Prüfung Informatik D-MATH/D-PHYS 23.1.2008

Dr. Bernd Gärtner, Prof. Juraj Hromkovič

Lösung.

Aufgabe 1.

a)

```
1 + 7 / 2
1 + 3
4
```

b)

```
x == 1 || 1 / (x - 1) < 1
1 == 1 || 1 / (x - 1) < 1
true || 1 / (x - 1) < 1
true
```

c)

```
!(1 && x) + 1
!(true && x) + 1
!(true && 1) + 1
!(true && true) + 1
!true + 1
false + 1
0 + 1
1
```

Punktvergabe. 6 Punkte für jede der Teilaufgaben. Minus 1 Punkt für jeden fehlenden Zwischenschritt. Minus 3 für falsches Endresultat. Wenn der Zwischenschritt nur aus der Auswertung der Variablen besteht, dann kann er weggelassen werden. Wenn es jedoch um eine Konversion `bool`→`int` oder umgekehrt geht, dann gibt es einen Abzug (siehe Teilaufgabe c)). Fehlende short-circuit Evaluation in Teilaufgabe b) wird mit minus 2 bestraft.

Aufgabe 2.

a) Das Problem tritt bei der Eingabe von 0 auf. Im ersten rekursiven Aufruf schon wird `haesslich(0)` aufgerufen, also treten wir in eine endlose Rekursion ein.

Eine mögliche Lösung ist, die 0 auch als hässlich zu definieren:

```
bool haesslich (unsigned int n)
{
  return
    (n == 0) || (n == 1) ||
    (n % 2 == 0) && haesslich (n / 2) ||
    (n % 3 == 0) && haesslich (n / 3) ||
    (n % 5 == 0) && haesslich (n / 5);
}
```

b) Im wesentlichen ist eine Zahl hässlich, wenn ihre Primfaktorzerlegung nur 2-en, 3-en und 5-en enthält. Zusätzllich ist auch die 0 und die 1 hässlich. Die folgende Postcondition beschreibt dies.

```
// POST: true g.d.w. die Eingabe entweder 0 oder 1 ist, oder eine
//       Zahl,deren Primfaktorzerlegung nur aus 2-en, 3-en und
//       5-en besteht.
```

Punktvergabe. Jeweils 10 Punkte für die beiden Teilaugaben a) und b).

a) 5 Punkte für den Fehler, das heisst "endless Rekursion". 5 Punkte für die Lösung. Punkteabzüge für falsche Fehler (z.B. die Aussage, es gäbe zuweilen kein return statement) und fehlerhafte Korrekturen (z.B. den Fall `n == 1` nicht mehr drin).

b) 7 Pukte für die Aussage "Zahlen, deren Primfaktorzerlegung nur aus 2-en, 3-en und 5-en" besteht. Maximal 3 Punkte für Aussagen wie "Zahlen, die Potenzen von 2, 3 oder 5 sind", oder "Zahlen, deren Primfaktorzerlegung eine Untermenge von {2, 3, 5} ist". Zusätzlich gibt es 3 Punkte für die richtige Aussage bezüglich der 0 und der 1.

Aufgabe 3. Eine Möglichkeit ist, die Funktion `gcd` aus der Vorlesung zu verwenden.

```
// POST: Der Rueckgabewert ist der groesste gemeinsame Teiler
//       von a und b.
unsigned int gcd(unsigned int a, unsigned int b) {
  if (b == 0) return a;
  return gcd(b, a % b); // b != 0
```

```
}

// PRE: a > 0, b > 0
// POST: Der Rueckgabewert ist true g.d.w. a und b teilerfremd sind.
bool teilerfremd(unsigned int a, unsigned int b) {
  return gcd(a, b) == 1;
}
```

Eine andere Möglichkeit ist natürlich, einfach alle Teiler bis zur kleineren der beiden Zahlen auszuprobieren.

```
// PRE: a > 0, b > 0
// POST: Der Rueckgabewert ist true g.d.w. a und b teilerfremd sind.
bool teilerfremd(unsigned int a, unsigned int b) {
  int smaller;
  if (a <= b)
    smaller = a;
  else
    smaller = b;
  for (unsigned int i = 2; i <= smaller; ++i) {
    if (a % i == 0 && b % i == 0)
      return false;
  }
  return true;
}
```

Punktvergabe. Insgesamt 20 Punkte für eine vollständig korrekte Lösung. Falls jemand die `gcd` Lösung verwendet, die Funktion `gcd` allerdings nicht hinschreibt, gibt es insgesamt 6 Punkte. Ansonsten gibt es Abzüge für Fehler. Hier eine ungefähre Zusammenstellung:

- Minus 1 für jeden Syntaxfehler ab dem zweiten (das heisst der erste ist noch frei). Damit sind zum Beispiel ein fehlendes Semikolon oder ein fehlender Abstand oder ähnlich gemeint.
- Minus 4 für falsche Initialisierung oder Abbruchbedingung der Schleife.
- Minus 2 wenn Teiler bis zu der grösseren der beiden Zahlen gesucht wird.
- Minus 4 für fehlde `return` Anweisungen.
- Minus 5-10 Punkte Abzug für fehlerhafte Implementation von `gcd`.
- Minus 2 für jeden Fehler bei der Verwendung der Funktion `gcd` respektive der Anwendung der modulo Tests.

Aufgabe 4. Die Aufgabenstellung fragt danach, die Problemstellung von UNIV auf die Problemstellung von HALT zu reduzieren. Das heisst, gegeben ein Algorithmus HALT, wie kann ich eine Algorithmus UNIV erstellen? Eine einfache Art das zu veranschaulichen ist mit einem Diagram wie in Abbildung 1.

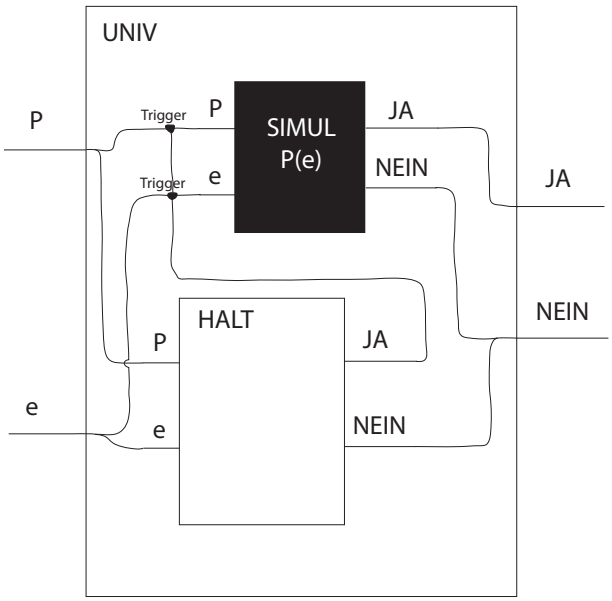

Abbildung 1: Reduktion von UNIV auf HALT.

Die Elemente in der Abbildung sind wie folgt zu lesen. Ein weisses Kästchen bezeichnet einen Algorithmus, das heisst wir können auch sicher gehen, dass der Block terminiert. Ein schwarzes Kästchen hingegen ist potentiell gefährlich, weil die darin verborgenen Anweisungen undendlich lange laufen könnten. Da wir davon ausgehen, dass es HALT gibt ist dessen Kästchen leer, das heisst es funktioniert per Voraussetzung. Nicht jedoch das Kästchen von UNIV, welches es gerade auszufüllen gilt. Am linken Rand des Kästchens bekommt der Algorithmus jeweils seine Eingaben. Am rechten Rand gibt er seine Ausgabe (JA oder NEIN) ab.

Das Kästchen, das mit SIMUL bezeichnet ist, lässt ein Programm, welches es als Eingabe bekommt, auf einer Eingabe, welche es ebenfalls als Eingabe bekommt, laufen – das heisst es simuliert das Programm auf einer Eingabe. Da es sich dabei um ein schwarzes Kästchen handelt, müssen wir zuerst mit dem HALT Algorithmus überprüfen, ob die Eingabe von UNIV (P und e) eine Programm-Eingabe Kombination darstellt, die hält. Wenn die Antwort NEIN ist, dann wissen wir schon, dass P die Eingabe e nicht akzeptieren wird, und UNIV kann ebenfalls NEIN ausgeben. Wenn die Antwort jedoch JA lautet, dann schalten wir SIMUL ein, um die Ausgabe von P auf e zu erhalten, welche wir dann gleich als Ausgabe von UNIV weiterleiten können. Die beiden Trigger Elemente sorgen also darfür, dass SIMUL nur belangt wird, wenn sichergestellt ist, dass P auf e wirklich hält.

Die Aufgabe kann auch mit einer entsprechenden Beschreibung (ohne Bild) gelöst werden.

Punktvergabe. Insgesamt 18 Punkte. Mehr als 9 Punkte sollten nur erteilt werden, wenn alle wesentlichen Elemente (HALT als building-block, korrekte Ein- und Ausgaben, SIMUL, richtige Richtung der Reduktion) vorhanden sind. Ansonsten ist die genaue Punktevergabe nach einem groben Quervergleich noch festzulegen.

Aufgabe 5.

a) Der Wert der Variablen `two` nach der ersten paar Schleifendurchläufe entwickelt sich wie folgt.

Schleifendurchgang	`two` dezimal	`two` binär
0	1	1.0
1	1.5	1.1
2	1.75	1.11
3	1.875	1.111
4	1.9375	1.1111
⋮	⋮	⋮

Das heisst, solange noch genügend bits in der Mantisse der Variablen `two` vorhanden sind, kommt bei der binären Darstellung in jedem Schleifendurchgang hinten eine 1 dazu.

b) Die korrekte Antwort ist iii), das heisst am Ende wird die 0 ausgegeben. Um dies zu begründen, muss man sich vergegenwärtigen, was in der binären Darstellung geschieht. In der Aufgabe a) haben Sie gesehen, dass die Variable `two` in jedem Schleifendurchgang eine 1 mehr speichern muss.

Der Typ `float` hat nach IEEE 754 eine Mantisse mit 24 bits zur Verfügung. Aufgrund der normalisierten Darstellung kann die Variable `two` also bis und mit dem 25. Schleifendurchgang alle Stellen speichern. Danach muss gerundet werden. Der exakte Wert der Approximation nach dem 26. Durchgang ist genau zwischen dem des 25. Durchgangs und der zu approximierenden 2. Da das round-to-even Schema angewendet wird, wird nach oben zur 2 gerundet.

Danach ändert sich an der Variablen `two` nichts mehr, da in allen folgenden Schleifendurchgängen ein immer kleiner werdender Betrag dazuaddiert wird. Dieser additive Term wird jeweils sogleich wieder abgerundet, um die gegebenen Platzverhältnisse zu befriedigen.

Das heisst, am Ende hat die Variable `two` exakt den Wert 2, und die Ausgabe ist somit exakt 0.

Punktvergabe. Jeweils 10 Punkte für a) und b).

a) Die 10 Punkte sind im Allgemeinen für die Einsicht in das richtige Schema (eine 1 mehr in jedem Schleifendurchgang) zu vergeben. Entsprechende Abzüge für

Fehler, oder teilweise falsches Schema. Minus 3 Punkte, wenn es eine Verschiebung des Laufindex `i` gibt.

b) Die richtige Begründung gibt 7 Punkte, die richtige Antwort 3 Punkte. Die Antwort i) ist mit 1 Punkt zu bewerten, plus eventuelle Zusatzpunkte aus der Begründung (vielleicht sind Teile davon richtig). Die Antwort ii) ist mit 0 Punkten zu bewerten, plus eventuelle Zusatzpunkte aus der Begründung.

Aufgabe 6. Die Teile a), b) und c) der Aufgabestellung sind in der folgenden Implementation vorhanden. Die Implementation verwendet keine Felder. Es sei erwähnt, dass die folgende Implementation bei weitem mehr liefert, als in der Aufgabenstellung gefordert war. Die entsprechenden Punkte wurden nur für die Deklaration der Datenmitglieder, für den Konstruktor und für die Funktion gewonnen erteilt.

```
#include <iostream>


class TicTacToe {

private:

  // Semantik fuer Spielerspeicherung:
  //  1 bedeutet Spieler 1
  // -1 bedeutet Spieler 2
  // alles andere ist spielerneutral

int feld1;
int feld2;
int feld3;
int feld4;
int feld5;
int feld6;
int feld7;
int feld8;
int feld9;

int spieler;

private:

  // Ausgabe des Spielersymbols
char symbol(int a) const {
if (a == 1) return 'x';
if (a == -1) return 'o';
return ' ';
```

```
}


public:
  // Klassenkonstruktor, alles Null-initialisiert
  TicTacToe() : feld1(0), feld2(0), feld3(0),
          feld4(0), feld5(0), feld6(0),
          feld7(0), feld8(0), feld9(0),
          spieler(1)
  {
  }


  // POST: Rueckgabewert ist der aktuelle Spieler, der in
  // der Klasse gespeichert wird.
  int aktuellerSpieler() const {
    return spieler;
  }


  // POST: true g.d.w. das entsprechende Feld noch leer ist.
  bool leeresFeld(unsigned int feld) const {
    switch (feld) {
      case 1: return feld1 == 0;
      case 2: return feld2 == 0;
      case 3: return feld3 == 0;
      case 4: return feld4 == 0;
      case 5: return feld5 == 0;
      case 6: return feld6 == 0;
      case 7: return feld7 == 0;
      case 8: return feld8 == 0;
      case 9: return feld9 == 0;
      default: return false;
    }
  }


  // PRE: spieler in {1,-1}, feld ist eines der unbesetzten
  // Felder
  // POST: Das entsprechende Symbol ist gesetzt und aktuelle
  //  Spieler hat gewechselt.
  void setzeZug(int spieler, unsigned int feld) {
    switch (feld) {
      case 1: feld1 = spieler;
        break;
      case 2: feld2 = spieler;
```

```
      break;
    case 3: feld3 = spieler;
      break;
    case 4: feld4 = spieler;
      break;
    case 5: feld5 = spieler;
      break;
    case 6: feld6 = spieler;
      break;
    case 7: feld7 = spieler;
      break;
    case 8: feld8 = spieler;
      break;
    case 9: feld9 = spieler;
  }
  this->spieler *= -1;
}


// PRE: Das Spiel befindet sich in einem gueltigen Zustand.
// POST: true falls das Spielbrett voll ist, false sonst.
bool brettVoll() const {
  for (unsigned int i = 1; i < 10; ++i) {
    if (leeresFeld(i))
      return false;
  }
  return true;
}

// POST: 1 wenn a positiv oder Null, -1 wenn a negativ
int vorzeichen(int a) const {
  if (a >= 0) return 1;
  else return -1;
}


// PRE: Das Spiel befindet sich in einem gueltigen Zustand.
// POST: Die Rueckgabe ist 1 falls Spieler 1 gewonnen hat,
//                        -1 falls Spieler 2 gewonnen hat und
//                         0 falls keine Aussage getroffen werden kann.
int gewonnen() const {
  int temp;

  temp = feld1 + feld2 + feld3;
  if (temp > 2 || temp < -2) return vorzeichen(temp);
  temp = feld4 + feld5 + feld6;
```

```
    if (temp > 2 || temp < -2) return vorzeichen(temp);
    temp = feld7 + feld8 + feld9;
    if (temp > 2 || temp < -2) return vorzeichen(temp);
    temp = feld1 + feld4 + feld7;
    if (temp > 2 || temp < -2) return vorzeichen(temp);
    temp = feld2 + feld5 + feld8;
    if (temp > 2 || temp < -2) return vorzeichen(temp);
    temp = feld3 + feld6 + feld9;
    if (temp > 2 || temp < -2) return vorzeichen(temp);
    temp = feld1 + feld5 + feld9;
    if (temp > 2 || temp < -2) return vorzeichen(temp);
    temp = feld7 + feld5 + feld3;
    if (temp > 2 || temp < -2) return vorzeichen(temp);

    return 0;
  }


  // PRE: Die Datenmitglieder haben gueltige Werte
  // POST: Das Spielbrett wird ausgegeben.
  void druckeBrett() const {
    std::cout << symbol(feld1) << " | " << symbol(feld2) << " | " <<
            symbol(feld3) << "\n" <<
            "---------\n" <<
            symbol(feld4) << " | " << symbol(feld5) << " | " <<
            symbol(feld6) << "\n" <<
            "---------\n" <<
            symbol(feld7) << " | " << symbol(feld8) << " | " <<
            symbol(feld9) << "\n";
  }
};


int main() {

  TicTacToe spiel = TicTacToe();
  int naechsterZug = 0;

  std::cout << "Das Brett sieht so aus; die Felder sind von";
  std::cout << "1 bis 9 nummeriert:\n";
  std::cout << "1" << " | " << "2" << " | " <<
        "3" << "\n" <<
        "---------\n" <<
        "4" << " | " << "5" << " | " <<
        "6" << "\n" <<
        "---------\n" <<
```

```
        "7" << " | " << "8" << " | " <<
        "9" << "\n";
  std::cout << "Das Spiel geht los...\n\n";

  while (spiel.gewonnen() == 0 && !spiel.brettVoll()) {
    spiel.druckeBrett();
    std::cout << "Gibt den Zug fuer Spieler ";
    std::cout << spiel.aktuellerSpieler() << " ein\n";
    std::cin >> naechsterZug;
    if (spiel.leeresFeld(naechsterZug)) {
      spiel.setzeZug(spiel.aktuellerSpieler(), naechsterZug);
    } else {
      std::cout << "Ein Fehler ist aufgetreten. Bitte noch einmal...\n";
    }
  }
  std::cout << "Game over!\n" << "Der Spieler ";
  std::cout << spiel.gewonnen() << " hat gewonnen.\n";
  spiel.druckeBrett();
}
```

Punktvergabe. Jeweils 8 Punkte für die drei Teile a), b) und c). Lösungen mit oder ohne Felder sind gleich zu bewerten.

a) 5 Punkte für geeignete Datenmitglieder. 3 Punkte für sinnvolle Semantik. Entsprechende Abzüge für Unzulänglichkeiten, z.B. minus 1 für fehlendes "aktueller Spieler" Feld, minus 1 für jeden syntaktischen Fehler ab dem zweiten (d.h. der erste ist noch frei).

b) 8 Punkte für korrekten Konstruktor. Abzüge für jeden syntaktischen oder semantischen Fehler minus 1. Initialisierung in Parameterliste oder Körper des Konstruktors sind gleich zu bewerten.

c) 2 Punkte für sinnvolle Postcondition. 5 Punkte für Implementation der Funktion.