

Write-only Algorithms

fill()

- The Standard Library defines a few algorithms that write container elements without reading any elements
- These are useful for populating containers
- `fill()` assigns the elements using a given value

```
vector<int> v(30);
```

```
// Define vector with 30 elements
```

```
fill(v.begin(), v.end(), 6);
```

```
// Set the value of all its elements to 6
```

fill_n()

- fill_n() is similar, but takes a number instead of the second iterator
- This determines how many elements (starting with the first iterator) will be assigned
- It will return an iterator pointing to the element after the last one which was assigned

// Assign the first 20 elements to 6 and the rest to 99

```
vector<int> v(30);
```

// Define vector with 30 elements

```
auto begin_rest = fill_n(v.begin(), 20, 6);
```

```
fill (begin_rest, v.end(), 99);
```

// Assign the remaining elements to be 99

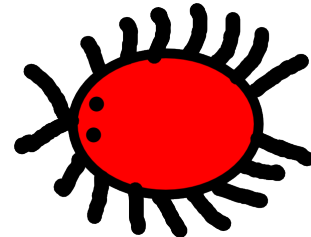
fill_n() and Uninitialized Container

- `fill_n()` is not passed a terminating iterator, so it cannot check whether the element it is writing to is valid

```
vector<int> v;           // Create an empty vector
fill_n(v.begin(), 20, 6); // Assign its first 20 elements - undefined behaviour
```

- We can avoid this problem by using a `back_inserter`

```
fill_n(back_inserter(v), 20, 6);
```



generate() example

```
class square {  
    private:  
        int n{0};  
    public:  
        int operator()() { ++n; return n * n; }  
};
```

```
vector<int> v(30);  
generate(v.begin(), v.end(), square());
```

// Define vector with 30 elements

// Populate it with the first 30 squares