Aufgabe 1. Hier testen wir grundlegendes Verständnis der Syntax und Rechenregeln von `C++`. Für jeden korrekten Typ und jeden korrekten Wert gibt es einen Punkt. Für falsche Antworten (auch wenn sie nahe bei der richtigen Lösung liegen, was immer das heisst, gibt es keinen Punkte.

Aufgabe	Typ	Wert	was testen wir?
(a)	`bool`	`false`	Syntax von Fliesskommaliteralen
(b)	`double`	0.5	Gemischte Ausdrücke, Operator-Präzedenz (unäres $-$)
(c)	`float`	0.5	Gemischte Ausdrücke, ganzzahlige Division
(d)	`bool`	`true`	Operator-Präzedenzen (logisch)
(e)	`bool`	`false`	Unsigned-Syntax und Unsigned-Arithmetik
(f)	`int`	11	Operator-Präzedenzen (arithmetisch)
(g)	`int`	1	Explizite Konversion (arithmetisch)
(h)	`bool`	`false`	Fliesskommarepräsentation
(i)	`bool`	`true`	Fliesskommarepräsentation
(j)	`double`	5.5	Linksassoziativität

Aufgabe 2.

```
// POST: gibt genau dann true zurueck, wenn n eine Quadratsumme ist.
// Falls der Rueckgabewert true ist, so gilt n = a^2 + b^2
bool is_square_sum (unsigned int n, unsigned int& a, unsigned int& b) {
// es genuegt, alle Paare (a,b) mit a >= b und a^2 <= n zu pruefen
for (a = 0; a * a <= n; ++a)
for (b = 0; b <= a; ++b)
if (n == a * a + b * b) return true;
return false;
}
```

Man kann das noch effizienter, oder auch weniger effizient machen. In die Bewertung geht nur die Korrektheit ein. Bei Lösungen, die auf `std::sqrt` beruhen, sollten wir grosszügig sein, auch wenn die Korrektheit nach Standard nicht garantiert ist, z.B.

```
bool is_square_sum (unsigned int n, unsigned int& a, unsigned int& b) {
// es genuegt, alle a mit a^2 <= n zu pruefen
for (a = 0; a * a <= n; ++a) {
// einziger Kandidat fuer b ist sqrt(n - a^2)
b = (unsigned int)std::sqrt(n - a * a);
if (n == a * a + b * b) return true;
}
return false;
}
```

Aufgabe 3. Hier gibt es für jede Funktion 10 Punkte.
Die Funktion f berechnet die grösste Zehnerpotenz kleiner oder gleich n. Die Funktion g dreht die Dezimaldarstellung der Zahl n um; für $n = 1254$ z.B. wird 4521 ausgegeben.

Sei $n = \sum_{i=0}^{p} n_i 10^i$ mit $n_p \neq 0$ (führende Stelle). Dann wird $r = n_0$ gesetzt und die Zahl ohne ihre letzte Ziffer rekursiv umgedreht (`k = g (n / 10)`). Zum Ergbnis $k = \sum_{i=0} p - 1 n_{p-i} 10^i$ wird dann $rh(n) = n_0 10^p$ addiert, und wir erhalten das umgederehte n.
Eine Besonderheit ist, dass abschliessende Nullen in der Eingabe im Resultat nicht erscheinen (z.B. $n = 1250 \rightarrow 521$), wenn jemand dies aber nicht angibt, sollten wir keine Punkte abziehen.

Aufgabe 4. Die Ausgabe ist 11. In den folgenden Kommentaren bezeichnen die doppelten oder dreifachen Namen Variablen gleichen Namens in geschachtelten Scopes. Es gibt einen Punkt für jeden von 9 korrekt annotierten Werten (unter Berücksichtigung von Folgefehlern) und einen Punkt für den Ausgabewert.

```
0: #include<iostream>
1:
2: int main() {
3: int i = 1; // i = 1
4: int k = 2; // k = 2
5: int l = 0; // l = 0
6: {
7: int k = i; // kk = i = 1 (aus Zeile 3)
8: l = k; // l = kk = 1 (aus Zeile 7)
9: {
10: int i = 3; // ii = 3
11: int k = i; // kkk = ii = 3 (aus Zeile 10)
12: l += k; // l = l + kkk = 4 (aus Zeilen 8, 11)
13: }
14: int l = 5; // ll = 5
15: l += k; // ll = ll + kk = 6 (aus Zeilen 14, 7)
16: i += l; // i = i + ll = 7 (aus Zeilen 3, 15)
17: }
18: l += i; // l = l + i = 11 (aus Zeilen 12, 16)
19:
20: std::cout << l << std::endl;
21: }
```

Aufgabe 5. (a) ist kanonisch:

```
private:
unsigned int u; // Absolutwert
bool n; // Vorzeichen (true heisst negativ)
```

Natürlich ist es auch korrekt, wenn das Vorzeichen z.B. mit einem `int` aus $\{-1, 1\}$ repräsentiert wird.
Bei (b) steckt Arbeit nur in `operator+=`, wenn man es richtig macht.

```
Int::Int (int x)
{
if (x < 0) {
u = -x;
n = true;
} else {
u = x;
n = false;
}
}

Int::Int (unsigned int x, bool negative)
: u (x), n (negative)
{}

Int Int::operator-() const {
return Int(u, !n);
}

// || u > v || u <= v
// ====================================
// (u,+)+(v,+) || (u+v,+)
// (u,+)+(v,-) || (u-v,+) || (v-u,-)
// (u,-)+(v,+) || (u-v,-) || (v-u,+)
// (u,-)+(v,-) || (u+v,-)
Int& Int::operator+=(const Int& y) {
if (n == y.n)
u += y.u;
else
if (u > y.u)
u -= y.u;
else {
u = y.u - u;
n = !n;
}
return *this;
}

Int& Int::operator-=(const Int& y) {
return operator+=(-y);
}
```

Aufgabe 6. Hier sind fünf Fehler zu finden, und für jeden gibt es 2 Punkte. Fünf Punkte gibt es noch für die (mehr oder weniger) korrekte Beschreibung der Programmsemantik.

- Es fehlt die `#include<iostream>`-Direktive, ohne die `std::cout` und `std::cin` nicht bekannt sind. Das Inkludieren von `cmath` ist kein Fehler, aber überflüssig.
- In Zeile 3 fehlen die Anführungszeichen um `n = ?`
- In den Zeilen 7 und 8 müssen alle `,` durch `;` ersetzt werden.
- In Zeile 9 muss es `==` anstatt `=` heissen; Da `i-j` und `i+j` L-Werte sind, kann ihnen ohnehin nichts zugewiesen werden.
- In Zeile 15 fehlt das abschliessende `;`

Das entsprechend korrigierte Programm zeichnet den Buchstaben "X" in einem $n \times n$-Raster, z.B. für $n = 9$:

```
* *
* *
* *
* *
*
* *
* *
* *
* *
```