*** CSC 016 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> Members ("vector.h") (5.1)

`v.add(val);` or `v += val;`	appends 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.toString()`	returns string representation of elements such as `"{1, 2, 3}"`	O(N)

Grid<T> and SparseGrid<T> Members ("grid.h", "sparsegrid.h") (5.1)

`g.fill(val);`	set every cell to store a given value	O(R*C)
`g.get(row, col)` or `g[row, col]`	returns value stored at given row/column	Grid O(1), sparse O(logN)
`g.inBounds(row, col)`	returns `true` if given row/column index is within (0, 0) ... (*R*, *C*)	O(1)
`g.numCols()   // or g.width()`	returns number of columns *C*	O(1)
`g.numRows()   // or g.height()`	returns number of rows *R*	O(1)
`g.resize(nCols, nRows);`	changes to have the given number of rows/cols; wipes all data	O(R*C)
`g.set(row, col, val);` or `g[row][col] = val;`	changes value stored at given row/column	Grid O(1), sparse O(logN)

Stack<T> Members ("stack.h") (5.2)

`s.clear();`	removes all elements	O(N)
`s.push(val);`	adds given value on top of the stack	O(1)
`s.pop()`	remove/return top value from stack; `pop`/`peek` throw exception if empty	O(1)
`s.peek()`	return top value without removing	O(1)
`s.isEmpty()`	returns `true` if there are no elements	O(1)
`s.size()`	returns number of elements	O(1)
`s.toString()`	string (right=top) such as `"{1, 2, 3}"`	O(N)

Queue<T> Members ("queue.h") (5.3)

`q.clear();`	removes all elements	O(N)
`q.enqueue(val);`	adds value to back of queue	O(1)
`q.dequeue()`	remove/return value from front; `dequeue`/`peek` throw if empty	O(1)
`q.peek()`	return front without removing	O(1)
`q.isEmpty()`	returns `true` if no elements	O(1)
`q.size()`	returns number of elements	O(1)
`q.toString()`	(left=front) e.g. `"{1, 2, 3}"`	O(N)

Set<T> and HashSet<T> Members ("set.h", "hashset.h") (5.5)

`s.add(val);` or `s += val;`	adds to set; if a duplicate, no effect	set O(log N), hash O(1)
`s.clear();`	removes all elements	O(N)
`s.contains(val)`	returns `true` if value is found in the set	set O(log N), hash O(1)
`s.first()`	returns first element from set (does not remove it)	set O(log N), hash 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	set O(log N), hash O(1)
`s.size()`	returns number of elements	O(1)
`s.toString()`	returns string such as `"{1, 2, 3}"`	O(N)
`s1 == s2, s1 != s2`	operators for set equality testing	O(N)
`s1 + s2,  s1 += s2;`	operators for union; adds elements of `s2` to `s1`	O(N)
`s1 * s2,  s1 *= s2;`	intersection; removes all from `s1` not found in `s2`	O(N)
`s1 - s2,  s1 -= s2;`	difference; removes all from `s1` that are found in `s2`	O(N)

Lexicon Members ("lexicon.h") (5.5)

`l.add(word);`	adds a word; if a duplicate, no effect	O(log N)
`l.clear();`	removes all words	O(N)
`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)
`l.isEmpty()`	returns `true` if there are no words in the lexicon	O(1)
`l.remove(word);`	removes word from lexicon, if present	O(log N)
`l.size()`	returns number of words	O(1)
`s.toString()`	returns string such as `"{a, ball, cat, zebra}"`	O(N log N)

*** CSC 016 MIDTERM REFERENCE SHEET ***

`Map<K, V>` and `HashMap<K, V>` Members (`"map.h"`, `"hashmap.h"`) (5.4)

m`.clear();`	removes all key/value pairs	O(N)
m`.containsKey(`***key***`)`	returns `true` if map contains a pair for the given key	map O(log N), hash O(1)
m`.get(`***key***`)` or ***m***`[`***key***`]`	returns value paired with the given key (a default value if the key is not present)	map O(log N), hash 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	map O(log N), hash O(1)
m`.remove(`***key***`);`	removes any existing pairing for the given key	map O(log N), hash O(1)
m`.size()`	returns number of key/value pairs	O(1)
m`.toString()`	returns string representation such as `"{a:90, d:60, c:70}"`	O(N)
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*.

String Members and Utility Functions (`<string>, "strlib.h"`) (3.2)

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`.c_str()`	returns the equivalent C string
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***`)` `realToString(`***double***`)`, `stringToReal(`***str***`)`	returns a conversion between numbers and strings
`equalsIgnoreCase(`***str1, str2***`)`	`true` if ***s1*** and ***s2*** have same chars, ignoring casing
`stringSplit(`***str, separator***`)`	breaks apart a string into a vector of smaller strings based on a separator
`toLowerCase(`***str***`)`, `toUpperCase(`***str***`)`	returns an upper/lowercase version of a string
`trim(`***str***`)`	returns string with any surrounding whitespace removed

char Utility Functions (`<cctype>`) (3.3)

`isalpha(`***c***`)`, `isdigit(`***c***`)`, `isspace(`***c***`)`, `isupper(`***c***`)`, `ispunct(`***c***`)`, `islower(`***c***`)`	returns `true` if the given character is an alphabetic character from a-z or A-Z, a digit from 0-9, an alphanumeric character (a-z, A-Z, or 0-9), an uppercase letter (A-Z), a space character (space, \t, \n, etc.), respectively
`tolower(`***c***`)`, `toupper(`***c***`)`	returns lower/uppercase equivalent of a character

istream Members (`<iostream>`) (Ch. 4)

f`.fail()`	returns `true` if the last read or `open` call failed (e.g. EOF, or file-not-found)
f`.open(`***filename***`);`	opens file represented by given string
f`.close();`	stops reading file
f`.get()`	reads and returns 1 character
`getline(`***f***`&, `***str***`&)`	reads line of input into a string by reference; returns a `true`/`false` indicator of success
f `>>` ***variable***	reads a whitespace-separated token of data from input into a variable
`promptUserForFile(f&, str&)`	Prompts user with string to enter filename; reprompts until valid, then opens stream.

Random Numbers (`"random.h"`)

`randomBool()`	returns a random `bool` of `true`/`false` with 50/50% probability
`randomChance(`***probability***`)`	returns a random `bool` of `true`/`false` with the given probability of `true` from 0..1
`randomInteger(`***min, max***`)`	returns a random integer in the range [*min-max*], inclusive
`randomReal(`***low, high***`)`	returns a random real number in the range [*low-high*), up to but not including *high*