Programování I

Martin Pergel, perm@kam.mff.cuni.cz

October 10, 2014

Information

- This semester finishes by a credit, exam is on the next semester.
- Requirements for being credited:
  - Test on linked lists,
  - program (written as a homework - including documentation),
  - active participance at assignments (will be justified on assignments).

Goals of this course:

- Programming in Pascal and later C#,
- algorithms
- and related theory.

Individual parts shall be parallelized.

Why Pascal:

- Myth: Pascal is obsolete!
- Reality: Pascal is well-tested.
- Java, C#, C programming language... – nice but complicated.
- Pascal: Disadvantage: Age Advantage: Simplicity.
- We: Borland Pascal (Free Pascal, GNU Pascal, Delphi),
- since Christmas: C# in Visual Studio.

Organizational affairs

- CodEx machine (alias Code Examiner and account on it)
- Account in computer lab at Mala Strana.
- Warning: Programming is a time-demanding skill!
- Literature: Kurt Mehlhorn: Algorithms and Data Structures (available on-line, individual chapters).
  Donald E. Knuth: The Art of Computer Programming.
  Niklaus Wirth: Algorithms + Data Structures = Programs [older book, algorithms seem more interesting than the language, available on-line]
- Any questions? [If so, pose them as soon as possible!]

Algoritms

Definition

Algorithms are rigorously defined and willingly created methods.

- An **algorithm** is a way (method) how to solve a particular problem.
- Realization of an algorithms produces expected output from a given input.
- Algorithm consists of individual *steps* called commands (*e.g.*, natural numbers addition).
- Any correct algorithm must be:
  - **finite** (*i.e.*, for any input it finishes after finitely many steps)
  - and **partially correct** (*i.e.*, whenever the algorithm finishes, it produces a correct output (correct answer for a given problem)).

Ways how to describe algorithms

Karel:

```
krok
krok
vlevobok
krok
```

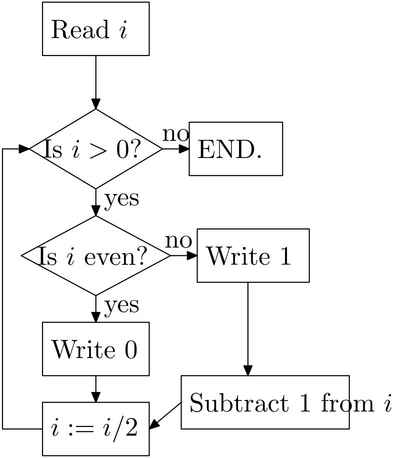
C progr. language:

```
while(i)
{ if(i%2) printf(1);
  else printf(0);
  i/=2;
}
```

Text:

```
Read i.
While i > 0:
   If i odd then write "1"
   else write "0"
   divide i by 2.
```

Program-flow-diagram:



Greatest common divisor

Ways how to find it:

- Find prime-decompositions and "compare",
- Euclid's-algorithm.

Observation: Given $a \geq b$ natural numbers dividable by a (also natural) number k then $a - b$ is also dividable by k.

Euclid's algorithm: version 1 (using subtraction)

```
read a and b.                            read(a); read(b);
1:  if b > a then swap values of a and b.
If b is zero then write a       if b=0 then write(a);
and end the algorithm.
Subtract b from a.                           a:=a-b;
GOTO 1:
```

Euclid's algoritmus: version 2 (using modulo)

read a and b.
1: if $b > a$ then swap values of a and b.
If b is zero then write a
and end the algorithm.
Let a be a division remainder of a and b
($a := a$ mod(b)).
GOTO 1:

Magic squares of an odd order

Easy algorithm with a hard proof of correctness:

6	1	8
7	5	3
2	9	4

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

1. Start in the middle of the top-most row.
2. Step one cell to the left and upwards and fill-in the numbers in increasing ordering
3. When the cell already contains a number, return one cell back and step one step downwards instead.

Prime-decomposition

- Ideas?
- Naive approach: Try one prime-number i after another from 2 to n and try to divide n/i.
- Less naive algorithm: Let $m := n$. Try only primes i from 2 to $\sqrt{m}$.It means: Let $i := 2$. When i is a factor of m (*i.e.*, m/i is an integer), let $m := m/i$., otherwise (else) increase i by one (*i.e.*, $i := i + 1$.
- Even less naive algorithm: Instead of primes try all the numbers. natural.
  Why does it work?

Towards the Sieve of Eratosthenes

How to generate all prime numbers less than n?

- Naive algorithm: Generate and test.
- Less naive approach: Generate and try to divide only by already generated primes.
- Eratosthenes: Generate all natural numbers $2 \dots n$. For $i \in \{2 \dots \sqrt{n}\}$ do the following: If i is a prime (*i.e.*, is not crossed-out), cross out all its nontrivial multiples (*i.e.*, $j := 2$, while $ij < n$ do cross-out number ij, increase j by one).