

# Problémy, algoritmy a programovací jazyky

IB110

# Problém

čo je problém?

---

# Problém

čo je problém?

výpočtový problém

- (nekonečná) množina vstupných inštancií  
*prípustné vstupy*
- špecifikácia požadovaných výstupov  
funkcia vstupných inštancií

# Riešenie problému

**Metóda** riešenia problému popisuje efektívnu cestu, ktorá vedie k nájdeniu požadovaných výstupov. Popis pozostáva z postupnosti inštrukcií, ktoré môže ktokoľvek realizovať.

existencia metódy riešenia problému znamená možnosť vypočítať riešenie *automaticky*

v danom kontexte hovoríme o **algoritme** pre riešenie problému

# Algoritmus — historický pohľad I

- koniec 19. storočia, priemyslová revolúcia, kauzálne chápanie sveta, zákony umožňujúce úplné pochopenie sveta
- program Davida Hilberta
  - (a) celá matematika sa dá vybudovať z konečnej sady vhodných axióm
  - (b) matematika vytvorená týmto spôsobom je kompletná v tom zmysle, že každé tvrdenie vyjadriteľné v jazyku matematiky sa dá dokázať alebo vyvrátiť v tejto teórii (z danej sady axiómov)
  - (c) existuje *metóda* pre dokázanie resp. vyvrátenie každého tvrdenia.
- kľúčový pojem metódy

# Algoritmus — historický pohľad II

- Kurt Gödel (1931) dokázal, že
  - (a) neexistuje žiadna úplná (rozumná) matematická teória; v každej korektnej a dostatočne veľkej teórii je možné formulovať tvrdenia, ktoré nie je možné vnútri tejto teórie dokázať. Aby bolo možné dokázať správnosť týchto tvrdení, je nutné k teórii pridať nové axiomy a vybudovať tak novú, väčšiu teóriu
  - (b) neexistuje metóda (algoritmus) pre dokazovanie matematických teorém
- pre svoj dôkaz potreboval presnú definíciu pojmu metóda (algoritmus) (*nie je možné dokázať neexistenciu algoritmu bez rigorózneho definície toho čo je a čo nie je algoritmus*)
- prvá formálna definícia pojmu algoritmu: Alan Turing (1936)
- ďalšie definície a ich ekvivalencia

# Základné otázky

Existujú problémy, ktoré nie sú riešiteľné automaticky (algoritmicky)?

Ak áno, ktoré problémy sú algoritmicky riešiteľné a ktoré nie?

# Programovacie jazyky I

- algoritmy musia byť zapísané jednoznačne a formálne
- programovací jazyk – jazyk pre špecifikáciu algoritmov
- program – zápis algoritmu v programovacom jazyku
- strojový kód vs. programovací jazyk vyššej úrovne



# Programovacie jazyky II

- ako prezentovať algoritmus reálnemu počítaču?
- ako “prinútiť” počítač, aby realizoval výpočet tak, ako sme to zamýšľali?
- programátori sú ľudia, uvažujú abstraktne a vyjadrujú sa pomocou slov a symbolov
- počítače sú stroje, ktoré dokážu realizovať veľmi jednoduché úlohy
- programovacie jazyky pomáhajú ľuďom komunikovať s počítačmi

# Vývojové diagramy

# Programovací jazyk vyššej úrovne

## jazyk pre popis algoritmov

### základné stavebné kamene

- riadiace štruktúry
- dátové štruktúry

# Riadiace štruktúry

- postupnosť príkazov
- podmienené vetvenie
- ohraničená iterácia
- podmienená iterácia

kombinácia riadiacich štruktúr - vnorenie

# Dátové typy

- premenné
- vektory, zoznamy
- polia, tabuľky
- zásobníky a fronty
- stromy a hierarchie
- ...

# Vlastnosti programovacích jazykov

- presná syntax
- presná sémantika

# Syntax

- nejednoznačnosti; presný význam pre pojmy, ktoré sa používajú v bežnej komunikácii
- Porovnanie “=” a priradenie “:=”

```
Java   if (x==y) z = 3; else z = 4;
FORTRAN IF (X .EQ. Y)
        THEN Z = 3
        ELSE Z = 4 END IF
Pascal if x = y then z := 3 else z:= 4
```

# Syntax - príklad

- hypotetický programovací jazyk PL

## Príklad

```
1 input $N$ ;  
2  $X := 0$ ;  
3 for  $Y$  from 1 to  $N$  do  
4      $X := X + Y$   
5 od  
6 output $X$ .
```



# Definícia syntaxe - syntaktické diagramy

# Definícia syntaxe — BNF

$\langle \text{príkaz} \rangle ::= \langle \text{for príkaz} \rangle \mid \langle \text{prirad'ovací príkaz} \rangle \mid \dots$   
 $\langle \text{for príkaz} \rangle ::= \langle \text{premenná} \rangle \mathbf{from} \langle \text{hodnota} \rangle \mathbf{to} \langle \text{hodnota} \rangle$   
...

BNF umožňuje generovať (syntakticky správne) programy

# Kontrola syntaktických chýb

- program sa nedá vygenerovať použitím BNF
- automatizovaný proces

# Sémantika

**for  $Y$  from 1 to  $N$  do**

- význam — odčítanie, príkaz pre tlačiareň, *dnes je pekný deň ???*
- význam podľa použitých slov ???
- čo ak  $N = 3.14$  ???
- presná sémantika je nevyhnutná !!!
- manuál (dokumentácia) ako definícia sémantiky ???

# Sémantika - príklad

## Príklad

procedúra  $P(s \text{ parametron } V)$

(1) **call**  $V(s \text{ parametrom } V)$ , výsledok ulož do  $X$

(2) **if**  $X = 1$  **then return** 0; **else return** 1

- procedúra, ktorá má ako svoj parameter názov procedúry
- syntaktická korektnosť
- **call**  $P(s \text{ parametrom } P)$  – aká je návratová hodnota?
- sémantika musí dať jednoznačnú odpoveď

# Výpočet programu

- od programu v jazyku vyššej úrovne k manipulácii s bitmi
- program je postupne transformovaný na strojovú úroveň
- transformácia
  - kompilácia
  - interpretácia
  - kombinovaný prístup

# Kompilácia

- program v jazyku vyššej úrovne je preložený do jazyka nižšej úrovne (jazyk symbolických adries)

**for** *Y* **from** 1 **to** *N* **do**

telo cyklu

**od**

**MOVE** 0, *Y* *do registra Y ulož 0*

*LOOP:* **CMP** *N*, *Y* *porovnaj hodnoty uložené v N a Y*

**JEQ** *REST* *ak rovnosť, tak pokračuj príkazom REST*

**ADD** 1, *Y* *pripočítaj 1 k Y*

telo cyklu

**JMP** *LOOP*

- prekladač

# Kompilácia - pokr.

- preklad z jazyka symbolických adries do strojového kódu
- assembler

Optimalizácia kódu počas kompilácie



# Interpretácia

- interpret postupuje príkaz po príkaze
- každý príkaz je okamžite preložený do strojového kódu a vykonaný

Možnosť lepšie sledovať výpočet.

Typicky jednoduchá tvorba interpretu.

Nemožnosť optimalizácii.

# Programovacie esperanto

- prečo rôzne programovacie jazyky?
- programovací jazyk poskytuje programátorovi istú úroveň abstrakcie
- potreba nových typov abstrakcií
  - vývoj hardwaru (*paralelné počítače a programovanie vo vláknach*)
  - nové aplikačné oblasti (*multimediálne aplikácie*)
- paradigmata - kategorizácia existujúcich programovacích jazykov

# Imperatívne programovanie

- prístup blízky ľudskému uvažovaniu
- imperatívne programovanie popisuje výpočet pomocou postupnosti príkazov a určuje presný postup, ako daný algoritmický problém riešiť
- pamäť počítača je súborom pamäťových miest organizovaných do rôznych dátových štruktúr
- program buduje, prechádza a modifikuje dátové štruktúry tak, že načíta dáta a mení hodnoty uložené v pamäti
- príkazy, v závislosti na vyhodnotení podmienok, menia stav premenných
- historicky najstaršie imperatívne programovacie jazyky: strojové jazyky
- FORTRAN, ALGOL, COBOL, BASIC, Pascal, C, Ada
- základné typy príkazov - priradenie, cykly, príkazy vetvenia

# Funkcionálne programovanie

- výpočtom funkcionálneho programu je postupnosť vzájomne ekvivalentných výrazov, ktoré sa postupne zjednodušujú
- výsledkom výpočtu je výraz v normálnej forme, ktorý sa nedá ďalej zjednodušiť
- program je chápaný ako jedna funkcia, ktorá obsahuje vstupné parametry. Výpočet je vyhodnotením tejto funkcie.
- imperatívny prístup: ako sa má vypočítať  
funkcionálny prístup: čo sa má vypočítať
- Erlang, Haskell, Lisp, ML, Oz, Scheme

# Logické programovanie

- postavené na použití matematickej logiky
- logický program je tvorený sadou pravidiel a jednoduchých logických tvrdení
- dotaz sa rieši prehľadávaním množiny pravidiel. Hľadá sa dôkaz, ktorý poskytuje odpoveď na zadaný dotaz

# Objektovo orientované programovanie

- pamäť počítača pozostáva z objektov
- s každým objektom sú asociované operácie, ktoré môže objekt poskytovať (prístupné ako metódy volania)
- program je súborom objektov, ktoré si posielajú správy obsahujúce požiadavky na realizáciu operácií a odpovede
- Smalltalk, Java, C++, Python, Ruby. Lisp