Prüfung — Informatik D-MATH/D-PHYS
12. 02. 2010 09:00–11:00

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Kandidat/in:

Name: .....

Vorname: .....

Stud.-Nr.: .....

Ich bezeuge mit meiner Unterschrift, dass ich die Prüfung unter regulären Bedingungen ablegen konnte und dass ich die allgemeinen Bemerkungen gelesen und verstanden habe.

Unterschrift: .....

Allgemeine Bemerkungen und Hinweise:

1. Überprüfen Sie die Vollständigkeit der ausgeteilten Prüfungsunterlagen (drei Blätter, bestehend aus 1 Deckseite, 4 Aufgabenseiten mit insgesamt 6 Aufgaben und 1 leere Seite)!
2. Falls Sie während der Prüfung durch irgendeine Behinderung oder Störung beeinträchtigt werden, melden Sie dies bitte sofort der Aufsichtsperson! Spätere Klagen können nicht akzeptiert werden.
3. Erlaubte Hilfsmittel: **keine. Einzige Ausnahme sind Wörterbücher**.
4. Betrugsversuche führen zu sofortigem Ausschluss und können rechtliche Folgen haben.
5. Pro Aufgabe ist höchstens eine gültige Version eines Lösungsversuches zulässig. Streichen Sie ungültige Lösungsversuche klar durch! Schreiben Sie auf separate Blätter, nicht auf die Aufgabenblätter!
6. Sie dürfen die Aufgaben in beliebiger Reihenfolge lösen. Konzentrieren Sie sich jeweils auf eine Aufgabe, aber teilen Sie sich Ihre Zeit ein!
7. Nach Ablauf der Prüfungszeit oder wenn Sie frühzeitig abgeben möchten, übergeben Sie Ihre Lösungen bitte einer Aufsichtsperson und verlassen zügig den Raum. **Jedes Ihrer Lösungsblätter muss mit Ihrem Namen beschriftet sein. Die Prüfungsblätter sind mit abzugeben!**
8. Die Prüfung ist bestanden, wenn Sie 60 von 120 erreichbaren Punkten erzielen.

1	2	3	4	5	6	$\sum$

Aufgabe 1. (21 Punkte) Geben Sie zu den folgenden Ausdrücken jeweils den Typ und den Wert an. Es gibt keine Punkte für die Teilschritte der Auswertung. Sie brauchen diese nicht hinzuschreiben.

Zu Beginn jeder der Teilaufgaben gilt: Die Variable `x` ist vom Typ `int` und hat den Wert 2.

a) `3 * 1 + 1.5 / 3`

b) `11 * 19 % 21 * 13 % 23 * 3 % 3`

c) `7 / x * 3.0 * x`

d) `x != 2 && x - 1 != 1 || x + 1 == 3`

e) `17 / 2 == 8.5 && 7 * 3 == 21.0`

f) `(++x - 1) / 2`

g) `x++ - 1 / 2`

Aufgabe 2. (4 / 4 / 10 Punkte) Betrachten wir das normalisierte Fliesskommazahlensystem $\mathcal{F}^*(2,2,-2,2)$. Das heisst eine Zahl kann in diesem System dargestellt werden, falls sie wie folgt geschrieben werden kann:

$$\pm(1 \cdot 2^0 + d_1 \cdot 2^{-1})\, 2^e,$$

wobei $e \in \mathbb{N}$, $-2 \leq e \leq 2$ und $d_1 \in \{0,1\}$. Beachten Sie, dass wegen der Normalisierung $d_0 = 1$.

Falls eine Zahl oder das exakte Resultat einer Berechnung in dem System nicht dargestellt werden kann, so wird zur nächsten Zahl gerundet, die dargestellt werden kann. In dieser Aufgabe gibt es immer eine eindeutige nächste Zahl.

a) Wie lautet die Binärexpansion der dezimalen Zahl 0.8?

b) Wie lautet die Darstellung von 0.8 im System $\mathcal{F}^*(2,2,-2,2)$?

c) Wie lautet das Resultat von $0.8 + 0.8 + 0.8 + 0.8 + 0.8$ wenn Sie diese Rechnung im System $\mathcal{F}^*(2,2,-2,2)$ durchführen? Stimmt es mit dem richtigen Resultat 4 überein? **Bemerkung:** Aus Teilaufgabe b) kennen Sie die Darstellung von 0.8 im System $\mathcal{F}^*(2,2,-2,2)$. Addieren Sie nun zuerst zwei dieser Zahlen zusammen und runden dann korrekt. Dann addieren Sie die nächste Zahl dazu und runden wieder korrekt, und so weiter. Nach der vierten Addition erhalten Sie das Schlussresultat.

Aufgabe 3. (15 Punkte) Implementieren Sie die Funktion `is_cube`, die testet, ob eine natürliche Zahl kubisch ist. Eine natürliche Zahl n ist kubisch, falls es eine andere natürliche Zahl a gibt, so dass $n = a^3$. Wir zählen hier übrigens 0 zu den natürlichen Zahlen. Die Funktion `is_cube` muss auch für die Eingabe 0 funktionieren.

```
// 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);
```

Aufgabe 4. (15 Punkte) Helfen Sie dem Kranführer! An einem Containerumschlagplatz gibt es drei Stellplätze mit den Nummern 1, 2 und 3. Auf dem ersten Platz stehen n Container aufeinander gestapelt. Die anderen beiden Stellplätze sind leer. Die Container sind von unterschiedlichem Gewicht und danach geordnet (der schwerste zuunterst). Sie sollen alle auf den Stellplatz 3 verschoben werden. Der Kran kann jedoch nur einen Container aufs Mal bewegen, und es darf nie ein schwererer Container auf einem leichteren stehen. Natürlich kann der Kran auch immer nur den obersten Container fassen und ihn nur zuoberst auf einem Stellplatz ablegen.

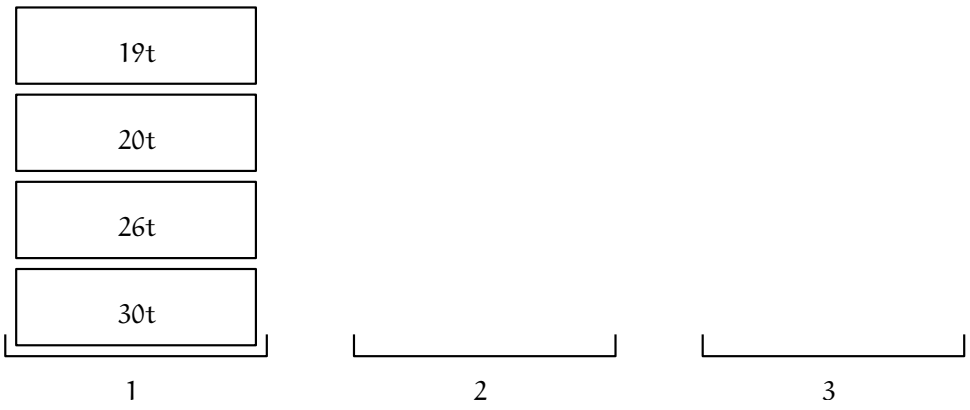

Abbildung 1: Ausgangssituation für $n = 4$ mit den Stellplätzen 1, 2 und 3. Die konkreten Gewichtsangaben dienen nur der Illustration und sind nicht relevant.

Implementieren Sie die Funktion

```
// 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(unsigned int n, unsigned int from, unsigned int to);
```

die eine Reihe von Anweisungen ausgibt, wie der Kranfahrer die Container bewegen soll, um den ganzen Stapel von Stellplatz 1 auf Stellplatz 3 zu verschieben. Schreiben Sie z.B. `move(2,1)` nach `std::cout` falls der oberste Container von Stellplatz 2 auf Stellplatz 1 gehoben werden soll.

Aufgabe 5. (7 / 8 / 8 Punkte) In dieser Aufgabe geht es darum, für ein Computerspiel die Anzeige von geometrischen Objekten im dreidimensionalen Raum zu programmieren. Gegeben ist eine Struktur `point`, die es Ihnen ermöglicht Punkte darzustellen.

```
struct point {
  double x, y, z;
}
```

a) Implementieren Sie die folgende Funktion, die den Abstand eines Punktes zum Ursprung berechnet.

```
// POST: Gibt den Abstand von p zum Ursprung an.
double distance(const point& p);
```

Tipp: Die Funktion `std::sqrt(double d)` berechnet die Wurzel einer Zahl `d`.

b) Schlagen sie einen `struct` mit dem Namen `gerade` vor, die verwendet werden kann, um Geraden im Raum darzustellen. Eine bestimmte Gerade muss dabei nicht notwendigerweise eindeutig dargestellt werden können, aber Sie sollten umgekehrt darauf achten, dass jedes Objekt des Typs `gerade` eine eindeutige Gerade im Raum bestimmt. Definieren Sie zu diesem Zweck nötigenfalls eine geeignete Invariante (`// INV:...`), die beim Benutzen des `structs` eingehalten werden muss.

c) Basierend auf Ihrem Vorschlag, implementieren Sie die folgende Funktion, die die Gerade durch zwei Punkte berechnet. Achten Sie dabei darauf, dass die Invariante, die Sie in Teilaufgabe b) definiert haben, eingehalten wird. Dies können Sie entweder mit einer geeigneten Vorbedingung (`// PRE:...`) oder durch entsprechende Tests innerhalb der Funktion erreichen.

```
// POST: Gibt eine gerade zurueck, die durch die Punkte a und b geht.
gerade berechne_gerade(const point& a, const point& b);
```

Aufgabe 6. (28 Punkte) Hier geht es darum, abzuschätzen wie lange ein effizienter Algorithmus zur Lösung gewisser Aufgaben benötigt. Als Ausgangslage bekommen Sie ein Feld mit n unsortierten `unsigned int` Zahlen und sollen die aufgeführten **Aufgaben in eine der folgenden Klassen einteilen**: Klasse A mit konstanter Anzahl von Operationen, Klasse B mit n, Klasse C mit $n \log n$ und Klasse D mit n^2 Operationen. Bei dieser Betrachtung können Sie konstante Faktoren vernachlässigen, d.h. wenn ein Algorithmus $3n^2$ Schritte braucht, dann gehört dieser in die Klasse D, ungeachtet des konstanten Faktors 3. Sie können annehnehmen, dass alle Standardoperationen in konstanter Zeit möglich sind. Konkret beinhaltet das die folgenden Operationen: Zugriff auf eine Zahl im Feld; Ausgeben einer Zahl; Das Addieren, Subtrahieren, Multiplizieren, Dividieren und Vergleichen zweier Zahlen; der Modulo-Operator angewandt auf zwei Zahlen.

Klasse	Anzahl Operationen	Beispiel
A	konstant	zwei Zahlen addieren
B	n	alle Zahlen ausgeben
C	$n \log n$	die Zahlen sortieren
D	n^2	jede Zahl n Mal ausgeben

Geben Sie in Worten eine **kurze Beschreibung Ihres Algorithmus** an. Z.B. "Durchlaufe das Feld und überprüfe bei jeder Zahl, ob sie gerade ist. Falls ja, dann gib die Zahl aus."

Hier noch ein umfassendes Beispiel: Sie sollen die kleinste Summe zweier Zahlen in dem Feld berechnen. Ein möglicher Algorithmus wäre folgender: "Sortiere die Zahlen und addiere die kleinsten beiden." Wir wissen, dass das Sortieren in die Klasse C fällt, und das Addieren der beiden Zahlen ist in konstanter Zeit möglich. Demnach würde dieser Algorithmus in die Klasse C fallen, da das Sortieren der aufwändigere Teil ist. Dies ist jedoch nicht optimal. Ein besserer Algorithmus ist: "Gehe durch das Feld hindurch und merke dir die beiden kleinsten Zahlen, die du bis und mit der aktuellen Stelle angetroffen hast. Danach, addiere die gefundenen Zahlen." Auf diese Weise gelangen wir zur Lösung, ohne das Feld vollständig zu sortieren. Wir durchlaufen das Feld ein Mal und machen jeweils maximal 2 Vergleiche. Somit fällt das Problem in die Klasse B. Die zweite Antwort ist **effizienter und erhält deshalb mehr Punkte**.

Bemerkung: Wenn wir hier von der "Differenz zweier Zahlen" sprechen, meinen wir immer den Absolutwert, d.h. immer die grössere Zahl minus die kleinere.

a) Gibt es mehr gerade oder ungerade Zahlen in dem Feld?

b) Gib die ersten 10 Zahlen des Feldes aus (falls $n \geq 10$).

c) Berechne die kleinste Differenz zweier Zahlen im Feld.

d) Berechne die grösste Differenz zweier Zahlen im Feld.

e) Gib für jede Zahl im Feld aus, welche anderen Zahlen in dem Feld Teiler sind.

f) Berechne die Summe aller Zahlen des Feldes.

g) Berechne die Summe der $\lceil n/2 \rceil$ kleinsten Zahlen im Feld.