Grundlagen

☑ **Lösung zu Aufgabe 1.1** Die erhaltene Ausgabe ist `1134`. Auf Zeile 1 wird eine Variable `a` zuerst definiert, danach wird ihr Wert auf `1` gesetzt und schliesslich wird dieser neue Wert `1` ausgegeben. Auf Zeile 2 wird eine neue Variable `b` als *Kopie* von `a` initialisiert; die spätere Modifikation von `a` auf den Wert `2` hat deshalb keinen Einfluss auf den ausgegebenen Wert von `b`. Auf Zeile 3 wird `c` definiert als *Referenz* auf die Variable `a`; hier ist deshalb die spätere Modifikation von `a` auf den Wert `3` bei der Ausgabe von `c` ersichtlich. Auf Zeile 4 wird `d` definiert als *Referenz* auf die von `c` referenzierte Variable `a`; auch das ist möglich und führt zur erneuten Ausgabe der Variable `a`, die nun den Wert `4` hat.

☑ **Lösung zu Aufgabe 1.2** Sämtliche fehlerhafte Zeilen sind in den Kommentaren des folgenden Codes markiert und kurz erklärt.

```
int x; std::cin >> x;

int a = x+1;
const int b = x+2;
constexpr int c = x+3; // ERR: Wert x+3 zur Kompilierzeit unbekannt
a = 1;
b = 2; // ERR: const-Variable ist nicht modifizierbar
c = 3; // ERR: constexpr-Variable ist nicht modifizierbar

int& ra = a;
int& rb = b; // ERR: nonconst-Referenz auf const-Variable verboten
int& rc = c; // ERR: nonconst-Referenz auf constexpr-Variable verboten
ra = 1;
rb = 2;
rc = 3;

const int& cra = a;
const int& crb = b;
const int& crc = c;
cra = 1; // ERR: const-Referenz ist nicht modifizierbar
crb = 2; // ERR: const-Referenz ist nicht modifizierbar
crc = 3; // ERR: const-Referenz ist nicht modifizierbar

constexpr int cca = a; // ERR: Wert a zur Kompilierzeit unbekannt
constexpr int ccb = b; // ERR: Wert b zur Kompilierzeit unbekannt
constexpr int ccc = c;
```

☑ **Lösung zu Aufgabe 1.3** Zum einen muss der Parameter `param` der Funktion `f` per Referenz genommen werden, so dass überhaupt die Möglichkeit besteht, den Wert der übergebenen Variable `x` innerhalb der Funktion zu ändern. Zum anderen muss `param` aber auch wieder per Referenz von `f` zurückgegeben werden, so dass beispielsweise auch der zweite und dritte Aufruf von `f` in dem verschachtelten Ausdruck `f(f(f(x)))` die übergebene Variable `x` ändern kann. Zu guter Letzt sollte man nicht vergessen, den berechneten Wert `param * param` tatsächlich auch wieder in `param` zu speichern.

```
double& f(double& param) {
  param = param * param;
  return param;
}
```