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.

Lexicon

```
Lexicon lex; Lexicon english(filename);
lex.addWord(word);
bool present = lex.contains(word);
bool pref = lex.containsPrefix(prefix);
int numElems = lex.size();
bool empty = lex.isEmpty();
lex.clear();

/* Elements visited in sorted order. */
for (string word: lex) { … }
```

HashMap

```
HashMap<K, V> map = {{k₁, v₁}, … {kₙ, vₙ}};
cout << map[key] << endl; // Autoinserts
map[key] = value; // Autoinserts
bool present = map.containsKey(key);
int numKeys = map.size();
bool empty = map.isEmpty();
map.remove(key);
map.clear();
Vector<K> keys = map.keys();

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

Stack

```
stack.push(elem);
T val = stack.pop();   // Removes top
T val = stack.peek();  // Looks at top
int numElems = stack.size();
bool empty = stack.isEmpty();
stack.clear();
```

Queue

```
queue.enqueue(elem);
T val = queue.dequeue(); // Removes front
T val = queue.peek();   // Looks at front
int numElems = queue.size();
bool empty = queue.isEmpty();
queue.clear();
```

HashSet

```
HashSet<T> set = {v₁, v₂, …, vₙ};
set.add(elem);
set += elem; set -= elem;
HashSet<T> s = set - elem; // or + elem
bool present = set.contains(elem);
set.remove(x); set -= x; set -= set2;
HashSet<T> unionSet = s1 + s2;
HashSet<T> intersectSet = s1 * s2;
HashSet<T> difference = s1 - s2;
T elem = set.first();
int numElems = set.size();
bool empty = set.isEmpty();
set.clear();

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

Vector

```
Vector<T> vec = {v₁, v₂, …, vₙ};
vec[index]; // Read/write
vec.add(elem);
vec += elem;
vec.insert(index, elem);
vec.indexOf(elem); // index or -1
vec.remove(index);
vec.clear();
int numElems = vec.size();
bool empty = vec.isEmpty();
vec.subList(start, numElems);

/* Visited in order. */
for (T elem: vec) { … }
```

string

```
str[index]; // Read/write
str.substr(start);
str.substr(start, numChars);
str.find(c); // index or string::npos
str.find(c, startIndex);
str += ch;
str += otherStr;
str.erase(index, length);

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

Grid

```
Grid<T> grid(nRows, nCols);
Grid<T> grid(nRows, nCols, fillValue);

int nRows = grid.numRows();
int nCols = grid.numCols();

if (grid.inBounds(row, col)) { … }

grid[row][col] = value;
cout << grid[row][col] << endl;

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