Project 2: An Arithmetic Expression Parser
Due: 11:59PM, Friday, October 29, 2004.

Project Objective:
1. Know the order of executions (precedence) and associativity of arithmetic operators (+, -, *, /, %)
2. Manipulate data using arrays
3. Practice loops (while, do..while, for) and program selection (if, if..else, switch)

Project Description:
You will implement an arithmetic expression parser--a program that takes an arithmetic expression and inserts parentheses to explicitly delineate the order of execution. For example, it will return (3+(5*2)) on input of 3+5*2. The input will only consist of single-digit integers (i.e. 0,1,2,...,9) and arithmetic operators (+, -, *, /, %). The maximum length of an input expression is 20; however, you must write the program using the preprocessor value MAX_LENGTH_OF_EXPRESSION 20. This will make your program more scalable. The project has two parts:

1. Print out welcome message and gather user input. The welcome message must read exactly as follows:

```
Welcome to the Arithmetic Expression Evaluator!
Give me an arithmetic expression and I will insert the appropriate parentheses per the C language's arithmetic operator precedence and associativity rules.
Expression:
```

The user's input should be stored into a character array. Make sure your array is large enough to store the expression with parentheses inserted.

2. Output the final expression and intermediate steps with each step adding exactly one set of parentheses to the expression that you printed out in the previous step. For example, the expression 5+3/-2 will output the following:

```
Expression: 5+3/-2
5+3/(-2)
5+(3/(-2))
(5+(3/(-2)))
```

While the resulting final expression will always be the same, the intermediate steps may be different, depending on your algorithm. In any case, the number of steps will be the same. Only print one per line.
For example, given the expression $5+3/\cdot2+4*\cdot3$, either of the following outputs is correct.

(a) $5+3/\cdot2+4*\cdot3$
    $5+(3/(\cdot2))+4*\cdot3$
    $(5+(3/(\cdot2)))+4*\cdot3$
    $(5+(3/(\cdot2)))+(4*\cdot3))$
    $((5+(3/(\cdot2)))+(4*\cdot3))$

(b) $5+3/\cdot2+4*\cdot3$
    $5+3/(\cdot2)+4*\cdot3$
    $5+(3/(\cdot2)+(4*\cdot3))$
    $(5+(3/(\cdot2)))+(4*\cdot3))$
    $((5+(3/(\cdot2)))+(4*\cdot3))$

Note: there are other ways to achieve the correct final expression as well. It is up to you to decide how you want to do it. We have supplied a sample algorithm; however, we encourage you to try and devise your own.

**Sample Algorithm:**

1. Insert user input into an array.
   
   $5 + 3 \div 2 \ 0 \ \ \ \ \$

2. Determine the first two operators in the statement.
   
   $5 + 3 \div 2 \ 0 \ \ \ ^\wedge \ ^\wedge$  

3. Determine the precedence of each operator.
4. If the second operator has greater precedence than the first, compare the second operator with the next operator until you find an operator with lesser precedence or discover the end of the array.
   
   $5 + 3 \div 2 \ 0 \ \ \ ^\wedge \ ^\wedge$  
   $5 + 3 \div (\cdot2) \ 0 \ \ \ ^\wedge \ ^\wedge$

5. Insert parentheses around the sub-expression, in this instance "-2".
   
   $5 + 3 \div (\cdot2) \ 0 \ \ \$
6. Repeat 2-5. Realize that you may now ignore any operators within parentheses, as a parenthesized sub-expression should now be considered as a single operand.

**Project Requirements:**

1. You must program using C under the Glue system and name your program `p2.c`.
2. Your project must compile successfully in Glue.
3. The expression must be stored in a character array that is scalable.
4. Your program must be properly documented.
5. Adhere to the output guidelines set in the Project Description section.
6. Submit your source `p2.c` electronically before the due date. Everyone is responsible for submitting successfully using the `sub114` command. Inability to do so is not a valid excuse for late submissions.

**Grading Criteria:**

Your program will be graded on correctness, coding coding style, and proper documentation. The rubric for this project will look very different. **All programs must compile under Glue for any credit.** You will receive up to 40% of your grade for a good faith effort. Correctness will be graded with specific expressions designed to test your program. Each test is an all or nothing grade. If the program executes correctly for that expression, you will receive all of the credit for that test; if not, you will receive no credit. Finally, you are always expected to thoroughly document your code and practice good coding style.

- Effort: 40%
- Correctness: 40%
  - 15% for primary tests (provided for to test with)
  - 25% for secondary tests (provided after grading)
- Good coding style: 10%
- Proper documentation: 10%