the boolean expression to evaluate now: (!0)

        Your expression is TRUE

Enter the boolean expression to evaluate now: (!1)

        Your expression is FALSE

Enter the boolean expression to evaluate now: (!6734)

        Your expression is FALSE

Enter the boolean expression to evaluate now: (5 > 9)

        Your expression is FALSE

Enter the boolean expression to evaluate now: (5 < 9)

        Your expression is TRUE

Enter the boolean expression to evaluate now: (5 == 4)

        Your expression is FALSE

Enter the boolean expression to evaluate now: (1 || 1)
```

```
        Your expression is TRUE
Enter the boolean expression to evaluate now: (1 || 0)
        Your expression is TRUE
Enter the boolean expression to evaluate now: (0 || 0)
        Your expression is FALSE
Enter the boolean expression to evaluate now: ((34 > 100) || (34 < 100))
        Your expression is TRUE
Enter the boolean expression to evaluate now: (1 && 0)
        Your expression is FALSE
Enter the boolean expression to evaluate now: (1 && 1)
        Your expression is TRUE
Enter the boolean expression to evaluate now: (0 && 0)
        Your expression is FALSE
Enter the boolean expression to evaluate now: ((34 > 100) || (!(34 < 100)))
        Your expression is FALSE
Enter the boolean expression to evaluate now: ((5 > 6) == 0)
        Your expression is TRUE
```