Exam A — Informatik D-MATH/D-PHYS
17. 12. 2013 13:15–14:55

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 5 assignments in total and one empty sheet for notes)! **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 14:40, 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 50 out of 100 points. **Good luck!**

1	2	3	4	5	$\sum$

Assignment 1. (18 Points) For each of the following 8 expressions, provide type and value! No points will be awarded for intermediate steps of the evaluation. Assume the IEEE 754 floating point standard.

Expression	Type	Value
`7u + 7u / 2u`		
`3.0f / 2u`		
`5u / 2 * 3`		
`0.2f == 1.0/5`		
`2 + 2013 % 2 * 3`		
`int(1.3) + 2.0`		

Assignment 2. (16 Points) For each of the following four code fragments, write down the sequence of numbers that it outputs!

---

a)
```
for (int i=100; i!=0; i/=3)
  std::cout << ++i << " ";
```
Output:

---

b)
```
void g(unsigned int x) {
  std::cout << x%4 << " ";
  if (x/4 > 0) g(x/4);
}
g(72);
```
Output:

---

c)
```
for(int i=1; i<10; i+=3)
  for(int j=i; j<10; j*=2)
    std::cout << j << " ";
```
Output:

---

d)
```
float d[] = {0.6, 0.7, 0.8, 0.9};
float i = 0;
do {
  std::cout << (i+=d[int(i)]) << " ";
} while (i<3);
```
Output:

Assignment 3. (8 / 16 Points) You plan to go on a vacation trip after christmas. You used the internet to gather information on your destination and you found a list with the predicted amount of rain on every single day. You store these numbers of type `float` in an array. The array could for instance look like this:

```
float rain[] = {9.5, 4.1, 3.2, 2.1, 5.3};
```

---

a) Write a function that returns the total amount of expected rain in a given interval. The total number of statements must not exceed the given number of lines.

```
// PRE: [begin, end) is a valid range
// POST: returns the sum of the elements in [begin, end)
float rainfall(const float* begin, const float* end)
{
                                                    // 1
                                                    // 2
                                                    // 3
                                                    // 4
}
```

---

b) When you go on vacation, you want to have the least amout of rain as possible. Write a function that finds the best interval of `k` days (i.e. the interval of length `k` with the least amount of rain) and returns the expected amount of rain in this inverval. For instance for $k = 2$ and the array `rain` as above, the answer should be `5.3`. You can use the function from part a). The total number of statements must not exceed the given number of lines.

```
// PRE: [begin, end) is a valid range of size at least k
// POST: returns the smallest sum of k consecutive elements
//       in [begin, end)
float best_holiday(const float* begin, const float* end, int k)
{
                                                    // 1
                                                    // 2
                                                    // 3
                                                    // 4
                                                    // 5
                                                    // 6
                                                    // 7
}
```

Assignment 4. (8 / 4 Points) Consider the Lindenmayer system $\mathcal{L} = (\Sigma, P, s)$ with alphabet $\Sigma = \{F, +, -\}$, initial word $s = F$, and the productions

σ	$\mapsto$	$P(\sigma)$
F	$\mapsto$	$+F+F$
$+$	$\mapsto$	$+$
$-$	$\mapsto$	$-$

Such a system generates an infinite sequence of words $s = w_0, w_1, \ldots$ as follows. To get the next word w_i from the previous word w_{i-1}, we simply substitute all symbols in w_{i-1} by their productions.

In our example, this yields

$$\begin{aligned} w_0 &= F \\ w_1 &= +F+F \\ w_2 &= ++F+F++F+F \end{aligned}$$

---

a) Provide a *recursive* definition of the function $t\colon \mathbb{N} \to \mathbb{N}$ with $t(n) = |w_n|$. This means, $t(n)$ is the length of the word w_n. You don't have to argue why your definition is correct. **Hint**: With your definition, you should get $t(0) = 1, t(1) = 4, t(2) = 10$.

---

b) Using a) (or any other way), compute the value $t(7) = |w_7|$. You don't have to argue why the value that you get is correct.

$t(7) =$

Assignment 5. (10 / 20 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_;
};
```

a) Implement a function that checks if the elements of a list are sorted in ascending order. For example the list with elements `|1 2 3|` is sorted in ascencing order, the list with elements `|3 1 2|` is not.

b) Implement a function that takes as input two sorted lists and writes the elements of both lists in inceasing order to the standard output (`std::cout`). That is, for the two lists `|2 4|` and `|1 3 5|` the output should be `1 2 3 4 5`.

You can assume that `iostream` is correctly included. However, you are not allowed to include any other functions or data containers form the standard library, for instance you can not use vectors.

```
// POST: returns true if and only if the elements of l are
//       sorted in ascending order
bool is_sorted(const List& l)
{









}

// PRE: the elements in l1 and l2 are sorted in ascending order
// POST: all the elements of both lists l1 and l2 are printed in
//       ascending order to std::cout
void output_merge(const List& l1, const List& l2)









}
```