Didaktika programovania pre SŠ 2

Garant predmetu:

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

Autori:

RNDr. Ľubomír Salanci, PhD.
KZVI FMFI UK, Bratislava

PaedDr. Monika Tomcsányiová, PhD.
KZVI FMFI UK, Bratislava

RNDr. Andrej Blaho, PhD.
KZVI 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 stredné školy

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 odborné knihy a web stránky, ktoré sú zamerané na programovanie. ďalej budeme analyzovať a diskutovať o vyučovaní ďalších tém z programovania, akými sú: podmienený príkaz, procedúry a funkcie, myš, polia. Pre tieto témy navrhneme vzorové vyučovacie hodiny a budeme ich analyzovať z pohľadu teórie poznávacieho procesu. V závere zostavíme a analyzujeme testy z programovania a navrhneme ich hodnotenie.

Obsah

Didaktika programovania pre SŠ 2 1
Identifikácia modulu 1
Zaradenie modulu 1
Abstrakt modulu 1
Obsah 2
Úvod 3
Cieľ modulu 3
Vstupné vedomosti 3
Požadované prerekvizity 3
Predpokladané vstupné vedomosti, skúsenosti a zručnosti 3
Kapitola 1: Rozcvička 4
Kapitola 2: Knihy, literatúra a web o programovaní 6
Odborné knihy 6
Webové stránky 7
Zhrnutie 8
Kapitola 3: Programovanie v jazyku Pascal 2 9
3.1 Podmienený cyklus `while`, jednoduché podmienky 10
3.2 Podmienený príkaz `if` 12
3.3 Procedúry 15
3.4 Funkcie 18
3.5 Myš 20
3.6 Polia 23
3.5 Súbory 26
Kapitola 4: Hodnotenie a klasifikácia 30
Oprava a hodnotenie 30
Tvorba úloh 36
Záver 37
Návody a riešenia 38
Čo sme sa naučili v tomto module 39
Preverenie výstupných vedomostí 39
Literatúra a použité zdroje 39

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 programovaním a riešením informatických problémov. Preto chceme objavovať zákonitosti alebo pravidlá, ktoré by prispeli k lepšiemu vyučovaniu informatiky.

Pre tento modul je nevyhnutné mať k dispozícii nasledujúce softvérové prostredia: Delphi alebo Lazarus. 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.
- Analyzoval webové stránky a odborné knihy o programovaní a spoznal rozdiel medzi nimi a učebnicami.
- Učil sa aplikovať teóriu poznávacieho procesu vo vyučovaní programovania.
- Spoznal metodiku vyučovania ďalších tém z programovania na SŠ.
- Naučil sa navrhovať a zostavovať vyučovacie hodiny z programovania.
- Zamýšľal sa nad korektným hodnotením a klasifikovaním svojich žiakov.

Vstupné vedomosti

Požadované prerekvizity

Účastník vzdelávania musí mať absolvované všetky nasledujúce moduly:

- Programovanie 1 až 9,
- Didaktika programovania,
- Didaktika programovania pre stredné školy 1.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

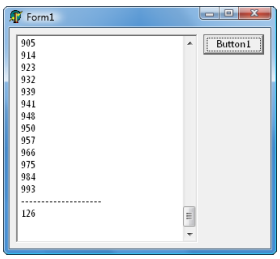
Pozná aspoň tieto programovacie jazyky: Pascal (Delphi / Lazarus), Logo (Imagine). Dokáže riešiť základné algoritmické problémy. Riešenie vie zapísať a odladiť v niektorom z programovacích prostredí. Vie čítať cudzie programy, dokáže im porozumieť, analyzovať ich.

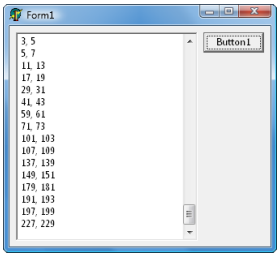
Pozná históriu vyučovania informatiky a programovania. Pozná súčasnú úlohu programovania v rámci informatiky na základnej a strednej škole. Je oboznámený s teóriou poznávacieho procesu vo vyučovaní programovania.

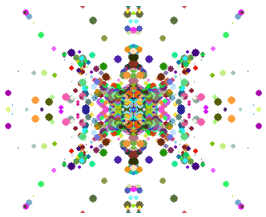
Pozná metodiku vyučovania prvých tém z programovania v jazyku Pascal, akými sú úvod do programovania, grafika, premenné, cyklus. Dokáže k nim navrhnúť vyučovaciu hodinu. Zvládne analyzovať iné podobné návrhy z pohľadu vyučovania.

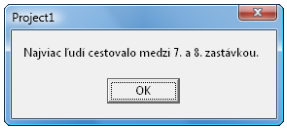
Kapitola 1: Rozcvička

Najskôr vyriešime niekoľko malých programátorských úloh. Okrem toho, že si pripomenieme programovanie v jazyku Pascal, budeme neskôr tieto úlohy, ako aj ich riešenia, analyzovať z pohľadu didaktiky programovania.









Aktivita 1.1

Diskusia

Rozdeľte sa do skupín. Každá skupina vyrieši jednu z nasledujúcich úloh:

1. Úloha: Poverčivému kráľovi sa prisnilo, že šťastie mu prinesú iba magické čísla. Preto si zavolal svojho dvorného počtára, aby mu také čísla našiel. Vraj to musia byť trojciferné čísla, ktorých súčet cifier je deliteľný 7. Aké čísla to boli? A koľko ich bolo? Pomôžte dvornému počtárovi a napíšte program, ktorý to zistí.

2. Úloha: Viete ako vyzerá pečiatka, ktorá označuje utajené dokumenty? Aj my chceme vytvoriť tajnú aplikáciu a označiť ju podobnými pečiatkami tak, že štyri pečiatky budú zobrazené v rohoch a jedna pečiatka v strede grafickej plochy. Pečiatka by mohla mať tvar červeného obdĺžnika, v ktorom by bolo napísané „Prísne tajné!“.

3. Úloha: Prvočíselné dvojčatá sú také dve prvočísla, medzi ktorými je rozdiel 2. Napríklad (3, 5), (5, 7), (11, 13), ... Sme zvedaví, koľko existuje prvočíselných dvojčiat medzi číslami od 1 po 1000. Vytvorte program, ktorý ich vypíše a nakoniec zobrazí aj ich počet.

4. Úloha: Pozerali ste sa niekedy do farebného kaleidoskopu? Vytvorte program, ktorý nám umožní kresliť pomocou myši podobné symetrické a farebné obrázky. Pri pohybe myšiu sa nakreslí pod kurzorom náhodne zafarbená bodky. A ďalšie 3 bodky sa nakreslia symetricky podľa vodorovnej a zvislej osi, ktoré prechádzajú stredom grafickej plochy.

5. Úloha: Autobus jazdí medzi dedinami Hornou a Dolnou. Na tejto trase je 10 zastávok. Na každej z nich nastupovalo a vystupovalo niekoľko ľudí. Počet ľudí, ktorý pribudol alebo ubudol v autobuse na jednotlivých zastávkach je uložený v 10 prvkovom poli `A`. Zistite, medzi ktorými zastávkami bolo v autobuse najviac ľudí. Napríklad, pre pole s prvkami

```
5, -2, 8, 13, -15, 7, 9, -12, 3, -16
```

program vypíše, že „`Najviac ľudí cestovalo medzi 7. a 8. zastávkou`“.

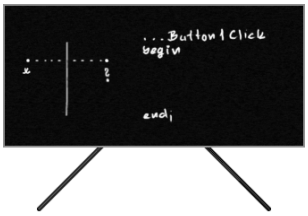
Pokyny k riešeniu a dobré rady:

- V skupine môžete spolupracovať.
- Úlohy môžete riešiť na papier alebo pomocou počítača.
- Všímajte si problémy, ktoré sami máte pri riešení úlohy.

Počítajte s tým, že nakoniec budete svoje riešenie prezentovať ostatným kolegom.

Aj žiaci zvyknú riešiť podobné úlohy. Stáva sa však, že niektorým žiakom sa nedarí. Ako by sme im pomohli? Vedeli by sme vysvetlili ideu riešenia žiakom alebo svojim kolegom?

Aktivita 1.2	Vysvetlite ideu riešenia svojom kolegom. Každá skupina si zvolí zástupcu, ktorý vystúpi pred ostatnými kolegami. Pozor: • Pri vysvetľovaní **nepoužívajte počítač ani projektor**. • Ak potrebujete kresliť, kreslite na tabuľu. • Nezapisujte celý program, zamerajte sa na jadro problému. • Vysvetlite, ako ste prišli na jednotlivé príkazy, konštrukcie a všeobecné výrazy (resp. ako by mali žiaci postupovať pri ich objavovaní). • Ideálne vystúpenie by malo trvať najviac 5 minút Jednotlivé vystúpenia pozorujte a robte si k nim poznámky.



Po každom vystúpení sa ho pokúsime analyzovať. 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?

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 1.4	Zoraďte úlohy podľa náročnosti. Povedzte, čo je na každej úlohe najťažšie.
Aktivita 1.5	Vedeli by ste ťažké úlohy zjednodušiť? Povedzte, ako.

Každú z úloh by sme mohli zaradiť ako úlohu na trénovanie poznatkov k určitej téme z programovania.

Aktivita 1.6	Povedzte: • Ku ktorej téme z programovania sa jednotlivé úlohy najviac hodia? • Aké vedomosti musia mať žiaci na to, aby zvládli jednotlivé úlohy vyriešiť samostatne?

Kapitola 2: Knihy, literatúra a web o programovaní

Štýl, akým sú písané odborné knihy alebo web stránky o programovaní, je v podstatnej miere odlišný od štýlu, akým sú písané učebnice pre školy.

Aktivita 2.1

Každý tím dostane odbornú knihu alebo odkaz webovú stránku.

Preštudujte ju a pokúste sa zodpovedať na nasledujúce otázky:

- O akom programovacom jazyku je?
- Aké prostredie používajú autori pri vysvetľovaní?
- Aké témy a v akom poradí sa preberajú? Stručne ich vymenujte.
- Aký je váš dojem?
- Viete si predstaviť, ako by ste knihu alebo webovú stránku použili vo svojej praxi? Povedzte.

Pri hodnotení si všímajte aj didaktickú stránku:

- Či sa pri vysvetľovaní nových pojmov používajú známe, už vysvetlené a zadefinované pojmy.
- Či je vysvetlenie primerane jednoduché, názorné, sprevádzané obrázkami alebo ilustráciami.
- Či autor používa motivácie, jednoduché ukážkové príklady, úlohy na precvičenie.
- Či má čitateľ, pre ktorého je kniha alebo webová stránka určená, šancu porozumieť textu.

Odborné knihy

Väčšina odbornej literatúry má nasledovnú štruktúru:

- Hovoria o typoch, syntaxi a konštrukciách programovacieho jazyka.
- Zameriavajú sa na vysvetľovanie komponentov, knižníc a funkcií.
- Niekedy začínajú jednoduchým príkladom (napríklad, zobrazenie správy „Hello Word!“), pokračujú vysvetľovaním komponentov, programovacích techník vo Windows alebo sa venujú programovaniu databáz.
- Ak autor uvádza príklady, často sa jedná o textovo alebo matematicky orientované ukážky.

V knihách nájdeme mnohostranové pasáže o typoch a o syntaktických pravidlách. Takto však nemôžeme postupovať pri vyučovaní v triede, pretože by sme žiakov unudili a zahltili encyklopedickými faktami.

Vidíme, že pri písaní odborných kníh si autor môže dovoliť iné postupy, aké používame v triede pri vyučovaní. Je to preto, lebo čitateľ knihy sa môže k prečítanému textu vrátiť alebo môže niektoré kapitoly preskočiť.

Odborná literatúra je výhodná pre skúseného programátora. V kníhkupectvách sa však stretneme aj s titulmi, ktoré majú pre začiatočníka na prvý pohľad sympatický názov: „Učíme sa programovať v...“ alebo „Programujeme v jazyku ... za 10 dní“ a podobné.

V niektorých prípadoch autori utvrdzujú čitateľa v tom, že si kúpil správnu knihu, pretože, kniha je určená aj pre začiatočníka – aj pre skúseného čitateľa. Naše skúsenosti sú iné. Z kníh, ktoré sme analyzovali, sme prišli k záveru, že takú knihu by sme začiatočníkovi neodporučili.

Webové stránky

Veľa informácií o programovaní, programovacích jazykoch aj algoritmoch nájdeme na webových stránkach.

Aktivita 2.2	Aké webové stránky o programovaní radi čítate?
Aktivita 2.3	Akým spôsobom používajú web stránky vaši žiaci na hodinách programovania? Aký mate na to názor?

Web stránky o programovaní sú výborným zdrojom informácií. Pri ich využívaní vo vyučovaní programovania však musíme byť veľmi opatrní. Je to preto, lebo z pohľadu vyučovania programovania platia aj o webových stránkach platia podobné skúsenosti, ako o odborných knihách. Najčastejšie pozorujeme nasledovné problémy:

- Stránky sú zamerané na použitie tej-ktorej technológie alebo na syntax programovacieho jazyka.
- Na stránkach často nájdeme prvý príklad „Hello world“, potom však v rýchlom slede uvidíme zoznam typov, zoznam programových konštrukcií, zoznam komponentov, zoznam knižníc a ich funkcií.
- Nie zriedka nájdeme stránky, ktoré do detailov vysvetľujú jednoduché veci – napríklad obsahujú ilustrovaný popis prostredia alebo postupu pri uložení projektu. Avšak náročné a ťažké programátorské koncepty sú vysvetlené nedostatočne, napríklad „čo je“ a „ako funguje“ premenná, je vysvetlené iba jednou vetou.
- Obsahujú príklady, v ktorých sa využívajú rôzne programátorské triky – také, ktoré začiatočníka skôr zmätú alebo pomýlia.

Vidíme, že takéto webové stránky obsahujú veľa technických detailov a siahodlhé encyklopedické zoznamy s informáciami, ktoré sú niekedy pre skúseného programátora veľmi cenné, ale pre začiatočníka úplne zbytočné.

Pri vyučovaní programovania totiž sledujeme iné ciele – rozvoj algoritmického myslenia, spoznávanie programátorských konceptov a objavovanie všeobecných vzťahov. Mnohé webové stránky však takéto ciele vôbec nesledujú. Ich cieľom je poskytnúť čo možno najviac informácií.

Ak si navyše uvedomíme, že ich tvorcovia nemajú ani žiadne pedagogické skúsenosti, malo by nám byť jasné, prečo sú vynechané alebo poprehadzované etapy poznávacieho procesu. Pre nás to znamená, že webové stránky nemôžeme chápať ako vhodný zdroj metodických postupov. Na reálnej vyučovacej hodine, sa podľa takýchto webových stránok dá veľmi ťažko učiť.

Naopak, na webových stránkach môžeme nájsť veľa zaujímavých námetov. Pritom však musíme byť veľmi obozretní, kritickí a musíme veľmi dobre zvažovať, ako ich zakomponujeme do vyučovania.

Zhrnutie

Písaná literatúra vyzerá vo všeobecnosti inak, ako zvykneme postupovať na vyučovacej hodine. Videli sme, že učebnice aj tematické zošity majú odlišný charakter od odbornej literatúry a webových stránok. Je to preto, lebo autori odborných kníh a webových stránok si stanovili iné ciele ako autori učebníc.

Môžeme si všimnúť, že knihy a webové stránky majú často encyklopedický charakter:

- Najmä v knihách sa autor drží matematickej precíznosti – výklad je často založený na definíciách, ktoré sú niekedy doplnené ukážkovým príkladom.
- Postupnosť tém: definícia jazyka a syntaxe, zoznam všetkých príkazov jazyka, zoznam štandardných knižníc a ich funkcií.
- Riešia sa známe a všeobecne platné algoritmy.

A hoci viaceré z nich sľubujú, že sú vhodné aj pre začiatočníka, v konečnom dôsledku, žiadna z nich pre začiatočníka nie je.

Správne učebnice kladú dôraz na to, aby žiaci zbierali skúsenosti:

- Nemajú encyklopedický charakter. Obsahujú iba nevyhnutne malé a potrebné množstvo teórie. Ponúkajú dostatočné množstve gradovaných úloh na precvičenie.
- Postupnosť tém začína zoznámením sa s prostredím. Postupne sa vysvetľujú základne programátorské (algoritmické) koncepty, učia sa riešiť (elementárne) algoritmické problémy, ale zároveň sa popri tom nové poznatky precvičujú.
- Autori sa pri vysvetľovaní nikam neponáhľajú a nechávajú žiakom dostatok času na spoznanie konceptov. Dôraz sa kladie na motivujúce a zábavné príklady, ilustrácie, trasovanie, postupné objavovanie zákonitostí.

Kapitola 3: Programovanie v jazyku Pascal 2

V tejto rozsiahlej kapitole sa budeme zamýšľať nad vyučovacími hodinami z programovania na strednej škole. Pokúsime sa spoločne vymyslieť a ukázať, ako by sme mali učiť nasledujúce témy:

- podmienený cyklus `while`, jednoduché podmienky,
- podmienený príkaz `if`,
- procedúry,
- funkcie,
- myš,
- polia.

Každá z tém je značne rozsiahla. Preto v skutočnosti v škole venujeme každej téme viacero vyučovacích hodín.

Aktivita 4.1

Vypracujte vzorovú vyučovaciu hodinu pre každú z uvedených tém.

Pracujte v tímoch. Každý tím si zvolí jednu tému:

- V tíme spracujte tému tak, ako by sa mala učiť na jednej 45 minútovej hodine.
- Pred ostatnými kolegami sa treba v pozitívnom svetle predviesť, ako viete rešpektovať etapy poznávacieho procesu.
- Keďže každá z tém je veľmi široká. Preto sa zamerajte iba na jej úvodnú časť.
- Výsledok práce prezentuje zástupca tímu pri tabuli.

Prezentácia bude mať dve časti.

Prvá časť vystúpenia bude rozprávanie o téme, teda bude predstavovať stručnú **analýzu témy**. Povedzte:

- zoznam predpokladov – čo už žiaci vedia,
- cieľ – čo chcete naučiť,
- zoznam pojmov a poznatkov, ktoré idete žiakov učiť.

Druhá časť bude zrýchlená **ukážka** toho, ako by mala prebiehať vyučovacia hodiny. Pri vystúpení treba predviesť ako by ste postupovali pri vyučovaní v triede:

- ako by ste žiakov uviedli do problému a motivovali,
- ako by ste spolu so žiakmi objavovali riešenie, prípadne, ako by ste vysvetlili nový koncept,
- čo budú riešiť žiaci samostatne.

Prvú časť odporúčame prezentovať pomocou projektora. Druhú časť, teda ukážku vyučovacej hodiny, prezentujte **bez projektora** – používajte iba kriedu a tabuľu.

Námety môžete čerpať z učebnice a materiálov DVUI. Ostatní kolegovia na konci zhodnotia vystúpenie.

Pripomeňme si etapy poznávacieho procesu:

Motivácia
↓
Zbieranie skúseností
↓
Zovšeobecňovanie, abstrakcia
↓
Poznatok
↓
Trénovanie poznatku

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 sme analyzovali v nasledujúcich kapitolách. Okrem stručného postupu uvádzame ku každej téme aj kúsok ukážky – záznamu z vyučovacej hodiny.

3.1 Podmienený cyklus `while`, jednoduché podmienky

Cieľ:

- naučiť používať príkaz cyklu **`while`**,
- zostavovať jednoduché podmienky, ktoré budú založené na porovnaní,
- porozumieť podmienke vykonávania cyklu.

Nové pojmy, poznatky a zručnosti:

- cyklus **`while`**, syntax príkazu **`while ... do`**,
- podmienka vykonávania cyklu.

Poznámky:

- Budeme používať iba jednoduché podmienky založené na porovnaní hodnôt.
- Treba si uvedomiť, že cyklus **`while`** používame v situáciách, keď nedokážeme dopredu stanoviť počet opakovaní.
- Ak na riešenie stačí použiť cyklus **`for`**, nemá zmysel nútiť žiakov použiť cyklus **`while`**.
- Poradie, v akom učíme cyklus **`while`** a podmienený príkaz **`if`** nie je striktne dané a môžeme ho vzájomne vymeniť.

Postup:

Táto úloha sa dá riešiť aj pomocou cyklu `for`. Treba si však dávať pozor na vzorec, ktorý počíta hranice cyklu.

Čo je zlé na vzorci: `Image1.Width div 50`, ktorým by sme počítali počet opakovaní?

Keby vám žiaci tvrdili, že v ich programe predsa taký vzorec funguje – ako by ste im ich tvrdenie vyvrátili?

1. Začneme riešením problému, ktorý sa pomocou cyklu **`for`** rieši nepríjemne, ale pomocou cyklu **`while`** bude riešenie jednoduché. Napríklad: „Nakreslite vedľa seba toľko vagónikov, koľko sa zmestí na šírku grafickej plochy. Vagónik je náhodne zafarbený obdĺžnik, ktorý má šírku 40 a výšku 20 pixlov. Medzi vagónmi je medzera 10 pixlov.“
2. Kreslíme na tabuľu:
   - naznačíme grafickú plochu,
   - nakreslíme vedľa seba niekoľko vagónov,
   - naznačíme ich polohu, rozmery a rozstupy,
   - polohu vagóna, ktorý budeme kresliť, označíme písmenom `X`.
3. Spolu so žiakmi budeme rozmýšľať nad tým:
   - Dokedy treba vagóniky kresliť? „Kým platí, že...“
   - Aké kritérium musí platiť pre posledný nakreslený vagón?
   - Ako by sme také kritérium zapísali v matematike?
4. Nakoniec by sme mali dospieť k neformálnemu slovnému zápisu:

   kým je `x+40 ≤ Image1.Width` *vykonávame príkazy*
   kreslíme vagón
   `x:=x+40+10;`

Pri trasovaní volíme takú kombináciu hodnôt, aby sme sa netrápili s písaním. Pri tom by zároveň mali žiaci vidieť, ako sa mení aspoň zopár krokov z opakovania tela cyklu.

5. Keďže sme v slovnom popise použili šikovné slovenské slová, ľahko ukážeme, ako spomínaný algoritmus zapíšeme pomocou konštrukcie **`while ... do`**. Program vyskúšame.
6. Odporúčame zostaviť trasovaciu tabuľku s konkrétnou hodnotou pre `Image1.Width` (napríklad, rovnou číslu `170`). V tabuľke by malo byť vidno, vedľa seba dvojicu: ako sa mení hodnota `x` a či je podmienka vykonávania cyklu `x+40 ≤ 170` splnená.
7. Ďalšie námety, príklady a úlohy na trénovanie nájdeme v učebnici [7] v kapitole 3.3

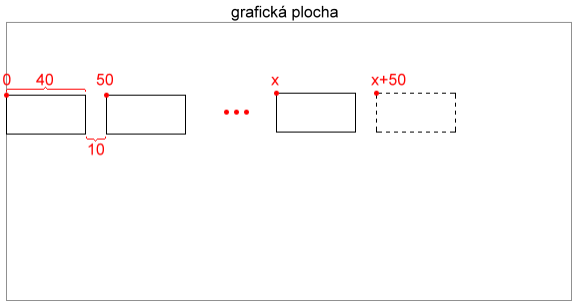
Pozor: Nezačíname cyklom so zloženými podmienkami. Nezačíname ukážkovým príkladom, ktorý rieši matematický problém. Typickou nevhodnou úlohou je hľadanie najmenšieho spoločného násobku (resp. NSD).

Ukážka:

motivácia

Na koľajniciach stoja vagóny. Sú zoradené vedľa seba s tým, že medzi dvoma vagónmi je malá medzera. Vytvorte program, ktorý nakreslí práve toľko vagónov, koľko sa ich zmestí na šírku grafickej plochy. Posledný vagón nesmie z plochy vytŕčať. Každý vagónik je široký 40 bodov a vysoký 20 bodov. Medzi vagónmi je medzera 10 bodov.

Načrtnime si umiestnenie vagónov a rozmyslime si súradnice kreslenia:



Vidíme, že sa oplatí používať premennú `x` na to, aby sme si pamätali súradnice vagónu, ktorý ideme kresliť. Ak poznáme `x`, ako nakreslíme jeden vagón?

```
Image1.Canvas.Rectangle(x, 100, x+40, 120);
```

Na začiatku bude v premennej `x` hodnota `0`. O koľko treba zmeniť `x` po nakreslení vagónu? ... O `50`.

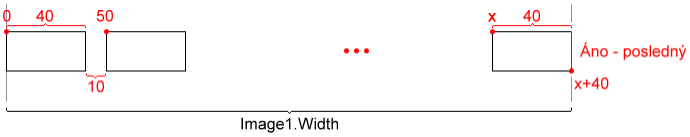
Program pre počítač by potom mohol vyzerať približne takto:

kým sa vagón zmestí do grafickej plochy, opakovane vykonávaj príkazy

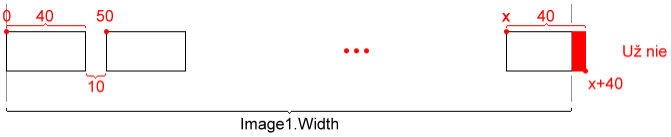
```
  Image1.Canvas.Rectangle(x, 100, x+40, 120);
  x:=x+40+10;
```

zbieranie skúseností

Ako počítač pozná, že *„sa vagón zmestí do grafickej plochy“*?



Posledný vagón sa do obrázka zmestí tak akurát. Ďalší sa už vôbec nevojde.



Takýto vagón vytŕča von. Ten už kresliť nesmieme.

Vieme nájsť nejaké všeobecné pravidlo, kedy sa ešte vagón zmestí na obrazovku? ... Ak platí, že `x+40 ≤ Image1.Width`.

Náš program bude vyzerať takto:

kým `x+40 ≤ Image1.Width`, *opakovane vykonávaj príkazy*

```
  Image1.Canvas.Rectangle(x, 100, x+40, 120);
  X:=X+40+10;
```

Predchádzajúcemu zápisu počítač ešte nerozumie. Zápis je však veľmi blízko tomu, čo počítač pozná. V jazyk Pascal zodpovedá takej programovej konštrukcii cyklus `while`. Pomocou neho zapíšeme riešenie úlohy takto:

```
x:=0;
while x+40<=Image1.Width do begin
  Image1.Canvas.Rectangle(x, 100, x+40, 120);
  X:=X+40+10;
end;
```

Cyklus `while x+40<=Image1.Width` čítame presne tak, ako sme si povedali: *„kým je x+40 menšie alebo rovné šírke plochy, vykonávaj príkazy...“*.

3.2 Podmienený príkaz `if`

Cieľ:

- porozumieť tomu, ako funguje vetvenie programu,
- naučiť používať podmienený príkaz pre rozhodovanie sa v programe v rôznych situáciách.

Nové pojmy, poznatky a zručnosti:

- konštrukcia **`if ... then ... else`**,
- konštrukcia **`if ... then`** bez vetvy **`else`**,
- syntaktické pravidlá, princíp fungovania príkazu,
- jednoduché podmienky v podmienenom príkaze,
- problémy s príkazom,
- pravda, nepravda.

Poznámky:

- Budeme sa snažiť rozpoznávať rôzne úrovne náročnosti problémov a úloh, ktoré riešime pomocou podmienených príkazov:

  I. Používame jeden podmienený príkaz s jednoduchou podmienkou:

```
if T<0 then Image.Canvas.TextOut(0, 0, 'Mrzne');
```

  II. Používame podmienený príkaz so zložitejším telom, používame ho v kontexte iných príkazov:

```
if T<18 then begin
  Image.Canvas.Font.Color:=clBlue;
  Image.Canvas.TextOut(0, 0, 'Zima');
end else begin
  Image.Canvas.Font.Color:=clRed;
  Image.Canvas.TextOut(0, 0, 'Teplo');
 end;
Image.Canvas.TextOut(0, 20, 'Vhodne sa obleč!');
```

  III. Používame podmienený príkaz v kombinácii s cyklom, keď testujeme riadiacu premennú, a popritom čosi nastavujeme (napríklad farbu) alebo si všímame iba určité hodnoty (vypisujeme iba párne čísla):

```
for i:=1 to 10 do begin
  if i mod 2=0 then Image.Canvas.Font.Color:=clRed;
  else Image.Canvas.Font.Color:=clBlue;
  Image.Canvas.TextOut(0, i*15, IntToStr(i));
end;
```

S podmieneným príkazom a vetvením programu súvisí aj konštrukcia **`case`**. Domnievame sa, že túto konštrukciu netreba učiť ihneď za témou o príkaze **`if`**, ale až oveľa neskôr a v situácii, keď bude výhodné takú konštrukciu použiť.

  IV. Používame vnorené podmienené príkazy:

```
if N<5 then Image.Canvas.TextOut(0, 0, 'Málo')
else
  if N>10 then Image.Canvas.TextOut(0, 0, 'Veľa')
  else Image.Canvas.TextOut(0, 0, 'Akurát');
```

  V. Používame podmienený príkaz so zloženou podmienkou, napríklad:

```
if (H<6) or (H>18) then
  Image.Canvas.TextOut(0, 0, 'Rozsvieť svetlo!')
```

VI. Ďalšie úlohy, v ktorých sa vyskytuje:

- rôzna kombinácia cyklov a podmienených príkazov, napríklad:

```
opakuj:=true;
while opakuj do begin
  ...
  if ... then opakuj:=false;
  ...
end;
```

- náročný algoritmus (napr. úlohy zo súťaží).

- Poradie, v akom učíme podmienený príkaz **if** a cyklus **while** nie je striktne dané a môžeme ho vzájomne vymeniť.
- Zložené podmienky učievame až vtedy, keď je to potrebné.

Postup:

1. Najskôr potrebujeme ukázať, ako funguje samotný podmienený príkaz. Preto zvolíme **triviálnu úlohu**, ktorú dokážeme riešiť jedným podmieneným príkazom **s jednoduchou podmienkou**. Napríklad: „Chceme vytvoriť program, ktorý nám poradí, ako sa máme obliecť. Ak bude vonku chladno, máme si obliecť sveter, inak nám poradí, aby sme si obliekli tričko.“
2. Problém so žiakmi analyzujeme:
   - Čo bude vstupom programu – (napr.: “Keďže náš počítač nedokáže merať vonkajšiu teplotu, zadá ju používateľ pomocou editovacieho políčka.“).
   - Ako sa **rozhodneme**, či je teplo alebo zima.
   - Algoritmus zapíšeme najskôr slovne a neformálne tak, aby sme ho neskôr dokázali šikovne prepísať:

     ```
     ak t<22 potom zobraz text 'Obleč si sveter'
     inak zobraz text 'Obleč si tričko'
     ```

   Je dôležité, aby sme použili vhodné slovenské slová, ktoré primerane vystihujú anglický preklad aj fungovanie príkazu.
3. Teraz algoritmus naprogramujeme:
   - Ukážeme, ako jednotlivé slová prekladáme, pričom jednotlivé slová príkazy zapisujeme v programovacom jazyku.
   - Upozorníme na miesta, kde sa (ne)píšu bodkočiarky.
   - Program vyskúšame.

   V programoch je cyklus akoby synonymom pre výkon, silu, masívnosť a podmienený príkaz pre rozum, logiku, inteligentné správanie.
4. So žiakmi sa zamýšľame a diskutujeme o tom, kedy sa vykoná ktorá vetva programu. Napríklad, ktorá **vetva** programu sa vykoná pre čísla: `-10`, `30`, alebo pre (hraničné) vstupy `21` a `22`.
5. Neskôr je potrebné riešiť sériu úloh, na ktorých postupne vidno:
   - že vetva môže byť zložený príkaz tvorený pomocou **begin** a **end**,
   - situácie, ako sa párujú vnorené príkazy **if** s vetvou **else**.
   - ako konštruujeme zložitejšie podmienky, atď.
6. Príklady a úlohy: v učebnici [7] v kapitole 3.4.

Pozor: Nezačíname komplexnou ukážkou s náročným algoritmom, ani s viacerými podmienenými príkazmi, s vnorenými podmienenými príkazmi, s komplikovanými podmienkami, ani s podmienenými príkazmi skombinovanými s cyklom, na ktorej „všetko predvedieme“. Napríklad, nevhodnou úvodnou úlohou je už aj úloha, v ktorej treba vypísať čísla od 1 po 10, pričom sú rôzne zafarbené párne a nepárne čísla (úloha je zbytočne komplikovaná tým, že používame cyklus a test na párnosť **if** `i` **mod** `2=0` **then** ..., lebo v tomto štádiu poznania sa zápis ťažkopádne vysvetľuje a trasuje). Takú úlohu je vhodné riešiť až neskôr – vtedy dobre poslúži ako úloha na trénovanie podmienených príkazov.

motivácia

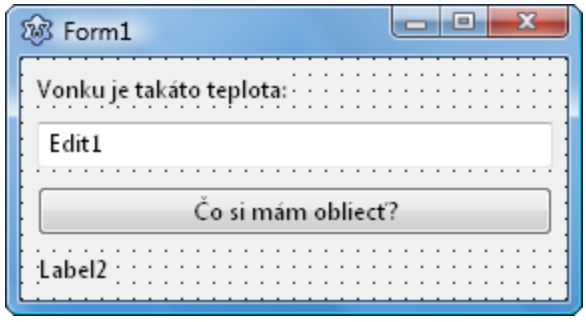
Ukážka:

1. Na jar býva zradné počasie. Vytvorme si program, ktorý nám poradí, ako sa máme obliecť. Ak bude vonku chladno, povie nám, aby sme si obliekli sveter. Inak nám odporučí, aby sme si obliekli tričko.

Teraz sa nám ešte nepodarí vyriešiť celú úlohu. Pokúsme sa ale vymyslieť a vyriešiť tie veci, ktoré zvládneme. Napríklad:

- Ako bude počítač merať vonkajšiu teplotu? ... Zatiaľ sme sa také veci neučili. Preto vystačíme s tým, že počítaču prezradíme teplotu, ktorú ukazuje náš teplomer.
- Ako zadáme túto teplotu počítaču? ... Pomocou posuvnej lišty alebo editovacieho políčko. Použime editvacie políčko.

2. Vytvorme program, ktorý umožní zadávať teplotu pomocou editovacieho políčka.



Formulár môže vyzerať napríklad takto.

3. Keď klikneme na tlačidlo, mal by náš program vykonať nasledovné:
   - získať číslo z editovacieho políčka,
   - nastaviť nápis Label2.Caption podľa hodnoty z editovacieho políčka.
4. Pamätáme sa, akým spôsobom získame číslo z editovacieho políčka? ... Áno, použijeme konverziu `IntToStr` alebo `FloatToStr`.

zbieranie skúseností

5. Výsledok uložíme do premennej, napríklad `t`. Aký príkaz napíšeme?

```
t:=IntToStr(Edit1.Text);
```

6. V premennej `t` máme teplotu. Pri akej teplote si zvykneme obliecť sveter, aby nám nebola zima? ... Dohodnime sa, že ak je menej než 22 stupňov, už si oblečieme sveter.
7. Teraz by nám mal počítač poradiť. Podľa teploty by sa mal rozhodnúť, akú správu nám zobrazí. Malo by to fungovať nejako takto:

   ak je `t<22` potom zobraz text `'Obleč si sveter'`
   inak zobraz text `'Obleč si tričko'`

8. Našťastie v programovacom jazyku Pascal sa možno takto sa rozhodovať. Preto predchádzajúci algoritmus zapíšeme nasledovne:

```
t:=IntToStr(Edit1.Text);
if t<22 then Label2.Caption:='Obleč si sveter'
else Label2.Caption:='Obleč si tričko';
```

*Pozor, pred **`else`** nedávame bodkočiarku!*

9. Príkaz **`if ... then ... else`** sa nazýva podmienený príkaz. Táto konštrukcia funguje presne tak, ako sme si pred chvíľkou povedali:
   - Ak je splnená podmienka, že `t<22`, vykoná sa príkaz za slovom **`then`**. Vtedy sa zobrazí správa so svetrom.
   - Inak sa tento príkaz nevykoná, ale vykoná sa príkaz až slovom **`else`**. Ten zobrazí správu o tričku.

3.3 Procedúry

Cieľ:

- naučiť definovať a volať vlastné procedúry,
- používať procedúry s jednoduchými parametrami.

Nové pojmy, poznatky a zručnosti:

- definícia procedúry,
- hlavička procedúry, meno procedúry, parametre, telo procedúry,
- použitie grafických príkazov vo vlastných procedúrach,
- volanie procedúry, návrat z procedúry,
- ako funguje odovzdávanie parametrov, použitie parametrov v tele procedúry.

Poznámky:

- Samotný pojem *parameter*, presnejšie to, „*ako parametre fungujú*“ alebo „*čo vidí procedúra*“, je pre mnohých žiakov veľmi náročné na pochopenie.
- Preto je v prvých programoch s procedúrami veľmi dôležité **trasovať volanie procedúry** a **znázorňovať parametre** tým, že kreslíme obrázky a škatuľky.
- Budeme rozpoznávať rôzne **úrovne náročnosti** toho, „*ako*“ alebo „*v akých*" situáciách používame vlastné procedúry:

I. Procedúra bez parametrov, napríklad:

```
procedure Kruzok;
begin
  Form1.Image1.Canvas.Ellipse(50-10, 30-10, 50+10, 30+10);
end;
```

Vo vlastných procedúrach nie sú priamo viditeľné komponenty formulára.

Preto niektorí učitelia učia radšej túto tému na tom, že sa, namiesto obyčajných procedúr, definujú metódy formulára alebo lokálne (vnorené) procedúry.

II. Procedúra s jednoduchými parametrami:

```
procedure Kruzok(x, y: Integer);
begin
  Form1.Image1.Canvas.Ellipse(x-10, y-10, x+10, y+10);
end;
```

III. Lokálne premenné v procedúrach.

IV. Volanie inej procedúry v tele inej procedúry.

```
procedure VelaKruzkov(n: Integer);
var
  i: Integer;
begin
  for i:=1 to 10 do Kruzok(i*20, 50);
end;
```

V. Procedúra s parametrami prenášanými odkazom:

```
procedure Zmen(var n: Integer);
begin
  n:=n+1;
end;
```

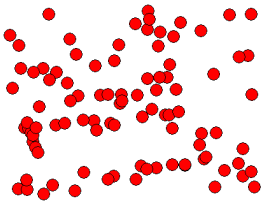
VI. Nad rámec strednej školy je napríklad:

- procedúra ako metóda (procedúra objektovej triedy),
- rekurzívne volanie,
- použitie pokročilejších programátorských techník (**forward** definícia, procedurálne typy, procedurálne parametre, ...),
- použitie procedúr v zložitejších algoritmoch alebo so zložitejšími údajovými štruktúrami.

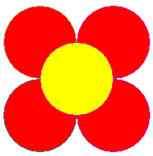
- Pokým budeme postupovať opatrne, začiatočníci zvládnu riešiť jednoduchšie úlohy na I. a II. úrovni. Parametre prenášané referenciou (**`var`** parametre) určite nebudeme učiť už na začiatku, a dokonca bez nich vydržíme aj dlhšiu dobu.
- Mali by sme si uvedomiť, prečo sa procedúry v programoch používajú:
  - Je to preto, aby sme v programe nemali rovnaký algoritmus na viacerých miestach. Znamená to, že taký algoritmus **naprogramujeme iba raz** a neskôr ho **viac krát používame** - vyvolávame.
  - V rozsiahlejších projektoch zvykneme jeden celistvý blok kódu rozsekať na menšie logické časti. Algoritmy, ktoré so sebou nesúvisia oddelíme. Tým celý program sprehľadníme a jednotlivé algoritmy zároveň pomenujeme.
  - Znamená to, že procedúry nám pomáhajú **upratať** program.
- Naši žiaci sa s procedúrou už stretli (napríklad, procedúra `Button1Click`). Zatiaľ však nevedia, čo presne procedúra znamená a ako niektoré mechanizmy fungujú.

Postup:

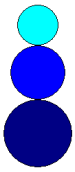
1. Mali by sme vymyslieť vhodne jednoduchú situáciu, keď sa procedúry používajú. Vhodná motivácia je napríklad takáto: „Veľakrát sme kreslili krúžky - chýbal nám príkaz na ich kreslenie."
2. Pripomenieme si, ako sa nakreslí náhodný krúžok (bez procedúry).
3. „Naučíme počítač nový príkaz `Kruzok`":
   - ukážeme miesto, **kde sa procedúra definuje**,
   - predvedieme, **ako sa procedúra definuje** - ako vyzerá hlavička a telo,
   - od teraz budeme rozlišovať **vlastné** a **formulárové** procedúry,
   - povieme, že vo vlastných procedúrach musíme písať: `Form1.Image1.Canvas...` preto, lebo *„vlastná procedúra nevidí komponenty, ktoré sú vo formulári"*.



4. Ak takýto program spustíme, uvidíme, že sa ešte nič nekreslí. Je to preto, lebo potrebujeme počítaču prikázať, aby našu procedúru vykonal. Preto, pri stlačení tlačidla, **vyvoláme** náš príkaz - procedúru `Kuzok`.
5. Predvedieme, teda **trasujeme**, ako prebieha **volanie procedúry** `Kruzok`:
   - čo sa stane, ak procesor natrafí na náš vlastný príkaz,
   - ako sa vykonáva telo procedúry a čo sa deje s parametrami,
   - návrat z procedúry.
6. Vyskúšame aj iné miesta, kde sa dá vlastná procedúra vyvolať. Napríklad, pri stlačení iného tlačidla sa vykoná cyklus, ktorý procedúru zavolá 100 krát.



7. Trénujeme so žiakmi procedúry bez parametrov. Vyskúšame zadefinovať iné procedúry: zmazanie obrazovky, kreslenie náhodnej úsečky. Ďalšie námety na trénovanie sú napríklad v učebnici [7] v kapitole 3.1.
8. Ďalej učíme definovať procedúry s parametrami: *„Nechceme, aby sa krúžok kreslil na náhodné miesto. Videli sme, že niektorým príkazom môžeme prezradiť, kam majú kresliť alebo vypisovať text. Aj my by sme chceli, aby náš príkaz poznal parametre. Potom by sme, napríklad krúžok so stredom v bode 100, 50, nakreslili takto:* `Kruzok(100, 50)`."
9. Zadefinujeme procedúru s parametrami.



10. Je extrémne dôležité, aby sme názorne **trasovali** volanie procedúry s parametrami s konkrétnymi hodnotami, napríklad `Kruzok(100, 50)`. Popri tom: kreslíme škatuľky, znázorňujeme to, **čo sa deje s parametrami** a ukazujeme, ako sa príkazy v procedúre „pozerajú" do škatuliek - parametrov. Zo skúseností vieme, že takéto detailné trasovanie bude potrebné v budúcnosti aj viackrát opakovať.

Ukážka:

1. Naša procedúra `Kruzok` kreslila kruh na náhodnej pozícii:

```
procedure Kruzok;
var
  X, Y: Integer;
begin
  X:=Random(400);
  Y:=Random(300);
  Form1.Image1.Canvas.Ellipse(X-10, Y-10, X+10, Y+10);
end;
```

zbieranie skúseností

2. Takáto procedúra je málo užitočná, pretože zvyčajne nepotrebujeme kresliť kruhy na náhodné miesta, ale na také, ktoré sami určíme. Napríklad, ani príkaz `TextOut` nevypisuje texty hocikam, ale iba na určité miesto:

```
... TextOut(100, 50, 'Ahoj')
```

Vieme povedať, kam sa vypíše pozdrav? ... Áno, pretože vieme, že prvé dva parametre príkazu určujú pozíciu textu.

motivácia

zbieranie skúseností

3. Aj naše vlastné príkazy - procedúry dokážu takto fungovať, ak ich to naučíme. Potom by sme mohli písať, napríklad:

```
Kruzok(100, 50) ... nakreslí krúžok so stredom v bode x=100, y=50,
Kruzok(200, 50) ... nakreslí krúžok so stredom v bode x=200, y=50.
```

Vidíme, že parametre pre procedúru `Kruzok` môžu mať rôzne hodnoty. Ako sa procedúra dozvie tieto hodnoty?

motivácia

4. Upravíme definíciu procedúry `Kruzok` - o chvíľu vysvetlíme, čo to znamená:

```
procedure Kruzok(X, Y: Integer);
begin
  Form1.Image1.Canvas.Ellipse(X-10, Y-10, X+10, Y+10);
end;
```

Aby procedúra vedela pracovať s parametrami, pomenujeme ich menami `X`, `Y`. Parametre pomenujeme priamo v hlavičke procedúry.

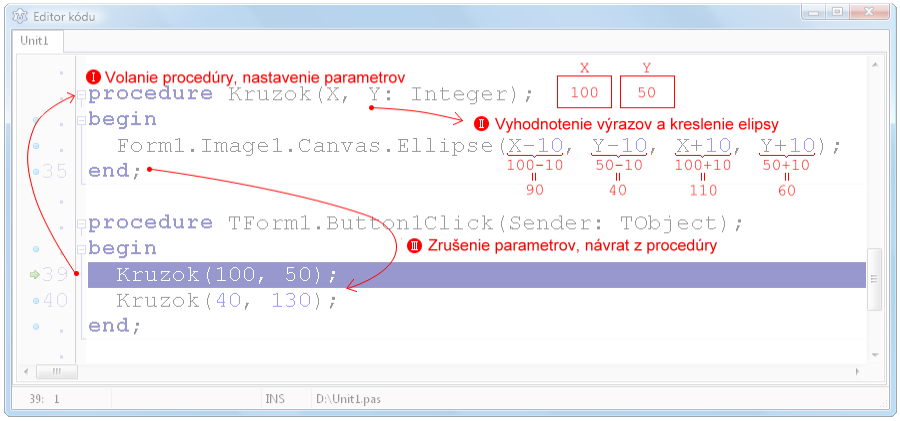
zbieranie skúseností

5. Vyskúšajte, či príkazy nakreslia krúžky na správne miesta:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Kruzok(100, 50);
  Kruzok(200, 50);
end;
```

6. Vidíme, že naša procedúra očakáva dva parametre - dve celé čísla. Poďme sa teraz pozrieť, čo sa deje pri volaní procedúry `Kruzok(100, 50)`:

motivácia



Pri volaní `Kruzok(100, 50)` vzniknú v pamäti počítača parametre `X`, `Y`. Tie sa správajú podobne, ako premenné. Pri volaní procedúry sa parametrom nastavia hodnoty tak, že `X`=100 a `Y`= 50. S týmito číslami pracuje príkaz `Ellipse`.

zbieranie skúseností

3.4 Funkcie

Cieľ:

- naučiť definovať a používať (volať) vlastné funkcie,
- naučiť pracovať s návratovou hodnotou funkcie.

Nové pojmy, poznatky a zručnosti:

- definícia funkcie, typ výsledku,
- návratová hodnota,
- volanie funkcie a spracovanie výsledku.

Poznámky:

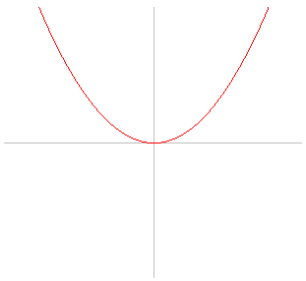
- Vzhľadom na to, že procedúry a funkcie sú do značnej miery príbuzné témy, rozpoznávame pri vyučovaní funkcií podobné úrovne náročnosti, ako pri procedúrach.
- Oproti procedúram sú funkcie mierne komplikovanejšie kvôli tomu, že funkcie vracajú nejaký výsledok. Preto potrebujeme vo funkcii určiť **návratovú hodnotu** a mimo funkcie (po zavolaní) výsledok spracovať.
- Tým, že žiaci poznajú do určitej miery pojem funkcie už z matematiky, získavame pri budovaní tohto pojmu značnú výhodu.
- Pred tým, ako budeme učiť definovať vlastné funkcie, odporúčame, aby už žiaci s niektorými funkciami pracovali, aby ich používali. Napríklad, aby vypísali funkčné hodnoty známych funkcií, nakreslili pravidelný n-uholník, nakreslili graf niektorej funkcie (`sin` alebo `cos`)

Pojem funkcie v programovaní je od matematického chápania tohto pojmu odlišný. Ako vieme, matematická funkcia musí pre rovnaký parameter *n* (ktorého hodnota pochádza z definičného oboru funkcie), vrátiť práve jednu funkčnú hodnotu f(*n*). V programovaní to tak nie je. Napríklad vieme, že funkcia `random(10)` má pre parameter `10` pripravených až 10 rôznych výsledkov...

Pri programátorských funkciách bývajú žiaci zaskočení z toho, že telo funkcie môže obsahovať podmienený príkaz či cyklus. To sú konštrukcie, na ktoré, neboli v matematike zvyknutí.

Postup:

1. Tému funkcie začíname učiť pripomenutím si toho, čo žiaci o funkciách vedia: aké funkcie poznajú, ako používame niektorú známu matematickú funkciu. Mali by uvidieť, že funkcia „dáva“ pre zadaný **parameter** výslednú hodnotu.
2. Motivácia môže vychádzať z matematiky: „Existujúce funkcie v jazyku Pascal nám niekedy nestačia. Napríklad, funkcia `f(x)=` x^2`+2x+4` nie je k dispozícii. Keby sme chceli počítať funkčné hodnoty pre `x=1`, `x=5` aj `x=10`, museli by sme počítať výrazy: `1*1+2*1+4`, `5*5+2*5+4` aj `10*10+2*10+4`. To je dosť nepohodlné...“
3. Prepíšeme funkciu `f(x)=` x^2`+2x+4` do programovacieho jazyka:
   - ukážeme miesto, kde sa funkcia definuje,
   - predvedieme, ako sa definuje - ako vyzerá hlavička funkcie,
   - priradením `Result:=x*x+2*x+4` **nastavíme výsledok funkcie**,
4. Funkciu otestujeme a zobrazíme výsledky pre `f(1)`, `f(5)` a `f(10)`.
5. Predvedieme a **trasujeme** volanie funkcie `f(1)`:
   - čo sa stane v situácii, keď v programe nastáva volanie našej funkcie,
   - čo sa stane s parametrom pri volaní funkcie,
   - ako sa počíta hodnota výsledku,
   - čo sa stane s výsledkom pri návrate z funkcie.
6. Trénujeme programovanie a používanie ďalších vlastných funkcií:
   - druhá mocnina čísla, súčet dvoch čísel, prevod stupňov na radiány,
   - súčet prvých *n* čísel (pomocou cyklu), zistenie minima a maxima a pod.
   - kreslíme grafy našich funkcií (bodkový, čiarový, stĺpcový).

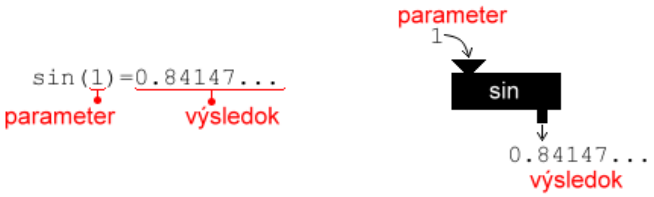


Graf funkcie f(x)=x^2/100.

Pozor: S matematikou to nepreháňame – sme na informatike. Nevhodné úlohy na funkcie sú také, ktoré vyžadujú použitie matematického algoritmu, napríklad výpočet spoločného deliteľa pomocou Euklidovho algoritmu (pokým tento algoritmus neprezradíme, žiaci ho prakticky nemajú šancu sami objaviť).

Ukážka:

1. Určite poznáte nejaké matematické funkcie. Aké? ... Sínus, mocnina...
2. Všimnime si, ako napríklad používame funkciu sínus:



Pre zadaný parameter funkcia dáva výslednú hodnotu. Mechanizmus, ako sa parameter zmení na výslednú hodnotu, je skrytý. Funkcia je pre nás zatiaľ iba čierna skrinka, ktorá nejaký spôsobom počíta výsledky.

zbieranie skúseností

3. Postupne sa však naučíme, ako funkcie počítajú svoje výsledky. Keďže sínus je pre príliš komplikovaný, ďalej budeme pracovať s funkciou: `f(x)=x²+2x+4`.

motivácia

4. Ako by sme vypísali funkčné hodnoty pre `f(1)`, `f(5)` a `f(10)`?

```
Image1.Canvas.TextOut(0, 100, FloatToStr(1*1+2*1+4));
Image1.Canvas.TextOut(0, 120, FloatToStr(5*5+2*5+4));
Image1.Canvas.TextOut(0, 140, FloatToStr(10*10+2*10+4));
```

Všade, kde sme vo výraze videli `x`, dosadili sme konkrétnu hodnotu.

zbieranie skúseností

5. Takýto postup je nešikovný, lebo sa pri rozpisovaní jednotlivých výrazov nesmieme pomýliť. Keby Pascal poznal našu funkciu `f`, potom by predchádzajúci program vyzeral takto:

```
Image1.Canvas.TextOut(0, 100, FloatToStr(f(1)));
Image1.Canvas.TextOut(0, 120, FloatToStr(f(5)));
Image1.Canvas.TextOut(0, 140, FloatToStr(f(10)));
```

Namiesto krkolomných výrazov by sme použili jednouchý zápis f(parameter).

zovšeobecňovanie

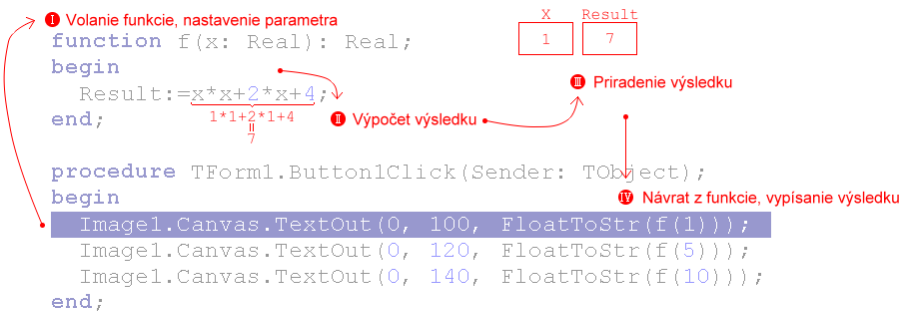
6. Preto teraz počítač naučíme, aby poznal našu funkciu `f`. Postupujeme takto:

```
function f(x: Real): Real;
begin
  Result:=x*x+2*x+4;
end;
```

7. Vidíme, že definovanie funkcie pripomína definovanie procedúry:
   - Hlavička funkcie však začína vyhradeným slovom **`function`**.
   - Nasleduje meno funkcie a zoznam parametrov - naša funkcia má meno `f` a očakáva jeden parameter `x` typu reálne číslo.
   - Na konci hlavičky je `:Real`. Znamená to, že výsledok, ktorý funkcia vypočíta, môže byť aj desatinné číslo.
   - Priradením `Result:=` určíme hodnotu, ktorá bude výsledkom funkcie.

zbieranie skúseností

8. Trasujeme, kreslíme a vysvetľujeme, čo sa deje pri vykonávaní príkazu:



`Result` fungujeme ako špeciálna premenná. Číslo `7` z tejto premennej sa použije ako výsledok volania funkcie `f(1)`.

3.5 Myš

Cieľ:

- naučiť spracovávať vstupné údaje, ktoré prichádzajú od počítačovej myši,
- vytvoriť jednoduchý kresliaci program.

Nové pojmy, poznatky a zručnosti:

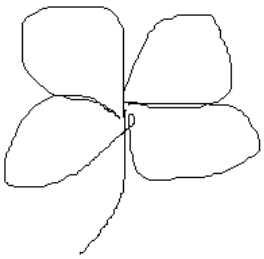
- udalosť,
- `OnMouseDown`, `OnMouseMove`, `OnMouseUp`,
- zisťovanie stavu myši – pozícia, stlačené tlačidlá,
- ako fungujú parametre `X`, `Y` v procedúrach, ktoré obsluhujú udalosti myši.

Poznámky:

- Algoritmy s myšou je vďačná téma. Pre žiakov sú motivujúce úlohy, v ktorých sa pomocou myši kreslí alebo ovláda nejaký objekt.
- Rozpoznávame rôzne stupne náročnosti:

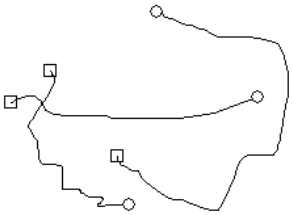
  I. Reagujeme iba na pohyb myši:

  - Zaujíma nás iba pozícia myši, zatiaľ netestujeme, či sú stlačené tlačidlá myši.
  - Napríklad: súradnice myši vypisujeme do grafickej plochy, kreslíme bodky na mieste, kde sa nachádza kurzor myši (na tomto príklade zároveň vidno, že medzi bodkami sú rôzne veľké medzery podľa toho, ako rýchlo myšou hýbeme).



  II. Reagujeme na stlačenie a pustenie tlačidla myši. Napríklad, pri stlačení tlačidla myši čosi nakreslíme a pri pustení tlačidla nakreslíme iný útvar.

  III. Niečo jednoduché realizujeme iba počas toho, ako je tlačidlo myši stlačené. Napríklad, kreslíme so stlačeným tlačidlom myši alebo naťahuje úsečku, ktorá sa dokreslí až pri pustení tlačidla myši.

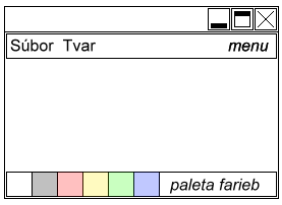


  IV. Označovanie objektov klikaním, napríklad:

  - Číslo objektu vieme triviálnym spôsobom vypočítať zo súradníc myši (napríklad, delením – ak sú objekty pravidelne rozmiestené).
  - Napríklad: Vyberáme farbu z palety farieb. Klikaním prefarbujeme štvorčeky na hracej ploche a pod.
  - Ak (v budúcnosti) použijeme časovač, klikáme na pohyblivý cieľ.

  V. Ťahanie (dragovanie) objektov:

  - Učíme sa testovať kolízie: v zozname objektov hľadáme ten objekt, nad ktorým sme stlačili tlačidlo myši.
  - Pri pohybe myši označený objekt napríklad presúvame.



  VI. Používanie myši v zložitejších situáciách:

  - Rôznym spôsobom reagujeme na druhé alebo viaceré stlačenia myši. Napríklad, pri prvom stlačení označíme prvý objekt, pri druhom označíme druhý objekt a zároveň s prvým aj druhým objektom čosi realizujeme.
  - Rôzne objekty reagujú rôznym spôsobom na udalosti myši.

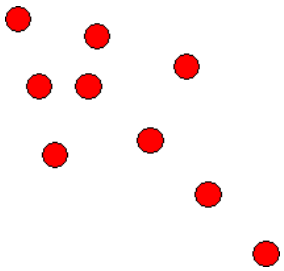
- Väčšina žiakov rýchlo zvládne riešiť problémy na úrovni I a II. Úroveň V a VI už môže byť nad rámec strednej školy.

Postup:

1. Motivácia bude založená na tom, že budeme skúmať, ako sa pracuje s myšou a neskôr sa pokúsime vytvoriť jednoduchý kresliaci program.
2. Zhrnieme známe fakty, ktoré sa týkajú počítačovej myši:
   - ako pracuje vstupné zariadenie myš,
   - čo znázorňuje šípka na obrazovke,
   - ktoré informácie o myši sú pre programy dôležité.
3. Vytvoríme jednoduchú pokusnú aplikáciu, na ktorej vysvetlíme princíp, ako v programe zisťujeme pozíciu myši:
   - aplikácia bude reagovať na pohyb myši (tlačidlá nepotrebujeme),
   - stačí, ak budeme používať iba komponent obrázok `Image1`,
   - najskôr budeme pri pohybe vypisovať iba súradnice myši, napríklad:
     ```
     Image1.Canvas.TextOut(0, 0, IntToStr(X));
     Image1.Canvas.TextOut(0, 15, IntToStr(Y));
     ```
4. Tak, ako pracujeme na tvorbe uvedenej aplikácie, budeme musieť vysvetľovať jeden nový a dôležitý programátorský koncept - **udalosti**:
   - Stručne objasníme, čo sú udalosti, napríklad: „*sú to dôležité okamihy, ktoré nastávajú v živote programu*“.
   - Uvedieme niektoré príklady udalostí: kliknutie tlačidlo Button1, pohyb myši, spustenie programu.
   - V *Inšpektore objektov* preskúmame udalosti, ktoré sa týkajú `Image1`.
   - Objasníme, ako fungujú parametre `X`, `Y` v procedúre, ktorá sa volá pri pohybe myši.
5. Preskúmame, ako program reaguje na pohyb myši. Žiaci by si mali odniesť nasledujúce skúsenosti:
   - **Kedy nastávajú udalosti**? ... Iba pri pohybe myši ponad `Image1`.
   - Ak ale stlačíme ľavé tlačidlo myši nad obrázkom `Image1`, potom pri pohybe myši nastáva udalosť `OnMouseMove` vždy. Vtedy môžu byť súradnice myši dokonca záporné!
6. Program upravíme tak, aby sa pri pohybe myši niečo kreslilo:
   ```
   Image1.Canvas.Ellipse(X-5, Y-5, X+5, Y+5);
   ```
7. Žiaci by si mali odniesť skúsenosť, že čím rýchlejšie hýbeme myšou, tým sú medzi bodkami medzery väčšie. Znamená to, že pozícia myši sa „meria“ iba v určitých okamihoch.
8. Ďalej skúsime pospájať medzery krátkymi úsečkami:
   ```
   Image1.Canvas.LineTo(X, Y);
   ```
9. Už takýto jednoduchý kresliaci program vyvoláva u mnohých žiakov prirodzenú motiváciu k tomu, aby sa čiary kreslili iba vtedy, keď držíme **stlačené tlačidlo myši**. Ďalej môžeme postupovať, napríklad takto:
   - Potrebovali by sme testovať, či je stlačené tlačidlo myši:
     ```
     if JeStlacene then Image1.Canvas.LineTo(X, Y)
     ```
   - Prezradíme, že existujú udalosti `OnMouseDown` a `OnMouseUp` a stručne vysvetlíme, pri akej príležitosti tieto udalosti nastávajú.
   - Navedieme žiakov na to, aby použili globálnu premennú `JeStlacene` a túto premennú nastavovali na hodnotu **`true`** alebo **`false`** pri vhodnej príležitosti (predpokladáme, že s typ `Boolean` už poznajú).
10. Námety na ďalšie úlohy sú v kapitole 3.6 v učebnici [7].

Je možné, že sa žiaci s pojmom udalosť stretli už skôr (v iných prostrediach). Záleží od postupu, ktorý sme si pri vyučovaní zvolili.

Udalosti a paradigma **udalosťami riadeného programovania** je veľmi dôležitý programátorský koncept. O udalostiach (ich vzniku, šírení v programe a spracovaní) sa dá rozprávať veľmi veľa. Našim cieľom však nie je robiť siahodlhú prednášku. Preto sa pri vysvetľovaní budeme koncentrovať iba na dôležité veci, ktoré priamo súvisia s vytvorením jednoduchého kresliaceho programu. Ideu udalostí však musíme stručne objasniť.



Pozor: Či je stlačené tlačidlo myši, vieme testovať aj bez použitia globálnej premennej a udalostí `OnMouseDown`, `OnMouseUp`. Stačí, ak by sme v procedúre obsluhujúcej udalosť `OnMouseMove` testovali parameter `Shift`:

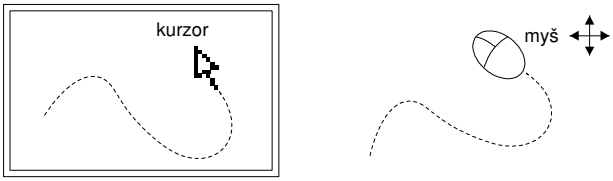
```
if ssLeft in Shift then Image1.Canvas.LineTo(X, Y);
```

Test, či je `ssLeft` prvkom množiny `Shift` však môžu naši žiaci chápať ako mágiu, pretože sa s množinami, ako údajovým typom v jazyku Pascal, ešte nestretli. Preto momentálne uprednostňujeme riešenie s globálnou premennou.

motivácia

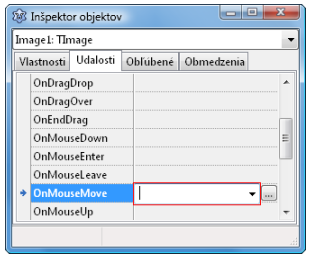
Ukážka:

1. "Doposiaľ sme používali myš iba na to, aby sme ňou ovládali iné programy, napríklad sme pomocou myši presúvali okná alebo klikali na tlačidlá. Teraz sa naučíme používať myš inak. Vytvoríme napríklad jednoduchý program, ktorý nám umožní kresliť obrázky pomocou myši. Poďme sa pozrieť, ako myš funguje.
2. K počítaču sú pripojené rôzne zariadenia. Niektoré označujeme ako vstupné, iné ako výstupné. Aké zariadenie je myš? ... Vstupné.
3. Tak, ako myšou hýbeme po stole, hýbe sa aj kurzor myši na obrazovke:



Pre programy je dôležité poznať pozíciu kurzora na obrazovke, ale aj to, ktoré tlačidlá myši sú stlačené.

4. Do formulára vložíme iba jeden komponent - obrázok `Image1`.
5. Preskúmajme záložku *Events* pre komponent `Image1` v *Inšpektore objektov*:



Events = udalosti. V ľavom stĺpci sú názvy jednotlivých udalostí.

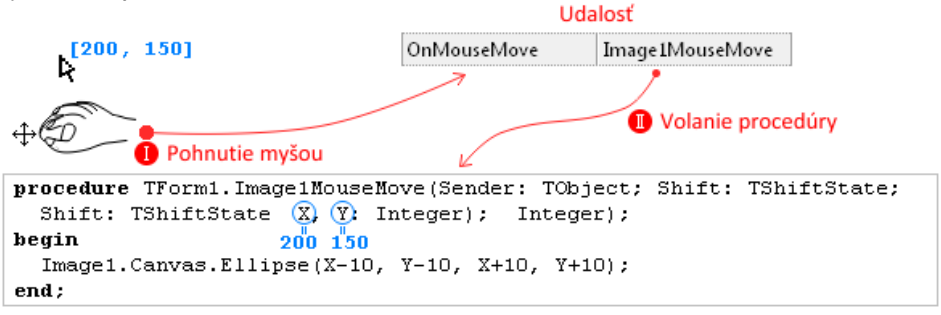
zbieranie skúseností

6. Na pohyb myši budeme reagovať pomocou mechanizmu udalostí:
   - Udalosť je dôležitý a významný okamih, ktorý nastal počas behu programu.
   - Udalosťou je pohyb myši, stlačenie tlačidla myši, ale napríklad aj spustenie programu alebo kliknutie na tlačidlo Button1.
   - Každý komponent nám umožňuje reagovanie iba na určité udalosti.
7. Nájdite udalosť *OnMouseMove*. Čo znamená po slovensky? ... *Pri pohybe myši*.
8. Vedľa `OnMouseMove` je prázdne políčko - dvojklikneme doň. Do políčka sa automatiky sa doplní `Image1MouseMove`. Zároveň vznikne nová procedúra:

```
procedure TForm1.Image1MouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin

end;
```

9. Doplňme do procedúry nasledujúci príkaz:
   `Image1.Canvas.Ellipse(X-5, Y-5, X+5, Y+5);`
10. Vyskúšame program. Trasujeme, kreslíme a vysvetľujeme, čo sa deje pri pohybe myši a ako príkaz `Ellipse` vie, kam má kresliť krúžok...



3.6 Polia

Cieľ:

- naučiť evidovať si a spracovávať veľké množstvo údajov v poli,
- pochopiť organizáciu poľa a ideu práce s prvkami poľa pomocou indexov.

Doposiaľ, ak nepočítame premenné, sme žiakov zväčša učili používať rôzne programové konštrukcie. Polia sú výnimočné, pretože pole predstavuje údajovú štruktúru, ktorá umožňuje uchovávať a organizovať veľké množstvo údajov.

Nové pojmy, poznatky a zručnosti:

- pole, prvok poľa, index, hranice poľa,
- syntaktické pravidlá pre deklarácia poľa (**`array ... of`**),
- operácia prístupu k prvkom poľa pomocou indexov `[...]`,
- nastavenie prvku poľa, použitie prvku vo výraze, výpis prvkov poľa.

Poznámky:

- Téma pole je pre žiakov veľmi **náročná**. Hoci indexy pomáhajú zapisovať riešenie problému tak, aby bolo kratšie a všeobecnejšie, výsledné abstraktné zápisy sú náročnejšie na pochopenie. Ťažkosti spôsobuje najmä nová idea – používanie indexov pre prístup k prvkom poľa.

V prvom ukážkovom príklade odporúčame voliť rozumne **malý počet prvkov**. Napríklad, na ilustrovanie základných situácií 3 prvky nestačia. Naopak, 100 prvkov je už veľa. 100 premenných sa nám nepodarí deklarovať. Ani celé 100 prvkové pole sa nám nepodarí nakresliť na tabuľu. Pri poliach treba rôzne situácie znázorňovať veľmi poctivo. Napríklad, ak si ušetríme námahu a zápisy v prvých príkladoch budeme skracovať pomocou troch bodiek . . ., môžeme takéto „vylepšenia“ očakávať aj v žiackych programoch.

- Je veľmi dôležité, aby sme **prvky poľa znázorňovali** a prvé programy pomaly a názorne **trasovali**. Preto kreslíme chlieviky (vedľa seba umiestnené škatuľky), nad ne píšeme indexy, do chlievikov vpisujeme konkrétne hodnoty podľa toho, ako sa program vykonáva.
- Budeme sa snažiť rozpoznávať rôzne **úrovne náročnosti** problémov a úloh, ktoré riešime pomocou poľa:

  I. Indexovanie konštantou.

  II. Jednoduché indexovanie prvkov poľa v cykle s riadiacou premennou, pričom zatiaľ nepoužívame žiadne testy, napríklad:
  - výpis prvkov,
  - kreslenie úsečiek podľa prvkov poľa,
  - nastavenie prvkov poľa náhodnými hodnotami.

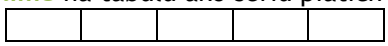
  III. Jednoduché algoritmy s poľami, v ktorých treba čosi zistiť, napríklad:
  - súčet prvkov poľa,
  - zistenie minima, maxima,
  - kopírovanie prvkov poľa podľa jednoduchých kritérií,
  - koľko prvkov v poli je záporných,
  - koľko prvkov je väčších, ako sú obaja susedia (lokálne maximum).

  IV. Pole, ako jednoduchý zoznam hodnôt (plus, v celočíselnej premennej evidujeme počet obsadených prvkov poľa):
  - výpis alebo kreslenie zoznamu,
  - pridávanie prvku do zoznamu,
  - odstraňovanie prvku zo zoznamu,
  - vyhľadanie hodnoty podľa jednoduchého kritéria.

  V. Témy, ktoré s poľom súvisia, ale výrazne prekračujú rámec strednej školy:
  - triediace algoritmy (či už nerekurzívny insertsort a heapsort alebo rekurzívny mergesort a quicksort atď.),
  - štruktúry (zásobník, rad, hašovacia tabuľka, graf),
  - stratégie riešenia (dynamické programovanie).
- Vidíme, že náročnosť úlohy s poľami závisí veľmi výrazne od toho, ako náročné algoritmy musíme pri riešení objaviť alebo použiť.

Postup:

1. Motiváciu pre použitie poľa môžeme nájsť v samotnej podstate toho, prečo sa polia v programovacom jazyku používajú. Začnime úlohou, napríklad:
   - „Chcem si evidovať, koľko kilometrov som prešiel na bicykli za posledných 5 dní.“
   - Úlohu najskôr riešime bez použitia poľa – teda s tým, že deklarujeme 5 celočíselných premenných: `a1`, `a2`, `a3`, `a4`, `a5`. Do premenných priradíme konštantné hodnoty. Napíšeme výraz, ktorý spočíta hodnoty v premenných a výsledok vypíšeme.
   - Opýtame sa, či si žiaci vedia predstaviť, ako by vyzeralo riešenie napríklad, pre celý rok. Mali by sme navodiť dojem, že *„použite 365 premenných týmto spôsobom by bolo maximálne nešikové“*.
2. Ukážeme, ako sa riešenie zjednoduší, ak v takejto situácii použijeme **pole**:
   - Najskôr pole **deklarujeme**: „`var A: array[1..5] of Integer`“.
   - Vysvetlíme, že *„v pamäti vznikne 5 premenných, ktoré budú umiestnené za sebou“*. Viac detailov ohľadom deklarácie nemá zmysel momentálne vysvetľovať – vysvetlíme ju postupne a neskôr.
   - Pole zároveň **kreslíme** na tabuľu ako sériu piatich krabičiek – chlievikov:

     `A` 
   - Vysvelíme a na obrázku zároveň ukážeme, čo sú **prvku poľa** a **indexy**. Chlieviky na tabuli popri tom očíslujeme.
3. Preskúmame, ako fungujú indexy:
   - “*Prvky poľa majú špeciálne mená:* `A[1], A[2], A[3], A[4], A[5].`“
   - Začíname s príkladmi, v ktorých sa postupne používa:
     - **konkrétne konštantné číslo ako index**: `A[1]:=15; A[2]:=9;`
     - **konštantný výraz ako index**: `A[2+1]:=31;`
     - **premenná ako index**: `i:=4; A[i]:=17;`
     - **výraz s premennou**: `A[i+1]:=12;`
   - Výsledky predchádzajúcich priradení popritom vpisujeme do chlievikov.
4. „Teraz skúsme prvky poľa vypísať.“ Podobne ako pri cykloch, aj teraz učíme žiakov **objavovať všeobecné vzťahy**. Preto najskôr úlohy riešime bez cyklu, pomocou piatich príkazov s konkrétnymi číslami ako indexmi.
5. Ďalej môžeme riešiť problémy:
   - „Nakreslite (stĺpcový) graf, ako sa cyklistovi počas 5 dní darilo.“
   - „Zistite, akú dlhú trasu prešiel cyklista“ (pri spočítavaní prvkov poľa postupujeme podobne, ako v bode 4).
   - „Zistite, koľko najviac kilometrov dokázal cyklista prejsť za jeden deň.“
   - „Zistite, v ktorý deň prešiel najviac kilometrov.“
   - „Vygenerujme počet kilometrov pre každý deň náhodne.“

Podobne, ako nám pri premenných pomáhal krabičkový model, pri poliach pomôže predstava číslovaných **chlievikov**.

Treba si zvyknúť na to, že pri poliach **intenzívne znázorňujeme** a kreslíme situácie, ktoré v programe nastávajú.

Abstraktné vzorce, ako `Suma:=Suma+A[i]` nesmieme prezrádzať žiakom ako hotový fakt: *„takto prvky sčítame“*. Takéto vysvetľovanie pôsobí to ako mágia. Naopak, radšej **venujeme čas** procesu zovšeobecňovania – teda tomu, aby žiaci videli, ako z konkrétnych zápisov vznikajú všeobecné vzorce. Hoci je tento proces časovo náročnejší, žiaci ho musia zažiť, aby dokázali aj v budúcnosti sami riešiť podobné problémy.

Pole farieb

Pozor:

- Nezačíname príkladom, kde ihneď kombinujeme cyklus so zápisom `a[i]`, ktorý je už do značnej miery abstraktný.
- V prvom ukážkovom príklade **nekombinujeme pole s náročným algoritmom**. Zlým úvodným príkladom je triedenie (alebo vyhľadávanie). Pole a triedenie by boli pre žiakov dve náročné veci, ktoré by sme riešili v jednom príklade.
- Nepoužívame komplikovanú motiváciu s veľkých počtom prvkov alebo s prvkami veľkej hodnoty, pri ktorých žiak nemá šancu rýchlo si overiť správnosť výsledku.
- Nezačíname riešením úlohy s komplikovanými pravidlami.
- V prvej úlohe odporúčame používať také (akurát veľké) hodnoty pre prvky poľa, aby sa nemýlili s indexmi.
- Žiakov je potrebné viesť k tomu, aby sami objavovali všeobecnejšie zápisy. Napríklad, nepovieme im, že *„takto* `for i:=1 to 10 do A[i]:=0` *vynulujeme prvky poľa“*. K uvedenému zápisu vedie niekoľko krokov, ktoré by sme mali tak zostaviť, aby sami žiaci nakoniec objavili uvedený zápis.

Ukážka:

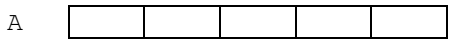
1. Pole, anglicky *array*, je skupina premenných. Skôr, ako môžeme pole používať, musíme ho deklarovať:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A: array[1..5] of Integer;
begin

end;
```

Pole deklarujeme podobne, ako iné premenné.

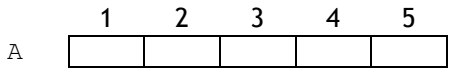
2. V pamäti počítača vznikne 5 premenných, ktoré budú umiestnené za sebou:



Každá škatuľka na obrázku je jedna celočíselná premenná.

3. Tieto premenné nazývame *prvky poľa*. Vidíme, že naše pole má 5 prvkov.

4. Prvky poľa sú označené číslami:



zbieranie skúseností

5. Tieto čísla nazývame indexy. Vidíme, že:
   - Prvý prvok poľa má index `1`.
   - Posledný prvok poľa má index `5`.
   - Aký index má prostredný prvok poľa? ... Má index `3`.

6. Deklaráciu `var A: array[1..5] of Integer` čítame takto: „*`A` je pole, ktoré má prvky s indexmi od `1` po `5`, pričom každý prvok poľa je celé číslo.*“

7. Vráťme sa k úlohe s cyklistom. Skúsme nastaviť jednotlivým prvkom poľa hodnoty podľa toho, koľko kilometrom cyklista v daný deň prešiel:

```
A[1]:=15;      // prvý deň prešiel 15 kilometrov
A[2]:=9;       // 2. deň ... 9 km
A[2+1]:=31;    // 3. deň ... 31 km
```

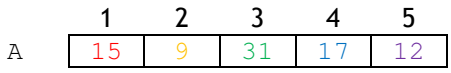
8. Indexy nemusia byť iba konštantné čísla alebo výrazy. Ak máme celočíselnú premennú `i`, budú fungovať aj nasledujúce príkazy:

```
i:=4;
A[i]:=17;      // 4. deň ... 17 km
A[i+1]:=12;    // 5. deň ... 12 km
```

Prvky poľa majú akoby špeciálne mená `A[index]`.

zovšeobecňovanie

9. Zároveň trasujme, čo vykonal každý z príkazov. Pole vyzerá takto:



10. Presvedčme sa, že prvky poľa majú naozaj také hodnoty, aké sme im nastavili:

```
Image1.Canvas.TextOut(0, 15, IntToStr(A[1]));
Image1.Canvas.TextOut(0, 30, IntToStr(A[2]));
Image1.Canvas.TextOut(0, 45, IntToStr(A[3]));
Image1.Canvas.TextOut(0, 60, IntToStr(A[4]));
Image1.Canvas.TextOut(0, 75, IntToStr(A[5]));
```

zbieranie skúseností

Vidíme, že príkazy sa opakujú. Ako závisí súradnica výpisu s indexom prvku?

11. Ak sa pozorne pozrieme na indexy a súradnice, objavíme takúto závislosť. Ak `i` je index, potom súradnicu počítame podľa vzorca `15*i`. Čiže môžeme písať:
    `Image1.Canvas.TextOut(0, 15*i, IntToStr(A[i]));`
    Keby sme `i` postupne menili od `1` po `5`, vypísali by sa všetky prvky.

3.5 Súbory

Cieľ:

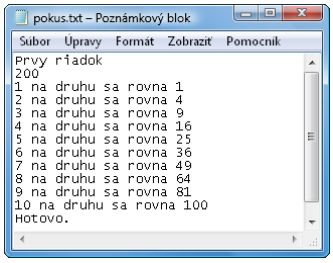
- porozumieť idei, ako sú údaje uložené v textovom súbore,
- naučiť zapisovať a čítať údaje z textového súboru.

Nové pojmy, poznatky a zručnosti:

- textový súbor, súborová premenná,
- otvorenie súboru, čítanie a zápis do súboru, zatvorenie súboru,
- princíp, ako sú v súbore uložené údaje,
- koniec riadku, začiatok a koniec súboru,
- súborová premenná - typ `TextFile`, príkazy `AssignFile`, `Reset`, `Rewrite`, `WriteLn`, `ReadLn` a `CloseFile`.

Poznámky:

1. Budeme sa snažiť rozpoznávať rôzne úrovne náročnosti problémov a úloh, ktoré riešime so súbormi:

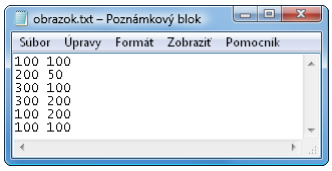


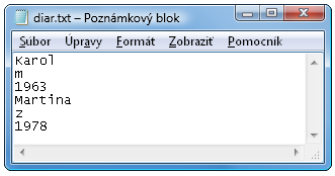
   I. Pomocou funkcií a metód (`LoadFromFile`, `SaveToFile`).

   II. Zápis a čítanie niekoľkých málo znakov a čísel do / z textového súboru. Zápis a čítanie konštantného počtu hodnôt do / z textového súboru.

   III. Práca s textovým súborom premenlivej dĺžky, čítanie do konca riadka, čítanie do konca súboru.

   IV. Nad rámec strednej školy sú témy, ako napríklad:
   - binárne súbory, práca s prúdmi (stream),
   - textové súbory so štruktúrou (`ini` súbory, `html` a pod.),
   - práca so súbormi, kde sa vyžadujú náročnejšie algoritmy (napríklad, lexikálna analýza, kompresia) alebo ukladanie rôznych štruktúr do súboru (objekty, stromy, grafy),
   - vyhľadávanie údajov v súbore (lineárne, binárne, hašovanie).





2. Súbory sú veľmi náročná téma preto, lebo:

   - Pri vytváraní a zápise do súboru žiaci neuvidia výsledok okamžite. To, čo program vykonával, sa dozvedia až po skončení programu, keď nájdu na disku počítača súbor a otvoria ho v textovom editore. Až potom sa presvedčia, že ich algoritmus správne pracuje.
   - Často riešime problémy typu „*Kam ste súbor uložili?*“. A naopak, ak si žiaci pripravia v textovom editore textový zlý súbor (napríklad taký, ktorý obsahuje iba 9 čísel namiesto 10, ktoré ich algoritmus očakával), program nebude správne pracovať. Pri čítaní zo súboru vznikne chyba a my musíme riešiť problémy typu „*Čo ste do súboru zadali?*“.
   - Pri programovaní treba dodržať schému: najskôr súbor otvoríme → niečo so súborom vykonáme → nakoniec súbor zatvoríme. Ak žiaci niektorý krok (zvyčajne prvý krok) vynechajú alebo ho spravia nesprávne (napríklad, uvedú zlé meno súboru), musíme hľadať problémy na viacerých miestach: v programe, na disku, alebo v samotnom súbore.
   - V porovnaní s tým, ako žiaci doposiaľ s údajmi pracovali, sú súbory výnimočné. Prečo? Pri prístupe k údajom v jednoduchých premenných alebo poliach boli žiaci naučený na to, že pamäť je prístupná okamžite. Napríklad, ak chceli nastaviť tretí prvok poľa, uviedli jeho index a priradili mu novú hodnotu. S textovými súbormi však pracujeme inak: ak chceme do súboru zapísať tretie číslo, musíme doň najskôr zapísať prvé dve čísla, až potom tretie. Znamená to, že k údajom pristupujeme postupne - od začiatku súboru. Takýto sekvenčný prístup k údajom je nová idea, na ktorú si musia žiaci zvyknúť.

3. Prvé programy musíme poctivo trasovať. Je veľmi dôležité, aby sme **obsah súboru** a **zmeny v súbore** priebežne kreslili na tabuľu tak, ako textový súbor uvidíme v textovom editore.

4. Neskôr, keď budeme spracovávať značky *koniec riadku* (`EOLN`), bude potrebné, aby sme znázorňovali obsah súboru tak, ako je súbor uložený na disku počítača – teda, ako **postupnosť znakov** a **značiek *koniec riadku***.

5. Veľmi záleží na tom, aké algoritmy pri práci so súbormi realizujeme. Ak budeme chcieť, aby žiaci riešili náročnejší algoritmus už na začiatku (napríklad, treba vyhľadať v súbore určité údaje), jeho náročnosť zakryje samotnú podstatu toho, ako so súbormi pracujeme.

Postup:

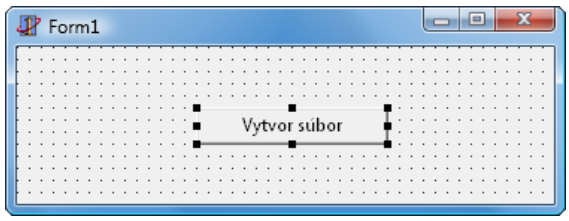
1. Motivácia pre vyučovanie textových súborov môže vyzerať napríklad takto: „*Na predchádzajúcej hodine sme vytvorili hru. Chceme, aby sa dosiahnuté skóre zapamätalo aj vtedy, keď počítač vypneme. Aké riešenie by ste navrhli?*“ ... „*Keby sme vedeli uložiť skóre do súboru, mohli by sme ho pri ďalšom spustení hry prečítať. Poďme sa preto pozrieť, ako pracujeme so súbormi*“.
2. Stručne a jednoducho vysvetlíme, čo chápeme pod pojmom **textový súbor**.
3. Najskôr sa učíme **vytvárať** textové súbory a **zapisovať** do nich údaje. V prvom príklade robíme iba drobné pokusy, na ktorých žiaci zbierajú elementárne skúsenosti s prácou so súbormi:
   - S textovými súbormi pracujeme prostredníctvom **súborovej premennej**, čiže premennej typu `TextFile`.
   - Povieme, ako príkazmi `AssignFile` a `Rewrite` **súbor otvoríme**. Vysvetlíme aj, čo „*otvorenie súboru*“ znamená a čo to spôsobí.
   - Príkazom `WriteLn` **zapíšeme** do textového súboru niekoľko konkrétnych údajov Zároveň na tabuli znázorňujeme, ako by sa náš textový súbor zobrazil v textovom editore.
   - Nakoniec povieme, ako príkazom `CloseFile` súbor **zatvoríme**. Vysvetlíme aj, prečo je potrebné súbor zatvoriť.
   - Po vykonaní programu bude potrebné skontrolovať, či na disku naozaj vznikol súbor s takým obsahom, aký sme chceli.
4. Na elementárnych úlohách trénujeme zápis do textového súboru, napríklad:
   - „Napíšte do textového súboru 100-krát ‚Mám rád programovanie‘.“
   - „Uložte do textového súboru čísla od `1` po `10`."
5. Potom učíme **čítať** údaje z textového súboru:
   - Bude potrebné, aby si žiaci sami vytvorili textový súbor v textovom editore (v programe *Notepad*, resp. *Poznámkový blok*) a uložili ho na správne miesto na disku počítača.
   - Pracujeme iba s malým textovým súborom, napríklad s takýmto:
     ```
     123
     ahoj
     45678
     ```
   - Ukážeme, ako súbor otvoríme príkazmi `AssignFile` a `Reset` na čítanie.
   - Pri **čítaní** údajov zo súboru postačí, ak zatiaľ žiakov naučíme používať príkaz `ReadLn`. Treba pamätať na to, že pri čítaní zo súboru musíme používať aj ďalšie premenné, do ktorých sa uložia načítané hodnoty.
   - Je potrebné trasovať a znázorňovať to, *ako* a *z ktorého* miesta zo súboru sa jednotlivé údaje načítavajú. Pomáhame si **metaforou *ukazovadla*** – pri čítaní sa pamätá miesto, kde sa v súbore práve nachádzame.
   - Nakoniec súbor zatvoríme.
6. Riešime elementárnych problémy, napríklad:
   - „V súbore máme 10 čísel. Prečítajte ich a vypíšte.“
   - „V textovom súbore je 10 čísel. Napíšte program, ktorý ich spočíta.“
   - „Vytvorte aplikáciu, ktorá počíta, koľko krát bola spustená.“

Najskôr učíme žiakov vytvárať a zapisovať do textových súborov, až potom ich učíme čítať a spracovávať údaje z textových súborov. Postupujeme tak preto, lebo zapisovanie do textového súboru je menej náchylné na chyby. Ak si uvedomíme, že pri čítaní textového súboru si musia žiaci v textovom editore najskôr nejaký súbor vytvoriť, musia ho uložiť na správne miesto a navyše zo súboru môžu čítať iba správne (číselné alebo textové) hodnoty, je nám jasné, že existuje väčšia šanca na to, aby ich programy nefungovali. Potom, namiesto toho, aby venovali energiu na porozumenie princípu fungovania textových súborov, míňajú energiu na hľadanie chyby. Preto učíme textové súbory čítať až po tom, ako žiaci začínajú tušiť, čo je vôbec textový súbor a už nadobudli základné skúsenosti s prácou s textovými súbormi.

motivácia

Ukážka:

1. Poďme sa pozrieť na to, ako pomocou príkazov v jazyku Pascal vytvoríme textový súbor. Textový súbor je taký súbor, ktorý vieme prečítať a editovať v hocijakom jednoduchom textovom editore, akým je napríklad aj *Notepad* (*Poznámkový blok*).
2. Viete, akú koncovku majú textové súbory? ... `.TXT`, `.DOC`, `.DOCX`...
3. My sa naučíme pracovať so súbormi, ktoré majú koncovku `.TXT`. Nebudeme teda používať súbory `DOC`, aké poznáme z programu Word. To preto, lebo také súbory majú príliš komplikovanú štruktúru. V nich sa okrem textu nachádzajú aj informácie o formátovaní textu, teda napríklad o tom, akým písmom je text napísaný, ako farbou alebo, či sa v texte nachádza aj nejaký obrázok a podobne.
4. Vytvorme aplikáciu, ktorá má iba jediné tlačidlo.



5. Po stlačení tlačidla vytvoríme textový súbor `pokus.txt`. Postupujeme takto:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  T: TextFile;
begin

end;
```

Deklarujeme súborovú premennú.

zbieranie skúseností

6. Zadeklarujeme premennú `T`, ktorá je typu `TextFile`. Typ `TextFile` je určený pre prácu so súborom a premennú `T` budeme zjednodušene nazývať *súborová premenná*. Táto premenná nám umožní pracovať s textovými súbormi.
7. Skôr, ako začneme do súboru zapisovať údaje, musíme vykonať príkazy, ktorými súbor otvoríme:

```
var
  T: TextFile;
begin
  AssignFile(T, 'pokus.txt');
  Rewrite(T);
```

8. Ako funguje otvorenie súboru:
   - Príkazom `AssignFile` prepojíme premennú `T` so súborom `pokus.txt`. Medzi premennou `T` a súborom `pokus.txt` vznikne asociácia, takže odteraz takmer všetky príkazy, ktoré vykonáme s premennou `T`, sa budú v skutočnosti diať so súborom `pokus.txt`.
   - Ako by sme preložili slovo `Rewrite`. ? ... *Prepíš*.
   - Príkaz `Rewrite` spôsobí, že na disku vznikne nový prázdny súbor `pokus.txt`. Ak už predtým na disku nejaký súbor `pokus.txt` existoval, prepíše sa našim novým a prázdnym súborom.
9. Teraz, keď je súbor otvorený, môžeme doň zapisovať údaje - čísla alebo texty. Robíme tak príkazom `WriteLn`:

```
  WriteLn(T, 'Ahoj');
```

10. Ako `WriteLn` funguje:
    - Meno príkazu `WriteLn` vzniklo zo slov *write line - zapíš riadok*.
    - `WriteLn(T, 'Ahoj')` zapíše do súboru riadok s textom `'Ahoj'`

pokus.txt

```
Ahoj
◀ sem zapíše WriteLn ďalší text
```

Predchádzajúci príkaz zapísal do súboru slovo Ahoj.

11. Vyskúšajme do súboru zapísať aj ďalšie údaje:

```
  WriteLn(T, 5*4);
  WriteLn(T, 'Koniec');
```

- `WriteLn(T, 5*4)` zapíše do druhého riadku číslo `20`.
- Do tretieho riadku sa napíše text `Koniec`.

pokus.txt

```
Ahoj
20
Koniec
```

Predchádzajúce tri príkazy `WriteLn` vytvorili takýto textový súbor.

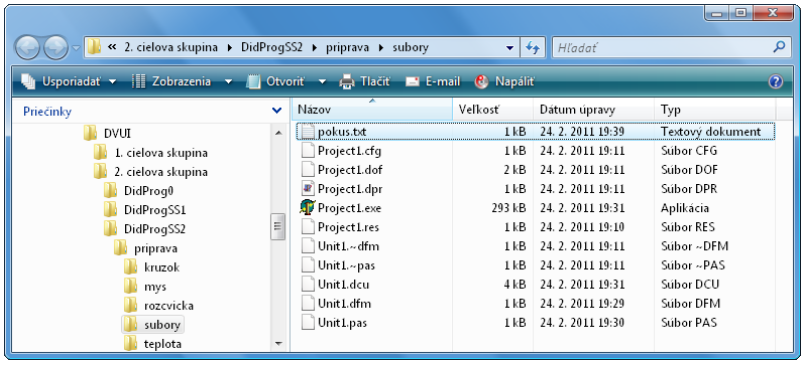
12. Nakoniec súbor zatvoríme:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  T: TextFile;
begin
  AssignFile(T, 'pokus.txt');
  Rewrite(T);
  WriteLn(T, 'Ahoj');
  WriteLn(T, 5*4);
  WriteLn(T, 'Koniec');
  CloseFile(T);
end;
```

zbieranie skúseností

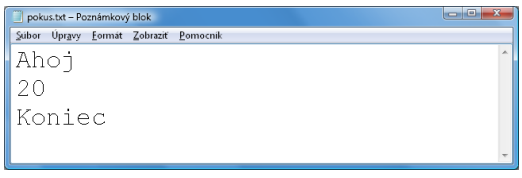
13. Príkazom `CloseFile` oznámime operačnému systému, že náš súbor je hotový, a že so súborom `pokus.txt` už nebudeme pracovať. Až potom môžu náš súbor bezpečne používať iné aplikácie.

14. Spustite program a vyskúšajte, čo sa stane, keď kliknete na tlačidlo `Button1`.



Ak sme postupovali správne, v okne aplikácie nič nové neuvidíme. Ale, na disku počítača nájdeme nový súbor `pokus.txt`.

15. Nájdite na disku súbor `pokus.txt` a presvedčte sa, že obsahuje texty, ktoré sme chceli do súboru zapísať. Súbor otvorte v programe *Notepad*.



Takto vyzerá text, ktorý vygeneroval náš program.

Kapitola 4: Hodnotenie a klasifikácia

V tejto kapitole sa venujeme hodnoteniu a klasifikácii z programovania. Najskôr si vyskúšame opravu krátkych testov – vybraných písomných prác, ktoré žiaci riešili. Potom sa pokúsime sami zostaviť pre našich žiakov podobné krátke testy, krátke písomné práce z programovania.

Oprava a hodnotenie

Na stranách 31 až 35 sme umiestnili tri testy – krátke 10 minútové písomné práce. Na začiatku je vždy uvedené zadanie testu, a potom nasleduje niekoľko vybraných žiackych riešení. Skôr, ako sa budeme venovať riešeniam žiakov, pozrime sa na zadania jednotlivých testov.

Aktivita 4.1	Každý test je zameraný na určitú tému z programovania. Jednotlivé zadanie si pozorne prečítajte a povedzte: • Na akú tému z programovania je test zameraný? • Aké programátorské kompetencie testujú jednotlivé úlohy?
Aktivita 4.2	Ako rýchlo by ste zvládli tieto úlohy vyriešiť?

Vidíme, že úlohy sú (z nášho pohľadu) veľmi jednoduché, neobsahujú žiadne zákerné otázky, ale testujú iba tie kompetencie, ktoré sme žiakov učili a so žiakmi trénovali.

Žiaci písali tieto testy v rôznych obdobiach školského roka. Pri vypracovávaní úlohy žiaci nemohli používať počítače, mohli však používať iba svoje vlastné poznámky. Samozrejme, úlohy sme zostavili tak, aby riešenia nenašli vo svojich zošitoch.

Zo všetkých riešení sme vybrali také, ktoré nás nútia zamyslieť sa nad ich hodnotením. Preto medzi ukážkami nájdeme zlé, menej správne alebo dobré, no zbytočne komplikované riešenia. Domnievame sa, že aj takú prácu treba hodnotiť „pozitívnou optikou“.

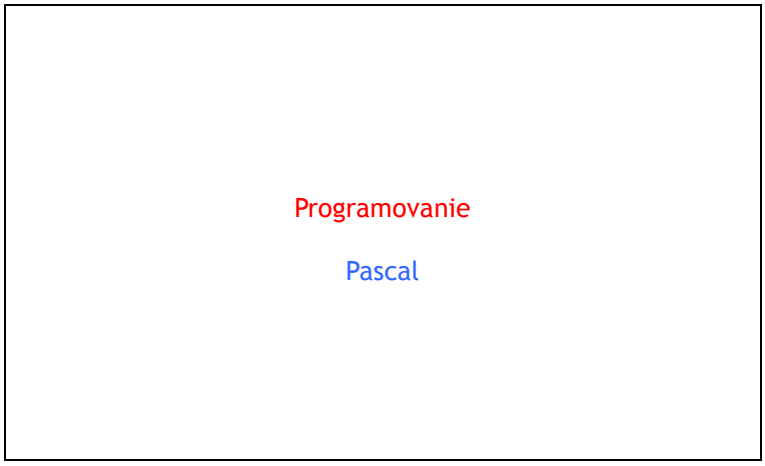
Aktivita 4.3	Opravte všetky žiacke riešenia. Prácu si môžete rozdeliť do skupín. Kvôli ďalšej aktivite však bude potrebné poznať názory viacerých ľudí. Preto potrebujeme, aby jeden príklad opravovali viacerí ľudia.

Pri opravovaní jednotlivých úloh pravdepodobne veľmi rýchlo narazíme na problém, aké hodnotenie žiakom udelíme.

Aktivita 4.4	Akú známku dáte žiakom za ich riešenie? Spoločne zostavte tabuľku, v ktorej uvidíte vaše hodnotenie, ako aj hodnotenie vašich kolegov. O rozdieloch diskutujte.

Test 1

Vo formulári máme vložený obrázok `Image1`, ktorý má rozmery 500 bodov na šírku a 300 bodov na výšku. Napíšte príkazy, ktoré písmom Arial veľkosti 20 vypíšu približne do stredu obrázka červený nápis *Programovanie* a pod to modrý nápis *Pascal* (súradnice odhadnite):



Riešenie 1:

```
Image1.Canvas.Font.Name=Arial;
Image1.Canvas.Font.Size:=20;
Image1.Canvas.Font.Color:=clRed;
Image1.Canvas.TextOut(299, 150, "Programovanie");
Image1.Canvas.Font.Color:=clBlue;
Image1.Canvas.TextOut(300, 151, "Pascal");
```

Riešenie 2:

```
Image1.Canvas.Font.Name='Arial';
Image1.Canvas.Font.Size:=20;
Image1.Canvas.Font.Color:=clRed;
Image1.Canvas.TextOut(200, 140, 'Programovanie');
Image1.Canvas.Font.Name='Arial';
Image1.Canvas.Font.Color:=clBlue;
Image1.Canvas.TextOut(220, 160, 'Pascal');
```

Test 2

Deklarujte 100 prvkové pole celých čísel. Potom napíšte príkazy, ktoré nastavia každý prvok poľa na náhodné číslo v rozsahu od 0 po 1000. Nakoniec zistite, koľko prvkov poľa má hodnotu menšiu, ako posledný prvok poľa.

Riešenie1:

```
var policko: array[0..99] of Integer;
var a, mensie: Integer;
begin
  mensie:=0;
  for a:=0 to 99 do policko[a]:=random(1001);
  for a:=0 to 99 do begin
    if (policko[a]<policko[99]) mensie:=mensie+1;
  end;
  Image1.Canvas.TextOut(0, 0, IntToStr(mensie));
end;
```

Riešenie2:

```
var i, c: Integer;
    cisla: array[1..100] of Integer;

c:=0;
for i:=1 to 100 do begin
  cisla[i]:=random(1000);
  if cisla[i]<cisla[100] then c:=c+1;
  c:=
  Image1.Canvas.TextOut(Image1.Width div 2, Image1.Height div 2, IntToStr(c));
```

Riešenie 3:

```
var a: array[1..100] of Integer;
    i, m, s: Integer;

 for i:=100 downto 1 do begin
   a[i]:=random(1000);
   m:=a[100];
   if a[i]<m then s:=s+1;
 end;
 Image1.Canvas.TextOut(20, 20, IntToStr(m));
```

Riešenie 4:

```
var a:array[1..100] of Integer;
    i: integer;

for i:=0 to 100 do begin
 a[i]:=random(1001);
 Image1.Canvas.TextOut(i*20, 0, IntToStr(a[i]));
 end;
if a[i]<100 then begin
  Image1.Canvas.TextOut(i*20, 0, );
end;
```

Test 3

Definujte funkciu `Sucin` s dvoma celočíselnými parametrami `a, b`. Výsledkom funkcie bude celé číslo rovné súčinu čísel od `a` po `b`, teda číslo rovné `a*(a+1)*...*b`. Napríklad, ak zavoláme funkciu `Sucin(4, 6)`, výsledkom bude číslo `120`. Nakoniec použite funkciu `Sucin` na to, aby ste vypísali súčin čísel od `5` po `13`.

Riešenie 1:

```
function Sucin(a: Integer; b: Integer): Integer;
begin
  for a:=n to b do n:=n*a;
  result:=n;
end;

... Button1Click
begin
  Form1.Image1.Canvas.TextOut(0, 0, IntToStr(Sucin(5, 13));
end;
```

Riešenie 2:

```
procedure Sucin(a, b: integer)
  var x: integer=1;
  for a:=a to b do begin
    x:=a*(a+1);
    a:=a+1;
  end;
  result:=a,b;

a:=StrToInt(Edit1.Text);
b:=StrToInt(Edit2.Text);
Image1.Canvas.TextOut(1, 2, IntToStr(Sucin()));
```

Riešenie 3:

```
function Sucin(a, b: integer): integer;
begin
  result:=a*b;
end;

procedure ... Click...
begin
  Image1.Canvas.TextOut(0, 0, IntToStr(Sucin(5, 13));
end;
```

Riešenie 4:

```
function sucin(a, b: integer): integer;
var i: integer;
begin
  i:=a;
  while i<=b do begin
    a=a*i;
    i:=i+1;
  end;
  result:=a;
end;

... Button1Click...
begin
  Image1.Canvas.TextOut(0, 0, IntToStr(sucin(5, 13));
end;
```

Tvorba úloh

V tomto aj predchádzajúcom materiáli sme sa venovali rôznym témam z programovania.

Aktivita 4.5	Aké formy skúšania, hodnotenia a klasifikácie používate? Diskutujte o ich výhodách a nevýhodách.

Pokúsme sa teraz navrhnúť krátke úlohy, ktoré by slúžili ako podklady pre hodnotenie žiakov.

Aktivita 4.6	Dohodnite sa na forme skúšania (napríklad, budete skúšať pri počítači alebo žiaci budú písať krátky písomný test). Potom si rozdeľte nasledujúce témy: • úvod do programovania a grafické príkazy, • premenné, • cyklus for, • podmienený cyklus while a jednoduché podmienky, • podmienený príkaz if, • procedúry, • funkcie, • myš, • polia. Pre svoju tému vymyslite zadanie krátkej úlohy, ktorú by mali vaši žiaci vyriešiť. Pri vymýšľaní úlohy si treba veľmi dobre uvedomiť, čo budete hodnotiť, čo považujete za dôležité a na čo budete pri hodnotení klásť dôraz. Pri vymýšľaní sa môžete inšpirovať aj našimi úlohami.

Všimnime si, ako vyzerajú naše úlohy z testov na strane 31 až 35:

- V úlohe vyžadujeme nájsť riešenie triviálneho problému.
- Na zápis riešenia stačí, ak žiaci používajú **iba známe**, **vyskúšané** a **natrénované** príkazy a konštrukcie.
- Pri riešení nevyžadujú ani encyklopedické znalosti toho, ako príkazy fungujú, ani objavenie žiadneho algoritmického triku, ani použite žiadnej novej syntaktickej konštrukcie.
- Zápis riešenia je krátky – má max. 10 riadkov.
- Držíme sa idey tzv. tretinového času: ak od žiakov očakávame, že úlohu vyriešia a napíšu za 10 minút, my sami by sme (ale aj naši kolegovia by) ju mali vyriešiť za 3 minúty.

Pravdepodobne, ak by sme mali inú skupinu žiakov a sledovali iné ciele, niektoré z uvedených bodov by sme mohli potlačiť (napríklad, na olympiáde z informatiky sa dokonca vyžaduje objavenie často aj veľmi prefíkaného algoritmu).

Aktivita 4.7	Prezentujte svoje zadanie a spôsob hodnotenia pred ostatnými kolegami.

Záver

V sérií materiálov s názvami Didaktika programovania, Didaktika programovania pre stredné školy 1 a Didaktika programovania pre stredné školy 2 sme sa venovali rôznym aspektom vyučovania programovania.

Po absolvovaní týchto materiálov by v nás mal zostať pocit, že vyučovanie programovania je veľmi zložitý proces, ktorý sa dá prirovnať k vyučovaniu matematiky. Priebeh a úspešnosť našej snahy ovplyvňuje veľmi veľa faktorov. Podobne, ako pri vyučovaní matematiky, aj pri vyučovaní informatiky sa musíme zamýšľať nad rôznymi variantmi, postupmi a detailmi.

Snáď aj toto sú dôvody, prečo je vyučovanie programovania medzi učiteľmi informatiky tak málo populárne. Treba si však uvedomiť, že postupne bude na stredné školy prichádzať tá generácia žiakov, ktorí sa naučili výborne pracovať s počítačom už na základnej škole. Takýto digitálne gramotní žiaci budú hravo využívať internet, posielať poštu, písať texty, kresliť obrázky alebo vytvárať prezentácie. Znamená to. že mnohé témy, ktoré boli dominantou stredoškolskej informatiky, prestanú byť pre týchto žiakov atraktívne. Pre učiteľa strednej školy tak vznikne vážny problém: *„aké aktivity robiť, aby žiakov informatika bavila?“* Domnievame sa, že programovanie môže poskytnúť nové témy, ktoré by oslovili žiakov a tí by nazbierali nové skúsenosti a zaujímavé zážitky.

Didaktiku programovania sme sa snažili koncipovať pragmaticky tak, aby tieto materiály boli prakticky užitočné pre učiteľa strednej školy, ktorý plánuje v rámci informatiky učiť programovanie. Považujeme za dôležité, aby sme:

- Porozumeli tomu, čo sa očakáva od programovania na strednej škole.
- Spoznali základné princípy, ako učiť rôzne témy z programovania.
- Sa zamýšľali nad vyučovaním z pohľadu poznávacieho procesu.
- Dokázali klasifikovať rôzne úrovne náročnosti príkladov, úloh a problémov.
- Pamätali na to, že žiaci môžu mať s porozumením problémy.
- Správne, pozitívne a férovo hodnotili našich žiakov.
- Mali informácie o existujúcich súťažiach pre žiakov.

Vidíme, že veľmi dôležitú úlohu pri vyučovaní programovania má samotný učiteľ. Dobrý učiteľ vie odhadnúť, čo žiakov motivuje, baví a zaujíma. Dokáže odpozorovať, čo žiaci momentálne zvládajú a vie pripraviť také jednoduché a názorné a zrozumiteľné príklady, pomocou ktorých naučí aj náročný koncept. Je si vedomí toho, že k všeobecnému vzorcu, zápisu alebo aj riešeniu vedie séria drobných krokov a úvah, ktoré musia žiaci vidieť, zažiť a natrénovať si. Inak budú chápať programovanie ako mágiu, ktorej nemajú šancu porozumieť. Zdá sa, že najväčším umením vyučovania je správne stanoviť hranicu, dokedy žiakov baví riešiť určité problémy, a kedy je čas posunúť sa ďalej tak, aby sa žiaci nenudili.

V týchto materiáloch sme sa prevažne venovali programovaniu v jazyku Pascal a prostrediam Delphi alebo Lazarus. Domnievame sa však, že mnohé didaktické princípy, ktoré sme v týchto materiáloch ukázali, sú aplikovateľné aj pri vyučovaní iných programovacích jazykov, napríklad C, Java, Python, a ďalších.

Pri ich písaní týchto materiálov sme vychádzali z problémov a pozorovaní, ktoré sme dlhé roky zbierali, zažívali a riešili pri vyučovaní programovania. Preto sme sa snažili v čo možno najväčšej miere písať tieto materiály tak, aby obsahovali také praktické rady, návody a odporúčania, ktoré budú užitočné pri reálnom vyučovaní programovania na hodinách informatiky.

Návody a riešenia

Aktivita 1.4:

Poradie úloh (nemusí byť jednoznačné) + čo je na každej úlohe najnáročnejšie:

- Úloha 2: Naprogramovanie procedúry s parametrami
- Úloha 1: Práca s ciframi a zostavenie podmienky, kedy je číslo magické.
- Úloha 4: Objavenie vzorca pre výpočet súradníc symetrických krúžkov.
- Úloha 3: Funkcia na zistenie prvočísla.
- Úloha 5: Počítanie ľudí v autobuse.

Aktivita 1.6:

- Úloha 1: Podmienený príkaz.
- Úloha 2: Procedúry s parametrami.
- Úloha 3: Funkcie.
- Úloha 4: Myš.
- Úloha 5: Polia.

Aktivita 4.1:

- Test 1: Úvod do programovania, farby, príkaz priradenia.
- Test 2: Polia.
- Test 3: Funkcie.

Aktivity 4.3 a 4.4:

Test1 – pri hodnotení je dôležité sledovať:

- poradie príkazov (napríklad, nastavenie farby pred výpisom),
- správne použitie príkazov (parametre, použitie priradenia),
- porozumenie, ako funguje súradnicová sústava – tolerujeme rozumný odhad súradníc.

Test 2 – pri hodnotení je dôležité sledovať:

- deklaráciu poľa (syntax, rozsah, typ prvkov poľa),
- cyklus pre nastavenie prvkov poľa,
- algoritmus pre zistenie počtu prvkov s danou vlastnosťou (cyklus, podmienka, počítanie).

Test 3 – pri hodnotení je dôležité sledovať:

- definíciu funkcie (syntax – hlavička, telo funkcie)
- algoritmus pre výpočet výsledku,
- nastavenie výsledku,
- správne použitie funkcie (volanie, hodnoty skutočných parametrov).

Pri opravovaní kladieme na syntax dôraz v závislosti od situácie. Napríklad, v prvom teste ešte môžeme sledovať písanie bodkočiarok, ale v ďalších testoch ich občasné vynechanie už nebudeme považovať za chybu. Neskôr, v náročnejších úlohách odpustíme aj také chyby, ktoré by nám v prvých testoch prekážali.

Aktivita 4.5:

Formy skúšania, napríklad:

- riešenie úlohy pri počítači,
- písomná práca – riešenie úlohy na papieri,
- test s otázkami s výberom odpovede a doplnením odpovede (podobné sme videli v maturitnom monitore),
- ústne skúšanie, skúšanie pri tabuli.

Čo sme sa naučili v tomto module

Zhrnutie

Spoznali a analyzovali sme odborné knihy a web stránky o programovaní. Videli sme, že sú písané veľmi odlišným štýlom, aký poznáme v učebniciach.

Analyzovali sme základné témy z programovania. Pre každú tému sme sa pokúsili navrhnúť vyučovaciu hodinu. Jednotlivé vyučovacie hodiny sme potom analyzovali a hodnotili nie len z odborného pohľadu, ale najmä z pohľadu didaktiky a poznávacieho procesu.

Nakoniec sme videli a analyzovali žiacke riešia. Pokúsili sme sa navrhnúť vlastné úlohu, ktoré by boli vhodné pre hodnotenia žiakov.

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] Bezáková D., Kučera, P., Lovászová G., Tomcsányi, P.: Programovanie 4 (Logo). Bratislava: ŠPÚ. ISBN 978-80-8118-017-0

[5] Blaho, A., Kalaš, I.: Tvorivá informatika. 1. zošit z programovania. Bratislava : SPN – Mladé letá, 2007. 48 s. ISBN 80-10-01223-7

[6] Bezáková D., Lovászová G., Kučera, P.: Programovanie 1. Bratislava: ŠPÚ. ISBN 978-80-89225-65-1

[7] Blaho, A. (2006) Informatika pre stredné školy. Programovanie v Delphi. Bratislava: SPN. ISBN 80-10-00421-9

[8] Blaho, A., Kubincová, Z., Salanci, Ľ. (2009) DVUI: Programovanie 2. Bratislava: ŠPÚ. ISBN 978-80-8118-007-1

[9] Blaho, A., Kubincová, Z., Salanci, Ľ. (2009) DVUI: Programovanie 3. Bratislava: ŠPÚ. ISBN 978-80-8118-014-9

[10] Blaho, A., Kubincová, Z., Salanci, Ľ. (2009) DVUI: Programovanie 4 (Pascal). Bratislava: ŠPÚ. ISBN 978-80-8118-018-7

[11] Hejný, M. a kol: Teória vyučovania matematiky. Bratislava: SPN 1990. ISBN 80-08-013443-3

[12] Hejny, M.: Understanding and structure, Proc. of CERME 3. Bellaria 2003. www.dm.unipi.it/~didattica/CERME3/proceedings/Groups/TG3/TG3_Hejny_cerme3.pdf

[13] 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 SŠ 2

Študijný materiál prešiel recenzným pokračovaním.

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

Počet strán 40

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 978-80-8118-090-3