

```
1 // Shaun Chemplavil U08713628
2 // shaun.chemplavil@gmail.com
3 // C/C++ Programming IV : Advanced Programming with Objects
4 // 152488 Raymond L. Mitchell III
5 // hw6.cpp
6 // Win10
7 // Visual C++ 19.0
8 //
9
10 #include <iostream>
11 #include <algorithm>
12 #include <exception>
13 #include <list>
14 #include <iterator>
15
16 using namespace std;
17
18 template <typename BidirectionalIterator>
19 bool palindrome(BidirectionalIterator first, BidirectionalIterator last)
20 {
21     bool out{false};
22
23     // Check if empty list
24     if (first != last)
25     {
26         // Retain the 'end' iterator to compare in "even element" case
27         // we will be treating this as a "reverse iterator"
28         BidirectionalIterator revItr = last;
29
30         // the end element will not contain data, so decrement
31         --revItr;
32
33         // Check to see if first and last elements are equal
34         out = (*first == *revItr);
35
36         while (out && (first != revItr))
37         {
38             // Break loop if we increment to end iterator
39             // This addresses lists with even number of elements, since
40             // iterator positions are irrelevant with BidirectionalIterators
41             if (++first == last)
42                 break;
43
44             // Decrement our 'reverse iterator' for comparison
45             --revItr;
46
47             // Break the loop if we do not encounter a match
48             out = (*first == *revItr);
49         }
50     }
51     return out;
52 }
```

```
53
54 template <typename ForwardIterator, typename OutputIterator>
55 void compress(ForwardIterator first, ForwardIterator last, OutputIterator result)
56 {
57     // Check if empty list
58     if (first != last)
59     {
60         // Save the first element for comparison
61         auto prevElement = *first;
62         // Place first element into output
63         result = prevElement; ++result;
64
65         // iterate until end of container is met
66         while (++first != last)
67         {
68             // if current element is not equal to previous unique element
69             // add to output and update prevElement
70             if (*first != prevElement)
71             {
72                 *result = *first;
73                 prevElement = *first;
74                 ++result;
75             }
76         }
77     }
78 }
79
80 void testPalindromeEmptyList()
81 {
82     bool result{true}, expected{false};
83
84     list<int> testInput;
85
86     try
87     {
88         result = palindrome(testInput.begin(), testInput.end());
89         if (result == expected)
90             clog << "testPalindromeEmptyList PASSED\n";
91         else
92             clog << "testPalindromeEmptyList FAILED: Expected " << expected
93                 << " got : " << result << "\n";
94     }
95     catch (...)
96     {
97         clog << "testPalindromeEmptyList FAILED\n";
98     }
99 }
100
101 void testPalindromeOddTrue()
102 {
103     bool result{false}, expected{true};
104 }
```

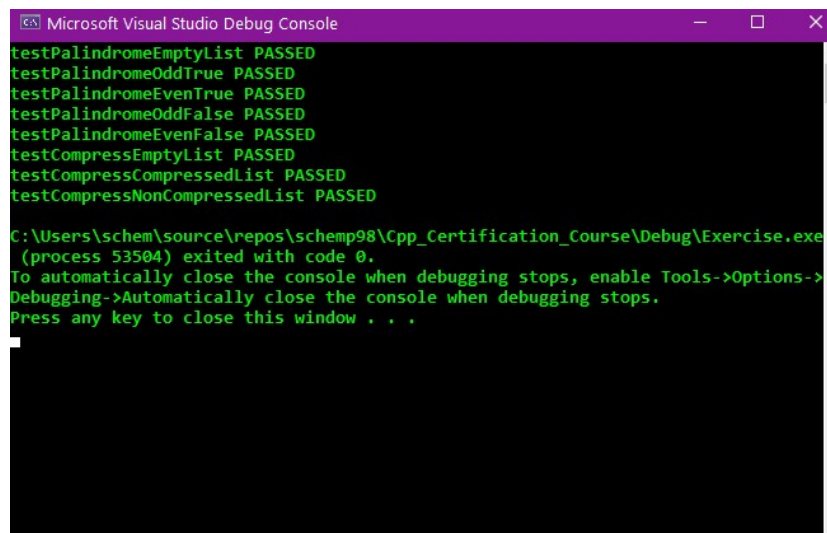
```
105     int data[] = {1, 2, 3, 4, 3, 2, 1};
106     list<int> testInput(data, data + 7);
107     try
108     {
109         result = palindrome(testInput.begin(), testInput.end());
110         if (result == expected)
111             clog << "testPalindromeOddTrue PASSED\n";
112         else
113             clog << "testPalindromeOddTrue FAILED: Expected " << expected
114                 << " got : " << result << "\n";
115     }
116     catch (...)
117     {
118         clog << "testPalindromeOddTrue FAILED\n";
119     }
120 }
121
122 void testPalindromeEvenTrue()
123 {
124     bool result{false}, expected{true};
125     int data[] = {1, 2, 3, 3, 2, 1};
126     list<int> testInput(data, data + 6);
127     try
128     {
129         result = palindrome(testInput.begin(), testInput.end());
130         if (result == expected)
131             clog << "testPalindromeEvenTrue PASSED\n";
132         else
133             clog << "testPalindromeEvenTrue FAILED: Expected " << expected
134                 << " got : " << result << "\n";
135     }
136     catch (...)
137     {
138         clog << "testPalindromeEvenTrue FAILED\n";
139     }
140 }
141
142 void testPalindromeOddFalse()
143 {
144     bool result{true}, expected{false};
145     int data[] = {1, 2, 3, 4, 5, 6, 7};
146     list<int> testInput(data, data + 7);
147     try
148     {
149         result = palindrome(testInput.begin(), testInput.end());
150         if (result == expected)
151             clog << "testPalindromeOddFalse PASSED\n";
152         else
153             clog << "testPalindromeOddFalse FAILED: Expected " << expected
154                 << " got : " << result << "\n";
155     }
156     catch (...)
```

```
157     {
158         clog << "testPalindromeOddFalse FAILED\n";
159     }
160 }
161
162 void testPalindromeEvenFalse()
163 {
164     bool result{true}, expected{false};
165     int data[] = {1, 2, 3, 4, 5, 6};
166     list<int> testInput(data, data + 6);
167     try
168     {
169         result = palindrome(testInput.begin(), testInput.end());
170         if (result == expected)
171             clog << "testPalindromeEvenFalse PASSED\n";
172         else
173             clog << "testPalindromeEvenFalse FAILED: Expected " << expected
174                 << " got : " << result << "\n";
175     }
176     catch (...)
177     {
178         clog << "testPalindromeEvenFalse FAILED\n";
179     }
180 }
181
182 void testCompressEmptyList()
183 {
184     list<int> testInput, result;
185     try
186     {
187         compress(testInput.begin(), testInput.end(), back_inserter(result));
188         if (result.empty())
189             clog << "testCompressEmptyList PASSED\n";
190         else
191             clog << "testCompressEmptyList FAILED: Expected Empty List output!\n";
192     }
193     catch (...)
194     {
195         clog << "testCompressEmptyList FAILED\n";
196     }
197 }
198 void testCompressCompressedList()
199 {
200     int data[] = {1, 1, 2, 2, 1, 1}, expectData[] = {1,2,1};
201     list<int> testInput(data, data + 6), expectedResult(expectData, expectData + 3),
202         result;
203
204     try
205     {
206         compress(testInput.begin(), testInput.end(), back_inserter(result));
207         if (result == expectedResult)
```

```

208     clog << "testCompressCompressedList PASSED\n";
209     else
210         clog << "testCompressCompressedList FAILED: Unexpected Result!\n";
211 }
212 catch (...)
213 {
214     clog << "testCompressCompressedList FAILED\n";
215 }
216 }
217 void testCompressNonCompressedList()
218 {
219     int data[] = {1, 2, 3, 1, 2, 3}, expectData[] = {1, 2, 3, 1, 2, 3};
220     list<int> testInput(data, data + 6), expectedResult(expectData, expectData +
221     6),
222     result;
223     try
224     {
225         compress(testInput.begin(), testInput.end(), back_inserter(result));
226         if (result == expectedResult)
227             clog << "testCompressNonCompressedList PASSED\n";
228         else
229             clog << "testCompressNonCompressedList FAILED: Unexpected Result!\n";
230     }
231     catch (...)
232     {
233         clog << "testCompressNonCompressedList FAILED\n";
234     }
235 }
236 int main(void)
237 {
238     testPalindromeEmptyList();           //1 a)
239     testPalindromeOddTrue();             //1 b)
240     testPalindromeEvenTrue();            //1 c)
241     testPalindromeOddFalse();            //1 d)
242     testPalindromeEvenFalse();           //1 e)
243
244     testCompressEmptyList();             //2 a)
245     testCompressCompressedList();        //2 b)
246     testCompressNonCompressedList();     //2 c)
247 }

```



```

Microsoft Visual Studio Debug Console
testPalindromeEmptyList PASSED
testPalindromeOddTrue PASSED
testPalindromeEvenTrue PASSED
testPalindromeOddFalse PASSED
testPalindromeEvenFalse PASSED
testCompressEmptyList PASSED
testCompressCompressedList PASSED
testCompressNonCompressedList PASSED

C:\Users\schem\source\repos\schemp98\Cpp_Certification_Course\Debug\Exercise.exe
(process 53504) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->
Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .

```