CS106B FINAL REFERENCE SHEET

For-each loop iteration over collection (not `Stack`,`Queue`,`PriorityQueue`) `for (type name : collection) { ... }`

** All Big-Oh runtimes listed are average-case; some methods perform differently under various cases.*

Vector<T>

`v.add(val)` or `v += val`	appends value to end of vector	O(1) *
`v.clear()`	removes all elements	O(1)
`v.get(i)` or `v[i]`	returns value at given index	O(1)
`v.insert(i, val)`	inserts at given index, shifting subsequent values right	O(N)
`v.isEmpty()`	returns `true` if there are no elements	O(1)
`v.remove(i)`	removes value at given index, shifting subsequent values left	O(N)
`v.set(i, val)` or `v[i] = val`	replaces value at given index	O(1)
`v.size()`	returns number of elements	O(1)
`v.subList(start, length)`	returns new vector containing subrange of elements	O(N)

Grid<T>

`g.get(row, col)` or `g[row][col]` or `g[location]`	returns value stored at given row/column location	O(1)
`g.inBounds(row, col)` or `g.inBounds(location)`	returns `true` if given row/column index is within (0, 0) ... (*R*, *C*)	O(1)
`g.numCols()`	returns number of columns *C*	O(1)
`g.numRows()`	returns number of rows *R*	O(1)
`g.set(row, col, val)` or `g[row][col] = val` or `g[location] = val`	changes value stored at given row/column location	O(1)

GridLocation

`GridLocation(row,col)`	constructor
`loc.row`	access row field
`loc.col`	access col field

GridLocationRange

`GridLocationRange(`
`  startRow, startCol,`
`  endRow, endCol)`
	constructor, start/end locations are inclusive

`r.contains(loc)`	returns `true` if loc contained in range
`r.isEmpty()`	returns `true` if range is empty
`r.startLocation()` `r.endLocation()`	returns start/end as GridLocation
`for (GridLocation loc: r)`	iterate over locations in range

Stack<T>

`s.clear()`	removes all elements	O(1)
`s.push(val)`	adds value to top of stack	O(1)
`s.pop()`	removes/returns top value `pop`/`peek` error if empty	O(1)
`s.peek()`	returns top value without removing	O(1)
`s.isEmpty()`	returns `true` if no elements	O(1)
`s.size()`	returns number of elements	O(1)

Queue<T>

`q.clear()`	removes all elements	O(N)
`q.enqueue(val)`	adds value to back of queue	O(1)
`q.dequeue()`	removes/returns front value `dequeue`/`peek` error if empty	O(1)
`q.peek()`	returns front value without removing	O(1)
`q.isEmpty()`	returns `true` if no elements	O(1)
`q.size()`	returns number of elements	O(1)

Set<T>, HashSet<T>

`s.add(val)` or `s += val`	adds to set; if a duplicate, no effect	O(log N), O(1)
`s.clear()`	removes all elements	O(N)
`s.contains(val)`	returns `true` if value is found in the set	O(log N), O(1)
`s.first()`	returns first element from set (does not remove it)	O(log N), O(1)
`s.isEmpty()`	returns `true` if there are no elements	O(1)
`s.isSubsetOf(s2)`	returns `true` if `s2` contains all elements of `s`	O(N)
`s.remove(val)` or `s -= val`	removes value from set, if present	O(log N), O(1)
`s.size()`	returns number of elements	O(1)
`s1 == s2, s1 != s2`	operators for set equality testing	O(N)
`s1.unionWith(s2)`	returns set of all elements of `s2` and `s1`	O(N)
`s1.intersect(s2)`	returns set of all elements in both `s1` and `s2`	O(N)
`s1.difference(s2)`	returns set of all elements `s1` not in `s2`	O(N)

CS106B FINAL REFERENCE SHEET

Map<K, V>, HashMap<K, V>

m.clear()	removes all key/value pairs	O(N)
m.containsKey(*key*)	returns true if map contains a pair for the given key	O(log N), O(1)
m.get(*key*) or *m*[*key*]	returns value paired with the given key (or a default value such as 0, false, "" if key is not present)	O(log N), O(1)
m.isEmpty()	returns true if there are no key/value pairs	O(1)
m.keys()	returns a Vector copy of all keys in the map	O(N)
m.put(*key*, *val*) or *m*[*key*] = *val*	adds a pairing of the given key to the given value	O(log N), O(1)
m.remove(*key*)	removes any existing pairing for the given key	O(log N), O(1)
m.size()	returns number of key/value pairs	O(1)
m.values()	returns a Vector copy of all values in the map	O(N)

A for-each loop on a map iterates over the *keys*, not the *values*.

PriorityQueue<V>

pq.clear()	removes all entries	O(1)
pq.dequeue()	removes/returns value of frontmost entry, frontmost = most urgent priority, dequeue/peek error if empty	O(log N)
pq.enqueue(*val*, *priority*)	adds entry for value with given priority	O(log N)
pq.isEmpty()	returns true if no entries	O(1)
pq.peek()	returns value of frontmost entry	O(1)
pq.peekPriority()	returns priority of frontmost entry	O(1)
pq.size()	returns number of entries	O(1)

Lexicon

l.contains(*word*)	returns true if the word is found in the lexicon	O(log N)
l.containsPrefix(*text*)	returns true if any word starts with this prefix text	O(log N)

Strings

str.at(*i*) or *s*[*i*]	character at a given 0-based index in the string
str.append(*str*)	add text to the end of a string *(in-place)*
str.compare(*str*)	return -1, 0, or 1 depending on relative ordering
str.erase(*i*, *length*)	delete text from a string starting at given index *(in-place)*
str.find(*str*) *str*.rfind(*str*)	returns the first or last index where the start of the given string or character appears in this string (or string::npos if not found)
str.insert(*i*, *str*)	add text into a string at a given index *(in-place)*
str.length() or *str*.size()	number of characters in this string
str.replace(*i*, *len*, *str*)	replaces *len* chars at given index with new text *(in-place)*
str.substr(*start*, *length*) or *str*.substr(*start*)	returns the next *length* characters beginning at index *start* (inclusive); if *length* is omitted, grabs from *start* to the end of the string
endsWith(*str*, *suffix*), startsWith(*str*, *prefix*)	returns true if the string begins or ends with the given prefix/suffix
integerToString(*int*), stringToInteger(*str*)	returns a conversion between numbers and strings
stringContains(*str*, *substr*)	true if *substr* contained in *str*
stringSplit(*str*, *separator*)	breaks apart a string into a Vector of substrings divided by separator
toLowerCase(*str*), toUpperCase(*str*)	returns an upper/lowercase version of a string
trim(*str*)	returns string with any surrounding whitespace removed

SimpleTest

```
STUDENT_TEST("Example test cases") {
     Vector<int> v;
     EXPECT(v.isEmpty());
     EXPECT_EQUAL(1 + 2, 3);
     EXPECT_ERROR(empty[0]);
}
```