Funktoren, Lambdas und Ranges

Aufgabe 11.1 Führen Sie folgendes Programm in Ihrem Kopf aus. Was ist die jeweilige Ausgabe bei den drei Aufrufen des Funktionen-Templates `print`?

```
template <typename Range>
void print(Range&& range) {
  for(const auto& x : range)
    std::cout << x << " ";
  std::cout << std::endl;
}
int main() {
  namespace rn = std::ranges; namespace vw = std::views;
  std::vector<int> vector = {1, 4, 2, 3};
  print(vector
      | vw::transform([](int x){ return 2*x + 1; }));
  rn::iota_view iota(1, 10);
  print(iota
      | vw::filter([](int x){ return x <= 2 || x >= 8; }));
  std::istringstream sstream("4 7 2 3 0 6 1 5 9 8");
  print(vw::istream<int>(sstream)
      | vw::take_while([](int x){ return x != 0; }));
}
```

Aufgabe 11.2 Schreiben Sie einen Komparator `ComparePerson` für den Typ `Person` (das heisst, einen Funktor mit zwei Argumenten vom Typ `const Person&` und mit `bool` als Rückgabewert). Mit diesem Komparator soll eine Liste von Personen absteigend nach Alter sortiert werden. Bei gleichem Alter soll normal aufsteigend alphabetisch nach dem Namen sortiert werden.

```
struct Person {
  std::string m_name;
  int m_age;
};

std::vector<Person> people = {{"Alice", 33}, {"Bob", 33}, {"Eve", 44}};
// Reihenfolge vor dem Sortieren: Alice, Bob, Eve
std::ranges::sort(people, ComparePerson{});
// Reihenfolge nach dem Sortieren: Eve, Alice, Bob
```

Aufgabe 11.3 Ersetzen Sie die mit `???` markierte Stelle in folgendem Code links durch eine Pipeline bestehend aus Ranges, Views und Lambda-Funktionen, so dass die gezeigte Ausgabe in der Box auf der rechten Seite gemacht wird.

```
const int m = 5, n = 3;
for(auto [i, ii] : ???) {
  std::cout << std::setw(n) << i  << " * "
            << std::setw(n) << i  << " = "
            << std::setw(n) << ii << std::endl;
}
```

```
30 *  30 = 900
25 *  25 = 625
20 *  20 = 400
15 *  15 = 225
10 *  10 = 100
 5 *   5 =  25
```

Etwas präziser ausgedrückt soll abhängig von den gegebenen Werten `m` und `n` eine absteigend sortierte Liste aller durch `m` teilbaren natürlichen Zahlen `i` und deren Quadrate `ii = i * i` ausgegeben werden, aber nur solange das Quadrat `ii` mit `n` oder weniger Dezimalstellen dargestellt werden kann.

Schlagen Sie selbstständig die Funktionen `std::make_pair` und `std::pow` in der Dokumentation der Standard-Bibliothek nach. Diese beiden Funktionen werden sich für diese Aufgabe als sehr nützlich herausstellen.