Prüfung Informatik D-MATH/D-PHYS 6.8.2008

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Allgemeines zur Punktevergabe. Kleinere Syntaxfehler (vergessenes Semikolon o.ä.) werden ohne Punktabzug toleriert. Ab dem zweiten jedoch wird je nach Schwere mindestens 1 Punkt abgezogen. Wenn das ganze nicht mehr als `C++` erkennbar ist, sondern nur eine Art Pseudocode bildet, gibt es 0 Punkte.

Aufgabe 1.

(a) Aufgrund der Links-Assoziativität des Divisionsoperators kann hier nur das äusserste Klammerpaar weggelassen werden, ohne die Auswertungssemantik zu ändern. Das heisst, die maximal reduzierte Klammerung lautet `99/(1.5/(3/2))`.

Die Auswertung ist wie folgt.

```
99/(1.5/(3/2))
99/(1.5/1)
99/1.5
66.0
```

(b) Das äusserste Klammerpaar kann wiederum weggelassen werden. Ebenso die beiden Klammerpaare, die das unäre Minus und Plus an ihren Operanden binden. Zudem kann das Klammerpaar, welches die linke Seite des Gleichheitstests umschliesst, wegfallen. Die maximal reduzierte Klammerung lautet somit `1-(-5>+3)==1`.

Die Auswertung ist wie folgt.

```
1-(-5>+3)==1
1-(false)==1
1-0==1
1==1
true
```

Punktvergabe. 10 Punkte für Teil a) und 10 Punkte für Teil b). 4 Punkte sollen jeweils für die korrekte Angabe der redundanten Klammerpaare und 6 für die korrekte Auswertung vergeben werden. Jeweils einen Punkt Abzug für Fehler, seien das Wert einer Auswertung, Typ einer Auswertung, oder zu viele/wenige Klammern weggelassen.

Aufgabe 2. Die Funktion `f` zählt die Stellen einer Zahl im Dezimalsystem. Die Funktion `g` gibt insgesamt `k` Zeichen auf den Ausgabestrom, wobei die Zahl `n` rechhtsbündig darin eingefügt ist. Die führenden Zeichen sind Leerschläge. Falls die Zahl `n` mehr als `k` Stellen hat, wird nichts ausgegeben.
Die Nachbedingungen könnten also wie folgt aussehen.

```
// POST: Returns the number of digits of n in decimal representation.
unsigned int f (unsigned int n);

// POST: Outputs the number n to the std output stream. In total, k
//       characters are output, the number n being right-aligned. The
//       leading characters are filled with blanks. If k is smaller
//       than the number of digits of n, then nothing is output.
void g (unsigned int n, unsigned in k);
```

Punktvergabe. 6 Punkte für die Nachbedingung von `f`, 9 Punkte für die Nachbedingung von `g`. Nachbedingung von `f`: 2 Punkte für "Rückgabe der Anzahl Stellen", 2 Punkte für "von n", 2 Punkte für den Verweis "n im Dezimalsystem", -1 Punkt für die Formulierung "gibt aus" anstatt "gibt zurck". Nachbedingung von `g`: 1 Punkt für Verwendung von `std::out`, 2 Punkte für die richtige Aussage bezüglich der Leerzeichen, 2 Punkte für die Ausgabe von n, 4 Punkte für die Aussage, dass keine Ausgabe erfolgt für $k <$`f(n)`. Es gibt einen Abzug von $1-4$ Punkten, für unklare Formulierungen.

Aufgabe 3.

(a) Das Problem tritt im ersten Durchlauf der äusseren Schleife auf. Die Laufvariable `j` hat irgendwann den Wert 99, was laut Abbruchbedingung der inneren Schleife erlaubt ist, und somit versucht das Programm in Zeile `05` auf `a[100]` zuzugreifen. Dies ist kein erlaubter Index. Deshalb erscheint die Fehlermeldung.

Bemerkung:
1. Man kann sogar sagen, dass Sie Glück hatten, diese Fehlermeldung zu erhalten, denn in `C++` überprüfen weder Kompiler noch Laufzeitsystem, ob Sie einen erlaubten Index referenzieren. Es kann sein, dass die unerlaubterweise referenzierte Speicherzelle manipuliert wird, ohne dass das Programm abstürtzt. Dies ist ein viel schlimmeres Verhalten, denn das Programm wird zwar scheinbar funktionieren, aber nicht das tun, was Sie beabsichtigt haben.

2. In der Aufgabenstellung gab es einen Fehler: Der Name `switch` ist ein reserviertes Schlüsselwort von `C++` und darf gar nicht als Variablenname verwendet werden. Dies war nicht der Fehler, nach dem wir gesucht haben, aber wäre dies von jemandem genannt worden, so hätte diese Person die volle Punktezahl erhalten.

Reparatur:
Die einfachsten Reparaturvorschläge sind

```
02: for(int i = 1; switch; ++i) {
```

oder

```
04:    for(int j = 0; j < 99 - i; ++j) {
```

(b) Nehmen wir einmal an, dass wir oben die erste Reparaturvariante gewählt haben. Als erstes stellen wir fest, dass die äussere Schleife höchstens 100 Mal ablaufen kann, denn dann ist `i` gleich 100 uns somit die Abbruchbedingung der inneren Schleife schon für `j` gleich 0 erfüllt. Die Variable `switch` hat keine Changce mehr, auf `true` gesetzt zu werden, und somit ist auch die Abbruchbedingung der ersten Schleife erfüllt.

Für `i` gleich 1 macht die innere Schleife nun 99 Durchläufe, für `i` gleich 2 sind es noch 98, und so weiter. Wenn wir bei `i` gleich 99 angelangt sind macht die innere Schleife genau einen Durchlauf, und für `i` gleich 100 keinen mehr. Die Bedingung in Zeile `05` wird also maximal

$$99 + 98 + \ldots + 1 = \frac{100 \cdot 99}{2} = 4950$$

Mal ausgeführt. Man kann sich auch leicht überlegen, dass dieses Kontingent unter Umständen tatsächlich ausgeschöpft wird. Wenn die Zahlen zu Beginn in aufsteigender Folge vorliegen, dann wird jedes Mal eine Vertauschung gemacht.

Diese Anzahl der maximal benötigten Vergleiche für Bubble Sort wächst übrigens quadratisch in der Anzahl der zu sortierenden Zahlen, das heisst $4950 \approx 100^2$.

Punktvergabe. 10 Punkte für Teil a) und 10 Punkte für Teil b). Bei a): 6 Punkte für die Beschreibung des Problems, und 4 Punkte für die Reparatur. Falls jemand das `switch` Problem als Fehlerquelle nennt, dann müssen wir kulant sein und alle 10 Punkte geben. -1 Punkt falls nicht angegeben wird, dass ein Zugriff auf `a[100]` versucht wird, -2 Punkte falls nicht angegeben wird, in welchem Schleifendurchlauf der Fehler passiert, -2 Punkte für falsche neue Obergrenze der `for`-Schleife. Bei b): Volle Punktezahl nur wenn jemand genau auf 4950 kommt, respektive die Formel $\binom{n}{2}$ für $n = 100$. Es soll argumentiert werden, weshalb die Variable `switch` irgendwann nicht mehr `true` sein kann und deshalb die äussere Schleife zu Ende geht. Fehlt dieses Argument, gibt es 3 Punkte Abzug. Aussagen wie "quadratisch in n" und ähnlich geben, sofern isoliert, maximal 3 Punkte, als zusätzliche Bemerkung 1 Pluspunkt. 2 Punkte für die isolierte Aussage, dass es 99 Vergleiche im ersten Durchlauf sind.

Aufgabe 4. Die folgende Lösung extrahiert alle (Prim-) Teiler, die kleiner sind als 100. Falls dabei am Ende die 1 übrigbleibt, handelt es sich bei der ursprünglichen Eingabe `x` um eine petty number, sonst nicht.

```
// POST: returns true if and only if no prime divisor of x
//       is larger than 99.
bool petty_number (unsigned int x) {
    for (int i = 2; i < 100; ++i) {
        while (x % i == 0) {
            x /= i;
        }
    }
    return (x == 1);
}
```

Eine andere Möglichkeit ist, das Problem rekursiv zu lösen.

```
// POST: returns true if and only if no prime divisor of x
//       is larger than 99.
bool petty_number (unsigned int x) {
    if (x == 1) return true;
    for (int i = 2; i < 100; ++i) {
        if (x % i == 0) {
            return petty_number(x / i);
        }
    }
    return false;
}
```

Punktvergabe. Unabhängig von der Effizienz der Lösung geben korrekte Implementationen 13 Punkte. Um die letzten 2 Punkte auch noch zu erhalten, sollte die Lösung einigermassen effizient sein, vergleiche Musterlösung.

Aufgabe 5. Der Hauptpunkt bei dieser Aufgabe ist, dass man eine Bijektion zwischen den ganzen Zahlen und der Menge der syntaktisch korrekten `C++` Programme herstellen muss. Wenn wir jedem Element der einen Menge eindeutig ein Element der anderen Menge zuordnen können (und umkgekehrt), dann sind die beiden Mengen gleich mächtig. Die einfachste Methode dies zu erreichen, ist es, anzugeben wie wir alle syntaktisch korrekten Programme deterministisch und der Reihe nach generieren können. Wir ordnen dann jedem Programm diejenige Zahl zu, die die Stelle bezeichnet, an welcher das Proramm in unserer Reihenfolge auftritt. Und das geht folgendermassen.
Erstens stellen wir fest, dass alle `C++` Programme über einem endlichen Alphabet verfasst sind. Zweitens können wir für dieses Alphabet eine lexikographische Ordnung festlegen, so dass wir alle Texte gleicher Länge in eine eindeutige Reihenfolge bringen können. Umgangssprachlich würden wir sagen, dass wir die Texte "alphabetisch sortieren". Die dritte Feststellung, die wir brauchen, ist nun, dass es eine endliche Anzahl von `C++` Texten von bestimmter Länge gibt.

Wir gehen nun wie folgt vor. Zuerst generieren wir in lexikographischer Reihenfolge **alle** Texte der Länge 1 über dem gegebenen Alphabet. Ein Kompiler hilft uns, die syntaktisch korrekten von den anderen zu unterscheiden. Bei der Länge 1 wird dies natürlich keiner sein. Doch wir fahren fort mit Texten der Länge 2, 3, und so weiter. Irgendwann wird uns der Kompiler melden, dass wir das erste syntaktisch korrekte `C++` Programm gefunden haben. Dies bezeichnen wir dann mit der Nummer 1. Und so weiter...
Gibt uns also jemand eine natürliche Zahl n, so können wir mit dem obigen Verfahren genau das n-te Programm generieren. Umgekehrt können wir die Ordnungszahl eines gegebenen Programmes ermitteln, indem wir bei dem obigen Verfahren jedes gültige Programm mit dem gegebenen Text vergleichen und die Zahl ausgeben, bei der die Übereinstimmung der beiden Texte festgestellt wird. Somit haben wir eine Bijektion zwischen den beiden Mengen erstellt.

Punktevergabe: Die wesentlichen Punkte der Argumentation sind folgende:

- Bijektion zwischen den beiden Mengen
- Endliches Alphabet für C++ Programme, also endliche Menge von Programmen einer bestimmten Länge
- Ordnung nach der Länge des Textes
- Ordnung lexikographisch
- Kompiler hilft uns `C++` Programme vom Kauderwelsch zu trennen

Jedes dieser Argumente ist etwa 4 Punkte wert. Lösungen die anders (und richtig) argumentieren sollen natürlich auch die volle Punktezahl bekommen können. Verweise auf die Vorlesung im Stile "Bei Hromkovic haben wir gesehen, dass wir alle `C++` Programme generieren können" geben keine Punkte.

Aufgabe 6.

```
class Matrix10 {

    private:

    // Subtask a)
    int values[10][10];


    public:

    // Constructor for b)
    Matrix10 () {
        for (int i = 0; i < 10; ++i) {
```

```
            for(int j = 0; j < 10; ++j) {
                values[i][j] = 0;
            }
        }
    }


    // Task c)
    // Getter method for entries
    // PRE: i < 10, j < 10
    // POST: Returns the value of the matrix at row i, column j.
    int getEntry(unsigned int i, unsigned int j) {
        return values[i][j];
    }

    // Setter method for entries
    // PRE: i < 10, j < 10
    // POST: The value of the matrix at row i, column j, is set
    //       to n.
    void setEntry(unsigned int i, unsigned int j, int n) {
        values[i][j] = n;
    }

    // Task d)
    // Tests whether the matrix is symmetric
    // POST: Returns true if and only if the matrix is symmetric, i.e.
    //       if the entry at row i, column j, is equal to the entry at
    //       row j, column i, for all choices of i and j.
    bool symmetric() {
        for (int i = 0; i < 10; ++i) {
            for (int j = i; j < 10; ++j) {
                if (values[i][j] != values[j][i]) {
                    return false;
                }
            }
        }
        return true;
    }

}
```

Punktvergabe. Jede der vier Teilaufgaben a), b), c) und d) erhalten 7 Punkte. Die 2 zusätzlichen Punkte erhält nur, wer ein sinnvolles `private` und `public` Konzept hin-

geschrieben hat. Das heisst die Daten der Matrix sollten `private` sein und der Rest `public`. Die Funktionen sollen korrekt in die Klasse eingebunden sein. Das heisst entweder im Klassenrumpf stehen, oder, falls ausserhalb, mit einem entprechenden `Matrix10::` and der richtigen Stelle versehen sein.