Aufgabe 1.

Postcondition für `f`. Rückgabewert ist die grösste Zahl $r \in \mathbb{N}$, für die $r^5 \leq x$. In anderen Worten $r = \lfloor \sqrt[5]{x} \rfloor$.

Precondition für `g`. $y^5 \leq x$ und $z^5 > x$.

Postcondition für `g`. Rückgabewert ist die grösste Zahl $r \in \mathbb{N}$, $y \leq r < z$, für die $r^5 \leq x$.
Es wird eine binäre Suche auf dem Intervall $[x, y)$ durchgeführt. Das Abbruchkriterium ist in Zeile 3 fomuliert. In Zeile 4 wird der Mittelwert a des aktuellen Suchintervalls bestimmt, dann in Zeile 5 und 6 der Wert a^5 berechnet und in Zeile 6 und 7 das Suchintervall für den nächsten rekursiven Aufruf entspr. angepasst.

Punktvergabe. 12 Punkte für die Postcondition für `g`, 8 Punkte für die Postcondition von `f`. Die Precondition ist nicht notwendig, aber falls nicht vorhanden, muss in der Postcondition stehen, was dann für die entspr. Werte von y und/oder z passiert. (Sonst 6 Punkte Abzug.)
Punkte gibt es grob nach folgendem Schema.

- Es hat etwas mit Potenzieren zu tun (2 Punkte);
- Es wird a^5 berechnet (5 Punkte);
- Es findet eine binäre/iterative o.ä. Suche statt. (8 Punkte);
- Es wird $\lfloor \sqrt[5]{x} \rfloor$ berechnet. (14 Punkte);
- Vollständig korrekte Postconditions. (20 Punkte).

Aufgabe 2.

```
bool square_free(unsigned int x)
{
  for (unsigned int i = 2; i*i < x; ++i)
    if (x % (i*i) == 0) return false;
  return true;
}
```

Die Formulierung der Aufgabe ist absichtlich etwas umständlich. Wenn es einen Teiler gibt, der Quadratzahl ist, so gibt es natürlich auch einen Primteiler, der Quadratzahl ist. Es müssen also nur alle Quadratzahlen kleiner als x getestet werden. Alle korrekten Lösungen geben volle Punktzahl, egal wie ineffizient und/oder umständlich sie geraten sind.

Aufgabe 3.

```
class perm {
public:
  unsigned int pi(unsigned int i) const;
  // PRE: i <= 2
  // POST: return image of i
  void concat(const perm& phi);
  // POST: set *this to *this o phi
private:
  unsigned int p;
  // Represent a permutation of {0,1,2} by a three digit number
  // with mutually distinct digits in base 3 system.
  // Least significant digit denotes the image of 1,
  // most significant digit denotes the image of 3.
  // For efficiency, we encode these numbers into a single integer p.
};
```

8 Punkte für die Repräsentation. Man kann natuerlich auch drei separate Variablen definieren, aber dann kann man unten nicht mit Schleifen arbeiten. Jede korrekte Repräsentation (d.h. alle Permutationen können dargestellt werden und jeder zulässige Wert entspricht einer Permutation) gibt jedenfalls volle Punktzahl. -3P falls nicht kommentiert und keine Invariante angegeben.

```
unsigned int pow3(unsigned int i)
{
  unsigned int d = 1;
  for (; i > 0; --i) d *= 3;
  return d;
}

unsigned int pi(unsigned int i)
{
  assert(i <= 2);
  return p/pow3(i)%3;
}
```

8 Punkte für die Implementation der Funktion. Fehlende Pre- oder Postcondition: je -3P.

```
void perm::concat(const perm& phi)
{
  unsigned int r = 0;
  for (unsigned int i = 0; i < 3; ++i) {
    unsigned int j = phi.p/pow3(i)%3; // image of i under phi
    j = p/pow3(j)%3; // image of j under p
```

```
    r += j*pow3(i);
  }
  p = r;
}
```

10 Punkte für die Implementation der Funktion. Abhängig von der Repräsentation gibt es hier natürlich verschiedene Möglichkeiten. Vielleicht gibt es Leute, die "einfach" die Gruppentafel implementieren (36 Fälle). Was korrekt ist, gibt volle Punktzahl.
Kein Punktabzug, falls die Operation umgekehrt (d.h. mit vertauschten Rollen der Argumente) implementiert wurde. -3P für fehlendes `const`.

Aufgabe 4.

a) `((1+((1*6)/4))==3)&&((2.1/1.3)<1.8)`
`((1+(6/4))==3)&&((2.1/1.3)<1.8)`
`((1+1)==3)&&((2.1/1.3)<1.8)`
`(2==3)&&((2.1/1.3)<1.8)`
`false&&((2.1/1.3)<1.8)`
`false`

b) `(((((12/6)/2)*3.0)+1)<4)||((2.0*2)>1)`
`((((2/2)*3.0)+1)<4)||((2.0*2)>1)`
`(((1*3.0)+1)<4)||((2.0*2)>1)`
`((3.0+1)<4)||((2.0*2)>1)`
`(4.0<4)||((2.0*2)>1)`
`false||((2.0*2)>1)`
`(2.0*2)>1`
`4.0>1`
`true`

Logische Klammerung: (6 Punkte). Jedes nicht korrekte Klammerpaar einen halben Punkt Abzug. **Eine einzelne** Klammer eines Paares vergessen $\longrightarrow$ noch kein Abzug.
Auswertung: (12 Punkte). Die beiden Operanden eines arithmetischen Operators gleichzeitig auswerten $\longrightarrow$ kein Abzug. Folgefehler einer falschen Klammerung $\longrightarrow$ kein Abzug. Short-circuit evalution vergessen: -3 Punkte. Jede andere der Operationen oder Typen nicht korrekt: je -1 Punkt.

Aufgabe 5. Die Schlüsselworte, für die es jeweils einen Punkt gibt, sind **fett** gekennzeichnet.

a) Ein Programm ist eine **Folge von Operationen**/Befehlen, die in einer **rechner-verständlich**en Form dargestellt werden. Ein Algorithmus löst ein Problem, wobei er in **endlicher Zeit** für **jede Eingabe** das korrekte Ergebnis liefert. Ein Algorithmus **löst ein konkretes Problem**, während ein Programm aus einer "sinnlosen" Folge von Operationen bestehen kann (Syntax $\leftrightarrow$ Semantik). Im Gegensatz zu einem Programm muss ein Algorithmus nicht in **rechner-verständlicher** Form beschrieben werden.

b) Ein Programm P akzeptiert eine Menge $L \subset \mathbb{N}$, wenn P für jede Eingabe $n \in L$ die Ausgabe "Ja" liefert und entweder "Nein" ausgibt oder unendlich lange arbeitet, falls $n \notin L$.

Für beide Fälle jeweils einen Punkt, wenn sie korrekt beschrieben werden.

c) Wir konstruieren eine zweidimensionale **Tabelle** $(a_{ij})_{i,j\in\mathbb{N}}$, deren Zeilen durch die **Folge** P_i**,** $i \in \mathbb{N}$**, aller Programme**, und deren Spalten durch die **Folge** i**,** $i \in \mathbb{N}$**, aller möglichen Eingaben** bestimmt sind. Hierbei ist $a_{ij} = 1$, falls die Eingabe j vom Programm P_i akzeptiert wird, und $a_{ij} = 0$, sonst. Nun ist

$$\textsc{Diag} := \{n \in \mathbb{N} \mid a_{nn} = 0\},$$

das heisst, die Mengen aller Zahlen $i \in \mathbb{N}$, so dass $i \notin M(P_i)$.

6 Punkte in jedem Fall, falls DIAG korrekt definiert wird, ob nun in Form einer Tabelle beschrieben oder nicht. Andernfalls gibt das Auftauchen der oben fettgedruckten Terme Anlass zur Vergabe jeweils eines Punktes.

d) Beweis durch Widerspruch: Nehmen wir an, es gibt ein Programm P, das DIAG akzeptiert. Da wir alle Programme in der Folge P_i, $i \in \mathbb{N}$, aufzählen, gilt $P = P_k$, für irgendein $k \in \mathbb{N}$. Betrachte das Verhalten von P auf der Eingabe k. Angenommen, P akzeptiert k. Dann ist nach Definition von $k \notin$ DIAG, im Widerspruch zur Annahme, dass P DIAG akzeptiert. Nehmen wir also an, dass P die Eingabe k nicht akzeptiert. Dann aber ist nach Definition $k \in$ DIAG, im Widerspruch zur Annahme, dass P DIAG akzeptiert. Dann aber kann es kein Programm P geben, welches DIAG akzeptiert.

6 Punkte in jedem Fall für einen korrekten Beweis, ob indirekt oder nicht. Die wesentlichen Merkmale sind, dass

- alle Programme erfasst sind/aufgezählt werden können (2 Punkte);
- und dass sich für jedes Programm P_i die akzeptierte Menge $M(P_i)$ im Element i von der Menge DIAG unterscheidet (Diagonalisierung; 3 Punkte).

Aufgabe 6. Keine Lösung angegeben.

Allgemein. Wie auch bei den Zwischenklausuren wird ein kleinerer Syntaxfehler (vergessenes Semikolon o.ä.) ohne Punktabzug toleriert. Ab dem zweiten jedoch wird je nach Schwere mind. 1 Punkt abgezogen. Wenn das ganze nicht mehr als `C++` erkennbar ist, sondern nur eine Art Pseudocode bildet, gibt es Null Punkte.