Prüfung — Informatik D-MATH/D-PHYS
25. 1. 2013 09:00–11:00

Dr. 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 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) Bestimmen Sie für jeden Ausdruck in der folgenden Tabelle den Typ (1 Punkt) sowie den Wert (2 Punkte) und tragen Sie beides in der Tabelle ein. Es ist nicht nötig Ihre Angaben zu begründen.

Expression	Type	Value
`10 + 4 / 2`		
`21 / 4`		
`10 / 3 / 2.0`		
`2 == 3 || 1 != 0 && 1 > 0`		
`3.0e3 / 1.0e4`		
`10128 % 4`		
`!  false || true`		

Aufgabe 2. (15 Punkte) Bestimmen Sie die jeweilige Ausgabe der folgenden drei Codefragmente.

a)
```
for (int i = 10; i >= 0; i /= 2)
  std::cout << i << " ";
```

Output:

---

b)
```
int i = 10;
int j = 0;
while (i != j)
  std::cout << i-- + ++j << " ";
```

Output:

---

c)
```
int arr[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
for (int l = 0; l < 3; ++l) {
  for (int k = 0; k < 3; ++k)
    std::cout << arr[k][l] << " ";
  std::cout << std::endl;
}
```

Output:

Aufgabe 3. (12 Punkte) Eine Zahl $n \in \mathbb{N}$ ist *vollkommen*, gdw. sie gleich der Summe ihrer positiven ganzzahligen Teiler ist. D.h., falls $n = \sum_{k \in \mathbb{N}, \text{ mit } k<n \wedge k|n} k$. Zum Beispiel ist $28 = 1 + 2 + 4 + 7 + 14$ vollkommen, $12 < 1 + 2 + 3 + 4 + 6$ hingegen nicht.

Ersetzen Sie `expr1`, `expr2` und `expr3` in unten stehendem Code durch je einem booleschen Ausdruck, so dass die gegebene Funktion `is_perfect` ihre Spezifikation implementiert, d.h., dass sie ihrer Vor- und Nachbedingung genügt.

```
// PRE:  true
// POST: This method returns true if the given n is a perfect number;
//       false otherwise.
bool is_perfect (unsigned int n)
{
  int sum = 0;
  for (int i = 1; expr1; i++)
    if (expr2)
      sum += i;
  return expr3;
}
```

expr1:

---

expr2:

---

expr3:

Aufgabe 4. (12 + 14 Punkte) Das *kleinste gemeinsame Vielfache* (*least common multiple*, lcm) zweier positiver ganzer Zahlen a und b ist die *kleinste positive* ganze Zahl, die durch a und b teilbar ist. Sei s eine Folge positiver ganzer Zahlen. Wir definieren das *kleinste gemeinsame Vielfache* von s als die *kleinste positive* Zahl, die sich durch alle Folgeglieder teilen lässt.

Ihre Aufgabe ist das Implementieren der folgenden beiden Funktionen, allerdings *ohne*

externe Bibliotheken zu nutzen. Die *gesamte* Anzahl an Anweisungen in Ihrer Lösung darf die vorgegebene Anzahl an Zeilen nicht überschreiten.

Tipp: Die folgenden Rechenregeln könnten nützlich sein:

1. $\mathrm{lcm}(a_1, a_2, \ldots, a_n) = \mathrm{lcm}(a_1, \mathrm{lcm}(a_2, \ldots, a_n))$
2. Das kleinste gemeinsame Vielfache der leeren Folge ist 1.

```
// PRE:  a,b > 0
// POST: returns the least positive integer
//       divisible by both a and b.
unsigned int lcm (unsigned int a, unsigned int b)
{
                                                  // 1
                                                  // 2
                                                  // 3
                                                  // 4
}
```

```
// PRE:  [begin,end) is a possibly empty
//       sequence of positive integers
// POST: returns the least common multiple
//       of all the integers in the given
//       range [begin, end)
unsigned int sequence_lcm(const unsigned int* begin,
                          const unsigned int* end)
{
                                                  // 1
                                                  // 2
                                                  // 3
                                                  // 4
}
```

Aufgabe 5. (18 + 8 Punkte) Ein *Binärbaum* (*binary tree*) ist eine Baumdatenstruktur in der jeder Knoten (*node*) höchstens zwei *Kindknoten* ((*child nodes*)) hat, nämlich einen linken (`left_`) und einen rechten (`right_`). Das Interface der Klasse `TreeNode` ist in Abb. 1 angegeben.

```
class TreeNode
{
public:
  // Constructor
  TreeNode(TreeNode* left = 0, TreeNode* right = 0);

  // POST: returns a constant pointer to the left child
  const TreeNode* get_left() const;
  // POST: returns a constant pointer to the right child
  const TreeNode* get_right() const;

private:
  TreeNode* left_;
  TreeNode* right_;
};
```

Figure 1: The interface of the class `TreeNode`

Das Interface der Klasse `Tree` ist in Abb. 2 angegeben.

```
#include "TreeNode.h"

class Tree {
public:
  // NOTE: This is a partial interface of the Tree class, this class
  //       also includes constructors, a destructor and other
  //       tree-manipulating methods (e.g. insertion and removal of
  //       tree nodes). However, they are not relevant for this
  //       assignment, hence, not provided.

  // POST: returns the number of leaves of *this
  int number_of_leaves() const;
  // POST: returns the number of leaves of the tree rooted at *current
  int number_of_leaves(const TreeNode* current) const;

private:
  // *this is rooted on the node root_
  TreeNode* root_;
};
```

Figure 2: The interface of the class `Tree`

Ein Knoten n eines Baumes T ist ein *Blatt* (*leaf*) von T gdw. n keine Kindknoten hat. So hat z.B. der leere Baum gar keine Blätter, während der Baum der nur aus dem einen Knoten n besteht genau ein Blatt hat, nämlich n selbst.

Ihre Aufgabe ist das Implementieren der folgenden Funktionen:

```
// POST: returns the number of leaves of the tree rooted at *current
int Tree::number_of_leaves(const TreeNode* current) const
{









}

// POST: returns the number of leaves of *this
int Tree::number_of_leaves() const
{









}
```

Aufgabe 6. (5 + 15 Punkte) Lösen Sie folgende Aufgabe unter Berücksichtigung der in **Aufgabe 5** gegebenen Definitionen eines *Binärbaumes* und eines *Blattes*:

a) Sei $minLeaves(n)$ eine Funktion, die für eine gegebene Anzahl an Knoten n eine *scharfe* obere Schranke für die Anzahl der Blätter angibt, die ein Baum mit n Knoten haben kann. Geben Sie die Definition von $minLeaves(n)$ and und begründen Sie, warum Ihre Definition richtig ist.

b) Beweisen Sie, dass ein Binärbaum mit n Knoten höchstens $\frac{n+1}{2}$ Blätter haben kann.