Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Didaktika programovania pre SŠ 1

Predmet: Didaktika programovania pre SŠ

Línia: Didaktika informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

Moderné vzdelávanie pre vedomostnú spoločnosť/Projekt je spolufinancovaný zo zdrojov ES

Didaktika programovania pre SŠ 1

Garant predmetu:

RNDr. Ľubomír Salanci, PhD.
KZVI FMFI UK, Bratislava
salanci@fmph.uniba.sk

Autori:

RNDr. Ľubomír Salanci, PhD.
FMFI UK, Bratislava
PaedDr. Monika Tomcsányiová, PhD.
FMFI UK, Bratislava
RNDr. Andrej Blaho, PhD.
FMFI UK, Bratislava

Identifikácia modulu

Aktivita projektu:	1.2 Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ
Línia aktivity:	Didaktika informatiky a informatickej výchovy
Predmet:	Didaktika programovania pre SŠ

Zaradenie modulu

Na základný modul Didaktika programovania nadväzujú moduly Didaktika programovania pre stredné školy 1 a 2. K týmto modulom existujú aj paralelné (alternatívne) moduly Didaktika programovania pre základné školy 1 a 2.

Modul má e-learningovú podporu

Abstrakt modulu

Moduly Didaktika programovania pre stredné školy 1 a Didaktika programovania pre stredné školy 2 sú zamerané na vyučovania informatiky a programovania na strednej škole.

V tomto module preskúmame a analyzujeme súčasné učebnice pre vyučovanie programovania. Budeme analyzovať a diskutovať o vyučovaní prvých tém z programovania, akými sú: úvod do programovania, grafika, premenné, cyklus. Navrhneme vzorové vyučovacie hodiny a budeme ich analyzovať z pohľadu teórie poznávacieho procesu.

Obsah

Úvod

Didaktikou programovania rozumieme vednú oblasť, v ktorej skúmame a hľadáme spôsoby, ako učiť programovanie. Snažíme sa porozumieť poznávaciemu procesu, tomu, ako vzniká nový poznatok, ale aj ťažkostiam, ktoré majú žiaci s porozumením algoritmov, s programovaním alebo riešením informatických problémov. Pritom chceme objavovať zákonitosti alebo pravidlá, ktoré by prispeli k lepšiemu vyučovaniu informatiky.

Pre tento modul je nevyhnutné mať k dispozícii nasledujúce softvérové prostredia: Cirkus šaša Tomáša, Imagine Logo, Karel, Scratch, Panák, IzyLogo, FreePascal, VBA, DevC. Tieto prostredia môžu byť k dispozícii v rámci intranetu.

Účastníci budú používať vlastné notebooky.

Výučba bude prebiehať v miestnosti s projektorom a s dostatočne veľkou tabuľou.

Cieľ modulu

Cieľom modulu je, aby účastník vzdelávania:

- Diskutoval o vyučovaní programovania na SŠ.
- Porozumel postaveniu, úlohe a cieľom programovania v informatike na a SŠ.
- Spoznal a analyzoval učebnice informatiky so zameraním na programovane.
- Vedel aplikovať teóriu poznávacieho procesu vo vyučovaní programovania.
- Spoznal metodiku vyučovania prvých tém z programovania na SŠ.
- Naučil sa navrhovať a zostavovať vyučovacie hodiny z programovania.

Vstupné vedomosti

Požadované prerekvizity

Účastník vzdelávania musí mať absolvované všetky nasledujúce moduly:

- Programovanie 1 až 9,
- Didaktika programovania.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

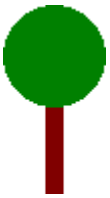
Pozná aspoň tieto programovacie jazyky: Pascal (Delphi / Lazarus), Logo (Imagine). Dokáže riešiť základné algoritmické problémy. Riešenie vie zapísať a odladiť v niektorom z programovacích prostredí. Vie čítať cudzie programy, dokáže im porozumieť, analyzovať ich.

Pozná históriu vyučovania informatiky a programovania a jeho súčasnú úlohu v rámci informatiky na základnej a strednej škole. Je oboznámený s teóriou poznávacieho procesu vo vyučovaní programovania.

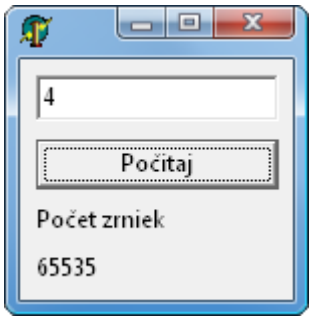
Kapitola 1: Rozcvička

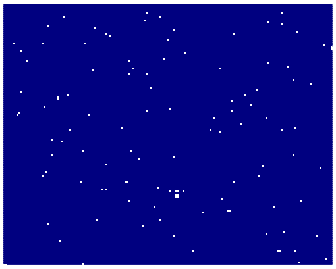
Aby sme sa správne naladili na programovanie, budeme teraz riešiť niekoľko triviálnych programátorských úloh. Okrem toho, že si pripomenieme programovanie v jazyku Pascal, dôležitejšie bude, že sa na jednotlivé úlohy budeme pozerať aj z pohľadu vyučovania informatiky.

Táto kapitola obsahuje sériu aktivít, pričom jednotlivé aktivity na seba nadväzujú. Prvá aktivita je zameraná na samotné riešenie malého programátorského problému. Ďalšie aktivity už súvisia so samotným vyučovaním – presnejšie s tým, ako by sme na riešenie úlohy naviedli žiakov, ktorí by mali s jej riešením ťažkosti. Preto sú niektoré aktivity zamerané aj na analýzu programátorských úloh.



Aby sme sa s úlohami zoznámili a spoznali problémy, ktoré na riešiteľa čakajú, najskôr úlohy vyriešime.





Aktivita 1.1

Rozdeľte sa do skupín – do 5-6 členných tímov. Každá skupina vyrieši jednu programátorskú úlohu v jazyku Pascal:

1. Úloha: Vytvorte program, ktorý nakreslí hviezdnu oblohu tak, že na modré pozadie sa nakreslí 1000 náhodne rozmiestnených bielych bodiek.

2. Úloha: Poznáte príbeh o prešibanom mudrcovi a šachovi, ktorý sľúbil za odmenu toľko zrniek, koľko sa zmestí na políčka šachovnice? Presnejšie tak, že na prvé políčko dáme jedno zrnko, a na každé ďalšie políčko dáme dvojnásobný počet zrniek? Pomôžte šachovi v rozhodovaní, či je taká odmena adekvátna a vytvorte preňho jednoduchý program. Do programu zadáme rozmery šachovnice a ten nám spočíta a povie, koľko zrniek by malo byť dohromady na šachovnici. Napríklad, pre vstup 2 (šachovnica 2×2) zobrazí výsledok 15 (=1+2+4+8).

3. Úloha: V lesnej škôlke rastú stromčeky. Zatiaľ sú malé a majú náhodnú výšku aj šírku – od 30 do 100 cm. Nakreslite 10 z nich na obrazovku vedľa seba tak, aby sa neprekrývali. Koruna stromu je zelený kruh a kmeň hnedý obdĺžnik.

4. Úloha: Vytvorte program, ktorý bude generovať umelecké farebné mozaiky. Kamienok mozaiky je štvorec so stranou 10 a mozaika má rozmery 20 štvorcov na šírku a 10 na výšku. Každý kamienok je náhodne zafarbený.

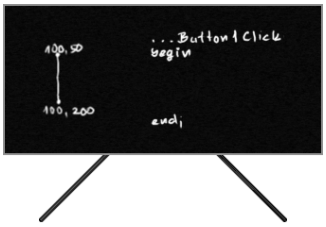
5. Úloha: V krajine Hromovo je stále zamračené a prší. Aby sme potešili jej obyvateľov, rozhodli sme sa vytvoriť program, ktorý nakreslí slniečko. Slniečko je žltý kruh a z neho vychádza napríklad 25 slnečných lúčov. Pozn.: Tvárme sa, že korytnačiu grafiku nemáme zatiaľ v jazyku Pascal k dispozícii.

V skupine môžete spolupracovať. Úlohy môžete riešiť na papier alebo pomocou počítača.

Rátajte s tým, že v ďalšej aktivite budete svoje riešenia prezentovať ostatným kolegom.

Ako správni a dobrí učitelia by sme mali byť schopní vysvetliť ideu riešenia aj žiakom (napríklad pomôcť tým, ktorým „to nejde“) tak aj svojim kolegom – učiteľom.

Aktivita 1.2 Vystúpenie	Vysvetlite ideu riešenia svojom kolegom. Každá skupina si zvolí zástupcu, ktorý vystúpi pred ostatnými kolegami. Pozor: • **Nepoužívajte počítač ani projektor**. • Ak potrebujete, kreslite na tabuľu. • Nezapisujte celý program. • Zamerajte sa na jadro problému. • Ak potrebujete písať príkazy, vysvetlite, ako ste prišli na jednotlivé zápisy príkazov. • Ideálne vystúpenie trvá najviac 5 minút.



Jednotlivé vystúpenia budeme pozorovať a budeme si k nim zapisovať poznámky. Po každom vystúpení sa pokúsime analyzovať ho. Navyše, k vystúpeniam sa budeme vracať v ďalších aktivitách. Budeme sledovať viacero rovín: od programátorskej a informatickej, cez didaktickú až po koncepčnú.

Aktivita 1.3	Zhodnoťte vysvetlenie vášho kolegu alebo kolegyne. • Bolo riešenie správne? • Čo by ste navrhli zlepšiť, vymeniť? • Čo sa vám páčilo a prečo?

Určite sme si všimli, že úlohu so slniečkom by sme v jazyku Imagine-Logo naprogramovali podstatne jednoduchšie, ako v jazyku Pascal.

Slniečko v Imagine-Logo:

```
nechfp "oranzova
nechhp 2
opakuj 25 [
  do 100
  vz 100
  vl 360/25]
bod 100
```

Aktivita 1.4	Čím to je, že kreslenie slniečka je v jazyku Pascal náročné?

Pokým sa nám zdá niektorá úloha príliš náročná, mali by sme sa pokúsiť úlohu zjednodušiť. Zjednodušovať úlohy a príklady je veľmi dôležitá schopnosť učiteľa na to, aby sa dokázal prispôsobiť poznatkom svojich žiakov.

Aktivita 1.5	Dokázali by ste zjednodušiť zadanie úlohy o slniečku? Ako? Povedzte svoje varianty.

Vidíme, že jednotlivé úlohy, ktoré sme riešili v prvej aktivite, sú rôzne náročné. Podobne, ako v úlohe o slniečku, je aj v ostatných úlohách náročný iba určitý koncept, poznatok, objav, kvôli ktorému môžu mať žiaci vážny problém s riešením. Ako učitelia by sme mali byť schopní rozpoznať v úlohách potenciálny zdroj komplikácií.

Aktivita 1.6	Identifikujte aj v ostatných úlohách to, čo je na ich riešení najťažšie – teda, čo môže robiť žiakom najväčší problém.

Úloha so slniečkom je ťažká, môžeme vidieť, že na jej riešenie stačí poznať „iba“ cyklus, príkazy pre kreslenie čiar a goniometrické funkcie sin a cos.

Aktivita 1.7	Identifikujte príkazy, programové konštrukcie a poznatky, ktoré musia žiaci mať na to, aby dokázali všetky úlohy. Vymenujte ich.

Keby sme chceli zadať úlohy z prvej aktivity aj našim žiakom, je možné, že niektoré z nich by sme mohli použiť už veľmi skoro, iné oveľa neskôr.

Aktivita 1.8	Zoraďte úlohy podľa náročnosti. Povedzte, pri akej príležitosti by ste mohli jednotlivé úlohy riešiť so žiakmi – aké vedomosti už musia mať žiaci na to, aby úlohu samostatne vyriešili?

Zhrnutie

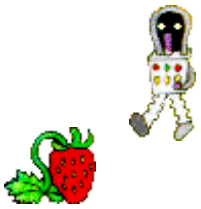
Úlohy z programovania by mali byť také, aby ich matematická náročnosť neprekryla informatické koncepty, ktoré chceme naučiť, trénovať a vyskúšať.

Kapitola 2: Súčasné učebnice programovania

V tejto kapitole sa bližšie pozrieme na slovenské a prípadne aj niektoré zahraničné učebnice, ktoré sa zaoberajú programovaním. Pravdepodobne nás neprekvapí, že vzhľadom na tradíciu a dobré skúsenosti s vyučovaním v programovacích jazykov Logo a Pascal, sú tieto učebnice venované práve uvedeným spomínaným jazykom.

V priebehu posledných 10 rokov vzniklo na Slovensku niekoľko oficiálnych učebníc o programovaní pre základné aj stredné školy:

- Tvorivá informatika – 1. zošit z programovania [4] ... pre základnú školu
- Algoritmy s Logom [5] ... pre strednú školu
- Algoritmy s Pascalom [6] ... pre strednú školu
- Programovanie v Delphi [7] ... pre strednú školu



Tematické zošity Algoritmy s Logom a Algoritmy s Pascalom sú už pomerne staré (roky vydania 1999, resp. 2002). V porovnaní s nimi sú učebnica Programovanie v Delphi a tematický zošit Tvorivá informatika – 1. zošit z programovania relatívne najnovšie (vyšli v roku 2006, resp. 2005). Napriek tomu sa oplatí pozrieť sa aj do starších učebníc, pretože v nich môžeme nájsť veľa zaujímavých príkladov alebo nás môžu potešiť svojim metodickým spracovaním.

Okrem týchto spomínaných učebníc, ktoré sú vyslovene zamerané na programovanie, existujú aj iné učebnice, v ktorých sa algoritmy alebo programovanie nejakým spôsobom vyskytujú. Spomeňme napríklad:

- učebnicu Informatickej výchovy pre 2. ročník ZŠ,
- učebnicu Informatika pre stredné školy.

Existuje veľa zahraničných učební programovania, ktoré sú blízke k našej filozofii vyučovania. Tie pochádzajú z krajín, ako napríklad z Poľska, Litvy alebo Anglicka.

Základná stredoškolská učebnica Informatika pre stredné školy obsahuje kapitolu o algoritmoch a programovaní s názvom Algoritmy a algoritmizácia. V učebnici Informatickej výchovy pre prvý stupeň základných škôl objavíme príklady, ktoré majú veľa spoločného so zostavovaním alebo čítaním návodov.

V nasledujúcich aktivitách postupne preskúmame dostupné učebnice, ktoré sa nejakým spôsobom venujú programovaniu.

Aktivita 2.1

Každý tím dostane učebnicu. Preštudujte ju a pokúste sa zodpovedať na nasledujúce otázky:

- Aký programovací jazyk sa používa?
- Aké prostredie používajú autori pri vysvetľovaní?
- Aké témy a v akom poradí sa preberajú? Stručne ich vymenujte.
- Aký je váš dojem?

Po preštudovaní zvoľte jedného zástupcu, ktorý v krátkosti prezentuje výsledky svojej skupiny pred ostatnými kolegami.

Pozn.: Na túto aktivitu nadväzuje aktivita 2.2 na konci tejto kapitoly.

Tvorivá informatika – 1. zošit z programovania

Tento tematický zošit je určený pre prímu až kvartu osemročných gymnázií a 2. stupeň základných škôl. Napriek tomu si myslíme, že môže byť zaujímavý aj pre učiteľa strednej školy, pretože z neho získame aspoň približnú predstavu o tom, čo by mohli zvládnuť niektorí šikovní žiaci základnej školy (samozrejme, nemôžeme predpokladať, že tak budú zvládať programovať všetci žiaci – práve naopak, skôr môžeme očakávať, že väčšina žiakov na základnej škole sa s programovaním nestretla).

Príklady v učebnici a vysvetľovanie programátorských konceptov je postavené na prostredí Imagine-Logo.

V jednotlivých kapitolách sa preberajú nasledujúce témy:

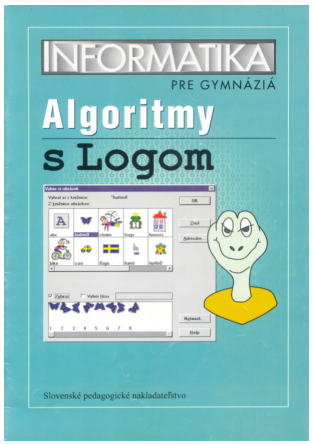
- oboznámenie sa s prostredím Imagine a základnými príkazmi korytnačky,
- rôzne nastavenia pera (farba, hrúbka, náhodná farba) a ďalších vlastností korytnačky (tvar korytnačky),
- jednoduché udalosti (kliknutie, ťahanie),
- práca s viacerými korytnačkami,
- príkaz cyklu, premenné, podmienený príkaz, vytváranie vlastných príkazov,
- animované príbehy a jednoduché procesy.

Vidíme, že žiaci sa už takto skoro stretávajú s paradigmami objektovo-orientovaného programovania a udalosťami riadeného programovania tým, že používajú viaceré korytnačky alebo reagujú na kliknutia myšou. Na takejto veľmi jednoduchej úrovni ešte nemusia poznať uvedené paradigmy, ale už s nimi dokážu pracovať.

Koncepcia zošita počíta s konštrukcionistickým prístupom k vyučovaniu programovania. Žiaci budú musieť programovať pri počítači a skúšať príklady, úlohy. Dôležité a pre žiakov motivujúce je to, že úlohy sú graficky orientované.

Tematický zošit sa dá do určitej miery chápať ako manifest toho, čo je možné realizovať v prostredí Imagine-Logo. Obsahuje veľa námetov na precvičenie a projekty. Zošiť sa nezaoberá vysvetľovaním programátorských konceptov. Predpokladáme, že mnohé úvodné a triviálne úlohy na zbieranie prvých skúseností si bude musieť učiteľ pripraviť sám.

Algoritmy s Logom



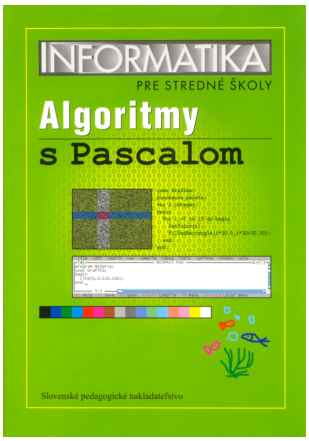
Tematický zošit Algoritmy s Logom je určený pre gymnáziá a stredné školy. Tento zošit je už relatívne starý a je postavený na vyučovaní programovania v staručkom prostredí Comenius Logo.

V zošite sú spracované témy:

- úvod do programovania v jazyku Logo,
- cyklus, procedúra, parametre, premenná, vetvenie programu,
- jednoduchá rekurzia,
- spracovanie vstupu z klávesnice a myši,
- algoritmy s viacerými korytnačkami a animovanými korytnačkami.

Napriek tomu, že je tento zošit pomerne starý, je z didaktického hľadiska spracovaný veľmi korektne.

Algoritmy s Pascalom



Aj tento tematický zošit je určený pre gymnáziá a stredné školy. Učebnica je postavená na vyučovaní programovania v prostredí TurboPascal.

Obsahuje nasledujúce témy:

- oboznámenie sa s prostredím a grafickými príkazmi,
- kreslenie geometrických útvarov, používanie farieb, náhodnosť,
- premenné, cyklus, procedúry, funkcie, parametre, vetvenie, reťazce a polia,
- spracovanie vstupov z klávesnice a myši, práca so zvukom

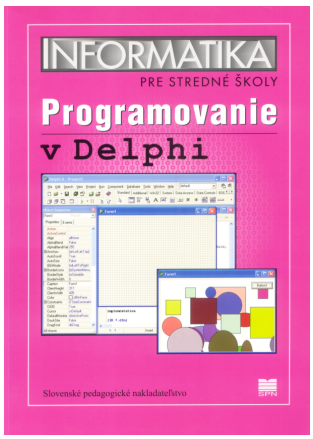
Aj keď sa v učebnici pracuje v už v historickom (a v súčasnosti už prekonanom) prostredí TurboPascal, vidíme snahu autorov o to, aby sa od začiatku využívala grafika. Znamená to, že už základné programové konštrukcie sa vysvetľujú na príkladoch, ktoré niečo kreslia. Rovnako aj študenti riešia úlohy, v ktorých má program čosi nakresliť.

Programovanie v Delphi

V porovnaní s ostatnými zošitmi má Programovanie v Delphi formu rozsiahlejšej učebnice. Obsahuje veľa vysvetľujúcich textov, riešených príkladov a úloh na samostatnú prácu.

Stručný prehľad tém z učebnice:

- úvod do prostredia, programovania a syntaxe,
- jednoduché algoritmy – postupnosti príkazov, premenné, cykly, vetvenie,
- algoritmy s reťazcami, poľami, textovými súbormi,
- algoritmy na prácu s myšou, algoritmy na jednoduchú animáciu,
- funkcie.



Na učebnici je cenný jej prístup k vyučovaniu programovania. Vidíme, že úvod do programovania v prostredí Delphi a v jazyku Pascal je postavený na grafických príkladoch. Väčšina iných kníh, aj zahraničných učebníc o programovaní, sa totiž pridržiava tradičnejšej postupnosti tém, kedy sa na začiatku pracuje v textovom (konzolovom) režime a riešia sa textové lebo matematické typy úloh.

Učebnica má spočiatku mierne tempo. Okrem grafických úloh v nej postupne nájdeme aj také, v ktorých sa niečo počíta alebo sa pracuje s textom. Pri vysvetľovaní nových pojmov (premenná, pole, súbor) alebo algoritmov chýbajú vysvetľujúce obrázky, ilustrácie alebo trasovanie. Preto si bude musieť učiteľ sám pomôcť s ich znázorňovaním. Tiež si budeme musieť dať pozor na posledné kapitoly (práca s textovými súbormi alebo výpočty – funkcie), ktoré majú v porovnaní so zvyškom učebnice príliš rýchle tempo. Tieto časti je potrebné preberať opatrne a nechať si na ne dostatok času.

Opakovanie

Pripomeňme si etapy poznávacieho procesu:

Motivácia	1. etapa
↓	
Zbieranie skúseností	2. etapa
↓	
Zovšeobecňovanie, abstrakcia	3. etapa
↓	
Poznatok	4. etapa
↓	
Trénovanie poznatku	5. etapa

Aktivita 2.2

Opäť pracujte v pôvodnom tíme a používajte svoju učebnicu. Teraz sa však zamerajte iba na jednu kapitolu v učebnici a analyzujte ju z pohľadu poznávacieho procesu.

Povedzte:

- Aká motiváciu autor použil?
- Na akých príkladoch alebo situáciách zbierajú žiaci prvé skúsenosti s novou témou?
- K akému poznatku autor žiakov nasmeruje?
- Aké úlohy na precvičenia sa v učebnici riešia?

Je možné, že rôzni autori rôznych učebníc niektoré etapy poznávacieho procesu vynechali alebo poprehadzovali. Napríklad:

- V niektorých učebniciach chýbajú motivácie (nie sú explicitne uvedené).
- Najskôr ponúkajú abstraktné a všeobecné informácie, až potom konkrétne. Typickým príkladom je cyklus, pri ktorom najskôr prezradia čitateľovi schému:
  ```
  for premenná:=od to po do príkaz
  ```
  My však vieme, že takáto schéma je až na úrovni 3. etapy poznávacieho procesu. Bez predchádzajúcich jednoduchých príkladov s cyklom a konkrétnymi hodnotami má uvedená schéma pre začiatočníka malý zmysel.

Ako učitelia potrebujeme takéto nedostatky prekovať a vymyslieť vlastné motivácie, prípadne jednoduché úvodné príklady.

Aktivita 2.3

Pokým sa v kapitole nedodržiavajú jednotlivé etapy poznávacieho procesu tak, ako sme sa ich učili, navrhnite a prezentuje vlastné správne riešenie.

Napríklad:

- Ak chýba motivácie, navrhnite vlastnú motiváciu.
- Ak chýbajú prvé elementárne príklady na zbieranie skúseností, navrhnite vlastný príklad.
- Ak sú prvé príklady príliš komplikované, povedzte, ako by ste ich zjednodušili.
- Ak sú niektoré etapy poprehadzované, povedzte, ako majú byť usporiadané v správnom poradí.

Zhrnutie

Analyzovali sme rôzne učebnice programovania. Zopakovali sme etapy poznávacieho procesu a učili sme sa tieto etapy hľadať a rozpoznávať v učebniciach.

Kapitola 3: Poradie tém pre jazyk Pascal

Pravdepodobne sme už počuli rôzne názory na to, aké témy sa učia v rámci informatiky a tematického celku **Postupy, riešenie problémov, algoritmické myslenie**, v čom žiaci programujú, akým spôsobom sa jednotlivé témy preberajú, do akej hĺbky alebo, aké úlohy sú schopní žiaci riešiť.

Strom možností

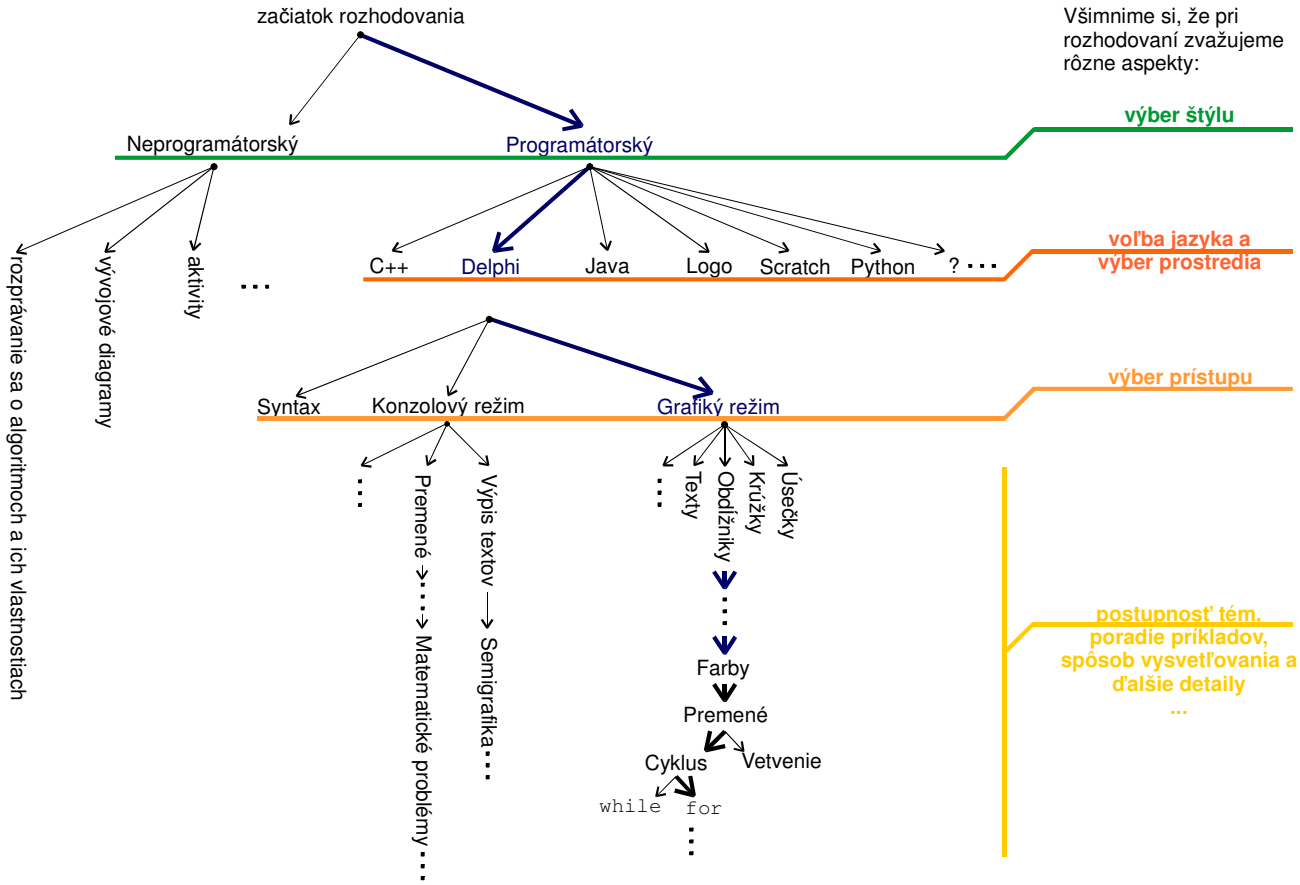
Existuje veľa spôsobov, ako môžeme programovanie vyučovať. Pokúsme sa teraz spomenúť si na rôzne varianty, ako v rámci informatiky učiť spomínaný tematický celok.

Aktivita 3.1

Povedzte, aké témy z programovania a v akom poradí učíte alebo by ste učili svojich žiakov.

Jednotlivé alternatívy zaznamenajte na tabuľu, napríklad v podobe stromu.

Na vyučovanie (nie len) algoritmov a programovania sa môžeme pozerať ako na proces, pri ktorom sa neustále **rozhodujeme** a zvažujeme rôzne varianty: prístupy, metódy, témy, poradie tém, motivácie, ukážkové príklady, úlohy na samostatné riešenie atď. Proces rozhodovania sa dá znázorniť ako strom rôznych možností:



Každý vrchol v strome predstavuje určitý bod, kedy sa rozhodujeme medzi rôznymi alternatívami. Počet alternatív, ktoré sme boli schopní zaznamenať do nášho stromu je obmedzený. Aj mierka je veľmi hrubá. V skutočnosti zvažuje mnohonásobne viac možností a detailov.

Všimnime si, že:

- V strome vidíme veľa alternatívnych ciest, ako môžeme postupovať pri vyučovaní programovania.
- V strome vidíme aj niektoré známe, ale už nie odporúčané spôsoby vyučovania (napríklad, vývojovými diagramy).
- Vieme si predstaviť, ako by vyzeral strom, keby všetci učitelia informatiky na celom svete pridali do stromu svoj vlastný variant vyučovania programovania?
- Okrem toho, môžeme do stromu zaradiť rôzne, napríklad aj náhodne zostavené, postupnosti tém. Vedeli by sme podľa takej postupnosti učiť?
- Nad každou postupnosťou, aj nad takou, ktorá vyzerá pre nás na prvý pohľad nezmyselne, sa môžeme zamýšľať, či by sme ju dokázali takým spôsobom učiť a za akú cenu.
- Čím viac alternatív do stromu pridávame a čím viac sa zamýšľame nad rôznymi detailmi, tým viac sa stáva strom rozsiahlym. Je to preto, lebo aj samotné vyučovanie programovania je ohromne komplexné.

Niektoré spôsoby vyučovania programovania, ktoré máme zachytené aj v našom strome, sú pre nás zaujímavé a poučné. Preto sa pozrieme na nasledujúce tri prístupy vyučovania programovania:

- **Neprogramátorská informatika** – rozprávanie sa o algoritmoch, aktivity,
- **Tradičný prístup** – začíname programovať v konzolovom režime,
- **Súčasný prístup** – začíname programovať s využitím graficky.

Aktivita 3.2	Zamyslite sa a povedzte, aké výhody a nevýhody majú spomínané tri prístupy k vyučovaniu programovania?

Informatika bez programovania

Tento prístup spočíva v tom, že o algoritmoch a programovaní sa so žiakmi najskôr rozprávame. Na počítači zatiaľ nič neprogramujeme.

Vývojové diagramy boli výhodné v časoch diernych štítkov, keď bol strojový čas drahý a prístup k počítačom ťažký.

V súčasnosti sú vývojové diagramy alebo rôzne algoritmické jazyky v prípade vyučovania začiatočníkov viac škodlivé, ako užitočné.

Charakteristické pre takýto prístup sú tieto prvky:

- Často sa začína tým, že učiteľ porozpráva alebo vysvetlí etapy riešenia problémov a tvorby softvéru. Rozpráva o vlastnostiach algoritmov alebo zložitosti algoritmov.
- Žiaci slovne popisujú známe činnosti, akými sú návody na varenie obľúbeného jedla alebo obsluhu výťahu.
- Žiaci sa hrajú na interpreter – procesor. Napríklad, navigujú vybraného spolužiaka tak, aby otvoril okno.
- Učiteľ sa so žiakmi zamýšľa nad tým, aké kritéria musí spĺňať jazyk, aby bol návod pre spolužiaka jednoznačný.
- Neskôr, niektoré algoritmy zapisujeme pomocou vývojových diagramov, učíme žiakov vývojové diagramy čítať, porozumieť im a hľadať v algoritmoch chyby.

Takýto prístup už dnes považujeme za škodlivý. Existuje viacero dôvodov, prečo je to tak:

- Keďže žiaci ešte nemajú predstavu o tom, čo je programovanie a ako programy fungujú, je teoretické rozprávanie o etapách a vlastnostiach algoritmov zbytočnou teoretickou záťažou, ktorú pravdepodobne nikto zo žiakov na začiatku nedocení.

Skúmať vlastnosti algoritmov je samozrejme pre informatiku veľmi dôležité, ale vôbec to neznamená, že týmto spôsobom musíme začínať vyučovanie informatiky a programovania v škole.

- Tým, že žiaci najskôr píšu programy na papier v nejakom algoritmickom jazyku alebo vo forme vývojových diagramov, komplikujeme žiakom porozumie informatických konceptov.

Algoritmický jazyk aj vývojové diagramy sú zbytočnou medzivrstvou a komplikáciou, ktorá leží medzi objavením riešenia problému a jeho praktickým overením. Okrem toho, že spustením programu na počítači žiak testuje riešenie, má radosť z toho, že sám niečo dokázal vytvoriť, naprogramovať.

Tradičný prístup

Pre tento prístup je charakteristické to, že:

- Žiaci programujú v klasickom prostredí TurboPascal, FreePascal, FPC.

- Žiaci od začiatku vytvárajú konzolové aplikácie, teda aplikácie, ktoré pracujú **v textovom režime** (nie grafické programy, ako v prostredí Logo).

Konzolové aplikácie boli typické pre 90. roky a prostredie DOS. Konzolové aplikácie pracujú v textovom režime: vstup od používateľa si pýtajú v dialógovom riadku, výsledky vypisujú ako texty alebo čísla na obrazovke. Pre väčšinu súčasných mladých ľudí je však takáto práca s počítačom cudzia.

- Preberá sa postupnosť tém: výpis na obrazovku, premenná, načítanie vstupu od používateľa, programové konštrukcie, ako napríklad podmienený príkaz, cyklus `for`, `while`, `repeat ... until`, polia, ...

- Niekedy sa v textovom režime zvyknú kresliť obrázky poskladané zo znakov ASCII. Grafika a práca v grafickom režime býva vyčlenená ako samostatná (a relatívne izolovaná) téma, často až nakoniec, akoby za odmenu.

- Dôraz sa kladie na riešenie prevažne matematických problémov a textovo orientovaných úloh, napríklad: napíšte program, ktorý nájde korene kvadratickej rovnice.

Takýto prístup sa v súčasnosti považuje už za prekonaný.

Súčasný prístup

V súčasnosti preferujme nasledujúci prístup:

- Žiaci pracujú v prostrediach, akými sú napríklad Scratch, Imagine, Delphi, Lazarus.

- **S grafikou** a **s grafickom** prostredí sa žiaci stretávajú už **od začiatku**. Pozor však, grafika je iba jedným, hoci veľmi užitočným, nástrojov pre vizualizáciu algoritmov.

- Postupnosť tém: grafické príkazy, niektoré (2 alebo 3) dôležité komponenty, premenná, cyklus `for`, podmienený príkaz, cyklus `while`, ...

- Grafika nie je vyčlenená ako samostatná téma, ale je prepojená so všetkými témami, ktoré ďalej nasledujú.

- Okrem toho, že sa riešia aj úlohy s grafickým výstupom, ako napríklad kreslenie obrázkov, tvorba nápisov, neskôr sa riešia aj matematicky a textovo orientované úlohy, avšak nekladie sa na ne primárny dôraz.

Takýto spôsob vyučovania kladie zvýšené nároky na učiteľa a vyučovanie. Napríklad, v prípade prostredia Delphi, sa často stretávame s názorom, že komponenty celý program komplikujú. V tomto prípade je však chyba na strane učiteľa:

- Vieme, že pre starších a veľmi skúsených učiteľov je pomerne náročné zmeniť klasické prostredie TurboPascal za prostredie Delphi. Je to preto, lebo programy v Delphi alebo Lazaruse využívajú objekty a udalosťami riadené programovanie. Preto je aj náročné upraviť programy, príklady a úlohy z prostredia TurboPascal. Takmer vždy ich treba úplne zahodiť a vytvoriť nanovo.

- Treba si uvedomiť, že v prípade prostredia Delphi alebo Lazarus nie je cieľom učiť komponenty, ani grafiku, ani mechanizmus udalostí, ani tvorbu „profesionálnych" aplikácií. Komponenty, grafika a udalosti sú iba prostriedky, ktoré, ak ich správne (skromne) využijeme, získame pre vyučovanie veľa výhod: vynikajúce motivácie alebo nástroje na vizualizáciu algoritmov. Naším cieľom teda bude stále učiť základy programovania.

Vidíme, že ihneď od začiatku využívame grafiku, učíme sa základné grafické príkazy, používame farby, kreslíme. To je veľký rozdiel od tradičných prístupov, kde sa grafika nachádza takmer na konci. Vôbec pritom nebudeme mať pocit, že sme zavčasu minuli zaujímavé úlohy. V skutočnosti totiž platí úplný opak – vďaka takémuto prístupu môžeme neskôr využívať oveľa širšiu paletu zaujímavých príkladov.

Poradie tém pre programovanie v Delphi alebo Lazarus

Predstavme si, že máme za úlohu učiť žiakov programovanie. Naši žiaci sú začiatočníci, ktorý ešte v živote neprogramovali. Chcú sa však naučiť programovať, a keďže počuli, že pre začiatočníkov je vhodný Pascal, chcú sa ho naučiť. Na škole máme k dispozícii prostredie Delphi alebo Lazarus.

Stanoviť správne poradie tém je veľmi náročná koncepčná úloha. Väčšinou, keď začíname budovať nový predmet, zostavíme poradie tém na základe vhodnej literatúry, skúseností kolegov zo zahraničia, z podobnej oblasti, atď. Návrh potom prakticky overujeme a iteratívne meníme, vylepšujeme. Nezriedka sa stáva, že tento proces trvá aj niekoľko rokov.

Aktivita 3.3	V spoločnej diskusii navrhnite vhodné poradie tém, v akom by ste žiakov učili programovanie.

Voľba programovacieho jazyka a výber vývojového prostredia majú veľký vplyv na to, aké témy a v akom poradí budeme učiť, a tiež do značnej miery ovplyvňujú to, aké príklady budeme so žiakmi riešiť:

- V prostredí Imagine:
  ovládanie korytnačky → kreslenie → cyklus → udalosti → ...

- V prostredí Delphi alebo Lazarus:
  úvod do prostredia → kreslenie → premenné → cyklus → vetvenie → ...

Zhrnutie

Videli sme rôzne prístupy k vyučovaniu programovania. Pokým sme sa rozhodli učiť začiatočníkov programovať v profesionálnom programovacom jazyku, používame prostredie Delphi alebo Lazarus od úplného začiatku. Rozhodne nezačíname učiť programovanie v textových prostrediach. Filozofiu „najskôr sa učíme programovať v prostredí TurboPascal alebo FreePascal, až potom Delphi alebo Lazarus" považujeme za škodlivú. Podobne, v súčasnosti preferujeme taký prístup k vyučovaniu programovania, keď sa grafike venujeme hneď na začiatku. To je veľký rozdiel od tradičných prístupov, kde sa grafika nachádza takmer na konci. Ako postupne uvidíme, vôbec nebudeme mať pocit, že sme zavčasu minuli zaujímavé úlohy. V skutočnosti totiž platí úplný opak – vďaka takémuto prístupu môžeme neskôr využívať oveľa širšiu paletu zaujímavých príkladov. Pritom žiakov naučíme (minimálne) rovnaké koncepty, ako keby sme ich učili tradičným prístupom.

Kapitola 4: Programovanie v jazyku Pascal 1

V tejto rozsiahlej kapitole sa budeme zamýšľať nad vyučovacími hodinami z programovania na strednej škole. Pokúsime sa spoločne vymyslieť a ukázať, ako by sme učiť nasledujúce témy v jazyku Pascal a v prostredí Delphi alebo Lazarus:

- úvod do programovania,
- príkaz priradenia,
- grafické príkazy,
- komponenty a ich vlastnosti,
- typy údajov,
- náhodné čísla,
- premenné,
- cyklus `for`.

Každá z tém je značne rozsiahla. Preto sa každej téme v škole v skutočnosti venujem typicky dve až osem vyučovacích hodín.

Aktivita 4.1

Vypracuje vzorovú vyučovaciu hodinu pre každú z uvedených tém.

Pracujte v tímoch. Každý tím si zvolí jednu tému, ktorú vidíme na začiatku tejto kapitoly:

- V tíme spracujte tému tak, ako by sa mala učiť na jednej 45 minútovej hodine.
- Keďže každá z tém je veľmi široká, zamerajte sa „iba“ na jej úvodnú časť.
- Výsledok práce prezentuje zástupca tímu pri tabuli.

Prezentácia bude mať dve časti.

Prvá časť bude obsahovať **analýzu témy**:

- zoznam predpokladov – čo už žiaci vedia,
- cieľ – čo chcete naučiť,
- zoznam pojmov a poznatkov, ktoré sa žiaci naučia.

Druhá časť bude zrýchlená **ukážka** toho, ako by mala prebiehať vyučovacia hodiny:

- treba rešpektovať etapy poznávacieho procesu (motivácia, zbieranie skúseností...),
- čo a akým spôsobom budete vysvetľovať,
- čo budú riešiť žiaci samostatne.

Je výhodné, ak budete prvú časť prezentovať pomocou projektora. Druhú časť – samotnú ukážku vyučovacej hodiny prezentujte **bez projektora** (používajte iba kriedu a tabuľu).

Námety môžete čerpať námety z učebnice a materiálov DVUI.

Ostatní kolegovia na konci zhodnotia vystúpenie.

Môžete predpokladať, že výučba prebieha v počítačovej učebni, každý žiak sedí sám pri počítači.

Videoprojektor však z akýchsi dôvodov nefunguje.

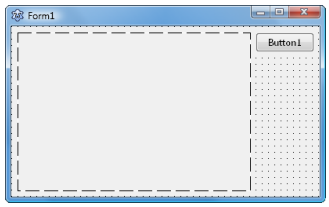
Môžete tiež predpokladať, že ste už zapísali do triednej knihy, prípadne zopakovali učivo z minulej hodiny.

Jednotlivé témy aj s fragmentmi ukážok uvádzame v nasledujúcom texte.

4.1 Úvod do programovania

Cieľ:

- zoznámiť žiakov s novým vývojovým prostredím,
- zoznámiť sa s ideou komponentov,
- vytvoriť prvý program, ktorý po stlačení tlačidla zobrazí pozdrav.



Nové pojmy, poznatky a zručnosti:

- formulár, komponent, spustenie a ukončenie programu,
- komponent, `Button` a `Image`, manipulácia s komponentmi, vlastnosť,
- procedúra, príkaz pre výpis textu.

Poznámky:

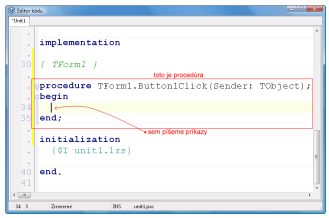
- Táto téma obsahuje veľmi veľa „technických“ detailov. Preto je dôležité, aby sme to s počtom informácií neprehnali – ich počet sa snažíme minimalizovať. Niektoré veci dokážu žiaci sami objaviť, niektoré úkony bude potrebné názorne predviesť (odporúčame použiť projektor).
- Prostredie Lazarus je momentálne vo vývoji a trpí množstvom nedostatkov. Pokým škola má na to prostriedky, odporúčame pracovať v prostredí Delphi.

V prostrediach Delphi aj Lazarus sa využívajú objekty. Znamená to, že s objektmi budú žiaci pracovať od začiatku. Objektovo-orientované programovanie je však náročné, preto jeho princípy nebudeme začiatočníkom vôbec vysvetľovať.

Namiesto zložitých vysvetlení použijeme vhodné metafory. Napríklad: „Image1.Canvas.TextOut je príkaz na výpis textu“. Ak to s metaforami nepreženieme, žiaci nám budú rozumieť. Detaily sa žiaci dozvedia postupne, až na ďalších vyučovacích

Postup:

1. Motivácia: „Budeme sa učiť vytvárať vlastné programy – také, ktoré môžeme spúšťať ako súbory exe.“
2. Spustíme prostredie a pomenujeme jeho základné časti. Venujeme sa iba tým, ktoré budeme na hodine používať. Prostredie odporúčame aj schematicky nakresliť na tabuľu.
3. Preskúmame komponenty. Ideu komponentov vysvetľujeme pomocou metafory: „vzhľad programu poskladáme z komponentov podobne, ako staviame stavbu z kociek Lega“. Ďalej:
   - Ukážeme, ako sa komponenty do formulára vkladajú (resp. odstraňujú).
   - Necháme žiakov, aby si prácu s komponentmi vyskúšali a chvíľu sa s nimi „pohrali“.
4. Program spustíme a upozorníme na to, že budeme rozlišovať formulár a spustený program (ten má tlačidlo na lište v operačnom systéme).
5. Úloha: „Vytvorte program s komponentmi *Button* a *Image*“. Ďalej už budeme pracovať iba s týmito komponentmi. Ich význam vysvetlíme takto:
   - „Tlačidlo (*Button*) budeme používať na spúšťanie príkazov.“
   - „Obrázok (*Image*) budeme používať na zobrazovanie výsledkov.„
6. Program spustíme a ukážeme, že tlačidlo zatiaľ nič nerobí. „Teraz program naučíme, aby nás po stlačení tlačidla pozdravil“:
   - Vo formulári dvojklikneme na tlačidlo. Ukážeme, že v okne s programom sa niečo zmenilo – vznikla procedúra.
   - Pojem *procedúra* zatiaľ vysvetľujeme iba tak, že je to „miesto, kam budeme zapisovať príkazy“.
   - Do procedúry, na správne miesto, zapíšeme príkaz pre výpis textu:
     ```
     Image1.Canvas.TextOut(100, 50, 'Ahoj');
     ```
   - Upozorníme na to, kam nesmieme napísať medzery. Upozorníme aj na symboly: bodky, apostrofy, čiarky, zátvorky, bodkočiarku.



V prostredí Lazarus musíme myslieť na to, že grafická plocha `Canvas` sa pri prvom použití zafarbí načierno. Ak nám to neskôr začne vadiť, naučíme žiakov príkaz na zmazanie plochy (napríklad príkazom `FillRect`).

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(100, 50, 'Ahoj');
end;
```

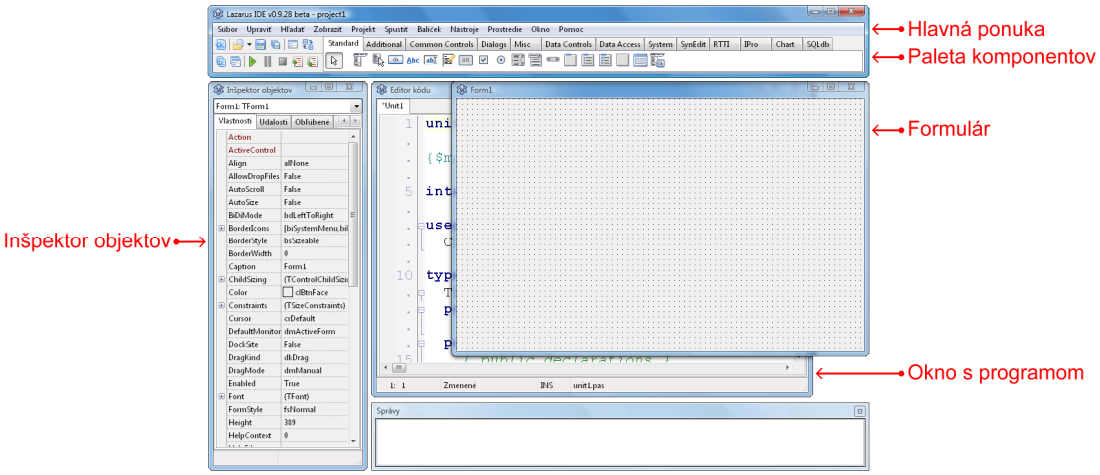
toto jedno slovo bez medzery – čiarka – apostrof – bodka – zátvorky – bodkočiarka

   - Program sa pokúsime spustiť a vyskúšať. Prípadne chyby odstránime.

Pozor: Cieľom nie je vysvetľovanie komponentov, ani ich vlastností, ani základy objektovo-orientovaného programovania, ani všetkého, čo v programe vidíme.

Ukážka:

1. Spustíme prostredie Delphi alebo Lazarus.

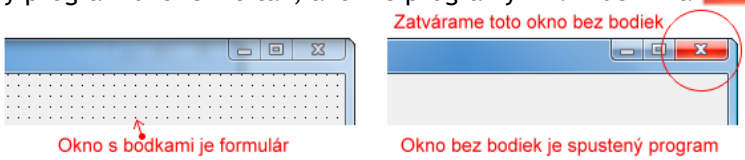


Prostredie má tieto časti: Hlavná ponuka, Inšpektor, Formulár, Program

2. Vytvoríme nový program (aplikáciu) príkazom z ponuky:
   - Delphi: File ▶ New ▶ Application
   - Lazarus: *Projekt* ▶ *Nový Projekt* ▶ *Projekt-Aplikácia*
3. Túto aplikáciu spustíme povelom tlačidlom alebo z ponuky:
   - Delphi: *Run* ▶ *Run*.
   - Lazarus: *Spustiť* ▶ *Spustiť*.
4. Na obrazovke uvidíme prázdne okno s nápisom `Form1`. Na lište bežiacich programov vo Windows uvidíme, že pribudol program s názvom *project1*.

zbieranie skúseností

5. Spustený program ukončíme tak, ako iné programy – kliknutím na .



Pozor, aby sme nezatvorili formulár!

6. Vzhľad aplikácie vytvoríme tým, že z **palety komponentov** postupne umiestnime **komponenty** do **formulára** (pripomína to skladanie stavby z kociek Lega):
7. Formulár predstavuje budúce okno našej aplikácie. Tak, ako teraz komponenty umiestnime do formulára, tak bude vyzerať naša aplikácia po spustení.
8. Teraz vložíme do formulára komponenty *Button* (**tlačidlo**) a *Image* (**obrázok**). Tlačidlom budeme spúšťať príkazy programu. Obrázok bude slúžiť ako výstup programu.

Komponent Button (tlačidlo) zo záložky Standard.

Komponent Image (obrázok) zo záložky Additional

9. Komponenty, ktoré sú už položené vo formulári môžeme pomocou myši presúvať a meniť im veľkosť.
10. Spustite program a vyskúšajte, ako sa správa.

4.2 Príkaz priradenia

Cieľ:

- porozumieť tomu, že záleží na poradí, v akom sa príkazy vykonávajú,
- porozumieť parametrom príkazu pre výpis textu,
- zoznámiť sa s priradením – zatiaľ iba na úrovni príkazu pre nastavenie písma,
- spoznávať súradnicovú sústavu (počítačová je oproti matematickej iná).

Programujem v Delphi

Nové pojmy, poznatky a zručnosti:

- `Font.Color, Font.Name, Font.Size,`
- príkaz s priradením `:=`

Postup:

Motivácia je založená na tom, že sa spoločne snažíme skúmať správanie sa programu, a zároveň riešiť drobné, elementárne problémy.

1. Spoločne so žiakmi budeme objavovať parametre príkazu `TextOut`:
   - Máme príkaz `Image1.Canvas.TextOut(100, 50, 'Ahoj')`
   - Úloha: „Chceme, aby počítač pozdravil práve nás, napríklad: Ahoj Janko!. Čo potrebujeme zmeniť? Vyskúšajte!“
   - Úloha: „Teraz chceme pozdrav vypísať približne do stredu grafickej plochy.“ Prezradíme, že čísla 100 a 50 určujú pozíciu textu.
   - Pýtame sa: „V čom je počítačová súradnica iná oproti matematickej?“
   - Doplníme, že čísla aj text medzi apostrofmi sú **parametre** príkazu.

Súradnice v týchto úlohách žiaci odhadujú. Tým získavajú prvé skúsenosť, ako funguje počítačová súradnicová sústava.

2. Spoznáme, že procedúra môže obsahovať **viac príkazov**:
   - Úloha: „Približne do stredu obrázku chceme zobraziť text *Programovanie* a pod neho *v Pascale*.“
   - Motivácia je založená na tom, že už nevystačíme iba s jedným príkazom. Riešenie však žiakom neprezrádzame. Ak bude potrebné, riešenie budeme postupne naznačovať.
   - Pozor na písanie `;`

Spočiatku je vhodné používať iba niekoľko farieb, ktorých mená dokážu žiaci ľahko vydedukovať (red, green, blue, yellow).

3. Spoznávame **príkaz pre nastavenie** farby písma:
   - Úloha: „Chceme farebné nápisy – slovo Pascal chceme červené“
   - Ak si uvedomíme riešenie, teda, že na vhodné miesto treba vsunúť príkaz
     `Image1.Canvas.Font.Color:=clRed;`
     potom túto úloha môžeme chápať ako motiváciou k **príkazu priradenia**.
   - Vysvetlíme, z čoho je zápis `:=` zložený.
   - „Vidíte v príkaze meno červenej farby?“
   - „Ako by vyzeral názov zelenej farby? Vyskúšajte, či funguje!“
   - „Chceme zafarbiť obe slová (*Programujem*, v *Delphi*) rôznymi farbami“

Programujem
v Pascale

4. Postupne sa naučíme nastavovať aj typ písma a veľkosť písma:
   - **Veľkosť písma** nastavíme takto: `Image1.Canvas.Font.Size:=24;`
   - **Typ písma**: `Image1.Canvas.Font.Name:='Courier New';`
   - Príklady vyskúšame. Žiakov sa oplatí pýtať: „Kam treba príkazy zapísať?“
   - Ďalšia úloha: „Vyskúšajte vypísať slová veľkým písmom *Arial*“

Je potrebné upozorňovať žiakov na pekné a správne **formátovanie programu**: 1 príkaz = 1 riadok.

5. Spolu so žiakmi vymyslíme **syntaktické pravidlo** pre nastavenie písma:
   Image1.Canvas.Font.vlastnosť := hodnota;
   Poznamenajme, že vo väčšine učebníc sa takýmito syntaktickými pravidlami začína – teda, autor ich čitateľovi prezradí. My sa však budeme snažiť, aby takéto pravidlá naši žiaci postupne objavovali.

6. Riešiť úlohy, v ktorých sa postupne kombinujú spomínané príkazy. Napríklad, vypisujú sa iné farebné pozdravy, „hviezdičky v rohoch obrazovky“...

Pozor: Cieľom nie je matematicky presne počítať súradnice – to budeme chcieť až neskôr. Cieľ nie je vedieť mená farieb naspamäť. Niektorým fontom sa nedajú meniť rozmery ľubovoľne. Nezadávame úlohy, ktoré vyžadujú viac ako 10 príkazov.

Ukážka:

1. Skúsime program naučiť to, aby nám po stlačení tlačidla pekným farebným nápisom prezradil, v akom jazyku programujeme.

motivácia

2. Texty, ktoré píšeme do grafickej plochy, môžeme písať **farebne** a rôznymi písmami. Napríklad, takto vypíšeme červený text:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.Font.Color:=clRed;
  Image1.Canvas.TextOut(50, 80, 'Programujem');
end;
```

zbieranie skúseností

3. V programe sme použili pre nastavenie farby písma nový **príkaz priradenia**. Tento príkaz obsahuje symboly `:=` (dvojbodka rovná sa). Na pravej strane od `:=` je názov červenej farby `clRed` (`cl` = skratka zo slova **color**, `Red` = červená). Príkaz pre nastavenia farby môžeme čítať nasledovne: *„obrázok 1, tvoja grafická plocha, tvoje písmo, jeho farba nastav na červenú“*.

4. Ako by ste vypísali pod slovo *„Programujem“* modrou farbou text *„v Pascale“*?
   - Ako by ste vypísali iba text *„v Pascale“*?
   - Videli sme, že červená farba má názov `clRed`. Ako by sa asi mohla nazývať modrá farba? ... `clBlue`.
   - Kam treba napísať `Image1.Canvas.Font.Color:=clBlue;`?

motivácia

```
procedure TForm1.Button1Click(Sender: Tobject);
begin
  Image1.Canvas.Font.Color:=clRed;
  Image1.Canvas.TextOut(30, 80, 'Programujem');
  Image1.Canvas.Font.Color:=clBlue;
  Image1.Canvas.TextOut(30, 120, 'v Pascale');
end;
```

zbieranie skúseností

5. Aké názvy by mohli mať farby: čierna, zelená, biela, žltá?
   - `clBlack`, `clRed`, `clGreen`, `clWhite`, `clYellow`

6. Aký názov by mohla mať hnedá farba?
   - To je podraz! Nebude to `clBrown` ale `clMaroon`.
   - Názvy farieb si nemusíme pamätať, lebo neskôr uvidíme, že sa dajú ľahko zistiť.

7. Čo sa stane, keby v programe chýbal prvý príkaz, ktorým nastavujeme červenú farbu? Ako sa program správa? Vyskúšajte.
   - Po prvom kliknutí na tlačidlo uvidíme vypísane čiernou farbou slovo *Programovanie* a červenou farbou *v Pascale*.
   - Po druhom kliknutí budú obe písma červené.

8. Vieme, že v program Word dokáže písať texty rôznymi fontmi: Arial, Times New Roman, Courier New... Aj v našom programe môžeme používať tieto **fonty** a vypisovať texty. Vyskúšajte:

motivácia

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.Font.Color:=clRed;
  Image1.Canvas.Font.Name:='Arial';
  Image1.Canvas.Font.Height:=24;
  Image1.Canvas.TextOut(30, 80, 'Programovanie');
  Image1.Canvas.Font.Name:=' Courier New';
  Image1.Canvas.Font.Color:=clBlue;
  Image1.Canvas.TextOut(30, 120, 'v Pascale');
end;
```

zbieranie skúseností

9. Vidíme, že vlastnosti písma, nastavujeme pomocou príkazu priradenia takto:

zovšeobecňovanie

meno vlastnosti `:=` *nová hodnota*

4.3 Grafické príkazy

Cieľ:

- spoznať základné príkazy pre kreslenie obdĺžnika a elipsy,
- porozumieť mechanizmu, ako sa kreslia elipsy, kruhy a štvorce,
- spoznať možnosť pre nastavenie farby výplne a obrysu.

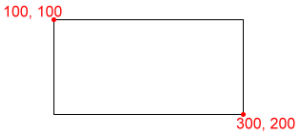
Nové pojmy, poznatky a zručnosti:

- grafická plocha, výplň, obrys,
- príkazy, nastavenia: `Rectangle`, `Ellipse`, `Brush.Color`, `Pen.Color`.

Poznámky:

- Táto téma, aj keď to tak nevyzerá, je pomerne rozsiala. Je potrebné nechať žiakom dostatok času na riešenie úloh a na to, aby si zvykli na zápisy, príkazy a ich fungovanie.
- Začneme kreslením obdĺžnikov, pretože pri kreslení elíps už využijeme poznatky z kreslenia obdĺžnikov.
- Je dôležité, aby žiaci postupne pochopili `Image1` ako komponent, `Canvas` ako jeho **súčiastku**, ktorá má nastavenie pre písmo, obrys, výplň a pozná rôzne príkazy pre kreslenie.

Grafika je pre žiakov sama o sebe (zatiaľ) dostatočne motivujúca. Preto ako motiváciu môžeme použiť skôr ozrejmenie toho, čo budeme so žiakmi ďalej robiť, napríklad: „Zatiaľ sme vypisovali iba texty. Teraz sa naučíme kresliť geometrické útvary ako sú obdĺžniky alebo elipsy.“ Prípadne môžeme ukázať, aké obrázky budeme kresliť.



Postup:

1. Činnosť príkazu `Rectangle` budeme objavovať spoločne so žiakmi:
   - Začneme priamo triviálnym príkladom: „**Obdĺžniky** kreslíme príkazom `Image1.Canvas.Rectangle(100, 100, 300, 200);`“
   - Diskutujeme o tom, ako sa obdĺžnik nakreslí aj o parametroch príkazu.
   - Obdĺžnik aj s uvedenými súradnicami nakreslíme na tabuľu.
   - Pýtame sa aj: „Aké rozmery má obdĺžnik?“
2. Bude potrebné, aby žiaci zbierali ďalšie skúsenosti s tým, ako príkaz funguje:
   - „Aký obdĺžnik nakreslí príkaz `Rectangle(100, 100, 200, 200)`?“
   - „Ako nakreslíme **štvorec** so stredom v bode 100, 100 a stranou dĺžky 50?“ Pripomeňme, že „štvorec je vlastne špeciálny prípad obdĺžnika“.
   - Predchádzajúce príklady sú vhodnou príležitosťou na to, aby sme žiakom prezradili, že parametre príkazu môžu byť aj **jednoduché výrazy** (zatiaľ iba s operátormi `plus`, `mínus`).
3. Náročnejšia úloha na trénovanie: “Pomocou obdĺžnikov nakreslite robota.“
4. Naučíme sa používať **obrys** a **výplň**. Ich použitie ukazujeme na jednoduchom príklade, napríklad kreslíme obdĺžnik so žltým vnútrom a červeným obrysom.
5. Úloha na trénovanie: “Pomocou obdĺžnikov nakreslite vlajku Nemecka.“
6. Podobný učíme aj kreslenie **elipsy** a **kruhu**. Nezabudnime, že žiakom treba vysvetliť ideu, že “elipsa sa vpisuje do neviditeľného obdĺžnika". Nakoniec zvolíme vhodnú úlohu na trénovanie, napríklad: “Nakreslite snehuliaka”.
7. Úlohy, v ktorých sa kombinuje kreslenie obdĺžnikov a elíps: „Nakreslite strom (pomocou kruhu a obdĺžnika)“, „Nakreslite značku *zákaz vjazdu*“, „nakreslite autíčko (karoséria z dvoch obdĺžnikov, kolesá z kruhov)“, ...
8. Neskôr naučíme aj kresliť **úsečky**, namiešať **vlastné farby** viď. [7].

Všimnime si, že:

- Začíname **triviálnym** príkladom.
- Potom riešime úlohy, ktoré majú **zložitejšie** parametre alebo parametre podľa stanovených kritérií.
- Nakoniec sú úlohy na **trénovanie**.

Úlohy **gradujeme** tým, že postupne kladieme určité nároky na parametre, ďalej kreslíme obrázky , ktoré obsahujú väčší počet obdĺžnikov, ktoré môžu byť prípadne zafarbené, až nakoniec kreslíme obrázky, v ktorých tieto veci kombinujeme.

Pozor: Nezačíname príkladom „na ktorom všetko vysvetlíme“. Nezačíname ani tým, že vymenujeme príkazy, ich parametre x_1, y_1, x_2, y_2 a budeme vysvetľovať „toto sú súradnice ľavého horného rohu, ...“. Takéto všeobecné zhrnutie patrí až na koniec a navyše má zmysel, ak ho objavujeme spoločne so žiakmi. Cieľom nie je ani kreslenie zložitých obrázkov – už aj kreslenie robota je na hranici únosnosti, pretože žiaci musia počítať súradnice pre 6 obdĺžnikov (hlava, trup, ruky a nohy = precízny výpočet 24 čísel!).

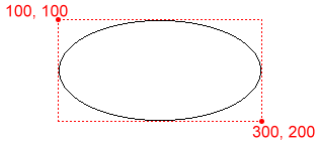
Ukážka:

motivácia

1. Okrem obdĺžnikov a štvorcov dokáže počítač kresliť elipsy a kruhy. Viete, ako vyzerá elipsa?
2. Na počítači kreslíme elipsu takto zvláštne – príkaz na kreslenie **elipsy** sa podobá príkazu na kreslenia obdĺžnika:

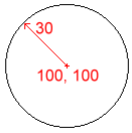
```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Image1.Canvas.Ellipse(100, 100, 300, 200);
end;
```

3. Ako príkaz funguje?



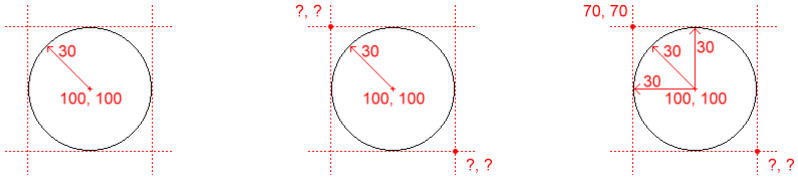
Súradnice v príkaze Ellipse sú vrcholy obdĺžnika. Ten sa však nenakreslí, ale nakreslí sa elipsa, ktorá je do vnútra obdĺžnika vpísaná.

4. Ako nakreslíme **kruh**? ... Podobne, ako štvorec. Kruh je špeciálny prípad elipsy.
5. Ako nakreslíme kruh so stredom v bode 100, 100 a polomerom 30?



zbieranie skúseností

6. Musíme si rozmyslieť súradnice štvorca, do ktorého sa vpíše kruh:



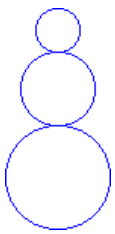
Ak si kruh nakreslíme a do obrázka postupne naznačíme jeho stred a polomer, ľahko prídeme na to, aké parametre potrebujeme zadať pre príkaz Ellipse.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Image1.Canvas.Ellipse(100-30, 100-30, 100+30, 100+30);
end;
```

7. Parametre v príkaze `Ellipse` by sme vedeli zapísať aj takto:

```
Image1.Canvas.Ellipse(70, 70, 130, 130);
```

8. Už vieme kresliť krúžky. Pokúste sa nakresliť takéhoto snehuliaka:



trénovanie poznatku

Snehuliak je poskladaný z troch snehových gúľ. Veľkosť krúžkov si sami stanovte. Na snehuliakovi musia byť presne zarovnané tak, aby nespadli. Modrú farbu čiar nastavíte príkazom: `Image1.Canvas.Pen.Color:=clBlue`.

4.4 Komponenty a ich vlastnosti

Cieľ:

- porozumieť pojmu vlastnosť,
- spoznať možnosť nastavovať komponentom vlastnosti počas behu programu,
- porozumieť operátoru bodka . zatiaľ iba na úrovni mechanizmu, ktorým oslovujeme komponent a jeho súčasti.

Nové pojmy, poznatky a zručnosti:

- vlastnosť, meno komponentu, nastavenie vlastnosti počas behu programu,
- mená vlastností `Caption`, `Left`, `Top`, `Width`, `Bottom`.

Porovnajme stratégie:

- Pri tradičnom prístupe sa príkaz priradenia spomína prvýkrát až pri téme premenná.
- Naši žiaci si začínajú zvykať na príkaz priradenie skôr, ako budú s premennými pracovať. Neskôr sa nemusia učiť zároveň dve nové veci (premenné a príkaz priradenia).

Treba si zároveň uvedomiť, že túto výhodu sme získali vďaka prostrediu Delphi (Lazarus). Je to preto, lebo nie každý jazyk (resp. prostredie) nám umožňuje aplikovať takýto prístup.

Poznámka: Táto téma bude mať charakter objavovania a skúmania zákonitostí, ako sa pracuje s vlastnosťami komponentov. Pravdepodobne budeme využívať motivácie „skúsime, čo sa stane...“,.“preskúmame, čo zápis znamená...“

Postup:

1. Preskúmame Inšpektor objektov:
   - Všimneme si **meno** komponentu v hornej časti okna Inšpektor. Predvedieme, že každý komponent má iné meno. Ak do formulára pridáme ďalšie tlačidlo, dostane svoje nové a jedinečné meno.
   - Všimneme si aj **zoznam vlastností**. Predvedieme, že po označení iného komponentu (napríklad `Image1`) vidíme iný zoznam.
2. Preskúmame tlačidlo a jeho **vlastnosti**. Je dôležité spoznať, že každá vlastnosť má svoj **názov** a **hodnotu**. Zameriame sa iba na vlastnosti:
   - `Caption` - zmeníme nápis napríklad na `Sem klikni`.
   - `Left`, `Top`, `Width`, `Height` - preskúmame, čo sa deje, ak ich meníme.
   - Upozorníme na to, že niektoré predchádzajúce vlastnosti sa zmenia aj vtedy, keď komponent myšou premiestnime alebo zmeníme jeho rozmery.
3. „V Inšpektore objektov sme nastavovali vlastnosti komponentov pred spustením programu. Teraz sa naučíme **nastavovať** vlastnosti aj počas behu programu“. Pri kliknutí na tlačidlo sa vykoná príkaz: `Button1.Caption:='Kuk';`
4. Vysvetlíme predchádzajúci zápis:
   - `Button1` je meno komponentu, `Caption` je názov vlastnosti.
   - Používame metaforu: „Bodkou . **oslovujeme** súčasti komponentu“
5. Ukážeme rôzne spôsoby nastavovania číselných vlastnosti. Experimentujeme s tlačidlom `Button1` - predpokladáme, že leží pri ľavom okraji:
   - Začneme jednoduchým príkladom: `Button1.Left:=100`.
   - Neskôr vyskúšame aj takýto zápis: `Button1.Left:=100+100`.
   - Nakoniec budeme analyzovať komplikovanejšie nastavenie:
     `Button1.Left:=Button1.Left+10`
     Zápis treba vysvetliť a prípadne trasovať. Vidíme aj, že v príklade sa prvý krát **zisťuje** hodnota nejakej vlastnosti.

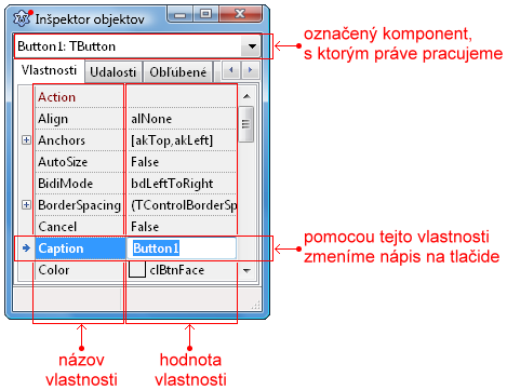
Všimnime si, že výraz na pravej strane od `:=` robíme zložitejším iba **po malých krokoch**. Robíme tak preto, aby žiaci dokázali vnímať a a vyhodnocovať rôzne situácie.

6. Spolu so žiakmi zhrnieme a objavíme predpis, ako sa nastavujú vlastnosti“
   komponent`.`*vlastnosť* `:=` *výraz*
   Z predchádzajúcich príkladov by mal vzísť poznatok, že *výraz* sa **najskôr vyhodnotí** a až podľa výslednej hodnoty sa **potom nastaví** vlastnosť.
7. Ďalšie námety na trénovanie:
   - Projekt *Ako sa máš?* na strane 12 v učebnici [7] (prípadne aj úloha 1).
   - Prvá úloha s **editovacím políčkom** v učebnici [7] v kapitole *2.3*.

Pozor: Je nesprávne, ak by učiteľ vysvetľoval význam všetkých vlastností rôznych komponentov.

Ukážka:

1. Po tom, ako sme formulára vložili tlačidlo, mohli sme pomocou myši nastaviť jeho polohu alebo veľkosť. Poloha aj veľkosť sú **vlastnosti** tlačidla. Tlačidlo má aj iné vlastnosti - anglicky *properties*. Napríklad, aj nápis na tlačidle, `Button1`, je vlastnosť. Aj tá sa dá zmeniť, môžeme tak urobiť v okne *Inšpektor Objektov*. Poďme teraz preskúmať, aké veci sa v tomto okne zobrazujú:



V okne Inšpektor Objektov sú teraz zobrazené vlastnosti tlačidla `Button1`.

2. Každá vlastnosť má svoje **meno** a **hodnotu**. Napríklad vlastnosť:
   - `Left` a `Top` určujú pozíciu tlačidla vzhľadom na okraje formulára,
   - Čo môžu znamenať vlastnosti `Width` a `Height`? ... šírku a výšku tlačidla,
   - `Caption` je nápis na tlačidle.
3. Upravte nápis na tlačidle tak, aby na ňom bolo napísané „*Klikni na mňa*“.
4. Zmeňte polohu a rozmery tlačidla tak, aby tlačidlo ležalo pri ľavo okraji a malo veľkosť na 150 bodov na šírku a 50 bodov na výšku.
5. Vlastnosti komponentov vieme meniť aj počas behu programu, napríklad takto:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Caption:='Kuk';
end;
```

*Príkaz čítame: „tlačidlo 1 - **tvoj** nápis - nastav na Kuk“.*

6. Ako nastavíme pozíciu tlačidla, aby malo od ľavého okraja vzdialenosť 100?

```
  Button1.Left:='Kuk';
```

7. Vidíme, že k vlastnostiam komponentov pristupujeme pomocou `.` (bodky):

```
komponent.vlastnosť
```

8. Vyskúšajte, čo sa stane, ak predchádzajúci príkaz takto zmeníme:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Button1.Left:=Button1.Left+100;
end;
```

9. Prečo tlačidlo skacká smerom vpravo? Trasujme, ako sa mení vlastnosť `Left`:

Po spustení programu		`Left=0`
Po 1. stlačení tlačidla	`Button1.Left:=Button1.Left+100;` (najskôr sa vyhodnotí výraz: `0 + 100`; potom sa nastaví vlastnosť: `100`)	`Left=100`
Po 2. stlačení tlačidla	`100 + 100`	`Left=200`
Po 3. stlačení tlačidla	`200 + 100`	`Left=300`

Vidíme, že hodnota vlastnosti `Left` sa po každom stlačení zväčšuje o 100. Keďže sa táto hodnota zväčšuje, tlačidlo sa posúva smerom vpravo.

motivácia

zbieranie skúseností

zovšeobecňovanie

zbieranie skúseností

zovšeobecňovanie

4.5 Typy údajov

Cieľ:

- porozumieť tomu, že pre počítač sú čísla, text a farby rôzne typy údajov,
- spoznať základné matematické operácie, typy údajov, konverziu typov,
- zároveň budeme túto tému chápať ako prípravu pre prácu s premennými.

Nové pojmy, poznatky a zručnosti:

- typ, reťazec `string`, celé číslo `Integer`, desatinné číslo `Real`,
- **`div`**, **`mod`**,
- konverzia typov, `IntToStr`, `FloatToStr` a `StrToInt`.

Poznámka: Budeme vychádzať zo skúseností, ktoré žiaci doposiaľ nazbierali. Preto si môžeme v tejto časti dovoliť **upratovať doposiaľ nazbierané skúsenosti**. Upratovanie vedomostí je dôležité, pretože nás môže posunúť v poznaní ďalej, smerom k abstraktnému uvažovania. Príliš skoré upratovanie – teda v čase, keď ešte žiaci nenazbierali dostatok skúseností – však pôsobí kontraproduktívne. Typickým príkladom zlého upratovania je, keď sa na prvej hodine dozvie začiatočník o rôznych typoch a ich vlastnostiach.

Postup:

Typy `Integer` a `string` sú pre nás momentálne najdôležitejšie. Ostatné typy budeme spomínať, až keď ich budú žiaci potrebovať. Robíme tak preto, aby sme žiakov zbytočnými nezaťažovali informáciami.

1. Žiaci videli, že „príkaz `TextOut` má parametre *číslo*, *číslo* a *text* – pre počítač sú toto rôzne **typy** údajov“. Povieme, že „typy majú svoje mená“:
   - Budeme používať typy `Integer` a `string`.
   - Oplatí sa porovnať ideu typov s množinami, napríklad: „Typ `Integer` je množina celých čísel typ `string` je **množina** textov.“
2. Žiaci by si mali postupne a nenásilne zvyknúť na odbornú terminológiu:
   - „Slovo `string` v programovaní prekladáme ako **textový** reťazec.“
   - „Príkaz `TextOut` má parametre typu `Integer, Integer, string`.“
3. Upozorníme na to, že hodnoty rôznych typov sa nedajú ľubovoľne zamieňať:
   - „`TextOut(0, 0, '123')` je iný typ, ako `TextOut(0, 0, 123)`"
   - „Niekedy chceme vypísať výsledok výpočtu `TextOut(0, 0, 1+2*3)`"
   - Je potrebné povedať, že predchádzajúci príkaz je nesprávny s odôvodnením, že „`TextOut` dokáže vypisovať iba textové reťazce“.

Motivácia k nutnosti konvertovať čísla na texty je založená na tom, že „príkaz `TextOut` je *nešikovný* a *nevie* vypisovať čísla“.

4. Pokračujeme vysvetlením: „Ak chceme vypisovať čísla, musíme ich zmeniť na text. Používame na to príkaz `IntToStr`, ktorý z čísla **vyrobí** text.“:
   - Ukážeme jeho použitie na elementárnom príklade `IntToStr(1+2*3)`.
   - Treba vysvetliť, z čoho vznikol názov `IntToStr` (t.j. **int**eger **to str**ing).
   - Budeme vypisovať aj hodnoty výrazov s inými operátormi (zátvorky, mínus, delenie), keďže „teraz môžeme používať počítač ako kalkulačku“.

Všimnime si, že zatiaľ `IntToStr` nenazývame funkciou.

5. Žiaci sa pravdepodobne stretnú s tým, že delenie „pokazí typ“ výsledku:
   - Typ výsledku `4/3` bude **desatinné číslo**, a to aj v prípade `4/2`.
   - Povieme, že desatinné čísla budeme učiť neskôr.
6. Naučíme žiakov používať **celočíselné delenie**. Princíp fungovania operátora **`div`** vysvetľujeme na jednoduchej ukážke, napríklad `7 div 3`.
7. Námety na trénovanie:
   - Zostaviť výraz/program, ktorý spočíta a vypíše súčet 5 celých čísel.
   - Zostaviť výraz, ktorý vypočíta a vypíše priemer 5 celých čísel.
   - Kedy bude výsledok celé číslo a kedy nie?

V úlohách na trénovanie budú výrazy s konštantami.

Pozor: Je nesprávne vymenovať zoznam typov, aké pozná jazyk Pascal. Z pohľadu žiakov to budú zbytočné informácie, z pohľadu vyučovania teda zbytočná strata času.

motivácia

zbieranie skúseností

Ukážka:

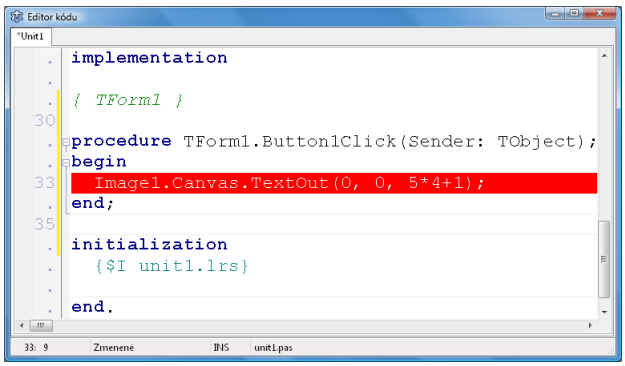
1. Všimnime si parametre príkazu pre výpis textu:

```
Image1.Canvas.TextOut(100, 50, 'Ahoj!');
                      číslo číslo text
```

*Prvé dva parametre v príkaze sú **čísla**. Tretí, posledný parameter je **text**.*

2. Máme teraz prirodzenú otázku - môžeme čísla a texty zamieňať? Môžeme napríklad napísať `Image1.Canvas.TextOut(0, 0, 123)`?
   - `Image1.Canvas.TextOut(0, 0, 123)` ... **nesprávne** `123` je číslo
   - `Image1.Canvas.TextOut(0, 0, '123')` ... **správne** `'123'` je text
3. Jazyk Pascal je citlivý na to, kde a aké typy údajov používame. Čísla a texty sú pre počítač rozdielne **typy údajov**.
4. Čísla `0`, `100`, `50` alebo `123` sú **celé čísla**. V matematike by sme povedali, že patria do množiny celých čísel, ktorú označujeme písmenkom $\mathbb{Z}$. V jazyku Pascal existuje podobná množina, ktorá sa nazýva `Integer` - *celé číslo*. Keďže pri programovaní miesto slova *množina* používame slovo *typ*, hovoríme, že:
   - *„100 je typu celé číslo“* alebo
   - *„prvé dva parametre príkazu `TextOut` musia byť typu celé číslo“.*
5. Aj texty `'Ahoj!'`, `'Kuk'` alebo `'Klikni na mňa'` alebo `'123'` sú v jazyku Pascal prvky špeciálnej množiny, ktorá má názov **`string`** - **textový reťazec**. Pri programovaní hovoríme, že:
   - *„text 'Ahoj!' je typu string“* alebo *„'Ahoj!' je typu textový reťazec“*, či
   - *„tretí, posledný parameter príkazu `TextOut` musí byť textový reťazec“.*
6. V zápisoch v programe odlíšime textové reťazce od čísel alebo príkazov tým, že znaky textového reťazca napíšeme medzi apostrofy.
7. Čo však máme robiť v situáciách, keď chceme zobraziť výsledky výpočtov? Skúsme, čo sa stane, ak sa pokúsime vypísať hodnotu výrazu `5*4+1` takto:

```
Image1.Canvas.TextOut(0, 0, 5*4+1);
```
... chyba



Program nespustíme. Červený riadok v programe zvýrazňuje riadok s chybou.

8. Takéto riešenie nechceme:

```
Image1.Canvas.TextOut(0, 0, '5*4+1');
```
... zle, zobrazí sa **`5*4+1`**

9. Správne riešenie vyzerá takto:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(0, 0, IntToStr(5*4+1));
end;
```

***`IntToStr`** je skratka zo slov Integer To String = zmeň Celé číslo Na Reťazec.*

10. Najskôr sa vyhodnotí aritmetický výraz `5*4+1`. Až potom sa výsledok, teda celé číslo 21, **skonvertuje** na postupnosť cifier 2 a 1, teda reťazec `'21'`.

4.5 Náhodné čísla

Cieľ:

- naučiť používať príkaz `random` na generovanie náhodných čísel

Nové pojmy, poznatky a zručnosti:

- náhodné číslo, príkaz `random`,
- použitie príkazu na generovanie čísel rôznych rozsahov.

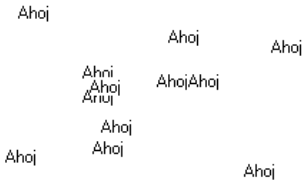
Náhoda a náhodné čísla hrajú v informatike a matematike významnú úlohu. Vďaka náhode dokážeme riešiť, aspoň približne, mnohé, zatiaľ prakticky neriešiteľné problémy. My však takéto vysoké informatické koncepty nebudeme teraz sledovať. Žiakov zatiaľ hravou formou zoznámime s náhodou a náhodnými číslami.

Poznámka: Každú z tém premenná, cyklus, podmienený príkaz atď., by sme dokázali učiť aj bez náhodných čísel. Náhodné čísla však budú mať pre nás niekoľko výhod:

- Náhodné čísla môžeme použiť na oživenie programu, ktorý kreslí rôzne náhodné variácie – náhoda bude slúžiť ako **motivácia**,
- pri úlohách, ktoré kreslia náhodne umiestnený krúžok bude potrebné používať premenné – náhodné čísla budú slúžiť ako **zdroj údajov**,
- pri cykloch budeme kresliť veľa náhodne rozmiestnených tvarov – náhodné čísla nám slúžiť ako **zdroj príkladov**.

Postup:

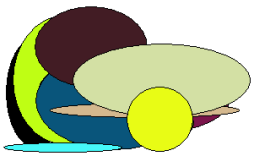
1. V krátkosti iba stručne naznačíme čosi **o náhode** v programoch a ihneď ukážeme príklad, v ktorom sa používa príkaz `Random`:
   - „Doposiaľ fungovali naše programy veľmi presne. Napríklad, pozdrav sa vypísal na súradnice, ktoré sme presne vypočítali. Pozdrav však môžeme vypísať aj na náhodné miesto takto:“
     `Image1.Canvas.TextOut(Random(400), Random(300), 'Ahoj')`
   - Povieme žiakom, aby program vyskúšali.
   - Opýtame sa, „Kam sa vypíše pozdrav, po stlačení tlačidla?“
   - Zhrnieme: „Vidíme, že program vypisuje pozdrav vždy na iné miesto.“
2. Bude potrebné vysvetliť, ako „príkaz“ `Random` pracuje: „`Random` si vymyslí **náhodné číslo**. Až toto číslo sa použije ako parameter príkazu `TextOut`. Napríklad, `Random(400)` vymyslí číslo, ktoré bude zaručenie menšie ako `400`. Presnejšie, vymyslené číslo bude ležať v intervale od `0` po `399`.“
3. Bude potrebné riešiť úlohy na spoznávanie a trénovanie náhodných čísel:
   - Úloha: „Aké čísla bude zobrazovať príkaz
     `Image1.TextOut(0, 0, IntToStr(Random(10))`?“
   - Postupne sa pýtame aj na to, aké čísla by sa vypisovali pre rôzne **parametre** `Random(100)`, `Random(2)`, `Random(1)`. A naopak, môžeme sa pýtať aj na to, aký parameter musíme použiť, aby sa generovali náhodné čísla **rozsahov** od `<0, 49>` alebo `<0, 50>`.
   - Komplikovanejšia úloha: „Vidíme, že príkaz `random` môžeme použiť ako hraciu kocku. Ako dosiahneme, aby sa zobrazovali čísla od `1` po `6`?“
   - Náročné úlohy: „Chceme generovať náhodné čísla z rozsahu `<-10, 10>`“, alebo „Chceme náhodne generovať iba čísla `+1` alebo `-1`.“
4. Ďalej môžeme:
   - Kresliť so žiakmi náhodne umiestnené červené obdĺžniky.
   - Naučiť žiakov používať náhodné farby (napr. `Random(256*256*256)`). Potom môžeme so žiakmi kresliť náhodne zafarbené geometrické tvary, náhodne umiestnené nápisy náhodnej veľkosti,...
   - Námety sú aj v učebnici [7] v kapitole 2.1.



Všimnime si, ako úlohy s generovaním náhodných čísel postupne gradujeme:

- najskôr sa pýtame na konkrétne hodnoty,
- potom na abstraktné.

Ulohy, v ktorých používame príkaz `Random` v rámci zložitejšieho výrazu a generujeme náhodné čísla podľa komplikovanejších kritérií, riešime až nakoniec.



Pozor: Náhodné čísla používame opatrne – vyhýbame sa rôznym trikom a efektom s náhodnými číslami ako aj komplikovaným výrazom s `Random`.

Ukážka:

motivácia

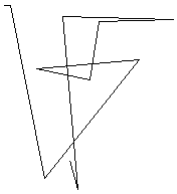
1. Naše programy sa zatiaľ správali veľmi presne - vždy nakreslili rovnaký útvar. Bolo to preto, lebo súradnice geometrického útvaru sa nemenili.
2. Vyskúšajme takýto príklad:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(Random(400), Random(300), 'Ahoj');
end;
```

3. Čo sa stane, keď klikneme na tlačidlo? Čo sa bude diať, keď na tlačidlo opakovane klikáme?
4. Pozdravy sa vypisujú vždy na iné miesta na obrazovke. *Random* znamená ***náhodné***. Random funguje takto:
   - `Random(400)` zvolí niektoré (náhodné) číslo spomedzi čísel od `0` po `399`.
   - `Random(300)` zvolí niektoré (náhodné) číslo spomedzi čísel od `0` po `299`.
5. Čo myslíte, čo bude kresliť tento program?

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.LineTo(Random(400), Random(300));
end;
```

zbieranie skúseností



Po stlačení tlačidla sa prikreslí náhodná úsečka.

6. Vidíme, že Random má jeden parameter. Viete prísť na to, ako tento parameter funguje? Zistite, aké čísla bude zobrazovať tento program:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(0, 0, IntToStr(Random(10)));
end;
```

Program bude zobrazovať čísla od 0 po 9.

zovšeobecňovanie

7. Parameter určuje rozsah hodnôt pre **vygenerovanie náhodného čísla**. `Random(n)` náhodne zvolí niektoré číslo spomedzi čísel od `0` po `n-1`.
8. Aké čísla by sa generovali, keby sme napísali:
   - `Random(100)` ... od 0 po 99
   - `Random(2)` ... od 0 po 1
   - `Random(1)` ... stále iba 0

zbieranie skúseností

9. Ako by sme vygenerovali náhodné čísla z takéhoto rozsahu:
   - od 0 po 49 ... `Random(50)`
   - od 0 po 50 ... `Random(51)`

trénovanie poznatku

10. Poznáte hru Človeče nehnevaj sa? Pri jej hre používame kocku, na ktorej padajú čísla od 1 po 6. Vytvorte program, ktorý bude simulovať hraciu kocku tak, že po stlačení tlačidla vygeneruje náhodné číslo od 1 po 6 a zobrazí ho do grafickej plochy.

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(0, 0, IntToStr(1+Random(6)));
end;
```

4.6 Premenné

Cieľ:

- porozumieť pojmu premenná a jej použitiu v jazyku Pascal,
- naučiť sa priraďiť hodnotu do premennej a použiť hodnotu premennej.

Treba myslieť na to, že premenné v matematike majú trochu iný význam, ako v počítači:

- Matematická premenná označuje neznámu, hľadanú hodnotu alebo parameter.
- Programátorská premenná je miesto v pamäti počítača na zapamätanie hodnoty.

Nové pojmy, poznatky a zručnosti:

- premenná, meno premennej, deklarácia,
- `var`, syntaktické pravidlá deklarácie, miesto kde deklaráciu píšeme,
- operácie s premennými: nastavenie, použitie, výpis, načítanie.

Poznámky:

- Túto tému sme čiastočne spracovali v module Didaktika programovania 1.
- Premenné sú pre programovanie veľmi dôležité, ale pre začiatočníka sú pritom veľmi náročné na porozumenie. Preto musíme pri vyučovaní postupovať veľmi obozretne a po malých krokoch.
- Žiaci môžu spoznať premenné už skôr, napríklad v prostrediach Imagine či Scratch. Teraz ale budeme predpokladať, že premenné ešte nepoznajú.

Postup:

Ukážková úloha nesmie byť komplikovaná. Inak si môžeme byť istí, že žiaci nebudú rozumieť problému ani riešeniu ani tomu, o čo nám nakoniec ide. My sme najskôr zvolili úlohu, ktorú zvládnu žiaci vyriešiť sami s doterajšími vedomosťami. Ak ale takéto riešenie vhodne komentujeme, vyvoláme v nich snahu, túžbu - teda motiváciu - riešenie zjednodušiť a sprehľadniť. Pri vysvetľovaní pritom využívame už existujúce skúsenosti žiakov so substitúciou (nejaký výraz označíme „písmenkom“) z matematiky a fyziky.

1. Začneme motiváciou - ukážkou, na ktorej žiaci uvidia zmysel použitia (lokálnej) premennej, napríklad:
   - „Do stredu grafickej plochy nakreslíme štvorček. Tvárime sa, že nepoznáme rozmery grafickej plochy. Preto musíme súradnice vypočítať z vlastností `Width` a `Height Image1`.“
   - Úlohu spoločne so žiakmi vyriešime:
     ```
     Image1.Canvas.Rectangle(
       Image1.Width div 2 - 10, Image1.Height div 2 - 10,
       Image1.Width div 2 + 10, Image1.Height div 2 + 10)
     ```
   - „Vidíme, že niektoré výrazy sa počítajú dvakrát, v matematike by sme ich označili písmenami `x` a `y`“ - tieto označenia nakreslíme.
   - „Riešenie, ktoré by sme zapísali pomocou `x` a `y`, by potom vyzeralo takto elegantne„ a výsledok napíšeme:
     ```
     Image1.Canvas.Rectangle(x-10, y-10, x+10, y+10)
     ```
   - Pripomenieme, že „v matematike sme `x` a `y` nazvali premenné“ a dokončíme, že „aj v programoch môžeme používať premenné.“
2. Je potrebné stručne ozrejmiť význam premenných v programovacom jazyku:
   - Čo je **premenná**: „Miesto v pamäti počítača.“
   - Ako funguje: „Do premennej **uložíme** nejakú hodnotu, napríklad vypočítame súradnicu stredu. Toto číslo si premenná **zapamätá**. Keď budeme súradnicu stredu potrebovať, pozrieme sa do premennej a zapamätané číslo prečítame, **zistíme**.“
   - Oplatí sa použiť prirovnania, napríklad: „Je to podobné, ako pamäť na kalkulačke `M`. Akurát, že v programe môžeme mať veľa takých pamätí“.
3. Ďalej sa nevyhneme **deklarácii** premennej:
   - Zásadne budeme používať lokálne premenné - tento pojem pred žiakmi zatiaľ nepoužívame.
   - „Ak chceme premenné používať, musíme najskôr takýto náš úmysel počítaču prezradiť. Robíme tak v zápise, ktorý nazývame **deklarácia**.“
   - Ukážeme, kde v programe píšeme `var X, Y: Integer;`
   - „Pri deklarácii vzniknú v pamäti počítača dve premenné `X` a `Y`“.
   - Zároveň aj kreslíme krabičky `X` a `Y` - **krabičkový model** je pri vysvetľovaní veľmi dôležitý aj užitočný.

4. Program dokončíme a popritom ho budeme trasovať:
   - Po priradení `X:=Image1.Width div 2` zapíšeme číslo do krabičky `X`.
   - Pri príkaze `Rectangle` naznačíme, ako prebehne výpočet parametrov.

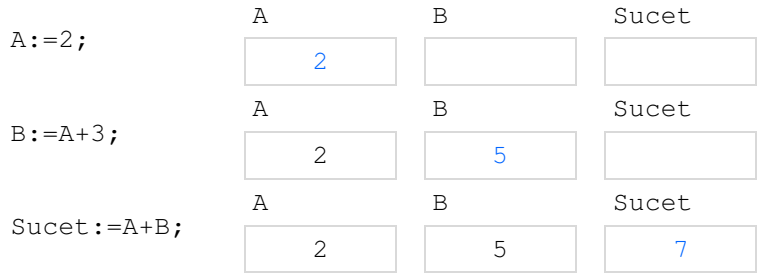
5. Ďalej odporúčame riešiť niekoľko úloh, v ktorých žiaci na rôznych situáciách spoznávajú premenné. Tieto úlohy budú mať experimentálny charakter a môžeme sa nad nimi zamýšľať pri tabuli:
   - Začneme deklaráciou: `var A, B, Sucet: Integer;`
   - Pýtame sa a kreslíme:
     - „Čo vznikne v pamäti počítača?“
     - „Ako sa budú premenné nazývať?“
     - „ Aké hodnoty si dokážu premenné zapamätať?“
   - Napíšeme príkazy (jednotlivými príkazmi sledujeme rôzne ciele):

     `A:=2;` ... jednoduché priradenie

     `B:=A+3;` ... použitie premennej a vyhodnotenie výrazu

     `Sucet:=A+B;` ... použitie premenných a vyhodnotenie výrazu

     `Image1.Canvas.TextOut(0, 0, IntToStr(Sucet));` ... **výpis** hodnoty premennej
   - Príkazy budeme **trasovať**, prípadne žiadať žiakov, aby nám s trasovaním pomohli. Je dôležité, aby sme do nakreslených krabičiek vpisovali údaje:



6. Riešime aj úlohy, v ktorých sa zadáva **číselný vstup** z editovacieho políčka, napríklad v učebnici [7] v kapitole 2.3.
7. Úlohy na trénovanie:
   - Podobné úlohy na trasovanie, ako v predchádzajúcom 5. bode.
   - Príklady v učebnici [7] v kapitolách 2.2, 2.4.
8. Neskôr učíme aj:
   - Lokálne a **globálne** premenné - dobrá motivácia na globálne premenné je úloha: „vytvorte program, ktorý počíta kliknutia na tlačidlo“.
   - Používanie premenných rôznych typov.
   - Pravidlá pre pomenovanie a deklarovanie premenných.

Treba si uvedomiť, že premenné v programoch majú niekoľko úloh:
- **uchovávajú** údaje (niekedy iba dočasne),
- sú nástrojom na **zovšeobecnenie** - umožňujú vytvárať všeobecnejšie riešenia problémov,
- sú nástrojom na **zefektívnenie** riešenia - rovnaký výpočet sa realizuje iba raz, výsledok sa potom viackrát použije.

K týmto cieľom by sme mali viesť aj žiakov a zostavovať úlohy, ktoré tieto cele sledujú a postupne naplnia.

Pozor:

- Neučíme premenné na ukážke s príkazom `random` - napríklad:
  ```
  a:=random(100);
  b:=a+10;
  ```
  Takýmto nevhodným príkladom skomplikujeme porozumenie premenným. Navyše príkazy budeme ťažko trasovať.
- Problém s výmenou obsahu dvoch premenných riešime neskôr. Rozhodne to nesmie byť prvá úloha, s ktorou sa žiaci, v súvislosti s premennými, stretnú.
- Prvé príklady s premennými, ktoré majú slúžiť žiakom na spoznanie premenných, nemôžeme kombinovať a sťažiť ešte aj tým, že riešime nejaký matematický problém: riešte rovnicu, sústavu rovníc, nájdite koreň funkcie...
- Aj napriek našej snahe môžu občas (niektorí) žiaci nesprávne porozumieť tomu, ako fungujú premenné alebo niektoré operácie, napríklad:
  - Premenné si pamätajú funkciu:

    `A:=1;` ... do premennej sa priradilo číslo

    `B:=A+2;` ... do premennej sa priradila funkcia `A+2`

    `A:=10;` ... preto, bude aj v premennej `B` číslo `12`
  - Ak zvolíme zlý model - napríklad nie krabičky, ale vrecúška s guľôčkami, môžu operáciu priradenia chápať ako presúvanie:

    `A:=10;` ... v premennej `A` bude `10`

    `B:=A;` ... v premennej `A` bude `0` alebo nič.

Zamyslite sa nad problémom výmeny obsahu dvoch premenných:
- Akú motiváciu by ste použili?
- Kedy má pre žiakov zmyslel riešiť takýto problém?
- Kedy by ste túto úlohu zaradili do vyučovania?
- Ako by ste pomohli žiakom pri riešení?

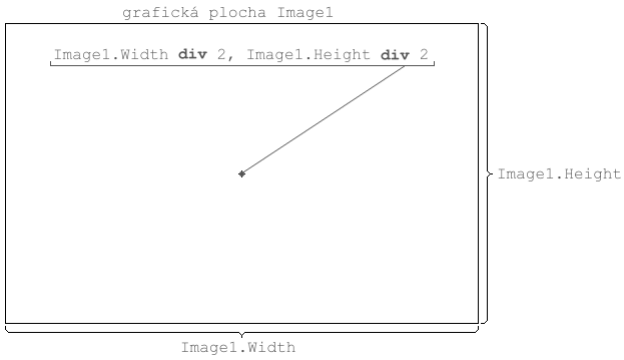
motivácia

Ukážka:

1. Videli ste už miesta označené ako „Tu je stred Európy“? Takých miest je na Slovensku niekoľko. Aj my chceme označiť stred grafickej plochy a to tak, aby bola značka iba jedná a bola umiestnená čo najpresnejšie. Značkou môže byť štvorček. Pozor ale, rozmery grafickej plochy nepoznáme a nechceme ich zisťovať ani pomocou Inšpektora Objektov.

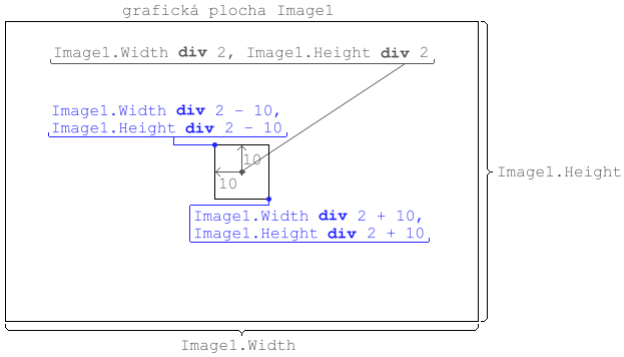
zbieranie skúseností

2. Zamyslime sa nad tým, ako vypočítame súradnice stredu grafickej plochy:



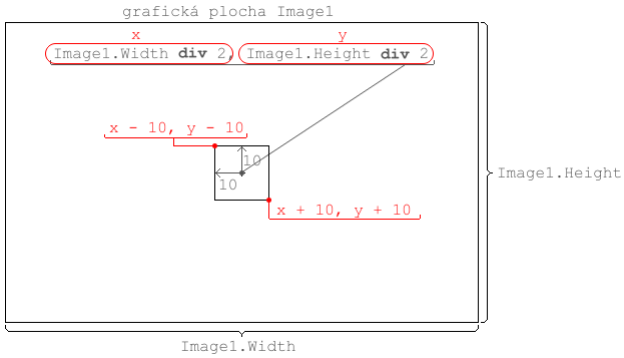
Rozmery grafickej plochy vieme zistiť tak, že sa pozrieme na vlastnosti `Image1.Width`, `Image1.Height`.

3. Súradnice stredu už poznáme. Poďme nakresliť štvorček, ktorý má dĺžku strany napríklad 20 bodov.



4. Vidíme, že v parametroch pre kreslenie štvorca sa viackrát počítajú výrazy:
   `Image1.Width div 2` a `Image1.Height div 2`.

5. V matematike by sme tieto výrazy označili písmenkami x a y:



Tak sa výpočet súradníc výrazne skráti.

6. Ako by potom vyzeral príkaz na kreslenie štvorca?

```
Image1.Canvas.Rectangle(x-10, y-10, x+10, y+10)
```

motivácia

7. V matematike sa x a y nazývajú premenné. Našťastie, premenné môžeme používať aj v našich programoch. Poďme sa teraz pozrieť na to, ako by sme to robí v jazyku Pascal...

4.7 Cyklus `for`

Cieľ:

- naučiť používať príkaz cyklu **`for`**,
- postupne učiť rozpoznávať opakujúce sa vzory v probléme, zovšeobecňovať a zapisovať riešenie pomocou cyklu.

Nové pojmy, poznatky a zručnosti:

- príkaz opakovanie, konštrukcia **`for ... to ... do`**,
- hranice cyklu, riadiaca premenná,
- telo cyklu, zložené telo cyklu pomocou **`begin`** a **`end`**

V jazyku C++ je cyklus `for` komplikovanejší, ako príkaz cyklu `while`. Preto by sme v jazyku C++ začínali s cyklom `while` a nie s cyklom `for`.

Poznámky:

- V jazyku Pascal začneme s konštrukciou cyklu **`for`** preto, lebo má relatívne jednoduchú syntax.
- Budeme sa snažiť rozpoznávať rôzne **úrovne náročnosti** problémov a úloh, ktoré pomocou cyklu riešime:

  I. V tele cyklu sa nepoužívajú premenné - (úlohy na úrovni príkazu `repeat` z Loga), napríklad:

```
for i:=1 to 10 do
  Image1.Canvas.LineTo(Random(200), Random(200));
```

  II. V tele cyklu sa používa riadiaca premenná, napríklad:

```
for i:=1 to 10 do
  Image1.Canvas.TextOut(0, i*20, IntToStr(i*i));
```

  III. V tele cyklu sa používa aj iná premenná, napríklad:

```
s:=0;
for i:=1 to 10 do s:=s+i;
```

  IV. Ďalšie úlohy, v ktorých sa vyskytujú rôzne kombinácie:
  - cyklus v cykle,
  - hranice cyklu sú konštantné alebo závisia od premenných,
  - náročný algoritmus (napr. úlohy zo súťaží).

- Treba si uvedomiť, že v úlohách s cyklami na úrovni II, III a IV musia žiaci, okrem aplikovania syntaktických pravidiel, vedieť aj **zovšeobecňovať**. A práve hľadanie opakujúceho sa vzoru a objavovanie všeobecného vzorca je dôležitá súčasť informatického myslenia a programovania. Tieto schopnosti zvykneme u žiakov rozvíjať tak, že často napíšeme niekoľko prvých príkazov s konkrétnymi hodnotami - napríklad:

```
Image.Canvas.TextOut(0, 1*20, IntToStr(1*1));
        - || -       (0, 2*20,  - || - (2*2));
        - || -       (0, 3*20,  - || - (3*3));
...
```

  Až teraz sa môžeme pokúsiť o diskusiu: „Ako bude vyzerať riešenie pre nejaké ďalšie číslo `i`?“

```
        - || -       (0, i*20,  - || - (i*i));
```

Takáto **klasifikácia** nám umožňuje pochopiť a rozmýšľať o tom, prečo sú (alebo môžu byť) rôzne úlohy pre žiakov rôzne náročné, zložité až neriešiteľné. Potom sa zamýšľame nad tým, ako úlohy prípadne aj extrémne zjednodušíme tak, aby medzi nimi nevznikali veľké skoky v náročnosti. Tento prístup sa snažíme aplikovať na všetky témy, ktoré učíme. Uvedená optika nazerania na vyučovanie nám pomáha vymýšľať kvalitné gradované série úloh tak, aby každý žiak dokázal príjemne hladko nazbierať dostatok vlastných skúseností. Výsledkom bude nielen to, že nám žiaci porozumejú, ale aj to, že získajú pozitívny vzťah k informatike a programovaniu. A domnievame sa, že práve toto je to najdôležitejšie, čo súčasná veda potrebuje.

Postup:

1. Začneme úlohou, keď treba veľa niečoho nakresliť. Napríklad: „Chceme nakresliť oblohu s veľkým množstvom hviezdičiek.“
   - Úlohu necháme riešiť žiakom s tým, aby najskôr kreslili jednu hviezdičku:

```
Image1.Canvas.TextOut(random(400), random(300), '*');
```

   - Pýtame sa: „Ako by sme nakreslili 2 hviezdičky?“
   - Žiaci by pravdepodobne napísali alebo skopírovali príkaz ešte raz.

Motivácia je založená na nešikovnom riešení, kým nepoužijeme cyklus.

- Požiadavku stupňujeme: „Ako by ste nakreslili 10 hviezdičiek?“, „Viete si predstaviť, ako by ste kreslili 100 alebo 1000 hviezdičiek?“
- Z predchádzajúcich aktivít by malo vyplynúť ponaučenie v zmysle:
  - „Pri kopírovaní sa nesmieme pomýliť. Sami musíme počítať, koľko príkazov `TextOut` už v programe máme.“
  - „Takéto riešenie je nešikovné. Napíšeme sa, a pritom sa veľakrát za sebou opakuje rovnaký príkaz. Takto sa programy nepíšu.“

S počtom hviezdičiek to nepreháňame, aby mohli program skontrolovať. Teda, či na obrazovke vidia práve 10 hviezdičiek

2. Predchádzajúci program prerobíme tak, aby fungoval pomocou cyklu:
   - „Veľa hviezdičiek nakreslíme pomocou cyklu oveľa jednoduchšie. Postupujeme takto.“
   - Deklarujeme premennú - jej význam vysvetľujeme iba tak, že „táto premenná bude slúžiť ako počítadlo - koľko hviezdičiek sa už nakreslilo“.
   - Napíšeme cyklus „**`for` `I:=1` `to` `10` `do`**“ a telo cyklu bude tvoriť iba príkaz `TextOut`, ktorý kreslí na náhodné miesto znak hviezdičky.

Žiakov treba upozorniť na pekné **formátovanie** programu. Ako učitelia musíme sami dodržiavať pravidlá formátovania a pekne zapisovať príkazy. A tiež musíme vyžadovať od žiakov, aby oni sami odsadzovali príkazy v blokoch `begin ... end`. Inak budú programy nečitateľné a ťažko sa v nich sa budú hľadať chyby.

3. Program **krokujeme** tak, aby žiaci videli, ako sa program vykonáva, ako funguje cyklus a ako sa opakovane vykonávajú príkazy z tela cyklu.

4. Objavujeme, ako funguje príkaz **`for`**.
   - So žiakmi experimentujeme a meníme **hranice cyklu**: `1` **`to`** `100`, `1` **`to`** `1000`, `0` **`to`** `10` alebo `100` **`to`** `0`.
   - Diskutujeme, koľko hviezdičiek sa kreslí v rôznych situáciách.
   - „Príkaz `Image1.Canvas.TextOut(...)` tvorí **telo cyklu**.“
   - Upozorníme na častú chybu - čo sa stane, ak za slovo `do` napíšeme `;`

5. Telo cyklu je iba jediný príkaz, avšak opakovať môžeme aj viac príkazov: „chcem, aby každá hviezdička mala svoju farbu“.
   - „Potrebujeme, aby sa opakovane vykonávali 2 príkazy:
     ```
     Image1.Canvas.Font.Color:=random(256*256*256);
     Image1.Canvas.TextOut(random(400), random(300), '*');
     ```
   - Tieto príkazy bude potrebné „uzavrieť“ medzi slová **`begin`** a **`end`**. Ich použitie niekedy vysvetľujeme ako „programové zátvorky“ alebo „zložený príkaz“.
   - Častou chybou žiakov je, že na tieto slová zabudnú a iba omylom odsadia príkazy - treba predviesť, ako zápisu rozumie počítač, čo sa vtedy stane.

Trasovacie tabuľky môžeme použiť nielen na to, aby žiaci videli, ako program pracuje. Môžeme ich použiť aj opačne, teda na to, aby žiaci zovšeobecnili alebo hľadali určité vzťahy. Namiesto tabuliek, zvykneme pri riešení grafických problémov často a **veľa kresliť** obrázky, na ktorých vidno vzťah medzi súradnicami a riadiacou premennou.

6. Preskúmame, ako funguje **riadiaca premenná**:
   - Necháme pod seba vypísať čísla od `1` po `10`
     ```
     for i:=1 to 10 do
       Image1.Canvas.TextOut(0, i*20, IntToStr(i))
     ```
   - Zostavíme „trasovaciu tabuľku, na ktorej vidno, ako sa mení premenná `i`, počíta súradnica `i*20` a vypisuje výsledok:

`i`	`i*20`	na obrazovke vidíme:
1	20	„1“
2	40	„2“
3	60	„3“

7. Úlohy na trénovanie a ďalšie námety:
   - Úlohy a témy z učebnice [7] v kapitole 2.5
   - Úlohy s číslami a s ciframi z učebnice [7] v kapitole 2.6.
   - Vnorený cyklus podľa učebnice [7] v kapitole 2.7.

Pozor: Tak, ako sme na začiatku písali v poznámke, pri učení cyklov postupujeme od úlohy typu I až k úlohám typu IV. Znamená to, že určite nezačíname príkladom typu „sčítajte prvky postupnosti...“

Ukážka:

motivácia

1. Poznáme hviezdnu oblohu? Chceme, aby nám počítač takú nakreslil. Na našej oblohe budú hviezdičky umiestnené náhodne.

zbieranie skúseností

2. Skúsme najskôr nakresliť jednu hviezdičku. Navrhnite spôsob, ako by ste ju nakreslili... Napríklad takto, príkazom `TextOut`:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(random(400), random(300), '*');
end;
```

3. Vyskúšajte, či program funguje.
4. Tak, a teraz skúsme nakresliť 5 hviezdičiek - príkaz označíme a pomocou schránky skopírujeme:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Image1.Canvas.TextOut(random(400), random(300), '*');
  Image1.Canvas.TextOut(random(400), random(300), '*');
  Image1.Canvas.TextOut(random(400), random(300), '*');
  Image1.Canvas.TextOut(random(400), random(300), '*');
  Image1.Canvas.TextOut(random(400), random(300), '*');
end;
```

5. Program vyskúšajte.
6. Ako by sme program upravili, aby sa kreslilo 10 hviezdičiek? ... Nakopírovali by sme všetkých 5 príkazov ešte raz.

motivácia

7. V niektorých počítačových hrách býva 100 alebo 1000 hviezdičiek. Vieme si predstaviť, ako by vyzeral taký program, keby sme príkazy kopírovali? Poriadne by sme sa natrápili. A to by sme sa nesmeli pri kopírovaní pomýliť, aby príkazov nebolo ani menej, ani viac... Týmto spôsobom sa programy nerobia.

zbieranie skúseností

8. V skutočnosti potrebujeme, aby sa príkaz `TextOut` pre kreslenie hviezdičky vykonal viac krát. To zabezpečíme pomocou iného špeciálneho príkazu **`for`** takto:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
begin
  for I:=1 to 10 do
    Image1.Canvas.TextOut(random(400), random(300), '*');
end;
```

*Deklarovali sme premennú `i`. Je úlohou bude, aby fungovala ako počítadlo. Vidíme, v príkaze **`for`** používame premennú `i`. Príkaz **`for`** počíta od 1 po 10. Neskôr si presne povieme, ako tento príkaz pracuje.*

9. Program vyskúšajte.
10. Skúsme si uhádnuť, čo treba v našom programe zmeniť, aby sa namiesto 10 hviezdičiek kreslilo 50 hviezdičiek. Vyskúšajte.
11. A ako nakreslíme 1000 hviezdičiek?
12. Príkaz **`for`** nazývame príkazom **cyklu**. Je to preto, lebo tento príkaz dokáže opakovane vykonávať iné príkazy - v našom prípade kreslenie hviezdičky.
13. Program upravte tak, aby sa:
    - hviezdičky kreslili na modré pozadie (napríklad `clNavy`),
    - aby sa hviezdičky kreslili bielou farbou (napríklad `clWhite`),
    - aby sa nepremazávali (`Image1.Canvas.Brush.Style:=bsClear`).

    Kam treba tieto príkazy napísať?

Zhrnutie

Mnohé témy z programovania sú veľmi rozsiahle a môžu nás veľmi ľahko zviesť k zlému vyučovaniu. V ukážkach, ktoré sme uvádzali pri jednotlivým témam, sme približne videli tempo, akým pri vyučovaní postupujeme. Je dôležité, aby sme postupovali po **malých krokoch**.

Úvodné témy sú náročne ani nie tak z algoritmického pohľadu, ako kvôli novému prostrediu, ktoré obsahuje veľa rôznych prvkov a technických detailov. Preto je dôležité, aby sme sa vždy snažili **minimalizovať** množstvo nevyhnutných detailov, technických informácií alebo konceptov, ktoré potrebujeme žiakom predkladať. My sme si napríklad zvolili iba 2 komponenty, ktoré nám umožnia elegantne realizovať veľké množstvo zaujímavých úloh. Ostatné komponenty si zatiaľ akoby nevšímame, pretože ich nepotrebujeme. Nevysvetľujeme ani základy objektovo orientovaného programovania, pretože tie momentálne nie sú našim cieľom.

Téma premenné a cyklus (a neskôr uvidíme, že aj niektoré ďalšie témy v programovaní) majú akoby dve roviny:

- Technická rovina spočíva v tom, že učíme žiakov syntax a to, ako premenné alebo cyklu fungujú.
- Abstraktná rovina je daná tým, že premenné aj cyklus prinášajú do programovania určitú mieru zovšeobecňovania a značnú dávku abstrakcie.

Znamená to napríklad, že žiakov učíme nie len syntaktické pravidlá a to, ako rôzne programové konštrukcie fungujú, ale aj to, ako majú hľadať a objavovať všeobecnejšie vzťahy alebo opakujúce sa vzory. Tieto schopnosti sú dôležité preto, lebo od nich závisí to, aké typy problémov dokážu v budúcnosti žiaci samostatne riešiť alebo to, či budú schopní čítať a rozumieť cudzím programom, ideám.

Naše skúsenosti ukazujú, že spomínané schopnosti sa nedajú naučiť rýchlo – „zo dňa na deň“. Naopak, tento proces je postupný a trvá dlho. Od nás si to vyžaduje, aby sme so žiakmi spočiatku riešili **malé**, **krátke** a **jednoduché** problémy alebo úlohy, na ktorých žiaci spoznávajú dôležité princípy (ako fungujú premenné v programoch, ako objavujeme rovnaké výrazy alebo, čo treba v programe opakovane vykonávať). Tieto schopnosti potom pozvoľna trénujeme na primeranom množstve mierne zložitejších úloh.

Riešenia a návody

Aktivita 1.2:

- Riešenie nekomplikujte zbytočnosťami ani programátorskými alebo vizuálnymi skrášleniami, ktoré úloha vyslovene nevyžaduje. Minimalizujte aj počet príkazov, či dĺžku zápisov napríklad vhodným použitím premenných.
- Premyslite si, čo pri tabuli poviete. Pred vystúpením si vytvorte krátky scenár.
- Pri vysvetľovaní hlavnej myšlienky postupujte od konkrétneho ku všeobecnému, tak ako sme sa učili o poznávacom procese (aby bolo vidieť, že všeobecné zápisy nespadli „z neba)
- Preto kreslite obrázky, naznačujte konkrétne súradnice, pozície. Znázorňujte premenné a ich obsah. Ukazujte, ako sa program vykonáva.

Aktivita 1.4:
Úloha je náročná kvôli požitiu funkcií sin a cos. Znamená to, že úloha nie je ani tak náročná z programátorskej, ako je náročná z matematickej stránky. Máme odpozorované, že väčšina ľudí má predstavu o funkciách sin a cos na úrovni „sin je taká vlnka“ alebo „sin je protiľahlá ku prepone“. Veľmi málo ľudí chápe tieto goniometrické funkcie vo vzťahu ku kružnici. Pritom definícia funkcií sin a cos sa opiera o jednotkovú kružnicu. Preto máme odpozorované, že žiaci mávajú obrovské problémy riešiť aj úlohy typu „znázornite vrcholy pravidelného 10 uholníka“.

Aktivita 1.5:
Nakreslite slniečko bez lúčov (teda, iba ako kruh). Nakreslite slniečok s lúčmi ako úsečkami s náhodným koncom. Nakreslite slniečko iba s horizontálnymi vertikálnymi a diagonálnymi lúčmi.

Aktivita 1.6:

- V úlohe 1: Syntax cyklu a kreslenia bodiek.
- V úlohe 2: Objavenie algoritmu pre postupného sčítavania zŕn.
- V úlohe 3: Výpočet súradníc.
- V úlohe 4: Cyklus v cykle.
- V úlohe 5: Použitie funkcií `sin` a `cos` pre výpočet súradníc lúčov.

Aktivita 1.7:
Premenná, cyklus **`for`**. Príkazy pre kreslenie bodov, obdĺžnikov, elíps, úsečiek, nastavenie farieb. Konverzia textu na číslo a naopak, sínus a kosínus, náhodné čísla.

Čo sme sa naučili v tomto module

Zhrnutie

Spoznali a analyzovali sme súčasné učebnice programovania a diskutovali sme o tom, nakoľko sú vhodné pre vyučovanie.

Učili sme sa analyzovať jednotlivé témy z programovania, vytvárať didakticky správnu postupnosť motivácií pre každú tému z programovania. Navrhovali sme, prezentovali a analyzovali ukážky vyučovacích hodím pre prvé – úvodné témy do programovania.

Preverenie výstupných vedomostí

Zvoľte si iný programovací jazyk, o ktorom sme v tomto dokumente nepísali a analyzujte ho. Kedy a prečo vznikol? Aké vlastnosti programovacieho jazyka by ocenili profesionálni programátori. A aké vlastnosti by ocenili učitelia a žiaci v škole?

Literatúra a použité zdroje

[1] Salanci, Ľ. (2010) a kol: Didaktika programovania. Bratislava: ŠPÚ. ISBN 978–80–8118–065–1
[2] Salanci, Ľ. (2010) a kol: Didaktika programovania 1. Bratislava: ŠPÚ. ISBN 978-80-8118-037-8
[3] Salanci, Ľ. (2010) a kol: Didaktika programovania 2. Bratislava: ŠPÚ. ISBN 978-80-8118-053-8
[4] Blaho, A., Kalaš, I.: Tvorivá informatika. 1. zošit z programovania. Bratislava : SPN – Mladé letá, 2007. ISBN 80-10-01223-7
[5] Varga, M., Blaho, A., Zimanová, R.: Algoritmy s Logom. Bratislava: SPN, 1999. ISBN 80-08-02965-X.
[6] Zimanová, R., Belušová, M., Varga, M.: Algoritmy s Pascalom. Bratislava: SPN, 2002. ISBN 80-08-03289-8.
[7] Blaho, A. (2006) Informatika pre stredné školy. Programovanie v Delphi. Bratislava: SPN. ISBN 80-10-00421-9
[8] Bezáková D., Lovászová G., Kučera, P.: Programovanie 1. Bratislava: ŠPÚ. ISBN 978-80-89225-65-1
[9] Bezáková D., Kučera, P., Lovászová G., Tomcsányi, P.: Programovanie 4 (Logo). Bratislava: ŠPÚ. ISBN 978-80-8118-017-0
[10] Blaho, A., Kubincová, Z., Salanci, Ľ. (2009) DVUI: Programovanie 2. Bratislava: ŠPÚ. ISBN 978-80-8118-007-1
[11] Blaho, A., Kubincová, Z., Salanci, Ľ. (2009) DVUI: Programovanie 3. Bratislava: ŠPÚ. ISBN 978-80-8118-014-9
[12] Blaho, A., Kubincová, Z., Salanci, Ľ. (2009) DVUI: Programovanie 4 (Pascal). Bratislava: ŠPÚ. ISBN 978-80-8118-018-7
[13] Hejný, M. a kol: Teória vyučovania matematiky. Bratislava: SPN 1990. ISBN 80-08-013443-3
[14] Hejny, M.: Understanding and structure, Proc. of CERME 3. Bellaria 2003.
[15] Blaho, A., Salanci, Ľ.(2009) DVUI: *Programovanie 1 - 9 pre 2CS*. Bratislava: ŠPÚ 2009.

Tento študijný materiál vznikol ako súčasť národného projektu Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika v rámci Aktivity „Vzdelávanie nekvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ“.

Autori © RNDr. Ľubomír Salanci, PhD.
PaedDr. Monika Tomcsányiová, PhD.
RNDr. Andrej Blaho, PhD.

Názov Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Podnázov Didaktika programovania pre stredné školy 1

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti PaedDr. Ján Beňačka, PhD.
RNDr. Peter Varša, PhD.

Počet strán 36

Náklad 300 ks

Prvé vydanie, Bratislava 2011

Vydal Štátny pedagogický ústav, Pluhová 8, 830 00 Bratislava, v súčinnosti s Univerzitou Pavla Jozefa Šafárika v Košiciach, Univerzitou Komenského v Bratislave, Univerzitou Konštantína Filozofa v Nitre, Univerzitou Mateja Bela v Banskej Bystrici a Žilinskou univerzitou v Žiline

Vytlačil BRATIA SABOVCI, s r.o., Zvolen

ISBN 978-80-8118-079-8