Prüfung Informatik D-MATH/D-PHYS 12.2.2010

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Aufgabe 1.

Die Variable `x` hat zu Beginn jeder Auswertung den Typ `int` und den Wert 2.

Ausdruck	Typ	Wert
`3 * 1 + 1.5 / 3`	`double`	`3.5`
`11 * 19 % 21 * 13 % 23 * 3 % 3`	`int`	`0`
`7 / x * 3.0 * x`	`double`	`18.0`
`x != 2 && x - 1 != 1 || x + 1 == 3`	`bool`	`true`
`17 / 2 == 8.5 && 7 * 3 == 21.0`	`bool`	`false`
`(++x - 1) / 2`	`int`	`1`
`x++ - 1 / 2`	`int`	`2`

Punktvergabe. Für jeden korrekt genannten Typ gibt es **+1 Punkt** und für jeden korrekt genannten Wert gibt es **+2 Punkte**. Insgesamt macht das **+21 Punkte**.

Aufgabe 2.

a) Die Binärexpansion der dezimalen Zahl 0.8 bricht nicht ab und lautet 0.110011001100.... Sie kann also geschrieben werden als $0.\overline{1100}$.

b) Natürlich ist $0.\overline{1100}$ nicht exakt darstellbar im System $\mathcal{F}^*(2,2,-2,2)$, und deshalb müssen wir runden. Die nächstgrössere Zahl ist $1.0 \cdot 2^0$, was einer dezimalen 1 entspricht. Die nächstkleinere Zahl in $\mathcal{F}^*(2,2,-2,2)$ ist $1.1 \cdot 2^{-1}$, was einer dezimalen 0.75 entspricht. Demnach runden wir natürlich ab, und die Darstellung von 0.8 entspricht $1.1 \cdot 2^{-1}$ oder in der ausführlichen Schreibweise $+(1 \cdot 2^0 + 1 \cdot 2^{-1})2^{-1}$.

c)

Berechnung	Exaktes Resultat	$\mathcal{F}^*(2,2,-2,2)$	Rundung
$1.1 \cdot 2^{-1} + 1.1 \cdot 2^{-1}$	$1.1 \cdot 2^0$	$1.1 \cdot 2^0$	keine
$1.1 \cdot 2^0 + 1.1 \cdot 2^{-1}$	$1.001 \cdot 2^1$	$1.0 \cdot 2^1$	$\downarrow$
$1.0 \cdot 2^1 + 1.1 \cdot 2^{-1}$	$1.011 \cdot 2^1$	$1.1 \cdot 2^1$	$\uparrow$
$1.1 \cdot 2^1 + 1.1 \cdot 2^{-1}$	$1.111 \cdot 2^1$	$1.0 \cdot 2^2$	$\uparrow$

Das Schlussresultat der Berechnung ist demnach $1.0 \cdot 2^2$, was exakt der dezimalen 4 entspricht. Das zeigt, dass sich Fehler manchmal wechselseitig aufheben können!

Punktvergabe. Es gibt **+4 Punkte** für die Teilaufgabe a). Der Korrektor kann die Hälfte der Punkte anrechnen, wenn jemand den Konversionsalgorithmus aus der Vorlesung anwenden will, sich aber verrechnet. Insbesondere gibt es **-2 Punkte** Abzug, wenn jemand die Periode vergisst. Wenn jemand das führende 0. nicht hinschreibt, gibt es **-1 Punkt**.

Die Teilaufgabe b) gibt ebenfalls **+4 Punkte**. Wurde in a) ein falsches Ergebnis berechnet, so gibt es hier trotzdem die volle Punktezahl, falls die Darstellung richtig angegeben wird *und* eine Rundung notwendig war. Falls jedoch aufgrund eines falschen Resultates in Teilaufgabe a) eine exakte Darstellung in $\mathcal{F}^*(2, 2, -2, 2)$ möglich ist (und diese auch korrekt angegeben wird), dann gibt es nur **+2 Punkte**.

Teilaufgabe c) gibt insgesamt **+10 Punkte**. Dabei sollte jede der vier einzelnen Additionen mit **+2 Punkten** bewertet werden. Falls sich jemand auf dem Weg verrechnet, so sollen die folgenden Additionen trotzdem mit der vollen Punktezahl bewertet werden, sofern sie korrekt sind (Folgefehler nicht bestrafen). Das gilt übrigens auch wenn schon mit einem falschen Resultat aus der Teilaufgabe b) gestartet wird. Falls bei einer Addition in die falsche Richtung gerundet wird, gibt es **-1 Punkt** Abzug. Falls aufgrund eines Folgefehlers keine eindeutige nächste Zahl vorliegt zu der gerundet werden soll, dann wird die Auswahl des Studenten in jedem Fall richtig gezählt.

Zusätzlich gibt es noch **+2 Punkte** für das korrekte Schlussresultat, respektive die Bemerkung, dass tatsächlich das exakte Resultat von 4 heraus kommt.

Aufgabe 3.
Die folgende Implementierung ist sehr simpel, tut aber das geforderte.

```
// POST: Gibt true zurueck, falls es eine natuerliche Zahl a gibt, fuer
//       die n == a*a*a gilt. Andernfalls wird false zurueck gegeben.
bool is_cube(unsigned int n) {
  for (int i = 0; i <= n; ++i) {
    if (i*i*i == n) return true;
  }
  return false;
}
```

Man kann natürlich noch implementieren, dass die Funktion früher abbricht, falls `i*i*i` schon grösser als `n` ist, oder ähnliches. Die Aufgabe verlangt jedoch nur nach der Korrektheit des Algorithmus und nicht nach Effizienz.

Punktvergabe. Bei dieser Programmieraufgabe gilt, dass die volle Punktezahl von **+15 Punkten** vergeben wird, sobald die Implementierung die Nachbedinung erfüllt. Natürlich muss die Funktion auch für die Eingabewerte `0` und `1` das richtige Resultat liefern, nämlich `true`. Sollte letzteres nicht der Fall sein gibt es pro Versäumnis einen Abzug von **-3 Punkten**. Bei Syntaxfehlern drücken wir beim ersten ein Auge zu. Danach gibt es

pro Syntaxfehler -**1 Punkt** Abzug; jedoch keinen vielfachen Abzug für die Wiederholung des selben Syntaxfehlers.

Die weitere Abstufung der Punkte ist vom Korrektor nach Sichtung der Abgaben zu bestimmen.

Aufgabe 4.
Diese Aufgabe stimmt exakt mit der Aufgabe "Towers of Hanoi" aus den Vorlesungsunterlagen überein. Es wurde lediglich der Kontext geändert. D.h. anstatt Scheiben auf Stiften sind es hier Container auf Stellplätzen, und die Bedingung, dass keine grössere Scheibe auf einer kleineren zu liegen kommt, wird hier von der Sortierung der Container nach dem Gewicht übernommen.

Hier also die Implementierung wie sie auch in den Lösungen aus den Vorlesungsunterlagen zu finden ist.

```
// PRE:  1 <= from, to <= 3; from != to
// POST: Gibt eine Reihe von Bewegungen aus, die ausgefuehrt werden
//       muessen, um n Container vom Stellplatz from auf den Stellplatz
//       to zu verschieben.
void move_containers (const unsigned int n, const int from, const int to)
{
  if (n > 0) {
    // move topmost n-1 containers from from to helper site 6-from-to
    move_containers(n-1, from, 6-from-to);
    // move bottommost container from from to to
    std::cout << "move(" << from << "," << to << ")";
    // move the n-1 containers from the helper site to to
    move_containers(n-1, 6-from-to, to);
  }
}
```

Punktvergabe. Diese Aufgabe gibt insgesamt **+15 Punkte**. Bei Syntaxfehlern drücken wir beim ersten ein Auge zu. Danach gibt es pro Syntaxfehler -**1 Punkt** Abzug; jedoch keinen vielfachen Abzug für die Wiederholung des selben Syntaxfehlers. Für andere Mängel aller Art kann der Korrektor um -**1** bis -**4 Punkte** von den folgenden Grundregeln abweichen.

Die groben Richtlinien für die weitere Punktevergabe sind wie folgt:

1. Eine blosse Ansammlung von `couts` gibt **0 Punkte**.
2. Eine Ansammlung von `couts`, welche in Schleifen verpackt sind, geben ebenfalls **0 Punkte**, falls bei den Schleifen kein sinnvolles Schema erkennbar ist.

3. Sinnvolle Schleifen für kleine Zahlen, d.h. $n <= 4$, geben **+5 Punkte**.
4. Führt jemand die Rekursionsidee oder das Stichwort "Hanoi" an, und sei es auch nur in einem Prosasatz, so gibt das **+5 Punkte**.
5. Hat jemand einen halbwegs sinnvollen Rekursionscode abgeliefert, dann gibt das **+10 Punkte**, auch wenn es nicht ganz funktionieren sollte.
6. Die Korrekte Lösung gibt natürlich die vollen **+15 Punkte**.

Bemerkung: Während der Prüfung stellte sich die Frage, ob man auch eine andere Funktionssignatur verwenden darf, z.B. vier anstatt drei Argumente. Dies möchten wir mit ja beantworten. Wichtig an dieser Aufgabe ist schlussendlich die Ausgabe der `move` Anweisungen. Wenn diese stimmen, dann gibt es auch die volle Punktezahl. Allerdings muss bei der abgeänderten Funktion auch die Vor- und Nachbedingung entsprechend angepasst worden sein. Es muss klar sein, wie der Aufrufer allfällige zusätzliche Parameter verwenden soll. Wenn dies nicht der Fall ist, dann gibt es einen Abzug.

Aufgabe 5.

a)
```
// POST: Gibt den Abstand von p zum Ursprung an.
double distance(const point& p) {
  return std::sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
}
```

b)
```
struct gerade {
  // INV: a != b
  point a, b;
}
```

Eine Gerade kann ganz einfach durch zwei Punkte dargestellt werden. Für die Eindeutigkeit, wie sie in der Aufgabenstellung verlangt ist, können wir fordern, dass die beiden Punkte nicht identisch sein dürfen.

c)
```
// PRE: a != b
// POST: Gibt eine gerade zurueck, die durch die Punkte a und b geht.
gerade berechne_gerade(const point& a, const point& b){
  gerade g;
  g.a = a;
  g.b = b;
  return g;
}
```

Aufgrund der gewählten Repräsentation ergibt sich die Implementierung der Funktion `berechne_gerade` ganz natürlich. Beachten sie, dass wir die Invariante des `structs gerade` einfach weiter propagieren, indem wir fordern, dass `a!=b` auch für einen Aufruf der Funktion `berechne_gerade` gelten muss.

Punktvergabe. Für die Teilaufgabe a) gibt es insgesamt **+7 Punkte**. Syntaxfehler geben je **-1 Punkt**.

Die Teilaufgabe b) wird mit insgesamt **+8 Punkten** belohnt. Dabei ist die Umsetzung der korrekten Invariante **+4 Punkte** wert und ein geeigneter `struct` ebenfalls **+4 Punkte**.

Die Teilaufgabe c) gibt insgesamt **+8 Punkte**. Hier werden **+2 Punkte** für die Einhaltung der Invarianten aus b) vergeben und **+6 Punkte** für die eigentliche Implementierung. Wie gehabt, Syntaxfehler **-1 Punkt**.

Es versteht sich von selbst, dass auch andere Lösungen möglich sind. Z.B. kann man eine Gerade durch einen Punkt und eine Richtung dargestellen. Eine solche Alternative verdient natürlich auch die volle Punktezahl, sofern die Anforderungen erfüllt werden.

Aufgabe 6.

Teilaufgabe	Klasse	Algorithmus
a)	B	"Gehe durch das Feld hindurch und unterhalte zwei Zähler. Wenn eine gerade Zahl angetroffen wird (Test mit % 2), erhöhe den einen Zähler. Wenn eine ungerade Zahl angetroffen wird, dann erhöhe den anderen Zähler. Am Ende vergleiche man die beiden Zähler."
b)	A	"Greife auf die ersten 10 Zahlen zu und gib sie aus."
c)	C	"Sortiere die Zahlen im Feld. Gehe durch das sortierte Feld hindurch und speichere den Index, an welchem bislang die kleinste Differenz zweier aufeinanderfolgender Zahlen gefunden wurde. Die beiden gesuchten Zahlen sind aufeinanderfolgend und finden sich nach dem Durchlaufen an dem Index, den man sich gemerkt hat. Berechne die Differenz dieser beiden Zahlen."
d)	B	"Finde die grösste und die kleinste Zahl in dem Feld. Berechne die Differenz der gefundenen Zahlen."
e)	D	"Für jede Zahl in dem Feld gehe man durch das ganze Feld hindurch und prüfe, welche der anderen Zahlen Teiler sind. Falls ein Teiler gefunden wird, gebe man ihn aus."
f)	B	"Gehe durch das Feld hindurch und unterhalte eine Variable, die die bisherige Summe unterhält. Bei jeder neuen Zahl addiere man sie dazu."
g)	C	"Sortiere die Zahlen in dem Feld. Berechne die Summe der ersten $\lceil n/2 \rceil$ Zahlen, indem man das Feld bis zur $\lceil n/2 \rceil$-ten Stelle durchlaufe, und die Summe in einer Variablen aufaddiere."

Punktvergabe. Jede der Teilaufgaben bekommt bei korrekter *und* optimaler Beantwortung **+4 Punkte**. Mit "optimal" sprechen wir hier die Laufzeiten resp. die Effizienz des Algorithmus an. Die optimalen Laufzeiten können der Tabelle oben entnommen

werden, welche wir ohne Beweis angeben.

Die detailierte Abstufung ist wie folgt. Wenn der Algorithmus eklatant falsch ist oder fehlt, gibt es keine Punkte, selbst wenn die Laufzeit stimmen sollte, weil man nicht entscheiden kann, ob die Laufzeit nur zufällig stimmt. Wenn der angegebene Algorithmus korrekt, aber nicht optimal ist, gibt es **+2 Punkte** für die korrekte Angabe der Laufzeit (bezüglich des suboptimalen Algorithmus), jedoch keine Punkte für den Algorithmus. Wenn der angegebene Algorithmus korrekt und optimal ist *und* die richtige Laufzeit angegeben wird, dann gibt es **+4 Punkte**.

Falls ein optimaler Algorithmus angegeben wird, aber keine oder eine falsche Laufzeit, dann gibt es je nach Teilaufgabe folgende Punkte:

a) **+1 Punkt**

b) **+1 Punkt**

c) **+2 Punkte**

d) **+2 Punkte**

e) **+2 Punkte**

f) **+1 Punkt**

g) **+2 Punkte**

Ende der Musterlösung zur Prüfung vom 12.2.2010.