Techniky řešení problémů

Jitka Kreslíková, Aleš Smrčka

2021

Fakulta informačních technologií
Vysoké učení technické v Brně

IZP – Základy programování

Techniky řešení problémů

- Algoritmy a řešení problémů
- Úloha strukturalizace v programování

Algoritmy a řešení problémů

- Strategie řešení problémů.
- Algoritmický problém.
- Datová abstrakce.

Algoritmy a řešení problémů

Základní pojmy:

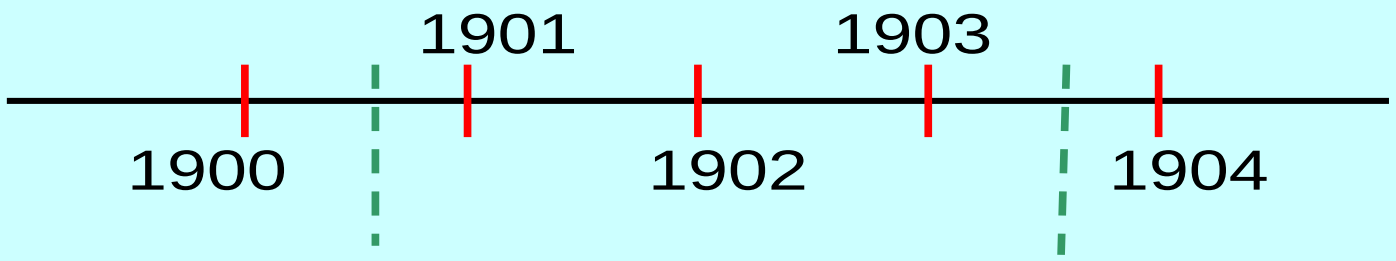
- **Informace** je poznatek (týkající se jakýchkoliv objektů, např. fakt, událostí, věcí, procesů nebo myšlenek, včetně pojmů), který má v daném kontextu specifický význam.
- **Data** jsou opakovaně interpretovatelná formalizovaná podoba informace vhodná pro komunikaci, vyhodnocování nebo zpracování.
- **Program** je jednoznačný předpis, podle kterého je počítač schopen provádět výpočty nějakého algoritmu.
- **Procesor** je prvek, kterému je svěřeno vykonávání algoritmu.

Strategie řešení problémů

- **dekompozice** (metoda shora dolů)
  - znamená chápat složité problémy jako kolekci jednodušších problémů
  - najít ve složitém problému takové hierarchické uspořádání, které umožní zapsat složité akce pomocí akcí jednodušších (ty mohou být stejným způsobem redukovány na akce ještě jednodušší)
  - tímto způsobem postupujeme tak dlouho, až dosáhneme natolik jednoduchých akcí, že máme prostředky k jejich řešení
  - výsledkem těchto úvah by mělo být „rozkouskování" problému na dílčí, relativně samostatné „podproblémy"

Strategie řešení problémů

Příklad:
Zadání: Vytvořte program, který vypočte počet dnů mezi dvěma kalendářními daty. Program musí zohlednit přestupné roky.



Vyřeš daný problém

1. Zjisti dvě data
2. Proveď výpočet
3. Zobraz výsledek

Strategie řešení problémů

1. Zjisti dvě data
   - Načti data ze souboru
     - otevři soubor
     - otestuj úspěšnost otevření souboru
     - čti řádku ze souboru
     - uzavři soubor
   - Extrahuj datum
   - Ověř správnost dat
     - zjisti **přestupný rok**
     - ověř měsíc
     - ověř den

Úkol:
Proveďte dekompozici do nižší úrovně

Čínský kalendář, Egyptský kalendář, Juliánský kalendář, Gregoriánský kalendář, typy kalendářů

Strategie řešení problémů

2. Proveď výpočet
   - vypočti počet celých roků mezi zadanými daty
   - vypočti počet **přestupných roků** mezi zadanými daty
   - vypočti počet dnů ode dne prvního data do konce roku prvního data
   - vypočti počet dnů od začátku roku druhého data do dne druhého data
   - vypočti celkový počet dnů mezi dvěma daty
3. Zobraz výsledek

Strategie řešení problémů

- **abstrakce** (metoda zdola nahoru)
  - z detailnějších částí se postupně vytvářejí obecnější a abstraktnější konstrukce
  - koncepční zjednodušení složitého problému ignorováním detailů
  - pojmenováním akcí a dočasným ignorováním detailů je možné celý složitý problém řešit po částech

Strategie řešení problémů

Příklad:

1. Zjisti dvě data
   - Načti data ze souboru
     - otevři soubor
     - ověř úspěšnost otevření souboru
     - čti řádku ze souboru
     - uzavři soubor
   - Extrahuj datum
   - Ověř správnost dat
     - zjisti **přestupný rok**
     - ověř měsíc
     - ověř den

Strategie řešení problémů

2. Proveď výpočet
   - vypočti počet celých roků mezi zadanými daty
   - vypočti počet **přestupných roků** mezi zadanými daty
   - vypočti počet dnů ode dne prvního data do konce roku prvního data
   - vypočti počet dnů od začátku roku druhého data do dne druhého data
   - vypočti celkový počet dnů mezi dvěma daty
3. Zobraz výsledek

Strategie řešení problémů

1. Zjisti dvě data
2. Proveď výpočet
3. Zobraz výsledek

Vyřeš daný problém

Princip použití dekompozice a abstrakce: princip modulárního programování.

programovací jazyky jej umožňují prostřednictvím funkcí, podprogramů, programových modulů

Algoritmický problém - úvaha

- při řešení úloh na (počítači) nás zajímají údaje (informace), respektive jejich interpretace
- úlohu řešíme tehdy, potřebujeme-li **nové** informace
- informace, na základě kterých úlohu řešíme, nazýváme **vstupní**, získané **výstupní**
- z hlediska řešení úloh reprezentujeme informace obvykle údaji (daty), hovoříme o **vstupních a výstupních údajích (datech)**

Algoritmický problém - úvaha

- řešit úlohu znamená transformovat vstupní údaje na výstupní
- o tom, jak probíhá řešení úlohy, jak probíhá transformace vstupních údajů na výstupní, rozhoduje v první řadě realizátor
- pokud je realizátor z hlediska dané úlohy dostatečně schopný, může realizovat transformaci přímo, v jednom kroku

Poznámka: jde o úlohy, které patří do základního repertoáru realizátora (např.: malá násobilka pro žáky základních škol, akce shodné s instrukcemi daného počítače)

Algoritmický problém - úvaha

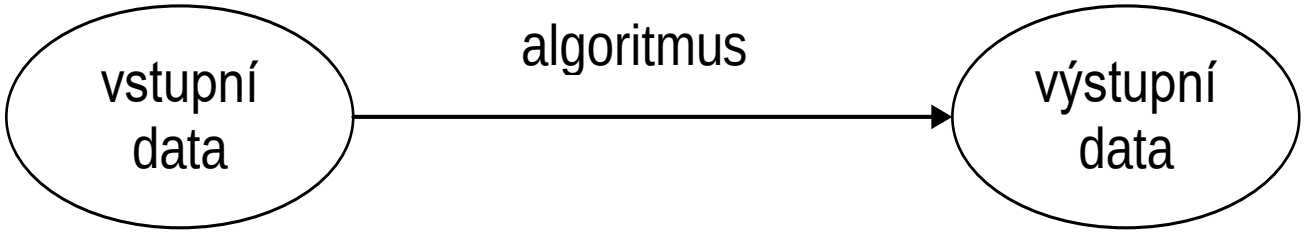
- většinou se však transformace nerealizuje přímo, ale vhodnou kompozicí primitivních činností (operací), které je realizátor schopný vykonat (u počítače - repertoár instrukcí)
- klíčovou otázkou řešení je tedy nalezení takové kompozice primitivních činností realizátora, která zabezpečí požadovanou transformaci vstupních údajů na výstupní
- výstupní údaje nejsou dané explicitně, ale implicitně, ve tvaru určitých podmínek, které musí výstupní údaje splňovat - budeme je nazývat **výstupní podmínky**

Algoritmický problém

- stejně tak musí splňovat určité podmínky i vstupní údaje, těmto budeme říkat **vstupní podmínky**
- často se vstupní podmínky shrnují do jedné podmínky, potom hovoříme o vstupní podmínce a analogicky o výstupní podmínce
- vstupní a výstupní podmínkou charakterizujeme daný problém, který potřebujeme řešit - specifikujeme to, co je třeba řešit
- na zápis vstupních a výstupních podmínek se kladou určité požadavky jako je jasnost a jednoznačnost

Algoritmický problém

Příklad:



např.:
2 celá čísla a,b
vstupní podmínka: a,b > 0

NSD (a,b)
největší společný dělitel

Úkolem programátora je najít algoritmus, který vstupní data (s respektováním vstupních podmínek) převede na data výstupní (splňující výstupní podmínky).

Algoritmický problém

Příklad:

- problém nalezení kořenů kvadratické rovnice $ax^2 + bx + c = 0$ s reálnými koeficienty můžeme charakterizovat:

Vstupní podmínka:

$$(a \in R) \cap (b \in R) \cap (c \in R)$$

a,b,c - vstupní proměnné
R - množina reálných čísel

Výstupní podmínka:

x1,x2 - hledané kořeny
výstupní proměnné

$$(x_1 \in R) \cap (x_2 \in R) \cap (ax_1^2 + bx_1 + c = 0) \cap (ax_2^2 + bx_2 + c = 0)$$

Algoritmický problém

- problém charakterizovaný vstupními a výstupními proměnnými, vstupní a výstupní podmínkou nazýváme **algoritmický problém**
- všeobecná pravidla určující postupnou transformaci vstupních údajů na výstupní nazýváme **algoritmus**
- algoritmus zapisujeme jako posloupnost elementárních kroků
- základní činností, kterou předpokládáme jako implicitní je, že jednotlivé kroky algoritmu se vykonávají postupně za sebou, jak jsou napsány, pokud není explicitně dáno jiné pořadí (skok)

Algoritmický problém

- každý krok algoritmu přispívá svojí činností do celkové transformace a současně rozhoduje o další činnosti algoritmu
- říkáme, že krok, který se právě realizuje, má řízení a po ukončení ho odevzdá dalšímu kroku, a to explicitně nebo implicitně
- označení program používáme pro algoritmy, které jsou formulované tak, že je může vykonat určitý typ procesoru
- program se skládá z množiny příkazů

Algoritmický problém

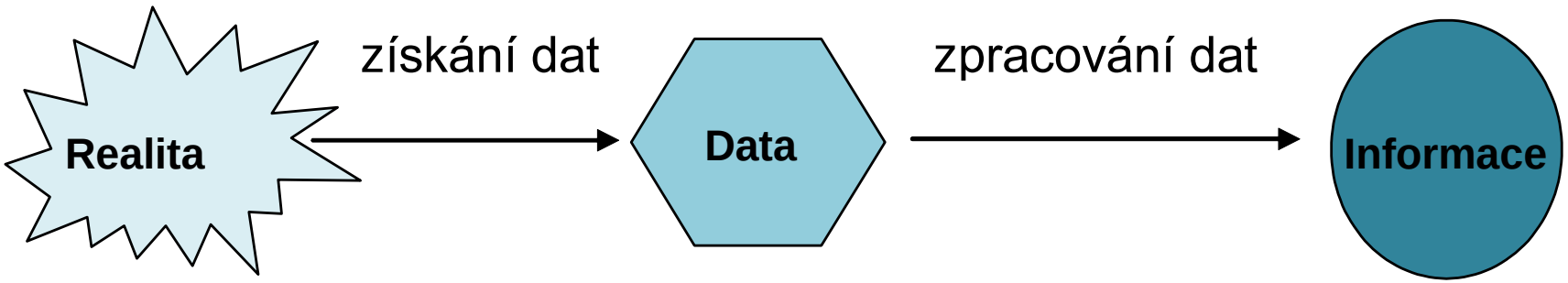
- program počítače musí vyhovovat do nejmenších detailů pravidlům nějakého programovacího jazyka
- pořadí příkazů v textu programu nemusí odpovídat časovému pořadí vykonávání odpovídajících akcí
- procesor - výkonná jednotka, která řídí akce podle příkazů, proces podle programu

Wirthův slogan [WiNi]

datové struktury + algoritmy = programy

Datová abstrakce

- datová abstrakce je proces získání popisu reality pomocí údajů (dat)
- s daty lze manipulovat bez znalosti jejich vnitřní implementace



Datová abstrakce

Abstrakce reality má 4 úrovně:

1. realita
2. abstrakce reálného světa, která zahrnuje jen ty vlastnosti objektů reality, které jsou pro řešení daného problému důležité
3. datová struktura - reprezentace vlastností objektů reality
4. reprezentace datové struktury na paměťovém médiu počítače

Datová abstrakce

Příklad: úrovně reprezentace údaje.
Je dán problém reprezentovat polohu objektu ve dvojrozměrném prostoru.

1. dvojice reálných čísel buď v kartézské nebo polární soustavě — ovlivněno vlastním problémem
2. zobrazení v pohyblivé řádové čárce, kde každé číslo sestává ze dvou čísel reprezentujících mantisu a exponent
3. číslo bude reprezentováno pomocí dvojkové soustavy
4. fyzikální princip reprezentace dvojkových číslic 0 a 1 — použitý nástroj a jeho fyzikální vlastnosti

Shrnutí

- konstrukci programů lze logicky rozdělit na dvě části:
  - specifikaci dat
  - specifikaci akcí
- data popisujeme datovými strukturami, algoritmy strukturami řídicími
- složitý algoritmus s jednoduchou reprezentací údajů versus jednodušší algoritmus s komplikovanějšími datovými strukturami
- požadavek moderního programovacího stylu je umění nalézt vhodnou reprezentaci dat pro řešenou úlohu

Shrnutí

- promyšlená datová struktura nám může ušetřit neuvěřitelné množství práce s algoritmy
- nevhodná reprezentace dat udělá z triviálního zadání neřešitelný úkol

!!! Dvakrát hledej vhodné datové struktury, jednou programuj algoritmy. !!!

Úloha strukturalizace v programování

- Programovací jazyk a struktura programu.
- Von Neumannův stroj – jazyky nižší úrovně, jazyky vyšší úrovně.
- Vztah stroje, programovacího jazyka a programu.

Programovací jazyk a struktura programu

- programovací jazyky původně vznikly pouze jako pomůcka pro urychlení psaní počítačového kódu
- rostoucí komplexnost programů přinesla problém těžce odhalitelných chyb v programech
- vznikl problém spolehlivosti programů
- možné řešení --» zdokonalení testování programů

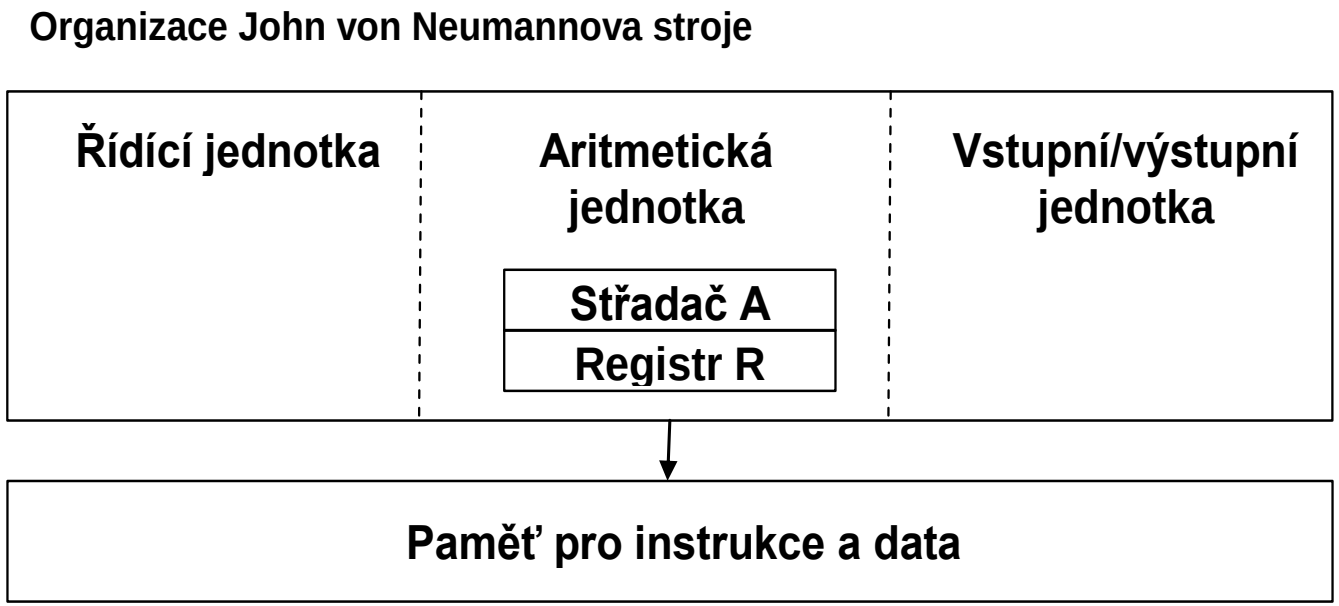
Programovací jazyk a struktura programu

E. W. Dijkstra v knize „Structured Programming” :
Testováním lze dosáhnout korektnosti programu pouze tehdy, pokud se zaměříme na jeho vnitřní strukturu.

- vnitřní struktura programu je velmi úzce spojena s použitým programovacím jazykem
- programovací jazyky nám umožňují (nebo dokonce nutí či naopak zabraňují) vytvářet programy se strukturou, odpovídající některému z paradigmat
- každý z nově vytvářených programovacích jazyků se snaží lépe nebo alespoň „jinak” podporovat jedno nebo kombinaci několika programovacích paradigmat

Von Neumannův stroj – jazyky nižší úrovně, jazyky vyšší úrovně

Koncepce počítačů (1947)



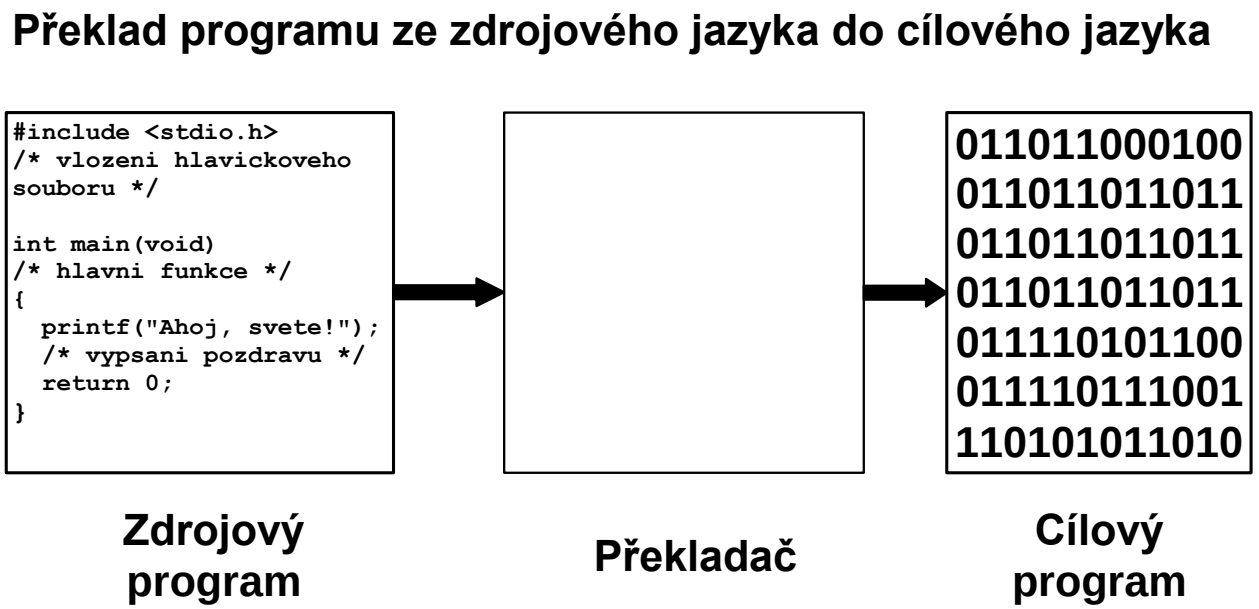
Jazyky nižší/vyšší úrovně

- jazyky nízké úrovně
  - strojový kód
  - jazyk symbolických adres
- jazyky vyšší úrovně
  - přístup kompilační
  - přístup interpretační

strojový kód	**assembler**	**vyšší progr. jazyk**
0000001010111001010	LOAD I	
0000001011111001000	ADD J	k = i + j
0000001100111010100 0	STORE K	

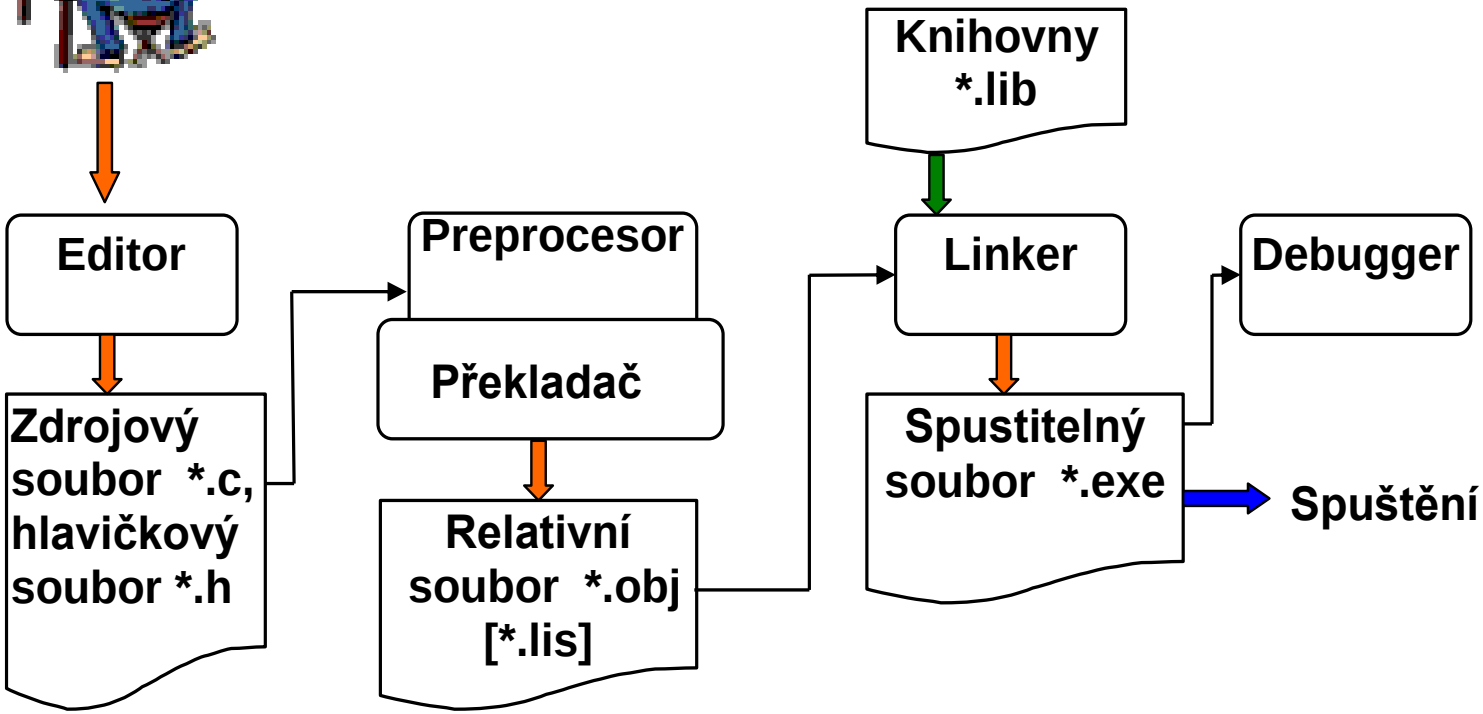
Jazyky nižší/vyšší úrovně

- **kompilátor** je překladač, který přeloží celý kód ze zdrojové formy do jazyku stroje najednou



Von Neumannův stroj – jazyky nižší úrovně, jazyky vyšší úrovně

Překlad programu ze zdrojového jazyka do cílového jazyka detailněji



Von Neumannův stroj – jazyky nižší úrovně, jazyky vyšší úrovně

- **interpret** (Awk, Perl) je překladač, který neprovádí přímý překlad do jazyka stroje
  - zdrojový program je těmito překladači postupně interpretován a cílový program jako celek nevzniká
  - překlad tak probíhá vlastně souběžně s během programu
  - výhoda - kód je možno za běhu programu modifikovat
  - nevýhody
    - překladače jsou pomalejší
    - náročnější na paměť
    - program není schopen fungovat bez přítomnosti překladače

Vztah stroje, jazyka a programu

Mezi jazykem stroje a vyššími programovacími jazyky existuje spolupráce, například:

- téměř každý procesor obsahuje instrukce pro porovnávání číselných hodnot
- jazyk C (a samozřejmě většina ostatních) poskytuje operátory porovnávání, které fungují nad čísly různých typů (int, float), ale také nad znaky
- v jazyce C (ale i v jiných) lze vytvořit funkce, které budou provádět porovnávání nad abstraktnějšími strukturami (textové řetězce, seznamy, ...) použitím základních porovnávacích operátorů, které poskytuje sám jazyk.

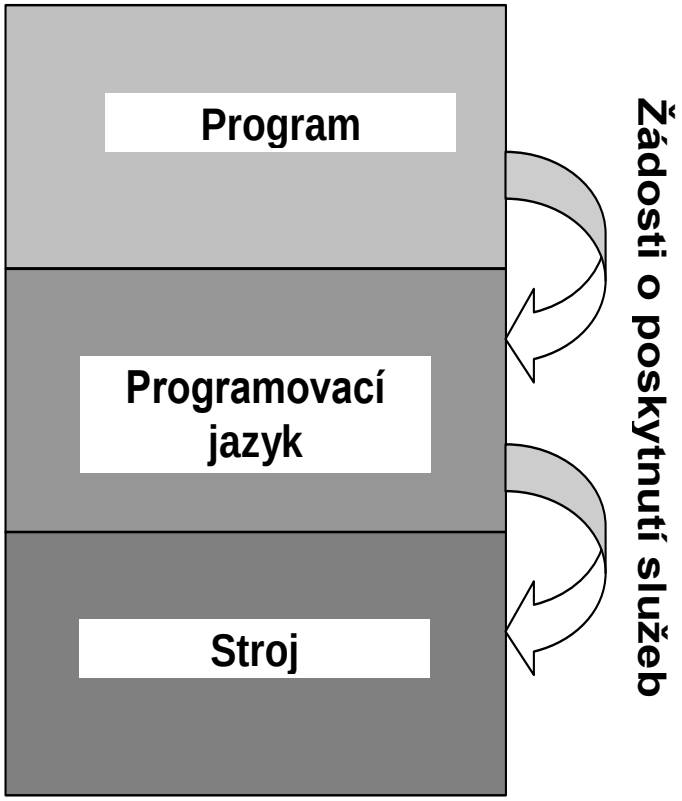
Vztah stroje, jazyka a programu

- stroj (resp. jazyk stroje), programovací jazyk a program představují tři na sobě závislé vrstvy
- každá vrstva využívá služeb vrstvy podřízené
- programovací jazyky proto představují přirozené a žádoucí rozšíření možností stroje.

Vztah stroje, jazyka a programu

Vztah mezi programem, jazykem a strojem

Každá z vrstev využívá služeb vrstvy nižší



Z kategorií služeb, které poskytují lze uvést:

- výpočetní model - jazyky mohou nechat vyniknout stroji nebo naopak charakteristiku stroje potlačit
- datové typy a operace - stroj většinou podporuje datové typy char, Int, float, ale neposkytuje typy jako jsou pole, záznamy atd.
- podpora abstrakce - je například možné zavést datový typ fronta, nad tímto datovým typem zavést operace a pak jej používat podobně jako strojový typ
- kontrola správnosti - mnoho programátorských chyb lze odhalit již za překladu.

Techniky řešení problémů

Úkoly k procvičení

Navrhněte algoritmus pro výpočet velikosti úhlu v trojúhelníku, který má velikosti stran *a,b,c*. Předpokládejte obvyklé označení stran a úhlů v trojúhelníku a dále předpokládejte, že zadané hodnoty budou splňovat trojúhelníkovou nerovnost. Výsledný úhel vypište ve stupních, minutách a vteřinách (zaokrouhleno na celé vteřiny). Pro výpočet úhlu použijte kosinovou větu.

Kontrolní otázky

1. Charakterizujte strategie řešení problémů na počítačích.
2. Jak je definován algoritmický problém?
3. Co je datová abstrakce?
4. Proč je důležitá strukturalizace v programování?
5. Co jsou jazyky nižší úrovně a jazyky vyšší úrovně. Jaký je mezi nimi rozdíl?
6. Jaký je rozdíl mezi kompilátorem a interpretem?
7. Vysvětlete von Neumannův princip počítače.
8. Jaký je vztah mezi strojem, jazykem a programem?