```
// Class: CSE 330
// Term: Spring 2015
// Instructor: George M. Georgiou
// Name(s):Mbusi Hlatshwayo, Ryan Paglinawan
// Lab 3
// Title: Infix to postfix expression conversion

#include <iostream>
#include <string>
#include <stack>

using namespace std;

// declare functions
string infixToPostfix(string expression);

bool prec(char operator1, char operator2);

bool isOperand(char variable);

bool isOperator(char variable);

int getWeightOfOperator(char oper);

bool isRightAssociative(char oper);


string infixToPostfix(string expression) // Evaluates postfix expression, returns
output
{
   stack<char> stack1;
   string postfix = "";

   // loop through string
   for (int i = 0; i < expression.length(); i++) {
      // keep going if delimeter
      if (expression[i] == ',' || expression[i] == ' ') {
         continue;
      } else if (isOperator(expression[i])) { // // else if operator pop from stack do op
and push to stack
         while((!stack1.empty()) && (stack1.top() != '(') && prec(stack1.top(),
expression[i])) {
            postfix += stack1.top();
            stack1.pop();
         }
         stack1.push(expression[i]);
```

```
      } else if (isOperand(expression[i])) { // if operand give to postfix
         postfix += expression[i];
      } else if (expression[i] == '(') {
         stack1.push(expression[i]);
      } else if (expression[i] == ')') {
         while ((!stack1.empty()) and (stack1.top() != '(')) {
            postfix += stack1.top(); // add top of stack to postfix
            stack1.pop();
         } if (!stack1.empty()) {
            stack1.pop();
         } else {
            cout << "No '('" << endl;
         }

      }

   }

   while (!stack1.empty()) {
      postfix += stack1.top();
      stack1.pop();
   }

   return postfix;
}

bool prec(char operator1, char operator2)
{
   int opWeight = getWeightOfOperator(operator1);
   int op2Weight = getWeightOfOperator(operator2);

   // left op should be given priority
   if(opWeight == op2Weight) {
      if(isRightAssociative(operator1)) {
         return false;
      }
      else {

         return true;

      }
   }

   return opWeight > op2Weight ?  true: false;
}
```

```
bool isOperand(char variable)         // if variable is one of these cases it is operand
return value
{
   if(variable >= '0' && variable <= '9')
      return true;

   else if(variable >= 'a' && variable <= 'z')
      return true;

   else if(variable >= 'A' && variable <= 'Z')
      return true;

   else
      return false;
}

bool isOperator(char variable)
{
   if(variable == '+' || variable == '-' || variable == '*' || variable == '/' || variable ==
'$')  { // if one of these cases is operator return true else return false
      return true;
   } else {
      return false;

   }

}

int getWeightOfOperator(char oper)
{
   int weight = 0;

   switch(oper) {        // assign weight based on parameter return value

      case '+':
         weight = 1;
         break;

      case '-':
         weight = 1;
         break;

      case '*':
         weight = 2;
```

```
            break;

        case '/':
            weight = 2;
            break;

        case '$':
            weight = 3;
            break;
    }

    return weight;
}

bool isRightAssociative(char oper)
{
    if(oper == '$') {
        return true;
    } else {
        return false;
    }

}

int main()
{

    string input;
    char choice;

    do
    {
        cout << "enter infix: ";

        getline(cin, input); // get userinput

        string postfix = infixToPostfix(input);

        cout << postfix << endl;

        cout << "More input? (y/n)";

        cin >> choice;

        cin.ignore();
```

```
        cout << endl;
    }
    while (choice == 'y' or choice == 'Y'); // if user needs more input keep going

}
```