Department of Mathematics
Dr. Ulf Lorenz
Christian Brandenburg

TECHNISCHE UNIVERSITÄT DARMSTADT

winter term 2008/2009
02/02/2009

Introduction to Mathematical Software

7th Exercise Sheet

Exercise 1 (Fibonacci Numbers)

(a) Write a routine `int fib(int n)` that computes the $n-th$ Fibonacci number by recursively calling itself.

(b) Now write an iterative routine for computing the $n-th$ Fibonacci using a `for`-loop.

(c) Test both algorithms with $n = 42$. What do you observe? Does this mean recursion as such is slow?

(d) What happens if you choose $n = 50$?

Exercise 2 (Matrix Multiplication)

Write a program to multiply two matrices.
More specifically, you are given an $l \times m$-matrix A and an $m \times n$-matrix B. Compute the $l \times n$-matrix C with $C = A \cdot B$.

Hints:

- use the `#define` preprocessor directive to set the dimensions of the matrices, e.g.
```
#define L 3
#define M 2
#define N 4
```
`L, M, N` can then be used like ordinary variables of type `int`.
- write a subfunction
```
void MatMatMult (double A[L][M], double B[M][N], double C[L][N])
```
that computes AB and stores the result in C.

Exercise 3 (Efficient Fibonacci)

An efficient way for computing large Fibonacci numbers is the matrix multiplication algorithm. This algorithm is based on the following observations:

The Fibonacci sequence can be described as a 2-dimensional system of difference equations

$$\begin{pmatrix} F_{n+2} \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n+1} \\ F_n \end{pmatrix}$$

where F_n denotes the n-th Fibonacci number.

This yields the following closed form for the computation of Fibonacci numbers

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix},$$

which is the basis of the algorithm.

From the closed form it follows that the n-th Fibonacci number is given as the upper left element of the matrix

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1},$$

so the problem of computing a Fibonacci number can be reduced to matrix multiplication. For an implementation, it is now important to perform the matrix multiplications efficiently. However, we only have to compute matrix powers, which a computer can do efficiently by using the binary representation of a natural number.

We will explain the procedure for the simpler case of raising a real number to an integral power. Say we want to compute a^d for $a \in \mathbb{R}, d \in \mathbb{N}$. The binary representation of d is given by a finite sequence $\{b_i\}_{i=0}^n, b_i \in \{0, 1\}$ for an implementation dependent $n \in \mathbb{N}$. Now,

$$a^d = \prod_{i=0}^{n} a^{b_i \cdot 2^i}.$$

So, instead of multiplying a d-times by itself, we use the following procedure: starting with $r = 1$, for i from 0 to n, we multiply r with $a^{b_i \cdot 2^i}$. In each iteration, the $a^{(2^i)}$ is computed as $(a^{(2^{i-1})})^2$

The procedure is summerized in the following algorithm:

Algorithm 1 Computation of the n-th Fibonacci number

input: natural number n
set $d = n - 1$, $A = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$, $M = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$
while $d > 0$ **do**
  if rightmost-bit of d is 1 **then**
    $M = M \cdot A$
  end if
  $A = A^2$
  shift d to the right by 1 bit
end while
output: $F_n = M_{00}$

- Implement this algorithm.

Hints:

- use `typedef unsigned long long int Integer` to be able to compute larger Fibonacci numbers than in exercise 1 (you should be able to compute F_{200} in the computer pool room).
- write a function `void MatMatMult (Integer M[2][2], Integer A[2][2])` for computing the product of two 2×2 matrices. The product should be written to `M`.
- write a function `void square (Integer A[2][2])` to compute the square of a matrix. The result should be returned in `A`.
- checking whether the rightmost-bit of `d` is 1 can be done by `(d & 1)` (`&` is the bitwise and operator).
- shifting `d` to the right by 1 bit can be done by `d = d >> 1;` (`>>` is the shift right operator).
- for `unsigned long long int`, use `%llu` in `printf` statements.