(8 – 1) Container Classes & Class Templates
D & D Chapter 18

Instructor - Andrew S. O’Fallon
CptS 122 (October 8, 2018)
Washington State University
Key Concepts

- Class and block scope
- Access and utility functions
- Container classes
- Iterators
- Class templates
A class’ data members (attributes) and member functions (operations) belong to the class’ scope.

Nonmember functions do not belong to any class’ scope; they are global namespace scope.

Within a class’ scope data members are directly accessible by the member functions.
Outside of the class’ scope, public members are accessed through one of three different handles:

- An object name, a reference to an object, or a pointer to an object
- Note: the “this” pointer is considered an implicit handle available only within an object

Local variables declared inside of a member function have block scope
Access Functions

- Functions that can read or display data are considered *access functions*.
- *Predicate* functions are access functions that test a condition and return true or false; generally we append “is” to the front of the name of the function.
  - isEmpty(), isFull(), etc.
Utility Functions

- A *utility* or *helper* function is a private member function used to support other member functions’ operations.
Container Classes (I)

- Classes designed to hold and organize a collection of other classes
  - Examples of *sequence* containers include: lists, vectors, etc.
  - Example of container *adapters* include: stacks, queues, etc.
    - Container adapters are adaptations or interfaces designed to restrict functionality for an already existing container – they provide a different set of functionality
    - The Standard Template Library (STL) stack and queue adapt the double-ended queue (deque)
Container Classes (II)

- Container classes are generally separated into four categories:
  - Sequence containers – represent *linear* data structures
    - Array, deque, list (doubly-linked), vector, forward_list (C++ 11)
  - Container adapters
  - Ordered associative containers – represent *nonlinear ordered* data structures
    - Set, multiset, map, multimap (CptS 223!)
  - Unordered associative containers – represent *nonlinear unordered* data structures
Properties of STL Sequence Containers (I)

- **Array**
  - Fixed size; direct access to any element

- **Deque**
  - Rapid insertions and deletions at front or back; direct access to any element

- **List**
  - Doubly linked list; rapid insertions and deletions anywhere
Properties of STL Sequence Containers (II)

- **Vector**
  - Rapid insertions and deletions at back; direct access to any element

- **Forward_list**
  - Singly linked list, rapid insertions and deletions anywhere; C++ 11
Properties of STL Container Adapters

- **Stack**
  - Last-in, first-out (LIFO)

- **Queue**
  - First-in, first-out (FIFO)

- **Priority_queue**
  - Highest priority element is always the first one out
Functions Common to Container Classes (I)

- **Default constructor** – initializes an empty container
- **Copy constructor** – initializes the container to be a copy of an existing container of the same type
- **Move constructor** – available in C++ 11 – moves the contents of an existing container into a new container of the same type without copying each element of the argument container
Functions Common to Container Classes (II)

- *Destructor* – performs housekeeping or cleanup when the container is no longer needed.
- *Empty* – returns *true* if there are no elements in the container; *false* otherwise.
- *Insert* – inserts an item into the container.
- *Size* – returns the number of elements in the container.
Functions Common to Container Classes (III)

- **Copy operator (=)** – copies the elements of one container into another container of the same type
- **Move operator (=)** – available in C++ 11 – moves the contents of one container into another without copying each element of the argument container
- **Max_size** – returns the maximum number of elements for a container
Functions Common to Container Classes (IV)

- **Begin** – overloaded to return an *iterator* that refers to the *first* element of the container
- **End** - overloaded to return an *iterator* that refers to the *next* position after the *end* of the container
- **Erase** – *removes* one or more elements from the container
- **Clear** – *removes all* elements from the container
- **Others exist!**
Iterators

- Similar properties to a \textit{pointer}
- An \textit{iterator} is any object that points to some element in a sequence of elements, and has the ability to iterate through the elements using $++$ and indirection ($*$) operators
- \textit{Containers} support the use of iterators
Class Templates

- We have already seen function templates, we will now extend the idea to classes
- *Class templates* allow for a way to easily specify a variety of related overloaded functions (*function-template specializations*) or classes (*class-template specializations*)
- Allows for *generic* programming
- Keyword *template* denotes the start of a class template
- STL containers are “templated”
Example using Class Templates

- Developed during lecture – see code posted to schedule
Next Lecture..

- More about class templates, data structures, and containers
References

Collaborators

- Jack Hagemeister