ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecturer: Prof. B. Gärtner

# Informatik für Mathematiker und Physiker HS13

## Exercise Sheet 8

Submission deadline: 15:15 - Tuesday 12th November, 2013
Course URL: http://www.ti.inf.ethz.ch/ew/courses/Info1_13/

## Assignment 1 – Skript-Aufgaben 111/112 (4 points)

a) Find pre- and postconditions for the following two recursive functions f and g.

```
void f (const unsigned int n)
{
  if (n == 0) {
    std::cout << "*";
    return;
  }
  f(n-1);
  f(n-1);
}
```

```
unsigned int g (const unsigned int n,
                const unsigned int b)
{
  if (n == 1) return 0;
  return 1 + g (n / b, b);
}
```

b) Prove or disprove for the recursive functions g and h that they terminate for all possible arguments. In this theory exercise, overflow should not be taken into account, i.e. you should pretend that the value range of unsigned int is equal to $\mathbb{N}$.

```
unsigned int h (const unsigned int n, const unsigned int m)
{
  if (n == 0) return 0;
  return 1 + h ((n + m) / 2, 2 * m);
}
```

## Assignment 2 – Skript-Aufgabe 114 (4 points)

The binomial coefficients $\binom{n}{k}, n, k \in \mathbb{N}$ may be defined in various equivalent ways. For example,

$$\binom{n}{k} := \frac{n!}{k!(n-k)!},$$

or

$$\binom{n}{k} := \begin{cases} 0, & \text{if } n < k \\ 1, & \text{if } n = k \text{ or } k = 0 \\ \binom{n-1}{k} + \binom{n-1}{k-1}, & \text{if } n > k, k > 0 \end{cases},$$

or

$$\binom{n}{k} := \begin{cases} 0, & \text{if } n < k \\ 1, & \text{if } n \geq k, k = 0 \\ \frac{n}{k}\binom{n-1}{k-1} & \text{if } n \geq k, k > 0 \end{cases}$$

a) Which of the three variants is best suited for implementation, and why? Argue theoretically.

b) Write and test a function `binomial(unsigned int n, unsigned int k)` that calculates $\binom{n}{k}$ according to the way that you have decided on in part a).

# Assignment 3

a) Modify the program `read_array.cpp` in such way that the resulting program `find_array.cpp` defines and calls the following function:

```
typedef std::vector<int>::const_iterator Cit;

namespace ifm {
  // PRE:  [begin, end) is a valid range
  // POST: return value is an iterator it in [begin , end) such
  //       that *it == x, or the iterator end if no such
  //       iterator exists
  Cit find (Cit begin, Cit end, const int x);
}
```

b) Use a template to define a generic function `ifm:find` that does not only find integers in vectors of integers, but also doubles in vectors of doubles and strings in vectors of strings, etc.

Implement the following function and define the preconditions.

```
namespace ifm {
  // PRE:
  // POST: return value is an iterator it in [begin , end) such
  //       that *it == x, or the iterator end if no such
  //       iterator exists
  template <typename Iterator, typename T>
  Iterator find(Iterator begin, Iterator end, const T x);
}
```

Does your function also work on arrays of integers? And on sets of integers? Use the program `read_array_test.cpp` that you find on the course webpage to test your function!

# Assignment 4 – Skript-Aufgabe 115 (4 points)

In how many ways can you own CHF 1? Despite its somewhat philosophical appearance, the question is a mathematical one. Given some amount of money, in how many ways can you partition it using the available denominations (bank notes and coins)? Today's denominations in CHF are 1000, 200, 100, 50, 20, 10 (banknotes), 5, 2, 1, 0.50, 0.20, 0.10, 0.05 (coins). The amount of CHF 0.20, for example, can be owned in four ways (to get integers, let's switch to centimes): $(20), (10, 10), (10, 5, 5), (5, 5, 5, 5)$. The amount of CHF 0.04 can be owned in no way, while there is exactly one way to own CHF 0.00 (you cannot have $4$ centimes in your wallet, but you *can* have no money at all in your wallet).

a) Solve the problem for a given input amount, by writing the following function (all values to be understood as centimes).

```
// PRE: [begin, end) is a valid nonempty range that describes
//      a sequence of denominations d_1 > d_2 > ... > d_n > 0
// POST: return value is the number of ways to partition amount
//       using denominations from d_1, ..., d_n
unsigned int partitions (unsigned int amount,
                         const unsigned int* begin,
                         const unsigned int* end);
```

On the course webpage you find the program `partitions_test.cpp` where you can plug-in your code and use it to test your function. Use the program to determine in how many ways you can own CHF 1, and CHF 10.

b) **Challenge.** Modify your program such that it computes the solution *quickly* (say in less than one second) also for large amounts like CHF 50 or CHF 100.