

```
1  /*
2      Salcedo, Salvador
3
4      CS A250
5      April 6, 2019
6
7      Lab 9
8  */
9
10 #include <iostream>
11 #include <string>
12 #include <vector>
13 #include <list>
14 #include <set>
15 #include <map>
16
17 using namespace std;
18
19 // Declaration function printVector.
20 // The function passes a vector and prints all
21 // the elements on one line, separated by a space.
22 // Use an iterator and a FOR loop.
23 void printVector(const vector<int>& a);
24
25 // Declaration function printList.
26 // The function passes a list and prints all
27 // the elements on one line, separated by a space.
28 // Use an iterator and a WHILE loop.
29 void printList(const list<int>& a);
30
31 // Declaration function printSet.
32 // The function passes a list and prints all
33 // the elements on one line, separated by a space.
34 // Use a range-based FOR loop. If you do not know
35 // what this means, search the Web.
36 void printSet(const set<int>& a);
37
38 int main()
39 {
40
41     /
42     *****
43     **
44     VECTORS
45     *****
46     /
47     cout << "  ***  STL VECTOR CLASS  ***  \n\n";
48
49     // Use the default constructor to declare an integer vector v1.
```

```
47     vector<int> v1;
48
49     // void push_back (const value_type& val);
50     // Use function push_back to insert the following values in v1: 12, 73, 41,
51     // 38, 25, 56, an 63 in this order.
52     v1.push_back(12);
53     v1.push_back(73);
54     v1.push_back(41);
55     v1.push_back(38);
56     v1.push_back(25);
57     v1.push_back(56);
58     v1.push_back(63);
59
60     // size_type size() const noexcept;
61     // Create a variable of type int named sizeV1 and store the size of the vector.
62     // Use function size to retrieve the size of the vector.
63     // Make sure you cast the return value of the function size to the appropriate type.
64     int sizeV1 = (int)v1.size();
65
66     // Use a FOR loop to print out the vector.
67     // Do NOT use an iterator.
68     for (int i = 0; i < sizeV1; i++) {
69         cout << v1[i] << " ";
70     }
71     cout << endl;
72
73     //void clear() noexcept;
74     // Call the function clear on vector v1.
75     v1.clear();
76
77     // size_type size() const noexcept;
78     // Call function size to print the size of v1.
79     cout << v1.size() << endl;
80
81     // size_type capacity() const noexcept;
82     // Call function capacity to output the capacity of v1.
83     cout << v1.capacity() << endl;
84
85     // Create an array of integers containing: 10,11,12,13,14,15,16,17,18,19
86     int a[] = {10,11,12,13,14,15,16,17,18,19};
87
88     // Use the default constructor to declare an integer vector v2.
89     vector<int> v2;
90
91     // void assign (InputIterator first, InputIterator last);
92     // Use function assign to copy elements 12, 13, 14, 15, and 16 in v2.
93     // One statement only.
```

```
94     v2.assign(a + 2, a + 7);
95
96     // Call the function printVector to print v2.
97     printVector(v2);
98
99     // const_reference back() const;
100    // Use the function back to output the last element in the vector
101    // (Notice that the back function returns a reference.)
102    cout << v2.back() << endl;
103
104    // Use the default constructor to declare an integer vector v3.
105    vector<int> v3;
106
107    // void assign (size_type n, const value_type& val);
108    // Use function assign to insert the values 7, 7, 7, 7, and 7.
109    // One statement only.
110    v3.assign(5, 7);
111
112    // Call the function printVector to print v3.
113    printVector(v3);
114
115    // const_reference at(size_type n) const;
116    // Use function at to replace the middle element with 100.
117    // (Notice that the at function returns a reference.)
118    v3.at(2) = 100;
119
120    // Call the function printVector to print v3.
121    printVector(v3);
122
123    // vector (const vector& x);
124    // Use the copy constructor to create a new vector v4 with the
125    // same elements of v3.
126    vector<int> v4 = vector<int>(v3);
127
128    // Call the function printVector to print v4.
129    printVector(v4);
130
131    // Create an iterator iterVector4 to point to the first element of v4.
132    vector<int>::iterator iterVector4 = v4.begin();
133
134    // Create an iterator iterVector2 to point to the second element of v2.
135    vector<int>::iterator iterVector2 = v2.begin() + 1;
136
137    // iterator insert (const_iterator position, InputIterator first,
138    //                  InputIterator last);
139    // Use function insert to insert the second, third, and fourth element
140    // of v2 as the first, second, and third element of v4.
141    // (Notice that the insert function returns an iterator,
142    //  but if we do not intend to use it, we can ignore it.)
```

```
142     v4.insert(iterVector4, iterVector2, v2.begin() + 4);
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.begin() + v4.size()), 3, 0);
153
154     // Call the function printVector to print v4.
155     printVector(v4);
156
157     // bool empty() const noexcept;
158     // const_reference back() const;
159     // void pop_back();
160     // Use a WHILE loop to remove and output each element of v2 backwards.
161     // Use function empty for the loop condition, function back to output
162     // the last element, and function pop_back to remove elements.
163     // (Notice that the insert function returns an iterator,
164     // but if we do not intend to use it, we can ignore it.)
165     while (!v2.empty()) {
166         cout << v2.back() << " ";
167         v2.pop_back();
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;
```

```
190     vector<int> v5;
191
192     // Use the overloaded assignment operator to copy all the elements of v2
193     // into v5.
194     v5 = v2;
195
196     // void resize (size_type n);
197     // size_type size() const noexcept;
198     // Delete the last element of v5 by using the functions resize and size
199     v5.resize((int)v5.size() - 1);
200
201     // Call the function printVector to print v5.
202     printVector(v5);
203
204     // Create an iterator iterVector5 to point to the first element of v5.
205     vector<int>::iterator iterVector5 = v5.begin();
206
207     // iterator erase (const_iterator first, const_iterator last);
208     // size_type size() const noexcept;
209     // Call the function erase to delete the second half of v5.
210     // Use function size to get the range.
211     // (Notice that the insert function returns an iterator,
212     //  but if we do not intend to use it, we can ignore it.)
213     v5.erase(v5.begin() + (v5.size() / 2), v5.begin() + v5.size());
214
215     // Call the function printVector to print v5 again.
216     printVector(v5);
217
218     // iterator erase (const_iterator position);
219     // Call the function erase to delete the first element of the vector.
220     // (Notice that the insert function returns an iterator,
221     //  but if we do not intend to use it, we can ignore it.)
222     v5.erase(v5.begin());
223
224     // Call the function printVector to print v5 again.
225     printVector(v5);
226
227     // Create a vector of integers named v6 containing numbers from 100 to 105.
228     // Using the copy constructor, create a vector named v7, a copy of v6.
229     vector<int> v6 = { 100, 101, 102, 103, 104, 105 };
230     vector<int> v7 = vector<int>(v6);
231
232     // iterator erase (const_iterator position);
233     // iterator insert (const_iterator position, const value_type& val);
234     // Erase element 103 from v7 and insert element 333 in its place,
235     // by using an iterator.
236     // Note that the function erase returns an iterator that can be used
237     // to insert 333 in the right position.
238     v7.insert(v7.erase(v7.begin() + 3), 333);
```

```
239
240     // Using a range-based FOR loop, print v7.
241     for (int i : v7) {
242         cout << i << " ";
243     }
244     cout << endl;
245
246     /
247     *****
248     **
249
250     LISTS
251     *****
252     /
253
254     cout << "\n-----";
255     cout << "\n *** STL LIST CLASS *** \n\n";
256
257     // Use the default constructor to create three lists of integers, intList1,
258     // intList2, and intList3.
259     list<int> intList1;
260     list<int> intList2;
261     list<int> intList3;
262
263     // void push_back (const value_type& val);
264     // Use the function push_back to insert the following values in the first
265     // list:
266     // 23 58 58 58 36 15 15 93 98 58
267     int b[] = { 23, 58, 58, 58, 36, 15, 15, 93, 98, 58 };
268     for (int i = 0; i < 10; i++) {
269         intList1.push_back(b[i]);
270     }
271
272     // Call function printList to print intList1.
273     printList(intList1);
274
275     // Using the overloaded assignment operator, copy elements of intList1 into
276     // intList2.
277     intList2 = intList1;
278
279     // Call function printList to print intList2.
280     printList(intList2);
281
282     // void sort();
283     // Using function sort, sort all elements in the second list.
284     intList2.sort();
285
286     // Call function printList to print intList2.
287     printList(intList2);
288
```

```

283     // void unique();
284     // Using function unique, removes all consecutive duplicates in the list.
285     intList2.unique();
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     int c[] = { 13, 23, 25, 136, 198 };
294     for (int i = 0; i < 5; i++) {
295         intList3.push_back(c[i]);
296     }
297
298     // Call function printList to print intList3.
299     printList(intList3);
300
301     // void merge (list& x);
302     // Add to the second list all elements of the third list(browse the
303     // list of functions in cplusplus.com to figure out which function
304     // you need to use).
305     // --> This is ONE statement only.
306     intList3.merge(intList2);
307
308     // Call function printList to print intList2.
309     printList(intList3);
310
311     /
312     *****
313     **
314
315     SETS
316     *****
317
318     /
319     cout << "\n\n-----";
320     cout << "\n *** STL SET CLASS *** \n\n";
321
322     // Create a set of ints named set1 using the initializer list to insert the
323
324     // following integers in this order: 2, 7, 5, 6, 9, 1 and 3.
325     set<int> set1{ 2, 7, 5, 6, 9, 1, 3 };
326
327     // Print the set using the printSet function you implemented.
328     printSet(set1);
329
330     // What do you notice in the printout?
331     //The items were arranged in ascending order
332     // size_type erase (const value_type& val);
333     // Use the function erase integer 9 from set1.

```

```
328     // Print out set1.
329     set1.erase(9);
330     printSet(set1);
331
332     // size_type erase (const value_type& val);
333     // Use the function erase integer 2 from set1, but
334     // this time use cout to print the return value.
335     // What is the return value?
336     cout << set1.erase(2); //it returns a reference to the value before it, in
        this case it was 1
337
338     // If you do not know what the return value is, then
339     // check set::erase in cplusplus.com
340     cout << endl;
341
342     // Print set1.
343     printSet(set1);
344
345     // iterator erase (const_iterator position);
346     // This function is different from the previous one,
347     // because instead of passing a value, it passes a
348     // position indicated by an iterator.
349     // Delete the second element in the set without creating
350     // an iterator variable and using the prefix increment
351     // operator.
352     set1.erase(++set1.begin());
353
354     // Print set1.
355     printSet(set1);
356
357     // pair<iterator,bool> insert (const value_type& val);
358     // Use the function insert to insert 4 and 8 in set1.
359     set1.insert(4);
360     set1.insert(8);
361
362     // Print set1.
363     printSet(set1);
364
365     set<int>::iterator first = set1.begin();
366     set<int>::iterator second = ++set1.begin();
367     // iterator erase(const_iterator first, const_iterator last);
368     // Use function erase to delete the first and second element
369     // in set1. Use the given iterators created above.
370     // Note that you should write one statement only.
371     set1.erase(first, ++second);
372
373     // Print set1.
374     printSet(set1);
375
```



```

376 // Your output should be: 5 6 7 8
377 // If it is not, you need to carefully view the function erase
378 // to understand how it works.
379
380 cout << "\n\n-----";
381 cout << "\n\nThe output for the next sections depends on your      ↗
    implementation.";
382
383 /                                                                 ↗
    *****
    **
384 MAPS
385 *****
    /
386 cout << "\n\n-----";
387 cout << "\n *** STL MAP CLASS *** \n\n";
388
389 // Create a few maps using the different constructors shown in the slides.
390 // Use the following functions to manipulate the maps:
391 // pair<iterator,bool> insert (const value_type& val);
392 // void insert (InputIterator first, InputIterator last);
393 // iterator erase(const_iterator position);
394 // size_type erase(const key_type& k);
395 // iterator erase(const_iterator first, const_iterator last);
396 // Print each map without creating a print function, but by using
397 // a loop.
398
399 /*Map 1*/
400 map<int, string> map1;
401 for (int i = 0; i < 10; i++) {
402     string d[] = { "null", "first", "second", "third", "fourth", "fifth",
403                  "sixth", "seventh", "eight", "ninth"};
404
405     map1.insert(make_pair(i, d[i]));
406 }
407
408 map<int, string>::const_iterator mIter1 = map1.cbegin();
409 for (mIter1; mIter1 != map1.cend(); mIter1++) {
410     cout << "(" << mIter1->first << ", " << mIter1->second << ")" << endl;
411 }
412
413 cout << "\n";
414
415 /*Map 2*/
416 map<int, string> map2;
417 for (int i = 1; i < 10; i++) {
418     string d[] = { "A", "B", "C", "D", "E", "F",
419                  "G", "H", "I", "J" };
420

```

```

421     map2.insert(make_pair(i, d[i - 1]));
422 }
423
424 map<int, string>::const_iterator mIter2 = map2.cbegin();
425 for (mIter2; mIter2 != map2.cend(); mIter2++) {
426     cout << "(" << mIter2->first << ", " << mIter2->second << ")" << endl;
427 }
428
429 /*****
430 *       Create statements using the functions below.
431 *       You might need to create new containers.
432 *       Is the output what you expected?
433 *****/
434
435
436 cout << "\n\n-----";
437 cout << "\n *** OTHER FUNCTIONS *** \n\n";
438
439 // list: void assign (size_type n, const value_type& val);
440
441 // vector: void assign (InputIterator first, InputIterator last);
442
443 // list: const_reference back() const;
444 // (Notice that this back function returns a reference.)
445
446 // list: void clear() noexcept;
447
448 // list: bool empty() const noexcept;
449
450 // vector: const_reference front() const;
451
452 // list: iterator insert (const_iterator position, const value_type& val);
453 // (Notice that the insert function returns an iterator.)
454
455 // list: void pop_back();
456
457 // list: void pop_front();
458
459 // list: void push_front (const value_type& val);
460
461 // list: void remove (const value_type& val);
462
463 // list: void reverse() noexcept;
464
465 // list: void splice (const_iterator position, list& x);
466
467 // list: void splice (const_iterator position, list& x, const_iterator i);
468
469 // list: void splice (const_iterator position, list& x, const_iterator

```

```
    first, const_iterator last);
470
471 // set: void swap (set& x);
472
473 // set: const_iterator find (const value_type& val) const;
474
475 cout << "\n\n-----";
476
477 cout << endl;
478 system("Pause");
479 return 0;
480 }
481
482 // Definition function printVector
483 void printVector(const vector<int>& a) {
484     vector<int>::const_iterator iter = a.cbegin();
485
486     for (iter; iter != a.cend(); iter++) {
487         cout << *iter << " ";
488     }
489     cout << endl;
490 }
491
492 // Definition function printList
493 void printList(const list<int>& a) {
494     list<int>::const_iterator iter = a.cbegin();
495
496     while (iter != a.cend()) {
497         cout << *iter << " ";
498         iter++;
499     }
500     cout << endl;
501 }
502
503 // Definition function printSet
504 void printSet(const set<int>& a) {
505     set<int>::const_iterator iter = a.cbegin();
506
507     for (iter; iter != a.cend(); iter++) {
508         cout << *iter << " ";
509     }
510     cout << endl;
511 }
```