Prüfung — Informatik D-MATH/D-PHYS — Lösung
9. 8. 2012 **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 (vier Blätter mit insgesamt 6 Aufgaben)! **Tragen Sie auf dem Deckblatt gut lesbar Namen, Vornamen und Stud.-Nr. ein.**
2. Erlaubte Hilfsmittel: **Keine. Einzige Ausnahme sind Wörterbücher.**
3. Betrugsversuche führen zu sofortigem Ausschluss und können rechtliche Folgen haben.
4. **Schreiben Sie Ihre Lösungen direkt auf die Aufgabenblätter!** Pro Aufgabe ist höchstens eine gültige Version eines Lösungsversuchs zulässig. **Tipp:** Lösungsentwürfe auf separaten Blättern vorbereiten und die fertige Lösung auf die Aufgabenblätter übertragen. Falls Sie eine Lösung ändern wollen, streichen Sie den alten Lösungsversuch klar erkennbar durch. Falls auf dem Aufgabenblatt nicht mehr genug Platz für Ihre neue Lösung vorhanden ist, benutzen Sie ein separates Blatt, das mit Ihrem Namen und der Aufgabennummer beschriftet ist.
5. Wenn Sie frühzeitig abgeben möchten, übergeben Sie Ihre Unterlagen bitte einer Aufsichtsperson und verlassen Sie den Raum.
6. **Ab 10.50 Uhr kann nicht mehr frühzeitig abgegeben werden. Bleiben Sie an Ihrem Platz sitzen, bis die Prüfung beendet ist und ihre Unterlagen von einer Aufsichtsperson eingesammelt worden sind.**
7. Die Prüfung ist bestanden, wenn Sie 60 von 120 Punkten erzielen. **Viel Erfolg!**

1	2	3	4	5	6	$\sum$

Aufgabe 1. (21 Punkte) Geben Sie zu den folgenden Ausdrücken jeweils den Typ (1 Punkt) und den Wert (2 Punkte) an. Es gibt keine Punkte für die Teilschritte der Auswertung. Sie brauchen diese nicht hinzuschreiben.

Ausdruck	Typ	Wert
`3 + 17 * 2`	`int`	37
`19 / 2`	`int`	9
`19 / 2 / 2.0`	`double`	4.5
`1 > 0 || 1 > 1 && 0 > 1`	`bool`	true
`1.0e3 / 2.0e2`	`double`	5
`123456 % 3`	`int`	0
`! true`	`bool`	false

Aufgabe 2. (12 Punkte) Geben Sie für die folgenden drei Codefragmente jeweils die Ausgabe an!

a)
```
for (int i=64; i>0; i/=2)
    std::cout << (i%2) << " ";
```

Ausgabe: `0 0 0 0 0 0 1`

---

b)
```
for (int i=1; i<=6; ++i)
  for (int j=1; j<=6; ++j)
    if (i+j==7) std::cout << "(" << i << "," << j << ")";
```

Ausgabe: `(1,6)(2,5)(3,4)(4,3)(5,2)(6,1)`

---

c)
```
int a[] = {2,0,4,1,3};
int i=0;
do {
  std::cout << i;
  i = a[i];
} while (i != 0);
```

Ausgabe: `02431`

Aufgabe 3. (20 Punkte) Ein Palindrom ist eine Zeichenfolge, die vorwärts wie rückwärts gelesen gleich ist, wie zum Beispiel anna oder maoam. Die Zeichenfolge Anna ist im Sinne dieser Aufgabe kein Palindrom. Geben Sie Ausdrücke expr1 und expr2 an, so dass die folgende Funktion korrekt ist.

```
// PRE:  [begin, end) ist ein gueltiger, moeglicherweise leerer
//       Bereich von Zeichen
// POST: gibt genau dann true zurueck, wenn der Bereich ein
//       Palindrom enthaelt, also vorwaerts wie rueckwaerts
//       gelesen die gleiche Zeichenfolge ergibt.
bool is_palindrome (const char* begin, const char* end)
{
  if ((begin >= end))
    return expr1;
  else
    return expr2;
}
```

expr1:
true

---

expr2:
*begin == *(end-1) && is_palindrome (begin+1, end-1)

Hinweis: Seien

```
char* s = "anna";
char* t = "hugo";
char* u = "maoam";
char* v = "nelke";
```

Eine korrekte Funktion liefert zum Beispiel die folgenden Werte:

Funktionsausdruck	Wert
is_palindrome (s, s+4)	*true*
is_palindrome (t, t+4)	*false*
is_palindrome (u, u+5)	*true*
is_palindrome (v, v+5)	*false*

Aufgabe 4. (18 / 2 Punkte) a) Ergänzen Sie die folgenden zwei Funktionen so, dass sich ein korrektes Hauptprogramm (`main`-Funktion) ergibt. Sie dürfen dabei weniger, aber **nicht mehr** als die vorgegebene Anzahl von Zeilen benutzen, wobei jede Zeile **höchstens eine** Anweisung enthalten darf. b) Welche Zahl ist die erste in der Ausgabeliste?

```
// PRE:  n > 0
// POST: gibt die Anzahl der Teiler
//       von n zurueck (inkl. 1, n)
int number_of_divisors (int n) {
  int result = 0;
  for (int i=1; i<=n; ++i)
    if (n % i == 0)
      ++result;
  return result;
}

// POST: listet alle Zahlen zwischen 1
//       und 1000 mit genau 4 Teilern
int main() {
  for (int n=1; n<=1000; ++n)
    if (number_of_divisors (n) == 4)
      std::cout << n << " ";
  return 0;
}
```

Erste Ausgabezahl: 6

Aufgabe 5. (5 / 12 Punkte) Gegeben seien n verschiedene reelle Zahlen, wobei wir annehmen, dass n eine Zweierpotenz ist, also $n = 2^k$ für eine geeignete natürliche Zahl k. Betrachten Sie den *Wimbledon-Algorithmus* zur Bestimmung des Maximums der n Zahlen durch ein Turnier, wobei in jeder Runde die Hälfte der verbleibenden Zahlen eliminiert wird. Dazu werden die jeweils verbleibenden Zahlen zu Paaren zusammengestellt, und von jedem Paar wird durch einen Vergleich die grössere Zahl bestimmt. Die "Gewinner" kommen in die nächste Runde. Der "Sieger" dieses Turniers ist dann das Maximum der n Zahlen.

a) Sei $T(n)$ die Anzahl der Vergleiche, die der Wimbledon-Algorithmus benötigt, um das Maximum von $n = 2^k$ Zahlen zu berechnen. Geben Sie eine Formel für $T(n)$ an! Sie müssen Ihre Antwort nicht begründen.

$$T(n) = n - 1$$

---

b) Wie viele *zusätzliche* Vergleiche zwischen Zahlen werden benötigt, um nach Ende des Turniers auch noch die zweitgrösste Zahl zu berechnen? Offenbar kann man die "Spiele" des Turniers ignorieren und mit $n - 2$ Vergleichen trivial das Maximum der $n - 1$ Kandidatenzahlen berechnen. Gesucht ist aber ein (wesentlich) besseres Verfahren. Beschreiben Sie Ihr Verfahren und geben Sie die dabei benötigte Anzahl von Vergleichen an!

Die zweitgrösste Zahl ist irgendwann im Turnier ausgeschieden. Das kann nur in einem "Spiel" gegen die grösste Zahl passiert sein. Die zweitgrösste Zahl ist also einer der "Gegner" der grössten Zahl. Bei 2^k Spielern hat das Turnier k Runden, wir müssen also das Maximum der k "Gegner" der grössten Zahl bestimmen. Das geht mit $k - 1$ Vergleichen.

Aufgabe 6. (30 Punkte) Definieren Sie eine Klasse `Averager`, die Durschnittsberechnungen durchführen kann. Hier ist ein Programm, das die gewünschte Funktionalität illustriert, indem es die durchschnittliche Körpergrösse dreier Menschen berechnet. Allgemein ist der Durchnitt einer Folge von $n > 0$ Werten definiert als die Summe der Werte, geteilt durch n.

```
int main() {
  Averager m;
  m.add_value (1.85);                              // Mensch 1
  m.add_value (1.79);                              // Mensch 2
  m.add_value (1.64);                              // Mensch 3
  std::cout << m.average_value() << std::endl;  // Ausgabe: 1.76

  return 0;
}
```

Geben Sie eine komplette Klassendefinition an, wobei Sie die im obigen Programm illustrierte Funktionalität durch geeignete Datenmitglieder und Mitgliedsfunktionen (Konstruktor, `add_value` und `average_value`) realisieren. Achten Sie auf korrekte Verwendung von `public` und `private` sowie auf Const-Korrektheit! **Jeder Funktionsrumpf darf aus maximal zwei Anweisungen bestehen.**

```
// Klasse zur Durschnittsberechnung von n Werten
class Averager {
private:
  unsigned int n;    // number of values
  double s;          // sum of values

public:
  Averager ()
    : n(0), s(0.0)
  {}

  void add_value (const double v)
  {
    s += v;
    n += 1;
  }

  // PRE: n > 0
  double average_value () const
  {
    assert (n > 0);
    return s/n;
  }
};
```