Tři otázky k výuce programování

Martin Mareš, KAM MFF UK, mj@ucw.cz

Úvodem

Již nějaký čas se na Matfyzu diskutuje o tom, jak by měl vypadat základní kurs programování a zejména jaký programovací jazyk by měli studenti potkat jako první. Toto je můj osobní pokus o zmapování problému a utřídění názorů, inspirovaný zkušenostmi z výuky programování na Matfyzu i ve středoškolském prostředí. Nepovažujte, prosím, nic z toho za konstatování faktu, nýbrž spíš za můj názor, fakty více či méně podložený. Správně by zajisté před téměř každou větou mělo stát *„Myslím si, že …“*

Děkuji ostatním cvičícím programování, organizátorům KSP a MO-P a mnoha dalším kolegům za nevysychající pramen připomínek, které mne k tomuto textu dovedly.

Proč?

K čemu všemu vlastně slouží základní přednášky z programování slouží? Co mají studenty naučit?

- *algoritmicky přemýšlet* – většina nastupujících studentů dokáže bez problémů zbastlit jednoduchý program v nějakém jazyce, ale mnozí mají problémy se základy algoritmického myšlení a nerozlišují mezi dřevorubeckými a efektivními algoritmy.
- *programátorské řemeslo* – každý by měl umět základní programátorské techniky od klasických typu rekurse či návrh shora dolů až po objekty, rozhraní a modularitu. K tomu také patří schopnost porozumění cizímu programu.
- *základní štábní kulturu* – co dodat?
- *domluvit se s ostatními* – to znamená nejen umět formulovat své myšlenky, ale také rozumět těm cizím a zejména rozumět základní odborné literatuře.

Jak?

Co by měl splňovat jazyk používaný pro základní kurs programování? (Neříkám, že tyto požadavky nejsou protichůdné …)

- *přímočarost* – jednoduché věci by měly jít zapsat jednoduše, bez zbytečné omáčky, které student nemůže rozumět. O pár konstrukcích lze zajisté říci „dělají tohle, časem pochopíte, proč“, ale každá další výrazně ztěžuje porozumění a hlavně znesnadňuje odhalování chyb.
- *odolnost* – začátečníci dělají chyby. Spoustu chyb. Hodně jim tedy pomůže, když lze většinu chyb odhalit již v zárodku: překlep obvykle způsobí syntaktickou chybu, přetečení pole běhovou chybu atd. Zde propadá klasické Céčko i C++.
- *průhlednost* – při pohledu na program (nejlépe i cizí) by mělo být jasné, co dělá. Studenti by měli poznat značnou část jazyka, aby mohli číst i cizí kód (například nahlížet do standardních knihoven nebo alespoň do deklarací jejich funkcí). Zkuste si představit, jak na složitější program bude reagovat někdo, kdo programovat umí, ale daný jazyk v životě neviděl.
- *abstrakce* – na jednu stranu je dobré, když jsou začátečníci oproštěni od technických detailů, jako je alokace paměti nebo propojování ukazatelů v seznamech. Ovšem také je potřeba, aby si uvědomovali, které operace jsou elementární a které již složité a pomalé, protože bez toho nedokáží posoudit efektivitu svých programů. Úroveň abstrakce je tedy nutné volit velice uvážlivě.
- *prostředí* – studenti by měli mít k dispozici kvalitní vývojové prostředí, nejlépe takové, které odpovídá složitosti programů, které píší. Spolehlivý debugger je nutností, spousta funkcí použitelných pouze na velké projekty spíše překáží. Hodně může pomoci, když jazyk dovede být interaktivní – zadávat příkazy (nebo spouštět již hotové funkce) a hned vidět, co dělají, v začátcích hodně pomůže.

- *rodokmen* – jazyk by měl být živý (nemá smysl tratit čas učením něčeho, co nikdo na světě nepoužívá) a také je vhodné, aby měl své vývojové příbuzné, protože pak usnadní učení dalších jazyků. Zde asi vedou jazyky z Céčkové rodiny, Pascal je takovou programátorskou latinou – jazyk sám už se moc nepoužívá, ale dal vzniknout spoustě „románských jazyků“, např. Object Pascalu, který je ještě stále populární.
- *praktičnost* – použitelný v praktickém životě, podporuje řemeslné dovednosti. Zde vedou moderní (módní?) aplikační jazyky, jako Java nebo C$^{\#}$, ale i staré dobré Céčko, které je pravděpodobně stále nejpoužívanějším jazykem v Unixovém světě.
- *návaznost* – mnozí (ale ne všichni) prváci přicházejí s nejrůznějšími programátorskými zkušenostmi. Asi je vhodné na ně navázat, ale spíše v konceptech než v detailech – pokud je někdo zvyklý na klasické proměnné a side-efektní operace, asi bude chápat snáz procedurální jazyk než čistě funkcionální. Na druhou stranu někdy také může být vhodné volbou nepříliš známého jazyka vyrovnat počáteční rozdíly mezi studenty, ale toho lze asi dosáhnout i rozumněji.
- *návaznost podruhé* – rovněž je důležité navázat na ostatní předměty. Není asi rozumné učit současně nízkoúrovňové detaily v Principech počítačů a programování v nějakém vysoce abstraktním jazyce a nic mezi tím, co by dávalo šanci si oba světy spojit. Na druhou stranu by mělo jít snadno navázat pokročilejšími jazyky. Některá zaměření studia také mají své typické jazyky, se kterými by se studenti měli během studia určitě setkat: aplikační programátoři by jistě měli umět Javu nebo C$^{\#}$, systémoví programátoři potřebují C, teoretici funkcionální jazyky.
- *dostupnost* – v neposlední řadě by mělo být vývojové prostředí dostupné všem studentům, to jest nejlépe zdarma pro všechny běžně používané operační systémy. Výhodou je, není-li potřeba instalovat žádné obrovské programové balíky a funguje-li překladač i na starším hardwaru, který se mezi studenty stále vyskytuje.
- *literatura* – k jazyku by měla existovat dostupná (nejlépe česká) úvodní příručka. Rovněž by znalosti jazyka měly stačit ke čtení algoritmů v běžné základní literatuře. Zde značně pokulhávají Céčkové jazyky, většina dobrých učebnic algoritmů používá pseudokód inspirovaný hlavně Pascalem a v menší míře funkcionálními jazyky (např. Rivest et al.: Introduction to Algorithms nebo novější Dasgupta et al.: Algorithms, podle mého názoru dvě vůbec nejlepší učebnice algoritmů, které ostatně doporučujeme k našim přednáškám).
- *cvičící* – také je potřeba sehnat spoustu lidí, kteří budou základní kurs cvičit. Cvičící by měli jazyk znát velice dobře, aby byli schopni pomoci, když si studenti s chováním svého programu nevědí rady. Proto by to měl být jazyk, který se lze rozumně snadno *celý* naučit.

Co tedy?

Vím o těchto možnostech:

- *nějaký „Céčkovitý“ jazyk* – Céčko a C++ opravdu vhodné nejsou, zbývá Java a C$^{\#}$ (a některé obskurnější jazyky, jako třeba Objective C, které si dovoluji zanedbat). Výhodou je návaznost na praxi, „rodokmen“ a ještě poměrně dobrá úroveň abstrakce, nevýhodami pak nedostatek přímočarosti a průhlednosti. Kompatibilita s algoritmickou literaturou také není valná. Nevím, jestli najdeme cvičícího, který by takový jazyk uměl dokonale.
- *nějaký funkcionální jazyk* – některé světové university začínají kurs programování jazykem funkcionálním, typicky nějakým dialektem Lispu nebo Haskellem. Na první pohled je to docela nezvyklé a s praxí nepříliš propojené, na druhou stranu například rekurse, která studentům v procedurálních jazycích tradičně dělá problémy, je v čistě funkcionálních (side-efektů prostých) jazycích pochopitelná mnohem snáz. Jednoduché programy se v Haskellu píší velice snadno (*čitelný* Quick-sort na 3 řádky),

trochu neintuitivní je I/O; grafika a okénka pravděpodobně mimo možnosti základní přednášky. Již se učí, takže s cvičícími nebudou problémy. Odolnost vůči chybám výtečná, ale chvíli trvá, než začátečník pochopí chybová hlášení překladačů.

- *nějaký skriptovací jazyk,* pravděpodobně Python – velkou výhodou je interaktivita, někdy je až moc abstraktní. V praxi se používá, i když ne tak široce jako C a spol. Není příliš ukecaný, jednoduché programy lze psát opravdu přímočaře. Clickací prostředí pro něj existuje (SPE, důkladně jsem ho ale ještě nezkoumal), ladění z příkazové řádky je díky interaktivnímu módu dosti příjemné. Podobá se pseudokódům používaným v literatuře. Lze v něm programovat procedurálně i (téměř) funkcionálně. Poměrně rozsáhlý, ale velice systematický a neskrývá pod pokličkou asi žádná velká překvapení.
- *Pascal* – na Matfyzu se učil mnoho let a myslím, že se osvědčil. Nezapře se, že je to jazyk primárně vymyšlený pro výuku programování. Proto je dosti přímočarý, až na pár drobných nesystematičností. Kvůli této přímočarosti není zase moc vhodný pro psaní větších programů (i když Object Pascal tyto neduhy částečně napravuje). Vyžaduje dosti low-level přístup k alokaci a ukazatelům, ale jinak velmi dobře odpovídá způsobu uvažování, na který jsou studenti zvyklí (ostatně, na části středních škol se učí a v KSP i MO-P se hojně používá). S literaturou je kompatibilní dobře, pseudokód ve většině knih je Pascalu dost podobný. Základní programátorské návyky i algoritmické myšlení podporuje velice dobře (ověřeno), pokročilejší partie programátorského řemesla nepříliš.

Jak je vidět, žádný jazyk neobstál ve všech kritériích a asi to ani není možné – požadavky kladené na jazyk pro programátory začátečníky a na takový, který se používá pro opravdové programování, jsou značně protichůdné.

Nabízí se začít nějakým praktickým jazykem, představit rovnou objektově orientované programování a vysvětlení detailů nechat na později. Myslím si ale, že to není vhodné. Na fakultu nastupuje mnoho studentů, kteří s programováním mají minimum zkušeností a i do jednoduchých věcí se dokáží snadno zamotat. Často vídám zápočtové programy, které mají stovky až tisíce řádků, ačkoliv dělají něco naprosto triviálního, protože se je autoři snažili napsat (např.) „důsledně objektově.“ Takové programy pak studenti často nedovedou odladit, protože nerozumějí tomu, co program dělá, do dostatečné hloubky.

Proto by začátek měl být pozvolný a prváci by si nejdříve měli osvojit jednoduché koncepty a naučit se je efektivně používat, než je začneme učit ty pokročilejší. Objekty, namespaces a podobné věci jsou v praxi nezbytné a student by se je jistě měl naučit, ale do prvního semestru sotva patří.

Samozřejmě by bylo pěkné vybrat si nějaký jazyk a nejdříve (řekněme v prvním semestru) v něm naučit programovat jednoduše a teprve později zavádět náročnější koncepty, ale to může v Javě nebo C# sotva fungovat, protože bez znalosti objektů nelze napsat ani jednoduchý program.

Lepší tedy bude vybrat si jazyky dva: jeden pro základní kurs programování (pravděpodobně stačí jednosemestrální), ve kterém se studenti naučí úplné základy. Na těch pak mohou stavět jak teoretičtější přednášky o algoritmech a datových strukturách, jednak výuka dalšího, pokročilejšího jazyka.

Jako základní jazyk se podle mého názoru nejvíce hodí buďto starý dobrý Pascal nebo interaktivní jazyk typu Python.

Pro pokročilejší kursy se pak nabízí buďto zůstat u Pascalu a rozšířit ho na Object Pascal, nebo se přesunout k Céčku a C#. Jelikož se ale požadavky jednotlivých oborů liší, mohlo by být šikovné od tohoto bodu výuku dalších jazyků „modularizovat,“ tj. učinit z nich volitelné předměty, přičemž pro každý obor bude nějaká podmnožina z nich povinná.

Nabízím tedy další variantu k diskusi, a to jak v rámci ctěné komise pro reformu studia, tak mezi studenty a kolegy cvičícími. Veškeré připomínky a náměty vítám. ♡