Exam — Informatik D-MATH/D-PHYS
9. 8. 2013 15:00–17:00

Prof. 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 (three two-sided sheets with 6 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 16:50, 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. (18 Points) For each of the following 6 expressions, provide type and value! No points will be awarded for intermediate steps of the evaluation.

Expression	Type	Value
`11u + 6u / 3u`		
`23 / 7`		
`10 / 4.0f / 2`		
`3.0e4 * 2.0e-3`		
`1612795 % 5`		
`double(1/2) == 0.5`		

Assignment 2. (15 Points) For each of the following three code fragments, write down the sequence of numbers that it outputs!

---

a)
```
for (int i=1; i<1000; i*=3)
  std::cout << i << " ";
```

Output:

---

b)
```
int a=0;
int b=1;
for (int h; (h=a)<50; b+=h)
  std::cout << (a=b) << " ";
```

Output:

---

c)
```
unsigned int x = 1;
do {
  std::cout << x << " ";
  x = (7 * x + 4) % 9;
} while (x != 1);
```

Output:

Assignment 3. (4 / 16 Points) A natural number $n \geq 1$ is called *simple* if it is of the form $n = 2^k \cdot 3^\ell$ for some natural numbers $k, \ell \geq 0$. In other words, n is simple if it is a product of a power of 2 and a power of 3. For example, all powers of 2, 3, or 6 are simple. Other simple numbers are $18 = 2 \cdot 3^2$ and $24 = 2^3 \cdot 3$.

The following is the skeleton for a function to test whether a given natural number is simple. Complete the skeleton to a correct function by providing the two expressions `expr1` and `expr2`!

```
// PRE: n > 0
// POST: returns true if and only if n is simple, that means n is of
//       the form 2^k * 3^l for some natural numbers k,l >= 0
bool is_simple (unsigned int n)
{
  if (n == 1)
    return expr1;
  else
    return expr2;
}
```

`expr1:`

`expr2:`

Assignment 4. (25 Points) A *permutation* of the set $\{1, 2, \ldots, n\}$ is a bijective function $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$. For example, this is a tabular representation of a permutation of $\{1, 2, 3, 4, 5\}$:

i	1	2	3	4	5
$\pi(i)$	5	3	2	4	1

An *inversion* in a permutation π is a pair (i, j) such that $1 \leq i < j \leq n$ and $\pi(i) > \pi(j)$. In the above example, $(1, 2)$ is an inversion, since $\pi(1) = 5 > 3 = \pi(2)$. The pair $(3, 4)$ is not an inversion, since $\pi(3) = 2 < 4 = \pi(4)$. Provide an implementation of the following function for counting the number of inversions in a given permutation. (The permutation above has 8 inversions.) The total number of statements must not exceed the given number of lines.

```
// PRE: the range [begin, end) stores a permutation
//      pi of the set {1,2,...,end-begin}, i.e. the
//      i-th element of the range is pi(i), for all
//      i in {1,2,...,end-begin}
// POST: the number of inversions of pi is returned
int inversions (const int* begin, const int* end)
{
                                                    // 1
                                                    // 2
                                                    // 3
                                                    // 4
                                                    // 5
                                                    // 6
}
```

Assignment 5. (30 Points) Recall from the lecture that a *list* is a data structure for storing a sequence of keys of the same type. Unlike an array, a list allows efficient insertion and removal "in the middle". For the type `int`, here are the parts of the definition of the class `List` that are relevant for this assignment.

```
class List{
public:
  ...
  // POST: returns const pointer to the first node
  const ListNode* get_head() const;
  ...
private:
  ...
  ListNode* head_;
  ...
};
```

Hence, a list stores a (possibly null) pointer to a head node whose key represents the first element of the list. Starting from the head node, we can traverse the list by following `get_next()` pointers of the class `ListNode` whose relevant part of the definition is given next.

```
class ListNode {
```

```
 public:
  ...
  int get_key() const;
  ListNode* get_next() const;
  ...
 private:
  int key_;
  ListNode* next_;
};
```

Implement an equality test for two lists in form of a global operator `operator==`! Two lists are equal if they store the same sequence of keys.

```
// POST: returns true if and only if l1 and l2
//       store the same sequence of keys
bool operator== (const List& l1, const List& l2)
{









}
```

Assignment 6. (9 / 3 Points) A *binary search tree* is either empty, or it has a *root* node with a *key* (a real number), a *left* subtree, and a *right* subtree, satisfying the following properties.

- The left subtree is a binary search tree that contains only nodes with keys *smaller* than the root key.
- The right subtree is a binary search tree that contains only nodes with keys *larger* than the root key.

Let T_N be the set of all binary search trees that contain all the keys from the set $\{1, \ldots, N\}$ for some given natural number $N \geq 0$. That means, each tree in T_N has N nodes. Figure 1 shows the sets T_1, T_2 and T_3.

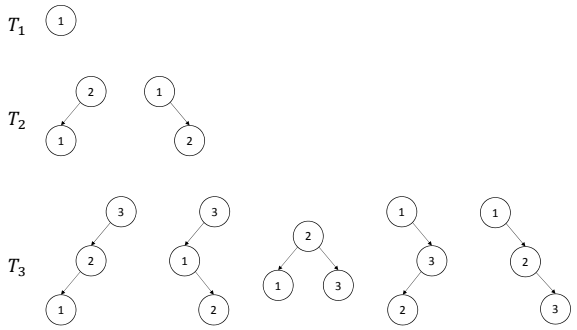

Figure 1: The sets T_1, T_2, and T_3

a) Provide a *recursive* definition of the function $f : \mathbb{N} \to \mathbb{N}$ such that $f(N) = |T_N|$. This means, $f(N)$ is the number of binary search trees with key set exactly $\{1, \ldots, N\}$. You don't have to argue why your definition is correct. **Hint**: With your definition, you should get $f(0) = f(1) = 1, f(2) = 2, f(3) = 5$.

---

b) Using a) (or any other way), compute the value $f(4)$. You don't have to argue why the value that you get is correct.

$$f(4) =$$