

Prüfung — Informatik D-MATH/D-PHYS
24. 1. 2014

09:00–11:00

Prof. Bernd Gärtner

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 doppelseitige Blätter mit insgesamt 6 Aufgaben und ein leeres Notizblatt)! **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 10:45 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		Σ

Aufgabe 1. (18 Punkte) Geben Sie für jeden der folgenden 6 Ausdrücke C++-Typ und Wert an! Zwischenschritte der Auswertung geben keine Punkte. Nehmen Sie an, dass `x` vom Typ `int` ist, und zu *Beginn jeder Teilaufgabe* den Wert 3 hat.

Ausdruck	Typ	Wert
<code>0 < 1 && 1 != 1 ++x > 3</code>		
<code>12.0 / 3u / 2u</code>		
<code>2 + x++</code>		
<code>1.4e3 * 0.2e-2</code>		
<code>2014 % 4 + x % 2</code>		
<code>x / 2 + 9.0f / 6</code>		

Aufgabe 2. (15 Punkte) Geben Sie für jedes der drei folgenden Code-Fragmente die Folge von Zahlen an, die das Fragment ausgibt!

a)

```
for (int i=0; i<100; i*=2)
    std::cout << ++i << " ";
```

Ausgabe:

b)

```
int i=0;
int j=5;
while (i != j)
    std::cout << (i+=2) - j++ << " ";
```

Ausgabe:

c)

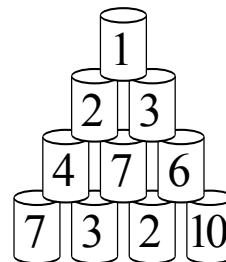
```
void f(unsigned int x) {
    if (x == 0)
        std::cout << "-";
    else {
        std::cout << "*";
        f(x-1);
        std::cout << "*";
    }
}
f(5);
```

Ausgabe:

Aufgabe 3. (23 Punkte) Beim Büchsenwerfen sind Büchsen pyramidenförmig angeordnet. Jeder Büchse ist eine bestimmte Punktzahl zugeordnet. Diese Punktzahlen sind in einem zweidimensionalen Feld (Array) gespeichert. Der erste Index des Feldes bestimmt die Reihe in der die Büchse steht und der zweite Index die Position innerhalb der Reihe. Wir zählen Reihen von oben nach unten, beginnend bei 0, und Positionen (Kolonnen) von links nach rechts, beginnend bei 0. Im Beispiel unten hat die Büchse in Reihe 2 an Position 0 den Wert 4 ($D[2][0] == 4$).

Wird eine Büchse getroffen, fällt diese herunter. Es fallen auch alle Büchsen die direkt oberhalb einer fallenden Büchse stehen herunter. Wird zum Beispiel die Büchse mit der Punktzahl 7 in der Mitte der Reihe 2 getroffen, fallen die Büchsen mit den Punktzahlen 7, 2, 3 und 1.

```
int D[4][4] = {{1, 0, 0, 0},
               {2, 3, 0, 0},
               {4, 7, 6, 0},
               {7, 3, 2, 10}};
```



Im folgenden finden Sie ein Skelett für eine Funktion `hit`, die die Summe der Punktzahlen auf den gefallen Büchsen berechnet, falls mit dem Wurf die Büchse an Position `c` in der Reihe `r` getroffen wurde. Ergänzen Sie das Skelett zu einer korrekten Funktionsdefinition, indem Sie die vier Ausdrücke `expr1`, `expr2`, `expr3` und `expr4` angeben! Sie können davon ausgehen, dass sie in der Funktion direkt auf das Feld `D` zugreifen können, d.h. `D` ist eine globale Variable.

```
// PRE:  the array D has size at least (n+1) x (n+1) (n = max(r, c)),
//       D[r][c] denotes the value of the can in row r, column c
//       where c <= r
// POST: returns the sum of the values of all cans that fall
//       if the can at position/column c in row r gets hit
int hit(int r, int c)
{
    if (r == 0)
        return expr1;
    if (c == 0)
        return expr2;
    if (c == r)
        return expr3;
    return expr4;
}
```

```
// PRE:  the array D has size at least (n+1) x (n+1) (n = max(r, c)),
//       D[r][c] denotes the value of the can in row r, column c
//       where c <= r
// POST: returns the sum of the values of all cans that fall
//       if the can at position/column c in row r gets hit
int hit(int r, int c)
{
    if (r == 0)
        return expr1;
    if (c == 0)
        return expr2;
    if (c == r)
        return expr3;
    return expr4;
}
```

expr1:

expr2:

expr3:

expr4:

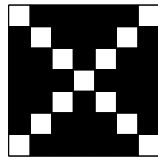
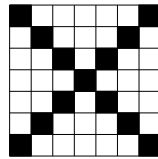
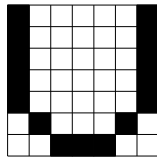
Aufgabe 4. (24 Punkte) Ihre Aufgabe ist es, eine Funktion zu implementieren, die eine Fließkommazahlen-Approximation der Exponentialfunktion

$$e^x := \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

liefert, für eine gegebene reelle Zahl x . Die Implementierung soll auf der Definition von e^x als unendliche Reihe basieren. Um eine Approximation von e^x zu erhalten, soll die Funktion die ersten Terme dieser Reihe aufsummieren (mit `double`-Arithmetik), *bis* die Addition des aktuellen Terms den bisher erhaltenen Wert nicht mehr ändert. Sie dürfen *keine* externen Bibliotheken benutzen. Die *gesamte* Anzahl an Anweisungen in Ihrer Lösung darf die vorgegebene Anzahl an Zeilen nicht überschreiten.

```
// POST: returns e^x
double exp (double x)
{
// 1
// 2
// 3
// 4
// 5
// 6
// 7
// 8
// 9
// 10
// 11
}
```

Aufgabe 5. (14 / 12 Punkte) Betrachten Sie die folgende partielle Deklaration einer Klasse zur Repräsentierung von Schwarz-Weiss Bildern mit 7×7 Pixeln. Hier sind drei Beispiele für solche Bilder.



← Bild für Aufgabenteil b)

```
// a class for 7x7 black-and-white pixel images
class image {
private:
    // bitmap[r][c] = true if and only if the pixel
    // in row r and column c is black
    bool bitmap[7][7];

public:
    // POST: *this is the empty image (all pixels are white)
    image ();

    // PRE: 0 <= r,c < 7
    // POST: the pixel in row r and column c in *this is set to black
    void set_pixel (int r, int c);

    // POST: the inverted image of *this is returned (black pixels
    // become white and white pixels become black)
    image operator! () const;
};
```

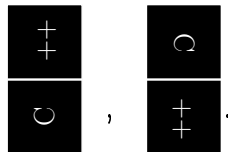
- a) Geben Sie korrekte Definitionen für die drei öffentlichen Mitgliedsfunktionen an! Die *gesamte* Anzahl an Anweisungen in Ihrer Lösung darf die jeweils vorgegebene Anzahl an Zeilen nicht überschreiten.

```
image::image()
{
// 1
// 2
// 3
}
```

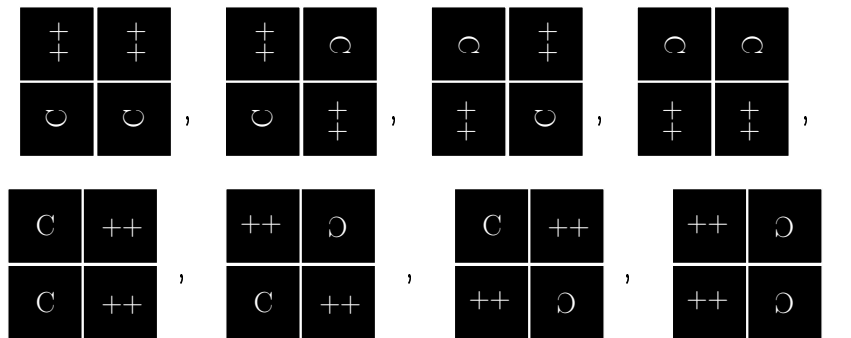

Aufgabe 6. (10 / 4 Punkte) Sei n eine positive natürliche Zahl. Ein Spielfeld der Grösse $2 \times n$ soll vollständig mit Dominosteinen der Grösse 1×2 belegt werden, wobei jeder Dominostein wie folgt aussieht:



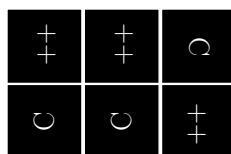
Sei $T(n)$ die Anzahl der verschiedenen Möglichkeiten, das Spielfeld zu belegen. Wir erhalten zum Beispiel $T(1) = 2$:



Für $n = 2$ ergibt sich $T(2) = 8$:



Für $n = 3$ zeigen wir hier nur eine der vielen Möglichkeiten:



- a) Geben Sie eine rekursive Formel für $T(n)$ an, für $n \geq 3$. Sie müssen die Korrektheit der Formel nicht beweisen.

$$T(n) = \quad , \text{ falls } n \geq 3.$$

- b) Berechnen Sie unter Benutzung von a), oder auf irgendeine andere Weise, den Wert $T(4)$.

$$T(4) =$$