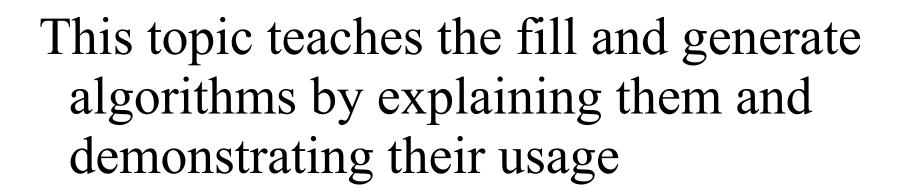
## Fill and Generate



- Figure 16.1 demonstrates algorithms fill, fill\_n, generate and generate\_n
- Algorithms fill and fill n
  - Set every element in a range of container elements to a specific value
- Algorithms generate and generate n
  - Use a generator function to create values for every element in a range of container elements
  - The *generator function* takes no arguments and returns a value that can be placed in an element of the container.

## fill( chars.begin(), chars.end(), '5' );

Uses the fill algorithm to place the character '5' in every element of chars from chars.begin() up to, but not including, chars.end()

- The iterators supplied as the first and second argument must be at least *forward iterators* 
  - They can be used for both input from a container and output to a container in the forward direction

Uses the fill\_n algorithm to place the character 'A' in the first five elements of vector chars

The iterator supplied as the first argument must be at least an output iterator

It can be used to write into a container in the forward direction

The second argument specifies the number of elements to fill The third argument specifies the value to place in each element

generate(chars.begin(), chars.end(), nextLetter);

Uses the generate algorithm to place the result of a call to *generator* function next-Letter in every element of vector chars from chars.begin() up to, but not including, chars.end()

The iterators supplied as the first and second arguments must be at least *forward iterators* 

generate n( chars.begin(), 5, nextLetter );

Uses the generate\_n algorithm to place the result of a call to generator function nextLetter in five elements of vector chars, starting from chars.begin()

The iterator supplied as the first argument must be at least an *output* iterator

When you look at the Standard Library algorithms documentation for algorithms that can receive function pointers as arguments

- You'll notice in the documentation that the corresponding parameters do *not* show pointer declarations

Such parameters can actually receive as arguments

- function pointers
- function objects
- or lambda expressions

For this reason, the Standard Library declares such parameters using more generic names

• For example, the generate algorithm's prototype is listed in the C++ standard document as:

- indicating that generate expects as arguments
  ForwardIterators representing the range of elements to
  process and a *Generator function*
- The standard explains that the algorithm calls the **Generator** function to obtain a value for each element in the range specified by the *ForwardIterators*
- The standard also specifies that the **Generator** must take no arguments and return a value of the element type



