CSCI 255: Lab #4 — STL: List, Stack, Queue

Darwin Jacob Groskleg

Printed: Tuesday, February 4th, 2020

Contents

Program Output & Questions

QUESTION Make a few cases for infix expressions and verify the postfix expressions generated are correct.

```
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab4-STL-List Stack Queue
gmake bin/infixToPostfix
g++ -Wall -Wextra -Wpedantic -Wdisabled-optimization -std=c++11 -stdlib=libc++ -foptimi
ze-sibling-calls -g -O0 -D_GLIBCXX_DEBUG -Iinclude -Isrc -isystem/usr/local/include -is
ystem/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr
/bin/../include/c++/v1 -isystem/Applications/Xcode.app/Contents/Developer/Toolchains/Xc
odeDefault.xctoolchain/usr/bin/../lib/clang/8.0.0/include -isystem/Applications/Xcode.a
pp/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include -isystem/usr/incl
ude  -c -o infixToPostfix.o infixToPostfix.cpp
g++ -Iinclude -Isrc -isystem/usr/local/include -isystem/Applications/Xcode.app/Contents
/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../include/c++/v1 -isystem/Appli
cations/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/../lib
/clang/8.0.0/include -isystem/Applications/Xcode.app/Contents/Developer/Toolchains/Xcod
eDefault.xctoolchain/usr/include -isystem/usr/include -Wall -Wextra -Wpedantic -Wdisabl
ed-optimization -std=c++11 -stdlib=libc++ -foptimize-sibling-calls -g -O0 -D_GLIBCXX_DE
BUG  -o bin/infixToPostfix infixToPostfix.o
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab4-STL-List Stack Queue
./bin/infixToPostfix
Infix Expression :: A*(B+C)/D
Postfix Expression :: ABC+ * D/

Infix Expression   :: 367+151/563        = 367
Postfix Expression :: 367 151 563 / +    = 367
CORRECT: postfix expression evaluates to expected result

Infix Expression   :: 2 * (3+1)/2        = 4
Postfix Expression :: 2 3 1 + * 2 /      = 4
CORRECT: postfix expression evaluates to expected result

Infix Expression   :: 51*(61+107)/461    = 18
Postfix Expression :: 51 61 107 + * 461 /        = 18
CORRECT: postfix expression evaluates to expected result

Infix Expression   :: ((15/(7-(1+1)))*3)-(2+(1+1))       = 5
Postfix Expression :: 15 7 1 1 + - / 3 * 2 1 1 + + -     = 5
CORRECT: postfix expression evaluates to expected result

Infix Expression   :: 7/7/(7+2)  = 0
Postfix Expression :: 7 7 / 7 2 + /      = 0
CORRECT: postfix expression evaluates to expected result
~/Dropbox/Documents/Terms/2019-09 - Fall/CSCI255-Fall2019/Lab4-STL-List Stack Queue
```

Figure 1: Console Sample: compiling and running with 5 different cases of infix expressions.

infixToPostfix.cpp

```
/* infixToPostfix.cpp
 * ------------------
 * CSCI 255
 * Lab #4:  STL: List, Stack, Queue (infix to postfix)
 * Authors: Darwin Jacob Groskleg, Man Lin
 * Date:    Tuesday, October 1st, 2019
 *
 * QUESTION: Make a few cases for infix expressions and verify the postfix
 *          expressions generated are correct. Submit the cases that you tested.
 *
 * PROGRAM OUTPUT
 * ------------------------------------------------------------------------------
 * Infix Expression :: A*(B+C)/D
 * Postfix Expression :: ABC+ * D/
 *
 * Infix Expression   :: 367+151/563     = 367
 * Postfix Expression :: 367 151 563 / +      = 367
 * CORRECT: postfix expression evaluates to expected result
 *
 * Infix Expression   :: 2 * (3+1)/2     = 4
 * Postfix Expression :: 2 3 1 + * 2 /      = 4
 * CORRECT: postfix expression evaluates to expected result
 *
 * Infix Expression   :: 51*(61+107)/461     = 18
 * Postfix Expression :: 51 61 107 + * 461 /      = 18
 * CORRECT: postfix expression evaluates to expected result
 *
 * Infix Expression   :: ((15/(7-(1+1)))*3)-(2+(1+1))     = 5
 * Postfix Expression :: 15 7 1 1 + - / 3 * 2 1 1 + + -      = 5
 * CORRECT: postfix expression evaluates to expected result
 *
 * Infix Expression   :: 7/7/(7+2)     = 0
 * Postfix Expression :: 7 7 / 7 2 + /      = 0
 * CORRECT: postfix expression evaluates to expected result
 * ------------------------------------------------------------------------------
 */
#include <iostream>
#include <stack>
#include <string>
#include <vector>

#include <cctype>   // for isdigit
#include <cstring>  // for strncopy

using namespace std;

int getWeight(char ch);
void infix2postfix(char infix[], char postfix[], size_t size);

double evalPostfix(string postfix_expression);
string infix2postfix(string infix);
int ctoi(char c) { return c - '0'; }
struct InfixEquation { string expression; int result; };

```

```
55 // Infix to postfix conversion in C++
56 // Input Postfix expression must be in a desired format.
57 // Operands and operator, both must be single character.
58 // Only '+'  ,  '-'  , '*', '/'  operators are expected.
59 int main() {
60     {
61         char infix[] = "A*(B+C)/D";
62         size_t size = strlen(infix);
63         char *postfix;
64         postfix = new char[size+1];
65         infix2postfix(infix, postfix, size);
66         cout << "Infix Expression :: " << infix;
67         cout << "\nPostfix Expression :: " << postfix;
68         cout << endl;
69         delete postfix;
70     }
71 
72     vector<InfixEquation> infix_expr_cases{
73         {"367+151/563",                 367 + (151/563) },
74         {"2 * (3+1)/2",                 2*(3+1)/2 },
75         {"51*(61+107)/461",             51*(61+107)/461 },
76         {"((15/(7-(1+1)))*3)-(2+(1+1))", ((15/(7-(1+1)))*3)-(2+(1+1))},
77         {"7/7/(7+2)",                   7/7/(7+2)}
78     };
79 
80     for (auto &eq : infix_expr_cases) {
81         cout << "\nInfix Expression   :: " << eq.expression
82              << "\t = " << eq.result;
83 
84         string postfix_expr = infix2postfix(eq.expression);
85         auto postfix_result = evalPostfix(postfix_expr);
86         cout << "\nPostfix Expression :: " << postfix_expr
87              << "\t = " << postfix_result;
88 
89         if (postfix_result == eq.result)
90             cout << "\nCORRECT: postfix expression evaluates to expected "
91             << "result\n";
92         else
93             cout << "\nINCORRECT: postfix expression fails to evaluate to "
94             << "expected result\n";
95     }
96 
97     return 0;
98 }
99 
100 // Calculate the result of the expression.
101 // Postfix has no need to use parentheses since there is no ambiguity.
102 //  O(n)
103 // CONSTRAINTS:
104 //  - does not handle floating point numbers in expressions
105 //  - does not handle negative numbers in expressions
106 double evalPostfix(string postfix_expression) {
107     stack<int> operand_stack;
108     int number = 0;
109     int num_width = 0;
110     for (auto &c : postfix_expression) {
```

```
111         // parse each operand (numbers) then push it to the stack
112         if (isdigit(c)) {
113             number = number * 10 + ctoi(c);
114             num_width++;
115         } else if (num_width > 0) {
116             // We know the last char was a digit.
117             operand_stack.push(number);
118 
119             // ASSUMPTION: expression will ALWAYS end in an operator, so the
120             // number (operand) will always get pushed to the stack.
121             number = num_width = 0;
122         }
123 
124         // handle operators
125         //  1. pop 2 operands off the stack,
126         //  2. perform operation,
127         //  3. place result on the stack.
128         if (getWeight(c) > 0) {
129             int left, right;
130             right = operand_stack.top(); operand_stack.pop();
131             left  = operand_stack.top(); operand_stack.pop();
132 
133             switch (c) {
134                 case '/': operand_stack.push(left / right); break;
135                 case '*': operand_stack.push(left * right); break;
136                 case '+': operand_stack.push(left + right); break;
137                 case '-': operand_stack.push(left - right); break;
138             }
139         }
140 
141         // ignore any other characters
142     }
143     return operand_stack.top();
144 }
145 
146 // Wrapper around its namesake for using string instead of cstring.
147 // Also does not mutate the infix expression argument.
148 string infix2postfix(string infix) {
149     size_t infix_size = infix.size();
150     // Add extra length for spaces to pad each number.
151     size_t pofix_size = infix_size * 2;
152 
153     // need a mutable strings
154     char* c_infix = new char[infix_size+1];
155     strncpy(c_infix, infix.c_str(), infix_size);
156     char* c_pofix = new char[pofix_size+1];
157 
158     infix2postfix(c_infix, c_pofix, infix_size);
159     string postfix{c_pofix, strlen(c_pofix)};
160 
161     delete [] c_infix;
162     delete [] c_pofix;
163     return postfix;
164 }
165 
166 // convert infix expression to postfix using a stack
```

```
167 // ADDED: spaces after end of numbers (whenever an operator is encountered
168 // SIZES: size of postfix must be at least 2 times greater than infix_size.
169 void infix2postfix(char infix[], char postfix[], size_t infix_size) {
170     stack<char> s;
171     int weight;
172     size_t i = 0;
173     size_t k = 0;
174     char ch;
175     // iterate over the infix expression
176     while (i < infix_size) {
177         ch = infix[i];
178         // strip spaces from infix expression
179         if (isspace(ch)) {
180             i++;
181             continue;
182         }
183         if (ch == '(') {
184             // simply push the opening parenthesis
185             s.push(ch);
186             i++;
187             continue;
188         }
189         if (ch == ')') {
190             // Last number before ) needs padding
191             if (isdigit(postfix[k-1])) postfix[k++] = ' ';
192 
193             // if we see a closing parenthesis,
194             // pop of all the elements and append it to
195             // the postfix expression till we encounter
196             // a opening parenthesis
197             while (!s.empty() && s.top() != '(') {
198                 postfix[k++] = s.top();
199                 s.pop();
200                 postfix[k++] = ' ';
201             }
202             // pop off the opening parenthesis also
203             if (!s.empty()) {
204                 s.pop();
205             }
206             i++;
207             continue;
208         }
209         weight = getWeight(ch);
210         if (weight == 0) {
211             // we saw an operand
212             // simply append it to postfix expression
213             postfix[k++] = ch;
214             // if last digit in expression, need to append a space
215             if (i >= infix_size-1 && isdigit(ch)) postfix[k++] = ' ';
216         } else {
217             // we saw an operator
218             // Pad the end of preceding number with a whitespace.
219             if (isdigit(postfix[k-1])) postfix[k++] = ' ';
220             if (s.empty()) {
221                 // simply push the operator onto stack if
222                 // stack is empty
```

```
223                 s.push(ch);
224             } else {
225                 // pop of all the operators from the stack and
226                 // append it to the postfix expression till we
227                 // see an operator with a lower precedence that
228                 // the current operator
229                 while (!s.empty() && s.top() != '(' &&
230                        weight <= getWeight(s.top()))
231                 {
232                     postfix[k++] = s.top();
233                     s.pop();
234                     postfix[k++] = ' ';
235                 }
236                 // push the current operator onto stack
237                 s.push(ch);
238             }
239         }
240         i++;
241     }
242     // pop of the remaining operators present in the stack
243     // and append it to postfix expression
244     while (!s.empty()) {
245         postfix[k++] = s.top();
246         s.pop();
247         postfix[k++] = ' ';
248     }
249     postfix[k] = 0; // null terminate the postfix expression
250 }
251
252 // get weight of operators as per precedence
253 // higher weight given to operators with higher precedence
254 // for non operators, return 0
255 int getWeight(char ch) {
256     // Missing case returns fall-through to the next case.
257     switch (ch) {
258         case '/':
259         case '*': return 2;
260         case '+':
261         case '-': return 1;
262         default : return 0;
263     }
264 }
```