iterator-model (§12.3) zorgt voor grote algemeenheid en efficiëntie. Iterators worden gebruikt om reeksen door te geven aan algoritmen uit de standaardbibliotheek. Bijvoorbeeld:

```
sort(v.begin(),v.end());
```

Zie hoofdstuk 11 en 12 voor details en meer containerbewerkingen.

Een andere manier om het elementen-aantal impliciet te gebruiken is een range-`for` lus:

```
for (auto& x : c)
      x = 0;
```

Hier wordt impliciet `c.begin()` en `c.end()` toegepast, wat ongeveer gelijk is aan de meer expliciete lus.

5.4.3 Invoer- en uitvoerbewerkingen

Voor paren integers betekent `<<` left-shift en `>>` right-shift. Voor `iostream`s zijn ze respectievelijk de uitvoer- en invoeroperator (§1.8, hoofdstuk 10). Voor details en meer I/O-bewerkingen, zie hoofdstuk 10.

5.4.4 Zelfgedefinieerde literals

Een van de doelen van klassen is dat programmeurs typen kunnen ontwerpen en implementeren die dicht tegen de ingebouwde typen aan zaten. Constructors bieden initialisatie die gelijk is aan of groter dan de flexibiliteit en efficiëntie van ingebouwde type-initialisatie, maar voor ingebouwde typen hebben we de beschikking over literals:

- `123` is een `int`.
- `0xFF00u` is een `unsigned int`.
- `123.456` is een `double`.
- `"Verrassing!"` is een `const char[12]`.

Het kan handig zijn om dergelijke literals ook voor een zelfgedefinieerd type te bieden. Dit kan door hun betekenis te definiëren met een geschikt achtervoegsel voor een literal, zodat we krijgen:

- `"Verrassing!"s` is een `std::string`.
- `123s` is `seconden`.
- `12.7i` is `imaginary`, zodat `12.7i+47` is `complex<double>` (namelijk `{47,12.7}`).