Univerzita Pardubice
Fakulta ekonomicko - správní

Návrh vhodného způsobu výuky algoritmizace

Jiří Pomikáček

Bakalářská práce
2008

SOUHRN

Práce se věnuje výuce algoritmizace a programování na vysokých školách. Uvádí stručný historický přehled programovacích jazyků a jejich různá dělení. Dále se práce mapuje výuku předmětu algoritmizace a programování na různých vysokých školách s ekonomicky zaměřenými studijními programy. V neposlední řadě se snaží testovat na několika vybraných programovacích jazycích jejich softwarové nástroje. A nakonec je snaha o shrnutí nabytých informací a za pomocí nich sestrojení „obecného sylabu“ pro výuku algoritmizace a programování.

KLÍČOVÁ SLOVA

algoritmizace, programování, výučování, programovací jazyky, sylaby

TITLE

The concept of education of algorythm development and programming

ABSTRACT

The work is about educating algorythm anf programming on the universities. There is brief historical survey of programming languages and theirs taxonomy. Next is chart of educatiing of subject algorythm development and programming on the economical universities. Not least there si endeavour of testing on some programming languages their software. In the end there is summary acquiried informaton and with them to make general sylabus for education of algorythm development and programming.

KEYWORDS

Algorythm development, programming, education, programming languages, syllabus

OBSAH

Úvod

Práce se zabývá nalezením vhodného způsobu výuky algoritmizace a programování na vybraných vysokých školách. Cílem této práce je získat přehled o různých způsobech výuky algoritmizace a programovaní a dále shrnout softwarové a programové možnosti v tomto předmětu. Také tyto způsoby výuky a softwarové vybavení zhodnotit a na základě toho se pokusit o návrh koncepce vhodného způsobu výuky algoritmizace a programování.

První kapitola stručně popisuje historický vývoj programovacích jazyků, jakým směrem se programovací jazyky vyvíjely. Následuje rozdělení programovacích jazyků podle různých kritérií. V závěru této kapitoly je nastíněno, k čemu je vlastní programování a v jakých souvislostech se s programováním můžeme setkat.

Ve druhé kapitole se seznámíme s dnešními metodami výuky programování. Nejprve je přiblíženo, jakými způsoby lze k samotné výuce programování přistoupit. Druhá kapitola také stručně pojednává o současných trendech výuky. Podle zvoleného klíče je v této kapitole vybráno několik vysokých škol a uvádí se zde jejich sylaby. Tento výběr vymezí hranice testování programovacích jazyků.

Třetí kapitola pojednává především o softwarové části předmětu algoritmizace a programování. Podle výše vybraných vysokých škol se zabývá jednotlivými nástroji pro výuku, jak po stránce teoretické, tak po stránce praktické. V kapitole jsou také jednotlivé nástroje hodnoceny.

Poslední kapitola se zabývá konečným návrhem výuky algoritmizace a programováním na vysokých školách. Jsou zde zúročeny předešlé informace o výuce a programovém vybavení. Pro jednodušší dokreslení je sestrojen návrh sylabu výuky.

1. Úvod do algoritmizace a programování

Tato kapitola popisuje historický vývoj programovacích jazyků a rozdělení jazyků podle jednotlivých kritérií. V poslední části této kapitoly je popsáno, k čemu může algoritmizace sloužit.

1.1. Historie programovacích jazyků

Vývoj programovacích jazyků určovaly a určují mnohé faktory; k nejzávažnějším z nich pravděpodobně podle [1] patří:

- vývoj technických prostředků,
- vývoj metod strojového překladu jazyků,
- vývoj metod programování,
- stále širší oblasti využívání počítačů,
- psychologické faktory.

V prvním období existence počítačů zastával roli programovacího jazyka výhradně strojový kód, nazývaný proto také strojový jazyk. Vyjadřovacími prostředky strojového jazyka jsou instrukce. Jejich sémantika je definována technickými prvky počítače. Z této skutečnosti vyplývá základní nedostatek programování ve strojovém jazyce - závislost na konkrétním typu počítače. Dalšími, z hlediska programování nepříznivými, rysy strojového jazyka je velmi nízká úroveň popisovaných akcí a pro člověka nepohodlný číselný zápis instrukcí. V současné době je používání strojového jazyka jako nástroje programování zcela ojedinělé. [1]

Existence prvních strojově nezávislých programovacích jazyků byla podmíněna zvládnutím techniky počítačem realizované transformace určitého textu na text reprezentující program ve strojovém jazyce. Tato technika transformace byla nazvána *překladem*.

Prvním krokem ve vývoji programovacích jazyků byla náhrada numericky kódované informace instrukcí strojového jazyka informací symbolickou. Vzniká tak důležitá třída programovacích jazyků - jazyky symbolických instrukcí.

Překladač jazyka symbolických instrukcí, jenž se nazývá *assembler*, je relativně velmi jednoduchý.

Příznačným rysem jazyka symbolických instrukcí je jeho strojová závislost. Syntax i sémantika instrukcí odráží takové rysy konkrétního počítače, jako je soubor operací a způsoby adresování operandů. Proto se dva jazyky symbolických instrukcí mohou od sebe podstatně lišit. [1]

Jazyk symbolických instrukcí patří do skupiny symbolických, strojově závislých jazyků. I když z hlediska vývoje programovacích jazyků patří k jazykům nejméně dokonalým, má určité rysy, které mu přisuzují i u počítačů 3. a 3,5. generace[1] důležité postavení v jejich programovém vybavení. Je to především rychlost překladu a zpracování přeloženého programu a možnost programování na elementárních úrovních, která je v některých případech nepostradatelná a ve vyšším programovacím jazyce nedostupná. Tyto vlastnosti vyplývají z těsné blízkosti jazyka symbolických instrukcí se strojovým jazykem značné výhody, a proto má v programovém vybavení své důležité postavení, zvláště při vytváření programů operačního systému.

Další vývoj směrem k vyšším programovacím jazykům sledoval podle [1] dva cíle:

- odstranit strojovou závislost jazyků, vyžadující od programátora detailní znalosti strojového jazyka a znemožňující použití programů vypracovaných pro jeden typ počítače na počítači jiného typu
- poskytnout programátorům takové prostředky pro popis algoritmů, které odrážejí povahu řešených problémů, a nikoliv technickou realizaci výpočetních procesů na počítači.

Éru skutečných vyšších programovacích jazyků zahájil až Fortran. Ve své první podobě byl Fortran opravdu snadno naučitelným jazykem vedoucím k maximálně efektivnímu programu. Programování tak přestalo být výsostnou doménu skupiny úzce specializovaných odborníků. Fortran byl však původně jazykem určeným pro vědeckotechnické výpočty, proto nemohl vyřešit úplně všechno. [1]

1.2. Rozdělení programovacích jazyků

Programovací jazyky lze dle [2] dělit několika způsoby a různých kritérií:

dle míry abstrakce:

- vyšší programovací jazyky (většina jazyků)
- nižší programovací jazyky (např. Assembler, částečně VHDL)

[1] Počítače třetí a vyšších generací jsou vybudovány na integrovaných obvodech, které na svých čipech integrují velké množství tranzistorů. U této generace se začíná objevovat paralelní zpracování více programů, které má opět za úkol zvýšit využití strojového času počítače. Generace nastupuje na začátku 70. let.

dle způsobu překladu a spuštění:

- kompilované programovací jazyky (např. Pascal, C)
- interpretované programovací jazyky (např. BASIC, Perl, Python, shell)
    - interpretované jazyky, které se pouze interpretují (z toho důvodu jsou pomalejší - proto většina jazyků má alespoň nějakou jinou možnost, pokud nejsou stejně zpomalované něčím jiným, jako třeba shell)
    - interpretované jazyky, které se překládají, ale pouze do mezikódu, nikoli do strojového kódu počítače (např. Java, Python)
    - interpretované jazyky, které se po spuštění za běhu programu překládají do strojového kódu počítače (např. Java)

Toto členění není absolutní, řada programovacích jazyků existuje v implementaci jak interpretované, tak kompilované (například zmíněná Java). Navíc jsou oba postupy někdy kombinovány, zdrojový kód je nejprve kompilován do mezikódu, který je poté interpretován.

Vyšší programovací jazyky se dále dělí dle [2] takto:

- Procedurální (imperativní)
    - Strukturované (např. C, BASIC)
    - Objektově orientované (např. Smalltalk, Java)
- Neprocedurální (deklarativní)
    - Funkcionální (např. Lisp, Haskell)
    - Logické (např. Prolog, Goedel)

1.3. Využití algoritmizace a programování

Může se zdát, že algoritmizace a programování jsou vhodné pouze k programování. Tato část má přiblížit důvody proč a k čemu může být použita algoritmizace a programování i pro studenty, kteří se konkrétně programováním nikdy zabývat nebudou.

Otázka typu: „Proč se učit matematiku, když nebudu matematikem?“ je od studentů řečena často, zde si uvedeme několik argumentů ohledně výuky algoritmizace:

- obdobně jako u matematiky je řešení úloh algoritmizací tréninkem mozku. Dnes za nás počítače řeší mnoho a spousta lidí nezaměstnává dostatečně svůj mozek, a tedy stejně jako svaly když se netrénují tak schopnosti mozku „atrofují“

- programování je proces řízení a vykonání instrukce nenásleduje bezprostředně po vydání pokynu. Toto oddálení trénuje představivost a abstrakci studenta.
- algoritmizace je více či méně matematická disciplína a tudíž bychom mohli namítnout, proč učit oba dva předměty k tréninku inteligence, ale umění naprogramovat si svoji činnost, schopnost přečíst si a porozumět návodu a postupovat podle něj a také schopnost návod samostatně sestavit matematiku přesahují. Jde o schopnosti sekundární gramotnosti a navíc je zde přítomnost počítače jako bezprostřední zpětné vazby.
- programování používá svoji symboliku, svůj jazyk, svoji strukturu, do které je třeba proniknout, což opět vede k trénování inteligence.

Málokdo si uvědomuje, že zrovna programování je činnost, kterou provozujeme od nejútlejšího dětství. Algoritmus totiž není obecně nic jiného, než specifikovaný postup vedoucí ke splnění zadaného úkolu. Jenže takový postup již navrhuje i batole, které se chce dostat ke hračce, na niž bez pomůcky nedosáhne. S postupujícím věkem jsou naše algoritmy stále složitější. Postupně se do nich dostávají procedury (poprosím staršího bráchu, aby mi sundal balón ze skříně, kam nedosáhnu), funkce (zeptám se kolemjdoucího, kolik je hodin), proměnné (musím si zapamatovat, kudy jsem šel, abych příště trefil), podmínky (když mne Lojza pozve na chatu, strávím víkend s ním, jinak zavolám Frantovi), cykly (dokud nenajdu peněženku, nemohu odejít do hospody) a další známé programátorské prvky. [3]

Když se začneme učit programovat, to nejdůležitější již většinou známe. Potřebujeme si pouze uvědomit své dosavadní znalosti a dovednosti a naučit se zapsat naši představu o řešení úlohy pomocí nějakého formalizmu a především s respektem k omezeným možnostem používaného počítače a programovacího jazyka. Otázkou tedy není, zda učit programování, ale jak jej učit a kdy s jeho výukou začít. Musíme je učit tak, abychom co nejvíce využili stávajících znalostí našich žáků a aby to, co je naučíme, mohli co nejlépe využít ve svém dalším životě, a to nezávisle na tom, budou-li programovat počítače nebo řešit zcela jiné úlohy. Nesmíme si ale plést výuku programování s výukou konkrétního programovacího jazyka. Při výuce programování by nemělo být důležité, jak se to či ono naprogramuje např. v Basicu. Měli bychom se soustředit především na vysvětlení základních principů, postupů, obratů a pak ukázat jejich realizaci v použitém programovacím jazyce. To však neznamená, že na použitém programovacím jazyce nezáleží.[3]

2. Různé způsoby výuky a jejich hodnocení

Tato kapitola přibližuje rozdílné přístupy k výuce algoritmizace a programování. Nastíněny jsou zde také současné trendy a závěrem je zvolen klíč k výběru vysokých škol kde se algoritmizace a programování vyučuje. Na závěr jsou podle tohoto klíče konkrétní školy vybrány.

2.1. Způsoby přístupů k výuce

V této části je popsáno šest základních přístupů k výuce algoritmizace a programování.

2.1.1. Nejdříve hardware (Hardware-first)

Zastánci tohoto přístupu tvrdí, že k tomu, aby studenti dokázali správně programovat, musí nejprve vědět, jak je počítač konstruován, protože jedině tak si mohou představit, jak bude jejich program prováděn. Výuka začíná výkladem spínacích obvodů, konstrukcí registrů a aritmetických jednotek, a teprve poté pokračuje výkladem konstrukce programů ve strojovém kódu a následně ve vyšších programovacích jazycích. Tato koncepce se uplatní pouze v několika speciálních oborech, protože většinou, zejména pak při tvorbě rozsáhlých aplikací, je považováno za optimální, je-li programátor od realizačního hardwaru co nejvíce odstíněn. [4]

2.1.2. Nejdříve algoritmy (Algorithms-first)

Tento přístup nevyužívá k výkladu některého z existujících jazyků, ale vykládá základní algoritmy za použití pseudokódu. Studenti se nejprve učí základní principy, aniž by se zdržovali laděním nějakých programů. Zkušenost však ukazuje, že právě absence této zpětné vazby a nemožnost si vše vyzkoušet je pro studenty demotivující. [4]

2.1.3. Nejdříve příkazy (Imperative-first)

Klasická, a stále nejpoužívanější metodika výuky. Při ní se studenti nejprve seznámí s klasickými programovými konstrukcemi a teprve pak s případnou objektově orientovanou nadstavbou. Zkušenost však ukazuje, že takto připravovaní studenti se nesžijí s objektově orientovaným paradigmatem tak dobře, jako studenti, kteří začali výuku hned prací s objekty, což je vzhledem k současnému významu OOP považováno za velký

handicap tohoto přístupu. Jednou z velkých nevýhod takto koncipovaných kurzů je pak to, že se jedná především o kurzy syntaxe a nikoliv o kurzy programování. Vyučující přednáší a cvičí tak, jakoby předpokládali, že umění programovat přijde se znalostí syntaxe jako vedlejší efekt. Nepřijde. [4]

2.1.4. Nejdříve funkce (Functional-first)

Tento přístup zavedli v osmdesátých letech v MIT. Jeho výhodou je sjednocení počáteční úrovně studentů, protože se zde setkají s jazykem, jehož filozofie je výrazně jiná než filozofie jazyků hlavního proudu. Tato odlišnost ale na druhou stranu mnohé ze studentů demotivuje, protože se nechtějí učit něco, co pak ve své praxi přímo nepoužijí. [4]

2.1.5. Nejdříve objekty (Objects-first)

Tato koncepce vychází ze skutečnosti, že OOP je zdaleka nejpoužívanější metodikou programování a mají-li si je studenti opravdu osvojit, musí se s ním setkávat od samého počátku výuky. Nevýhodou tohoto přístupu je, že objektově orientované jazyky bývají koncipovány jako komplexní a studenti si pak někdy připadají jejich složitostí zcela zahlceni. Je přitom jedno, zda jde o složitost vlastního jazyka, jak je tomu např. v případě jazyka C++, nebo o složitost standardní knihovny, jak je tomu v případě jazyka Java. Kurzy je proto třeba koncipovat tak, aby k tomuto zahlcení nedošlo. [4]

2.1.6. Nejdříve zeširoka (Breadth-first)

Zastánci této koncepce tvrdí, že by se studenti měli nejprve seznámit s problematikou počítačové vědy v co největší šířce, a teprve pak se soustředit na takové detaily, jakým je např. programování. Absolventi těchto kurzů přistupují k řešení problémů z většího nadhledu a jsou jej často schopni chápat v celé jeho šíři. Kritici však této koncepci vytýkají, že odkládá výuku programování a tím i na ni navazující předměty o jeden až dva semestry, což není vždy vyváženo lepšími výchozími znalostmi studentů. [4]

2.2. Současné trendy při výuce programování

Programování doznalo v posledních 25 letech řadu naprosto zásadních změn. Měnily se nejenom používané programovací jazyky, ale měnila se i filozofie celého programování.

Dávno již skončily doby, kdy programátoři vyvíjeli většinou samostatné programy a potřebovali se vedle syntaxe použitého jazyka naučit především dovednostem, jak správně algoritmizovat ten či onen problém. Nyní se musí místo toho naučit začleňovat své programy do existujících systémů, ovládat řadu nejrůznějších technologií a standardů a zvládat rozsáhlé knihovny.

Ať si to již připouštíme nebo ne, většina obtížných algoritmických problémů je nyní již vyřešena a jejich řešení jsou zařazena do běžně dostupných knihoven. Současné programování již dávno opustilo doby, kdy jsme je mohli zařazovat mezi umění a přesunulo se do kategorie technologií. Základní dovedností, kterou musí současný programátor ovládat, není vymýšlení nových postupů, ale především správná aplikace postupů dříve vymyšlených.

Dosavadní historie vývoje nejrůznějších programovacích nástrojů a knihoven však přinesla řadu nových poznatků a zkušeností. Jedněmi z nejdůležitějších bylo docenění významu znovupoužitelnosti a snadné modifikovatelnosti dříve vytvořených programů. Znovupoužitelnost a snadná modifikovatelnost jsou v současné době klíčovými zaklínadly všech zkušených programátorů a manažerů softwarových projektů.

Optimální by bylo, kdyby se stejně jako vlastní programování vyvíjela i metodika jeho výuky. Metodika výuky se však ve většině oblastí za vývojem daného předmětu výrazně opožďuje. Sebekriticky si přiznejme, že mnozí učitelé často připravují žáky na styl programování, který možná byl před 20 lety progresivní, ale v současné době je již dávno překonaný. O to překonanější bude v době, kdy budou jejich studenti vstupovat do praxe.

V současné době se při vývoji prakticky všech velkých projektů používá objektově orientované programování. Prakticky všechny vysoké školy a s nimi i značná část středních škol proto učí své studenty programovat objektově. Na špičkových vzdělávacích pracovištích a konferencích pedagogů se již neřeší otázka, zda učit objektově orientované programování, ale jak je učit. [4][3]

2.3. Výběr vysokých škol

Jedním z možných způsobů, jak se seznámit s dnešními metodami výuky algoritmizace a programování na vysokých školách, je prostudování jejich sylabů. K výběru vysokých školy byl použit následující klíč:

1. vysoká škola
2. program ekonomického zaměření
3. předmět algoritmizace a programování

4. předmět určený pro začátečníky

ad 1 Výběr byl zúžen na vysokoškolské vzdělání z důvodu tématu práce.

ad 2 Jelikož se ekonomické obory vyučují i na fakultách s jiným než s ekonomickým zaměřením, bylo nutné výběr zúžit na konkrétní studijní program a ne na fakultu, která by výběr značně omezila.

ad 3 Jedná se o předmět algoritmizace a programování.

ad 4 Kurz algoritmizace a programování by měl být pro úplné začátečníky programování studenti by měli být základními uživateli PC.

Tyto faktory by neměly ovlivnit konečný výběr, proto je klíč neobsahuje:

- zda se jedná o kombinované studium či prezenční studium,
- pro jaký stupeň vzdělání je kurz určen (magisterský, bakalářský, atp.),
- způsob zakončení předmětu (zkouška či zápočet, ústně nebo písemně),
- rozsah hodin vyučovaných týdně,
- název předmětu nemusí být totožný s klíčem (ad 3) jde o obsah výuky.

Po aplikaci uvedeného klíče zůstalo šest vysokých škol viz Tabulka 1.

Tabulka 1 - Vybrané VŠ a vyučované jazyky Zdroj:Vlastní

Název vysoké školy	Fakulta	Studijní program	Název předmětu	Programovací jazyk použitý v kurzu
Česká zemědělská univerzita	Provozně ekonomická fakulta	Systémové inženýrství a informatika	Algoritmizace a programování	Borland C
Jihočeská univerzita	Ekonomická fakulta	Systémové inženýrství a informatika	Algoritmy a datové struktury I	JAVA
Mendelova zemědělská a lesnická universita	Provozně ekonomická fakulta	Inženýrská informatika	Algoritmizace	Pascal
Technická univerzita	Fakulta mechatroniky a mezioborových inženýrských studií	Aplikované vědy a informatika	Algoritmizace a programování 1	JAVA
Univerzita Hradec Králové	Fakulta informatiky a managementu	Aplikovaná informatika	Algoritmy a algoritmizace	-
Univerzita Pardubice	Fakulta ekonomicko – správní	Systémové inženýrství a informatika	Algoritmizace a programování	Pascal

Zde je vidět velká škála programovacích jazyků při výuce programování na vysokých školách. Na dvou vysokých školách je jako nástroj používána JAVA, na dvou vysokých školách Pascal, na jedné je to Borland C a v Hradci Králové se k výuce přistupuje metodou bez použití nástroje. Z výsledků tedy vyplývá, že není důležité jakým nástrojem se algoritmizace vyučuje, ale jak je vyučována.

2.4. Cíle výuky vybraných škol

V této části si pro jednodušší orientaci uvedeme cíle uvedené v sylabu jednotlivých škol. Může nám to pomoci v konečné fázi při návrhu vhodného způsobu výuky algoritmizace a programování.

Česká zemědělská univerzita

Cílem předmětu je seznámit posluchače se základy algoritmizace, ozřejmit způsob sestavování algoritmu členěním problému na dílčí kroky, ukázat ztvárnění algoritmu ve zvoleném programovacím jazyce (Borland C). [5]

Jihočeská univerzita

Úvodní kurz, který seznamuje studenty se základními algoritmy a datovými strukturami a technikami, které se využívají při analýze asymptotické časové a paměťové náročnosti. Kurz ideově vychází z pojmu ADT (abstract data type) a při implementaci se využívají návrhové vzory v programovacím jazyku Java. [6]

Mendelova zemědělská a lesnická universita

Osvojení si zásad řešení problému s využitím počítačů. Algoritmizace je prezentována jako způsob myšlení, je orientována do oblasti jak vědecko-technických výpočtů, tak i hromadného zpracováni dat. Vedlejším cílem předmětu je studenta naučit používat programovací jazyk Pascal jako prostředek pro zápis algoritmu. [7]

Technická univerzita

Předmět je úvodem do problematiky programování v programovacím jazyce vyšší úrovně. Studenti se seznámí se základními postupy při algoritmizaci úloh a s realizací algoritmů

zaměřených na zpracování čísel a jejich posloupností pomocí výrazových prostředků programovacího jazyka Java. [8]

Univerzita Hradec Králové

Uvědomit si potřebu jasné formulace úlohy a způsobu řešení. Vyjádření algoritmu ve formě vývojového diagramu. Vztah program a data. Některé obecné pojmy vztahující se k programování. postup při tvorbě programu. [9]

Univerzita Pardubice

Cílem předmětu je seznámit studenty se základy informatiky a technického vybavení počítačů, základy zápisu pomocí vývojových diagramů a základy programovacího jazyka Pascal. [10]

3. Softwarové vybavení pro výuku algoritmizace

Tato kapitola popisuje, jaký programovací jazyk si vybrat pro účely výuky, dále jazyky popisuje obecně a v poslední části samotné programovací jazyky testuje.

3.1. Jaký programovací jazyk si vybrat

Na začátek je uvedeno devatero jednoduchých zásad, které by nám měli pomoci k výuce algoritmizace a programování. Podle [11] jsou to tyto zásady:

- Co nejdříve umožnit tvorbu programů.
- Nepředbíhat, tj. nepoužívat prvky jazyka, které ještě nebyly vyloženy.
- Informace je třeba předávat po malých „soustech“.
- Doprovodné příklady musí vyžadovat aktivní použití nových poznatků.
- Příklady musí být zajímavé.
- Řešení nesmí být příliš zašuměná. (Šumem jsou myšleny části programu, které nesouvisí bezprostředně s logickou řešení, ale bez jejichž přítomnosti program neběží.)
- Od samotného začátku je třeba vštěpovat zásady moderního programování.
- Studenti se musí naučit programy nejen vytvářet, ale také ladit.
- Předkládat řešení netriviálních problémů.

Než se začneme zamýšlet nad tím, jak tyto zásady naplnit, měli bychom se zamyslet nad tím, zda se chystáme učit programování nebo pouze programovací jazyk. V druhém případě je naše situace mnohem jednodušší. Můžeme předpokládat, že naši žáci již programovat umějí a budeme mít proto daleko svobodnější volbu ve výběru jazyka, prostředí i příkladů.

Typickým příkladem učebnic, které jsou určeny programátorům s jistými počátečními zkušenostmi, jsou Eckelovy knihy Thinking in xxx (z nich jsou do češtiny přeložené Myslíme v C++ a Myslíme v jazyku Java), ve kterých autor ve svých příkladech opravdu neukazuje nic jiného, než to, jak jazyk přesně funguje. Autor v nich totiž předpokládá, že jeho čtenáři již umějí programovat a že je tudíž při studiu jeho knih opravdu nezajímá nic jiného, než přesná funkce a způsob zápisu jednotlivých konstrukcí jazyka, ale ty chtějí vysvětlit do všech detailů. A má pravdu. Proto jeho knihy vyhrály několik cen a drží se na špici hitparád. [11]

Chcete-li se však věnovat výuce základů programování, musíte mít neustále na paměti, že vaši čtenáři či posluchači nemají s programováním doposud žádné zkušenosti a že značná část toho, co jim vysvětlujete, je pro ně tak trochu zprávou z jiného světa. Proto musíte všechny výše uvedené zásady úzkostlivě dodržovat. To ale automaticky ovlivňuje výběr programovacího jazyka i vývojového prostředí.

Všichni asi víte, že programovací jazyky přicházejí a odcházejí. Zásady tvorby dobrého programu jsou však pro všechny jazyky (včetně těch exotických) v podstatě stejné. Pokud se proto někdo naučí dobře programovat v jednom jazyce, můžeme očekávat, že bude dobře programovat i v jazyce, na který přejde posléze. Z toho ale vyplývá velice příjemný závěr:

Při výběru programovacího jazyka, jehož prostřednictvím budou studenti seznamováni se světem programování, se nemusí vyučující omezovat na jazyky, které budou studenti používat ve své další praxi, ale může jazyk vybrat tak, aby se s jeho pomocí studenti co nejsnadněji a co nejlépe naučili potřebné základy. Z něj pak může posléze přejít na jazyk, s nímž se budou posléze setkávat ve své praxi. Na vstupní programovací jazyk je jediná podmínka: aby přechod z tohoto vstupního jazyka na jazyky běžně používané byl co nejjednodušší a nejpřirozenější. [11]

3.2. Přehled programovacích jazyků

Zde je uveden stručný popis jednotlivých programovacích jazyků, které se vyučují na vybraných vysokých školách (viz Tabulka 1). Je zde také přidán jazyk Logo z důvodu toho, že se nevyskytuje v žádném sylabu ekonomicky zaměřeného programu. Jelikož se s ním obecně algoritmizace a programování vyučuje.

3.2.1. Pascal

Návrh programovacího jazyka Pascal pochází ze začátku 70. let od profesora Niklause Wirtha z Vysoké školy technické v Curychu. Autor sledoval návrhem jazyka tyto cíle:

- vytvořit jazyk vhodný pro výuku programování, který by byl založen na omezeném počtu srozumitelných konstrukcí,
- navrhnout strukturu jazyka tak, aby bylo snadné implementovat Pascal na většině tehdejších počítačů.

První verze Pascalu byla publikována r. 1971, o 3 roky později (1974) byla uveřejněna opravená definice jazyka. V roce 1981 byla vydána norma ISO. Vedle toho vznikla řada komerčních implementací Pascalu, které se od Pascalu dle normy ISO více či méně odchylovaly, zejména zavedením dalších konstrukcí zjednodušujících praktické programování. V oblasti PC dosáhla patrně největšího úspěchu implementace Turbo Pascal firmy Borland. Objektové rozšíření Pascalu se pak stalo i základem systému Delphi téže firmy. [12]

3.2.2. C

Jazyk, který patří naprosto neodmyslitelně k UNIXu. Vytvořili jej Ken Thompson a Dennis Ritchie (jejich verze se označuje jako K&R C) právě pro potřeby vývoje prvního unixu. C je jazyk podporující strukturované programování, umožňuje přímý přístup do paměti pomocí ukazatelů a aplikace v něm jsou velmi kompaktní. Hodí se na systémové i aplikační programování a navíc v něm bývají napsány interpretery vyšších jazyků, o kterých zde bude také zmínka. V C je napsán třeba MPlayer, Linux (=jádro), gcc a webový server Apache. [13]

3.2.3. Java

Nazývaná také Cobolem[2] 21. století. Jazyk vytvořil James Gosling. Jeho syntaxe vychází z jazyka C++, stejně jako on je objektově orientovaný, ovšem Java dále poskytuje virtuální stroj s automatickou správou paměti, velice silnou základní knihovnou a faktickou platformní nezávislost, protože překladač překládá do mezikódu (bytecode), který je teprve potom prováděn virtuálním strojem. Java je silná v programování pro web, v rozsáhlých systémech, nebo naopak v malých zařízeních (mobilní telefony) a trochu méně v psaní běžných aplikací. V Javě je napsán třeba jEdit, Apache Tomcat, Jboss nebo Azureus. [13]

3.2.4. LOGO

Logo je jednoduchý funkcionální programovací jazyk, který byl navržen ve firmě BBN (Cambridge, Massachusetts) v roce 1967 původně pro výuku myšlení, ale je spojen především s výukou programování dětí. Logo je vhodným nástrojem i pro výuku a implementaci technik umělé inteligence.

V červenci 2008 existovalo 187 implementací jazyka Logo, každý s různou kvalitou. Na českých a slovenských školách jsou často používány verze Imagine Logo a Comenius Logo. Logo řadíme mezi dětské programovací jazyky, kam se řadí ještě Karel, Baltík, Petr a další. [14]

3.2.5. Petr

Petr je vizuální programovací nástroj určený k snadné a rychlé tvorbě programů pro Windows 95/98/NT/ME/2000/XP. Jeho hlavní charakteristikou je grafické vyjádření struktury programu. Části programu se skládají pomocí myši k sobě jako skládačka. Díky kontrole smysluplnosti kombinací prvků již při vytváření programu odpadá jakákoliv možnost vzniku syntaktické chyby. Vyjádření programu stromovou strukturou přináší radikální zvýšení přehlednosti programu. Tvorba programu se stává nebývale snadnou, přehlednou a pružnou.

Petr je určen pro nejširší okruh uživatelů. Již děti předškolního věku si v něm kreslí obrázky a učí chodit svého králíčka. Pro zkušené programátory je tu naopak připraveno velké množství bohatých funkcí. Petr si neklade za cíl vychovávat ze všech uživatelů programátory. Snaží se zpřístupnit tvorbu programů každému, kdo chce tvořit. Nejsou

[2] COBOL (COmmon Business Oriented Language) vznikl v roce 1959. Snaha o vytvoření jazyka byla inicializována potřebou jazyka, který umožní vytvoření programu v minimálním časovém úseku a s minimálním úsilím. Zápis programu v jazyce, který je blízký angličtině.

zapotřebí žádné znalosti z oblasti programování ani z oblasti počítačů. Vše je zajištěno nepřeberným množstvím funkcí. Je to účinný prostředek k procvičování logického myšlení, představivosti a estetického cítění. [15]

3.3. Testování programovacích jazyků

V této kapitole budou, jak již nadpis napovídá, testovány programovací jazyky. Pro porovnání jazyků a jejich softwaru bude v každém jazyku naprogramován stejný program. Každý software jazyka bude testován z těchto hledisek:

1. uživatelské prostředí
2. čitelnost kódu
3. ovládání - intuitivnost/snadnost osvojení
4. ladění chyb

Každé hledisko bude hodnoceno na škále od 1 – 5 jednotlivá čísla budou znamenat obdobně jako ve škole:

1. vynikající
2. dobré
3. průměrné
4. uspokojivé
5. špatné

V testování se nebudou zohledňovat tyto hlediska:

- Dostupnost (zda je software volně ke stažení a nebo se musí obědnávat z kamenného obchodu)
- Kupní cena (zda je software zdarma nebo je potřeba zakoupit licenci)
- Zda existuje více druhů softwaru pro jednotlivé programy. Počítáme s tím, že rozdíly, které mají softwary jsou z celkového hlediska marginální

Pro účely testování byly vybrány tyto nástroje:

Tabulka 2 - Software pro programovací jazyky Zdroj: vlastní

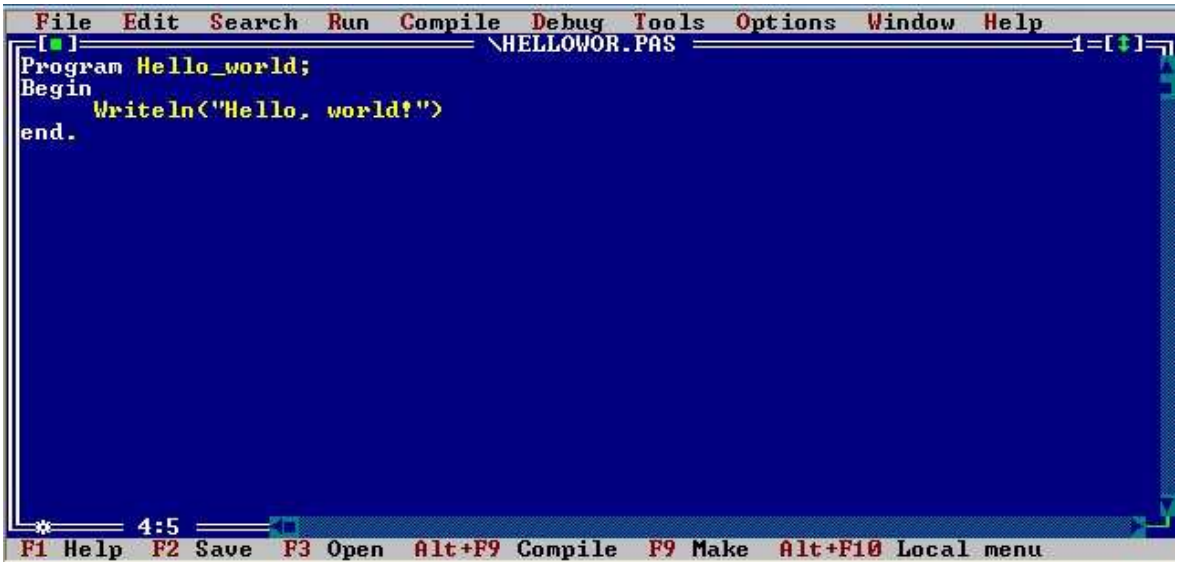
Programovací jazyk	Název softwaru	Verze softwaru	Výrobce
Pascal	Turbo Pascal 7.0	7.0	Borland International, Inc.
C	Lcc-win32 Wedit	3.3	Jacob Navia
JAVA	BlueJ	2000-2007	Michael Kolling, John Rosenberg

Logo	MSWLogo	6.5b	Softronics, Inc.

Pro účely testování byly vybrány tyto programy k naprogramování:

- Prvním programem je tzv. „Hello, world!“ Tento program jest programátorskou klasikou. Jedná se zde o text, který je programem vypsán na obrazovku.
- Možnost sestrojení trojúhelníku podle zadaných tří stran podle „věty sss“ Zde program určí zda lze trojúhelník sestrojit a pokud ano o jaký typ trojúhelníku se jedná.

3.3.1. PASCAL



Obrázek 1 - Kód Pascal "Hello, world!" Zdroj: Vlastní

Kód programu „sss“

```
program Trojuhelnik;
uses crt;
var
  a,b,c : integer;

begin
 clrscr;
 writeln('Zadej strany trojuhelniku a,b,c');
 readln(a);
 readln(b);
 readln(c);
 clrscr;

 IF (a = b) and (b = c) and (a = c) THEN
  begin
    writeln('Rovnostranny');
  end;
```

```
 IF ((a = b) and (a <> c))
 or ((a = c) and (a <> b))
 or ((b = c) and (a <> c)) THEN
  begin
    writeln('Rovnoramenny');
  end;

 IF ((a * a) + (b * b) = (c * c))
 or ((a * a) + (c * c) = (b * b))
 or ((b * b) + (c * c) = (a * a)) THEN
  begin
    writeln('Pravouhly');
  end;

 IF (a + b < c) or (a + c < b) or (b + c < a) THEN
  begin
    writeln('Neexistuje');
  end;
 readln;
end.
```

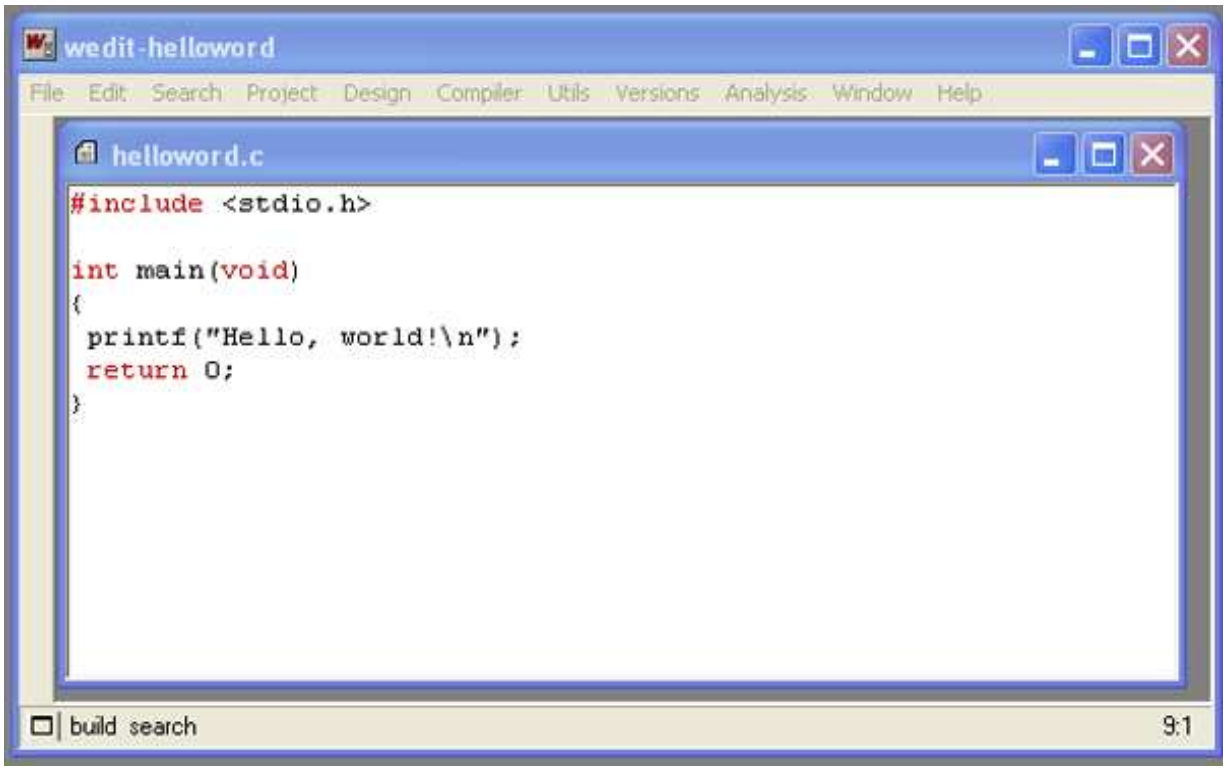
uživatelské prostředí - 3
Uživatelské prostředí tohoto programu vypadá sice zastarale, ale díky své jednoduchosti je pro uživatele přehledné a jednoduché. Prostředí je na dnešní dobu, kdy vládne při ovládání většiny programů myš, poněkud těžkopádné, ale po prvních pár minutách uživatel zjistí, že myš není potřeba.

čitelnost kódu - 3
Čitelnost kódu je s ostatními programovacími jazyky na stejné úrovni, po několika prvních programech se dá v syntaxi docela jednoduše orientovat.

ovládání - intuitivnost/snadnost osvojení - 3
Jak jsem psal již výše, přehlednost programu je na dnešní dobu poněkud těžkopádná. Naštěstí se dají používat jednoduché klávesové zkratky pro rychlé používání funkcí.

ladění chyb - 2
Pokud je při kompilování programu nalezena chyba v syntaxi, je vcelku přesně identifikována a je zobrazeno na jakém místě se chyba nachází. Tedy při jednodušších chybách se vypíše např. co jest na místě očekáváno. Složitější chyby však uživatel musí bez jakékoli další nápovědy sám odstranit.

3.3.2. C



Obrázek 2 - Kód C "Hello, world!" Zdroj: Vlastní

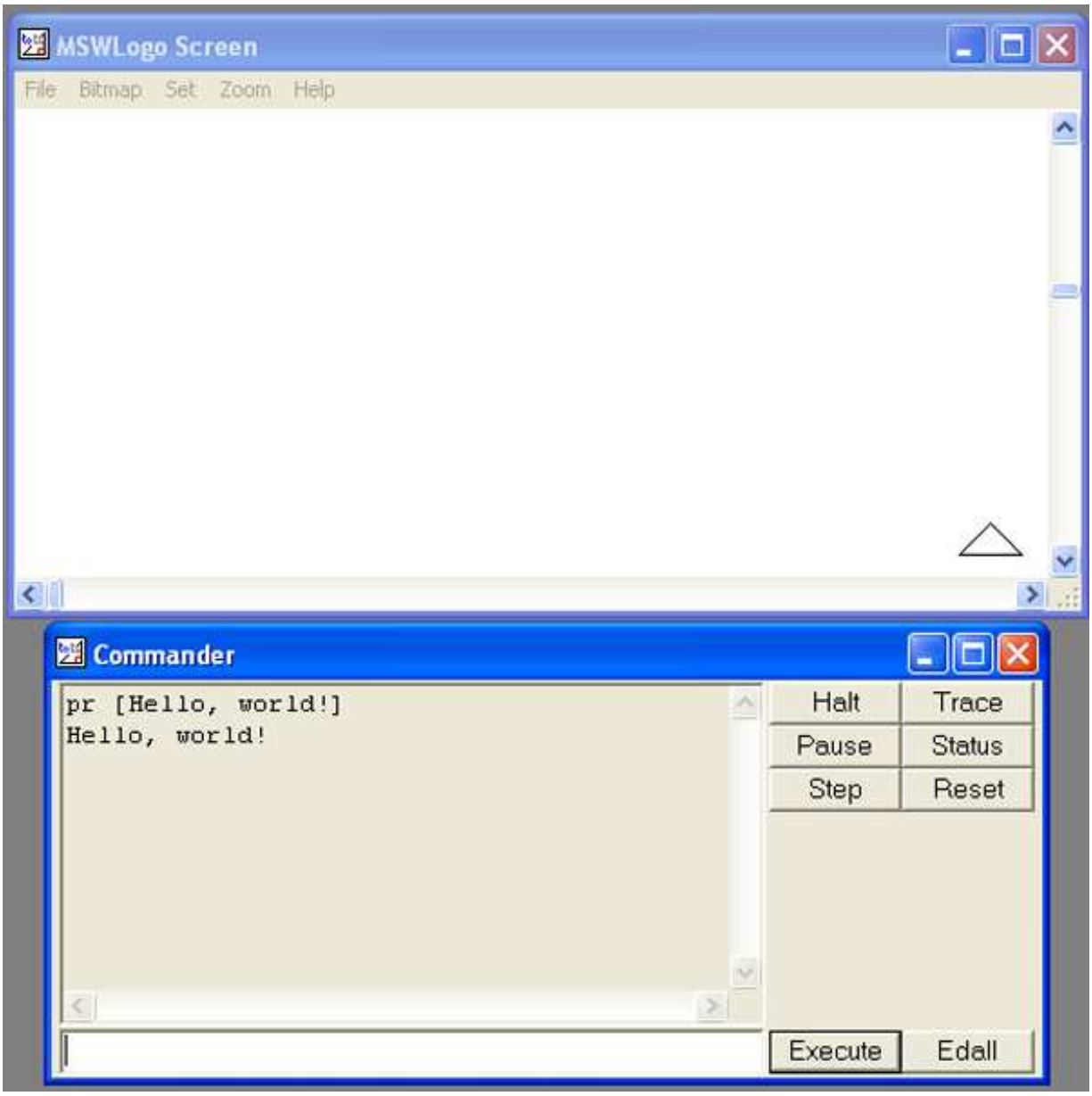
uživatelské prostředí - 4
Na dnešní dobu, ve které vývojový nástroj vznikl, je uživatelské prostředí opravdu odstrašující. Sice vyniká jednoduchostí, ale chybí zde nějaká lišta s nástrojikterá by práci studentovi usnadňovala.

čitelnost kódu - 3
Kód je vzhledem ke stáří jazyka na stejné úrovni s ostatními obdobnými jazyky. a

ovládání - intuitivnost/snadnost osvojení - 3
Ovládání programu je i přes jeho nedostatky popsané výše uspokojující. Jednoduše uživatel nemá problém nají požadované ovládací prvky, protože zde není nic nadbytečného.

ladění chyb - 5
Chyba v syntaxi je programem nalezena, ale je zde jeden velký nedostatek. Po nakompilování programu do paměti se pamět při znovukaompilování novými daty paměť nepřehraje. A zůstává tedy první nakopilovaný obsah. Pamět se nevyčistí ani po restartu aplikace. Tak si může připadat uživatel neznalý tohoto programu. Je složité načítat další program tak, aby mohl být slušně spuštěn.

3.3.3. LOGO



Obrázek 3 - Kód Logo "Hello, world" Zdroj: Vlastní

Kódy na vytvoření trojúhelníků

```
TO Lichobeznik
CS
FD 100
RT 150
FD 50
HOME
END

TO rovnoramenny
CS
RT 20
```

```
FD 50
RT 90
FD 50
HOME
END
```

```
TO rovnostranny
CS
FD 100
RT 120
FD 100
RT 120
FD 100
 END
```

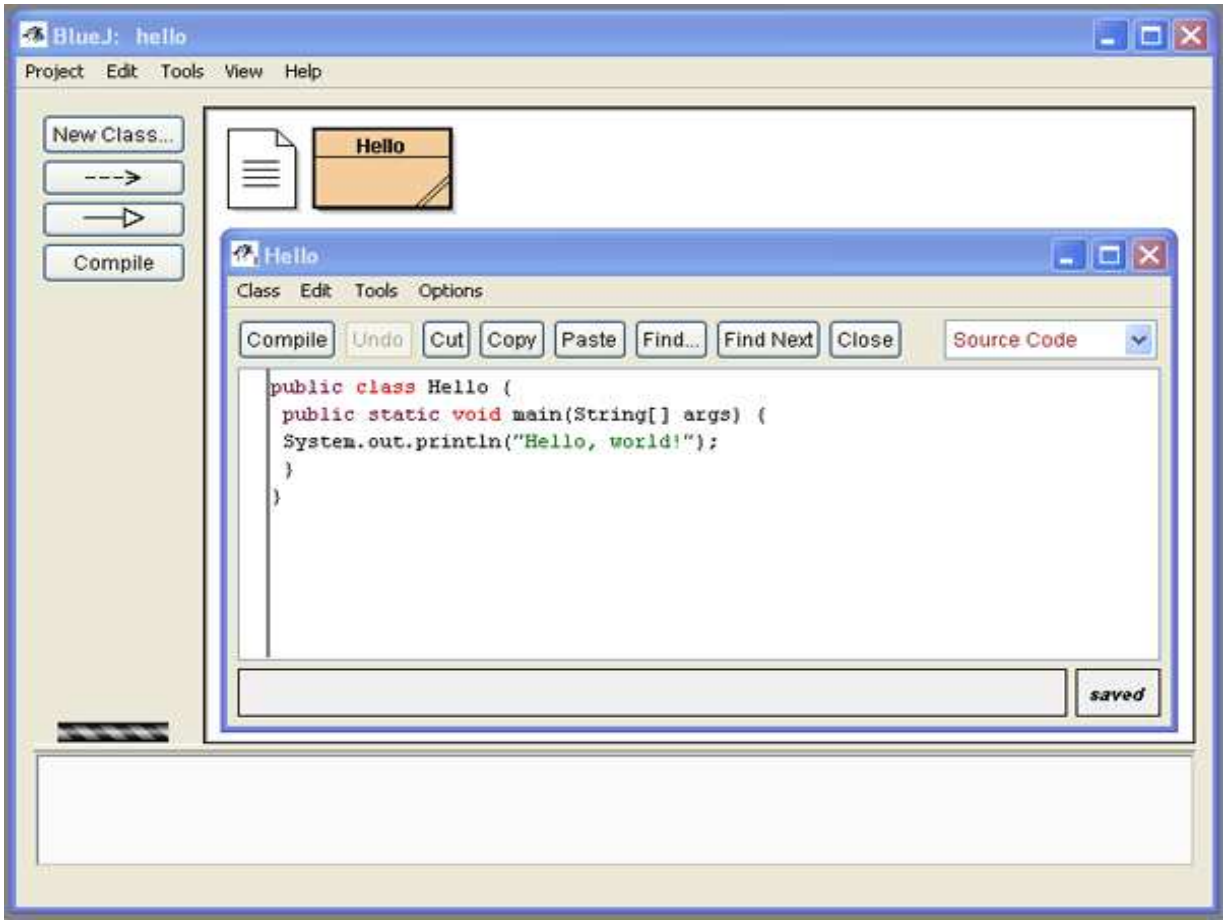
uživatelské prostředí - 3
Svou jednoduchostí ani nenadchne ani nezklame.

čitelnost kódu - 3
Snadnější kódový zápis zde ještě nebyl. Syntaxe programu zde bohužel není znázorněna graficky a to bohužel znehodnocuje jeho využití. Při vytváření programu „sss“ bylo ustoupeno prostředí Loga a program byl upraven.

ovládání - intuitivnost/snadnost osvojení - 2
Ovládací prvky jsou na místech kde by je uživatel očekával. Jednoduchost programu je inspirující uživatel má po chvíli práce s programem chuť zkoušet více a více. Zde není jednoduchost na škodu , ale naopak v jednoduchosti je síla.

ladění chyb – 5
Když program nemůže dál, tak program nemůže dál. Zde je veliký nedostatek, který znehodnocuje použitelnost programu.

3.3.4. JAVA



Obrázek 4 - Kód JAVA "Hello, world!" Zdroj: Vlastní

Kód programu „sss“

```
import java.util.*;

public class troj {
   static {Locale.setDefault(Locale.US);}

   static double zadaniStran(String y) {
      try {
         double x;
         Scanner sc = new Scanner (System.in);
         System.out.println("Zadej stranu " +y);
         x = sc.nextDouble();
         while (x <= 0) {
            System.out.println("Strana musi byt vetsi nez 0. Zadejte znovu");
            x = sc.nextDouble();
         }
         return x;
      } catch (InputMismatchException e) {
            return -1;
      }
   }
```

```
   static void pocitani() {
      double a = 0,b = 0,c = 0;
      int i = 1;
      a = zadaniStran("a");
      while (a == -1) a = zadaniStran("a");
      b = zadaniStran("b");
      while (b == -1) b = zadaniStran("b");
      c = zadaniStran("c");
      while (c == -1) c = zadaniStran("c");
      double x,y,z;
      double[] pole = {a, b, c};
      Arrays.sort(pole);
      x = pole[0];
      y = pole[1];
      z = pole[2];

      if (x + y >= z) {
         System.out.print("trojuhelnik existuje");

         if (x == y || x == z || z == y) {
            if (x == y && x == z) System.out.println(" a je rovnostranny");
            else System.out.print(" a je rovnoramenny");
         }

         if (Math.sqrt((x*x) + (y*y)) == Math.sqrt(z*z)) System.out.println(" a take je
pravouhly");
      }
      else System.out.println("Trojuhelnik neexistuje");
   }

   public static void main(String[] args) {
      pocitani();
   }

}
```

uživatelské prostředí - 3

Opět se setkáváme se zcela chudým uživatelským prostředím ve srovnání s datem vzniku programovacího nástroje. Při hlubší práci s programem uživatele nic nenadchne, ani nic výrazně nezklame.

čitelnost kódu - 3

Zdrojový kód se stává o něco více složitějším, ale při delší práci a větších zkušenostech to již obtíže nedělá. Syntaxe je opět zvládnutá skvěle.

ovládání - intuitivnost/snadnost osvojení - 4

Zde je na uživateli požadováno větší soustředění.

ladění chyb – 3

Zde je ladění chyb na docela běžnéí úrovni. Pokud nastal problém v syntaxi program upozorní na problematické místo.

Tabulka 3 - Výsledek testování programů Zdroj: Vlastní

Programovací jazyk	uživatelské prostředí	čitelnost kódu	ovládání - intuitivnost/snadnost osvojení	ladění chyb	celkem
Pascal	3	3	3	2	11
C	4	3	3	5	15
JAVA	3	3	4	3	13
Logo	3	3	2	5	13

Tabulka 3 ukazuje, že žádný z testovaných programů nijak významně nevyčnívá. To podtrhuje teorii, že při výuce nezáleží jakým nástrojem se bude vyučovat, ale jak se bude algoritmizace a programování vyučovat.

4. Návrh koncepce způsobu výuky algoritmizace

V poslední kapitole se výše nabyté informace použijí na zvolení koncepce způsobu výuky algoritmizace a programování na vysokých školách.

4.1. Zvolení cíle výuky

Cílem výuky by nemělo být naučit programovat v nějakém konkrétním jazyce, ale naučit se programování jako takovému. A to nejen konkrétně naprogramovat program. Ale ve vhodných případech jako programátor uvažovat. Snažit se k problému přistupovat podobně jako při tvorbě algoritmu programu na počítači.

4.2. Zvolení způsobu výuky a jazyku

Z předchozích kapitol je jednoznačná volba přístupu k výuce programování. Zcela jistě nelze počítat s tím, že student bude mít chuť učit se programování bez programování. Tím odpadá přístup „Algorithms-first“. „Hardware-first“ a „Breadth-first“ nejsou moc vhodné pro obecnou výuku programování pro jejich široké pojmutí programování. Ze zbývajících tří přístupů není žádný nejvhodnější. Proto je zde zvolen vlastní přístup „Breadth &

programm - first “ je to zcela jednoduché určitě začít zeširoka kvůli pozdějšímu nadhledu při řešení problémů.

Neznamená to však zanedbávat praktické programování. Z jazykem Logo se studentova mysl nezatíží mnohými podmínkami syntaxe atd a může nplnou rozvinout svůj potenciál. Při volbě jiného jazyka než Logo hrozí studentovo odmítnutí zajímat se o přemět jako takový kvůli kvantům informací, které si musí zapamotvat aby mohl slušně cokoli naprogramovat.

.

4.3. Sestrojení sylabu

Z nasbíraných informací je sestaven „obecný sylabus“ předmětu Algoritmizace a programování

Tabulka 4 - Sylabus

Algoritmizace a programování - 2008	
Kredity :	
Akreditace :	
Garant :	
Rozsah :	Přednáška 2 [HOD/TYD], Cvičení 2 [HOD/TYD]
Zápočet před zkouškou :	Ano
Způsob ukončení/forma zkoušky :	Zkouška
Cíle předmětu	
Cílem předmětu je seznámit studenty se základy informatiky a technického vybavení počítačů, základy zápisu pomocí vývojových diagramů a základy programovacího jazyka Logo. Student by měl být stimulován k samostatnému řešení problému.	
Požadavky na studenta	
Zápočet: Semestrální práce - vypracovat a obhájit samostatně vytvořený a odladěný program na vybrané téma.	
Přehled probírané látky	
Úvod, vymezení pojmu počítač, počítačový systém. 1,5 Technické a programové vybavení počítačů. 1,5 Postup řešení úloh na počítači. 2 Algoritmizace úloh. 2 Základní pojmy v algoritmizaci a programování. 2	

V sylabus vychází ze sylabu z Univerzity pardubice. Je však zásadně odlišný ve dvou věcech. Jazyk ve kterém je vyučován .Ale hlavě co není na první pohled patrné. To je rozložení výuky do semestru. V přehledu probírané látky je naznačeno kolik týdnů by se mělo věnovat jaké oblasti. Na programovací jazyk zbývá největší část je to pro to, aby se daly řešit i složitější úlohy, které by měla většina studentů bez obtíží zbvládnout.

Závěr

V práci byla zmíněna stručná historie programovacích jazyků a dělení programovacích jazyků podle různých kritérií. Byly nastíněny základní metody používané při výuce algoritmizace a programování a bylo zjištěno jaké programovací jazyky se k výuce programování používají na školách s ekonomickým studijním programem. Nepodařilo se sice najít jeden nejlepší univerzální způsob pro výuku algoritmizace a programování, protože škála je velice široká a každý způsob výuky má svá pro a proti, ale v závěru práce byl navržen jeden konkrétní koncept na výuku algoritmizace a programování v podmínkách ekonomického vzdělání.

Literatura

[1] *Historie programovacích jazyků* [online]. 2006 [cit. 2008-04-04]. Dostupný z WWW: <http://www.fi.muni.cz/usr/jkucera/pv109/2000/xkrubova.htm>.

[2] *Programovací jazyk* [online]. 2002 [cit. 2008-07-07]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Programovací_jazyk>.

[3] PECINOVSKÝ, Rudolf. Proč učit programování na základní škole. *Česká škola* [online]. 2001 [cit. 2008-09-08]. Dostupný z WWW: <http://www.ceskaskola.cz/ICTveskole/AR.asp?ARI=2868&CAI=2129>.

[4] PECINOVSKÝ, Rudolf. Současné trendy v metodice výuky programování. *Česká škola* [online]. 2006 [cit. 2008-04-07]. Dostupný z WWW: < http://www.ceskaskola.cz/ICTveskole/Ar.asp?ARI=102900&CAI=2129>.

[5] *Výpis předmětů* [online]. 2008 [cit. 2008-08-14]. Dostupný z WWW: <http://wp.czu.cz/cs/index.php/?r=1067&mp=subjects.info&idPredmet=2191>.

[6] *Popis předmětu KIN/ADS1* [online]. 2008 [cit. 2008-08-14]. Dostupný z WWW: <http://stag-web.jcu.cz/apps/stag/prohlizeni/pg$_prohlizeni.sylabus?kat=KIN&predm=ALG1&rok=2008>.

[7] *Sylabus předmětu ALG - Algoritmizace* [online]. 2007 [cit. 2008-08-14]. Dostupný z WWW: <http://is.mendelu.cz/katalog/syllabus.pl?predmet=36714>.

[8] *Popis předmětu MTI/ALP1* [online]. 2008 [cit. 2008-08-14]. Dostupný z WWW: <http://stag.tul.cz/apps/stag/prohlizeni/pg$_prohlizeni.sylabus?kat=MTI&predm=ALP1&rok=2008>.

[9] *Sylabus ALGORIT - Algoritmy a algoritmizace* [online]. 2008 [cit. 2008-08-14]. Dostupný z WWW: <http://lide.uhk.cz/pdf/ucitel/dusekfr1/IPRG1/IPRG1_sy.htm>.

[10] *Studijní agenda UPa* [online]. 2000 [cit. 2008-08-14]. Dostupný z WWW: <http://stag.upce.cz/apps/stag/prohlizeni/pg$_prohlizeni.sylabus?kat=USII&predm=PALG&rok=2008>.

[11] PECINOVSKÝ, Rudolf. Jak vybírat programovací jazyk, v němž budeme učit. *Česká škola* [online]. 2002 [cit. 2008-07-04].

[12] *Pascal (programovací jazyk)* [online]. 2008 [cit. 2008-07-07]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Pascal_(programovac%C3%AD_jazyk)>.

[13] *Který programovací jazyk si vybrat?* [online]. 2008 [cit. 2008-08-07]. Dostupný z WWW: <http://www.linuxexpres.cz/praxe/ktery-programovaci-jazyk-si-vybrat>.

[14] *LOGO (Programovací jazyk)* [online]. 2008 [cit. 2008-07-06]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Logo_(programovac%C3%AD_jazyk)>.

[15] Gemtree Software, s.r.o.. *Gemtree Software - vizuální programovací nástroj Petr* [online]. 1999 , 23. ledna 2007 [cit. 2008-5-13]. Dostupný z WWW: <http://www.gemtree.cz/>.

[16] Wikimedia.*Wikipedie, otevřená encyklopedie* [online]. 2002 [cit. 2008-08-17]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Pascal_(programovací_jazyk)>.

[17] Wikimedia. *Wikipedie, otevřená encyklopedie* [online]. 2002 [cit. 2008-07-07]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Programovací_jazyk>.

[18] *Informačního systém Masarykovy univerzity* [online]. 2004 [cit. 2008-08-14]. Dostupný z WWW: <http://is.muni.cz/predmety/predmet.pl?fakulta=1433;obdobi=2824;kod=IB001>.

[19] Fakulta informačních technologií VUT. *Fakulta informačních technologií VUT* [online]. 2007 [cit. 2008-08-14]. Dostupný z WWW: <http://www.fit.vutbr.cz/study/course-l.php.cs.iso-8859-2?id=5525>.

[20] Max&PJ. *Studijní agenda UTB* [online]. 2007 [cit. 2005-08-14]. Dostupný z WWW: <http://www.stag.utb.cz/apps/stag/prohlizeni/pg$_prohlizeni.sylabus?kat=UAI&predm=AQPCJ&rok=2007>.

[21] Oddělení podpory a rozvoje informačních systémů Výpočetního centra VŠE. *PES VSE* [online]. 2005 [cit. 2008-08-14]. Dostupný z WWW: <http://pes.vse.cz/main.php?action=PRED_INFO&id_predmetu=4IT101>.

[22] Max&PJ. *Studijní agenda UPa* [online]. 2008 [cit. 2008-08-14]. Dostupný z WWW: <http://stag.upce.cz/apps/stag/prohlizeni/pg$_prohlizeni.sylabus?kat=KID&predm=DAZPP&rok=2008>.

Seznam obrázků

Seznam tabulek

Přílohy

Příloha 1: Česká zemědělská univerzita – sylabus zdroj [5]
Algoritmizace a programování

Zkouška: zápočet a kombinovaná zkouška
Předpoklady:
Výpočetní systémy
Cíl předmětu:
Cílem předmětu je seznámit posluchače se základy algoritmizace, ozřejmit způsob sestavování algoritmu členěním problému na dílčí kroky, ukázat ztvárnění algoritmu ve zvoleném programovacím jazyce (Borland C)

Program přednášek

1. Úvod, organizace. Číslo v matematice a v počítači.
2. IT - rozjetý vlak.
3. Algoritmus jako základ programování.
4. Počítač, HW, SW. Paměť, data, instrukce, adresy.
5. Data. Jednoduché proměnné.
6. Instrukce. Přiřazení, testy, přístup na data.
7. Programové struktury. Složený příkaz, podprogram, repetice, cyklus, iterace, rekurze.
8. Datové struktury. Pole, záznam, řetězec.
9. Knihovny, operační systém. Systémové zdroje a manipulace s nimi.
10. Rekapitulace 1. Souvislosti, o nichž jsme nehovořili. Objektové programování.
11. Data. Vyhledávání, řazení. Lineární spojové seznamy, stromy.
12. Spojové seznamy, vkládání, vyjímání, řazení.
13. Rekapitulace 2. Co bychom neměli zapomenout.
14. Rezerva.

Program cvičení

1. Počítač a jak to vypadá uvnitř, IDE-C část 1.
2. Zápis algoritmu, interpretace čísla v počítači
3. Sestavování algoritmu, IDE-C část 2.
4. Nácvik čtení strukturogramu, programátorská fantazie při sestavování algoritmu
5. Malá písemka, IDE-C opakování.
6. Manipulace s daty, IDE-C sledování činnosti programu.
7. Zadání semestrální práce.
8. Textové řetězce.
9. Příklad čtení vstupních parametrů a manipulace se soubory.
10. Příprava na velkou písemku.
11. Příklad lineárního spojového seznamu na pevném poli (index místo pointru).

12. Velká písemka.
13. Druhá šance na získání zápočtu, diskuze.
14. Zápočet.

Kmenová literatura

1. Brechlerová D., Jelen S.,Veselý A., Richta K. : Algoritmizace a programování v „C“,
2. Kukal J. : Myšlením k algoritmům, EDUCA ‘99
3. Borland International Inc. : TURBO C++ 3.00
4. Data Becker Gmbh : PC intern 3.0 Systemprogrammierung, 1992

Literatura

1. Brechlerová D., Jelen S.,Veselý A., Richta K. : Algoritmizace a programování v „C“,
2. Kukal J. : Myšlením k algoritmům, EDUCA ‘99
3. Borland International Inc. : TURBO C++ 3.00
4. Data Becker Gmbh : PC intern 3.0 Systemprogrammierung, 1992

Obory ve kterých je předmět vyučován

Systémové inženýrství.

Příloha 2: Jihočeská univerzita – sylabus zdroj: [6]

Algoritmy a datové struktury I - 2008	
Předchozí verze :	2007
Kredity :	3
Akreditace :	Akreditováno
Garant :	Beránek Ladislav, Ing. CSc.
Rozsah :	Přednáška 2 [HOD/TYD]
Způsob ukončení/forma zkoušky :	Zápočet/

Cíle předmětu

Úvodní kurz, který seznamuje studenty se základními algoritmy a datovými strukturami a technikami, které se využívají při analýze asymptotické časové a paměťové náročnosti. Kurz ideově vychází z pojmu ADT (abstract data type) a při implementaci se využívají návrhové vzory v programovacím jazyku Java.

Požadavky na studenta

Získání potřebného počtu bodů v průběžné kontrole a zpracová zápočtového projektu.

Literatura

Doporučená:
CORMEN, T. H. - LEISERSON, CH. E. - RIVEST, R. L.: Introduction to Algorithms. The MIT Press, 1994.
GOODRICH, M. T. - TAMASSIA, R.: Data Structures and Algorithms in Java. John Wiley & Sons, 2001.
PREISS, B. R.: Data Structures and Algorithms whit Object-Oriented Design Patterns in Java. John Wiley & Sons, 2000.

Vyučovací metody

Monologická (výklad, přednáška, instruktáž)
Hodnotící metody
Písemná zkouška
Studijní programy, do kterých je předmět zařazen

Studijní program	Typ	Forma	Obor	Etapa	Rok	Blok
B6209 - Systémové inženýrství a informatika	Bakalářský	Prezenční	6209R003-Ekonomická informatika	1	2008	Povinné předměty

Příloha 3: Mendelova zemědělská a lesnická universita sylabus zdroj: [7]
Sylabus předmětu ALG - Algoritmizace (FBE - WS 2007/2008)

Garant předmětu:	doc. Ing. Arnošt Motyčka, CSc.
Garantující pracoviště:	Ústav informatiky (DI FBE)
Dotace hodin (př./cv.):	3/2

Vyučován pro formu:	prezenční
Typ studia předmětu:	normální, konzultační
Forma výuky:	přednáška, cvičení
Ukončení a kredity:	zkouška (7 kreditů)
Cíl předmětu:	Osvojení si zásad řešení problému s využitím počítačů. Algoritmizace je prezentována jako způsob myšlení, je orientována do oblasti jak vědecko-technických výpočtů, tak i hromadného zpracováni dat. Vedlejším cílem předmětu je studenta EI naučit používat programovací jazyk Pascal jako prostředek pro zápis algoritmu.
Obsah předmětu:	1. Algoritmizace jako způsob myšlení (dotace 9/0) a. Strukturalizace postupu řešení; nástroje, struktury b. Modulárni přístupy; dekompozice, modularita c. Objektově orientované přístupy 2. Algoritmus; jeho reprezentace a obraty (dotace 14/14) a. Algoritmizace na bázi strukturovaného jazyka b. Algoritmizace základních obratů v oblasti zpracování dat c. Ověřování správnosti algoritmů d. Algoritmizace vědecko-technických výpočtů e. Algoritmizace hromadného zpracování dat

	3. Programovací jazyk Pascal- (dotace 21/14) a. Příkazové struktury b. Datové struktury; jednoduché, strukturované, dynamické, objekty c. Podprogramy; procedury a funkce d. Některé metody návrhu programu e. Implementace prostředí s jazykem Pascal; implementacne závislé a nezávislé prvky jazyka, specifika konkrétní implementace f. Tvorba programu; hromadné zpracování dat, vědecko-technické výpočty
Metody předmětu:	Klasické členění, 42 hodin přednášek, 28 hodin cvičení. Zpracování projektu z oblasti řešení úlohy hromadného zpracování dat a vědecko-technického výpočtu.
Ukončení předmětu:	Závěrečná písemná zkouška. Ke zkoušce dodáváno souhrnné hodnocení studenta cvičícím vždy na základě písemných testů.
Doporučená literatura:	Hruška, T., Pascal pro začátečníky, Praha: SNTL, 1989 Motyčka, A., Algoritmizace na bázi jazyka Pascal , Brno: CERN VUT, 1994 Novotná, H., Motyčka, A., Vybrané pasáže z Turbo Pascalu, Brno: CERN VUT, 1995 Rábová, Z. et al., Počítače a programování, Brno: ES VUT, 1980 Rybička, J., Programové vybavení počítačů (algoritmizace), Brno: ES VŠZ, 1992
Studijní plány:	B-SE-EI Economical informatics, prezenční forma, počáteční období ZS 2007/2008
Vyučující předmětu:	doc. Ing. Arnošt Motyčka, CSc. (garant, přednášející, zkoušející, cvičící) Ing. Roman Malo, Ph.D. (přednášející, cvičící) Mgr. Tomáš Foltýnek, Ph.D. (cvičící) Ing. Jan Turčínek (cvičící)

Příloha 5: Technická univerzita – sylabus zdroj: [8]

Algoritmizace a programování 1 - 2008	
Kredity :	5
Akreditace :	Akreditováno

Garant :	Královcová Jiřina, doc. Ing. Ph.D.
Rozsah :	Přednáška 2 [HOD/TYD], Cvičení 2 [HOD/TYD]
Zápočet před zkouškou :	Ano
Způsob ukončení/forma zkoušky :	Zkouška/Kombinovaná

Cíle předmětu

Předmět je úvodem do problematiky programování v programovacím jazyce vyšší úrovně. Studenti se seznámí se základními postupy při algoritmizaci úloh a s realizací algoritmů zaměřených na zpracování čísel a jejich posloupností pomocí výrazových prostředků programovacího jazyka Java.

Požadavky na studenta

Účast na cvičeních. Realizace zadané samostatné práce. Složení zkoušky

Přehled probírané látky

Témata přednášek:

1. Vývojové prostředí. Struktura programu.Základní lexikální elementy.
2. Proměnné, konstanty, typy.
3. Konzolový vstup a výstup.
4. Číselné typy - konstantní hodnoty, operace, standardní funkce. Priorita operátorů, konstrukce výrazů.
5. Generování náhodných čísel.
6. Logický typ. Znakový typ.
7. Logické příkazy. Logické výrazy a jejich konstrukce.
8. Příkazy cyklu.
9. Algoritmizace úloh.
10. Objektové programování. Třídy a metody. Parametry metod.
11. Typ pole, základní algoritmy.
12. Třídění polí.
13. Vyhledávací algoritmy.
14. Vícerozměrná pole. Rekurze, její využití. Rekurzivní metody.

Náplň cvičení:

1. Vývojové prostředí. Základní prostředky.
2. Načítání z konzole, výpis na konzoli.
3. Výpočty reálných výrazů.
4. Výpočty celočíselných výrazů.
5. Logické příkazy. Logické výrazy. Rozhodovací algoritmy. 6.Rozhodovací strom.
7. Cykly. Základní algoritmy s opakováním.
8. Výpočet faktoriálu, mocniny, největší společný dělitel, výpočet hodnoty funkce jako součtu řady.
9. Zpracování posloupnosti číselných hodnot bez jejich uložení do struktury pole. Dva způsoby načtení.
10. Komplexnější algoritmy využívající rozhodování a cykly.
11. Metody třídy, členění kódu, parametry metod.
12. Použití struktury pole. Načtení pole. Zpracování pole čísel.
13. Základní algoritmy. Návrh metod pro zpracování pole.
14. Algoritmy třídění. Realizace, porovnání. Algoritmy vyhledávání. Realizace, porovnání.

Použití vícerozměrného pole a jeho zpracování. Rekurzivní metody.
Literatura
Základní: HEROUT, P. Učebnice jazyka Java. Kopp, České Budějovice, 2003. VIRIUS, M. Java pro zelenáče. Neokortex, Praha, 2001 Doporučená: CORMEN, T. H. Introduction to algorithms. The MIT Press, Cambridge, Massachusetts, 2001. HORTON, I. Java 5. Neokortex, Praha, 2006. SEDGEWICK, R. Algoritmy v C. SoftPress, 2003. WRÓBLEWSKI, P. Algoritmy datové struktury a programovací techniky. Computer Press, Brno, 2004.
Podmiňující předměty
Získané způsobilosti
Student získá znalosti v oblasti základních postupů algoritmizace úloh a realizace algoritmů zaměřených na zpracování čísel a jejich posloupností v programovacím jazyce Java.
Vyučovací metody
Pracovní činnosti (dílny) Přednášení
Hodnotící metody
Písemná zkouška Ústní zkouška Známkou
Studijní programy, do kterých je předmět zařazen

Studijní program	Typ	Forma	Obor	Etapa	Rok	Segment	Blok
B3918 - Aplikované vědy a informatika	Bakalářský	Prezenční	3902R047-Modelování a informatika	1	2008	FM-B3918-MI	FM-bak MI - povinné předměty

Příloha 6: Univerzita Hradec Králové – sylabus zdroj: [9]
ALGORIT - Algoritmy a algoritmizace

Kreditové hodnocení předmětu	3
Garant předmětu	Rabe Vlasta
Garantující katedra	FY - katedra fyziky a informatiky
Způsob ukončení předmětu	z - zápočet
Forma zkoušky	pu - písemná a ústní
Rozsah výuky předmětu	P - přednáška 9 hod. za semestr

Anotace předmětu	Algoritmus, algoritmizace a vývojové diagramy. Obecně o programování. Základy matematické logiky.
Cíle předmětu a charakteristika získaných dovedností	Uvědomit si potřebu jasné formulace úlohy a způsobu řešení. Vyjádření algoritmu ve formě vývojového diagramu. Vztah program a data. Některé obecné pojmy vztahující se k programování. postup při tvorbě programu.
Osnova předmětu ve vztahu k časovému rozvrhu výuky	První část: 1. Úvod - algoritmus a vztah k programování 2. Algoritmizace a formy vyjádření algoritmů 3. VD - značky, značky, zápis základních struktur 4. VD - zápis algoritmu 5. VD - řešení úloh I. (rozhodování) 6. VD - řešení úloh II. (cykl s pevným počtem opak.) 7. VD - řešení úloh III. (iterace) 8. VD - řešení úloh IV. (součet řady) 9. VD - řešení úloh V. (podprogram, třídění) 10. Programování - příkaz, výraz, proměnná 11. Programování - datové struktury 12. Programování - podprogramy a funkce 13. Programování - zdrojový text a překladač 14. Programování - postup při tvorbě programu Druhá část: Základy matematické logiky - výrokový počet, operace s výroky, množiny, vztahy, operace, kartézský součin, zobrazení. Relační algebra.
Literatura, na níž je předmět vystavěn	Taufer,I.-Hrubina,J.-Taufer,J.: Algoritmy a algoritmizace - vývojové diagramy. MAFY Hradec Králové, 2001, ISBN 80-86148-49-1
Literatura doporučená studentům	ČSN ISO 5807 Zpracování informací – Dokumentační symboly a konvence pro vývojové diagramy toku dat, programů a systémů, síťové diagramy programů a diagramy zdrojů systémů. Praha: Český normalizační institut, 1995 ČSN 36 9030 Značky vývojových diagramů pro systémy zpracování dat. Praha: Úřad pro normalizaci a měření, 1977
Způsob a pravidla výsledné klasifikace předmětu	Zápočet: 1) učast min.50% 2) vypracování seminární práce (řešení úlohy ve formě VD)

Příloha 7: Univerzita Pardubice – sylabus zdroj: [10]

Algoritmizace a programování - 2008	
Kredity :	5

Akreditace :	Akreditováno
Garant :	Fabián Petr, doc. Ing. CSc. , Panuš Jan, Ing.
Rozsah :	Přednáška 2 [HOD/TYD], Cvičení 2 [HOD/TYD]
Zápočet před zkouškou :	Ano
Způsob ukončení/forma zkoušky :	Zkouška/

Cíle předmětu

Cílem předmětu je seznámit studenty se základy informatiky a technického vybavení počítačů, základy zápisu pomocí vývojových diagramů a základy programovacího jazyka Pascal.

Požadavky na studenta

Zápočet:
Zpracovat úlohy v cvičení s úspěšností min. 60%.
Semestrální práce - vypracovat a obhájit samostatně vytvořený a odladěný program na vybrané téma.
Náhradní podmínky:
Test s hodnocením minimálně 70%.

Přehled probírané látky

Úvod, vymezení pojmu počítač, počítačový systém.
Technické a programové vybavení počítačů.
Postup řešení úloh na počítači.
Algoritmizace úloh.
Základní pojmy v algoritmizaci a programování.
Programovací jazyky a jazyk Pascal.
Konstanty, proměnné, výrazy a příkazy.
Bloková struktura programu.
Údajové typy.
Rozdělení údajových typů.
Jednoduché příkazy.
Standardní procedury vstupu a výstupu údajů.
Strukturované příkazy.
Strukturované údajové typy.
Principy objektově orientovaného programování.

Literatura

Základní:
JEŽOWICZ, E., LAGA, J. Základy programování v jazyku Pascal.
JINOCH,J., MUELLER, K., VOGEL,J. Programování v jazyku Pascal. SNTL Praha, 1985.
KVOCH,M., JANČÍK ,J. Sbírka úloh z jazyka PASCAL. České BUdějovice, 1995.
PECINOVSKÝ, R. Cesta k profesionalitě - Základy algoritmizace.

Rozšiřující:
WIRTH, N. Algoritmy a štruktúry údajov. ALFA Praha, 1987.

Předpoklady - další informace k podmíněnosti studia předmětu

Získané způsobilosti
Schopnost analyzovat problém, vytvořit a zapsat algoritmus formou vývojového diagramu a programu v programovacím jazyce PASCAL. Porozumnění pojmů výpočetní techniky a programování.
Vyučovací metody
Metody práce s textem (učebnicí, knihou) Metody samostatných akcí Monologická (výklad, přednáška, instruktáž)
Hodnotící metody
Písemná zkouška Posouzení zadané práce Ústní zkouška
Studijní programy, do kterých je předmět zařazen

Studijní program	Typ	Forma	Obor	Etapa	Rok	Segment	Blok
B6209 - Systémové inženýrství a informatika	Bakalářský	Prezenční	6209R019-Informatika ve veřejné správě	1	2008	SII - IVS - Bc - P	Povinné předměty