Handout – Iteratoren und Algorithmen

Iteratoren

Ein Iterator zeigt auf Elemente eines Containers. Er unterstützt:

- Traversieren (++, *Inkrement-Operator*, --, +=, -=) um ihn zu verschieben
- Dereferenzieren (*, *Dereferenzierungs-Operator*) um auf das Element zuzugreifen

Beispiel

```
for (vector<int>::iterator it = a.begin();
     it != a.end(); ++it) {
    print(*it);
}
```

ist äquivalent zu:

```
for (int i = 0; i < a.size(); ++i) {
    print(a[i]);
}
```

- `a.begin()` ist ein Pointer auf das E `a[0]`
- `a.end()` entspricht einem "past-the-end" Iterator, der auf ein Element nach dem letzten zeigt

Achtung:

- Verwechsle `it` (Position) nicht mit `*it` (Wert)
- `*it.x` wird nicht als `*(it.x)` interpretiert, sondern als `(*it).x`. `it->x` ist äquivalent zu `(*it).x`.

Ranges

Ein Range (Bereich) wird definiert durch ein halboffenes Intervall. Der Bereich $[\ell, r)$ beschreibt die Elemente $\ell, \ell+1, \ldots, r-2, r-1$. Der gesamte Vector ist daher `v.begin(), v.end()`.

Benutzung

Sortieren

```
vector<int> a{1, 4, 5, 5, 2, 5};
sort(a.begin(), a.end());
// a: 1, 2, 4, 5, 5, 5
```

Iterator-Bereiche sind links-inklusive und rechts-inklusive:

```
vector<int> a{1, 4, 2, 5, 2, 5};
sort(a.begin()+1, a.end()-2);
// a: 1, 2, 4, 5, 2, 5
```

Reihenfolge ändern

Reverse dreht einen Vektor um:

```
vector<int> a{1, 2, 3, 4, 5};
reverse(a.begin(), a.end()); // a ist 5, 4, 3, 2, 1
```

Rotate verschiebt einen Vektor nach links:

```
vector<int> a{1, 2, 3, 4, 5};
rotate(a.begin(), a.begin() + 1, a.end());
// a ist 2, 3, 4, 5, 1
rotate(a.begin(), a.begin() + 1, a.end());
// a ist 3, 4, 5, 1, 2
rotate(a.begin(), a.begin() + 3, a.end());
// a ist 1, 2, 3, 4, 5
```

Minimum

```
vector<int> a{1, 4, 5, 5, 2, 5};
vector<int>::iterator it =
    min_element(a.begin(), a.end());
if (it == a.end()) {
    print("Die Liste ist leer");
} else {
    print("Minimum:", *it);
}
```

Ein Element zählen

```
vector<int> a{1, 4, 5, 5, 2, 5};
print("es hat", count(a.begin(), a.end(), 5),
      "mal eine 5 in a");
// ergibt "es hat 3 mal eine 5 in a\n"
```

Ein Element finden

```
vector<int> a{1, 4, 5, 5, 2, 5};
vector<int>::iterator it = find(a.begin(), a.end(), 5);
if (it == a.end()) {
    print("es gibt keine 5.");
} else {
    print("das Element", *it, "existiert in der Liste");
}
```

Einen Vektor füllen

```
vector<int> a{1, 4, 5, 5, 2, 5};
fill(a.begin(), a.end, 0); // a: 0, 0, 0, 0, 0, 0
```

```
vector<int> a{1, 4, 5, 5, 2, 5};
iota(a.begin(), a.end(), 0); // a: 0, 1, 2, 3, 4, 5
```

Elemente löschen

```
vector<int> a{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
a.erase(a.end() - 2);
// a: 0, 1, 2, 3, 4, 5, 6, 7, 9
a.erase(a.begin() + 3, a.begin() + 5);
// a: 0, 1, 2, 5, 6, 7, 9
a.erase(a.begin(), a.end()); // a ist leer
```

Duplikate löschen

```
vector<int> a{1, 4, 5, 5, 2, 5};
sort(a.begin(), a.end()); // sortieren
// a: 1, 2, 4, 5, 5, 5
a.erase(unique(a.begin(), a.end()), a.end());
// a: 1, 2, 4, 5
```

Funktionen als Argumente

Umgekehrt sortieren

```
bool is_greater_than(int lhs, int rhs) {
    return lhs > rhs;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
sort(a.begin(), a.end(), is_greater_than);
// a: 8 8 7 6 5 4 3 2 2
```

Ungerade Elemente finden

```
bool ungerade(int n) {
    return n % 2 == 1;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    find_if(a.begin(), a.end(), ungerade);
if (it != a.end()) {
    print("gerader Wert gefunden:", *it);
} else {
    print("alle Werte sind ungerade");
}
```

Spezielle Elemente löschen

```
bool ungerade(int n) {
    return n % 2 == 1;
}

vector<int> a{2, 6, 3, 5, 8, 4, 2, 7, 8};
vector<int>::iterator it =
    remove_if(a.begin(), a.end(), ungerade);
// a: 2 6 8 4 2 8 2 7 8
a.erase(it, a.end());
// a: 2 6 8 4 2 8
```

Dokumentation

https://en.cppreference.com/w/cpp/algorithm