Chapter 11
Generic Collections

3rd Edition
Computing Fundamentals with C++
Rick Mercer
Franklin, Beedle & Associates
Chapter 11
A Container with Iterators

• Build your own collection class to store any type of element
• Better understand classes with data members, constructors, and member functions
• Better understand how to develop functions that involve vector processing
Collection classes

• Programmers often use many *collections*
• Collection classes have the main purpose of storing a collection of elements
• Standard collection classes include `vector<type>`, `stack<type>`, `queue<type>`, `list<type>`
• All of these take a type argument, which is the type of elements that the collection stores
Passing Types as Arguments

- C++ has a template mechanism that allow the type of element to be set when compiled.
- It is a type enclosed two special symbols `<type>`.
- This allows us to have a vector of int, double, string, BankAccount, ...
  - The collection can only store that type of element.
  - With a type argument, we only need one collection class.
- In this presentation, a `Bag` type is implemented with templates to allow for `Bag<int> aBag;`
class Bag<Type>

- The Bag class developed here will review class definitions, vector processing, and show it is possible to pass a type like int or double as an argument.
- A bag (aka multi-set) is the most general collection.
  - Bags store a collection of elements not in any particular order and are not necessarily unique.
  - Operations include:
    - bag::add
    - bag::remove
    - bag::occurrencesOf
    - bag::size
Using a Bag object

• This code should compile, all assertions should pass

```java
Bag<int> aBag;
aBag.add(5);
aBag.add(4);
aBag.add(4);
assert(aBag.occurrencesOf(5) == 1);
assert(aBag.occurrencesOf(4) == 2);

assert(aBag.occurrencesOf(99) == 0);
assert(!aBag.remove(99));

assert(aBag.remove(5));
assert(aBag.occurrencesOf(5) == 0);
assert(aBag.remove(4));
assert(aBag.occurrencesOf(4) == 1);
```
The Data Members and Constructor

// File name: Bag.h
#include <vector>

template<
class Type> // Allow type arguments
class Bag {
private:
    std::vector<Type> elements; // Can be any type
    int n;

public:
    // --constructor
    Bag() {
        elements.resize(20); // size 20 is arbitrary
        n = 0;
    }
The `Bag::add` operation adds all new elements to the "end" of the vector. The vector may be resized.

```cpp
// Add element and increase the size (n) by 1
void add(Type const& element) {
    // First make sure there is enough capacity
    if (n == elements.size()) {
        // Grow the vector's capacity by 10
        elements.resize(n + 10);
    }
    // Then add element at the end of the vector
    elements[n] = element;
    // Increase the number of elements
    n++;
}
```
Bag::size

• The \texttt{Bag::size} operation simply returns \( n \), that increases by 1 in \texttt{add} and will decrease by 1 in \texttt{remove}

\begin{verbatim}
// Return the number of elements
// that are currently in this Bag
int size() const {
    return n;
}
\end{verbatim}
Bag::remove

• The Bag::remove operation begins by finding the index of the value to be removed

```cpp
// pre: removalCandidate must define ==
// post: If found, value is removed from this Bag.
// If object is not in this Bag, return false.
bool remove(Type const& value) {
    // Find the index of the element to remove
    // or let index be out of range when not found
    int index = 0;
    while (index < n && value != elements[index]) {
        index++;
    }
    // . . .
```
Bag::remove

- If not found, return false. If found, overwrite it with the most recently added element

```cpp
// element[subscript] == value if found,
// otherwise subscript == size (not found).
if (index == n) {
    return false;
}
else {
    // Overwrite value with the last element
    elements[index] = elements[n - 1];
    // and decrease size by 1
    n--;  
    // Report success to the client
    return true;
}
} // End remove member function
Trace Bag::remove

• Assume this state of aBag<int> where n==4

<table>
<thead>
<tr>
<th>vector location</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>element[0]</td>
<td>5</td>
</tr>
<tr>
<td>element[1]</td>
<td>4</td>
</tr>
<tr>
<td>element[2]</td>
<td>4</td>
</tr>
<tr>
<td>element[3]</td>
<td>9</td>
</tr>
</tbody>
</table>

• After aBag::remove(4) when n-- makes n==3

<table>
<thead>
<tr>
<th>vector location</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>element[0]</td>
<td>5</td>
</tr>
<tr>
<td>element[1]</td>
<td>4 9</td>
</tr>
<tr>
<td>element[2]</td>
<td>4</td>
</tr>
<tr>
<td>element[3]</td>
<td>9</td>
</tr>
</tbody>
</table>

This 9 is no longer in the Bag since n == 3
Bag::occurrencesOf

- Bag::occurrencesOf iterates over the vector to count how often value exists in this Bag

```cpp
// Return how often value exists in this Bag
int occurrencesOf(Type const& value) const {
    int result = 0;
    for (int i = 0; i < n; i++) {
        if (value == elements[i])
            result++;
    }
    return result;
}
```