Prüfung Informatik D-MATH/D-PHYS 9.8.2010

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Aufgabe 1.

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

Aufgabe	Ausdruck	Typ	Wert
a)	`x[0] * 2.5 / 2`	`double`	`2.5`
b)	`111 % 10 - a * 2`	`int`	`-1`
c)	`z == false`	`bool`	`true`
d)	`x[0] = b`	`double`	`0.0`
e)	`z && b == 1 || a == 1`	`bool`	`true`
f)	`&*p`	Zeiger auf `double`	`x[0]`
g)	`*p = a + 3`	`double`	`4.0`
h)	`&x[0] == p`	`bool`	`true`
i)	`a == *(p = &x[1])`	`bool`	`true`
j)	`++p`	Zeiger auf `double`	`x[1]`

Punktvergabe. Für jeden korrekt genannten Typ gibt es **+1 Punkt** und für jeden korrekt genannten Wert gibt es **+1 Punkte**. Insgesamt macht das **+20 Punkte**.

Aufgabe 2.

a) Eine Menge ist *überabzählbar*, wenn sie nicht endlich ist, und es keine Bijektion zwischen ihr und $\mathbb{N}$ gibt.

Um zeigen, dass $\mathbb{R}$ keine Bijektion mit $\mathbb{N}$ zulässt, reicht es zu zeigen, dass es keine Bijektion zwischen dem Intervall $(0, 1] \subset \mathbb{R}$ und $\mathbb{N}$ gibt.

Sei (r_i) irgend eine - möglicherweise unendliche - sich nicht wiederholende Folge von Zahlen aus $(0, 1]$. Falls die Folge (r_i) alle Zahlen aus $(0, 1]$ enthält, so erhielten wir eine Bijektion $i \longleftrightarrow r_i$. Um dies zu widerlegen, beweisen wir, dass es für jede Folge (r_i) eine Zahl $d \in (0, 1]$ gibt, die nicht in (r_i) vorkommt.

Betrachten wir eine Niederschrift der Folge (r_i).

$$
\begin{array}{lllllll}
r_1 & = & 0. & r_{11} & r_{12} & r_{13} & \dots \\
r_2 & = & 0. & r_{21} & r_{22} & r_{23} & \dots \\
r_3 & = & 0. & r_{31} & r_{32} & r_{33} & \dots \\
r_4 & = & 0. & r_{41} & r_{42} & r_{43} & \dots \\
r_5 & = & 0. & r_{51} & r_{52} & r_{53} & \dots \\
\vdots & & & & & &
\end{array}
$$

Die Ziffern $r_{ij} \in \{0, 1, \dots, 9\}$ sind hier die Dezimalstellen von r_i. Wir definieren die Dezimalstellen von $d = 0.d_1d_2d_3\dots$ als

$$d_i = \begin{cases} 4 & \text{falls } r_{ii} = 5 \\ 5 & \text{sonst} \end{cases}$$

Damit ist sichergestellt, dass $d \notin (r_i)$, denn d unterscheidet sich von einem beliebigen r_i in mindestens einer Stelle. Da die r_{ii} in der Niederschrift eine Diagonale bilden, nennt man d auch *Diagonalzahl*. q.e.d.

b) Es reicht, zu zeigen, dass wir die Menge aller Programme in eine eindeutige Reihenfolge bringen können. Sei p_i das i-te Programm. Dann gibt es offensichtlich die Bijektion $i \leftrightarrow p_i$.

Wir sortieren die Programme auf die folgende Weise.

- Nach der Anzahl ihrer Zeichen (inklusive Leerschläge). Ein kürzeres Programm kommt vor einem längeren.
- Wenn zwei Programme die selbe Länge haben, dann ergibt sich die Ordnung nach dem Alphabet, resp. durch eine beliebig festgelegte Ordnung aller möglichen Zeichen, die vorkommen können.

Bleibt noch zu erläutern, wie wir diese Programme generieren können. Wir gemäss der oben stehenden Reihenfolge einfach alle Texte. Für jeden Text lassen wir einen Compiler prüfen, ob es sich um ein gültiges Programm handelt. Falls nein, dann streichen wir es einfach aus der Aufzählung.

Punktvergabe.

a) Die entscheidende Aussage ist, dass wir für jede vermeintlich erschöpfende Liste von reellen Zahlen, die als Dezimalzahlen geschrieben wurden, eine Zahl d finden können, die nicht in der Liste enthalten ist. Dafür gibt es insgesamt **+7 Punkte**. Für eine konkrete Bildungsvorschrift von d und Bezugnahme auf die Diagonalelemente der Niederschrift gibt noch einmal **+3 Punkte**. Es ist nicht so wichtig, dass das Diagonalargument auf *alle* möglichen Folgen angewandt wird. Die aufsteigend sortierte Reihenfolge ist ausreichend, um die Idee zu zeigen.

Falls das Argument so geht: "Zu jeder *endlichen* Tabelle von Zahlen, kann man immer noch eine dazufügen" gibt es **+5 Punkte**.

Keine Punkte gibt es für Beweise, die falsch sind oder auch für die rationalen Zahlen funktionieren wuerden (z.B. zwischen zwei reellen Zahlen gibt es immer

noch eine, deshalb muss die Menge überabzählbar sein; Pi hat keine endliche Darstellung, also ist R überabzählbar,...). Ebenso gibt es keine Punkte, wenn zwar irgendwo eine Tabelle gemalt ist, aber klar wird, dass der Kandidat hier einfach nur ein Bild im Kopf hatte, ohne dessen Bedeutung zu kennen.

Subtilitäten sollen nicht gewichtet werden (zum Beispiel "Reduktion auf Intervall (0,1]", "nicht abbrechende Dezimalzahlen", oder keine eigentliche Widerspruchsannahme).

b) Keine Punkte gibt es, wenn die einzig sinnvolle Beobachtung die ist, dass das Alphabet endlich ist.

2-3 Punkte für falsche Beweise, die noch einen sinnvollen Kern haben, z.B. Abbildung der Zeichen auf Zahlen, Aneinanderreihung der Zahlen ergibt eindeutige Zahl für ein Programm. Probleme: mehrere Programme könnten die gleiche Zahl haben; wir haben immer noch keine Bijektion (nicht alle Zahlen kommen vor), keine Unterscheidung zwischen Texten und Programmen.

5 Punkte, falls die alphabetische Sortierung angegeben ist, aber das Kriterium "Länge" fehlt.

7-8 Punkte, falls das Kriterium Länge angegeben ist, dann aber nur gesagt wird, bei fester Länge gibt es endlich viele Texte, ohne zu sagen, wie sie sortiert sind.

8 Punkte, wenn der Aspekt der syntaktischen Korrektheit ignoriert wird (also alle Texte sortiert werden).

Aufgabe 3.
Die eleganteste Lösung ist vermutlich die rekursive.

```
// POST: Gibt auf dem Standardausgabestrom (std::cout) 2^n Sterne aus.
void print_stars(unsigned int n) {
  if (n == 0) {
    std::cout << "*";
    return;
  }
  print_stars(n-1);
  print_stars(n-1);
}
```

Man kann es auch iterativ lösen. Das heisst, zuerst 2^n ausrechnen und dann in einer `for`-Schleife 2^n einzelne Sterne ausgeben. Dies führt jedoch relativ schnell zu `integer`-Überlauf. Trotzdem wird diese Variante auch mit der vollen Punktezahl bewertet.

Für die Berechnung von 2^n gibt es mehrere Möglichkeiten. Alle werden mit der vollen Punktezahl bewertet, wenn sie korrekt ausgeführt sind, obwohl es bei der einen oder anderen Nachteile gibt.

- Berechnung mit einer eigenen Funktion, resp. mit einer `for`-Schleife.

- Berechnung mit dem shift-Operator (<<)
- Zuhilfenahme der Funktion `std::pow` aus der `cmath` Bibliothek. **Vorsicht**: Ein korrektes Resultat ist nicht garantiert.

Punktvergabe. Für eine korrekte Lösung gibt es **+10 Punkte**. Abzüge wegen Unzulänglichkeiten werden vom Korrektor festgelegt.

Wie bei allen Programmieraufgaben soll gelten, das 1-2 kleinere Syntaxfehler noch ohne Punkteabzug toleriert werden. Für jeden weiteren Fehler gibt es dann aber **-1 Punkt**.

Aufgabe 4.

Achtung: In der Aufgabenstellung gibt es einen Fehler. In Zeile 3 muss der Typ `double` lauten anstatt `int`. Wenn jemand dies gemerkt hat, vergeben wir zwei Zusatzpunkte. Wenn jemand aber argumentiert, aufgrund dieses Fehlers sei die Aufgabenstellung nicht wohldefiniert, so lassen wir dieses Argument nicht gelten, denn es ist trotzdem klar, was die Funktion macht.

a) Das Sortierverfahren nennt sich *Insertion-Sort*. Der Index der äusseren Schleife gibt an, bis wo das Feld `c` korrekt sortiert ist. Zu Beginn der Schleife gilt immer, dass der Bereich `c[0],...,c[i-1]` aufsteigend sortiert ist. Die innere Schleife sorgt dafür, dass der sortierte Bereich um eins wächst, indem sie das Element `c[i]` so lange nach links verschiebt, bis es an der richtigen Stelle steht. Dabei ist zu bemerken, dass das Element `c[i]` nicht stellenweise verschoben wird. Es wird Platz geschafft und dann an der richtigen Stelle eingefügt. Daher *Insertion*-Sort.

b) Die Anweisung in Zeile 8 wird $n-1$ Mal ausgeführt.

c) Die Antwort lautet
$$\sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} = \frac{n^2}{2} - \frac{n}{2}.$$
Dies ergibt sich aus den beiden Schleifen-Ausdrücken, selbst wenn man völlig ausser acht lässt, wie die Werte des Feldes `c` geartet sind. Eine kleine Subtilität ist die Anwendung der Kurzschlussauswertung. Wenn (`index > 0`) schon `false` ergibt, dann wird (`value < c[index-1]`) gar nicht mehr ausgewertet.

d) Für eine absteigende Folge werden tatsächlich $n^2/2-n/2$ der gefragten Vergleiche getätigt. Also zum Beispiel für `c = {n,n-1,...,1.0}`.

e) Ja. Zum Beispiel *Merge-Sort*. Dort ist der führende Term der Anzahl der Vergleiche $n \log n$.

Punktvergabe. Wie in der Prüfung angegeben geben die einzelnen Teilaufgaben **+8 / +3 / +8 / +3 / +3 Punkte**.

Aufgabe 5.

a)
- Die Variablen `h_`, `m_` und `s_` sind dazu da, die Anzahl der Stunden, Minuten und Sekunden eines bestimmten Zeitpunktes zu speichern.
- Die Funktion `tick()` ist dazu da, den gespeicherten Zeitpunkt um eine Sekunde in die Zukunft zu verschieben.
- Die Funktion `time(...)` ist dazu da, um die gespeicherte Zeit in den Referenzparametern `h`, `m` und `s` auszulesen.
- Der Konstruktor `Clock(...)` ist dazu da, eine Instanz der Klasse `Clock` zu generieren und mit den korrekten Werten zu initialisieren.

b)
- Der erste Fehler ist in Zeile 05. Ein Konstruktor kann nicht `const` sein.
- Der zweite Fehler ist in Zeile 10. Die Funktion `tick` hat keinen Rückgabewert. Die Zeile sollte `void tick()` lauten.
- Der dritte Fehler wird in Zeile 25 gemeldet. Es wird eine nicht-`const` Funktion aus einer `const` Funktion aufgerufen. Das geht nicht. Der Aufruf von `tick()` ergibt an dieser Stelle unabhängig davon auch wenig Sinn. Wir werten den Fehler aber nur als erkannt, wenn auf die `const`-Problematik hingewiesen wird.

c) Die Datenmitglider der Klasse werden über die sogenannte Initialisierungsliste initialisiert. Der Konstruktor tut sehr wohl etwas.

d)
```
// POST: Die Zeit, die in c gespeichert ist,
//       wurde im Format h:m:s auf o ausgegeben.
std::ostream& operator<< (std::ostream& o, const Clock& c){
  unsigned int h;
  unsigned int m;
  unsigned int s;
  c.time(h, m, s);
  o << h << ":";
  o << m << ":";
  o << s;
  return o;
}
```

Punktvergabe.

a) Für die richtige Beschreibung der sechs Mitglieder (`h_`, `m_`, und `s_` beschreiben den Wert einer Uhrzeit, `Clock` initialisiert diesselbe, `time` gibt sie aus und `tick` erhöht sie um eine Sekunde) jeweils **+1 Punkt**.

b) Für jeden richtig entdeckten und beschriebenen Fehler jeweils **+2 Punkte**. Das `const` in der Signatur von `time` ist kein Fehler, sondern der Aufruf von `tick`. Die Nennung gibt deshalb nur **+1 Punkt**.

c) Insgesamt **+4 Punkte**.

d) **+9 Punkte** nur, wenn alles korrekt ist, ansonsten **+8 Punkte**. Falscher oder fehlender Aufruf von `time` -**2 Punkte**. Falsche oder fehlende Definition und Verwendung lokaler Variablen für den Aufruf von `time` -**2 Punkte**. Falsche Verwendung der Ausgabeoperatoren oder Ausgabestroms -**2 Punkte**. Falscher oder fehlender Rückgabewert -**2 Punkte**.

Wie in der Prüfung angegeben geben die einzelnen Teilaufgaben **+6 / +6 / +4 / +9 Punkte**.

Aufgabe 6.

Es gibt natürlich viele Möglichkeiten, Teil a) zu lösen. Allen liegt aber die Idee zugrunde, die alternierende Summe mit einer Schleife auszurechnen. Eine "dumme" (aber korrekte) Lösung besteht darin, in jeder Iteration die Potenz von x sowie die entsprechende Fakultät neu auszurechnen (z.B. mit Hilfsfunktionen- oder -schleifen). Die intelligente Lösung führt Potenz und Fakultät jeweils korrekt nach, wie folgt.

a)
```
// POST: Es wird eine Approximation von sin(x) berechnet, die die
//       ersten 20 Terme der Potenzreihe beruecksichtigt.
double sinus(double x) {
   double result = x;
   double temp = x;

   for (int i = 1; i < 20; ++i) {
      temp *= x / (2*i+1) * x / (2*i);
      if (i % 2 == 0) {
        result += temp;
      } else {
        result -= temp;
      }
   }

   return result;
}
```

b) Nein, vermutlich nicht. Das Problem liegt darin, dass die Approximationsformel für grosse Werte von x numerisch nicht stabil ist. Rundungsfehler können sich vervielfältigen und das Resultat kann extrem falsch sein. Eine einfache Behebung ergibt sich aus der Periodizität der Funktion $\sin(x)$. Da $\sin(x) = \sin(x - 2\pi)$ gilt, können wir den Eingabewert immer in den Bereich $[0, 2\pi]$ verschieben. In diesem Bereich zeigt sich relativ grosse numerische Stabilität, weil zu berechnenden Zahlen nicht ganz so gross werden.

   Lassen Sie uns noch bemerken, dass `double`-Überlauf bei 20 Termen noch kein besonders grosses Problem ist, da $10'000^{41} < 2^{1023}$. Dabei ist 41 der Exponent des letzten Termes und 1023 ein typischer maximaler Exponent von `double` Implementierungen. Wenn man allerdings noch grössere Zahlen eingibt, dann kann auch das irgendwann ein Problem sein. Deshalb werten wir diese Antwort auch.

Punktvergabe. Für die Teilaufgabe a) gibt es **+14 Punkte**. Für die Teilaufgabe b) gibt es **+6 Punkte**.

a) Folgende Abzüge wurden angewendet: **-2 Punkte** wenn in jeder Iteration die Potenzen und Fakultäten neu berechnet wurden. **-3 Punkte** wenn im Quelltext einfach x^y oder $y!$ geschrieben wurde. **-2 Punkte** wenn undefinierte Funktionen verwendet wurden. **-1 Punkte** pro syntaktische Fehlertyp. **-1 bis -2 Punkte** für ungenaue Schleifenzählung.

   Wenn die Lösung grundsätzlich schlecht ist, dann wird von 0 nach oben gezählt. **+2 Punkte** für eine vernünftige Schleife. **+2 Punkte** für die korrekte Berechnung der Potenzen und Fakultäten.

b) **+3 Punkte** für die richtigen Argumente, was die Genauigkeit der Funktion angeht. **+3 Punkte** für die Ausnützung der Periodizität der Sinus Funktion.

Ende der Musterlösung zur Prüfung vom 9.8.2010.