Source File:	`~/2336/41/lab41.(C|CPP|cpp|c++|cc|cxx|cp)`
Input:	under control of `main` function
Output:	under control of `main` function
Value:	5

This problem requires you to write a function to convert an infix expression to postfix format. The evaluation of an infix expression such as $A + B * C$ requires knowledge of which of the two operations, $+$ or $*$, should be performed first. In general, $A + B * C$ is to be interpreted as $A + (B * C)$ unless otherwise specified. We say that multiplication takes *precedence* over addition. Suppose that we would now like to convert $A + B * C$ to postfix. Applying the rules of precedence, we begin by converting the first portion of the expression that is evaluated, namely the multiplication operation. Doing this conversion in stages, we obtain

`A + B * C`	*Given infix form*
`A + B C *`	*Convert the multiplication*
`A B C * +`	*Convert the addition*

The major rules to remember during the conversion process are that the operations with highest precedence are converted first and that after a portion of an expression has been converted to postfix it is to be treated as a single operand. Let us now consider the same example with the precedence of operators reversed by the deliberate insertion of parentheses:

`( A + B ) * C`	*Given infix form*
`A B + * C`	*Convert the addition*
`A B + C *`	*Convert the multiplication*

Note that in the conversion from "`A B + * C`" to "`A B + C *`", "`A B +`" was treated as a single operand. The rules for converting from infix to postfix are simple, provided that you know the order of precedence.

We consider four binary operations: addition, subtraction, multiplication, and division. These operations are denoted by the usual operators, `+`, `-`, `*`, and `/`, respectively. There are two levels of operator precedence. Both `*` and `/` have higher precedence than `+` and `-`. Furthermore, when unparenthesized operators of the same precedence are scanned, the order is assumed to be left to right. Parentheses may be used in infix expressions to override the default precedence.

The postfix form requires no parentheses. The order of the operators in the postfix expressions determines the actual order of operations in evaluating the expression, making the use of parentheses unnecessary.

Write a function to convert an infix expression to its equivalent postfix expression. The function will receive two arguments: a string containing the infix expression and an output stream where the output is to be written. You may assume that the infix expression is *error-free*. An arbitrary number of whitespace characters may occur between any two symbols in an expression. A symbol may be an operand (a single uppercase letter), an operator (`+`, `-`, `*`, or `/`), a left parenthesis, or a right parenthesis. Your function should write the equivalent postfix expression to the given output stream. The output should be formatted as shown below.

Write a second function that will receive two character arguments, each representing an operator (`+`, `-`, `*`, or `/`). The function returns **true** if the first operator has precedence greater than or equal to the second operator and **false** otherwise.

```
I. Pseudocode for converting an infix expression to a postfix expression
   A. Initialize a stack of characters to hold the operator symbols and parentheses.
   B. do
      1. if the next input is whitespace
         a) continue
      2. else if the next input is a left parenthesis
         a) read the left parenthesis and push it onto the stack.
      3. else if the next input is an operand
         a) read the operand and write it to the output.
      4. else if the next input is an operator
         a) while
            (1) The stack is not empty, and
            (2) The next symbol on the stack is not a left parenthesis, and
            (3) The next symbol on the stack is an operator with precedence greater than or equal
                to the next input symbol
         b) do
            (1) Print the top operator and pop it
         c) When the above loop terminates, read the next input symbol, and push this symbol onto
            the stack.
      5. else
         a) Read and discard the next input symbol (which should be a right parenthesis). Print the
            top operation and pop it; keep printing and popping until the next symbol on the stack
            is a left parenthesis. (If no left parenthesis is encountered, then print an error message
            indicating unbalanced parentheses, and halt.) Finally, pop the left parenthesis.
   C. while there is more of the expression to process
   D. Print and pop any remaining operations on the stack. (There should be no remaining left paren-
      theses; otherwise, the input expression did not have balanced parentheses.)
```

A sample `main` function for testing your function is shown in Figure 1. Commands to compile, link, and run this assignment are shown in Figure 2. To use the `Makefile` as distributed in class, add a target of `lab41` to `targets2srcfiles`.

```
#include <iostream>
#include <cstdlib>
#include <string>

using namespace std;

// Function infix2Postfix accepts an infix expression and converts to the
// equivalent postfix expression. The postfix expression is written to
// ostream out.
void infix2Postfix(string infix, ostream& out);

// Function hasPrecedenceGreaterThanOrEqualTo accepts two character
// parameters, each representing an arithmetic operator (+, -, *, /).
// The function returns true if operator1 has precedence greater than
// or equal to operator2 and false otherwise.
bool hasPrecedenceGreaterThanOrEqualTo(char operator1, char operator2);

int main()
{
  string infix;

  while (getline(cin, infix))
  {
    cout << "Infix:   " << infix << endl;
    cout << "Postfix: ";
    infix2Postfix(infix, cout);
    cout << endl;
  }

  return EXIT_SUCCESS;
}
```

Figure 1. `/usr/local/2336/src/lab41main.C`

```
newuser@csunix ~> cd 2336
newuser@csunix ~/2336> ./getlab.ksh 41
  * Checking to see if a folder exists for Lab 41. . .No
  * Creating a folder for Lab 41
  * Checking to see if Lab 41 has sample input and output files. . .Yes
  * Copying input and output files for Lab 41
    from folder /usr/local/2336/data/41 to folder ./41
  * Checking to see if /usr/local/2336/src/lab41main.C exists. . .Yes
  * Copying file /usr/local/2336/src/lab41main.C to folder ./41
  * Checking to see if /usr/local/2336/include/lab41.h exists. . .No
  * Copying file /usr/local/2336/src/Makefile to folder ./41
  * Adding a target of lab41 to targets2srcfiles
  * Touching file ./41/lab41.cpp
  * Edit file ./41/lab41.cpp in Notepad++
newuser@csunix ~/2336> cd 41
newuser@csunix ~/2336/41> ls
01.dat      01.out      Makefile     lab41.cpp    lab41main.C
newuser@csunix ~/2336/41> make lab41
g++ -g -Wall -std=c++11 -c lab41main.C -I/usr/local/2336/include -I.
g++ -g -Wall -std=c++11 -c lab41.cpp -I/usr/local/2336/include -I.
g++ -o lab41 lab41main.o lab41.o -L/usr/local/2336/lib -lm -lbits
newuser@csunix ~/2336/41> cat 01.dat
A+B-C
A + B * C
( A + B ) / ( C - D )
( (  A   +   B  )  *  (  C   -   D  ) + E )    /    ( F + G)
A*B+(C-D)-E
((A))
A / B + (C + D) - E
A / B + C + D - E
newuser@csunix ~/2336/41> cat 01.dat | ./lab41
Infix:   A+B-C
Postfix: A B + C -
Infix:   A + B * C
Postfix: A B C * +
Infix:   ( A + B ) / ( C - D )
Postfix: A B + C D - /
Infix:   ( (  A   +   B  )  *  (  C   -   D  ) + E )    /    ( F + G)
Postfix: A B + C D - * E + F G + /
Infix:   A*B+(C-D)-E
Postfix: A B * C D - + E -
Infix:   ((A))
Postfix: A
Infix:   A / B + (C + D) - E
Postfix: A B / C D + + E -
Infix:   A / B + C + D - E
Postfix: A B / C + D + E -
newuser@csunix ~/2336/41> cat 01.dat | ./lab41 > my.out
newuser@csunix ~/2336/41> diff 01.out my.out
newuser@csunix ~/2336/41>
```

Figure 2. Commands to Compile, Link, & Run Lab 41