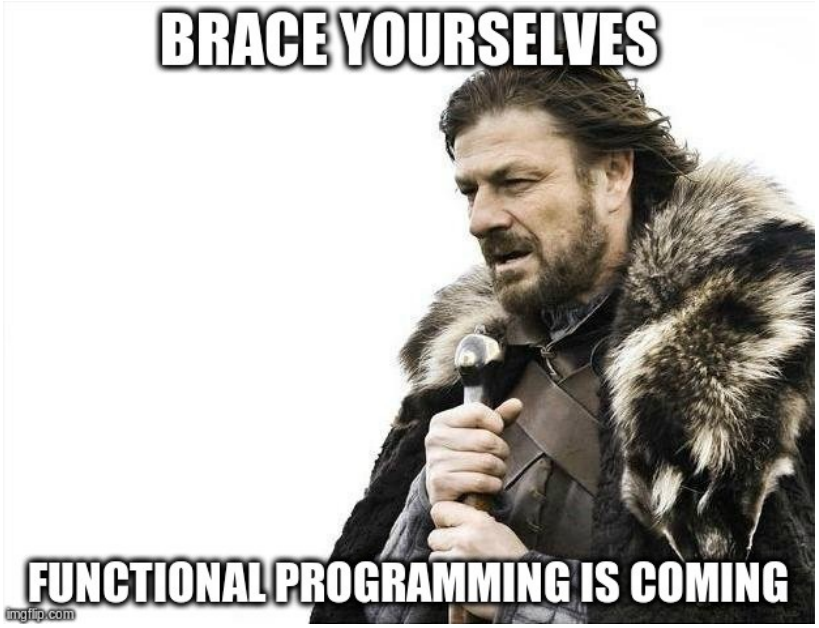
Algorithmen und Lambda

- Einführung std::algorithm
- Lambdas und Funktionsobjekte
- for_each
- Rotate
- Stable Partition
- Transform
- Accumulate
- Execution Policies in C++17



Einführung std::algorithm

- Teil der STL seit Anfang an (1993)
- Sehr mächtig da:
  - Auf jeden Container anwendbar
  - Sehr gut getestet
  - Viele Standartfälle bereits abgedeckt
  - Performance-Optimiert
  - Leicht lesbar
  - **Kombinierbar**
- Aber vor C++11 selten verwendet worden, da Funktionsobjekte umständlich sind.
- http://en.cppreference.com/w/cpp/algorithm

Lambdas und Funktionsobjekte

- Funktionsobjekte haben
  - Einen public Default-Konsturktor
  - Einen public überladenen () - Operator
- Für Prädikate, Generatoren, Comperator, Operationen, etc. für die Algorithmen gedacht.
- Leider sehr umständlich zu Handhaben
- Schwierig, lokale Variablen einzubinden
- https://godbolt.org/g/QpJKdy

Lambdas und Funktionsobjekte

- Die stl bietet bereits einige fertige Funktionsobjekte
  - http://en.cppreference.com/w/cpp/utility/functional
- https://godbolt.org/g/7sKuaL

Lambdas und Funktionsobjekte

- Seit C++ 11: Lambdas
  - Kurzschreibweise für Funktionsobjekte
  - Extrem Flexibel
  - Mehr Optimierungspotential für den Compiler, da Funktion nur lokal verwendet wird
  - Können direkt in den Algorithmen-Aufruf eingebaut werden
- Syntax gewöhnungsbedürftig.
  - (Parameter) {Funktionsinhalt} → noch wie bei normaler Funktion
  - [Capture] () {} → Gibt an, wie lokale Variablen „gefangen“ werden sollen
    - & bedeutet per Referenz
    - = bedeutet per Kopie
    - Auch individuell belegbar [&i, &j, =k] (){}
  - Rückgabewert nach () mittel „ - > Rückgabetyp“
    - Wird auch implizit generiert
- https://godbolt.org/g/jKLdge

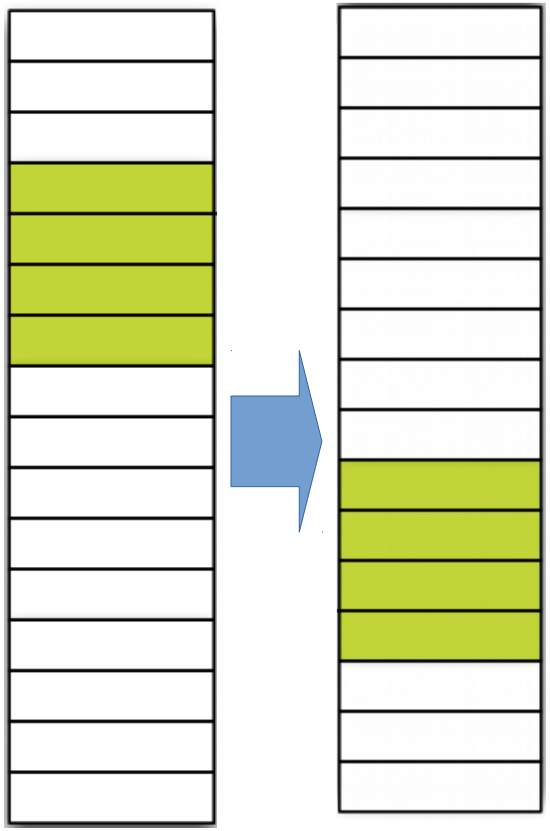
for_each

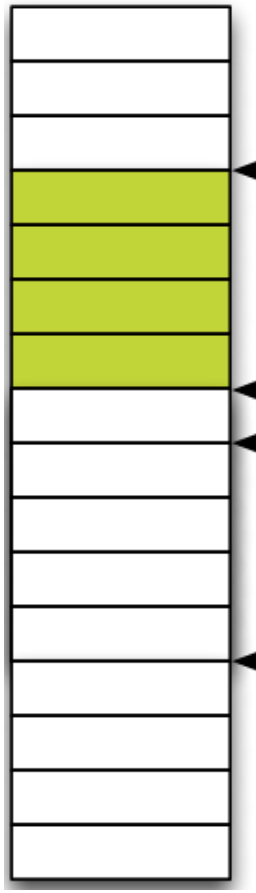
- Eine Schleife über alle Werte zwischen den beiden angegeben Iteratoren
- Kann über den ganzen Container laufen, muss aber nicht.
- Kein Performance unterschied zu anderen Schleifen
- https://godbolt.org/g/tdq2rW
- http://quick-bench.com/fjyHmrBgwFRe3qQyN9m0FXJECPE

Rotate

- Schiebt eine Sequenz an einen andere Stelle
- Sortierung der Elemente wird beibehalten
- Übergabewerte sind
  - Iterator zum Anfang der Original-Sequenz
  - Iterator zum Anfang der zu verschiebenden Sequenz
  - Iterator zum Ende der Original-Sequenz
- Rückgabewert ist ein Iterator auf das Ende der ersetzen Sequenz (C++11)
- https://godbolt.org/g/2DyyFN

Rotate

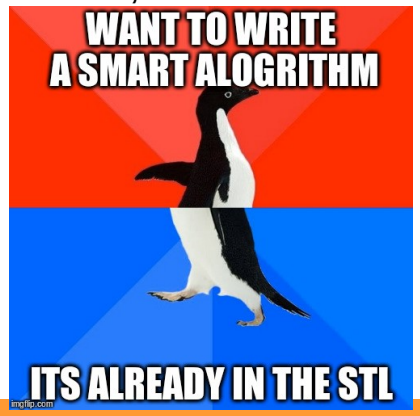


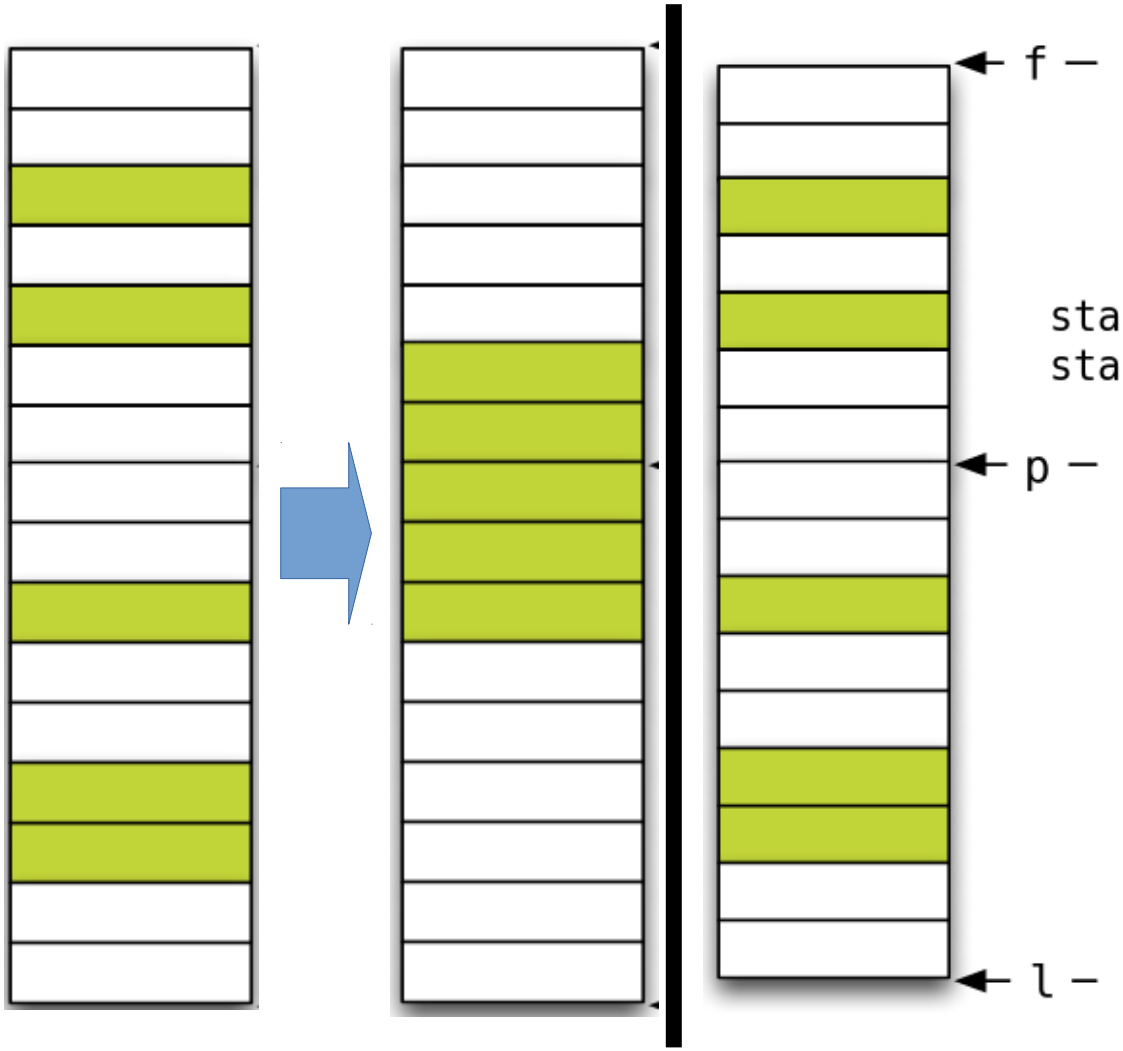



```
if (p < f) return { p, rotate(p, f, l) };
if (l < p) return { rotate(f, l, p), p };
return { f, l };
```

Stable Partition

- Verschiebt die Werte, die ein gewisses Prädikat erfüllen nach vorne, den Rest nach hinten
- stable_partition garantiert, dass die relative Ordnung der Elemente beibehalten wird
  - Ansonsten partition
- Übergabewerte sind
  - Iterator auf Anfang der Sequenz
  - Iterator auf Ende der Sequenz
  - Prädikat nach dem Sortiert wird
- Rückgabewert ist Iterator auf den ersten Wert, der das Prädikat nicht erfüllt
- https://godbolt.org/g/dFDcN6





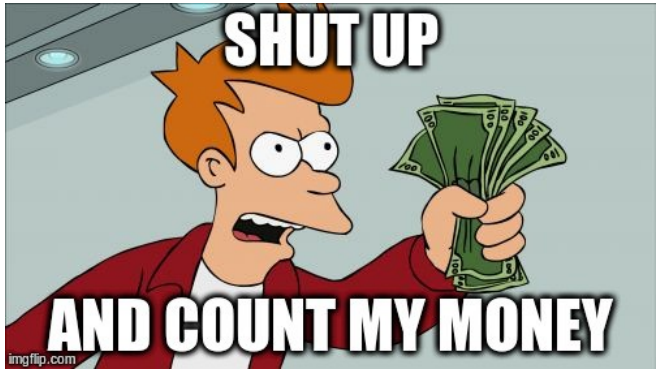

```
stable_partition(f, p, not1(s))
stable_partition(p, l, s)
```

Transform

- Führt eine Operation auf die Sequenz aus und schreibt das Ergebnis in eine neue Sequenz
- Nutzer muss garantieren, dass in der Ausgabesequenz genug Speicherplatz zur Verfügung steht
- Übergabewerte sind
  - Iterator auf den Start der Original-Sequenz
  - Iterator auf das Ende der Original-Sequenz
  - Iterator auf den Start der Ergebnis-Sequenz
  - Operation
- Rückgabewert ist ein Iterator auf das nächste Element, nach dem Ende der Ergebnis-Sequenz
- https://godbolt.org/g/A1Ky6Q

Accumulate

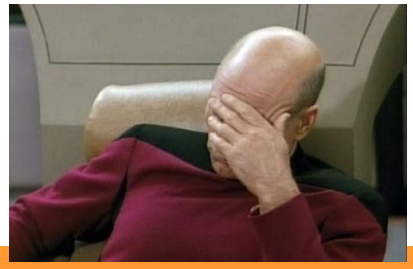
- Führt „+“ - Operation oder selbst definierte Operation auf eine Sequenz aus und gibt das Ergebnis zurück
- Die selbst definierte Operation, muss den Initialwert und den Wert aus der Sequenz als Übergabewert haben
- Übergabewerte sind
  - Iterator auf den Start der Sequenz
  - Iterator auf das Ende der Sequenz
  - Initial-Wert
  - Operation (optional, ansonsten wird „+“ ausgeführt)
- Rückgabewert ist das Ergebnis
- **Ist in <numeric> und nicht in <algorithm>**
- https://godbolt.org/g/vXveJv



Execution Policies in C++17

- Ab C++17 kann bei fast allen Algorithmen ein zusätzlicher Parameter angegeben werden, wie der Algorithmus ausgeführt wird.
- Allerdings nur „Empfehlung“ an den Compiler
  - **std::execution::sequenced_policy**: Sequenzielle Ausführung, keine Parallelisierung
  - **std::execution::parallel_policy**: Parallele Ausführung in seperaten Threads, Ausführungsreihenfolge soll garantiert werden
  - **std::execution::parallel_unsequenced_policy**: Parallele und Vektorisierte Ausführung, Ausführungsreihenfolge wird nicht garantiert

- **Aber: Data Races, Deadlocks etc. müssen vom User behandelt werden!!!**



Nächstes Mal:
Die Utility-Library