Funktoren, Lambdas und Ranges

☑ **Lösung zu Aufgabe 11.1** Die Ausgabe des vorgegebenen Programms ist wie folgt:

```
3 9 5 7
1 2 8 9
4 7 2 3
```

☑ **Lösung zu Aufgabe 11.2** Folgende Definition des Komparators führt zu der gewünschten Sortier-Reihenfolge. Wir vergleichen zuerst anhand des Alters und zwar mit Hilfe des Operators `>`, so dass die ältere Person zuerst in der sortierten Reihenfolge drankommt. Bei gleichem Alter werden die Namen mit Hilfe des sonst üblichen Operators `<` verglichen, so dass gleichaltrige Personen auf die normale alphabetische Art und Weise sortiert werden.

```
struct ComparePerson {
  bool operator()(const Person& p1, const Person& p2) const {
    return p1.m_age >  p2.m_age
       || (p1.m_age == p2.m_age && p1.m_name < p2.m_name);
  }
};
```

☑ **Lösung zu Aufgabe 11.3** Sei der Namensraum `namespace vw = std::views` definiert. Dann führt beispielsweise folgender Ausdruck an Stelle von `???` zum korrekten Ergebnis:

```
const int m = 5, n = 3;
for(auto [i, ii] : vw::iota(1)
  | vw::filter(    [](auto i){ return i % m == 0;                    })
  | vw::transform( [](auto i){ return std::make_pair(i, i*i);       })
  | vw::take_while([](auto p){ return p.second < std::pow(10.0, n); })
  | vw::reverse)
{
  std::cout << std::setw(n) << i  << " * "
            << std::setw(n) << i  << " = "
            << std::setw(n) << ii << std::endl;
}
```

Bei dieser Lösung ist zu beachten, dass es sich bei der ursprünglichen Range `vw::iota(1)` um eine (im Prinzip) unendliche Folge von Zahlen handelt. Durch die Verwendung von `vw::take_while` wird sie aber in eine endliche Folge überführt. Diese Tatsache ist äusserst wichtig für die darauffolgende Anwendung von `vw::reverse`. Hätten wir nämlich die Pipeline beispielsweise mit `vw::iota(1) | vw::reverse` begonnen, dann hätten wir sozusagen nach dem letzten Eintrag in einer unendlichen Folge verlangt, was einfach keinen Sinn ergibt.