George Webster
CSCI 313
HW2—Question 2: Prefix Stack

A prefix expression is one written with the operators before the operands, unlike in between them, which is called infix. For example, the expression 2 + 4 becomes + 2 4. For computing, this has numerous advantages over an infix expression. Much like a postfix expression, it can be read without the need for parentheses, which allows it to be done without significant parsing by the program. Also like postfix, it can be done with a stack.

The simplest way to read a prefix expression is to read it backwards, as a postfix expression. The reason for this is that there are often multiple operands that need to be saved before any numbers are encountered. This can be done with a stack as well, but then keeping track of which numbers are where and which to do operations on can become extremely confusing. As a result, this program takes much of it from the Malik code for evaluating a postfix expression, with a few changes.

The first change is that the code reads the entire expression as a string. This requires that each expression in the input file to have its own line, which is a departure from Malik's postfix code. Taking the entire expression allows the program to then simply print that string into the output file without having to print each number as it goes along. One other difference is that the indicator # for a number should no longer be used. The code uses the function isdigit, from the cctype library, to determine if the character of the string is a number.

Once the string is read from the file, it is passed to the evaluateExpression function. This function accepts the entire string, and so evaluateExpression no longer takes an input stream, because it doesn't have to read anything but the string, and it no longer takes a single char. Inside evaluate expression, a for-loop starts at the end of the string, and decrements back to the front of

the string. The program is designed to ignore the character '=', because whether it's there or not, it's the end of the line that constitutes the end of the expression. The numbers are still added to the stack, and when an operand is encountered, it proceeds to the evaluateOpr function.

The function evaluateOpr is identical to the one for postfix, but with one small change. The first number popped off the stack becomes op1, and the second number becomes op2. This is because, as the string is being read backwards, the top of the stack once an operand is reached will be the leftmost number. By convention, for subtraction and division, the leftmost number is the numerator or the number being subtracted from. In postfix, the rightmost number would be on top of the stack, so it becomes op2.

Once the expression is fully evaluated, the result is written with the printResult function, which checks that the expression is still ok and that the stack, once the expression has been completed, contains only one number, the result. If it does, it gets printed to the ouput file and the loop moves on to the next expression. Once the loop is finished, the filestreams are closed.

There are myriad ways to solve a prefix arithmetic problem, but given that there was already postfix code available, it was a simple process to adapt it to a prefix expression.