

CS106B MIDTERM REFERENCE SHEET

You can perform a for-each loop over any collection other than Stack and Queue: **for (type name : collection) { ... }**
** All Big-Oh runtimes listed are average-case; some methods perform differently under various cases.*

Vector<T>

<code>v.add(val)</code> or <code>v += val</code>	append value to end of vector	$O(1)$ *
<code>v.clear()</code>	remove all elements	$O(1)$
<code>v.get(i)</code> or <code>v[i]</code>	return value at given index	$O(1)$
<code>v.insert(i, val)</code>	insert value at given index, shifting subsequent values right	$O(N)$
<code>v.isEmpty()</code>	return true if no elements	$O(1)$
<code>v.remove(i)</code>	remove value at given index, shifting subsequent values left	$O(N)$
<code>v.set(i, val)</code> or <code>v[i] = val</code>	replace value at given index	$O(1)$
<code>v.size()</code>	return count of elements	$O(1)$
<code>v.subList(start, length)</code>	create vector copy containing subrange of elements	$O(N)$

Grid<T>

<code>g.get(row, col)</code> or <code>g[row][col]</code> or <code>g[location]</code>	return value at given row/column location	$O(1)$
<code>g.inBounds(row, col)</code> or <code>g.inBounds(location)</code>	return true if given row/column location is within grid bounds	$O(1)$
<code>g.locations()</code>	return GridLocationRange for entire grid	$O(1)$
<code>g.numCols()</code>	return count of columns	$O(1)$
<code>g.numRows()</code>	return count of rows	$O(1)$
<code>g.set(row, col, val)</code> or <code>g[row][col] = val</code> or <code>g[location] = val</code>	replace value at given row/column location	$O(1)$

GridLocation

<code>GridLocation(row, col)</code>	constructor
<code>loc.row</code>	access row field
<code>loc.col</code>	access col field

GridLocationRange

<code>GridLocationRange(startRow, startCol, endRow, endCol)</code>	constructor, start/end locations are inclusive
<code>r.contains(loc)</code>	return true if location contained in range
<code>r.isEmpty()</code>	return true if range is empty
<code>r.startLocation()</code>	return start/end as GridLocation
<code>r.endLocation()</code>	
<code>for (GridLocation loc: r)</code>	iterate over locations in range

Stack<T>

<code>s.clear()</code>	remove all elements	$O(1)$
<code>s.push(val)</code>	add value to top of stack	$O(1)$
<code>s.pop()</code>	remove/return top value pop/peek error if empty	$O(1)$
<code>s.peek()</code>	return top value without removing	$O(1)$
<code>s.isEmpty()</code>	return true if no elements	$O(1)$
<code>s.size()</code>	return count of elements	$O(1)$

Queue<T>

<code>q.clear()</code>	remove all elements	$O(N)$
<code>q.enqueue(val)</code>	add value to back of queue	$O(1)$
<code>q.dequeue()</code>	remove/return front value dequeue/peek error if empty	$O(1)$
<code>q.peek()</code>	return front value without removing	$O(1)$
<code>q.isEmpty()</code>	return true if no elements	$O(1)$
<code>q.size()</code>	return count of elements	$O(1)$

Set<T>

<code>s.add(val)</code> or <code>s += val</code>	add value to set; if a duplicate, no effect	$O(\log N)$
<code>s.clear()</code>	remove all elements	$O(N)$
<code>s.contains(val)</code>	return true if value contained in set	$O(\log N)$
<code>s.first()</code>	return first element from set (does not remove it)	$O(\log N)$
<code>s.isEmpty()</code>	return true if no elements	$O(1)$
<code>s1.isSubsetOf(s2)</code>	return true if s2 contains all elements of s1	$O(N)$
<code>s.remove(val)</code> or <code>s -= val</code>	remove value from set if contained	$O(\log N)$
<code>s.size()</code>	return count of elements	$O(1)$
<code>s1 == s2, s1 != s2</code>	operators for set equality testing	$O(N)$
<code>s1.unionWith(s2)</code>	change s1 to add all elements of s2	$O(N \log N)$
<code>s1.intersect(s2)</code>	change s1 to remove all elements not in s2	$O(N \log N)$
<code>s1.difference(s2)</code>	change s1 to remove all elements of s2	$O(N \log N)$

CS106B MIDTERM REFERENCE SHEET

Map<K, V>

<code>m.clear()</code>	remove all key/value pairs	$O(N)$
<code>m.containsKey(key)</code>	return true if map contains a pair for given key	$O(\log N)$
<code>m.get(key)</code> or <code>m[key]</code>	return value paired with given key (or a default value such as 0 , false , "" if key is not present)	$O(\log N)$
<code>m.isEmpty()</code>	return true if no key/value pairs	$O(1)$
<code>m.keys()</code>	create Vector copy of all keys	$O(N)$
<code>m.put(key, val)</code> or <code>m[key] = val</code>	add a pairing of given key to given value	$O(\log N)$
<code>m.remove(key)</code>	remove any existing pairing for given key	$O(\log N)$
<code>m.size()</code>	return count of key/value pairs	$O(1)$
<code>m.values()</code>	create Vector copy of all values	$O(N)$

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

Lexicon

<code>lex.contains(word)</code>	return true if given word contained in lexicon	$O(1)$
<code>lex.containsPrefix(prefix)</code>	return true if any word in lexicon starts with given prefix	$O(1)$

string, strlib.h

<code>str.at(i)</code> or <code>s[i]</code>	return character at given 0-based index
<code>str.append(text)</code>	add text to end of string (<i>in-place</i>)
<code>str.compare(str2)</code>	return -1, 0, or 1 depending on relative ordering
<code>str.erase(i, length)</code>	delete text of given length starting at given index (<i>in-place</i>)
<code>str.find(text)</code>	return first index of matching text (or string::npos if not found)
<code>str.insert(i, text)</code>	add text at a given index (<i>in-place</i>)
<code>str.length()</code> or <code>str.size()</code>	return count of characters
<code>str.replace(i, length, text)</code>	replace given length chars at given index with text (<i>in-place</i>)
<code>str.substr(start, length)</code> or <code>str.substr(start)</code>	return new string consisting of length characters from given start index if length argument omitted, grabs from start index to end of string
<code>endsWith(str, suffix)</code> , <code>startsWith(str, prefix)</code>	return true if string begins or ends with the given prefix/suffix
<code>integerToString(i)</code> , <code>stringToInteger(str)</code>	conversion between number and string
<code>stringContains(str, text)</code>	return true if text contained in string
<code>stringSplit(str, separator)</code>	divide a string into Vector of substrings divided by separator
<code>toLowerCase(str)</code> , <code>toUpperCase(str)</code>	return new upper/lowercase string

char

<code>isalpha(c)</code> , <code>isdigit(c)</code> , <code>isspace(c)</code> , <code>ispunct(c)</code> , <code>islower(c)</code> , <code>isupper(c)</code> ,	return true if character is alphabetic character from a-z or A-Z, digit 0-9, whitespace character (space, \t, \n, etc.), punctuation mark (#, \$, !, etc.) respectively
<code>tolower(c)</code> , <code>toupper(c)</code>	return lower/uppercase equivalent (unchanged if not alpha)

random.h

<code>randomChance(probability)</code>	return random bool of true/false with the given probability of true from 0..1
<code>randomInteger(min, max)</code>	return random integer in range $[min-max]$, inclusive
<code>randomReal(low, high)</code>	return random real number in range $[low-high)$, up to but not including <i>high</i>

SimpleTest

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