```cpp
/*****************************************************************************
 * AUTHOR           : Nick Reardon
 * Assignment #3    : Stacks, Queues, Dequeus
 * CLASS            : CS1D
 * SECTION          : MW - 2:30p
 * DUE DATE         : 02 / 03 / 20
 *****************************************************************************/
#include "main.h"

using std::cout; using std::endl;


int main()
{

    /*
     * HEADER OUTPUT
     */
    PrintHeader(cout, "Prompt.txt");

    /*****************************************************************************/

    std::ifstream strFile;
    strFile.open("stringInput.txt");

    std::ifstream numFile;
    numFile.open("doubleInput.txt");

    std::stack<std::string> strStack_STL;
    std::stack<double> numStack_STL;

    linkedStackType<std::string> strStack;
    linkedStackType<double> numStack;

    linkedQueueType<std::string> strQueue;
    linkedQueueType<double> numQueue;

    DLinkedList<std::string> strDeque;
    DLinkedList<double> numDeque;

    cout << "Reading from file into string stacks/queue/deque" << endl;
    while (strFile)
    {
        string temp;
        getline(strFile, temp);
        if (temp != "")
        {
            strStack_STL.push(temp);
            strStack.push(temp);
            strQueue.addQueue(temp);
            strDeque.addFront(temp);
        }
```

```
53      }
54
55      cout << "Reading from file into double stacks/queue/deque" << endl;
56      while (numFile)
57      {
58          double temp = -99999999999999;
59          numFile >> temp;
60          if (temp != -99999999999999)
61          {
62              numStack_STL.push(temp);
63              numStack.push(temp);
64              numQueue.addQueue(temp);
65              numDeque.addFront(temp);
66          }
67      }
68
69      //************************************************
70
71
72      cout << endl << "   --- PART A ---" << endl << endl;
73
74
75      printStackSTL(strStack_STL);
76      printStackSTL(numStack_STL);
77
78
79      cout << endl << "   --- PART B ---" << endl << endl;
80
81
82      cout << "Deleting Jordyn from the STL stack" << endl;
83      std::string tempStr;
84      while (tempStr != "Jordyn")
85      {
86          tempStr = strStack_STL.top();
87          strStack_STL.pop();
88          cout << "deleting " << tempStr << endl;
89      }
90      cout << endl;
91
92      cout << "Deleting 200.12 from the STL stack" << endl;
93      double tempNum = -99999999999999999;
94      while (tempNum != 200.12)
95      {
96          tempNum = numStack_STL.top();
97          numStack_STL.pop();
98          cout << "deleting " << tempNum << endl;
99      }
100     cout << endl;
101
102     printStackSTL(strStack_STL);
103     printStackSTL(numStack_STL);
104
```

```
105
106     cout << endl << "   --- PART C ---" << endl << endl;
107
108
109     cout << "Printing singly linked list stacks: " << endl << endl;
110     strStack.printStack();
111     numStack.printStack();
112
113
114     cout << endl << "   --- PART D ---" << endl << endl;
115
116
117     cout << "Deleting Jordyn from the linked list stack" << endl;
118     tempStr.clear();
119     while (tempStr != "Jordyn")
120     {
121         strStack.pop(tempStr);
122     }
123     cout << endl;
124
125     cout << "Deleting 200.12 from the linked list stack" << endl;
126     tempNum = -99999999999999999;
127     while (tempNum != 200.12)
128     {
129         numStack.pop(tempNum);
130     }
131     cout << endl;
132
133     strStack.printStack();
134     numStack.printStack();
135
136
137     cout << endl << "   --- PART E ---" << endl << endl;
138
139     cout << "Printing singly linked list queues: " << endl << endl;
140
141     numQueue.printQueue();
142     strQueue.printQueue();
143
144
145     cout << endl << "   --- PART F ---" << endl << endl;
146
147
148     cout << "Deleting Jordyn from the linked list queue" << endl;
149     tempStr.clear();
150     while (tempStr != "Jordyn")
151     {
152         strQueue.deQueue(tempStr);
153     }
154     cout << endl;
155
156     cout << "Deleting 200.12 from the linked list queue" << endl;
```

```
157     tempNum = -99999999999999999;
158     while (tempNum != 200.12)
159     {
160         numQueue.deQueue(tempNum);
161     }
162     cout << endl;
163
164     numQueue.printQueue();
165     strQueue.printQueue();
166
167
168     cout << endl << "   --- PART G ---" << endl << endl;
169
170     cout << "Printing doubly linked list deques: " << endl << endl;
171
172     strDeque.printDeque();
173     numDeque.printDeque();
174
175
176     cout << endl << "   --- PART H ---" << endl << endl;
177
178
179     cout << "Deleting Jordyn, using pop front, from the doubly linked list deque"
            << endl;
180     tempStr.clear();
181     while (tempStr != "Jordyn")
182     {
183         tempStr = strDeque.front();
184         strDeque.removeFront();
185     }
186     cout << endl;
187
188     cout << "Deleting 200.12, using pop back, from the doubly linked list deque"
          << endl;
189     tempNum = -99999999999999999;
190     while (tempNum != 200.12)
191     {
192         tempNum = numDeque.back();
193         numDeque.removeBack();
194     }
195     cout << endl;
196
197     strDeque.printDeque();
198     numDeque.printDeque();
199
200
201     cout << endl << "   --- PART I ---" << endl << endl;
202
203     cout << "Testing Parentheses Algorithm - using singly linked list stack " <<
          endl << endl;
204
205     tempStr = "(12x + 6) (2x - 4)";
```

```
206     cout << endl << tempStr << endl;
207     if (areParanthesisBalanced(tempStr))
208         cout << "Balanced" << endl;
209     else
210         cout << "Not Balanced" << endl;
211
212     tempStr = "{2x + 5} (6x+4)";
213     cout << endl << tempStr << endl;
214     if (areParanthesisBalanced(tempStr))
215         cout << "Balanced" << endl;
216     else
217         cout << "Not Balanced" << endl;
218
219     tempStr = "{2x + 7) (12x + 6)";
220     cout << endl << tempStr << endl;
221     if (areParanthesisBalanced(tempStr))
222         cout << "Balanced" << endl;
223     else
224         cout << "Not Balanced" << endl;
225
226
227     tempStr = "{{8x+5) - 5x[9x+3]})";
228     cout << endl << tempStr << endl;
229     if (areParanthesisBalanced(tempStr))
230         cout << "Balanced" << endl;
231     else
232         cout << "Not Balanced" << endl;
233
234
235     tempStr = "(((4x+8) - x[4x+3])))";
236     cout << endl << tempStr << endl;
237     if (areParanthesisBalanced(tempStr))
238         cout << "Balanced" << endl;
239     else
240         cout << "Not Balanced" << endl;
241
242
243     tempStr = "[(5x - 5) - 4x[6x + 2]]";
244     cout << endl << tempStr << endl;
245     if (areParanthesisBalanced(tempStr))
246         cout << "Balanced" << endl;
247     else
248         cout << "Not Balanced" << endl;
249
250
251     tempStr = "{(8x+5) - 6x[9x+3]]";
252     cout << endl << tempStr << endl;
253     if (areParanthesisBalanced(tempStr))
254         cout << "Balanced" << endl;
255     else
256         cout << "Not Balanced" << endl;
257
```

```
258
259     system("pause");
260     return 0;
261
262
263
264
265 }
266
267 bool areParanthesisBalanced(const string& expression)
268 {
269     linkedStackType<char> s;
270     char x;
271
272     for (int i = 0; i < expression.length(); i++)
273     {
274         if (expression[i] == '(' ||
275             expression[i] == '[' ||
276             expression[i] == '{')
277         {
278             s.push(expression[i]);
279         }
280         else
281         {
282             if (!s.isEmptyStack())
283             {
284                 switch (s.top())
285                 {
286                 case '(':
287                     if (expression[i] == ')')
288                     {
289                         s.pop();
290                     }
291                     else if (expression[i] == ']' ||
292                         expression[i] == '}')
293                     {
294                         return false;
295                     }
296                     break;
297
298                 case '[':
299                     if (expression[i] == ']')
300                     {
301                         s.pop();
302                     }
303                     else if (expression[i] == ')' ||
304                         expression[i] == '}')
305                     {
306                         return false;
307                     }
308                     break;
309
```

```
310                 case '{':
311                     if (expression[i] == '}')
312                     {
313                         s.pop();
314                     }
315                     else if (expression[i] == ')' ||
316                         expression[i] == ']')
317                     {
318                         return false;
319                     }
320                     break;
321                 }
322             }
323             else
324             {
325                 if (expression[i] == ')' ||
326                     expression[i] == ']' ||
327                     expression[i] == '}')
328                 {
329                     return false;
330                 }
331             }
332         }
333     }
334     return s.isEmptyStack();
335 }
```