Prüfung Informatik D-MATH/D-PHYS 26.1.2009

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Aufgabe 1.

a) (i) Das ist in der Tat so. Die Antwort ist **Ja**.

(ii) Das ist natürlich falsch. Die Antwort ist **Nein**. In Teilaufgabe b) gibt es ein Beispiel dafür. Nämlich den Befehl `If... then go to line...` in Zeile 5.

(iii) Hier ist die Antwort auch **Nein**. Das folgende Programm ist ein Gegenbeispiel:

```
1. Go to line 2
2. Go to line 1
3. End
```

(iv) Die Antwort ist **Ja**. Ein Algorithmus ist so definiert, dass er das beschriebene tut. Wenn eine Berechnungsvorschrift für den erlaubten Parameterbereich nicht terminiert oder nicht das richtige Resultat liefert, dann ist sie kein Algorithmus.

(v) Die Antwort ist **Nein**. Ein Algorithmus sollte allgemein formuliert sein. Es mag zwar streitbar sein, was jetzt "ohne weiteres" für die Übertragung in eine andere Programmiersprache bedeutet - und ganz objektiv gesehen kann der Aufwand auch beträchtlich sein - aber ein Algorithmus ist sicher nicht an eine konkrete Sprache "gebunden".

b) Das Programm berechnet $a \operatorname{div} b$ und $a \bmod b$ für positive Zahlen, also ganzzahlige Division mit Rest. Das Resultat der Division wird in Zeile 6 (`Register(3)`) und der Rest in Zeile 7 (`Register(1)`) ausgegeben. Die Idee ist folgendermassen. Es wird so lange b von a abgezogen wie der Wert in `Register(1)` (in dem ursprünglich a steht) grösser oder gleich b ist. Gleichzeitig wird gezählt, wie oft dies möglich ist. Dies ist der Wert der Division. Was dann am Ende noch in `Register(1)` übrig bleibt ist der Rest der Division.

a) Es tritt ein Problem auf, wenn $b = 0$, das heisst, wenn in `Register(2)` eine 0 eingelesen wird. Dies entspricht ja auch der mathematischen Intuition, dass man nicht durch 0 dividieren darf. Wenn $b = 0$ wird bei der Subtraktion in Zeile 9 am Wert von `Register(1)` nie etwas verändert. Somit wird der Vergleich in Zeile 5 immer positiv ausfallen; und wir springen wieder zu Zeile 9. Das Programm terminiert nicht. Beachten Sie, dass dieses Verhalten unabhängig vom Wert von a ist.

Punktvergabe.

a) Einen Punkt für jede richtige Antwort.

b) Maximal 10 Punkte. Wird von "ganzahliger Division mit Rest" oder "div und mod" gesprochen, so gibt es die volle Punktezahl. Werden bei der Spezifikation der Ausgabe das Resultat der Division und der Rest verwechselt, so gibt es 3 Punkte Abzug. Für teilweise zutreffende Aussagen, wie zum Beispiel "wenn $a < b$, dann wird a ausgegeben" können bis zu 4 Punkte (nach Ermessen des Korrektors) vergeben werden.

c) Es gibt 5 Punkte für die Aussage, dass das Programm nicht terminiert, falls $b = 0$. Es gibt 2 Punkte Abzug, wenn dieses Verhalten auch vom Wert von a abhängig gemacht wird (was es nicht ist).

Aufgabe 2.

a)

```
3 + 3 / 2 + 3.0 / 2
3 + 1 + 3.0 / 2
4 + 3.0 / 2
4 + 3.0 / 2.0
4 + 1.5
4.0 + 1.5
5.5
```

Es ist nicht festgelegt in welcher Reihenfolge `3/2` und `3.0/2` ausgewertet werden. Es ist jedoch durch die Links-Assoziativität des Plusoperators festgelegt, dass `3 + 1` zuerst verrechnet wird, bevor noch `1.5` dazu addiert wird.

b)

```
y * 7.0 + (y - 1) * 3.0
false * 7.0 + (y - 1) * 3.0
false * 7.0 + (false - 1) * 3.0
0 * 7.0 + (false - 1) * 3.0
0 * 7.0 + (0 - 1) * 3.0
0.0 * 7.0 + (0 - 1) * 3.0
0.0 + (0 - 1) * 3.0
0.0 + (-1) * 3.0
0.0 + (-1.0) * 3.0
0.0 + (-3.0)
-3.0
```

Bei einer arithmetischen Operation von `bool` und `int` wird der boolsche Wert zu einer ganzen Zahl konvertiert. Der Standard schreibt vor, dass `false` zu `0` und `true` zu irgend einem anderen Wert (meistens `1`) konvertiert wird.

c)

```
! ! y || (y = true)
! ! false || (y = true)
! true || (y = true)
false || (y = true)
false || true
true
```

Hier ist wichtig, dass die linke Seite des Oder-Operators (||) zuerst ausgewertet wird, da wir auf die short-circuit Evaluation achten müssen. Diese schlägt jedoch fehl und somit wird auch die rechte Seite noch ausgewertet. Hier ist dann zu beachten, dass es sich nicht um einen Vergleich, sondern um eine Zuweisung handelt. Diese gibt den neuen Wert der beschriebenen Variable zurück. Somit ist der Wert der rechten Seite `true`; damit auch der ganze Ausdruck.

Punktvergabe. 6 Punkte für jede der Teilaufgaben. Minus 0.5 Punkte für jeden fehlenden Zwischenschritt. Am Ende sollte die Punktezahl (zu Gunsten des Studenten) auf die nächste ganze Zahl gerundet werden. Minus 3 für falsches Endresultat. Fehlende short-circuit Evaluation in Teilaufgabe c) wird mit minus 2 geahndet. Wird in Teilaufgabe c) die Zuweisung mit einem Vergleich verwechselt, so gibt es (nur) 2 Punkte Abzug. In diesem Falle sind also trotz falschem Ensresultat bis zu 4 Punkte möglich.

Aufgabe 3. Wir haben die Funktion `prim` zur Verfügung, die testet, ob eine Zahl eine Primzahl ist.

```
// PRE: n > 0
// POST: Der Rueckgabewert ist true g.d.w. n eine Primzahl ist.
bool prim(unsigned int n);
```

Im folgenden implementieren wir nun den Test, ob eine Zahl "aufeinanderfolgend" ist.

```
// PRE: n > 0
// POST: Der Rueckgabewert ist true g.d.w. die Primfaktorzerlegung
//       von n nur aus aufeinanderfolgenden Primzahlen besteht.
bool aufeinanderfolgend(unsigned int n);

unsigned int first = 0; // smallest prime factor
unsigned int last = 0; // largest prime factor


//  look for the smallest prime factor
for (unsigned int i = 2; i < n; ++i) {
    if (n % i == 0) {
```

```
            first = i;
            break;
        }
    }

    // n itself is prime (or 1)
    if (first == 0) return true;

    // look for the largest prime factor
    for (unsigned int i = n - 1; i > 1; --i) {
        if (n % i == 0) {
            if (prim(i)) {
                last = i;
                break;
            }
        }
    }

    // there is only one type of prime factor
    if (first == last) return true;

    // check for all prime factors in between
    for (unsigned int i = first; i <= last; ++i) {
        if (n % i != 0)
            if (prim(i))
                return false;
    }
    return true;
```

Die Idee ist folgende. Wir suchen zuerst nach dem kleinsten und nach dem grössten Primfaktor. Wenn die beiden gleich sind, dann können wir schon `true` zurück geben, da es dann nur diesen einen Faktor gibt (Es handelt sich bei `n` um eine Primzahlpotenz). Falls dies jedoch nicht der Fall ist, überprüfen wir alle Zahlen zwischen diesen beiden Faktoren. Falls wir dabei eine Zahl finden, die kein Teiler aber eine Primzahl ist, dann "fehlt" einer der Primteiler, und die Zahl `n` ist nicht aufeinanderfolgend. Andernfalls, können wir einfach `true` zurückgeben.

Punktvergabe. Dies war sicherlich eine der schwierigsten Aufgaben der Prüfung, deshalb gibt es ganze 30 Punkte für eine korrekte Lösung. Die Lösung sollte syntaktisch korrekt sein. Ein bis zwei kleinere Syntaxfehler können jedoch ohne Ahndung verziehen werden. Darüber hinaus gibt es für jeden Syntaxfehler einen Punkt Abzug. Dabei ist zu beachten, dass mehrfach vorkommende Fehler der selben Art nicht mehrfach geahndet werden, sondern ebenfalls nur einen Punkt Abzug nach sich ziehen.

Desweiteren sollten bei der Korrektur folgende Richtlinien eingehalten werden.

- Eine korrekte Lösung gibt immer die volle Punktezahl, egal wie "unschön" oder ineffizient.
- Wird die 1 richtig abgehandelt, so gibt das 4 Punkte.
- Werden Primzahlen richtig abgehandelt, so gibt das 4 Punkte.
- Werden Primzahlpotenzen richtig abgehandelt, so gibt das züsatzlich noch 6 Punkte.
- Werden allgemeine Zahlen richtig abgehandelt (nicht aber die 1 und Primzahlpotenzen), so gibt es 16 Punkte.
- Es werden keine "aufeinanderfolgenden" Zahlen verworfen (keine false negative) gibt 8 Punkte.
- Ein fehlender Teiler wird nicht gefunden (viele false negative) gibt -10 Punkte.
- Für ein Programm, dass nicht terminiert, oder keine Rückgabewerte hat (selbst wenn dies nur manchmal der Fall ist) gibt es - je nach Schwere - bis zu -15 Punkte.
- Bedarf das Programm einer wesentlich Änderung, damit es korrekt funktioniert gibt es 30 - 6 = 24 Punkte
- Wird `int a[n];` verwendent, gibt es -2 Punkte (Es handelt sich um statische Allokation mit einer variablen Länge).
- Fehlen `delete` Operationen am Ende des Programmes, gibt es -1 Punkt.
- Werden Primteiler erkannt: 4 Punkte.
- Wird von einem Primteiler ausgehend die nächste Primzahl gesucht: 2 Punkte.
- Wird der kleinste und/oder grösste Primteiler gefunden: 5 Punkte.
- Gibt es eine Schleife zum "wegdividieren" von Teilern: 2 Punkte.

Aufgabe 4.

a) Die Schlüsselwörter `private` und `public` dienen dazu, den Zugriff auf Klassenmitglieder zu regeln. Funktionen und Variablen, die mit `private` bezeichnet sind (bei Klassen ist dies ist auch der Standardwert, falls man nichts Explizites hinschreibt), sind nur innerhalb der Klasse selbst erreichbar. Funktionen und Variablen, die mit `public` bezeichnet sind, können von einem Programm aufgerufen oder verändert werden, das eine Instanz der Klasse hält. In Structs gibt es keine Möglichkeit, das Schlüsselwort `private` zu verwenden. Alles ist standardmässig `public`.

Das Schlüsselwort `private` ist demnach sinnvoll, wenn man gewisse Details der Implementation vor dem Benutzer der Klasse verstecken will.

b) Das Beispiel in der Vorlesung war eine Klasse zur Repräsentation von rationalen Zahlen. Hier mag es sinnvoll sein, die Variablen für den Zähler und den Nenner zu verstecken, damit zum Beispiel der Nenner nicht auf den Wert 0 gesetzt werden kann.

```
class rational {
   private:
      int n;
      int d; // INV: d != 0
};
```

Punktvergabe.

a) Diese Teilaufgabe gibt bis zu 8 Punkte. 4 Punkte gibt es für die allgemeine Erklärung der Funktionalität von `private` und `public`. 4 Punkte gibt es für die Aussage, dass man gewisse Details der Implementation **verstecken** will.

b) Diese Teilaufgabe gibt bis zu 8 Punkte. 4 Punkte für ein korrektes Beispiel. Die weiteren 4 Punkte gibt es, wenn auch noch eine sinnvolle Begründung angegeben wird. Es ist völlig in Ordnung, wenn einfach das Beispiel aus der Vorlesung erwähnt wird.

Aufgabe 5.

a) Die gefragten Werte sind nach dem Ablauf der Funktion wie folgt.

```
b[0] == 1
b[1] == 4
b[2] == 6
b[3] == 4
b[4] == 1
```

b) Die Nachbedingung der Funktion ist wie folgt.

```
// PRE: [b, e) ist ein gueltiger Bereich
// POST: Die Funktion berechnet die Zeile (e - b) des
//       Pascalschen Dreiecks.
void f(unsigned int* b, unsigned int* e);
```

Punktvergabe. Für jede der Teilaufgaben gibt es 10 Punkte.

a) Für die korrekten Werte gibt es 2 Punkte. Für korrekte Zwischenwerte während des Ablaufs der äusseren Schleife gibt es maximal 4 zusätzliche Punkte.

b) Hier gilt es zu beachten, dass das Pascalsche Dreieck auch unter dem Namen Tartaglia Dreieck bekannt ist. Generell gilt, dass es nicht um den Namen "Pascalsches Dreieck" geht, sondern um die Berechnungsvorschrift. Falls also jemand beschreibt, wie man das Pascalsche Dreieck konstruiert und dann sagt, dass jetzt die Zeile (e - b) ausgegeben wird, so gibt dies natürlich die volle Punktezahl. Wird das Pascalsche Dreiecks korrekt dargestellt, aber keine Interpretation oder Benennung angegeben: -2 Punkte. Es gibt einen -4 Punkten falls keine oder eine falsche Zeilenangabe gemacht wurde. Richtige Zeile, aber Angabe in n statt in b - e: -2 Punkte. Andere analytische (falsche) Beschreibungen: Pro korrekt beschriebener Position 1 Punkt (bei korrekter Symmetrie daher je 2 Punkte), jedoch maximal 6 Punkte. Dies bezieht sich auf teilweise richtige Aussagen wie zum Beispiel "im ersten und im letzten Feld steht immer eine 1", oder "im zweiten und im zweitletzten Feld steht immer $e - b - 1$", oder "die Werte sind symmetrisch".

Aufgabe 6. Als erstes halten wir einmal fest, dass es nicht darauf ankommt, wie viele der b_i tatsächlich den Wert 1 haben. Wenn wir zeigen können, dass eine Zahl der Form

$$b' = b_i 2^{-i},\ i \in \mathbb{N} \text{ und } b_i \in \{0, 1\}$$

immer eine endliche Darstellung im Dezimalsystem hat, dann wird das auch für eine endliche Summer solcher Zahlen der Fall sein. Natürlich können wir dann auch annehmen, dass $b_i = 1$ ist, denn andernfalls ist $b' = 0$, was trivialerweise eine endliche Darstellung im Dezimalsystem hat.

Nun beachte man, dass man diese Zahl wie folgt schreiben kann,

$$b' = b_i \frac{5^i}{5^i \cdot 2^i} = 1 \cdot 5^i \cdot 10^{-i} = 5^i \cdot 10^{-i},\ i \in \mathbb{N}.$$

Dies ist nun eine Summe von 5^i Zahlen der Form 10^{-i} und hat also nach dem vorher gesagten eine endliche Darstellung.

Punktvergabe. Die Aufgabe gibt insgesamt 16 Punkte.

4 Punkte gibt es für die Beobachtung, dass man die Aussage auf die entsprechende Aussage fuer Zweierpotenzen reduzieren kann.

12 Punkte gibt es für die Gleichung $2^{-i} = 5^i 10^{-i}$, auch wenn nicht genau gesagt wird, wie sie benutzt wird.

2 Punkte gibt es für die schlichte Behauptung (ohne Argument), jede Zweierpotenz habe eine endliche Dezimaldarstellung.

Keine Punkte gibt es für "Argumente" des folgenden Typs:

- Die Behauptung ist klar, da $\{0, 1\}$ eine Untermenge von $\{0, ..., 9\}$ ist.

- Jede Zweierpotenz ist auch eine Zehnerpotenz.
- Die Binärzahlen sind eine Teilmenge der Dezimalzahlen.