Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

ŠTÁTNY PEDAGOGICKÝ ÚSTAV
NATIONAL INSTITUTE FOR EDUCATION

Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika

Didaktika programovania 2

Predmet: Didaktika programovania

Línia: Didaktika informatiky a informatickej výchovy

Operačný program VZDELÁVANIE

EURÓPSKA ÚNIA
Európsky sociálny fond

Európska únia
Európsky sociálny fond

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

Didaktika programovania 2

Identifikácia modulu

Aktivita projektu:	1.3 Ďalšie vzdelávanie kvalifikovaných učiteľov informatiky na 2. stupni ZŠ a na SŠ
Línia aktivity:	Didaktika informatiky a informatickej výchovy
Predmet:	Didaktika programovania

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, KAI FMFI UK, Bratislava

Zaradenie modulu

Modul Didaktika programovania 2 je druhým modulom, ktorý sa venuje téme, akým spôsobom učiť programovanie na druhom stupni základnej a na strednej škole.

Abstrakt modulu

Vyučovanie programovania je najdôležitejšou súčasťou informatiky. Je nevyhnutné, aby sa účastníci vzdelávania dozvedeli o postupnosti tém a o metodike toho, ako učiť programovať svojich žiakov na základnej alebo strednej škole. Tiež je dôležité, aby si aj prakticky vyskúšali navrhovať vyučovacie hodiny k jednotlivým témam.

Tento modul nadväzuje na modul Didaktika programovania 1, ktorý bol viac zameraný na vyučovanie programovania na základnej škole. Preto sa v tomto module budeme viac venovať vyučovaniu programovania na strednej škole. Budeme tiež analyzovať knihy a učebnice, pripravovať vyučovacie hodiny pre žiakov na strednej škole a otvoríme tému hodnotenia programovania. Tiež sa zoznámime s informatickými súťažami, ktoré sú vhodné pre žiakov na základných a stredných školách.

Obsah

Úvod

V tomto module sa venujeme didaktike programovania na základnej a strednej škole. Didaktikou programovania rozumieme vednú oblasť, v ktorej skúmame a hľadáme spôsoby, ako učiť algoritmy a programovanie. Snažíme sa porozumieť poznávaciemu procesu, tomu, ako vzniká nový poznatok, ale aj ťažkostiam, ktoré majú žiaci s porozumením algoritmov alebo s programovaním. Pritom chceme objavovať zákonitosti alebo pravidlá, ktoré by prispeli k lepšiemu vyučovaniu informatiky.

Pre tento modul bude potrebné, aby mali účastníci k dispozícii svoje počítače, a aby bola pre každú 5-6 člennú skupinu účastníkov vzdelávania dostupná učebnica Programovanie v Delphi [1], prípadne materiály Programovanie 2, 3 [7][8]. Výučba bude prebiehať v miestnosti s projektorom a s dostatočne veľkou tabuľou.

Cieľ modulu

Cieľom modulu je, aby účastník vzdelávania:

- Porozumel postaveniu, úlohe a cieľom programovania v informatike na základnej a strednej škole.
- Aby dokázal analyzovať tematické celky z programovania na strednej škole.
- Vedel navrhnúť a zostaviť vyučovacie hodiny pre programovanie na strednej škole.

Vstupné vedomosti

Požadované prerekvizity

Účastník vzdelávania musí mať absolvované všetky moduly týkajúce sa programovania, t.j. Programovanie 1 až 3, Programovanie 4 (Pascal) alebo Programovanie 4 (Imagine) a tiež aj predmet Didaktika programovania 1.

Predpokladané vstupné vedomosti, skúsenosti a zručnosti

Pozná aspoň tieto programovacie jazyky: Pascal (Delphi / Lazarus), Logo (Imagine).

Pozná a rozumie didaktike programovania na základnej škole.

Kapitola 1: Programovacie jazyky a prostredia (1h)

História programovacích jazykov siaha už do 50. rokov minulého storočia. V nasledujúcom zozname vidíme vybrané známe programovacie jazyky usporiadané podľa roku ich vzniku (resp. roku, kedy sa začali používať):

- 1954 – FORTRAN (FORmula TRANslator)
- 1958 – LISP (LISt Processor)
- 1958 - ALGOL (ALGOrithmic Language)
- 1959 – COBOL (COmmon Business Oriented Language)
- 1962 – APL
- 1964 – **BASIC** (Beginner's All-purpose Symbolic Instruction Code)
- 1968 – **Logo** (Logic Oriented Graphic Oriented)
- 1971 - **C**
- 1971 - **Pascal**
- 1972 - Smalltalk
- 1975 - Scheme
- 1978 - SQL
- 1983 - C++
- 1985 – Object Pascal
- 1987 - Perl
- 1991 - **Python**
- 1991 – Visual Basic
- 1993 - Ruby
- 1993 - Lua
- 1995 - Delphi
- 1995 - **Java**
- 1995 - JavaScript
- 1995 - **PHP**
- 2000 - **C#**

Programovacie jazyky nevznikli okamžite, ale od ich prvotného návrhu až po oficiálne uverejnenie prešlo niekoľko rokov. V určitých prípadoch boli už aj počas vývoja k dispozícii rôzne beta-verzie jazyka. Preto sa niekedy nedá presne povedať, kedy ten-ktorý programovací jazyk vznikol.

Niektoré jazyky „vznikli" aj tým, že sa premenovali už existujúce jazyky.

Existuje veľa dôvodov, prečo vzniklo toľko programovacích jazykov. Napríklad, prvé programovacie jazyky vznikali preto, aby programátori nemuseli programovať počítače v strojovom kóde alebo assembleri. Programovacie jazyky z posledného obdobia vznikajú ako reakcia na požiadavku vytvárať internetové aplikácie. Vďaka programovacím jazykom sú (by mali byť) programy pre človeka čitateľnejšie, zrozumiteľnejšie.

Assembler (alebo aj jazyk symbolických adries) je nízkoúrovňový jazyk, v ktorom pomocou slov zapisujeme inštrukcie procesora. Jednej inštrukcii v jazyku assembler zodpovedá jedna inštrukcia v strojovom kóde.

Takmer všetky uvedené jazyky sa v priebehu rokov vyvíjali. Dokonca sa vzájomne medzi sebou aj rôznym spôsobom ovplyvňovali. Súčasné verzie programovacích jazykov sú oproti pôvodným verziám značne zdokonalené. Napríklad, v 80. rokoch boli jazyky C a Pascal rozšírené o objektovo-orientované programovanie.

V 80. rokoch mala každá väčšia počítačová firma, ktorá chcela vo svete niečo znamenať, svoj programovací jazyk alebo svoj kompilátor.

Mnohé programovacie jazyky si však so sebou ťahajú aj nedostatky, ktoré vyplývajú z ich prapôvodnej koncepcie. Považujeme za **dôležité**, aby sa žiak – začiatočník učil programovanie vo vhodnom programovacom jazyku a programovacom prostredí. Nie je totiž jedno, koľko technických detailov a syntaktických pravidiel musí začiatočník prekonať, zvládnuť. Čím je ich viac a čím sú komplikovanejšie, tým sa programovanie stáva pre začiatočníka menej prístupné, prípadne až frustrujúce. Prečo je to tak?

Pokým sa skúsený vývojár trápi iba so syntaxou nového jazyka, chybovými hláseniami alebo používaním nového prostredia, musí sa začiatočník navyše trápiť aj s niečím oveľa náročnejším a pre informatiku dôležitejším. Tým je **porozumenie základným algoritmickým konceptom**: premenná, cyklus, vetvenie programu, pole, štruktúrovaná premenná, súbor, atď.

Aktivita 1.1	Preskúmajte rôzne programovacie jazyky a prostredia.

BASIC

Programovací jazyk BASIC vznikol v roku 1964 (John Kemeny and Thomas Kurtz). V tomto období ešte neexistovali osobné počítače. Minipočítače alebo sálové počítače sa používali na riešenie matematických problémov a na tento účel sa používal jazyk FORTRAN. Matematické úlohy a ani jazyk FORTRAN neboli pre každého ľahké. Preto vznikol programovací jazyk BASIC ako jazyk pre technické univerzity na prípravu pre programovanie v náročnejšom jazyku FORTRAN.

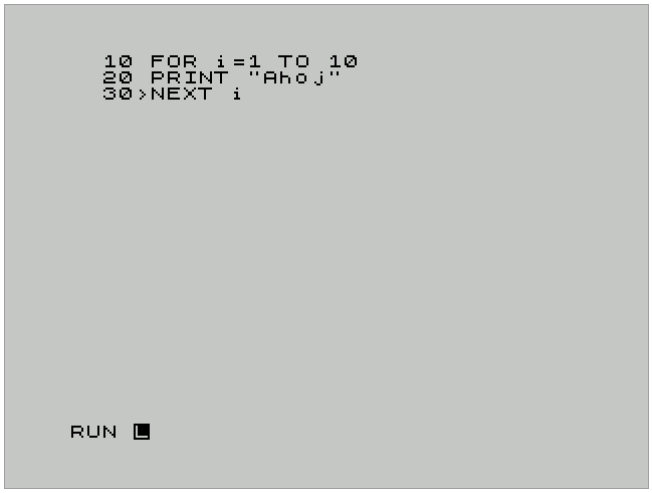
Popularita jazyka BASIC dosiahla vrchol v 80. rokoch v ére 8-bitových počítačov. V súčasnosti sa s týmto jazykom môžeme stretnúť v prostredí Visual BASIC alebo v balíku Microsoft Office, kde sa používa ako jazyk na tvorbu makier.

Ukážka programu v jazyku BASIC:

```
for i=1 to 10
  print "Ahoj"
next i
```

Keďže je tento jazyk veľmi jednoduchý, zdalo by sa, že je vhodný pre začiatočníkov. BASIC z 80. rokov však spôsobil veľa problémov, pretože vychoval veľa programátorov so zlými návykmi. Pôvodná verzia jazyka bola neštruktúrovaná a programátori museli využívať príkaz `GOTO` - skok v programe. Týmto príkazom sa realizovalo vetvenie programu alebo cykly. Intenzívne používanie tohto príkazu spôsobilo, že programy boli neprehľadné a veľmi ťažko čitateľné. Aj pri vzniku chyby sa ťažko dalo sledovať, odkiaľ nastal skok na chybný riadok.

Programátori, ktorí boli odchovaní na príkaze `GOTO`, mali veľké problémy (resp. sa nechceli skokov vzdať) pri prechode do štruktúrovaného programovacieho jazyka, napríklad Pascal, v ktorom už existovali konštrukcie **`if then else`** alebo **`while do`**. Jazyk BASIC tak *pokazil* niekoľko generácií programátorov.



Ukážka prostredia BASIC na počítači ZX Spectrum

Prostredie programu BASIC na 8-bitovom počítači bolo veľmi jednoduché. Na obrazovke sa zväčša zobrazovali riadky programu. Zapisovanie alebo úprava programu prebiehala v samostatnom dialógovom riadku, v ktorom sme mohli editovať iba jeden riadok z programu.

BASIC (neštruktúrovaný):

```
10 INPUT A
20 IF A<0 THEN GOTO 50
30 PRINT "zaporne"
40 GOTO 60
50 PRINT "kladne"
60 GOTO 10
```

Príkaz `GOTO` vykoná skok. Napríklad, `GOTO 10` skočí na riadok `10` (teda na začiatok programu).

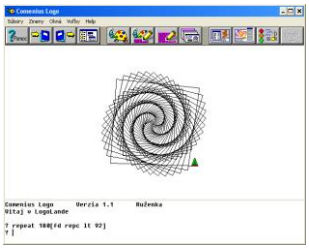
PASCAL:

```
A:=IntToStr(Edit.Text);
if A<0 then
 ShowMessage('záporné')
else
 ShowMessage('kladné');
```

Aj jazyk Pascal pozná príkaz GOTO. Avšak sa neodporúča používať (dokonca začiatočníkom jeho použitie v škole zakazujeme).

Logo

Programovací jazyk Logo vznikol v roku 1967 (Wally Feurzeig, Seymour Papert). Autori ho pôvodne navrhli pre účely hravého vyučovania matematiky, práce so slovami a vetami. Syntax jazyka Logo vychádza z funkcionálneho jazyka LISP. S jazykom Logo je spojená korytnačia grafika. Tá sa stala súčasťou jazyka Logo až neskôr, keď vznikla



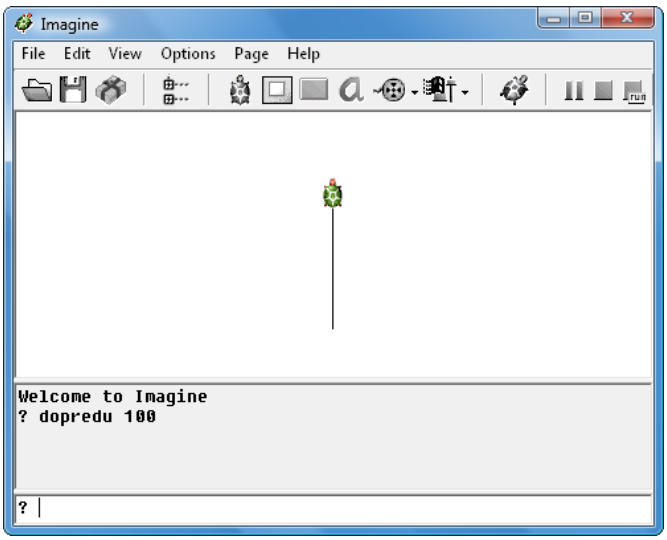
Prostredie Comenius Logo je už staršie a v porovnaní s prostredím Imagine vyzerá jednoduchšie. V prostredí Imagine však dokážeme ľahko vytvárať zábavnejšie projekty.

snaha riadiť reálny (korytnačí) robot. Postupne sa Logo stalo jazykom – prostredím, ktoré sprístupnilo počítače a programovanie deťom. Logo zažilo svoj vzostup najmä v 80. rokoch, keď sa osobné počítače stali dostupnejšie.

Ukážka programu v jazyku Logo:

```
repeat 10 [print "Ahoj]
```

Tento jazyk sa v súčasnosti úspešne používa pri vyučovaní programovania. Z pohľadu didaktiky je nesmierne cenná najmä **jednoduchosť** a **interaktívnosť** programovacieho prostredia. Znamená to napríklad, že príkaz `dopredu 100`, ktorý zadáme do príkazového riadku, sa okamžite vykoná a na obrazovke vidíme výsledok – **kresbu**.



Ukážka prostredia IMAGINE Logo

Pokým nepotrebujeme vytvárať príliš zložité projekty, má programovací jazyk pre začiatočníka jednoduchú syntax, jednoduché pravidlá, ktoré sa dajú vysvetliť priamo alebo prostredníctvom prístupných metafor (napríklad, korytnačka ako metafora pre grafické pero). Cenné sú aj zmysluplné chybové hlásenia. Nevýhodou jazyka Logo je, že už v mierne zložitejších situáciách si musíme dávať pozor na vyhodnocovanie výrazov. Napríklad, pokým výrazy neumiestnime do zátvoriek, vyhodnocujú sa inak, ako by hovorila intuícia: `zoznam 1` alebo `zoznam 1 2 3` sú zlé zápisy, správne sú `(zoznam 1)` alebo `(zoznam 1 2 3)`.

C a C++

Keďže každý typ procesora mal svoj vlastný jazyk assembler, programy napísané v assembleri boli väčšinou neprenositeľné na iné typy počítačov. Jazyk C vznikol s cieľom vytvoriť „prenositeľný assembler“.

Prvotný vývoj jazyka C začal v roku 1969 a prebiehal až do roku 1973 (Dennis Ritchie, Bellove laboratória). Programovací jazyk C je spätý s vývojom operačného systému Unix. Jazyk C vznikol ako náhrada za assembler, v ktorom sa dovtedy operačný systém Unix programoval. Tým, že sa do jazyka C postupne dodali štruktúrované údajové typy, stal sa jazyk C dostatočne silný na to, aby sa v ňom dali efektívne programovať systémové aplikácie (alebo aj jadro operačného systému).

Ukážka programu v jazyku C++:

```
#include <iostream.h>

void main()
{
    for (int i=1; i<=10; i++)
        cout << "Ahoj" << endl;
}
```

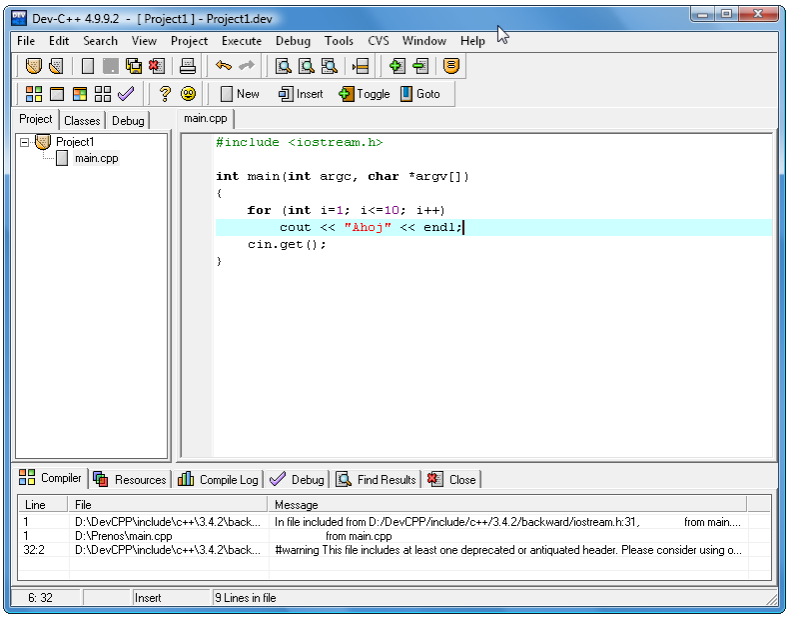
Začiatkom 80. rokov sa začína hlásiť k slovu objektovo-orientované programovanie. Jazyk C (podobne, ako iné programovacie jazyky, napríklad ani Pascal) neobsahoval podporu pre objektovo-orientované programovanie. Vývoj objektového jazyka C začína v roku 1979 a v roku 1983 sa objavuje prvýkrát pod názvom C++.

Jazyk C sa stal populárny najmä vďaka svojej vyjadrovacej sile, krátkym a strohým zápisom a množstvu knižníc s funkciami, ktoré pre tento jazyk existujú. Jazyk C ovplyvnil vývoj aj mnohých iných programovacích jazykov (napríklad Java, C#).

Pre programovanie v jazyku C++ existuje veľa prostredí. Napríklad Microsoft Visual C, C++ Builder, Dev-C++ a ďalšie.

Jazyky C a C++ sú aj v súčasnosti medzi mnohými vývojármi veľmi populárne.

Jazyky C a ich klony sa používajú v rôznych situáciách. Okrem programovania operačných systémov a aplikácií sa napríklad používajú aj pri programovaní elektronických súčiastok (jednočipových počítačov).



Prostredie Dev-C++

S vyučovaním jazyka C++ je spojených niekoľko nepríjemných problémov:

- Veľmi zjednodušene sa dá povedať, že v jazyku C++ má takmer každý zápis nejaký význam. Napríklad, `"Sucet je" + Sucet` nezlepí dva reťazce (teda výsledkom nebude reťazec `"Sucet je 123"`), ale sčíta smerníky. Výsledok sa prejaví chybným výpisom na obrazovke alebo pádom programu. Pri vyučovaní programovania nastáva veľa podobných situácií. Aby sme dokázali žiakom pomôcť, musíme porozumieť mnohým podobným „trikovým“ zápisom a komplikovaným syntaktickým pravidlám.

- Kompilátor jazyka C++ málo kontroluje naše chyby. Napríklad nekontroluje rozsah hodnôt (tzv. range checking, ktorý poznáme z jazyka Pascal). Preto nám kompilátor dovolí indexovať neexistujúce prvky poľa `A[-1]=123`. Nekontroluje ani pretečenie aritmetiky (sčítanie dvoch veľkých čísel). Takéto chyby v programe sa veľmi ťažko hľadajú.

- Mnohé chybové hlásenia kompilátora nie sú zrozumiteľné.

Z predchádzajúceho vyplýva, že vyučovanie jazyka C a C++ kladie na učiteľa veľmi veľké nároky.

PASCAL

Programovací jazyk Pascal vznikal v rokoch 1968-9 a verejnosti bol sprístupnený v roku 1970 (Niklaus Wirth). Jazyk bol od počiatku navrhovaný ako malý, ale pritom silný a efektívny programovací jazyk, ktorý by bol vhodný predovšetkým (avšak nie

iba) pre vyučovanie programovania, pre získanie dobrých programátorských návykov a získavaní skúseností pri práci s údajovými štruktúrami.

Ukážka programu v jazyku Pascal:

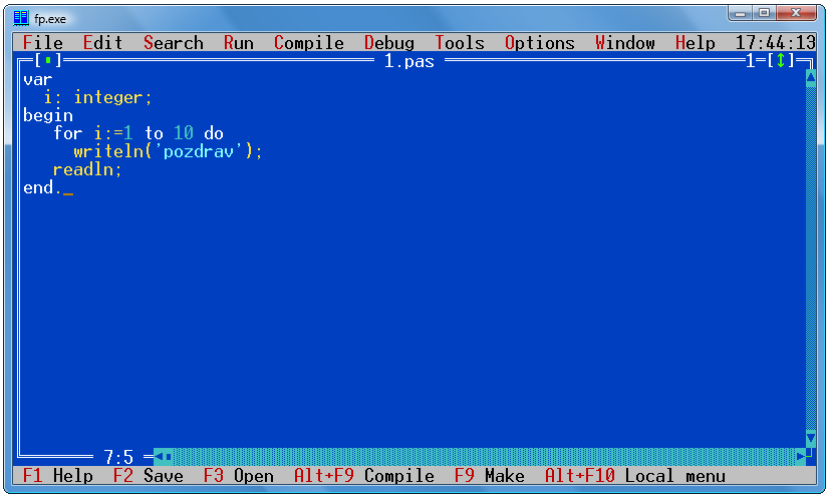
```
var i: integer;

begin
  for i:=1 to 10 do
    writeln('Ahoj');
end.
```

Firma Borland stála za vývojom obľúbeného prostrediaTurboPascal.

V 80. a 90. rokoch sa tieto prostredia označovali skratkou IDE (Integrated Development Environment = integrované vývojové prostredie). IDE prostredie v sebe obsahovalo editor programu, kompilátor a nástroje na trasovanie a ladenie programov.

V 90. rokoch, s nástupom prostredia Delphi, sa pre vývojové prostredie používalo označenie RAD (Rapid Application Development = rýchly vývoj aplikácie). To preto, lebo pomocou komponentov mohli programátori rýchlo zostaviť vzhľad aplikácie.

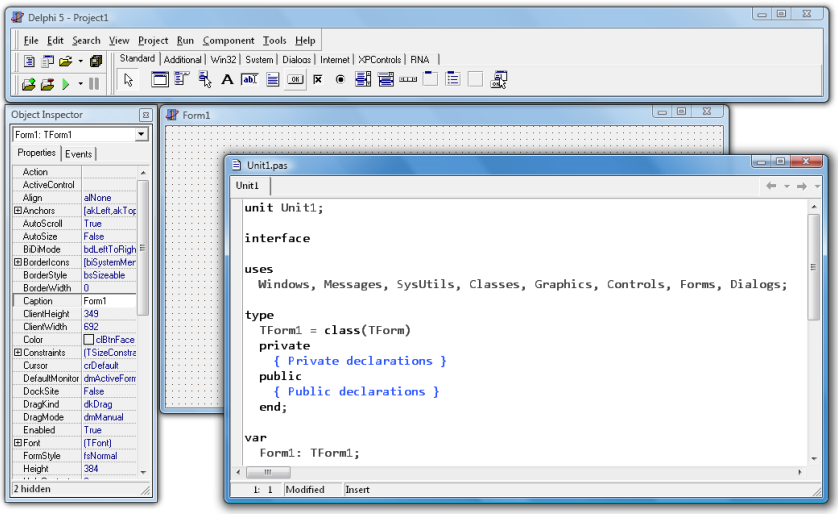
Objekty boli do jazyka Pascal zavedené až v roku 1986, čím vznikol jazyk Object Pascal. V tomto období boli mnohé vlastnosti objektovo orientovaného programovania dostatočne preskúmané. Preto sú určité vlastnosti objektovo orientovaného programovania implementované v jazyku Pascal premyslenejšie ako v jazyku C++.



Prostredie programu FreePascal vychádza z legendárneho a obľúbeného vývojového prostredia TurboPascal.

V priebehu 90. rokov nastáva éra Windows, mení sa spôsob práce s počítačom a jeho aplikáciami. Doposiaľ bežala väčšina aplikácií v textovom režime. Nové aplikácie však ponúkajú grafické používateľské rozhranie. Preto aj tvorcovia oknových aplikácií musia využiť iné stratégie programovania. V roku 1995 vzniká prostredie Delphi, ktoré pomocou komponentov umožnilo elegantne vytvárať plnohodnotné oknové aplikácie.

Jazyk Pascal ovplyvnil vývoj jazykov Java aj C#.



Prostredie Delphi

Z didaktického hľadiska ostáva jazyk Pascal stále vhodným kandidátom na vyučovanie programovania. Existuje k nemu veľa metodických materiálov a postupov, nezaťažuje žiakov-začiatočníkov mnohými technickými podrobnosťami ani komplikovanými syntaktickými pravidlami. Pre začiatočníkov môžu byť problémom súčasné prostredia Delphi alebo Lazarus, ktoré, pokým nenakonfigurujeme, vyzerajú pomerne zložito. Niektorí pokročilí žiaci jazyk Pascal nemajú radi. Zdá sa im príliš rozvláčny a nie až tak trendový, ako céčkové programovacie jazyky.

Jazyk FreePascal a prostredie Lazarus sa veľmi úspešne používajú aj na celosvetovej súťaži v programovaní žiakov stredných škôl – na medzinárodnej olympiáde z informatiky. V súťaži si žiaci vyberajú jeden z jazykov Pascal, C a C++.

JAVA

V roku 1995 vznikla prvá oficiálna verzia programovacieho jazyka Java (James Gosling, Sun Microsystems). Začiatok vývoja tohto jazyka však siaha až do roku 1991. Prvotným úmyslom bolo vytvoriť jazyk, ktorým by sa programovali rôzne elektronické zariadenia. Aj vďaka prenositeľnosti aplikácií medzi rôznymi počítačovými platformami a rastúcej popularite internetu sa jazyk Java stal veľmi rozšírený a obľúbený najmä v počítačovom priemysle.

Syntax jazyka Java vychádza z céčkových programovacích jazykov. Jazyk Java je oproti jazyku C++ v mnohom vylepšený a niektoré nedostatky jazyka C++ sú v jazyku Java opravené. V porovnaní s jazykom C++ je jazyk Java silne objektový.

Ukážka programu v jazyku Java:

```
public class pozdrav {
    public static void main(String args[])
    {
        for (int i=1; i<=10; i++)
            System.out.println("Ahoj");
    }
}
```

Z pohľadu didaktiky je Java do značnej miery rozporuplný jazyk. Keďže Java vychádza z jazyka C++, je prvým zdrojom problémov náročná syntax samotného jazyka. Ďalšou nepríjemnou vlastnosťou je prílišná „objektovosť“, z ktorej vyplývajú zložité zápisy pri manipulácii s objektmi. Aj mnohé technické detaily kladú prekážky porozumeniu základným konceptom. Príkladom môže byť ošetrenie výnimiek pri práci so súbormi, ťažkopádna tvorba grafických aplikácií, aplikácií s používateľskými rozhraním alebo nešikovná práca s reťazcami.

Musíme si uvedomiť, že programovací jazyk Java bol pôvodne navrhovaný pre počítačový priemysel, pre profesionálne tímy programátorov. Nebol navrhovaný pre školské prostredie ani pre vyučovanie programovania. Dobré vlastnosti jazyka Java tak majú šancu oceniť najmä skúsení programátori, ktorí sú už zvyknutí na komplikované syntaktické zápisy, technológie a paradigmy. Navyše, momentálne chýba aj vhodná a overená metodika pre vyučovanie začiatočníkov v jazyku Java. Preto sa domnievame, že pre neskúseného začiatočníka je momentálne Java veľmi náročným úvodom do programovania.

Aktivita 1.1

Ktoré vlastnosti programovacích jazykov alebo prostredí sú pre vás dôležité?

Môžete si napríklad všímať kritériá:

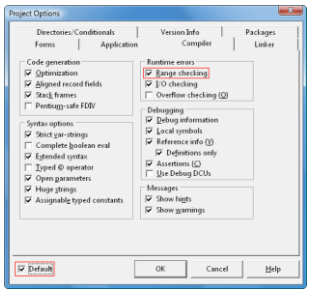
- syntaktické pravidlá
- zmysluplnosť chybových hlásení
- podpora grafiky
- rýchlosť
- ...

Ktoré z nich považujete za dôležité pre profesionálnu tvorbu programov a ktoré pre vyučovanie programovania?

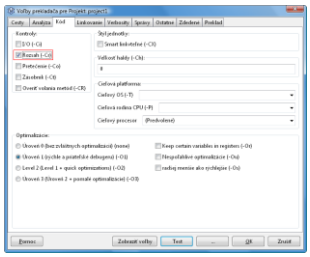
Aktivita 1.2	Porovnajte programovacie jazyky z hľadiska ich vhodnosti pre vyučovanie programovania: Zostavte tabuľku, v ktorej • jeden rozmer tvoria vami stanovené kritériá pre vyučovanie z predchádzajúcej aktivity, • druhý rozmer tvoria vám známe programovacie jazyky alebo prostredia. Tabuľku vyplňte. Ktorý jazyk alebo prostredie spĺňa najviac kritérií a ktorý najmenej?

Zhrnutie

Pre vyučovanie je dôležite poznať výhody a nevýhody programovacích jazykov aj vývojových prostredí. Ak sme si vedomí alebo, ak vieme odhadnúť, aké problémy nás očakávajú, môžeme lepšie uvažovať o ich použití pri vyučovaní programovania v škole.



Kontrola rozsahov v Delphi (obrázok hore) alebo Lazarus (obrázok dole).



Pre začiatočníka, je zapnutá kontrola rozsahov dôležitou pomôckou, ktorá výrazne pomáha pri hľadaní chýb v programoch. Preto túto voľbu nechávame pri vyučovaní zapnutú aj na žiackych počítačoch.

Mnohé vývojové prostredia vyzerajú po inštalácii komplikovane. Zobrazujú mnoho okien, líšt, príkazov. Pritom mnohé z nich žiaci nebudú používať. Preto zvykneme po inštalácii prostredie upraviť tým, že poskrývame pre žiakov nepotrebné veci. Žiaci potom pracujú v takom „upratanom“ prostredí.

Syntax jazyka určuje, do akej miery sú pravidlá zápisov programov komplikované (v C++ sú značne komplikované), ako rýchlo vieme napísať jednoduché programy, a koľko vecí musíme pritom „zatajiť“. Napríklad:

- Jazyk C++ má veľmi zložité pravidlá a porozumieť niektorým zápisom alebo správaniu sa programu v istých situáciách vyžaduje poznať mnohé „triky“, resp. detaily.
- Aj v jazyku Logo, keď začneme vytvárať zložitejšie výrazy, môžeme mať problém s porozumením zápisov.

Programovacie jazyky majú aj rôzne vylepšenia. Napríklad:

- Kompilátor jazyka Pascal nás môže kontrolovať, či používame správne indexy pri práci s prvkami poľa (tzv. rangechecking – kontrola rozsahov).
- V niektorých jazykoch C++ (napríklad, v prostredí C++Buidler) elegantne fungujú automatické konverzie. Napríklad, pri výpise číselných výsledkov v príkaze `TextOut` nemusíme používať `IntToStr`.

Mnohé súčasné programovacie jazyky sú **kompilované**. Pri tvorbe programov je dôležité, ako nás kompilátor informuje o chybách v programe. Napríklad, chybové hlásenia mnohých kompilátorov jazykov C++ a Java sú ťažko pochopiteľné (pre začiatočníka nezrozumiteľné).

Rovnako dôležitý význam, ako programovací jazyk, má aj programovacie **prostredie**. Napríklad:

- Výhodou prostredia Imagine je to, že žiaci pracujú priamo s grafikou. Nakreslený obrázok je pre začiatočníka motivujúcejší ako vypísané texty alebo čísla.
- Naopak, korytnačia grafika je spočiatku komplikovanejšia, pretože žiak musí pracovať s uhlami a predstavovať si, ako riadi korytnačku relatívnymi príkazmi pre otáčanie vľavo alebo vpravo.

Môžeme tiež pozorovať, že programovacie prostredia sa postupne vyvíjali a komplikovali.

- Všimnime si jednoduché prostredia, z éry 8-bitových počítačov. Často však majú primitívne možnosti editovania programov.

- Prehľadné prostredie TurboPascal z 80. až 90. rokov. Nevýhodou je práca v textovom režime – grafika v prostredí DOS je zastaraná, použitie myši problematické.

- Súčasné profesionálne prostredia (Delphi, Lazarus, C++Builder, Visual C++,...) sú komplexné a zložité. Poskytujú však komfortný zápis a ladenie programov.

Domnievame sa, že výber programovacieho jazyka a vývojového prostredia je veľmi dôležitý pre žiaka – začiatočníka. Dôležitý je aj pre učiteľa. Zlý výber znamená, že sa natrápia nielen žiaci, ale veľmi sa bude trápiť aj učiteľ:

- Jazyky C++ alebo Java sú veľmi náročné na zvládnutie. Obsahujú priveľa zložitých syntaktických pravidiel, technických detailov, „magických“ zápisov, ktoré sú poskladané z rôznych symbolov. Žiaci teda budú musieť okrem – pre začiatočníka – náročných algoritmických konceptov, zvládnuť aj spomínané komplikácie s jazykom.

- Žiaci ľahko napíšu zlý kód. Napríklad, v jazyku C++ sa začiatočníci často mýlia a namiesto porovnania (`a==0`) píšu v podmienenom príkaze priradenie (`a=1`). Alebo, namiesto zreťazenia sčítajú smerníky (`"Vysledok="+1000*500`). Napriek tomu kompilátor preloží takýto zlý program, ktorý ale nebude správne pracovať, bude sa správať podivne, bude padať na čudných miestach. Väčšina začiatočníkov nemá šancu takéto chyby nájsť a opraviť ich. Učiteľ im bude musieť pomáhať a vysvetľovať chyby (čo je v istých situáciách problematické).

- Niektoré kľúčové programátorské koncepty sa v niektorých (najmä profesionálnych) jazykoch realizujú veľmi komplikovane. Vyučovanie úvodných programátorských tém (napr. premenná, cyklus, pole, grafika) musí byť potom veľmi často doplnené o netriviálne technické detaily (napr. statické identifikátory, výnimky, dynamicky vytvárané polia).

Aby dokázal učiteľ žiakom pomôcť, bude musieť spoznať veľa rôznych pascí, zlých situácií a nesprávnych kombinácií, ktoré sú často dôsledkom syntaxe jazyka. Tiež si treba uvedomiť, že pri voľbe nevhodného jazyka alebo prostredia nie je jednoduché venovať sa naraz 15 žiakom, zvládnuť hľadanie, opravu a prípadne aj vysvetľovanie chýb. Vyučovanie v žiadnom prípade nemôže vyzerať tak, že v triede je 15 žiakov, učiteľ sa ešte dvom venuje, pretože tí ho ešte vnímajú, prípadne s ním súhlasia a ostatní žiaci netušia, o čom učiteľ rozpráva.

Vidíme teda, že v školskom prostredí sú do značnej miery dôležité iné kritériá, ako v profesionálnych programátorských tímoch, ktoré pracujú na veľkých projektoch.

Aktivita 1.3	Uvažujte a diskutujte nad tým, aká je úloha programovania v škole. Do akej miery má a dokáže škola reagovať na požiadavky softvérového priemyslu?

Kapitola 2: Učebnice a knihy

V tejto kapitole preštudujeme učebnice, knihy a webové stránky, ktoré sa zaoberajú programovaním.

Aktivita 2.1	Ktoré učebnice, knihy alebo webové stránky so zameraním na programovanie poznáte a odporučili by ste ich svojim kolegom?
Aktivita 2.2	Zostavte 5 až 6 členné tímy. Každý tím dostane učebnicu alebo knihu. Preštudujte ju a pokúste sa zodpovedať na nasledujúce otázky: • Aký programovací jazyk sa používa? • Aké prostredie používajú autori pri vysvetľovaní? • Aké témy a v akom poradí sa preberajú? Stručne ich vymenujte. • Aký je váš dojem? Na túto aktivitu nadväzuje nasledujúca aktivita.
Aktivita 2.3	Teraz sa zamerajte iba na jednu kapitolu a analyzujte ju. Povedzte: • Aká téma je v kapitole spracovaná? • Ako nás autor motivuje? • Ako autor postupuje pri vysvetľovaní? • Aké typy úloh rieši? • Aký je váš názor na postup, ktorý zvolil autor? Každý tím si vyberie jedného člena, ktorý prerozpráva a prezentuje ostatným výsledok práce svojho tímu.

Súčasné učebnice programovania

Učebnice sú dôležitou súčasťou vyučovania. V súčasnosti sú v školách k dispozícii učebnice:

- Tvorivá informatika – 1. zošit z programovania [2]
- Programovanie v Delphi [1]
- Algoritmy s Logom [3]
- Algoritmy s Pascalom [4]

Tematické zošity Algoritmy s Logom a Algoritmy s Pascalom sú už pomerne staré v porovnaní s učebnicou Programovanie v Delphi (vyšla v roku 2006) alebo tematickým zošit Tvorivá informatika – 1. zošit z programovania (vyšla v roku 2005).

Pre úplnosť je potrebné povedať, že aj základná stredoškolská učebnica Informatika pre stredné školy obsahuje kapitolu o algoritmoch a programovaní s názvom Algoritmy a algoritmizácia.

Tvorivá informatika – 1. zošit z programovania

Tento tematický zošit je určený pre prímu až kvartu osemročných gymnázií a 2. stupeň základných škôl. Obsahuje niekoľko kapitol, v ktorých sa postupne vysvetľuje programovanie v jazyku Logo a prostredí Imagine. Nájdeme tu spracované nasledujúce témy:

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

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

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

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

Programovanie v Delphi

Programovanie v Delphi má formu učebnice.

Stručný obsah učebnice vyzerá takto:

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

Na učebnici je cenný jej prístup k vyučovaniu programovania. Vidíme, že úvod do programovania v jazyku Pascal je postavený na grafických príkladoch. Väčšina iných kníh, aj zahraničných učebníc o programovaní, sa totiž pridržiava tradičného postupu. V nich od začiatku prevláda textový (konzolový) režim a riešenie matematický úloh.

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

Algoritmy s Logom

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

Tematický zošiť obsahuje témy:

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

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

Algoritmy s Pascalom

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

Obsahuje nasledujúce témy:

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

Aj keď sa v učebnici pracuje v už neexistujúcom prostredí TurboPascal, vidíme snahu autorov o to, aby sa už od začiatku využívala grafika. Znamená to, že už základné programové konštrukcie sa vysvetľujú na príkladoch, ktoré niečo kreslia, alebo študenti riešia úlohy, v ktorých treba niečo nakresliť.

Odborná literatúra

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

V niektorých prípadoch dokonca autori utvrdzujú čitateľa, že si kúpil správnu knihu, pretože, podľa autora, je kniha aj pre začiatočníka aj pre skúseného čitateľa. Z kníh, ktoré sme analyzovali však tento slogan znamenal, že kniha pre začiatočníka nie je.

Väčšina odbornej literatúry má nasledovnú štruktúru:

- buď hovoria o typoch, syntaxi a konštrukciách programovacieho jazyka,
- alebo sa zameriavajú na vysvetľovanie komponentov, knižníc a funkcií,
- alebo kombinujú obidva prístupy – začínajú jednoduchým príkladom (zobrazením správy „Hello Word!“), ihneď na to pokračujú vysvetľovaním komponentov, programovacích techník vo Windows alebo tvorby 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.

Webové stránky

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

Aktivita 2.2	Aké webové stránky o programovaní radi používate?
Aktivita 2.3	Ako nás motivuje autor webových stránok pri jednotlivých témach?

Mnohé webové stránky o programovaní sú z pohľadu didaktiky problematické. Najčastejšie pozorujeme nasledovné problémy:

- Stránky sú zamerané na použitie tej-ktorej technológie alebo na syntax programovacieho jazyka. Znamená to, že na stránkach nájdeme prvý príklad „Hello world“, potom uvidíme zoznam typov, programových konštrukcií, komponentov, zoznam knižníc a ich funkcií. Pri vyučovaní však sledujeme iné ciele – rozvoj algoritmického myslenia objavovanie postupov na riešenie problémov. Na takýchto stránkach však chýba vysvetlenie základných algoritmických koncepcií.

- Nie zriedka nájdeme stránky, ktoré do detailov vysvetľujú jednoduché veci, avšak náročné a ťažké témy sú popísané viac ako stručne. Napríklad, do detailov nájdeme ilustrovaný popis prostredia alebo postupu pri uložení projektu. Premennú však nájdeme vysvetlenú jednou vetou. Zo skúseností však vieme, že samotná téma premennej je pre začiatočníka ťažká. Ďalej však nasledujú v rýchlom slede komplikované témy, príklady s tým, že v niektorých z nich sa ešte aj využívajú rôzne programátorské triky.

Vidíme, že takéto webové stránky obsahujú veľa technických detailov, ktoré sú pre začiatočníka zbytočné. Naopak, chýbajú viaceré etapy poznávacieho procesu. Je to preto, lebo ich tvorcovia nemajú pedagogické skúsenosti.

Na webových stránkach môžeme nájsť veľa pozitívnych inšpirácií. Pritom však musíme byť veľmi obozretní, kritickí a musíme dobre zvažovať, ako ich zakomponujeme do vyučovania.

Zhrnutie

Písaná literatúra vyzerá vo všeobecnosti inak, ako zvykneme postupovať na vyučovacej hodine. Navyše, aj učebnice a tematické zošity majú od odbornej literatúry a webových stránok odlišný charakter. 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.

Správne učebnice kladú dôraz na skúsenosti žiakov:

- Obsahujú iba nevyhnutne potrebné množstvo teórie, dôraz sa kladie na príklady, ilustrácie, úlohy na precvičenie.
- Postupnosť tém: zoznámenie sa s prostredím, postupne sa učia a precvičujú základné algoritmické konštrukcie.
- Žiaci sa postupne učia riešiť (elementárne) algoritmické problémy.

Kapitola 3: Poradie tém

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

- V prostredí Imagine: ovládanie korytnačky – kreslenie – cyklus – udalosti ...
- V prostredí Delphi: kreslenie – premenné – cyklus – podmienený príkaz ...

Aktivita 3.1	Povedzte, aké témy z programovania a v akom poradí učíte alebo by ste učili svojich žiakov. Nakreslite strom možností.

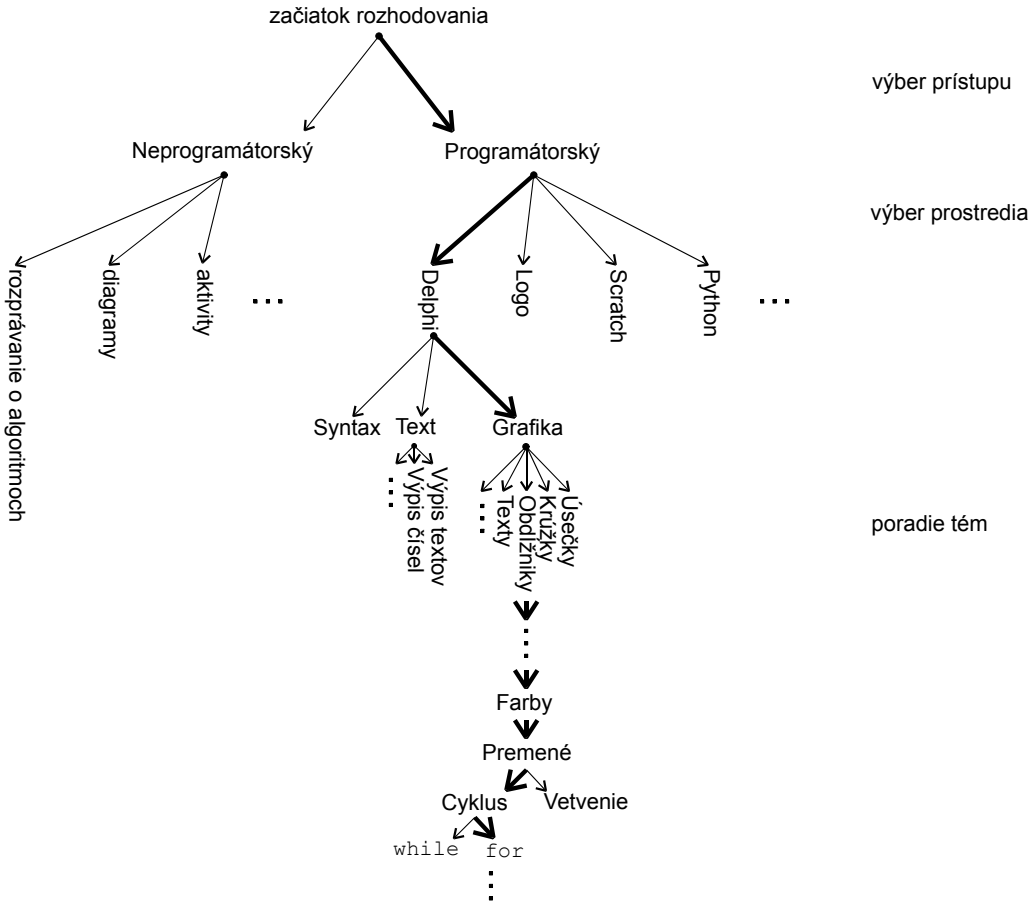
Pravdepodobne budeme počuť veľa názorov na to, ako učiť programovanie. Na vyučovanie algoritmov a programovania sa pozeráme ako na proces, pri ktorom sa neustále **rozhodujeme** a zvažujeme rôzne varianty: prístupy, metódy, témy, poradie tém, motivácie, ukážkové príklady, úlohy na samostatné riešenie atď. Proces rozhodovania môžeme znázorniť ako strom možností:

Strom možností ilustruje komplexnosť vyučovania programovania.

V strome vidíme aj niektoré známe, ale už nie odporúčané spôsoby vyučovania (napríklad, začíname vývojovými diagramami).

Každý môže pridať do stromu svoj vlastný variant vyučovania, svoju vlastnú postupnosť tém. Navyše, do stromu môžeme zaradiť rôzne, napríklad aj náhodne zostavené, postupnosti tém. Strom bude nakoniec extrémne rozsiahly.

Nad každou postupnosťou, aj nad takou, ktorá vyzerá pre nás na prvý pohľad nezmyselne, sa môžeme zamýšľať, či by sme ju dokázali takým spôsobom učiť a za akú cenu.



Strom možností je v skutočnosti veľmi rozvetvený. Každý vrchol v strome predstavuje určitý bod v rozhodovaní sa medzi rôznymi alternatívami. Mierka je veľmi hrubá – zväčša zvažuje oveľa viac možností, detailov

V strome vidíme veľa alternatívnych ciest, ako môžeme pri vyučovaní programovania postupovať.

Rozprávanie o algoritmoch

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

Charakteristické pre takýto prístup sú aktivity:

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

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

- Keďže žiaci nemajú predstavu o tom, čo je programovanie a ako programy fungujú, je teoretické rozprávanie o etapách a vlastnostiach algoritmov zbytočnou teoretickou záťažou, ktorú pravdepodobne nikto zo žiakov na začiatku nedocení. Skúmať vlastnosti algoritmov je samozrejme pre informatiku veľmi dôležité, ale neznamená to, že týmto spôsobom musíme začínať.

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

- Tým, že žiaci najskôr píšu programy na papier v nejakom algoritmickom jazyku alebo vo forme vývojových diagramov, komplikujeme žiakom porozumie nových poznatkov. Algoritmický jazyk aj vývojové diagramy sú zbytočnou medzivrstvou a komplikáciou, ktorá leží medzi objavením riešenia problému a jeho praktickým overením tým, že program spustíme na počítači a otestujeme.

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

Tradičný prístup

Pre tento prístup je charakteristické to, že:

- Žiaci programujú v klasickom prostredí TurboPascal, FreePascal, FPC.
- Žiaci od začiatku vytvárajú konzolové aplikácie (t.j. nie grafické programy, ako v prostredí Logo).

Konzolové aplikácie boli typické pre prostredie DOS. Konzolové aplikácie pracujú v textovom režime: vstup od používateľa si pýtajú v dialógovom riadku, výsledky vypisujú ako texty alebo čísla na obrazovke.

- Postupnosť tém: výpis na obrazovku, premenná, načítanie vstupu od používateľa, podmienený príkaz, cyklus **`for`**, **`while`**, **`repeat ... until`**, **`polia`**, ... Grafický režim býva ako samostatná téma často až nakoniec (akoby za odmenu).
- Dôraz sa kladie na riešenie prevažne matematických problémov a textovo orientovaných úloh (napríklad, napíšte program, ktorý nájde korene kvadratickej rovnice).

Takýto prístup sa v súčasnosti považuje už za prekonaný.

Súčasný prístup

V súčasnosti preferujme nasledujúci prístup:

- Žiaci pracujú v prostrediach, akými sú napríklad Imagine, Delphi, Lazarus.
- Od začiatku sa využívajú najmä grafické možnosti (grafika je jedným z nástrojov pre vizualizáciu algoritmov).
- Postupnosť tém: grafické príkazy, nastavujeme vlastnosti komponentov, premenná, cyklus `for`, podmienený príkaz, cyklus `while`, ...
- Od začiatku sa riešia aj úlohy s grafickým výstupom: nakreslenie obrázkov, tvorba nápisov. Riešia sa aj matematicky a textovo orientované úlohy, avšak nekladie sa na ne primárny dôraz.
- Tento prístup kladie zvýšené nároky na učiteľa. Napríklad, učiteľa často zvádza učiť komponenty, naklikávanie aplikácií.

Takýto spôsob vyučovania kladie zvýšené nároky na vyučovanie. Napríklad, v prípade prostredia Delphi, sa často stretávame s názorom, že komponenty celý program komplikujú – chyba je však v učiteľovi. Treba si uvedomiť, že v prípade prostredia Delphi nie je cieľom učiť komponenty, ani grafiku, ani mechanizmus udalostí, ani tvorbu „profesionálnych“ aplikácií. Komponenty, grafika a udalosti sú iba prostriedky, ktoré, ak ich správne (skromne) využijeme, získame pre vyučovanie veľa výhod: vynikajúce motivácie alebo nástroje na vizualizáciu algoritmov. Naším cieľom teda bude stále učiť základy programovania.

Poradie tém pre programovanie v Delphi alebo Lazarus

V súčasnosti preferujeme približne takéto poradie tém:

1. Úvod do programovania, prostredie a komponenty
2. Grafické príkazy, jednoduché kreslenie
3. Vlastnosti, typy
4. Premenné
5. Cyklus `for`
6. Podmienený cyklus `while`, jednoduché podmienky
7. Podmienený príkaz `if`, zložené podmienky
8. Procedúry a funkcie
9. Myš
10. Animácie a časovač
11. Znaky a znakové reťazce
12. Textové súbory
13. Polia
14. ...

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

Všimnime si na našej postupnosti, že:

- Grafike sa venujeme hneď na začiatku. To je veľký rozdiel od tradičných prístupov, kde sa grafika nachádza takmer na konci. Ako postupne uvidíme, vôbec nebudeme mať pocit, že sme zavčasu minuli zaujímavé úlohy. V skutočnosti však platí úplne opak – vďaka takémuto prístupu môžeme neskôr využívať oveľa širšiu paletu zaujímavých príkladov.
- Prostredie Delphi používame od úplného začiatku. V mnohých knihách sa programovanie učí najskôr v klasickom konzolovom prostredí, až potom v prostredí Delphi. My však budeme postupovať uvážene a tak, aby bol náš prístup pre žiakov pútavejší a užitočnejší. Pritom žiakov naučíme (minimálne) rovnaké koncepty, ako keby sme ich učili tradičným prístupom.

Každá z tém je značne rozsiahla, preto sa každej z tém zvykneme venovať viac vyučovacích hodín, napríklad v rozsahu od 2–8 hodín.

Kapitola 4: Programovanie v jazyku Pascal

V tejto kapitole sa venujeme vyučovaniu programovania v jazyku Pascal (v prostrediach Delphi alebo Lazarus) pre žiakov na strednej škole.

Pripomeňme si, že konštruktivistický spôsob vzniku poznatku sa formuje v určitých etapách:

Motivácia
↓
Zbieranie elementárnych skúseností
↓ ... formovanie vzťahov
Zbieranie rozširujúcich skúseností, objavovanie pravidiel
↓ ... zovšeobecňovanie, abstrakcia
Poznatok
↓
Trénovanie poznatku

Aby sme dokázali programovanie kvalitne učiť, musíme tému nielen ovládať, ale musíme aj rozumieť didaktickým postupom a naučiť sa ich správne používať.

Pritom je dôležité, aby žiaci nazbierali **dostatok vlastných** skúseností (t.j. učiteľ neprezentuje svoje vedomosti, skúsenosti). Majstrovstvo, teda **kvalita učenia** nespočíva v tom, že učiteľ porozpráva všetko, čo sám vie, ale v tom, že povie toľko, koľko žiaci potrebujú počuť.

Aktivita 4.1

Zostavte 5 až 6 členné tímy. Každý tím si zvolí jednu z tém, ktoré sme uviedli na konci predchádzajúcej kapitoly. Tému alebo jej časť spracujte v tíme tak, ako by sa mala učiť na jednej 45 minútovej hodine. Pri príprave môžete čerpať námety z učebnice Programovanie v Delphi [1], prípadne z materiálov DVUI [7], [8], [9]. Nakoniec zástupca tímu prezentuje výsledok pomocou projektora a pri tabuli.

Prezentáciu začnete analýzou tematického celku. Uveďte:

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

Ďalej predveďte, ako by mala vyzerať vzorová hodina. Teda, ako budete postupovať pri vyučovaní:

- čo a akým spôsobom budete vysvetľovať,
- čo budú riešiť žiaci samostatne.

Je nevyhnutné, aby ste rešpektovali všetky kritéria poznávacieho procesu. Treba:

- začať a používať vhodné motivácie,
- vymyslieť primeranú gradovanú sériu príkladov a úloh na zbieranie skúseností,
- navrhnúť ďalšie príklady alebo úlohy, ktoré budú žiaci samostatne riešiť.

Ostatní kolegovia na konci zhodnotia vystúpenie.

Môžete predpokladať, že výučba prebieha v počítačovej učebni, každý žiak sedí sám pri počítači, k dispozícii máte dataprojektor a tabuľu. Môžete tiež predpokladať, že ste už zapísali do triednej knihy, prípadne zopakovali učivo z minulej hodiny.

4.1 Úvod do programovania

Cieľ:

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

Nové pojmy, poznatky a zručnosti:

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

Poznámky:

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

Postup:

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

Namiesto zložitých vysvetlení použijeme vhodné **metafory**. Napríklad: „Image1.Canvas.TextOut je príkaz na výpis textu“.

Ak to s metaforami nepreženieme, žiaci nám budú rozumieť. Detaily sa žiaci dozvedia postupne, až na ďalších vyučovacích hodinách.

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

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

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

4.2 Príkaz priradenia

Cieľ:

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

Nové pojmy, poznatky a zručnosti:

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

Postup:

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

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

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

   Spočiatku je vhodné používať iba niekoľko farieb, ktorých mená dokážu žiaci ľahko vydedukovať (red, green, blue, yellow).
4. Postupne sa naučíme nastavovať aj typ písma a veľkosť písma:
   - **Veľkosť písma** nastavíme takto: `Image1.Canvas.Font.Size:=24;`
   - **Typ písma**: `Image1.Canvas.Font.Name:='Courier New';`
   - Príklady vyskúšame. Žiakov sa oplatí pýtať: „Kam treba príkazy zapísať?“
   - Ďalšia úloha: „Vyskúšajte vypísať slová veľkým písmom *Arial*“
5. Spolu so žiakmi vymyslíme **syntaktické pravidlo** pre nastavenie písma:
   Image1.Canvas.Font.vlastnosť := hodnota;
   Poznamenajme, že vo väčšine učebníc sa takýmito syntaktickými pravidlami začína – teda, autor ich čitateľovi prezradí. My sa však budeme snažiť, aby takéto pravidlá naši žiaci postupne objavovali.

   Je potrebné upozorňovať žiakov na pekné **formátovanie programu**: 1 príkaz = 1 riadok.
6. Riešiť úlohy, v ktorých sa postupne kombinujú a trénujú spomínané príkazy. Napríklad, vypisujú sa iné farebné pozdravy, „hviezdičky v rohoch obrazovky“...

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

4.3 Grafické príkazy

Cieľ:

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

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

Nové pojmy, poznatky a zručnosti:

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

Poznámky:

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

Postup:

1. Činnosť príkazu `Rectangle` budeme objavovať spoločne so žiakmi:
   - Začneme priamo triviálnym príkladom: „**Obdĺžniky** kreslíme príkazom `Image1.Canvas.Rectangle(100, 50, 300, 150);`“
   - Diskutujeme o tom, ako sa obdĺžnik nakreslí aj o parametroch príkazu.
   - Obdĺžnik aj s uvedenými súradnicami nakreslíme na tabuľu.
   - Pýtame sa aj: „Aké rozmery má obdĺžnik?“

Všimnime si, že:

- začíname **triviálnym** príkladom,
- potom riešime úlohy, ktoré majú **zložitejšie** parametre alebo parametre podľa stanovených kritérií,
- nakoniec sú úlohy na **trénovanie**.

2. Bude potrebné, aby žiaci zbierali ďalšie skúsenosti s tým, ako príkaz funguje:
   - „Aký obdĺžnik nakreslí príkaz `Rectangle(100, 100, 200, 200)`?“
   - „Ako nakreslíme **štvorec** so stredom v bode 100, 100 a dĺžkou strany 50?“ Pripomeňme, že „štvorec je vlastne špeciálny prípad obdĺžnika“.
   - Predchádzajúce príklady sú vhodnou príležitosťou na to, aby sme prezradili, že parametre príkazu môžu byť aj **jednoduché výrazy** (zatiaľ iba s operátormi `plus, mínus`)
3. Komplexnejšia úloha na trénovanie: “Pomocou obdĺžnikov nakreslite robota.“
4. Naučíme sa používať **obrys** a **výplň**. Ich použitie ukazujeme na jednoduchom príklade, napríklad kreslíme obdĺžnik so žltým vnútrom a červeným obrysom.
5. Úloha na trénovanie: “Pomocou obdĺžnikov nakreslite vlajku Nemecka.“
6. Podobný princíp použijeme pri **elipse** a **kruhu**. Nezabudnime, že žiakom treba vysvetliť ideu, že “elipsa sa vpisuje do neviditeľného obdĺžnika". Nakoniec zvolíme vhodnú úlohu na trénovanie, napríklad: “Nakreslite snehuliaka”.
7. Úlohy, v ktorých sa kombinuje kreslenie obdĺžnikov a elíps: „Nakreslite strom (pomocou kruhu a obdĺžnika)“, „Nakreslite značku *zákaz vjazdu*“, „nakreslite autíčko (karoséria z dvoch obdĺžnikov, kolesá z kruhov)“, ...
8. Neskôr naučíme aj kresliť **úsečky**, namiešať **vlastné farby** viď. [1].

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

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

4.4 Komponenty a ich vlastnosti

Cieľ:

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

Nové pojmy, poznatky a zručnosti:

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

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

Porovnajme stratégie:

- v tradičnej literatúre sa príkaz priradenia spomína prvýkrát až pri téme premenná,
- v našom prístupe žiaci spoznajú príkaz priradenia ešte skôr - pri nastavovaní farby a teraz, keď učíme pracovať s vlastnosťami.

Treba si zároveň uvedomiť, že našu strategickú výhodu sme získali vďaka prostrediu Delphi (Lazarus). Totiž, nie každý jazyk resp. prostredie nám umožňuje aplikovať takýto prístup.

Postup:

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

   Všimnime si, že výraz na pravej strane od := robíme zložitejším iba **po malých krokoch**. Robíme tak preto, aby žiaci dokázali vnímať a a vyhodnocovať rôzne situácie.
6. Spolu so žiakmi zhrnieme a objavíme predpis, ako sa nastavujú vlastnosti“
   komponent.vlastnosť := výraz
   Z predchádzajúcich príkladov by mal vzísť poznatok, že *výraz* sa **najskôr vyhodnotí** a až podľa výslednej hodnoty sa **potom nastaví** vlastnosť.
7. Ďalšie námety na trénovanie:
   - Projekt *Ako sa máš?* na strane 12 v učebnici [1] (prípadne aj úloha 1).
   - Prvá úloha s **editovacím políčkom** v učebnici [1] v kapitole *2.3*.

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

4.5 Typy údajov

Cieľ:

- porozumieť tomu, že pre počítač sú čísla, text a farby rôzne typy údajov,
- spoznať základné matematické operácie, typy údajov, konverziu typov.

Nové pojmy, poznatky a zručnosti:

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

Poznámka: Budeme vychádzať zo skúseností, ktoré žiaci doposiaľ nazbierali. Zároveň budeme túto tému chápať ako prípravu pre prácu s premennými.

Postup:

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

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

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

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

Všimnime si, že zatiaľ `IntToStr` nenazývame funkciou.

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

V úlohách na trénovanie budú výrazy s konštantami.

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

4.5 Náhodné čísla

Cieľ:

- naučiť používať príkaz `random` na generovanie náhodných čísel

Nové pojmy, poznatky a zručnosti:

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

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

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

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

Postup:

1. V krátkosti iba stručne naznačíme čosi **o náhode** v programoch a ihneď ukážeme príklad, v ktorom sa používa príkaz `Random`:
   - „Doposiaľ fungovali naše programy veľmi presne. Napríklad, pozdrav sa vypísal na súradnice, ktoré sme presne vypočítali. Pozdrav však môžeme vypísať aj na náhodné miesto takto:“
   - `Image1.Cavnas.TextOut(Random(400), Random(300), 'Ahoj')`
   - Povieme žiakom, aby program vyskúšali.
   - Opýtame sa, „Kam sa vypíše pozdrav, po stlačení tlačidla?“
   - Zhrnieme: „Vidíme, že program vypisuje pozdrav vždy na iné miesto.“

2. Bude potrebné vysvetliť, ako „príkaz“ `Random` pracuje: „`Random` si vymyslí **náhodné číslo**. Až toto číslo sa použije ako parameter príkazu `TextOut`. Napríklad, `Random(400)` vymyslí číslo, ktoré bude zaručenie menšie ako `400`. Presnejšie, vymyslené číslo bude ležať v intervale od `0` po `399`.“

3. Bude potrebné riešiť úlohy na spoznávanie a trénovanie náhodných čísel:
   - Úloha: „Aké čísla bude zobrazovať príkaz `Image1.TextOut(0, 0, IntToStr(Random(10))`?“
   - Postupne sa pýtame aj na to, aké čísla by sa vypisovali pre rôzne **parametre** `Random(100)`, `Random(2)`, `Random(1)`. A naopak, môžeme sa pýtať aj na to, aký parameter musíme použiť, aby sa generovali náhodné čísla **rozsahov** od `<0, 49>` alebo `<0, 50>`.
   - Komplikovanejšia úloha: „Vidíme, že príkaz `random` môžeme použiť ako hraciu kocku. Ako dosiahneme, aby sa zobrazovali čísla od `1` po `6`?“
   - Náročné úlohy: „Chceme generovať náhodné čísla z rozsahu `<-10, 10>`“, alebo „Chceme náhodne generovať iba čísla `+1` alebo `−1`.“

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

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

Nakoniec riešime až také problémy, v ktorých už nestačí iba zmeniť parameter pre `Random`, ale žiaci musia prísť aj na to, že treba použiť výraz s týmto príkazom.

4. Ďalej môžeme:
   - Kresliť so žiakmi náhodne umiestnené červené obdĺžniky.
   - Naučiť žiakov používať náhodné farby (napr. `Random(256*256*256)`). Potom môžeme so žiakmi kresliť náhodne zafarbené geometrické tvary, náhodne umiestnené nápisy náhodnej veľkosti,...
   - Námety sú aj v učebnici [1] v kapitole 2.1.

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

4.6 Premenné

Cieľ:

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

Nové pojmy, poznatky a zručnosti:

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

Poznámky:

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

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

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

Postup:

1. Začneme motiváciou - ukážkou, na ktorej žiaci uvidia zmysel použitia (lokálnej) premennej, napríklad:
   - „Do stredu grafickej plochy nakreslíme štvorček. Tvárime sa, že nepoznáme rozmery grafickej plochy. Preto musíme súradnice vypočítať z vlastností `Width` a `Height Image1`.“
   - Úlohu spoločne so žiakmi vyriešime:
     ```
     Image1.Canvas.Rectangle(
       Image1.Width div 2 - 10, Image1.Height div 2 - 10,
       Image1.Width div 2 + 10, Image1.Height div 2 + 10)
     ```
   - „Vidíme, že niektoré výrazy sa počítajú dvakrát, v matematike by sme ich označili písmenami `x` a `y`“ - tieto označenia nakreslíme.
   - „Riešenie, ktoré by sme zapísali pomocou `x` a `y`, by potom vyzeralo takto elegantne„ a výsledok napíšeme:
     ```
     Image1.Canvas.Rectangle(x-10, y-10, x+10, y+10)
     ```
   - Pripomenieme, že „v matematike sme `x` a `y` nazvali premenné“ a dokončíme, že „aj v programoch môžeme používať premenné.“

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

2. Je potrebné stručne ozrejmiť význam premenných v programovacom jazyku:
   - Čo je **premenná**: „Miesto v pamäti počítača.“
   - Ako funguje: „Do premennej **uložíme** nejakú hodnotu, napríklad vypočítame súradnicu stredu. Toto číslo si premenná **zapamätá**. Keď budeme súradnicu stredu potrebovať, pozrieme sa do premennej a zapamätané číslo prečítame, **zistíme**.“
   - Oplatí sa použiť prirovnania, napríklad: „Je to podobné, ako pamäť na kalkulačke `M`. Akurát, že v programe môžeme mať veľa takých pamätí“.

3. Ďalej sa nevyhneme **deklarácii** premennej:
   - Zásadne budeme používať lokálne premenné - tento pojem pred žiakmi zatiaľ nepoužívame.
   - „Ak chceme premenné používať, musíme najskôr takýto náš úmysel počítaču prezradiť. Robíme tak v zápise, ktorý nazývame **deklarácia**.“
   - Ukážeme, kde v programe píšeme `var X, Y: Integer;`
   - „Pri deklarácii vzniknú v pamäti počítača dve premenné `X` a `Y`“.
   - Zároveň aj kreslíme krabičky `X` a `Y` - **krabičkový model** je pri vysvetľovaní veľmi dôležitý aj užitočný.

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

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

     ```
     A:=2;                  ... jednoduché priradenie
     B:=A+3;                ... použitie premennej a vyhodnotenie výrazu
     Sucet:=A+B;            ... použitie premenných a vyhodnotenie výrazu
     Image1.Canvas.TextOut(0, 0, IntToStr(Sucet));
                            ... výpis hodnoty premennej
     ```

   - Príkazy budeme trasovať, prípadne žiadať žiakov, aby nám s trasovaním pomohli. Je dôležité, aby sme do nakreslených krabičiek vpisovali údaje.

6. Riešime aj úlohy, v ktorých sa zadáva **číselný vstup** z editovacieho políčka, napríklad v učebnici [1] v kapitole 2.3.

7. Úlohy na trénovanie:
   - Podobné úlohy na trasovanie, ako v predchádzajúcom 5. bode.
   - Príklady v učebnici [1] v kapitolách 2.2, 2.4.
   - Príklady, ktoré sme uviedli v predchádzajúcom module [7]

8. Neskôr učíme aj:
   - Lokálne a **globálne** premenné - dobrá motivácia na globálne premenné je úloha: „vytvorte program, ktorý počíta kliknutia na tlačidlo“.
   - Používanie premenných rôznych typov.
   - Pravidlá pre pomenovanie a deklarovanie premenných.

Pozor:

- Neučíme premenné na ukážke s príkazom `random` - napríklad:

  ```
  a:=random(100);
  b:=a+10;
  ```

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

    ```
    A:=1;      ... do premennej sa priradilo číslo
    B:=A+2;    ... do premennej sa priradila funkcia A+2
    A:=10;     ... preto, pri vypísaní premennej B, uvidíme číslo 12
    ```

  - Ak zvolíme zlý model - napríklad nie krabičky, ale vrecúška s guľôčkami, môžu operáciu priradenia chápať ako presúvanie:

    ```
    A:=10;     ... v premennej A bude 10
    B:=A;      ... v premennej A bude 0 alebo nič.
    ```

Treba si uvedomiť, že premenné v programoch majú niekoľko úloh:

- **uchovávajú** údaje (niekedy iba dočasne),
- sú nástrojom na **zovšeobecnenie** - umožňujú vytvárať všeobecnejšie riešenia problémov,
- sú nástrojom na **zefektívnenie** riešenia - rovnaký výpočet sa realizuje iba raz, výsledok sa potom viackrát použije.

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

Zamyslite sa nad problémom výmeny obsahu dvoch premenných:

- Akú motiváciu by ste použili?
- Kedy má pre žiakov zmyslel riešiť takýto problém?
- Kedy by ste túto úlohu zaradili do vyučovania?
- Ako by ste pomohli žiakom pri riešení?

4.7 Cyklus `for`

Cieľ:

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

Nové pojmy, poznatky a zručnosti:

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

Poznámky:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Postup:

1. Začneme úlohou, keď treba veľa niečoho nakresliť. Napríklad: „Chceme nakresliť oblohu s veľkým množstvom hviezdičiek.“
   - Úlohu necháme riešiť žiakom s tým, aby najskôr kreslili jednu hviezdičku:
     ```
     Image1.Canvas.TextOut(random(400), random(300), '*');
     ```
   - Pýtame sa: „Ako by sme nakreslili 2 hviezdičky?“
   - Žiaci by pravdepodobne napísali alebo skopírovali príkaz ešte raz.

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

Motivácia je založená na nešikovnom riešení, kým nepoužijeme cyklus.

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

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

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

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

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

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

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

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

Trasovacie tabuľky môžeme použiť nielen na to, aby žiaci videli, ako program pracuje. Môžeme ich použiť aj opačne, teda na to, aby žiaci zovšeobecnili alebo hľadali určité vzťahy.

Namiesto tabuliek, zvykneme pri riešení grafických problémov často a **veľa kresliť** obrázky, na ktorých vidno vzťah medzi súradnicami a riadiacou premennou.

7. Úlohy na trénovanie a ďalšie námety:
   - Úlohy a témy z učebnice [1] v kapitole 2.5
   - Úlohy s číslami a s ciframi z učebnice [1] v kapitole 2.6.
   - Vnorený cyklus podľa učebnice [1] v kapitole 2.7 alebo modulu [7].

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

4.8 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.
- Pokým stačí používať `for` cyklus, nemá zmysel nútiť žiakov používať 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` ktorý počíta 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` *opakujeme príkazy*
   kreslíme vagón
   `X:=X+40+10;`

Pri trasovaní volíme takú kombináciu hodnôt, aby sme sa pri trasovaní a písaní príliš nenatrápili. Zároveň by malo byť vidieť, ako sa mení aspoň zopár krokov, opakovaní tela cyklu.

5. Keďže sme v slovnom popise použili šikovné slovenské slová, ľahko ukážeme, ako spomínaný algoritmus zapíšeme pomocou konštrukcie `while ... do`.
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 výraz `X+40 ≤ 170`.
7. Ďalšie námety, príklady a úlohy na trénovanie nájdeme v učebnici [1] v kapitole 3.3

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

4.9 Podmienený príkaz `if`

Cieľ:

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

Nové pojmy, poznatky a zručnosti:

- konštrukcia `if ... then ... else`, a `if ... then` bez vetvy `else`,
- syntaktické pravidlá, fungovanie príkazu, problémy s príkazom,
- jednoduché podmienky, zložené podmienky, logické výrazy.

Poznámky:

- Je dôležité aby sme rozpoznali rôzne úrovne náročnosti problémov, ktorých riešenie vyžaduje:
  - jeden podmienený príkaz s jednoduchou podmienkou,
  - podmienený príkaz so zložitejším telom v kontexte iných príkazov,
  - podmienený príkaz v kombinácii s cyklom:
    - keď testujeme riadiacu premennú (a popritom nastavujeme napríklad farbu),
    - alebo si všímame iba určité hodnoty (vypisujeme iba párne čísla).
  - kombináciu vnorených podmienených príkazov,
- Poradie, v akom učíme podmienený príkaz `if` a cyklus `while` nie je striktne dané a môžeme ho vzájomne vymeniť.

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.

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'
     ```

3. Teraz algoritmus naprogramujeme:
   - Ukážeme, ako jednotlivé slová prekladáme a príkazy popritom aj zapisujeme v programovacom jazyku.
   - Upozorníme na miesta, kde sa (ne)píšu bodkočiarky.
   - Program vyskúšame.
4. So žiakmi sa zamýšľame a diskutujeme o tom, kedy sa vykoná ktorá vetva programu, ktorá **vetva** programu sa vykoná pre čísla: `-10`, `30`, ako aj pre hraničné vstupy `21` a `22`.
5. Neskôr je potrebné riešiť sériu úloh, na ktorých postupne vidno:
   - že vetva môže byť zložený príkaz z `begin` a `end`,
   - situácie, ako sa spárujú vnorené príkazy `if` s vetvou `else`.
   - ako konštruujeme zložitejšie podmienky, atď.
6. Príklady a úlohy: v učebnici [1] v kapitole 3.4 alebo v moduloch [7] a [12].

Je dôležité, aby sme použili vhodné slovenské slová, ktoré primerane vystihujú anglický preklad aj fungovanie príkazu.

V programoch je cyklus akoby synonymom pre silu, vytrvalosť, výkon a podmienený príkaz zase pre rozum, logiku, inteligentné správanie.

Pozor: Nezačíname komplexnou ukážkou (s viacerými príkazmi, s komplikovanými podmienkami, v kombinácii s cyklom), na ktorej „všetko predvedieme“.

4.10 Polia

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

Cieľ:

- naučiť evidovať si a spracovávať veľké množstvo údajov,
- pochopiť idey poľa a práce s prvkami poľa pomocou indexov.

Nové pojmy, poznatky a zručnosti:

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

Poznámka: Téma pole je pre žiakov veľmi **náročná**. Komplikácie v porozumení spôsobuje najmä nová idea – používanie indexov pre prístup k prvkom poľa.

V 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. Pritom pri poliach treba situácie znázorňovať veľmi poctivo. Napríklad, ak si ako učitelia ušetríme námahu a zápisy v prvých príkladoch budeme skracovať pomocou troch bodiek `...`, môžeme takéto „vylepšenia“ očakávať aj vo viacerých žiackych programoch.

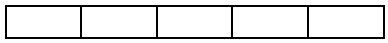
Podobne, ako nám pri premenných pomáhal krabičkový model, pri poliach pomôže predstava číslovaných **chlievikov**.

Treba si zvyknúť na to, že pri poliach **intenzívne znázorňujeme** a kreslíme.

Abstraktné vzorce, ako `Suma:=Suma+A[i]` nesmieme žiakom prezrádzať ako akúsi mágiu. 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 sami zažiť, aby dokázali aj v budúcnosti sami riešiť podobné problémy.

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 premenné spočíta 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 je 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 tesne za sebou“. Pole zároveň **kreslíme** na tabuľu ako sériu piatich krabičiek – chlievikov:

     A 
   - Vysvelíme a na obrázku ukážeme, čo sú **prvku poľa** a **indexy**. Chlieviky popritom na tabuli očíslujeme.
3. Preskúmame, ako fungujú indexy:
   - “Prvky poľa majú špeciálne mená: `A[1]`, `A[2]`, `A[3]`, `A[4]`, `A[5]`.“
   - Začíname s príkladmi, situáciami, v ktorých sa postupne používa:
     - index s konkrétnym konštantným číslom: `A[1]:=15; A[2]:=9;`
     - index s konštantným výrazom: `A[2+1]:=31;`
     - index ako premenná: `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** – najskôr riešime úlohy bez cyklu, pomocou piatich príkazov s konkrétnymi číslami ako indexmi.
5. Ďalej môžeme prvky poľa spočítať (postupujeme podobne, ako v bode 4).

Pozor: Nezačíname príkladom, kde ihneď kombinujeme cyklus s abstraktným zápisom `a[i]`. V prvom ukážkovom príklade nekombinujeme pole s náročným algoritmom (napríklad triedenie, vyhľadávanie – tie sú pre žiakov náročné). 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.

Kapitola 5: Súťaže

Na Slovensku sa realizuje viacero súťaží z informatiky [13][14].

Aktivita 5.1	Aké informatické súťaže poznáte, o akých ste počuli?
Aktivita 5.2	Do ktorých súťaží sa vaši žiaci zapájajú?
Aktivita 5.3	Aké máte skúsenosti s informatickým súťažami vy?

Mnohé z nich sú určené pre žiakov strednej školy, existuje však niekoľko súťaží, ktoré sú svojou náročnosťou nastavené aj pre žiakov na základnej škole.

Súťaže pre základné školy:

- Baltík ... http://baltik.infovek.sk
- Cologobežka ... http://edi.fmph.uniba.sk/~tomcsanyiova/ImagineLogoCup
- First Lego League ... http://www.firstlegoleague.sk
- iBobor ... http://ibobor.sk
- ImagineCup ... http://edi.fmph.uniba.sk/~tomcsanyiova/ImagineLogoCup
- Miniprog
- Palma Junior ... http://di.ics.upjs.sk/palma
- RoboCup Junior ... http://www.robotika.sk

Súťaže pre stredné školy:

- iBobor ... http://ibobor.sk
- Korešpondenčný Seminár z Programovania(KSP) ... http://www.ksp.sk
- Liaheň ... http://liahen.ksp.sk
- Olympiáda v Informatike(OI) ... http://oi.sk
- Palma ... http://palma.strom.sk
- ProFIIT ... http://www2.fiit.stuba.sk/ProFIIT
- RoboCup Junior ... http://www.robotika.sk
- Zenit ... http://siov.cmsromboid.sk/zenit

Súťaže pre stredoškolákov a dospelých:

- Haluz ... http://www.haluz.org
- Istrobot ... http://www.robotika.sk
- Internet Problem Solving Contest(IPSC) ... http://ipsc.ksp.sk
- Topcoder ... http://www.topcoder.com/tc
- USACO ... http://train.usaco.org

Súťaže mávajú rôznu formu. Napríklad:

- Olympiáda v Informatike má teoretické a praktické úlohy – niektoré sa riešia iba na papieri a iné na počítači. Stredoeurópska a medzinárodná súťaž má iba praktické úlohy pri počítači (programuje sa v jazykoch C, C++ alebo Pascal).
- iBobor je online súťaž, ktorá je zameraná na IKT ale aj riešenie problémov.

Každá zo súťaží má svoj cieľ a oslovuje určitú skupinu žiakov. Napríklad:

- OI, Zenit, alebo KSP sú zamerané na riešenie algoritmických problémov. Znamená to, že treba vyriešiť, prípadne naprogramovať zadanú úlohu.
- iBobor je súťaž pre všetkých žiakov. Jej hlavným cieľom je podporiť záujem o informačné a komunikačné technológie.

Niektoré súťaže majú sústredenie, napríklad KSP má sústredenie ako odmenu pre úspešných riešiteľov alebo OI vyberá a trénuje súťažiacich pre medzinárodné kolo.

Informatické súťaže považujeme za veľmi cennú súčasť vzdelávania. Keďže sú pre žiakov **nepovinné**, ich autori často zostavujú veľmi netradičné, zaujímavé až zábavné úlohy. Pritom takéto úlohy môžu byť niekoľkonásobne náročnejšie ako tie, ktoré sa bežne riešia na hodinách informatiky.

Informatické súťaže považujeme za dôležité aj preto, lebo do značnej miery informatiku popularizujú. Na súťažiach zažijú žiaci veľmi odlišnú atmosféru od tej, ktorá panuje v škole. Tým, že sa vzájomne porovnávajú so svojimi rovesníkmi, vzniká medzi nimi obrovská motivácia riešiť problémy. Úspešní riešitelia potom zažijú aj úžasnú chuť úspechu alebo víťazstva. Pri riešení problémov nadobudnú žiaci množstvo vlastných skúseností a nových poznatkov – takých, ktoré v škole nemajú šancu získať.

Je prirodzené, že v škole sa nemôže učiteľ venovať iba nadaným žiakom. Preto sú mnohé súťaže orientované na talentovaných žiakov. Ukazuje sa, že takýto nenásilný a prirodzený mechanizmus, kde sa vzájomne dopĺňa formálne a neformálne vzdelávanie je v súčasnosti veľmi výhodný pre žiakov, učiteľov aj vysoké školy.

Kapitola 6: Hodnotenie

Hodnotenie žiakov je súčasťou nášho vzdelávacieho systému. Preto sa aj v rámci informatiky musíme zamýšľať nad tým, ako budeme hodnotiť našich žiakov.

Na zadanie si musíme dať veľký pozor. Testovaciu úlohu treba vymyslieť tak, aby bolo zadanie:

- **zrozumiteľné** (jednoducho, a pritom jasne a formulované požiadavky na riešenie),
- žiakmi **riešiteľné** (napríklad, zlá úloha je, ak od žiakov očakávame triediaci algoritmus, hoci sme predtým podobné problémy neriešili),
- **férové** (teda, nie hračky so slovíčkami, aby sa pri riešení nevyužívali rôzne triky, exotické finty),
- **jednoznačné** (aby sa úloha nedala pochopiť inak - vtedy musíme uznať žiakove argumenty a jeho riešenie),
- formálne **správne** (napríklad, musíme syntakticky správne deklarovať pole).

Aktivita 6.1	Ako by ste vyskúšali žiaka z programovania, napríklad, či dokáže používať pole? Vymyslite zadanie, navrhnite hodnotenie.
Riešenie	Predpokladáme, že žiaci už riešili niekoľko príkladov s poľami aj v kombinácii s cyklami a podmienenými príkazmi. Zadanie: Chceme zistiť, koľko spolužiakov má výšku 180 cm a viac. Výšky spolužiakov už máme uložené v poli: `var V: array[1..27] of Integer;` Napíšte kúsok programu, ktorý uloží do premennej `N` počet tak vysokých spolužiakov. Očakávame, že žiaci budú riešiť úlohu samostatne na papier. Riešenie by nemalo trvať dlhšie ako 10 minút. Hodnotenie: • Predpokladané riešenie: `N:=0;` 1 bod `for i:=1 to 27 do` 2 body `if V[i]>=180 then N:=N+1` 5 bodov • Súčet bodov 8 • Známka podľa vzorca `(10-body) div 2`

Na hodnotenie sa pozeráme **pozitívnou optikou** (použijeme známe porovnanie s pohárom: nehovoríme, že pohár je poloprázdny, ale naopak, že je poloplný). Našim cieľom nie je dokázať žiakom, čo nevedia, ale zistiť, čo vedia a ako tomu rozumejú. Tomuto prístupu nám napomáha aj **pripočítanie bodov** za splnené kroky pri riešení, resp. hodnotenie za to, koľko stačí v programe iba opraviť, aby fungoval.

Vysvetlenie riešenia:

- Nebudeme zisťovať teoretické vedomosti. Budeme hodnotiť riešenie veľmi jednoduchého problému. Na to, aby sme videli, že žiaci dokážu pracovať s prvkami poľa, nám absolútne vyhovuje práve takáto **jednoduchá úloha**.
- Pri hodnotení sledujeme, či žiaci **vedia**:
  - prechádzať prvky poľa pomocou cyklu s vhodnými hranicami,
  - že v tele cyklu treba použiť podmienený príkaz,
  - zostaviť podmienku,
  - v podmienke použiť správnu syntax pre prístup k prvku poľa,
  - vykonať inkrementáciu pri splnenej podmienke.
- Bodmi vyjadrujeme váhu a náročnosť úvah, ktoré musel žiak uskutočniť pri vymýšľaní riešenia. Napríklad, posledných 5 bodov „za `if`“ znamená, že žiak môže získať 0, 1, 2, ... až 5 bodov podľa toho, nakoľko dobre zvládol príkaz zostaviť. Znamená to, že aj tento príkaz môžeme rozmeniť na drobné kroky:
  - za `if` ... +1 bod (bod by žiak získal, aj keby nič iné nenapísal),
  - za správne použite indexu `v[i]` v podmienke ... +1 bod,
  - za test >=180 ... +1 bod,
  - za N:=N+1 ... +1 bod,
  - za správne telo cyklu ... +1 bod.
- Aj napriek našej dobrej príprave určite žiaci objavia iné, možno až netypické, a pritom správne riešenie. My by sme mali byť natoľko flexibilní a ústretoví, aby sme dokázali posúdiť a bodovať aj také riešenia.

Nehodnotíme umelecký dojem.

Aktivita 6.2	Viete si predstaviť aj nesprávne hodnotenie predchádzajúcej úlohy? Podeľte sa oň.

Čo sme sa naučili v tomto module

Zhrnutie

Diskutovali sme o vlastnostiach programovacích jazykov a vývojových prostredí a o ich vhodnosti pre vyučovania programovania začiatočníkov v škole.

Analyzovali, navrhovali a hodnotili sme niekoľko tém z programovania na strednej škole z pohľadu didaktiky.

Venovali sme sa informatickým súťažiam.

Diskutovali sme o hodnotení žiakov z programovania.

Preverenie výstupných vedomostí

Zvoľte tému z programovania v jazyku Pascal alebo v inom programovacom jazyku. Analyzujte ju a zostavte metodický materiál pre jednu vyučovaciu hodinu tak, aby rešpektoval spomínané didaktické pravidlá.

Literatúra a použité zdroje

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

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

[3] Varga, M., Blaho, A., Zimanová, R.: Algoritmy s Logom. Bratislava: Slovenské pedagogické nakladateľstvo, 1999. ISBN 80-08-02965-X.

[4] Zimanová, R., Belušová, M., Varga, M.: Algoritmy s Pascalom. Bratislava: Slovenské pedagogické nakladateľstvo, 2002. ISBN 80-08-03289-8.

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

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

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

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

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

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

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

[12] Blaho, A., Salanci, Ľ.(2009) DVUI: *Programovanie 1 – 9 pre 2CS*. Bratislava: ŠPÚ 2009.

[13] Lipková J.: Informatické súťaže na Slovensku,. DidInfo 2009. Banská Bystrica: UMB. ISBN 278-80-8083-720-4.

Webové stránky:

[14] http://people.ksp.sk/~julka/sutaze/

[15] http://www.statpedu.sk/

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

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

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

Podnázov Didaktika programovania 2

Študijný materiál prešiel recenzným pokračovaním.

Recenzenti PaedDr. Daniela Bezáková, PhD.
RNDr. Gabriela Lovászová, PhD.

Počet strán 36

Náklad 400 ks

Prvé vydanie, Bratislava 2010

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 978-80-8118-053-8