Stream Input und Output

☑ **Lösung zu Aufgabe 9.1** Die erhaltenen Ausgaben sind wie folgt und die wichtigsten Überlegungen dazu werden weiter unten noch einmal näher erläutert:

```
100 // 1
010 // 2
011 // 3
001 // 4
100 // 5
```

Das `good`-Bit (erste Stelle) ist dann gesetzt, wenn weder das `eof`-Bit noch das `fail`-Bit gesetzt ist (und auch nicht das `bad`-Bit, was aber für diese Aufgabe nicht relevant ist). Im Prinzip zeigt dieses `good`-Bit an, dass eine weitere Eingabe über den vorhandenen Stream möglich ist.

Das `eof`-Bit (zweite Stelle) wird zu dem Zeitpunkt gesetzt, wenn das Ende des Eingabestroms erreicht ist. In diesem Zusammenhang ist das Verhalten der Funktion `std::getline` bei `5` bemerkenswert, weil mit dem Zeichen `'o'` zwar bereits das letzte Zeichen im Eingabestrom konsumiert wird, aber der Stream trotzdem noch nicht realisiert, dass danach nichts mehr kommt. Ebenso interessant ist, dass bei `4` zwar versucht wird, zwei Strings einzulesen, aber trotzdem das Ende des Streams nicht erreicht wird. Dies liegt daran, dass die erste Eingabe eines Werts vom Typ `double` scheitert und danach (wegen des gesetzten `fail`-Bits) keine weiteren Zeichen aus dem Eingabestrom mehr entfernt werden.

Das `fail`-Bit (dritte Stelle) ist dann gesetzt, wenn die vorhergehende Eingabe-Operation nicht erfolgreich war. Dies kann passieren, wenn der Stream bereits leer ist (so wie bei `3`) oder wenn die im Stream vorhandene Zeichenfolge nicht dem verlangten Typ entspricht (so wie bei `4`).

☑ **Lösung zu Aufgabe 9.2** Der gezeigte Code berechnet die korrekte Anzahl ignorierter Zeichen und zählt das Resultat auch richtig zu der übergebenen `int`-Referenz hinzu. Das Problem jedoch ist, dass immer noch ein zusätzliches Zeichen ignoriert wird. Bei der letzten Überprüfung der Schleifenbedingung wird trotzdem `is.get()` aufgerufen und somit ein zusätzliches Zeichen aus dem Stream entfernt. Um genau dies zu verhindern, könnte stattdessen beispielsweise `is.peek()` verwendet werden.

☑ **Lösung zu Aufgabe 9.3** Wir verwenden jeweils ein Semikolon `';'` als Trennsymbol nach der Ausgabe der drei einzelnen Member-Variablen. Dies erleichtert uns die Arbeit beim Einlesen, weil das Ende der Repräsentation einer Member-Variablen sehr einfach erkennbar ist.

```
std::ostream& operator<<(std::ostream& os, const Student& s) {
  os << s.m_bachelor << ';' << s.m_name << ';';
  for(double grade : s.m_grades) os << ' ' << grade;
  return os << ';';
}
std::istream& operator>>(std::istream& is, Student& s) {
  // Eingabe m_bachelor:
  if(is >> s.m_bachelor && is.peek() == ';' && is.get() == ';') {
    // Eingabe m_name:
    if(std::getline(is, s.m_name, ';') && !is.eof()) {
      // Eingabe m_grades:
      s.m_grades.clear();
      for(double grade; is.peek() != ';' && is >> grade; )
        s.m_grades.push_back(grade);
      if(is && is.get() == ';')
        return is; // Eingabe erfolgreich
    }
  }
  is.setstate(std::ios_base::failbit);
  return is; // Eingabe gescheitert
}
```