```cpp
/*
    Li, Gellert

    CS A250
    28th April, 2018

    Lab 11
*/

#include <iostream>
#include <string>
#include <vector>
#include <list>

using namespace std;

// Declaration function printVector.
// The function passes a vector and prints all
// the elements on one line, separated by a space.
// Use an iterator and a FOR loop.
void printVector(const vector<int> &v);

// Declaration function printList.
// The function passes a list and prints all
// the elements on one line, separated by a space.
// Use an iterator and a WHILE loop.
void printList(const list<int> &aList);


int main()
{

    /*********************************************************************
            VECTORS
    *********************************************************************/
    cout << "  ***  STL VECTOR CLASS  ***  \n\n";

    // Use the default constructor to declare an integer vector v1.
    vector<int> v1;

    // void push_back (const value_type& val);
    // Use function push_back to insert the following values in v1: 12, 73, 41,
    // 38, 25, 56, an 63 in this order.

    v1.push_back(12);
    v1.push_back(73);
    v1.push_back(41);
    v1.push_back(38);
    v1.push_back(25);
    v1.push_back(56);
    v1.push_back(63);

```

```
53      // size_type size() const noexcept;
54      // Create a variable of type int named sizeV1 and store the size of the
          vector.
55      // Use function size to retrieve the size of the vector.
56      // Make sure you cast the return value of the function size to the
          appropriate type.
57      int sizeV1 = v1.size();
58
59      // Use a FOR loop to print out the vector.
60      // Do NOT use an iterator.
61      for (auto i : v1) {
62          cout << i << " ";
63      }
64      cout << endl;
65
66      //void clear() noexcept;
67      // Call the function clear on vector v1.
68      v1.clear();
69
70      // size_type size() const noexcept;
71      // Call function size to print the size of v1.
72      cout << v1.size() << endl;
73
74      // size_type capacity() const noexcept;
75      // Call function capacity to output the capacity of v1.
76      cout << v1.capacity() << endl;
77
78      // Create an array of integers containing: 10,11,12,13,14,15,16,17,18,19
79      int arr[] = { 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 };
80
81      // Use the default constructor to declare an integer vector v2.
82      vector<int> v2;
83
84      // void assign (InputIterator first, InputIterator last);
85      // Use function assign to copy elements 12, 13, 14, 15, and 16 in v2.
86      // One statement only.
87      v2.assign(arr + 2, arr + 7);
88
89      // Call the function printVector to print v2.
90      printVector(v2);
91
92      // const_reference back() const;
93      // Use the function back output the last element in the vector
94      // (Notice that the back function returns a reference.)
95      cout << v2.back() << endl;
96
97      // Use the default constructor to declare an integer vector v3.
98      vector<int> v3;
99
100     // void assign (size_type n, const value_type& val);
101     // Use function assign to insert the values 7, 7, 7, 7, and 7.
102     // One statement only.
```

```
103     v3.assign(5, 7);
104
105     // Call the function printVector  to print v3.
106     printVector(v3);
107
108     // const_reference at(size_type n) const;
109     // Use function at to replace the middle element with 100.
110     // (Notice that the at function returns a reference.)
111     v3.at(v3.size() / 2) = 100;
112
113     // Call the function printVector to print v3.
114     printVector(v3);
115
116     // vector (const vector& x);
117     // Use the copy constructor to create a new vector v4 with the
118     // same elements of v3.
119     vector<int> v4 = v3;
120
121     // Call the function printVector to print v4.
122     printVector(v4);
123
124     // Create an iterator iterVector4 to point to the first element of v4.
125     vector<int>::iterator iterVector4 = v4.begin();
126
127     // Create an iterator iterVector2 to point to the second element of v2.
128     vector<int>::iterator iterVector2 = v2.begin() + 1;
129
130     // iterator insert (const_iterator position, InputIterator first,
          InputIterator last);
131     // Use function insert to insert the second, third, and fourth element
132     // of v2 as the first, second, and third element of v4.
133     // (Notice that the insert function returns an iterator,
134     //   but if we do not intend to use it, we can ignore it.)
135     v4.insert(iterVector4, iterVector2, iterVector2 + 3);
136
137     // Call the function printVector to print v4.
138     printVector(v4);
139
140     // iterator insert (const_iterator position, size_type n, const value_type&
          val);
141     // Use the function insert to insert three 0s at the end of v4.
142     // (Notice that the insert function returns an iterator,
143     //   but if we do not intend to use it, we can ignore it.)
144     v4.insert(v4.end(), 3, 0);
145
146     // Call the function printVector to print v4.
147     printVector(v4);
148
149     // bool empty() const noexcept;
150     // const_reference back() const;
151     // void pop_back();
152     // Use a WHILE loop to remove and output each element of v2 backwards.
```

```
153     // Use function empty for the loop condition, function back to output
154     // the last element, and function pop_back to remove elements.
155     // (Notice that the insert function returns an iterator,
156     //   but if we do not intend to use it, we can ignore it.)
157     while (!v2.empty()) {
158         cout << v2.back() << " ";
159         v2.pop_back();
160     }
161     cout << endl;
162
163     // void resize (size_type n, const value_type& val);
164     // Use function resize to insert three times number 4 in v2.
165     v2.resize(3, 4);
166
167     // Call the function printVector to print v2.
168     printVector(v2);
169
170     // const_reference front() const;
171     // Use function front to output the first element in v4.
172     // (Notice that the front function returns a reference.)
173     cout << v4.front() << endl;
174
175     // void swap (vector& x);
176     // Use function swap to swap v2 with v4.
177     v2.swap(v4);
178
179     // Call the function printVector to print v2.
180     printVector(v2);
181
182     // Create a new vector v5;
183     vector<int> v5;
184
185     // Use the overloaded assignment operator to copy all the elements of v2
186     // into v5.
187     v5 = v2;
188
189     // void resize (size_type n);
190     // size_type size() const noexcept;
191     // Delete the last element of v5 by using the functions resize and size
192     v5.resize(v5.size() - 1);
193
194     // Call the function printVector to print v5.
195     printVector(v5);
196
197     // Create an iterator iterVector5 to point to the first element of v5.
198     vector<int>::iterator iterVector5 = v5.begin();
199
200     // iterator erase (const_iterator first, const_iterator last);
201     // size_type size() const noexcept;
202     // Call the function erase to delete the second half of v5.
203     // Use function size to get the range.
204     // (Notice that the insert function returns an iterator,
```

```
205     //   but if we do not intend to use it, we can ignore it.)
206     v5.erase(iterVector5 + v5.size() / 2 , iterVector5 + v5.size());
207
208     // Call the function printVector to print v5 again.
209     printVector(v5);
210
211     // iterator erase (const_iterator position);
212     // Call the function erase to delete the first element of the vector.
213     // (Notice that the insert function returns an iterator,
214     //   but if we do not intend to use it, we can ignore it.)
215     v5.erase(iterVector5, iterVector5 + 1);
216
217     // Call the function printVector to print v5 again.
218     printVector(v5);
219
220     // Create a vector of integers named v6 containing numbers from 100 to 105.
221     // Using the copy constructor, create a vector named v7, a copy of v6.
222     vector<int> v6 = { 100, 101, 102, 103, 104, 105 };
223     vector<int> v7 = v6;
224
225     // iterator erase (const_iterator position);
226     // iterator insert (const_iterator position, const value_type& val);
227     // Erase element 103 from v7 and insert element 333 in its plase,
228     // by using an iterator.
229     // Note that the function erase returns an iterator that can be used
230     // to insert 333 in the right position.
231     vector<int>::iterator iterV7 = v7.begin();
232     v7.insert(v7.erase(iterV7 + 3), 333);
233
234     // Using a range-based FOR loop, print v7.
235     for (auto i : v7) {
236         cout << i << " ";
237     }
238     cout << endl;
239
240     /*****************************************************************************
241            LISTS
242     *****************************************************************************/
243
244     cout << "\n\n-----------------------------------------------------";
245     cout << "\n  ***  STL LIST CLASS  ***  \n\n\n";
246
247     // Use the default constructor to create three lists of integers, intLis1,
248     // intList2, and intList3.
249     list<int> intList1, intList2, intList3;
250
251
252     // void push_back (const value_type& val);
253     // Use the function push_back to insert the following values in the first
        list:
254     // 23 58 58 58 36 15 15 93 98 58
255     intList1.push_back(23);
```

```
256     intList1.push_back(58);
257     intList1.push_back(58);
258     intList1.push_back(58);
259     intList1.push_back(36);
260     intList1.push_back(15);
261     intList1.push_back(15);
262     intList1.push_back(93);
263     intList1.push_back(98);
264     intList1.push_back(58);
265
266     // Call function printList to print intList1.
267     printList(intList1);
268
269     // Using the overloaded assignment operator, copy elements of intList1 and
          intList2.
270     intList2 = intList1;
271
272     // Call function printList to print intList2.
273     printList(intList2);
274
275     // void unique();
276     // Using function unique, remove all consecutive duplicates in the first
          list.
277     intList1.unique();
278
279     // Call function printList to print intList1.
280     printList(intList1);
281
282     // void sort();
283     // Using function sort, sort all elements in the second list.
284     // (Notice that the function sort can be used only if there are no
          duplicates.)
285     intList2.sort();
286
287     // Call function printList to print intList2.
288     printList(intList2);
289
290     // void push_back (const value_type& val);
291     //Insert the following elements in the third list:
292     //  13 23 25 136 198
293     intList3.push_back(13);
294     intList3.push_back(23);
295     intList3.push_back(25);
296     intList3.push_back(136);
297     intList3.push_back(198);
298
299     // Call function printList to print intList3.
300     printList(intList3);
301
302     // void merge (list& x);
303     // Add to the second list all elements of the third list(browse the
304     //  list of functions in cplusplus.com to figure out which function
```

```
305     //  you need to use).
306     // --> This is ONE statement only.
307     intList2.merge(intList3);
308
309     // Call function printList to print intList2.
310     printList(intList2);
311
312
313     /*********************************************************************
314     *       Create statements using the functions below.
315     *       Is the output what you expected?
316     *********************************************************************/
317
318     cout << "\n(The next output section is determined by your implementation.)\n\n";
319
320     // void assign (size_type n, const value_type& val);
321
322     // void assign (InputIterator first, InputIterator last);
323
324     // const_reference back() const;
325     // (Notice that this back function returns a reference.)
326
327     // void clear() noexcept;
328
329     // bool empty() const noexcept;
330
331     // const_reference front() const;
332
333     // iterator insert (const_iterator position, const value_type& val);
334     // (Notice that the insert function returns an iterator.)
335
336     // void pop_back();
337
338     // void pop_front();
339
340     // void push_front (const value_type& val);
341
342     // void remove (const value_type& val);
343
344     // void reverse() noexcept;
345
346     // void splice (const_iterator position, list& x);
347
348     // void splice (const_iterator position, list& x, const_iterator i);
349
350     // void splice (const_iterator position, list& x, const_iterator first,
        const_iterator last);
351
352     // void swap (list& x);
353
354
```

```
355     cout << "\n\n-----------------------------------------------------";
356
357     cout << endl;
358     system("Pause");
359     return 0;
360 }
361
362 // Definition function printVector
363 void printVector(const vector<int> &v) {
364     vector<int>::const_iterator iter = v.cbegin();
365     vector<int>::const_iterator iterEnd = v.cend();
366
367     for (iter; iter != iterEnd; ++iter) {
368         cout << *iter << " ";
369     }
370     cout << endl;
371 }
372
373 // Definition function printList
374 void printList(const list<int> &aList) {
375     list<int>::const_iterator iter = aList.cbegin();
376     list<int>::const_iterator iterEnd = aList.cend();
377     while (iter != iterEnd) {
378         cout << *iter << " ";
379         ++iter;
380     }
381     cout << endl;
382 }
```