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

Dr. Bernd Gärtner

Examinee:

Last name: .....

First name: .....

Stud. no.: .....

With my signature I attest, that I was able to sit the exam under regular conditions and that I read and understood the general remarks.

Signature: .....

General remarks:

1. Verify the completeness of the exam given to you (four pages with six assignments in total)! **Fill in the title sheet, i.e., state your name and your student number, in a legible form.**
2. Allowed utilities: **None, except dictionaries.**
3. Cheating or attempts to cheat lead to immediate disqualification and may entail legal consequences.
4. **Write your solutions directly on the assignment sheets!** Only one solution attempt per assignment will be considered. **Hint:** Prepare your solutions on separate sheets and transfer only the final solutions to the assignment sheet. Clearly cross out solution attempts that you do not want to be considered. If the space provided on the assignment sheets is insufficient for your solution attempts, use separate sheets and label them with the assignment numbers and your name.
5. If you want to submit early, hand all relevant documents over to one of the invigilators before leaving the room.
6. **After 10:50, it is no longer possible to submit early. Please remain seated until the exam ends and until your documents have been collected by the invigilators.**
7. The exam's pass mark is 60 out of 120 points. **Good luck!**

1	2	3	4	5	6	$\sum$

Assignment 1. (21 Points) For each expression in the following table, determine its type (1 point) and its value (2 points) and fill them in the provided table. There are no extra points for writing down your reasoning and you are not required to do so.

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`		

Solution

Expression	Type	Value
`10 + 4 / 2`	`int`	`12`
`21 / 4`	`int`	`5`
`10 / 3 / 2.0`	`double`	`1.5`
`2 == 3 || 1 != 0 && 1 > 0`	`bool`	`true`
`3.0e3 / 1.0e4`	`double`	`0.3`
`10128 % 4`	`int`	`0`
`!  false || true`	`bool`	`true`

Assignment 2. (15 Points) For each of the following 3 code fragments, determine their output.

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:

Solution

a) `10 5 2 1 0 0 0 0 0 ...`

This code fragment contains an infinite loop.

b) `11 11 11 11 11`

c)
```
1 4 7
2 5 8
3 6 9
```

Assignment 3. (12 Points) A number $n \in \mathbb{N}$ is called *perfect* if and only if it is equal to the sum of its proper divisors, that is, $n = \sum_{k \in \mathbb{N},\, \text{s.t. } k<n \,\wedge\, k|n} k$. For example, $28 = 1+2+4+7+14$ is perfect, while $12 < 1+2+3+4+6$ is not.

Replace `expr1`, `expr2` and `expr3` in the code below with 3 boolean expressions (one for each) so that the given function `is_perfect` correctly implements its specification, i.e. complies with its pre- and post-conditions.

```
// 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:

Solution

```
expr1:  i < n
expr2:  n % i == 0
expr3:  sum == n
```

Assignment 4. (12 + 14 Points) The *least common multiple* (lcm) of two positive integers a and b is the *smallest positive* integer that is divisible by both a and b. Let s be a sequence of positive integers. We say that the *least common multiple* of the given sequence s is the *smallest positive* integer that is divisible by all the numbers in s.

Your task is to provide an implementation for the following two functions. However, you are *not* allowed to use any external libraries. The *total* number of statements in your solution should not be greater than the number of lines provided.

Hint: The following facts may help you with your implementation:

1. $\mathrm{lcm}(a_1, a_2, \ldots, a_n) = \mathrm{lcm}(a_1, \mathrm{lcm}(a_2, \ldots, a_n))$
2. The least common multiple of the empty sequence is 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
}
```

Solution

```
/*
 Least Common Multiple
 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)
{
for (unsigned int temp = a; ; temp += a)
if (temp % b == 0)
return temp;
}

/*
 PRE: [begin,end) is a possibly empty sequence of positive integers
 POST: returns the least common multiple of all the integers in the giv
*/
unsigned int sequence_lcm(const unsigned int* begin, const unsigned int
{
return (begin >= end) ? 1 : lcm (*begin, sequence_lcm (begin+1, end));
}
```

Assignment 5. (18 + 8 Points) A *binary tree* is a tree data structure in which each node has at most 2 *child nodes* that are denoted as `left_` and `right_`. The interface of the class `TreeNode` is depicted in Figure 1.

The interface of the class `Tree` is depicted in Figure 2.

```
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`

```
#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`

A node n of a tree T is called a *leaf* of T if and only if n has no child nodes. E.g. an empty tree has no leaves and a tree containing only one node n, has exactly one leaf, namely the leaf n.

Your task is to provide the implementation of the following functions:

```
// 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
{
```

```
}
```

Solution

```
// POST: returns the number of leafs of the tree
   rooted at *current
int Tree::number_of_leafs(const TreeNode* current)
   const
{
if (current == 0)
return 0;
if (current->get_left() == 0 && current->get_right()
   == 0)
return 1;
return number_of_leafs(current->get_left())
       + number_of_leafs(current->get_right());
}

// POST: returns the number of leafs of *this
int Tree::number_of_leafs() const
{
return number_of_leafs(root_);
}
```

Assignment 6. (5 + 15 Points) Assuming the definitions of a *binary tree* and a *leaf* from **Assignment 5**, your tasks are:

a) Let $\mathcal{T}_n$ be a set of all binary trees with n nodes such that $n \geq 0$ and let $leaves(T)$ be a function that returns the number of leaves of the given binary tree T. Let $minLeaves(n) = min(\{leaves(T) : T \in \mathcal{T}_n\})$. What are the possible values of $minLeaves(n)$ and explain how they relate to the value of n? Argue the correctness of your answer.

b) Prove that a binary tree with n nodes has at most $\frac{n+1}{2}$ leaves.

Solution

a)

$$\mathit{minLeaves}(n) = \begin{cases} 0, & \text{if } n \leq 0 \\ 1, & \text{if } n > 0 \end{cases}$$

The reasoning is that if a tree is empty (i.e. $n = 0$), the tree has no leaves by definition and for any $n > 0$, the nodes can form a chain, hence, the tree has only one leaf.

b) By induction on n. For $n = 0, 1$ the statement holds, since in these cases, there are $0 < (0 + 1)/2$ and $1 = (1 + 1)/2$ leaves. Now let $n > 2$ and suppose the statement holds for all smaller n. Consider the left and right subtrees T_ℓ, T_r of the root, with $n_\ell \geq 0$ and $n_r \geq 0$ nodes, respectively, where $n_\ell + n_r + 1 = n$. Using the inductive hypothesis, we know that the tree has at most

$$\frac{n_\ell + 1}{2} + \frac{n_r + 1}{2} = \frac{n_\ell + n_r + 2}{2} = \frac{n + 1}{2}$$

leaves.