OTE: Ohjelmointitekniikka
Programming Techniques

course homepage: `http://www.cs.uku.fi/~mnykanen/OTE/`

Week 50/2008

In these exercises, when you are asked to “develop an algorithm”, it means giving its pre- and postconditions, loop invariants and bounds in sufficient detail so that its correctness can be checked by others as well.

Exercise 1. It seems intuitively plausible that the following two GCL code patterns would mean the same thing. After all, the first might be the result of the “guard first” approach (lectures, Section 3.2.1) and the second of the “commands first” approach (lectures, Section 3.2.3) to the same programming problem.

if inside **do**	fused into **do**
do *outer guard* $\rightarrow$	
if *inner guard* 1 $\rightarrow$	**do** *outer guard* $\wedge$ *inner guard* 1 $\rightarrow$
command 1	*command* 1
[] *inner guard* 2 $\rightarrow$	[] *outer guard* $\wedge$ *inner guard* 2 $\rightarrow$
command 2	*command* 2
[] *inner guard* 3 $\rightarrow$	[] *outer guard* $\wedge$ *inner guard* 3 $\rightarrow$
command 3	*command* 3
[] $\vdots$	[] $\vdots$
[] *inner guard* k $\rightarrow$	[] *outer guard* $\wedge$ *inner guard* k $\rightarrow$
command k	*command* k
fi	**od**
od	

But do they really mean the same thing? Why?

Exercise 2. Consider the Welfare Crook example in the lectures (Section 3.3.3).

(a) The current version assumes that there is a solution. Change the code so that it works even when there is none.

(b) Explain how the code can be reused to find all the solutions.

Exercise 3. The well-known *Fibonacci sequence* is defined as follows:

$$\begin{aligned} \mathrm{fib}(0) &= 0 \\ \mathrm{fib}(1) &= 1 \\ \mathrm{fib}(m+2) &= \mathrm{fib}(m+1) + \mathrm{fib}(m). \end{aligned}$$

Develop an algorithm to compute $\mathrm{fib}(n)$ given $n \in \mathbb{N}$.

Exercise 4. The *convolution* of two number arrays a and b is the array c such that every $c[k]$ is the sum of all products $a[i] \cdot b[k-i]$.

(a) What is the index range of c in terms of the index ranges of a and b?

(b) What is the smallest and largest value for index i when computing $c[k]$?

(c) Develop an algorithm for computing c given a and b as inputs.

Exercise 5. Most practical programming languages do not have the full mathematical number types such as $\mathbb{N}$. Instead, they usually impose some implementation-dependent limits such as "32-bit `unsigned int`egers". To get around such limitations, there are arbitrary precision arithmetic libraries, which provide arbitrarily long numbers, up to the memory limit of the computer. (One such public domain library is the GNU Multiple Precision Library `gmp` available at `http://gmplib.org/`.)

One simple arbitrary precision representation for $n \in \mathbb{N}$ is a zero-based array a of Booleans where element $a[i]$ tells whether or not bit i is set in the bit pattern of n. For example, the bit pattern of $n = 13$ is $\mathbf{1101}_2$, so a could be any array of the form

0	1	2	3	. . . upper(a)
TRUE	FALSE	TRUE	TRUE	these (if any) are all FALSE

which is at least long enough to store all its **1**-bits.

(a) Give a mathematical expression which calculates the number represented by such an array.

(b) Develop an algorithm for adding two numbers represented by such arrays into a third such array. Your algorithm must *not* manipulate the represented numbers, but instead work within their array representations. (These numbers would namely overflow the machine registers when executing the library code.)

(HINT: The simplest algorithm is the one you learned already on first grade at school for decimal numbers. . .)