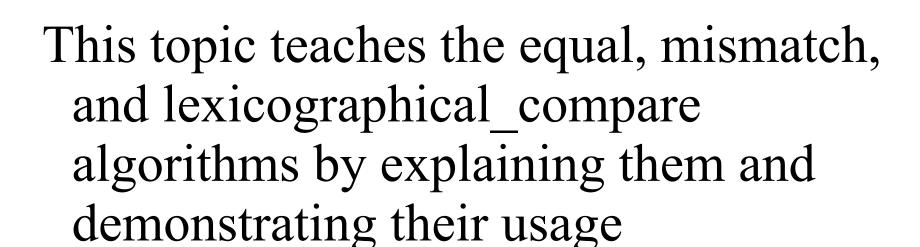
## Equal Mismatch Lexicographical\_Compare



• Figure 16.2 demonstrates comparing sequences of values for equality using algorithms equal, mismatch and lexicographical\_compare.

```
bool result =
  equal(a1.cbegin(), a1.cend(), a2.cbegin());
```

Uses the equal algorithm to compare two sequences of values for equality

The second sequence must contain at least as many elements as the first

- equal returns false if the sequences are *not* of the same length

The == operator (whether built-in or overloaded) performs the element comparisons

The three iterator arguments must be at least *input iterators*They can be used for input from a sequence in the *forward* direction

## result = equal(a1.cbegin(), a1.cend(), a3.cbegin());

- Another version of equal takes a binary predicate function as a fourth parameter
  - Receives the two elements being compared and returns a bool value indicating whether the elements are equal
  - Useful in sequences that store objects or pointers to values rather than actual values, because you can define one or more comparisons
- For example, you can compare Employee objects for age, social security number, or location rather than comparing entire objects
- You can compare what pointers refer to rather than comparing the pointer values
  - The addresses stored in the pointers

```
auto location = mismatch(a1.cbegin(),
a1.cend(), a3.cbegin());
```

- Returns a pair of iterators indicating the location in each sequence of the *mismatched* elements
- If all the elements match
  - The two iterators in the pair are equal to the end iterator for each sequence
- The three iterator arguments must be at least *input iterators*

```
cout << "\nThere is a mismatch between a1 and a3 at location "</pre>
     << ( location.first - a1.begin() )
     << "\nwhere al contains "
     << *location.first << " and a3 contains "
     << *location.second
     << "\n\n";
```

- Determines the actual location of the mismatch in the arrays with the expression location.first - a1.begin()
  - Evaluates to the number of elements between the iterators
- This corresponds to the element number in this example
  - Because the comparison is performed from the beginning of each array
- As with equal, there is another version of mismatch that takes a binary predicate function as a fourth parameter

```
IOHNS HOPKINS
```

```
result = lexicographical compare
   (begin ( c1 ), end ( c1 ),
    begin ( c2 ), end ( c2 ));
```

- Uses the lexicographical compare algorithm to compare the contents of two char built-in arrays
- Four iterator arguments must be at least *input iterators* 
  - Pointers into built-in arrays are random-access iterators
- The first two iterator arguments specify the range of locations in the first sequence
- The last two specify the range of locations in the second sequence

- Use the C++11 begin and end functions to determine the range of elements for each built-in array
- While iterating through the sequences, the lexicographical\_compare checks if the element in the first sequence is less than the corresponding element in the second sequence.
  - If so, the algorithm returns true
  - If the element in the first sequence is greater than or equal to the element in the second sequence
    - The algorithm returns false.
- This algorithm can be used to arrange sequences *lexicographically*
- Typically, such sequences contain strings

This topic taught the equal, mismatch, and lexicographical compare by explaining them and demonstrating their usage