Container Syntax Reference

Here’s a quick reference sheet for the syntax of common operations on the different container types. This list isn’t comprehensive – for that, visit the Stanford C++ Library Reference website.

This version, which will be available at the midterm, was updated on February 3, 2020.

<table>
<thead>
<tr>
<th>Lexicon</th>
<th>HashMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexicon lex; Lexicon english(filename);</td>
<td>HashMap&lt;K, V&gt; map = {{k₁, v₁}, ... {kₙ, vₙ}};</td>
</tr>
<tr>
<td></td>
<td>cout &lt;&lt; map[key] &lt;&lt; endl; // Autoinserts</td>
</tr>
<tr>
<td>addWord(word);</td>
<td>map[key] = value; // Autoinserts</td>
</tr>
<tr>
<td></td>
<td>bool present = map.containsKey(key);</td>
</tr>
<tr>
<td>contains(word);</td>
<td>int numKeys = map.size();</td>
</tr>
<tr>
<td>containsPrefix(prefix);</td>
<td>bool empty = map.isEmpty();</td>
</tr>
<tr>
<td>isEmpty();</td>
<td>map.clear();</td>
</tr>
<tr>
<td>isNotEmpty();</td>
<td>Vector&lt;K&gt; keys = map.keys();</td>
</tr>
<tr>
<td></td>
<td>/* Elements visited in sorted order. */</td>
</tr>
<tr>
<td>lex.clear();</td>
<td>for (K key: map) { ... }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stack</th>
<th>Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack.push(elem);</td>
<td>queue.enqueue(elem);</td>
</tr>
<tr>
<td>T val = stack.pop(); // Removes top</td>
<td>T val = queue.dequeue(); // Removes front</td>
</tr>
<tr>
<td>T val = stack.peek(); // Looks at top</td>
<td>T val = queue.peek(); // Looks at front</td>
</tr>
<tr>
<td>int numElems = stack.size();</td>
<td>int numElems = queue.size();</td>
</tr>
<tr>
<td>bool empty = stack.isEmpty();</td>
<td>bool empty = queue.isEmpty();</td>
</tr>
<tr>
<td>stack.clear();</td>
<td>queue.clear();</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HashSet</th>
<th>Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>HashSet&lt;T&gt; set = {v₁, v₂, ..., vₙ};</td>
<td>Vector&lt;T&gt; vec = {v₁, v₂, ..., vₙ};</td>
</tr>
<tr>
<td>set.add(elem);</td>
<td>vec[0] = {v₁, v₂, ..., vₙ};</td>
</tr>
<tr>
<td>set += elem; set -= elem;</td>
<td>vec.add(elem);</td>
</tr>
<tr>
<td>HashSet&lt;T&gt; s = set - elem; // or + elem</td>
<td>vec += elem;</td>
</tr>
<tr>
<td>bool present = set.containsKey(elem);</td>
<td>vec.insert(index, elem);</td>
</tr>
<tr>
<td>set.remove(x); set -= x; set -= set2;</td>
<td>vec.erase(index);</td>
</tr>
<tr>
<td>HashSet&lt;T&gt; unionSet = s1 + s2;</td>
<td>int numElems = vec.size();</td>
</tr>
<tr>
<td>HashSet&lt;T&gt; intersectSet = s1 * s2;</td>
<td>bool empty = vec.isEmpty();</td>
</tr>
<tr>
<td>HashSet&lt;T&gt; difference = s1 - s2;</td>
<td>vec.subList(start, numElems);</td>
</tr>
<tr>
<td>T elem = set.first();</td>
<td>/* Visited in order. */</td>
</tr>
<tr>
<td>int numElems = set.size();</td>
<td>for (T elem: vec) { ... }</td>
</tr>
<tr>
<td>bool empty = set.isEmpty();</td>
<td></td>
</tr>
<tr>
<td>set.clear();</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>string</th>
<th>Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>str[index]; // Read/write</td>
<td>Grid&lt;T&gt; grid(nRows, nCols);</td>
</tr>
<tr>
<td>str.substr(start);</td>
<td>Grid&lt;T&gt; grid(nRows, nCols, fillValue);</td>
</tr>
<tr>
<td>str.substr(start, numChars);</td>
<td></td>
</tr>
<tr>
<td>str.find(c); // index or string::npos</td>
<td>int nRows = grid.numRows();</td>
</tr>
<tr>
<td>str.find(c, startIndex);</td>
<td>int nCols = grid.numCols();</td>
</tr>
<tr>
<td>str += ch;</td>
<td>if (grid.inBounds(row, col)) { ... }</td>
</tr>
<tr>
<td>str += otherStr;</td>
<td>grid[row][col] = value;</td>
</tr>
<tr>
<td>str.erase(index, length);</td>
<td>cout &lt;&lt; grid[row][col] &lt;&lt; endl;</td>
</tr>
</tbody>
</table>

/* Visited in no particular order. */
for (T elem: vec) { ... }

/* Visited in no particular order. */
for (K key: map) { ... }

/* Visited left-to-right, top-to-bottom */
for (T elem: grid) { ... }

/* Visited in order. */
for (T elem: set) { ... }

/* Visited in order. */
for (char ch: str) { ... }