Programování

Úvod a základní principy

There are
10 types
of people
in the world:

Those who
understand binary
and those
who don't.

Martin Urza

Co je programování?

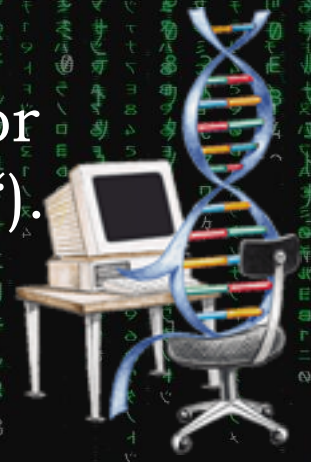
- Řešení úloh s pomocí počítače.
  - Počítače jsou extrémně rychlé a velmi hloupé.
  - Lidé jsou sice pomalejší, ale umí (nebo by většinou alespoň měli by umět) myslet.
  - Dobrý programátor ovládá umění spojit lidskou inteligenci s rychlostí strojů.
- Schopnost převádět problémy do řeči strojů.
- Způsob, jak donutit počítač, aby dělal to, co chceme, aby dělal.
- Rozhodně je programování dovednost psát programy.
  - Ale co je to program?

Programy a algoritmy

- Co je to program?
  - Zápis algoritmu v řeči strojů.
- A co je algoritmus?
  - Přesný návod či postup, kterým lze vyřešit daný typ úlohy.
    - Teoreticky by mohl algoritmus řešit pouze jednu instanci úlohy. V tom případě ale typicky nevyužívá rychlosti strojů.
    - Granularita kroků v návodu (tedy jak obecné může být zadání jednoho kroku) závisí čistě na platformě, pro kterou je algoritmus určen.
      - Algoritmus pro počítač ze Star Treku může obsahovat mnohem obecnější kroky než pro počítač současný.

Problém algoritmizace

- Lidé přemýšlejí diametrálně odlišným způsobem, než jakým fungují stroje.
  - Lidský mozek je (většinou) schopen abstrakce.
  - Stroje zatím dokáží pouze transformovat určité vstupy na jiné výstupy (i když velmi rychle).
- Algoritmus tvoří člověk, ale vykonává ho stroj.
  - **Při psaní algoritmu NELZE vycházet z toho, že stroj problému rozumí!!**
  - Program udělá vždy PŘESNĚ to, co programátor napíše, nic navíc (ani v případě, že je to „jasné“).
    - Věci jsou „jasné“ lidem, počítače jsou hloupé.
  - Programátor musí umět „hovořit řečí strojů“.



Příklady algoritmů pro současná PC

- Nalezení vyššího z čísel 1 a 2 (příklad algoritmu, kde není využita rychlost stroje).
  - Jako výsledek vrať 2 a skonči.
- Nalezení nejvyššího ze dvou čísel (*a*,*b*):
  - Pokud *a* > *b*, vrať *a*, jinak vrať *b*. Pak skonči.
- Nalezení nejvyššího z řady čísel:
  - Je-li řada prázdná, skonči chybou.
  - První číslo řady označ *nejvyšší*.
  - Postupně pro každé další číslo řady (označ vždy *prvek*), pokud *nejvyšší* < *prvek*, označ *prvek* jako *nejvyšší*.
  - Jako výsledek vrať *nejvyšší* a skonči.

Příklady nealgoritmických postupů

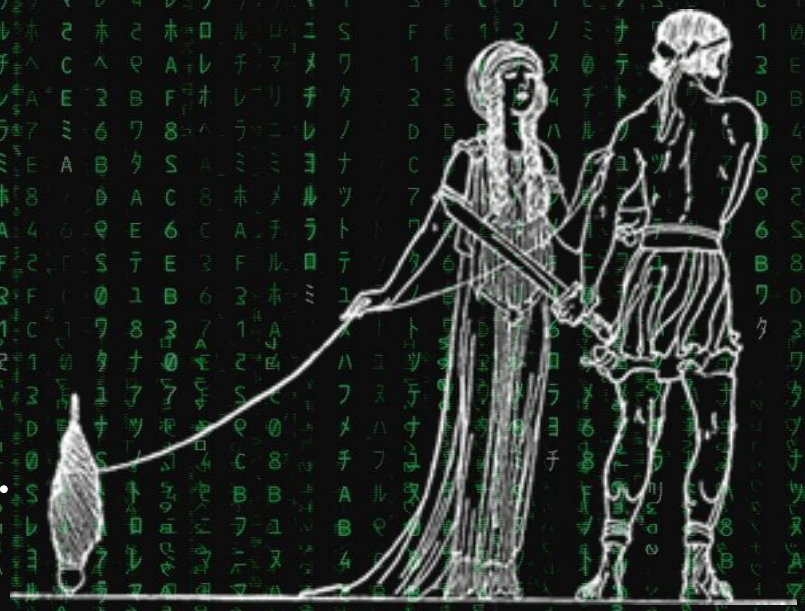
- Pro lidi:
  - Přeložte text do češtiny.
  - Zjistěte, zda se daný program vždy zastaví.
  - Nevystavujte se při práci zbytečnému riziku.
  - Proveďte bezpečnostní audit.

- Pro počítače ze Star Treku:
  - Zjisti, jestli je planeta vhodná pro výsadek.
  - Ověř, jestli nám od cizí lodi nehrozí nebezpečí.
- Tyto postupy jsou lidem jasné, ale nejedná se o algoritmy pro dnešní počítače.
- Ani nespadají do definice algoritmů (podle teorie vyčíslitelnosti).

Ariadnin algoritmus

- Ariadna, Theseus a Minotaurus (od docenta Holana, který báji doplnil fixou). Úkolem je **určitě** najít Minotaura (je-li v labyrintu) a bez bloudění se vrátit.
  - V každé místnosti postupuj odpředu podle následující tabulky:
    - Je tu Minotaurus → zab ho a vrať se po niti až k Ariadně.
    - Je tu více chodeb, ze kterých vede nit → motej nit zpět a chodbu značkuj fixou.
    - Je možnost jít chodbou, která není označena fixem → vydej se tam.
    - Je tu Ariadna → statický Minotaurus neexistuje.
    - Je tu jen jedna chodba, ze které vede nit → motej nit zpět a chodbu značkuj fixou.

Ariadnin algoritmus



- Jaké má vlastnosti?
  - Vždy existuje nějaké pravidlo.
  - Vždy skončí.
  - Cesta zpět se nebude křížit.
  - Nit neustále určuje cestu zpět.
  - Zastaví se u Ariadny.
  - Existuje-li statický Minotaurus (tedy takový, který se nepohybuje), najde ho.
- Ariadnin algoritmus by bylo možné provést i bez fixy, ale museli bychom mít delší nit.
  - Nit je analogií pro paměť – dala by se reprezentovat jako posloupnost míst, kterými prochází.

Je algoritmus správně napsaný?

- **Správnost algoritmu nelze algoritmicky ověřit.**
  - Nejen proto, že pojem „správnost“ je vágní.
  - Algoritmicky není ověřitelné ani to, zda se algoritmus zastaví!
    - Halting problem (velmi známý)
  - Algoritmická neověřitelnost správnosti algoritmu je dokázána.
    - **Je velký rozdíl mezi tím, když je dokázáno, že něco nejde, a tím, že na něco ještě nikdo nepřišel.**
      - Algoritmus, který bude říkat cokoliv zajímavého o jiných algoritmech, nikdy nebude existovat.
      - Program, který porozumí přirozenému jazyku, zatím neexistuje (a pravděpodobně ještě dlouho nebude), ale není vyloučeno, že by (někdy) existovat mohl.

Problémy s ověřováním správnosti algoritmů

- U všech „zajímavých“ algoritmů nestačí k ověření správnosti konečný počet kroků (dokonce ani k ověření toho, jestli algoritmus vůbec skončí).
- Příklad (starověký čínský algoritmus):

  - Úkol: Zjistěte, zda je číslo n prvočíslo.
  - Postup: Je-li n dělitelem výrazu $(2^n - 2)$, pak je n prvočíslo, jinak není.
  - Příklad: 28 – 2 = 254 (není dělitelné osmi), takže 8 není prvočíslo. 211 – 2 = 2046 (dělitelné jedenácti), takže 11 je prvočíslo.
  - Problém: Pro číslo 341 algoritmus selže (kdo to mohl čekat?), funguje tedy jen pro čísla od jedné do 340.

Sémantika a syntax

- O sémantice a syntaxi má smysl mluvit v souvislosti s algoritmy, respektive jejich elementárními kroky.
- Sémantikou rozumíme význam jednotlivých kroků algoritmu, je to vlastně myšlenka.
  - Příklad: Sémantika zápisu 2 + 4 je většinou matematický součet čísel 2 a 4 (může být ale i jiná).
- Syntax je pak zápis myšlenek (sémantiky).
  - Příklad: Operaci mocnění dvojky na čtvrtou můžeme vidět zapsanou jako 2^4 nebo 2^4 , což jsou syntakticky dva různé zápisy stejné věci.
- Sémanticky stejný algoritmus se popisuje různou syntaxí v závislosti na tom, čemu nebo komu je algoritmus určen.

Programovací jazyky

- Stejnou sémantiku lze zapsat syntakticky různě (v rozdílných jazycích).
  - Různé jazyky se z valné většiny liší jen syntaxí.
  - Sémanticky se jazyky liší pouze na úrovni jednotlivých příkazů (granularitou kroků algoritmů).
    - Drtivá většina programovacích jazyků (vlastně všechny „normální“) je stejně silná. To znamená, že co jde napsat v jednom, lze i ve všech ostatních (a naopak – co v jednom jazyce není možné napsat, nelze v žádném).
- Programovací jazyky (určené pro počítače) jsou oproti jazykům přirozeným (určeným pro lidi) jednoduché a strohé, aby jim stroje rozuměly.
  - Celé to „porozumění“ spočívá jen v tom, že každý příkaz jazyka lze popsat instrukcemi procesoru.

Rozdíly mezi programovacími jazyky

- Jestliže lze ve všech programovacích jazycích napsat totéž, není jedno, který použijeme?
  Ne!! Jazyky se liší. Čím?
  - Rychlostí programů napsaných v daném jazyce.
  - Rychlostí a pohodlností psaní kódu pro programátora.
  - Specializací na určitý typ úloh.
  - Knihovnami, které pro daný jazyk existují.
- Tato kritéria mají signifikantní vliv na výsledky (a cenu) práce.
  - Fatální a častou chybou bývá špatné použití PHP.
  - Obrovské urychlení může někdy přinést Prolog.

Dělení programovacích jazyků

- Dle míry abstrakce jsou jazyky různě „vysoké“.
  - Vyšší jazyky používají větší míru abstrakce - jedním příkazem lze vykonat mnoho instrukcí (→ kratší kód).
  - Čím vyšší jazyk, tím bývá většinou pohodlnější (rychlejší) programování, avšak pomalejší kód.
- Podle způsobu překladu a spuštění:
  - Kompilované (programovací) jsou většinou rychlé.
  - Interpretované (skriptovací) bývají vyšší a pomalé.
- Dle způsobu programování:
  - Procedurální (imperativní), těmi se budeme zabývat.
  - Neprocedurální (deklarativní), programuje se v nich jinak, nebudeme se jimi zabývat.

Umění programovat

- **Dovednost programování *N*E*S*P*O*Č*Í*V*Á* ve znalosti co největšího počtu jazyků !!!!**
  - Navzdory tomu, že si to mnoho lidí myslí.
  - **Není pravdou**, že dobrý programátor musí umět syntaxi mnoha jazyků.
    - Šikovný programátor se nové jazyky učí relativně rychle.
    - Dobrý programátor použije i neznámý jazyk lépe než ten, kdo programovat neumí a takzvaně „umí“ onen jazyk.
  - Dovednost programování stojí stranou od znalosti jazyků (která je bez schopnosti programovat k ničemu).
  - Umění tvorby programů je o sémantice, nikoliv o syntaxi.
  - Protože jazyky jsou důležité, spadá do umu programování i volba vhodného jazyka pro řešení problému.

Jak se naučit programovat

- Byly časy, kdy se programování učilo psaním programů na papír.
- Byly i jiné (pozdější) časy, kdy se programování vyučovalo jen jako ovládnutí jednoho jazyka.
- Je dobré zvolit kompromis.
  - Učit se programovat, ale využít k tomu i konkrétní jazyk pro větší názornost.
  - S dalšími jazyky začít až po dosažení jisté úrovně v umění programovat.
- **Nezapomínat, že jazyk je jen prostředkem k učení.**
- V kterém jazyce začít? V Pascalu.

Jazyk Pascal

- Zastaralý a nemoderní jazyk, který už se nevyvíjí.
- Byl navržen a vytvořen za účelem učení programování, proto je pro výuku vhodný.
- Dnes už se v praxi nepoužívá.
- Existují i volně šiřitelné překladače (Free Pascal).
- Vývojové prostředí pro DOS a tvorbu konzolových aplikací. Roku 1995 vyvinula firma Borland grafické prostředí Delphi k tvorbě aplikací pro Windows.
  - S nástupem Delphi měl jazyk Pascal obrovský potenciál, v té době nic podobného na trhu nebylo.
    - Borland však potenciálu nevyužil, Pascal/Delphi programátoři používají dnes jen zřídka.

Rekapitulace

- Programování je o algoritmizaci problémů, aby je mohly řešit stroje, které jsou daleko rychlejší než člověk.
- Počítače jsou hloupé, algoritmy musí být přesné a nelze předpokládat, že stroj udělá něco navíc jen proto, že „je to přece jasné“.
- **Umění programovat NENÍ znalost co největšího počtu jazyků!!** Při algoritmizaci jde o sémantiku, ne o syntaxi.
- Různé jazyky jsou vhodné pro různá využití. Použití špatného jazyka je častá chyba. Dobrý programátor umí zvolit pro zadaný úkol vhodný jazyk.