Chapter 16
Containers

FORDHAM

CISC 5004 - Programming C++
Sam Kamens
Fall, 2022

Range-Based for loop

- Iterate through each element in a vector or container
  - aka **for each loop**
- Example
- Why use this?
  - Simpler code
  - Avoids incorrect range in for loop
- To modify the container elements, use a reference
  - Example
- Can also use `auto`
  - `for (auto gradeVal : examGrades) {`
  - `for (auto& gradeVal : examGrades) {`

Standard Template Library

- C++ Template Classes providing common data structures and functions
- Components:
  - Algorithms
    - Designed to be used on a range of elements
    - Act on containers
    - e.g. sorting, searching
  - Containers
    - Store objects and data
    - Support appropriate methods for the container type
  - Functions
    - Overload the function call operator
  - Iterators
    - Work in a sequence of values

list

- Container of ordered elements
  - Sequence
  - Can not be accessed by index (i.e. not a vector)
  - e.g. `list<int> numberList;`
  - Implemented as a doubly linked list
- Supported functions
- Iterating
  - **iterator** - keep track of current list position without using an index
    - e.g. `list<int>::iterator iter;`
    - `*iter` provides the value of the current element (not a pointer)
  - Example

list

- Modifying a list using an iterator
  - Functions
- Iterating using a range-based for loop
  - No iterator needed
  - Example

pair

- Container with two data elements
  - Building block for other types of containers
- Can use any two types
- `#include <utility>`
- Example

map

- Container mapping keys to values (using Pairs)
  - aka “associative container”
  - Ordered (see also `unordered_map`)
- `emplace()` associates a key with a value (i.e. adds a Pair)
  - At most one value per key
- `at()` returns the value associated with a key
  - Update by assigning the result of `at()`
- Iterating a map gives pairs
- Example

map

- Determining if a key exists
  - If a key is not in the map, `at()` throws an `out_of_range` exception
  - Use `count()` to check if a map contains the specific key
  - Coding example
- Common map functions
- `[]` operator
  - Can add or access map entries
  - If attempting to access and the key is not found, a default entry is created
- `emplace()` vs. `insert()`
  - `emplace()` creates the pair for you
  - `insert()` requires that the entry be created in advance

set

- Collection of unique elements
  - ordered (see also `unordered_set`)
- `insert()` - Adds a new item to the set
  - Will not add a duplicate item
- `erase()` - Removes an element if it exists
- `count()` - Returns 1 if the item is in the set, 0 otherwise
- `size()` - Returns the number of items in the set
- Examples: 1, 2
- Coding Example

queue

- Ordered collection of elements, supports insertion at the tail and retrieval from the head
- push() - adds an element to the end of the queue
- front() - returns the element at the front/head of the queue
- pop() - removes the element at the head of the queue
- Common functions
- Example

deque

- Pronounced “deck” (shrug)
- aka Double Ended Queue
- Ordered container supporting element insertion and removal at both ends
- Common functions
- Can be used as a **stack**
  - Insert and remove from the front

find() function

- Algorithm
  - `#include <algorithm>`
- Find a specific value in a range of elements
- Can be used if:
  - The container has iterators
  - The elements support comparison (`operator==`)
- Example
- find_if()
  - Search for an element that satisfies a boolean condition
  - Example

sort() function

- Sorts based on iterators
- Requires that the element type supports `operator<`
- Example
- Can use a custom comparator
  - Additional argument to sort()
  - Example
- Sorting custom data types/classes
  - Overload `operator<`
  - Define custom comparators
  - Example

C++ Examples

- Shopping List
- Palindrome