```cpp
// Shaun Chemplavil U08713628
// shaun.chemplavil@gmail.com
// C/C++ Programming IV : Advanced Programming with Objects
// 152488 Raymond L. Mitchell III
// hw6.cpp
// Win10
// Visual C++ 19.0
//

#include <iostream>
#include <algorithm>
#include <exception>
#include <list>
#include <iterator>

using namespace std;

template <typename BidirectionalIterator>
bool palindrome(BidirectionalIterator first, BidirectionalIterator last)
{
   bool out{false};

   // Check if empty list
   if (first != last)
   {
      // Retain the 'end' iterator to compare in "even element" case
      // we will be treating this as a "reverse iterator"
      BidirectionalIterator revItr = last;

      // the end element will not contain data, so decrement
      --revItr;

      //  Check to see if first and last elements are equal
      out = (*first == *revItr);

      while (out && (first != revItr))
      {
         // Break loop if we increment to end iterator
         // This addresses lists with even number of elements, since
         // iterator positions are irrelevant with BidirectionalIterators
         if (++first == last)
            break;

         // Decrement our 'reverse iterator' for comparison
         --revItr;

         // Break the loop if we do not encounter a match
         out = (*first == *revItr);
      }
   }
   return out;
}
```

```cpp

template <typename ForwardIterator, typename OutputIterator>
void compress(ForwardIterator first, ForwardIterator last, OutputIterator result)
{
   // Check if empty list
   if (first != last)
   {
      // Save the first element for comparison
      auto prevElement = *first;
      // Place first element into output
      result = prevElement; ++result;

      // iterate until end of container is met
      while (++first != last)
      {
         // if current element is not equal to previous unique element
         // add to output and update prevElement
         if (*first != prevElement)
         {
            *result = *first;
            prevElement = *first;
            ++result;
         }
      }
   }
}

void testPalindromeEmptyList()
{
   bool result{true}, expected{false};

   list<int> testInput;

   try
   {
      result = palindrome(testInput.begin(), testInput.end());
      if (result == expected)
         clog << "testPalindromeEmptyList PASSED\n";
      else
         clog << "testPalindromeEmptyList FAILED: Expected " << expected
         << " got : " << result << "\n";
   }
   catch (...)
   {
      clog << "testPalindromeEmptyList FAILED\n";
   }
}

void testPalindromeOddTrue()
{
   bool result{false}, expected{true};

```

```
   int data[] = {1, 2, 3, 4, 3, 2, 1};
   list<int> testInput(data, data + 7);
   try
   {
      result = palindrome(testInput.begin(), testInput.end());
      if (result == expected)
         clog << "testPalindromeOddTrue PASSED\n";
      else
         clog << "testPalindromeOddTrue FAILED: Expected " << expected
         << " got : " << result << "\n";
   }
   catch (...)
   {
      clog << "testPalindromeOddTrue FAILED\n";
   }
}

void testPalindromeEvenTrue()
{
   bool result{false}, expected{true};
   int data[] = {1, 2, 3,  3, 2, 1};
   list<int> testInput(data, data + 6);
   try
   {
      result = palindrome(testInput.begin(), testInput.end());
      if (result == expected)
         clog << "testPalindromeEvenTrue PASSED\n";
      else
         clog << "testPalindromeEvenTrue FAILED: Expected " << expected
         << " got : " << result << "\n";
   }
   catch (...)
   {
      clog << "testPalindromeEvenTrue FAILED\n";
   }
}

void testPalindromeOddFalse()
{
   bool result{true}, expected{false};
   int data[] = {1, 2, 3, 4, 5, 6, 7};
   list<int> testInput(data, data + 7);
   try
   {
      result = palindrome(testInput.begin(), testInput.end());
      if (result == expected)
         clog << "testPalindromeOddFalse PASSED\n";
      else
         clog << "testPalindromeOddFalse FAILED: Expected " << expected
         << " got : " << result << "\n";
   }
   catch (...)
```

```
   {
      clog << "testPalindromeOddFalse FAILED\n";
   }
}

void testPalindromeEvenFalse()
{
   bool result{true}, expected{false};
   int data[] = {1, 2, 3, 4, 5, 6};
   list<int> testInput(data, data + 6);
   try
   {
      result = palindrome(testInput.begin(), testInput.end());
      if (result == expected)
         clog << "testPalindromeEvenFalse PASSED\n";
      else
         clog << "testPalindromeEvenFalse FAILED: Expected " << expected
         << " got : " << result << "\n";
   }
   catch (...)
   {
      clog << "testPalindromeEvenFalse FAILED\n";
   }
}

void testCompressEmptyList()
{
   list<int> testInput, result;
   try
   {
      compress(testInput.begin(), testInput.end(), back_inserter(result));
      if (result.empty())
         clog << "testCompressEmptyList PASSED\n";
      else
         clog << "testCompressEmptyList FAILED: Expected Empty List output!\n";
   }
   catch (...)
   {
      clog << "testCompressEmptyList FAILED\n";
   }
}
void testCompressCompressedList()
{
   int data[] = {1, 1, 2, 2, 1, 1}, expectData[] = {1,2,1};
   list<int> testInput(data, data + 6), expectedResult(expectData, expectData + 3),
      result;

   try
   {
      compress(testInput.begin(), testInput.end(), back_inserter(result));
      if (result == expectedResult)
```

```
            clog << "testCompressCompressedList PASSED\n";
         else
            clog << "testCompressCompressedList FAILED: Unexpected Result!\n";
      }
      catch (...)
      {
         clog << "testCompressCompressedList FAILED\n";
      }
   }
   void testCompressNonCompressedList()
   {
      int data[] = {1, 2, 3, 1, 2, 3}, expectData[] = {1, 2, 3, 1, 2, 3};
      list<int> testInput(data, data + 6), expectedResult(expectData, expectData +
       6),
         result;
      try
      {
         compress(testInput.begin(), testInput.end(), back_inserter(result));
         if (result == expectedResult)
            clog << "testCompressNonCompressedList PASSED\n";
         else
            clog << "testCompressNonCompressedList FAILED: Unexpected Result!\n";
      }
      catch (...)
      {
         clog << "testCompressNonCompressedList FAILED\n";
      }
   }
   int main(void)
   {
      testPalindromeEmptyList();       //1 a)
      testPalindromeOddTrue();         //1 b)
      testPalindromeEvenTrue();        //1 c)
      testPalindromeOddFalse();        //1 d)
      testPalindromeEvenFalse();       //1 e)

      testCompressEmptyList();         //2 a)
      testCompressCompressedList();    //2 b)
      testCompressNonCompressedList(); //2 c)
   }

```