Überladen von Methoden/Operatoren

Aufgabe 5.1 Überlegen Sie sich bei den folgenden Codezeilen der Reihe nach, ob es sich jeweils um eine gültige Definition (respektive Überladung) einer Funktion `f` handelt, oder ob die Signatur mit einer früheren Funktion im Konflikt steht und deshalb zu einem Compiler-Fehler führt.

```
char f() { return '\n'; }
bool f() { return true; }
void f(char a) {}
void f(char b) {}
void f(bool a) {}
void f(char a, bool b) {}
void f(bool a, char b) {}
void f(int& i) {}
void f(int&& i) {}
void f(const int& i) {}
```

Aufgabe 5.2 Sei `Rational` ein Struct, das eine rationale Zahl (also einen Bruch) mit Zähler `m_num` (Numerator) und Nenner `m_denom` (Denominator) vom Typ `int` darstellt. Die implizite Konvertierung von einem `int` wird unterstützt, indem die Ganzzahl n auf den Bruch $\frac{n}{1}$ abgebildet wird. Zudem sei der Vergleichsoperator `operator<` als Member-Funktion implementiert. Welche der vier Vergleiche zweier Zahlen entweder vom Typ `Rational` oder `int` führen zu einem Compiler-Fehler in folgendem Code? Was müsste man ändern, damit alle vier Vergleiche einwandfrei funktionieren?

```
struct Rational {
  int m_num, m_denom;
  Rational(int num, int denom = 1)
  : m_num(num), m_denom(denom) {}
  bool operator<(const Rational& other) {
    return m_num * other.m_denom < other.m_num * m_denom;
  }
};

Rational r = {1,2}, s = {3,4};
if(r < s) {} // Rational < Rational
if(s < 5) {} // Rational < int
if(6 < r) {} //      int < Rational
if(7 < 8) {} //      int < int
```

Aufgabe 5.3 Erweitern Sie folgende Klasse `FancyVector`, indem Sie den Indexoperator `operator[]` implementieren. Wie Ihnen dies von anderen Programmiersprachen vielleicht schon bekannt ist, sollen neben den positiven Indizes auch negative Indizes unterstützt werden. Das heisst konkret für ein Objekt `v` vom Typ `FancyVector`, dass `v[-1]` beispielsweise den letzten Eintrag von `v.m_vec` zurückgeben soll, `v[-2]` soll den zweitletzten Eintrag zurückgeben und so weiter.

Achten Sie darauf, dass alle Einträge per Referenz zurückgegeben werden und somit auch von ausserhalb der Klasse verändert werden können. Stellen Sie aber trotzdem sicher, dass beispielsweise der Inhalt eines Objekts vom Typ `const FancyVector` nicht verändert werden kann.

```
class FancyVector {
private:
  std::vector<int> m_vec;
public:
  explicit FancyVector(size_t size, int value = 0)
  : m_vec(size, value) {}
};
```