

```
1  /*
2      CodeLovers
3
4      Nguyen Da
5      Ton, An
6      Banh, Alex
7
8      CS A250
9      April 28, 2018
10
11      Lab 11
12  */
13
14  #include <iostream>
15  #include <string>
16  #include <vector>
17  #include <list>
18
19  using namespace std;
20
21  // Declaration function printVector.
22  // The function passes a vector and prints all
23  // the elements on one line, separated by a space.
24  // Use an iterator and a FOR loop.
25  void printVector(const vector<int> &v);
26
27  // Declaration function printList.
28  // The function passes a list and prints all
29  // the elements on one line, separated by a space.
30  // Use an iterator and a WHILE loop.
31  void printList(const list<int> &l);
32
33
34  int main()
35  {
36
37      /*****
38          VECTORS
39          *****/
40      cout << "   *** STL VECTOR CLASS   ***   \n\n";
41
42      // Use the default constructor to declare an integer vector v1.
43      vector<int> v1;
44
45      // void push_back (const value_type& val);
46      // Use function push_back to insert the following values in v1: 12, 73, 41,
47      // 38, 25, 56, an 63 in this order.
48      v1.push_back(12);
49      v1.push_back(73);
50      v1.push_back(41);
51      v1.push_back(38);
52      v1.push_back(25);
```

```
53     v1.push_back(56);
54     v1.push_back(63);
55
56     // size_type size() const noexcept;
57     // Create a variable of type int named sizeV1 and store the size of the vector.
58     // Use function size to retrieve the size of the vector.
59     // Make sure you cast the return value of the function size to the appropriate type.
60     int sizeV1 = static_cast<int>(v1.size());
61
62     // Use a FOR loop to print out the vector.
63     // Do NOT use an iterator.
64     for (int i = 0; i < sizeV1; ++i)
65         cout << v1[i] << " ";
66
67     //for (auto e : v1)
68     //    cout << e << " ";
69
70     cout << endl;
71
72     //void clear() noexcept;
73     // Call the function clear on vector v1.
74     v1.clear();
75
76     // size_type size() const noexcept;
77     // Call function size to print the size of v1.
78     cout << v1.size() << endl;
79
80     // size_type capacity() const noexcept;
81     // Call function capacity to output the capacity of v1.
82     cout << v1.capacity() << endl;
83
84     // Create an array of integers containing: 10,11,12,13,14,15,16,17,18,19
85     int arr[] = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19};
86
87     // Use the default constructor to declare an integer vector v2.
88     vector<int> v2;
89
90     // void assign (InputIterator first, InputIterator last);
91     // Use function assign to copy elements 12, 13, 14, 15, and 16 in v2.
92     // One statement only.
93     v2.assign(arr + 2, arr + 7);
94
95     // Call the function printVector to print v2.
96     printVector(v2);
97
98     // const_reference back() const;
99     // Use the function back output the last element in the vector
100    // (Notice that the back function returns a reference.)
101    cout << v2.back() << endl;
102
```

```
103 // Use the default constructor to declare an integer vector v3.
104 vector<int> v3;
105
106 // void assign (size_type n, const value_type& val);
107 // Use function assign to insert the values 7, 7, 7, and 7.
108 // One statement only.
109 v3.assign(5, 7);
110
111 // Call the function printVector to print v3.
112 printVector(v3);
113
114 // const_reference at(size_type n) const;
115 // Use function at to replace the middle element with 100.
116 // (Notice that the at function returns a reference.)
117 v3.at(2) = 100;
118
119 // Call the function printVector to print v3.
120 printVector(v3);
121
122 // vector (const vector& x);
123 // Use the copy constructor to create a new vector v4 with the
124 // same elements of v3.
125 vector<int> v4(v3);
126
127 // Call the function printVector to print v4.
128 printVector(v4);
129
130 // Create an iterator iterVector4 to point to the first element of v4.
131 vector<int>::iterator iterVector4 = v4.begin();
132
133 // Create an iterator iterVector2 to point to the second element of v2.
134 vector<int>::iterator iterVector2 = v2.begin() + 1;
135
136 // iterator insert (const_iterator position, InputIterator first,
137 //                  InputIterator last);
138 // Use function insert to insert the second, third, and fourth element
139 // of v2 as the first, second, and third element of v4.
140 // (Notice that the insert function returns an iterator,
141 // but if we do not intend to use it, we can ignore it.)
142 v4.insert(iterVector4, iterVector2, iterVector2 + 3);
143
144 // Call the function printVector to print v4.
145 printVector(v4);
146
147 // iterator insert (const_iterator position, size_type n, const value_type&
148 //                  val);
149 // Use the function insert to insert three 0s at the end of v4.
150 // (Notice that the insert function returns an iterator,
151 // but if we do not intend to use it, we can ignore it.)
152 v4.insert(v4.end(), 3, 0);
153
154 // Call the function printVector to print v4.
```

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

```

205 // Create an iterator iterVector5 to point to the first element of v5.
206 vector<int>::iterator iterVector5 = v5.begin();
207
208 // iterator erase (const_iterator first, const_iterator last);
209 // size_type size() const noexcept;
210 // Call the function erase to delete the second half of v5.
211 // Use function size to get the range.
212 // (Notice that the insert function returns an iterator,
213 // but if we do not intend to use it, we can ignore it.)
214 v5.erase(iterVector5 + (v5.size()/2), v5.end());
215
216 // Call the function printVector to print v5 again.
217 printVector(v5);
218
219 // iterator erase (const_iterator position);
220 // Call the function erase to delete the first element of the vector.
221 // (Notice that the insert function returns an iterator,
222 // but if we do not intend to use it, we can ignore it.)
223 v5.erase(iterVector5);
224
225 // Call the function printVector to print v5 again.
226 printVector(v5);
227
228 // Create a vector of integers named v6 containing numbers from 100 to 105.
229 // Using the copy constructor, create a vector named v7, a copy of v6.
230 vector<int> v6 = {100, 101, 102, 103, 104, 105};
231 vector<int> v7(v6);
232
233 // iterator erase (const_iterator position);
234 // iterator insert (const_iterator position, const value_type& val);
235 // Erase element 103 from v7 and insert element 333 in its place,
236 // by using an iterator.
237 // Note that the function erase returns an iterator that can be used
238 // to insert 333 in the right position.
239 v7.insert(v7.erase(v7.begin() + 3), 333);
240
241 // Using a range-based FOR loop, print v7.
242 int v7Size = static_cast<int>(v7.size());
243
244 for (int i = 0; i < v7Size; ++i)
245     cout << v7[i] << " ";
246
247 /*****
248     LISTS
249 *****/
250
251 cout << "\n\n-----";
252 cout << "\n *** STL LIST CLASS *** \n\n\n";
253
254 // Use the default constructor to create three lists of integers, intList1,
255 // intList2, and intList3.
256 list<int> intList1, intList2, intList3;

```

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

```
305 // Call function printList to print intList3.
306 printList(intList3);
307
308 // void merge (list& x);
309 // Add to the second list all elements of the third list(browse the
310 // list of functions in cplusplus.com to figure out which function
311 // you need to use).
312 // --> This is ONE statement only.
313 intList2.merge(intList3);
314
315 // Call function printList to print intList2.
316 printList(intList2);
317
318
319 /*****
320 *      Create statements using the functions below.
321 *      Is the output what you expected?
322 *****/
323
324 cout << "\n(The next output section is determined by your implementation.)\n" &
    "\n";
325
326 vector<int> v;
327
328 // void assign (size_type n, const value_type& val);
329 v.assign(10, 9);
330 printVector(v);
331
332 // void assign (InputIterator first, InputIterator last);
333 int myArr[] = {21, 22, 23, 24, 25, 26, 27, 28, 29};
334 v.assign(myArr + 1, myArr + 5);
335 printVector(v);
336
337 // const_reference back() const;
338 // (Notice that this back function returns a reference.)
339 cout << v.back() << endl;
340
341 // void clear() noexcept;
342 v.clear();
343 cout << "Size: " << v.size() << endl;
344 cout << "Capacity: " << v.capacity() << endl;
345
346 // bool empty() const noexcept;
347 list<int> l = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
348 printList(l);
349
350 // const_reference front() const;
351 cout << l.front() << endl;
352
353 // iterator insert (const_iterator position, const value_type& val);
354 // (Notice that the insert function returns an iterator.)
355 vector<int> vec1 = {11, 12, 13, 14, 15, 16, 17, 18, 19, 20};
```

```
356     printVector(vec1);
357
358     vec1.insert(vec1.begin() + 6, 600);
359     printVector(vec1);
360
361     // void pop_back();
362     for (int i = 0; i < 4; ++i)
363     {
364         vec1.pop_back();
365     }
366
367     printVector(vec1);
368
369     // void pop_front();
370     for (int i = 1; i < 4; ++i)
371     {
372         l.pop_front();
373     }
374
375     printList(l);
376
377     // void push_front (const value_type& val);
378     l.push_front(500);
379     printList(l);
380
381     // void remove (const value_type& val);
382     l.remove(8);
383     printList(l);
384
385     // void reverse() noexcept;
386     l.reverse();
387     printList(l);
388
389     // void splice (const_iterator position, list& x);
390     list<int> l2 = { 101, 105, 108, 106, 109, 103 };
391
392     list<int>::const_iterator pos = l.cbegin();
393     ++pos;
394     ++pos;
395
396     l.splice(pos, l2);
397     printList(l);
398
399     // void splice (const_iterator position, list& x, const_iterator i);
400     l2.splice(l2.cbegin(), l, pos);
401     printList(l2);
402
403     // void splice (const_iterator position, list& x, const_iterator first,
404                   // const_iterator last);
405     l.splice(l.cbegin(), l2, l2.begin(), l2.end());
406     printList(l);
```



```
407     // void swap (list& x);
408     list<int> l3 = { 202, 204, 203 };
409
410     l.swap(l3);
411     printList(l);
412
413     cout << "\n\n-----";
414
415     cout << endl;
416     system("Pause");
417     return 0;
418 }
419
420 // Definition function printVector
421 void printVector(const vector<int> &v)
422 {
423     vector<int>::const_iterator iter = v.cbegin();
424     vector<int>::const_iterator iterEnd = v.cend();
425
426     for (iter; iter != iterEnd; ++iter)
427         cout << *iter << " ";
428
429     cout << endl;
430 }
431
432 // Definition function printlist
433 void printList(const list<int> &l)
434 {
435     list<int>::const_iterator iter = l.cbegin();
436     list<int>::const_iterator iterEnd = l.cend();
437
438     while (iter != iterEnd)
439     {
440         cout << *iter << " ";
441         ++iter;
442     }
443
444     cout << endl;
445 }
```