Prüfung Informatik D-MATH/D-PHYS 4.8.2009

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Aufgabe 1.

a) (i) Die Antwort ist **Nein**. In der Vorlesung wurde gezeigt, dass es keinen Algorithmus gibt, der das Halteproblem entscheidet.

(ii) Das ist durchaus so; die Antwort lautet **Ja**. Vielleicht haben sie schon einmal von CISC und RISC Architekturen gehört, wobei dies für **C**omplex resp. **R**educed **I**nstruction **S**et **C**omputer steht.

(iii) Das ist falsch. Die Antwort lautet **Nein**. Sowohl die Mantisse als auch der Exponent der Fliesskommazahl haben eine endliche Darstellung. Das heisst, die beiden können nicht beliebig klein sein.

(iv) Die Antwort lautet **Nein**. Die Programmiersprache `C++` ist eben keine Registermaschinen-Sprache, sondern eine Programmiersprache höherer Ordnung. Die Motivation für solche Sprachen ist eben gerade, dass ein Algorithmus in einer allgemeinen Weise formuliert werden kann, die nicht von der tatsächlichen Rechnerarchitektur abhängt.

(v) Die Antwort lautet **Nein**. Ein Programm muss überhaupt nichts sinnvolles tun. Insbesondere kann es auch sein, dass ein Programm einen Algorithmus umsetzen sollte, es aber für bestimmte Eingaben das falsche Resultat liefert.

b) Seien a und b die beiden Eingabewerte, die in `Register(1)` resp. `Register(2)` eingelesen werden. Das Programm berechnet die Potenz a^b und gibt das Resultat in Zeile 9 aus.

Es gibt jedoch zwei Spezialfälle zu betrachten. 1. Wenn b eine negative Zahl ist, dann läuft das Programm unendlich lange und es wird nie etwas ausgegeben. 2. Wenn b keine ganze Zahl ist, dann läuft das Programm ebenfalls unendlich lange. Dies liegt daran, dass bei dem Vergleich in Zeile 5 nie Gleichheit erreicht wird. Man beachte dabei, dass in unserem Registermaschinen-Modell beliebige reelle Eingaben erlaubt sind, und dass die Berechnungen exakt sind.

Für alle anderen Eingaben (insbesondere $a = 0$ und/oder $b = 0$) funktioniert das Programm korrekt.

Punktvergabe.

a) Für jede korrekt beantwortete Frage gibt es **+1 Punkt**. Es gibt keinen Abzug für falsch angekreuzte Antworten.

b) **+4 Punkte** für die Aussage, dass "a^b berechnet und in Zeile 8 ausgegeben wird". Für die Aussage "Falls $b \notin \mathbb{N} \cup \{0\}$, dann läuft das Programm unendlich lange" gibt es **+6 Punkte**, und einen entsprechenden Abzug für leicht abweichende Charakterisierungen. Wenn ausgesagt wird, dass das Programm für $b \in \mathbb{N} \cup \{0\}$ funktionert, nicht aber, dass es andernfalls unendlich lange läuft, so gibt das nur **+5 Punkte** und auch entsprechende Abzüge für Ungenauigkeiten (falls z.B. die 0 vergessen wird).

Falls der Eingabebereich von a in irgendeiner Weise eingeschränkt wird, dann gibt das **-3 Punkte**.

Aufgabe 2.

Aufgabe	Typ	Wert	L-/R-Wert
a)	`bool`	`true`	R-Wert
b)	`int`	`5`	R-Wert
c)	`int`	`0`	L-Wert
d)	`double`	`0.5`	R-Wert
e)	`int`	`2`	R-Wert
f)	`int`	`2`	L-Wert

Punktvergabe. Für jede der 6 Teilaufgaben gibt es drei Punkte. Jeweils **+1 Punkt** für den korrekten Typ, **+1 Punkt** für den korrekten Wert und **+1 Punkt** für die korrekte Angabe L-/R-Wert. Das macht 18 Punkte insgesamt.

Aufgabe 3. Die Implementierung einer rekursiven Funktion drängt sich auf. Als Datentyp wollen wir `unsigned int` verwenden.

```
// PRE:  Alle Werte erlaubt
// POST: Der Aufruf binom(n,k) berechnet
//       den Binomialkoeffizienten "n tief k"
unsigned int binom(unsigned int n, unsigned int k) {
    if (n < k) return 0;
    if (k == 0 || n == k) return 1;
    return n * binom(n-1, k-1) / k;
}
```

Man beachte, dass die Reihenfolge der Auswertung in der Rückgabeanweisung wichtig ist. Teil man nämlich zuerst n durch k dann kommt nicht in jedem Fall eine ganze Zahl heraus. Das heisst, die ganzzahlige Division würde das Resultat verfälschen.

Punktevergabe Es gibt **+5 Punkte** für die geeignete Wahl der Datentypen und eine wohlgeformte Funktionssignatur. Es kann durchaus auch `int` verwendet werden, aber dann muss in der Vor- und Nachbedingung klargestellt werden, welche Werte erlaubt sind, d.h. die negativen sollen ausgeschlossen werden. Es gibt **+3 Punkte** für die korrekte Abhandlung des Falles $n < k$. Es gibt **+3 Punkte** für die korrekte Abhandlung des Falles $k = 0$ oder $n = k$. Es gibt **+6 Punkte** für einen richtigen rekursiven Aufruf. Es gibt **+5** Punkte für die richtige Reihenfolge bei der Auswertung des Rückgabewertes.

Für syntaktische Fehler gibt es **-1 Punkt** pro Fehler, wobei der erste Fehler ohne Abzug durchgehen kann. Ebenso soll für wiederholte Fehler nicht mehrfach abgezogen werden.

Aufgabe 4.

```
// PRE: n > 3
// POST: Es wird die Ulam Spirale als Gitterpunktbild auf dem Bildschirm
//       ausgegeben. Dabei werden die ersten n natuerlichen Zahlen
//       beruecksichtigt.
void ulam_spirale(unsigned int n) {

    int indexX = 1; // Startindex
    int indexY = 0; // Startindex

    // Setze einen Gitterpunkt fuer die Zahlen 2 und 3.
    print(indexX, indexY);
    print(indexX, ++indexY);

    // Die aktuelle Zahl, die geschrieben werden soll.
    unsigned int counter = 4;
    // Die Laenge der Seite, die wir abschreiten wollen.
    unsigned int sidelength = 2;

    while (counter <= n) {
        // gehe links
        for (int i = 0; i < sidelength; ++i) {
            if (counter > n) break;
            --indexX;
            if (prim(counter))
                print(indexX, indexY);
            ++counter;
        }
        // gehe runter
        for (int i = 0; i < sidelength; ++i) {
            if (counter > n) break;
            --indexY;
            if (prim(counter))
```

```
                print(indexX, indexY);
            ++counter;
        }
        ++sidelength;
        // gehe rechts
        for (int i = 0; i < sidelength; ++i) {
            if (counter > n) break;
            ++indexX;
            if (prim(counter))
                print(indexX, indexY);
            ++counter;
        }
        // gehe hoch
        for (int i = 0; i < sidelength; ++i) {
            if (counter > n) break;
            ++indexY;
            if (prim(counter))
                print(indexX, indexY);
            ++counter;
        }
        // erhoehe Schrittlaenge
        ++sidelength;
    }

}
```

Punktvergabe. Das genaue Schema ist vom Korrektor zu bestimmen. Es soll gelten, dass ein korrektes Resultat die volle Punktezahl erhält, egal wie umständlich die Implementierung ist. Als grobe Einteilung können die folgenden Anhaltspunkte gelten: Bis zu **+10 Punkte** für korrekte Syntax und korrekte Verwendung der beiden Funktionen `prim` und `print`. Bei einem Programm, dass offensichtlich nicht versucht, die Aufgabe zu lösen, werden diese Punkte nicht vergeben (d.h. es reicht nicht eine leere Funktion hinzuschreiben um diese 10 Punkte zu erhalten). Ein Syntaxfehler kann ohne Abzug verziehen werden, jeder weitere gibt **-1 Punkt**, wobei wiederholte Fehler nicht mehrfach geahndet werden. Für die eigentlich Logik des Programmes vergeben wir **+20** Punkte.

Es soll keinen Abzug geben für Programme, die unter Umständen mehr als n Werte ausgeben, sollten diese Anzahl nicht mehr als die nächstgrössere Quadratzahl sein. Dies entspricht dem Umstand, dass eine Windung der Spirale fertig gezeichnet wird, obwohl der verlangte Wert schon worden ist. Wird jedoch mehr gezeichnet, dann soll dies als "milder" semantischer Fehler betrachtet werden, und ein entsprechender Abzug angesetzt werden.

Aufgabe 5.

a)
```
// PRE: [b, e) und [o, o+(e-b)) sind zwei gueltige
//      und disjunkte Bereiche.
// POST: der Bereich [b,e) wird in umgekehrter Reihenfolge
//       in den Bereich [o, o+(e-b)) kopiert
void f (int* b, int* e, int* o)
{
  while (b != e) *(o++) = *(--e);
}
```

Hier war nur die Nachbedingung zu ergänzen.

b) Der erste Aufruf `f(a, a+5, a+5)` ist nicht gültig, weil der Bereich `[a+5, a+10)` nicht gültig ist. Der zweite Aufruf `f(a, a+2, a+3)` ist in Ordnung. Der dritte Aufruf `f(a, a+3, a+2)` ist nicht gültig, weil die beiden Bereiche `[a, a+3)` und `[a+2, a+5)` nicht disjunkt sind.

Punktevergabe.

a) Für die Aussage "Es wird vom einen Bereich in den anderen kopiert" gibt es **+6 Punkte**. Für die Aussage "Das Kopieren geschieht in umgekehrter Reihenfolge" gibt es **+8 Punkte**.

b) Für die richtige Einordnung eines Aufrufes gibt es **+2 Punkte**. Sollte beim ersten und dritten Aufruf die Begründung fehlen, gibt es **-1 Punkt**.

Aufgabe 6. Herr Plestudent irrt. Gäbe es nämlich eine solche Darstellung, so könnte man daraus auch eine endliche Binärdarstellung konstruieren, was bekanntermassen nicht geht. Um eine solche Binärdarstellung zu erhalten, schreiben wir $d_i \in \{0, \dots, 15\}$ als vierstellige Binärzahl:

$$d_i = d_{i_3}2^3 + d_{i_2}2^2 + d_{i_1}2^1 + d_{i_0}2^0, \quad d_{i_3}, d_{i_2}, d_{i_1}, d_{i_0} \in \{0, 1\}.$$

Wir erhalten dann

$$\sum_{i=1}^{p} d_i 16^{-i} = \sum_{i=1}^{p} d_i 2^{-4i} = \sum_{i=1}^{p} (\sum_{j=0}^{3} d_{i_j} 2^j) 2^{-4i} = \sum_{i=1}^{p} \sum_{j=0}^{3} d_{i_j} 2^{j-4i},$$

und dies ist eine endliche binäre Fliesskommazahl.

Weniger formal argumentieren wir wie folgt: Um aus einer p-stelligen hexadezimalen Fliesskommazahl

$$0.d_1 \cdots d_p$$

eine binäre Fliesskommazahl zu erhalten, spalten wir einfach jede Hexadezimalziffer d_i in vier Binärziffern $d_{i_3}, d_{i_2}, d_{i_1}, d_{i_0}$ auf und erhalten die $4p$-stellige äquivalente Binärzahl:

$$0.d_1 d_2 \cdots d_p = 0.d_{1_3} d_{1_2} d_{1_1} d_{1_0} \cdots d_{p_3} d_{p_2} d_{p_1} d_{p_0}.$$

Alternativ kann auch der folgende Algorithmus zur Umwandlung einer Nachkommadezimalzahl in eine Nachkommahexadezimalzahl angegeben werden. Wandelt man mit dessen Hilfe 0.1 (dezimal) um, so ergibt sich $0.1\overline{9}$ (hexadezimal). Der Algorithmus funktioniert analog zu dem Algorithmus zur Umwandlung von Nachkommabinärzahlen, den wir in der Vorlesung gesehen haben. Sei d die Zahl die wir umwandeln wollen (in der Aufgabenstellung $d = 0.1$), und h die hexadezimalzahl, die enstehen soll.

1. $i = 0$;
2. $i = i + 1$;
   $d = 16 * d$;
3. Als i-te Nachkommastelle von h setzen wir den ganzzahligen Teil von d (dies ist höchstens 15).
4. Falls der Nachkommateil von d gleich Null ist, beenden wir. Andernfalls gehen wir zurück zu Ziffer 2.

Mit diesem einfachen Algorithmus findet man $h = 0.1\overline{9}$.

Als weitere Mglichkeit lassen wir folgendes gelten: Nimm an, es gäbe in endliches p. Die Gleichung

$$\frac{1}{10} = \sum_{i=1}^{p} d_i 16^{-i}$$

kann man schreiben als

$$16^p = 10 \sum_{i=1}^{p} d_i 16^{p-i}.$$

Beide Seiten sind nun eine ganze Zahl, aber die linke Seite ist nicht durch 5 teilbar. Das ist ein Widerspruch. Deshalb, kann p nicht endlich sein.

Punktevergabe. Wenn eine der obigen Varianten (oder etwas gleichwertiges) beschrieben wird, dann gibt es **+8 bis +15 Punkte**, je nach dem, wie genau die Argumentation ist, und wie viele "kleine" Fehler, dass die Beschreibung enthält. Insbesondere gibt es sogleich die volle Punktezahl, wenn einfach $(0.1)_{10} = (0.1\overline{9})_{16}$ angegeben wird.

Sollte argumentiert werden, dass Herr Plestudent irrt, weil in der Primfaktorenzerlegung von 10 eine 5 vorkommt und dies kein Teiler von 16 ist, so stimmt das zwar, kann aber ohne Beweis nicht die volle Punktezahl erhalten (das gibt dann höchstens **+10 Punkte**).

Sollte nur die Aussage $16^{-i} = 2^{-4i}$ oder $16 = 2^4$ oder "aus 1 mach 4 Stellen" dastehen, so gibt das **+5 bis +10 Punkte**, je nachdem, was da sonst noch erwähnt wird, und was davon gegebenfalls falsch ist. Ein Spezialfall, der häufig auftritt ist folgender: Es wird "$16^{-i} = 2^{-4i}$" erwähnt und das sei dann eine Fliesskommazahl zur Basis 2. Es wird jedoch völlig vernachlässigt, was mit den Koeffizienten geschehen soll. Das gibt **+7 Punkte**.

Für die richtige Aussage bezüglich Herr Plestudent gibt es **+2 bis +5 Punkte**, sollte die Begründung fehlen, oder völlig falsch sein.

In manchen Prüfungen wird das Argument vorgebracht, dass die Potenzen von 16 immer eine 6 als letzte Ziffer haben und dass das zur Folge hat, dass im Nenner niemals das 10-fache des Zählers bekommen kann, oder so ähnlich. Das Argument kann durchaus stringent gemacht werden, aber das ist in den meisten Fällen nicht geschehen. Für eine solche unvollständige Begründung in dieser Art gibt es **+5 Punkte**.