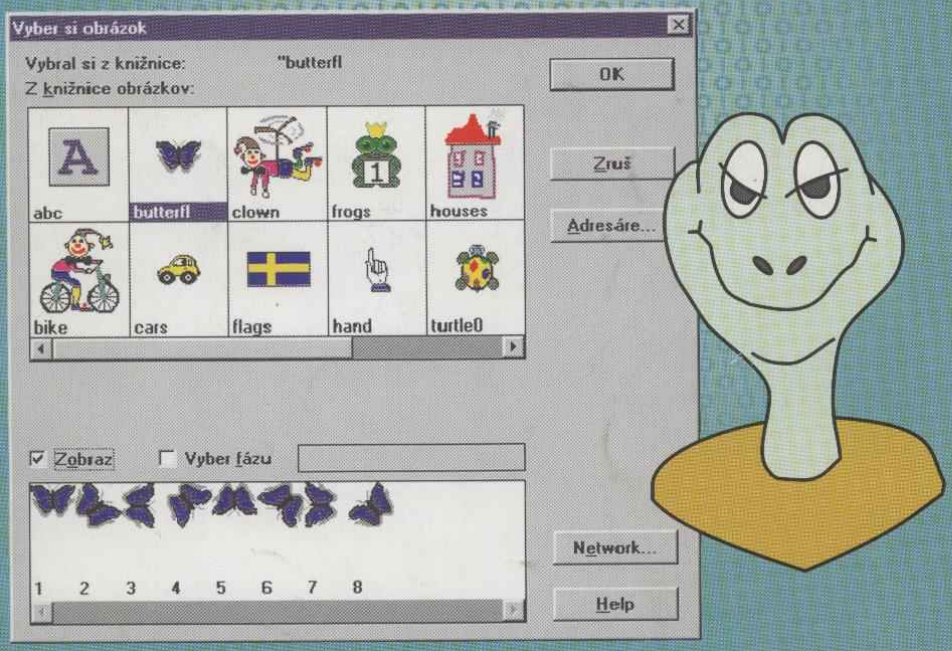
INFORMATIKA

PRE GYMNÁZIÁ

Algoritmy s Logom



Slovenské pedagogické nakladateľstvo

Obsah

Predhovor

Programovací jazyk Logo vznikol koncom 60-tych rokov v USA. Na rozdiel od iných programovacích jazykov má Logo za cieľ predovšetkým učenie, skúmanie a experimentovanie. Ponúka deťom, študentom a učiteľom prostredie pre *nové spôsoby učenia sa:* pre učenie sa vlastným objavovaním, postupné konštruovanie svojich znalostí, pre objavovanie súvislostí, riešenie problémov a pre vlastnú tvorbu. Rozvíja logické a algoritmické myslenie, dáva nám do rúk jednoduchý, a predsa veľmi silný nástroj na vyjadrovanie svojich myšlienok a skúmanie nových pojmov. Napríklad v informatike, v matematike, na hudobnej či výtvarnej výchove... prakticky na každej hodine, kde prevláda chuť hľadať, tvoriť a postupovať novými cestami.

Držíte v rukách tematický zošit, ktorý bude vaším sprievodcom pri oboznamovaní sa s programovacím jazykom Logo. V zošite je minimum teórie, všetky nové pojmy vysvetľujeme na riešených príkladoch. Okrem prečítania a pochopenia týchto príkladov by ste si ich mali tiež naprogramovať, pretože niektoré cvičenia sa na ne odvolávajú a riešením býva častokrát iba malá modifikácia. Pre veľmi šikovných študentov sme na záver niektorých kapitol zaradili *Ďalšie úlohy*. Tieto obsahujú náročnejšie úlohy, ktoré nemusia zvládnuť všetci študenti.

Zošit je rozdelený na 13 kapitol. V prvých siedmich kapitolách sa prostredníctvom korytnačej geometrie oboznámite so základnými algoritmickými a programátorskými pojmami ako cyklus, procedúra, vstupy, vetvenie výpočtu, rekurzia... . V ďalších kapitolách sa naučíte používať také prostriedky Loga (vstupy z klávesnice a od myši, viacnásobné korytnačky, tvary a animácie), pomocou ktorých sa dajú veľmi jednoducho naprogramovať zaujímavé a graficky príťažlivé projekty.

Niektoré príklady sme navrhli tak, že sa k nim v neskorších kapitolách vraciame, po zvládnutí nových príkazov ich dopĺňame a vylepšujeme. Mali by ste si preto systematicky a prehľadne ukladať jednotlivé projekty, aby vám nerobilo problémy sa k nim s odstupom času vrátiť a meniť ich.

V jednotlivých kapitolách sme definície nových pojmov zvýraznili rámikom. Odporúčame vám, hlavne na začiatku, prepísať si ich na samostatný papier a ten používať pri počítači ako ťahák.

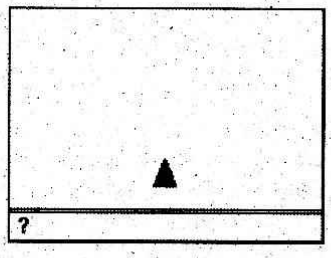
Veríme, že vám tento tematický zošit prinesie veľa radosti pri oboznamovaní sa nielen s programovacím jazykom Logo, ale aj pri objavovaní tajomstiev informatiky.

Autori

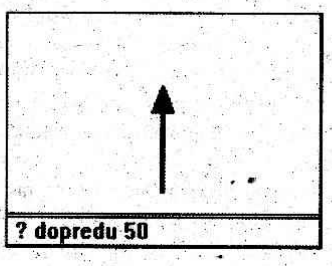
1 Úvod do Loga

V jazyku Logo má hlavnú úlohu korytnačka – je to v podstate grafické pero, ktoré sa na základe príkazov programátora pohybuje po grafickej ploche, pričom za sebou môže zanechávať stopu (kreslí čiary). Pozícia korytnačky je znázornená zeleným trojuholníkom – na začiatku sa nachádza v strede obrazovky. Trojuholník vyjadruje aj smer natočenia korytnačky. Tento smer je veľmi dôležitý pre jej pohyb, lebo základným príkazom **dopredu** ju posunieme v momentálnom smere.

Všimnime si, že obrazovka sa skladá z dvoch častí: grafickej časti (tu „žije" korytnačka) a textovej (pod rozdeľovacou čiarou) – tu komunikujeme s Logom, sem zadávame príkazy pre korytnačku (preto to nazývame príkazový riadok). Niekedy sa tu dozvieme aj výsledky výpočtov, alebo sem môže Logo vypisovať nejaké správy pre nás (najčastejšie správy o chybách). Pri písaní textu do počítača, pokým nestlačíme kláves **Enter**, riadok sa nevykoná a môžeme ho ešte opravovať a meniť.



Korytnačka, t. j. grafické pero, môže mať rôznu farbu, hrúbku, toto pero môže byť v zdvihnutej polohe (pri pohybe korytnačka nezanecháva stopu) alebo spustené (bude kresliť). Prvý príkaz, ktorým sa naučíme niečo s korytnačkou robiť, je príkaz **dopredu**. Tento príkaz posunie korytnačku o určitý počet *korytnačích krokov* dopredu. Ak je pritom jej pero spustené, nakreslí sa čiara. Počet krokov, ktoré má korytnačka po tomto príkaze prejsť, zadávame za príkaz **dopredu** a hovoríme, že tento príkaz má jeden vstup (hovorí sa mu aj parameter), napr.: **dopredu 50**.



ÚLOHA

1. Zistite, akú dlhú dráhu prejde korytnačka po postupnom zadaní týchto príkazov (predpokladáme, že obrazovka bola na začiatku čistá):

```
dopredu 10
dopredu 15
dopredu 20
dopredu 25
dopredu 30
```

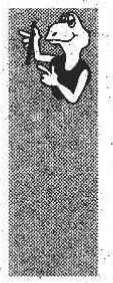
V ďalšom texte budeme väčšinou predpokladať, že na začiatku nejakej úlohy je korytnačka v svojej „počiatočnej" pozícii (hovoríme jej aj domovská pozícia). Dosiahneme to napríklad príkazom **zmaz**. *Tento príkaz nielenže zmaže obsah obrazovky, ale zároveň korytnačku presťahuje do domovskej pozície a nepotrebuje pritom žiadny vstup.*

Pozrime sa podrobnejšie na vstup príkazu **dopredu**: vstupom nemusí byť len celé kladné číslo, ale napríklad aj číslo s desatinnou časťou a dokonca aj záporné číslo. Význam desatinných čísel ako vstupov do príkazu **dopredu** uvidíme neskôr, zatiaľ stačí porozumieť, že korytnačka po vykonaní postupnosti príkazov:

```
dopredu 21.4
dopredu 5.5
dopredu 3.1
```

prejde naozaj dĺžku presne 30 krokov. Záporné čísla ako vstupy sú zaujímavejšie: korytnačka nepôjde dopredu, ale dozadu (bude cúvať).

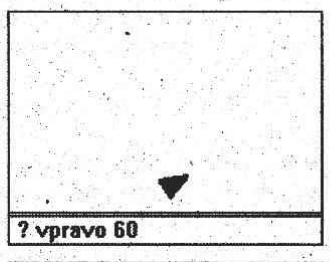
ÚLOHA



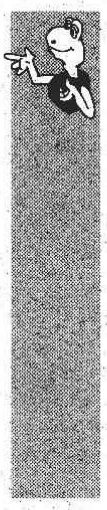
2. Zistite, koľko krokov prejde a akú dlhú čiaru nakreslí táto postupnosť príkazov:

```
dopredu 100
dopredu -90
dopredu 80
dopredu -70
dopredu 60
dopredu -50
```

Ďalšou skupinou príkazov na ovládanie korytnačky sú otočenie **vlavo** a **vpravo** o nejaký počet stupňov. Oba tieto príkazy majú jeden vstup, ktorý určuje počet stupňov otočenia príslušným smerom. Podobne ako príkaz **dopredu**, aj tieto dva nemusia mať len celočíselné vstupy. Korytnačka zvládne aj desatinné čísla a aj záporné uhly.



Na druhom obrázku si môžeme všimnúť, že do príkazového riadka sme za sebou zapísali dva príkazy – vo všeobecnosti ich môže byť aj viac. Treba si dávať pozor na oddeľovanie vstupov od príkazu aspoň jednou medzerou a tiež na to, že medzi dvomi príkazmi v riadku musí byť aspoň jedna medzera.



Cvičenia

1. Zistite vzťah medzi príkazmi **vpravo 50**, **vlavo -50**, **vpravo 410** a **vlavo 310**.

2. Bez zapísania na počítači – len na papieri – zistite, čo nakreslí postupnosť príkazov:
   dopredu 100 vlavo 90 dopredu 40 dopredu -80

3. Zistite, ktoré písmená veľkej abecedy alebo číslice by ste vedeli nakresliť pomocou korytnačích príkazov.

4. Natočte najprv korytnačku o uhol 25 stupňov a urobte dlhú čiaru 10 000 krokov. Vyslovte hypotézu, prečo korytnačka „nevypadla" z grafickej plochy, ale objavila sa niekde inde.

Už sme spomínali, že korytnačka môže mať zdvihnuté pero, a vtedy sa po zadaní príkazu **dopredu** presúva bez kreslenia čiary. Dvojica príkazov (oba sú bez vstupu) **pero.hore** a **pero.dolu** mení stav pera. Pri štarte Loga má korytnačka spustené pero (teda **pero.dolu**), a preto pri presúvaní kreslí čiary. Príkaz **pero.hore** zdvihne pero, napríklad môžeme zapísať:

```
dopredu 20 pero.hore dopredu 30 pero.dolu dopredu 20
```

V tomto príklade sa nakreslí prerušená čiara.

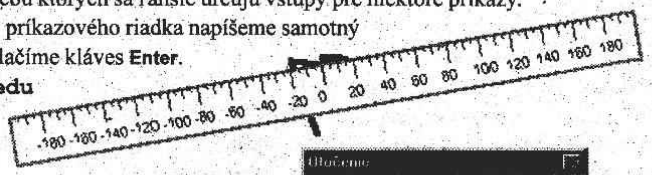
Oboznámili sme sa s niekoľkými základnými príkazmi. Teraz ich zhrnieme do tabuľky, pričom pridáme aj príkaz **vzad**. Hoci sme ho doteraz neuviedli, označuje pre nás známe "cúvanie" (preto platí, že **vzad *dĺžka*** je to isté ako **dopredu *-dĺžka***)

dopredu číslo	**do**	- prejde dopredu
vzad číslo	**vz**	- prejde dozadu
vpravo číslo	**vp**	- otočí sa vpravo
vlavo číslo	**vl**	- otočí sa vľavo
pero.hore	**ph**	- zdvihne pero hore
pero.dolu	**pd**	- spustí pero dole
zmaz		- zmaže obrazovku, korytnačka ide do domovskej pozície

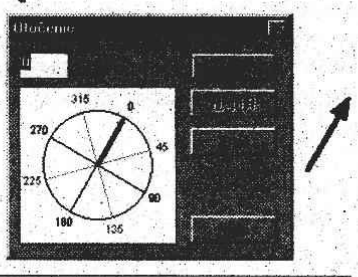
Všimnime si druhý stĺpec tabuľky: sú to preddefinované skratky. Niektoré príkazy Loga majú aj svoj skrátený tvar, vďaka čomu skúsenejší programátor môže rýchlejšie a úspornejšie (niekedy aj menej čitateľne) zapísať postupnosť príkazov. Pokúste sa zistiť, čo nakreslí táto postupnosť príkazov:

```
do 50 vl 30 do 30 vz 30 vp 30 do 30 vz 30 vp 30 do 30
```

Niekedy sa nám môže hodiť to, že Logo v istých situáciách ponúka *pomôcky*: sú to dialógové okná alebo iné grafické časti, pomocou ktorých sa ľahšie určujú vstupy pre niektoré príkazy. Pomôcku vyvoláme tak, že do príkazového riadka napíšeme samotný príkaz bez zadania vstupu a stlačíme kláves **Enter**. Napríklad, pre príkazy **dopredu** alebo **vzad** nám môže pomôcť pravítko:



Pre príkazy **vlavo** a **vpravo** sa objaví takáto pomôcka:



Grafické pero korytnačky môže byť prepnuté na kreslenie ľubovoľnou farbou. Každá farba je v počítači zakódovaná svojím číslom (niekedy trojicou čísel). Napríklad číslo 0 reprezentuje čiernu farbu, 1 tmavomodrú a 15 bielu. Príkaz, ktorý umožní zmeniť farbu grafického pera je

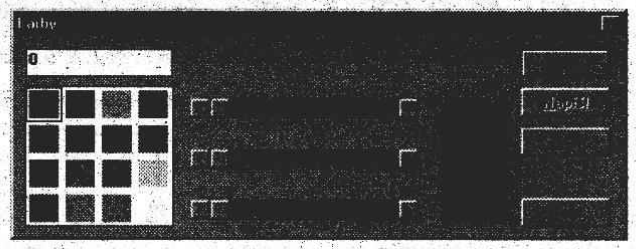
zmen.farbu.pera (alebo skrátene **zmen.fp**)

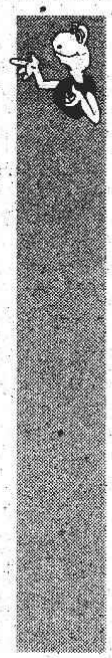
Očakáva jeden vstup, napríklad:

```
zmen.fp 1
```

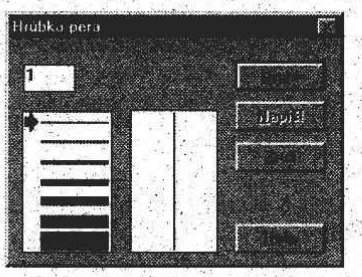
Po nastavení farby pera na tmavomodrú, bude korytnačka kresliť tmavomodré čiary.

Tieto číselné kódy si netreba pamätať, lebo aj v tomto prípade nám Logo pomôže pomôckou: do príkazového riadka zapíšeme len **zmen.fp** a stlačíme kláves **Enter**. Otvorí sa pomôcka, v ktorej si vyberieme požadovanú farbu a stlačíme *Urob!* Kód zvolenej farby sa dopíše ako vstup za meno príkazu.

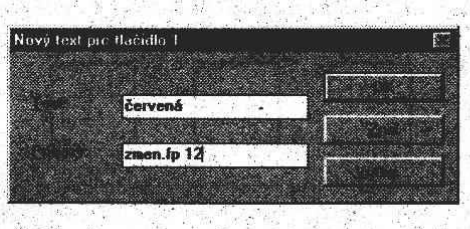


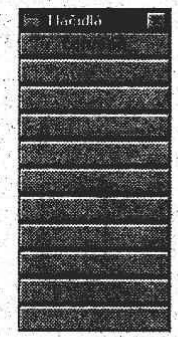


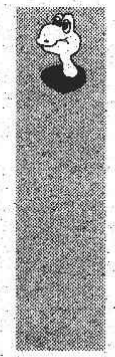
Hrúbkou pera korytnačky môže byť ľubovoľné celé číslo a opäť nám pomôže pomôcka. Príkaz, ktorý mení túto hrúbku je **zmen.hrubku.pera** alebo skrátene **zmen.hp**.



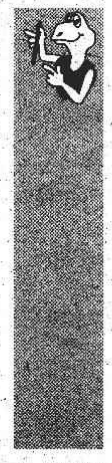
Pri práci v Logu si môžeme prácu zjednodušiť a sprehľadniť použitím tlačidiel. Tlačidlá zobrazíme (a znovu ukryjeme) stlačením klávesu **F10**. *Kliknutím pravým tlačidlom myši na niektoré tlačidlo sa zobrazí dialógové okno. V ňom zadáme text, ktorým bude tlačidlo reprezentované, a príkazy, ktoré sa vykonajú stlačením tohto tlačidla.*



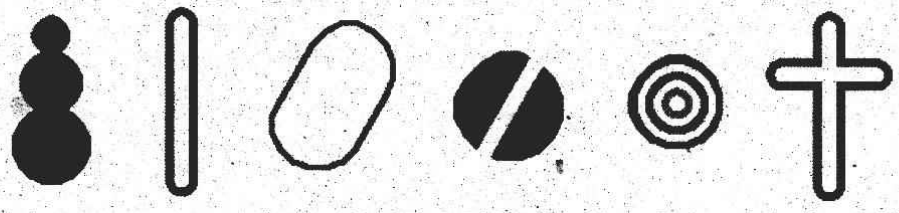




Cvičenie



5. Pomocou hrubých rôzne zafarbených čiar nakreslite tieto obrázky (kruh sa dá nakresliť ako veľmi hrubá veľmi krátka čiara, napr. **zmen.hp 50 dopredu 0**, kružnicu získame nakreslením dvoch rôzne veľkých kruhov so spoločným stredom):



2 Príkaz opakuj

Nakreslime korytnačkou štvorec s dĺžkou strany 100 krokov:

```
dopredu 100 vpravo 90
dopredu 100 vpravo 90
dopredu 100 vpravo 90
dopredu 100 vpravo 90
```

Pri kreslení štvorca nemusíme písať každý riadok znova.
Stlačením šípky hore zobrazíme do príkazového riadka naposledy napísaný riadok. Tento potom môžeme podľa potreby upraviť a stlačením klávesu **Enter** *znova vykonať. Viacnásobné stláčanie šípky hore (podobne i šípky dolu) nám umožňuje prechádzať postupnosťou doteraz vykonaných riadkov s príkazmi.*

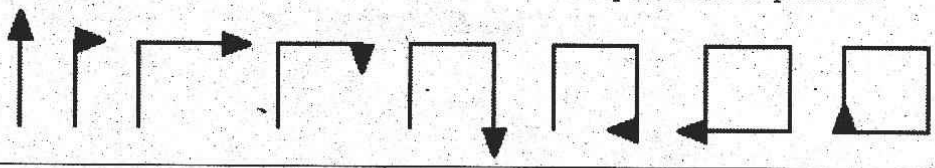
Uvedené riešenie nie je veľmi elegantné. Štyrikrát sa v ňom opakujú tie isté príkazy. Na takéto prípady pozná Logo užitočný príkaz: **opakuj**.

Príkaz **opakuj** vykonáva určený počet-krát postupnosť príkazov, uvedenú v zátvorkách:
opakuj *číslo* **[***postupnosť príkazov***]**
číslo vyjadruje počet opakovaní. V zátvorkách môže byť i viac príkazov.

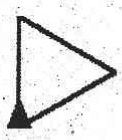
Takže náš štvorec môžeme bez veľkej námahy nakresliť takto:

```
opakuj 4 [dopredu 100 vpravo 90]
```

Ako sa korytnačka správa pri vykonávaní týchto príkazov? Prejde dopredu 100 krokov a otočí sa vpravo o 90 stupňov. Potom príkazy zopakuje: **dopredu 100, vpravo 90**. Tretíkrát **dopredu 100, vpravo 90**, a nakoniec dokončí štvorec: **dopredu 100** a **vpravo 90**.

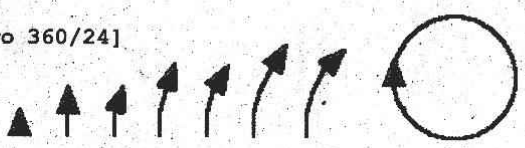


Príklad 1. Nakreslime príkazom **opakuj** rovnostranný trojuholník s dĺžkou strany 150 krokov.
Riešenie:
Trikrát sa tu zopakujú dva príkazy: **dopredu** a **vpravo**. Korytnačka prejde najskôr dĺžku 150 krokov, a potom sa otočí o nejaký uhol. V tomto prípade sa musí otočiť o vonkajší uhol rovnostranného trojuholníka, teda o 120 stupňov. Túto postupnosť príkazov zopakuje celkovo trikrát.
Riešením je preto príkaz **opakuj 3 [dopredu 150 vpravo 120]**

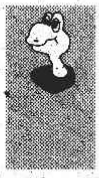


Ďalším zaujímavým geometrickým útvarom je kružnica.
V korytnačej grafike kružnicu kreslíme ako pravidelný mnohouholník, napríklad 24-uholník. Pri jeho kreslení korytnačka prejde dopredu napríklad 20 krokov a otočí sa o $\frac{360}{24}$ stupňov (viete prečo?). Znovu prejde dopredu 20 krokov, otočí sa... . Celkovo 24-krát zopakuje tieto dva príkazy, čo zapíšeme:

```
opakuj 24 [dopredu 20 vpravo 360/24]
```



Pri kreslení kružnice potrebujeme korytnačku otáčať o $\frac{360}{24}$ stupňov. Nemusíme sa namáhať počítaním ani hľadaním kalkulačky. Logo si poradí so zápisom **vpravo 360/24**. *Najskôr vypočíta hodnotu výrazu $\frac{360}{24}$, a až potom sa korytnačka otočí vpravo o uhol s vypočítanou veľkosťou. Podobne sa výrazy (i zložitejšie) používajú ako vstupy aj v ostatných príkazoch:* **vlavo**, **dopredu**, **vzad**. *Môžeme ich použiť i v príkazoch* **zmen.fp**, **zmen.hp** *a* **opakuj**. *Ak je totiž hodnotou výrazu desatinné číslo, ako vstup sa použije iba jeho celá časť.*



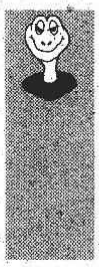
Príklad 2. Príkazom **opakuj** nakreslíme oko.

Riešenie:

Obrázok oka sa skladá z troch rôzne veľkých kružníc. Kružnicu sme kreslili ako 24-uholník so stranou dĺžky 20 bodov. Nakreslíme ju znova:

```
opakuj 24 [dopredu 20 vpravo 15]
```

Teraz potrebujeme ďalšiu, tentoraz menšiu kružnicu. Zmenšíme preto dĺžku strany 24-uholníka o polovicu – i kružnica zmenší svoj polomer o polovicu: **opakuj 24 [do 10 vp 15]**. Pri najmenšej kružnici zmenšíme dĺžku strany ešte viac: **opakuj 24 [do 5 vp 15]**.

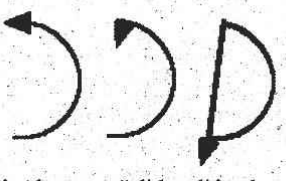


Príklad 3. Nakreslime korytnačkou písmeno D.

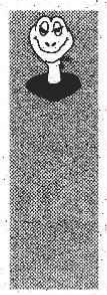
Riešenie:

Na zobrazenie písmena D potrebujeme nakresliť polkružnicu. Kružnicu sme nakreslili pomocou 24-uholníka. Ak z neho zostrojíme len 12 strán, máme polkružnicu:

```
opakuj 12 [dopredu 10 vlavo 15]
```

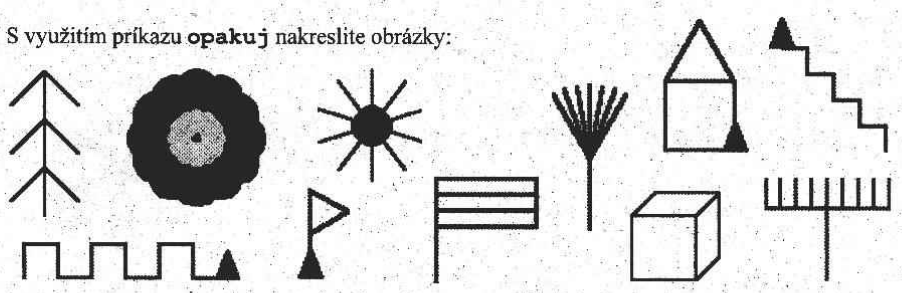


Teraz už musí korytnačka iba prejsť do bodu, z ktorého začala kresliť. Ak sme začali kresliť z domovskej pozície korytnačky, stačí použiť príkaz **domov**. V opačnom prípade použijeme pomôcky príkazov **vlavo** a **dopredu**.



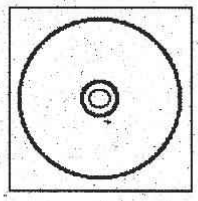
Cvičenia

1. S využitím príkazu **opakuj** nakreslite obrázky:



2. Nakreslite pravidelný 5-, 6-, a 8-uholník.

3. Nakreslite obrázok disku.
Vedeli by ste nakresliť sústredné kruhy aj bez použitia príkazu **opakuj**?



4. Nakreslite písmená abecedy: G, J, O, C, B, R, ...

5. Nakreslite takéto obrázky:

V geometrii často používame prerušované čiary. V Logu môžeme takúto čiaru nakresliť tak, že prejdeme korytnačkou časť čiary s perom dolu, potom časť s perom hore, potom zasa s perom dolu a časť s perom hore.... Teda napríklad **opakuj 6 [do 5 ph do 5 pd]**.

Skúsme teraz nakresliť takouto prerušovanou čiarou štvorec:

```
opakuj 6 [do 5 ph do 5 pd]
vpravo 90
opakuj 6 [do 5 ph do 5 pd]
vpravo 90
```

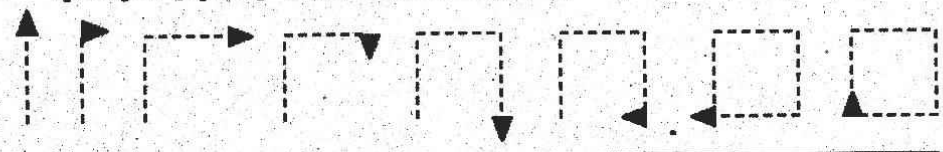
... a znovu to isté! V tomto prípade štyrikrát opakujeme dva príkazy:

opakuj 6 [do 5 ph do 5 pd] a **vpravo 90**

Naozaj iba dva príkazy. Hoci je príkaz **opakuj** zložený z viacerých príkazov, chápeme ho ako jeden samostatný príkaz. V našom príklade ako príkaz, ktorý nakreslí prerušovanú čiaru dĺžky 60 krokov.

Požadovaný švorec potom nakreslíme takto:

```
opakuj 4 [opakuj 6 [do 5 ph do 5 pd] vp 90]
```



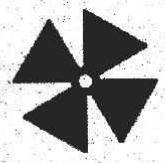
Príkaz **opakuj** môže obsahovať v postupnosti príkazov i ďalší príkaz **opakuj**:
opakuj počet1 [. opakuj počet2 [postupnosť príkazov] . .]
Každý príkaz **opakuj** musí mať svoj počet opakovaní a svoju postupnosť príkazov, ktoré má vykonať, uzavretú v hranatých zátvorkách [ ].

Príklad 4. Nakreslime vrtuľu veterného mlyna.

Riešenie:

Tento obrázok sa skladá zo štyroch rovnostranných trojuholníkov, ktoré majú spoločný vrchol. Pri ich kreslení si stačí uvedomiť, že sú navzájom pootočené o rovnaký uhol $\frac{360}{4}$ stupňov. Preto po nakreslení jedného trojuholníka príkazom **opakuj 3 [do 50 vp 120]** otočíme korytnačku **vpravo 360/4**. Celé to zopakujeme štyrikrát:

```
opakuj 4.[opakuj 3 [do 50 vp 120] vp 360/4]
```



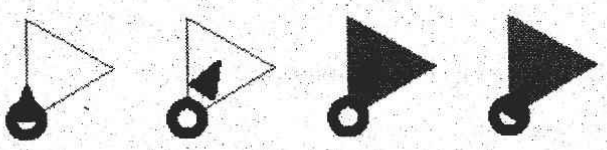
Vnútornú časť nakreslíme príkazmi:

```
zmen.fp 0 zmen.hp 20 bodka
zmen.fp 15 zmen.hp 10 bodka
```

Na vyfarbenie trojuholníkov použijeme príkaz **vypln**. Týmto príkazom korytnačka vyplní svoje okolie ohraničené inými farbami, ako je farba bodu, na ktorom stojí. Farbu výplne môžeme nastaviť príkazom **zmen.farbu.vyplne číslo** (krátko **zmen.fv**). Kým nepoužijeme príkaz **zmen.fv**, korytnačka vypĺňa farbou pera. Zmenou farby pera sa zmení i farba výplne.

Vyfarbíme jeden trojuholník:

```
zmen.fv 6 ph
vp 30 do 15
vypln
vz 15
```



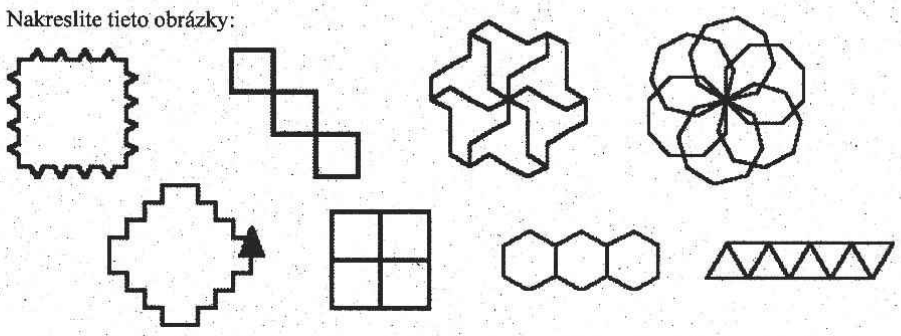
... a všetky štyri trojuholníky:

```
zmen.fv 6 ph vp 30
opakuj 4 [do 15 vypln vz 15 vp 360/4]
```

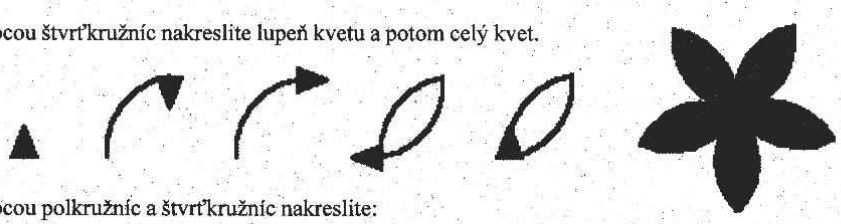
Cvičenia

6. Nakreslite prerušovanou (alebo bodkočiarkovanou či bodkovanou) čiarou obrázky z predchádzajúcich cvičení.

7. Nakreslite tieto obrázky:



8. Pomocou štvrťkružníc nakreslite lupeň kvetu a potom celý kvet.



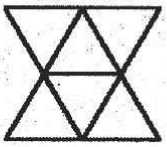
9. Pomocou polkružníc a štvrťkružníc nakreslite:

Ďalšie úlohy

1. Bez toho, aby ste nasledujúce príkazy programovali, pokúste sa zistiť, čo nakreslia. Potom príkazy napíšte a overte tak svoje riešenia:

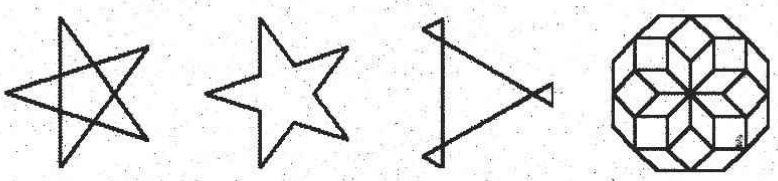
```
opakuj 90 [do 180 vp 178]
opakuj 18 [do 50 vp 45 vz 20 vl 25]
```

2. Nasledujúcou postupnosťou príkazov by mala korytnačka nakresliť tento obrazec. Nahraďte bodky správnym príkazom.



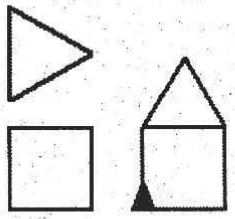
```
zmaz vl 30
opakuj 2 [do 50 vp 120 ~
          opakuj 2 [do 100 vp 120] ~
                    do 50 ..................]
do 50
```

3. Nakreslite:



3 Definujeme vlastné príkazy

Naučili sme sa pomocou príkazu **opakuj** elegantne kresliť štvorec a rovnostranný trojuholník:
štvorec: **opakuj 4 [dopredu 50 vpravo 90]**
trojuholník: **opakuj 3 [dopredu 50 vpravo 120]**



Z týchto tvarov chceme zložiť a nakresliť najskôr jeden domček a potom celý rad domov. Najskôr domček:

```
opakuj 4 [dopredu 50 vpravo 90]
dopredu 50 vpravo 30
opakuj 3 [dopredu 50 vpravo 120]
vlavo 30 vzad 50
```

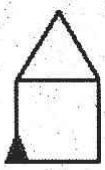
Keby sme chceli nakresliť celý rad domov, posúvali by sme teraz korytnačku po ploche a vždy, keď by sme chceli nakresliť dom, museli by sme znovu napísať toto množstvo príkazov. Ani sa o to nepokúšajte! Logo nám totiž ponúka krajšie riešenie. Naučíme korytnačku nový príkaz (procedúru), ktorý nakreslí jeden domček, a tento príkaz budeme potom používať ako hociktorý iný príkaz. Príkaz **nauc.sa** má takýto tvar:

```
nauc.sa meno.nového.príkazu
  postupnosť príkazov
koniec
```

Meno.nového.príkazu je slovo, ktoré nesmie obsahovať medzeru. Ak sa meno skladá z viacerých slov, obyčajne sa namiesto medzery používa bodka.

Ako teda zadefinujeme nový príkaz, ktorý nakreslí domček? Do príkazového riadka napíšeme: **nauc.sa nakresli.dom** a stlačíme **Enter**.
Po tomto úkone sa Logo dostane do režimu definovania príkazov: zmení sa výzva **?** na znak **>**. Príkazy, ktoré budeme teraz zapisovať sa nebudú vykonávať, ale budú tvoriť telo príkazu. Režim definovania príkazov ukončíme zapísaním príkazu **koniec**.

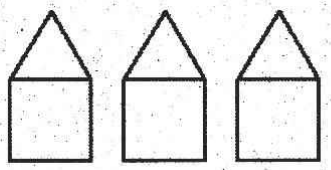
```
? nauc.sa nakresli.dom
> opakuj 4 [dopredu 50 vpravo 90]
> dopredu 50 vpravo 30
> opakuj 3 [dopredu 50 vpravo 120]
> vlavo 30 vzad 50
> koniec
```



Definovali sme príkaz **nakresli.dom**. Odteraz vždy, keď budeme chcieť nakresliť domček, stačí do príkazového riadka zapísať nový príkaz **nakresli.dom**.

Takto zadefinovaný príkaz môžeme zrušiť príkazom **zrus "meno.príkazu**
Pozor: *v tomto príkaze musia byť pred menom príkazu, ktorý chceme zrušiť, úvodzovky a medzi nimi a menom príkazu nesmie byť medzera. Neskôr si ukážeme aj jednoduchší spôsob rušenia nami definovaných príkazov.*

Príklad 1. Nakreslime rad troch domov.
Riešenie:



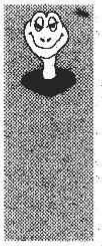
Máme nakresliť tri domy v pravidelných odstupoch vedľa seba. V jednom kroku príkazu **opakuj** preto nakreslíme dom naším príkazom **nakresli.dom**, so zdvihnutým perom presunieme korytnačku do začiatočného bodu ďalšieho domu, správne ju natočíme a toto celé zopakujeme celkovo trikrát:

```
opakuj 3 [nakresli.dom vp 90 ph do 70 pd vl 90]
```

Príklad 2. Napíšme príkazy, ktoré nakreslia štvorec a rovnostranný trojuholník.
Riešenie:

```
? nauc.sa stvorec
> opakuj 4 [dopredu 50 vpravo 90]
> koniec

? nauc.sa trojuholnik
> opakuj 3 [dopredu 50 vpravo 90]
> koniec
```



Pozrime sa ešte raz pozorne na príkaz **nakresli.dom**. Príkazmi **opakuj** v ňom nakreslíme najskôr štvorec, potom trojuholník. Práve sme zadefinovali príkazy **stvorec** a **trojuholnik**, ktoré ich nakreslia za nás. Oba príkazy **opakuj** preto môžeme nahradiť týmito novými príkazmi. Skúsme to:

```
? nauc.sa nakresli.dom
CHYBA: Príkaz nakresli.dom už existuje, použi preto príkaz EDIT
```

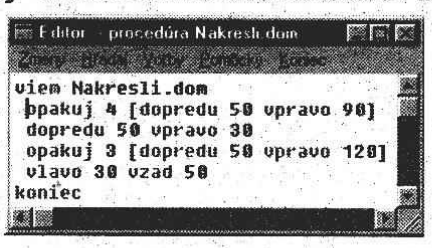
Príkazom **nauc.sa nakresli.dom** sa vlastne snažíme znovu zadefinovať príkaz, ktorý už definovaný je. Chybová správa nám však ponúka riešenie. Na editovanie vlastných príkazov slúži príkaz **edituj** (skratka **ed**):

edituj "*meno.prikazu*

Ak ako ***meno.prikazu*** napíšeme meno už nami definovaného príkazu, otvorí sa textový editor príkazov a v ňom tento príkaz.
Ak zadáme nové meno, najskôr sa tento príkaz vytvorí (nebude obsahovať žiadne príkazy) a potom sa otvorí editor.

Základné príkazy jazyka Logo nemožno editovať.
V príkaze **edituj** nesmieme zabudnúť napísať pred meno príkazu úvodzovky.

Opravme teraz náš príkaz **nakresli.dom**. Do príkazového riadka zapíšeme: **edituj "nakresli.dom**. Otvorí sa editor a v ňom náš príkaz:

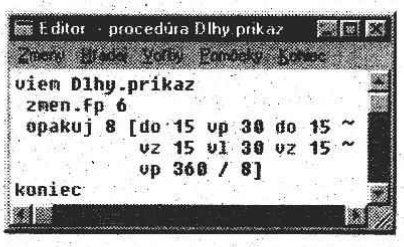


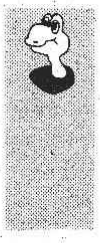
Teraz ho môžeme opraviť:

```
viem nakresli.dom
  stvorec
  dopredu 50 vpravo 30
  trojuholnik
  vlavo 30 vzad 50
koniec
```

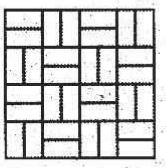
Editovanie ukončíme kliknutím myšou na položku **Koniec** v menu, alebo stlačením klávesu **F12**. Ak sa v zápise nami definovaného príkazu vyskytne syntaktická chyba, program nás na to upozorní a nedovolí ukončiť editovanie, pokým ju neodstránime.

*Pri definovaní nových príkazov budeme často používať príkazy, ktorých dĺžka presiahne dĺžku editovacieho okna. Väčšiu prehľadnosť dosiahneme rozdelením takéhoto dlhého príkazu na viac riadkov. Aby bolo jasné, že ide stále o jeden príkaz, na koniec každého riadka takto rozdeleného príkazu **musíme**, ako rozdeľovací znak, zapísať znak vlnovku ~.*





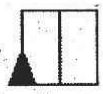
Príklad 3. Nakreslime tento ornament.



Riešenie:

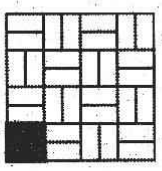
Keď sa pozorne zahľadíme na obrázok, zistíme, že ho môžeme rozdeliť na niekoľko rovnakých častí, ktoré sa v ornamente opakujú. Stačí nám teda napísať príkaz, ktorý nakreslí túto malú časť obrázka, a potom v inom príkaze už iba určiť, ako bude táto základná časť v obrázku poukladaná.

Základným stavebným prvkom v celej mozaike je tento tvar:



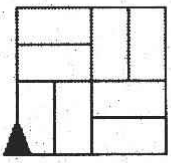
Zvoľme dĺžku strany štvorca 40. Príkaz, ktorý ho nakreslí, bude vyzerať takto:

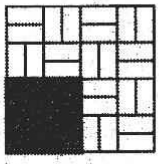
```
viem kamen
  opakuj 4 [do 40 vp 90]
  vp 90 do 20 vl 90 do 40 vz 40 vp 90 vz 20 vl 90
koniec
```



Z kameňa môžeme teraz zostaviť väčšiu časť ornamentu. Nazvime ju **vzor**:

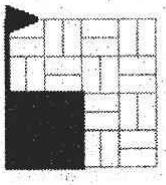
```
viem vzor
  opakuj 4 [kamen ~
            do 80 vp 90]
koniec
```

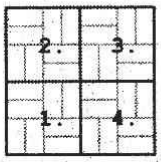


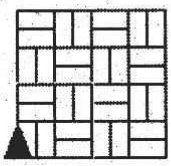


A konečne záver nášho snaženia:

```
viem ornament
  opakuj 4 [vzor ~
            do 160 vp 90]
koniec
```

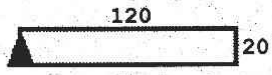






Cvičenia

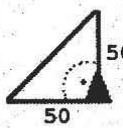
1. Definujte príkazy, ktoré budú kresliť nasledujúce tvary:



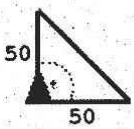
obdlznik

kruznica

polkruh



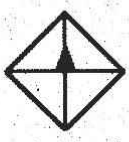
pravy.p



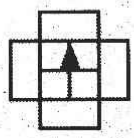
pravy.1

Pri posledných dvoch pravouhlých trojuholníkoch z obrázka nevidieť dĺžku prepony. Vieme ju ľahko vypočítať z Pytagorovej vety. Ani teraz nemusíme siahať po kalkulačke, Logo si poradí i s odmocninou. Slúži na to operácia **sqrt**. *Preponu nakreslíme príkazom* **do sqrt 50*50 + 50*50**, *resp.* **do 50*sqrt 2**.

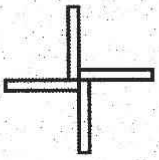
2. Pri definovaní príkazov, ktoré kreslia nasledujúce obrázky, porozmýšľajte o vhodnom použití doposiaľ definovaných príkazov.



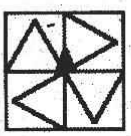
stvorce1



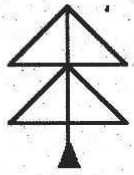
stvorce2



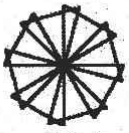
obdlzniky



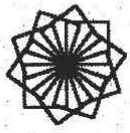
kriz



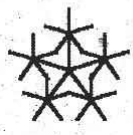
strom



trojuholniky

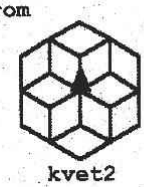


kvet1



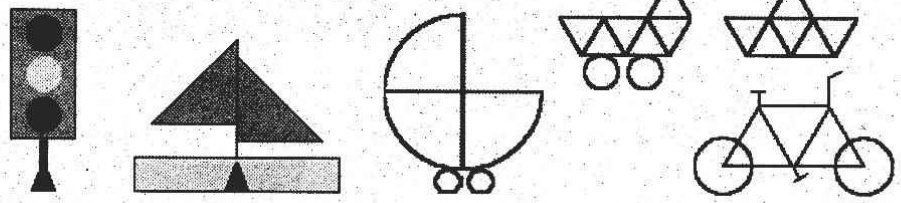
vlocka

kocky



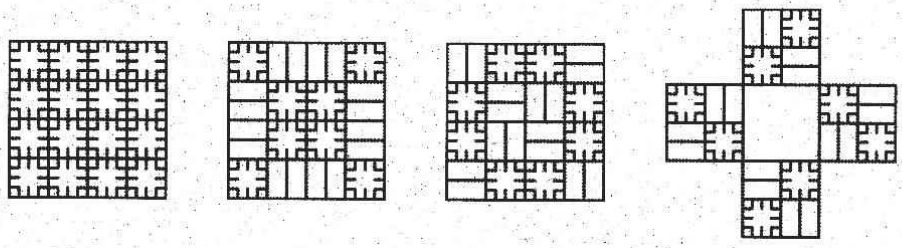
kvet2

3. Napíšte príkazy, ktoré nakreslia tieto obrázky:



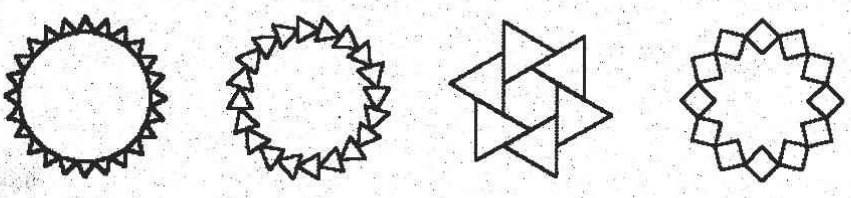
Ďalšie úlohy

1. Napíšte príkaz, ktorý nakreslí jeden z dvoch základných kameňov týchto skladačiek (druhý už máte zapísaný v príklade 3). Potom zadefinujte príkazy, ktoré ornament nakreslia:



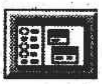
2. Pokúste sa vymyslieť iný, vlastný ornament.

3. Zapíšte príkazy, ktoré nakreslia tieto obrázky:

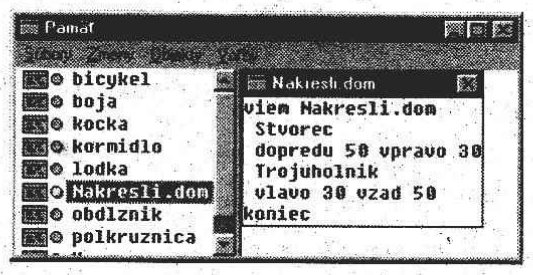


`slnko` `veniec` `hviezda` `kvet3`

Všetky nami definované príkazy si môžeme prezerať v okne Pamäť. Otvoríme ho stlačením klávesu **F4**, *alebo kliknutím na štvrté tlačidlo v hornom páse tlačidiel. Zatvoríme ho opätovným stlačením klávesu* **F4**, *alebo kliknutím na to isté tlačidlo.*



Dvojkliknutím na meno procedúry, alebo stlačením klávesu **Enter** *sa telo označeného príkazu rozbalí v pravej časti okna. Ak chceme príkaz editovať, dvojklikneme naň v pravej časti okna (alebo stlačíme kláves* **F11**).

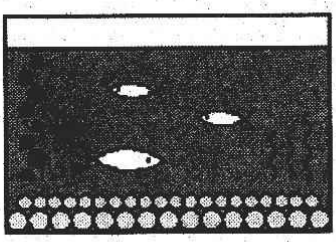


V okne pamäť môžeme tiež príkaz zrušiť, alebo pridať nový. Ak chceme niektorý príkaz zrušiť, označíme ho a v menu okna v položke Objekty vyberieme Zruš. Nový príkaz pridáme pomocou Pridaj procedúru v položke Objekty. Do priameho režimu sa vrátime zatvorením okna Pamäť.

4 Príkazy so vstupmi

Precvičme si, čo sme sa doteraz naučili. Nakreslime si takéto akvárium plné rýb:
Nádrž: **opakuj 2 [do 200 vp 90 do 300 vp 90]**
Spodný riadok kamienkového dna sú kružnice so stranou dĺžky 3:

```
viem kruznica3
  opakuj 24 [do 3 vp 15]
koniec
```



Horný riadok dna sú kružnice so stranou dlhou 2 kroky:

```
viem kruznica2
  opakuj 24 [do 2 vp 15]
koniec
```

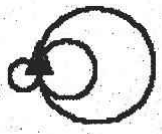
Predchádzajúce príkazy sa od seba líšia iba číslom, ktoré vyjadruje dĺžku strany 24-uholníka. Príkaz, ktorý nakreslí kružnicu so stranou dĺžky 1, by sa od nich odlišoval opäť iba dĺžkou: konštantu 2 by sme prepísali na 1.
Namiesto definovania takmer rovnakých príkazov zadefinujeme jeden všeobecnejší príkaz, ktorý bude kresliť rôzne veľké kružnice. V ňom namiesto konkrétnej číselnej konštanty použijeme *parameter*, ktorý bude predstavovať dĺžku strany: **opakuj 24 [do :dlzka vp 15]**.
Každý príkaz používajúci parameter musí mať tento parameter uvedený i v hlavičke za menom príkazu.

```
viem kruznica :dlzka
  opakuj 24 [dopredu :dlzka vpravo 15]
koniec
```

Meno parametra je **dlzka**, dvojbodka pred ním znamená, že pracujeme s jeho hodnotou. Preto je nutné napísať dvojbodku tesne pred meno parametra.

Konkrétnu hodnotu parametra príkazu určíme pri volaní tohto príkazu. Napríklad volaním **kruznica 3** nadobudne parameter **dlzka** hodnotu 3 a korytnačka nakreslí kružnicu s dĺžkou strany 3. Môžeme si tiež predstaviť, že číslo 3 „vstupuje" do príkazu kružnica, budeme preto častejšie namiesto pojmu parameter hovoriť o *vstupe* do príkazu.

Hodnotou vstupu do príkazu **kruznica** môže byť ľubovoľné číslo:



```
kruznica 1.5
kruznica -1
```

Nezabudnite, že medzi menom príkazu a hodnotou vstupu musí byť medzera!
Viete, aký je rozdiel medzi volaním **kruznica 3** a **kruznica3**?

Keďže sme zvyknutí určovať kružnicu jej stredom a polomerom, ukážeme si ešte iné riešenie, v ktorom bude korytnačka kresliť kružnicu zo stredu a vstupom do príkazu bude dĺžka polomeru kružnice:

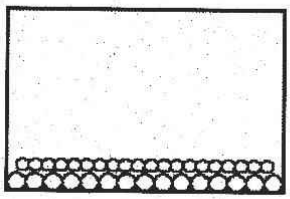
```
viem kruznica.r :polomer
  ph do :polomer vp 90 vp 7.5 pd
  opakuj 24 [do (2*3.14*:polomer)/24 vp 15]
  ph vl 7.5 vl 90 vzad :polomer pd
koniec
```

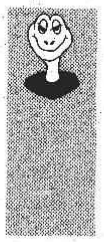
} dĺžka kružnice / 24

Príklad 1. Napíšme príkaz, ktorý nakreslí kamienkové dno akvária:

Riešenie:

```
viem kamene
  ph do 10 vp 90 do 10
  opakuj 15 [kruznica.r 10 ph do 20]
  ph vl 90 do 17 vl 90 do 27
  opakuj 20 [kruznica.r 7 ph do 14]
koniec
```





Príklad 2. Napíšme príkaz **ryba** so vstupom **velkost**, ktorý bude kresliť ryby rôznej veľkosti.

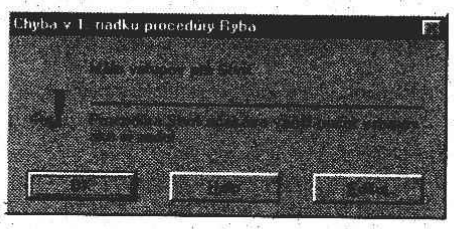
Riešenie:

Obrázok ryby sa skladá z dvoch rovnako veľkých štvrťkružníc. Zadefinujeme preto najskôr príkaz **stvrt** so vstupom **krok**, ktorý nakreslí štvrťkružnicu ako časť 24-uholníka.
Veľkosť štvrťkružnice určí hodnota vstupu **krok**.

```
viem stvrt :krok
  opakuj 6 [dopredu :krok vpravo 15]
koniec
```

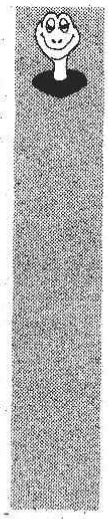
Pomocou príkazu **stvrt** teraz nakreslíme rybu:

```
viem ryba :velkost
  vp 45  stvrt
  vp 90 + 20
  stvrt  vp 25
koniec
```



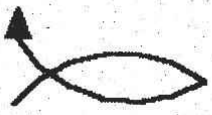
Zadajme tento príkaz v príkazovom riadku:

```
? ryba 20
Chyba v 1. riadku procedúry ryba: Málo vstupov pre stvrt
```



Pri volaní príkazu **stvrt** sme zabudli určiť hodnotu jeho vstupu **krok**. Keďže veľkosť ryby vyjadruje parameter **velkost**, jeho hodnota určuje tiež veľkosť štvrťkružnice. Vstupom do príkazu **stvrt** bude preto priamo hodnota parametra **velkost**.

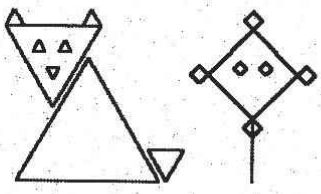
```
viem ryba :velkost
  vp 45  stvrt :velkost
  vp 90 + 20
  stvrt :velkost  vp 25
koniec
```

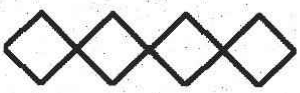


Cvičenia

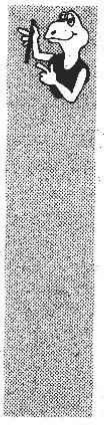
1. Napíšte príkazy **stvorec** a **trojuholnik** – oba so vstupom **strana**. Potom pomocou nich nakreslite tieto obrázky.

2. Využite definované príkazy pri kreslení týchto obrázkov:





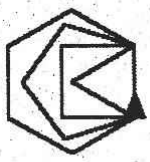
3. Napíšte príkaz **stvorce** so vstupom **pocet**, ktorý nakreslí daný počet štvorcov so stranou 50 krokov. Štvorce sú otočené o 45 stupňov, treba ich nakresliť jedným ťahom (po každej čiare môže korytnačka prejsť len raz).



Príklad 3. Naprogramujme príkaz, ktorý nakreslí ľubovoľný *n*-uholník so stranou 50 krokov, pričom **n** bude vstupom do tohto príkazu.

Riešenie:

```
viem n.uholnik :n
  opakuj :n [do 50 vl 360 / :n]
koniec
```

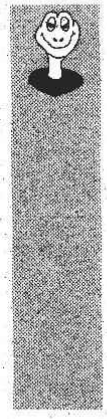


```
n.uholnik 3
n.uholnik 4
n.uholnik 5
n.uholnik 6
```

Príkaz **n.uholnik** so vstupom **n**, ktorý sme zadefinovali v predchádzajúcom príklade, síce umožňuje nakresliť rôzne pravidelné *n*-uholníky, ale vždy s rovnakou dĺžkou strany. Na to, aby sme ním mohli kresliť ľubovoľne veľké útvary, použijeme ďalší vstup, ktorý bude určovať dĺžku strany.

```
viem n.uholnik :n :strana
  opakuj :n [do :strana vl 360 / :n]
koniec
```

Vo všeobecnosti môže mať nami zadefinovaný príkaz ľubovoľný počet vstupov. Pri viacerých vstupoch však musíme dávať dobrý pozor na to, čo ktorý označuje. Preto je výhodné pomenovať ich zmysluplnými (i keď dlhšími) menami, ktoré vyjadrujú význam ich hodnôt.

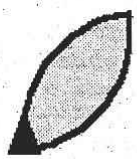


Príklad 4. Zadefinujme príkaz **lupen**, ktorý nakreslí jeden lupeň kvetu. Príkaz bude mať dva vstupy: **velkost** – bude označovať veľkosť a **farba** – farbu lupeňa.

Riešenie:

V riešení využijeme už známy príkaz **stvrt**, ktorý kreslí štvrťkružnicu:

```
viem lupen :velkost :farba
  opakuj 2 [stvrt :velkost vp 90]
  ph vp 45 do 3
  zmen.fv :farba
  vypln vz 3 vl 45 pd
koniec
```

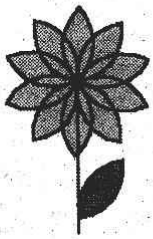


Pri volaní tohto príkazu musíme dávať pozor na poradie, v akom sú zapísané jeho vstupy. Aký bude rozdiel vo výstupoch po vykonaní **lupen 15 2** a **lupen 2 15**? Kým výsledkom prvého volania bude veľký lupeň vyplnený zelenou farbou (**velkost** =15, **farba** =2), druhé volanie nakreslí maličký lupeň vyfarbený bielou farbou (**velkost** = 2, **farba** = 15).

Príklad 5. Napíšme príkaz, ktorý nakreslí kvet. Ako vstupy do tohto príkazu zadefinujme veľkosť, farbu a počet lupeňov.

Riešenie:

```
viem kvet :velkost :farba :pocet
  opakuj :pocet [lupen :velkost :farba ~
                 vp 360 / :pocet]
  vz 10 * :velkost lupen :velkost 2 vz :velkost
koniec
```



Malou modifikáciou príkazu **n.uholnik** získame jeden veľmi zaujímavý príkaz:

```
viem poly :n :strana :uhol
  opakuj :n [do :strana vp :uhol]
koniec
```

Mnohé útvary, ktoré sme kreslili, sa dajú nakresliť jediným volaním príkazu **poly**:
štvorec: **poly 4 50 90**
kružnica: **poly 24 10 15**.

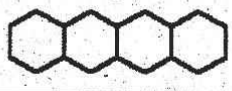
Vhodná voľba vstupov do príkazu **poly** nám tiež môže zjednodušiť kreslenie obrázkov zložených z jednoduchých tvarov.

Napríklad dom:

```
poly 5 50 90
poly 3 50 -120
```

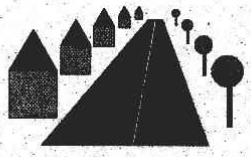
alebo rad šesťuholníkov:

```
vp 60
opakuj 4 [poly 8 20 60 vl 120]
```



Cvičenia

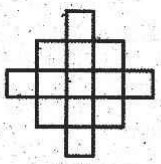
4. Nakreslite obrázky:



ulica

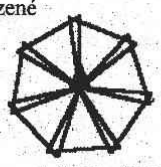
netopier

list

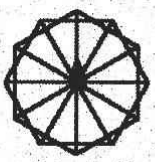


kriz

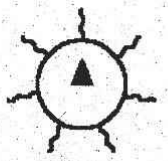
5. Nakreslite zobrazené útvary:



trojuholniky 7

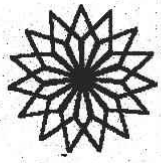


trojuholniky 12

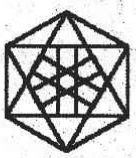


slnko

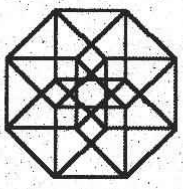
kormidlo



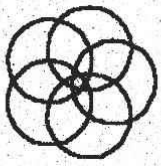
ruza



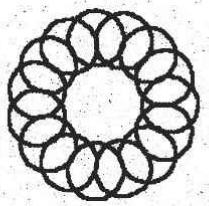
stvorce 6



stvorce 8



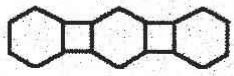
kruhy 5



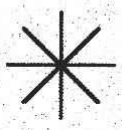
kruhy 15

6. Naprogramujte tieto obrázky použitím čo najmenšieho počtu príkazov. Využite pritom príkaz **poly**.

p.kvet

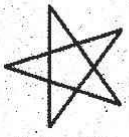


opasok



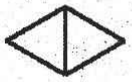
vlocka

lopta



hviezda

7. Napíšte príkaz **koso** so vstupom **strana**, ktorý nakreslí kosoštvorec aj s jednou svojou uhlopriečkou jedným ťahom. Môžete pritom použiť iba príkaz **poly**.

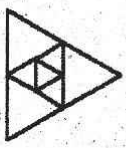


Ďalšie úlohy

1. Napíšte príkaz **siet :m :n :s**, ktorý nakreslí štvorcovú sieť s rozmermi $m \times n$ s dĺžkou strany jedného štvorca s.

2. Dodefinujte príkaz tak, aby po vykonaní nakreslil obrázok jedným ťahom. Strana najväčšieho trojuholníka má dĺžku 100, menšieho 50 a najmenšieho 25 krokov.

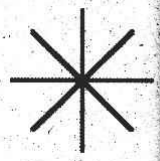
```
prikaz trojuholniky
  opakuj 3 [.............]
koniec
```



5 Definujeme operácie

Vieme definovať a používať príkazy, ktoré pracujú so vstupmi. Naprogramujme napríklad príkaz, ktorý nakreslí hviezdu s daným počtom ramien:

```
viem hviezda :pocet
  opakuj :pocet [do 100 vz 100 vp 360 / :pocet]
koniec
```



Hviezdu na obrázku nakreslíme príkazom **hviezda 8**. Vstupom príkazu však nemusí byť iba konštanta. Môže ním byť i náhodne vygenerované číslo. Získame ho použitím procedúry **nahodne**. Ak tejto procedúre ako vstup dáme celé kladné číslo **n**, na výstupe dostaneme celé číslo z intervalu $\langle 0, n-1 \rangle$. Napríklad volaním **nahodne 4** dostaneme jedno z čísel 0, 1, 2 alebo 3.

Ak Logo spustíme viackrát za sebou, zistíme, že výsledkom operácie **nahodne** *sú rovnaké postupnosti čísel. Aby sme získali vždy iné čísla, musíme generátor náhodných čísel ešte pred prvým použitím operácie* **nahodne** *zamiešať príkazom* **nahodne.nahodne**.

Príklad 1. Zvoľme vstup do príkazu **hviezda** tak, aby nakreslil hviezdu so 4, 5 alebo 6 ramenami.
Riešenie:
Vykonaním procedúry **nahodne 3** získame náhodným výberom celé číslo z intervalu $\langle 0, 2 \rangle$. Ak ku každému z týchto čísel pričítame číslo 4, dostaneme číslo z intervalu $\langle 4, 6 \rangle$. Riešením nášho príkladu je preto volanie **hviezda 4 + nahodne 3**. V ňom najskôr operácia **nahodne 3** vygeneruje číslo, k tomuto číslu Logo pričíta 4 a až tento výsledok je vstupom príkazu **hviezda**.

Aj keď sa zdá, že volanie **hviezda nahodne 3+4** *má rovnaký efekt ako volanie v našom riešení, nie je to pravda. Operácia sčítania má vyššiu prioritu ako operácia* **nahodne**. *Preto sa najskôr vyhodnotí výraz* 3 + 4 *a až jeho hodnota sa stane vstupom operácie* **nahodne**. *Číslo z intervalu* $\langle 0, 2 \rangle$ *získame takto:* **(nahodne 3) + 4**.

Procedúry, ktoré majú výstup (dávajú ako výsledok nejakú hodnotu) nazývame *operácie*. Operáciami sú napríklad tieto základné procedúry Loga:

hrubka.pera – na výstupe dostaneme ako celé číslo nastavenú hrúbku pera korytnačky
farba.pera – výsledkom operácie je aktuálna farba pera korytnačky reprezentovaná celým číslom z intervalu $\langle 0, 15 \rangle$, kde 0 predstavuje čiernu farbu, 1 modrú atď.
farba.vyplne – výsledkom je farba podľa nastavenia farby výplne

Zistime aktuálnu farbu pera korytnačky:

```
? farba.pera
CHYBA: Neviem, čo robiť s 0
```

Operácie nikdy nepoužívame ako samostatné príkazy, pretože nám dávajú na výstupe nejakú hodnotu, s ktorou musíme niečo urobiť. Najčastejšie túto hodnotu používame ako vstup do inej procedúry, tak ako sme to urobili s operáciou **nahodne** pri kreslení hviezdy. Ak chceme iba zistiť hodnotu operácie (v našom prípade farbu pera), môžeme ju vypísať do textovej plochy príkazom **zobraz** (krátko **zo**). Uvedomte si, že aj v tomto prípade je výsledok operácie vstupnou hodnotou príkazu **zobraz**.

```
? zobraz farba.pera
0
```

Cvičenia

24uholnik

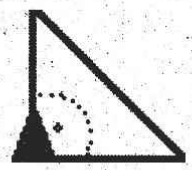
1. Nakreslite 24-uholník podľa obrázka. Každá jeho strana je nakreslená inou farbou a hrúbkou pera. (Návod: **zmen.hp hp + 1**).

2. Napíšte príkaz, ktorý nasimuluje hádzanie kockou. Bude generovať celé čísla z intervalu $\langle 1, 6 \rangle$.

3. Pomocou príkazu **hviezda** nakreslite niekoľko hviezd s náhodným počtom ramien (5, 6 alebo 7) s náhodnou dĺžkou (jedno z čísel 30, 50, 70).

4. Napíšte príkaz, ktorý nakreslí náhodný počet hviezd s rôznym počtom ramien a rôznej veľkosti na náhodné miesta na ploche. (Návod: skôr ako nakreslíte hviezdu, otočte korytnačku o náhodný uhol 90 – 180 stupňov a so zdvihnutým perom ju posuňte dopredu o nejaký násobok 25 krokov.)

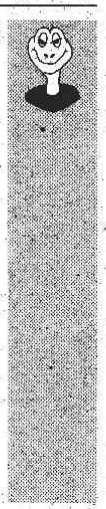
Vráťme sa k cvičeniu 1 v kapitole *3 Definujeme vlastné príkazy.* Našou úlohou bolo nakresliť pravouhlý trojuholník. Dĺžku jeho prepony sme vypočítali z Pytagorovej vety pomocou druhej odmocniny. Použili sme pritom operáciu **sqrt** a preponu sme nakreslili príkazom **dopredu sqrt 50*50 + 50*50**. Druhú mocninu sme však museli vypočítať „ručne", pretože Logo nepozná operáciu, ktorá by ju vypočítala za nás. Zadefinujme teraz operáciu, ktorá vypočíta druhú mocninu čísla:



```
viem umocni :cislo
  vysledok :cislo * :cislo
koniec
```

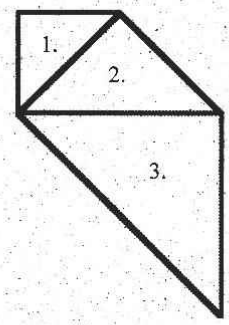
Operáciu v Logu definujeme podobne ako príkaz. Musíme však určiť, čo bude jej výslednou hodnotou. Na to používame príkaz **vysledok** (krátko **vy**).

Príklad 2. Nakreslime takúto skupinu rovnoramenných pravouhlých trojuholníkov.



Riešenie:

Pozorne si prezrime obrázok. Pri jeho kreslení budeme musieť viackrát počítať dĺžku prepony trojuholníka. Zadefinujme najskôr operáciu, ktorá ju vypočíta.




```
viem prepona :dlzka.odvesny
  vysledok sqrt 2 * umocni :dlzka.odvesny
koniec
```

Túto operáciu teraz výhodne použijeme pri kreslení trojuholníka:

```
viem pravouhly.trojuholnik :odvesna
  do :odvesna vp 90 do :odvesna
  vp 135 do prepona :odvesna vp 135
koniec
```

Prvý trojuholník nakreslíme príkazom **pravouhly.trojuholnik 50**. Odvesna druhého trojuholníka je rovnako dlhá ako prepona prvého. Po otočení korytnačky **vp 45** ho nakreslíme príkazom **pravouhly.trojuholnik prepona 50**. Znovu korytnačku otočíme **vp 45** a tretí trojuholník nakreslíme príkazom **pravouhly.trojuholnik prepona prepona 50**.

V matematických operáciách často vystupujú konštanty ako Ludolfovo číslo π alebo Eulerovo číslo e. Niekedy je výhodné zadefinovať si vlastné operácie, ktorých hodnotami sú práve tieto konštanty. **viem pi**

```
    vysledok 3.14
koniec
```

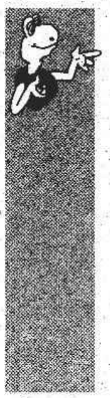
Odteraz môžeme operáciu **pi** používať vo výrazoch, napríklad **zobraz pi/4**.

Cvičenia

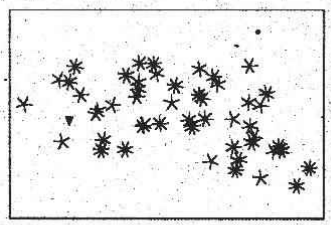
5. Kružnicu sme nakreslili ako 24-uholník, ktorého dĺžku strany určil vstup **dlzka**. Zadefinujte operáciu **polomer :dlzka**, ktorá vypočíta polomer tejto kružnice. Zadefinujte tiež operáciu **strana :r**, ktorá vypočíta „dĺžku strany kružnice" s polomerom *r*.

6. Zadefinujte operácie, ktoré budú premieňať veľkosť uhla v stupňovej miere do oblúkovej a naopak.

6 Jednoduchá chvostová rekurzia



Naprogramujme jednoduchý šetrič obrazovky. Použijeme už zadefinovaný príkaz **hviezda**, ktorý nakreslí hviezdu s daným počtom ramien danej dĺžky.

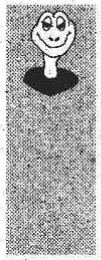


```
viem setric.obrazovky
  hviezda 5 + nahodne 5
  vp 90 + nahodne 90
  ph do 10 + 25 * nahodne 5 pd
  setric.obrazovky
koniec
```

Korytnačka nakreslí hviezdu, posunie sa a zavolá ten istý príkaz **setric.obrazovky**, čiže v novej pozícii nakreslí druhú hviezdu, presunie sa a opäť zavolá **setric.obrazovky**... . Tento proces sa opakuje donekonečna.

Procedúra **setric.obrazovky** volá vo svojom tele samu seba. Takéto volanie nazývame *rekurzívne*. Ak je volanie posledným príkazom v procedúre, hovoríme o *chvostovej rekurzii*.

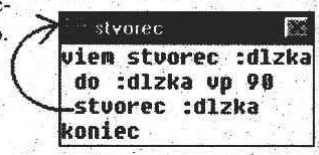
Nekonečnú rekurziu (rovnako ako vykonávanie ktoréhokoľvek iného príkazu) zastavíme stlačením klávesu **F12**, *alebo kliknutím na tlačidlo s nápisom* **STOP** *v hornom páse tlačidiel.*



Príklad 1. Napíšme príkaz, ktorým bude korytnačka donekonečna kresliť štvorec.

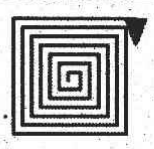
Riešenie:

Najjednoduchšie by bolo dopísať na koniec nami už definovaného príkazu **stvorec** opätovné volanie príkazu **stvorec**. V tomto prípade však stačí, aby korytnačka donekonečna opakovala iba dva príkazy: **dopredu** a **vpravo**.




```
viem stvorec :dlzka
  do :dlzka vp 90
  stvorec :dlzka
koniec
```

Skúsme riešenie predchádzajúceho príkladu trochu modifikovať:



```
viem iny.stvorec :dlzka
  do :dlzka vp 90
  iny.stvorec :dlzka + 3
koniec
```

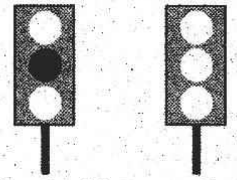
V tomto rekurzívnom volaní nezachovávame pôvodnú dĺžku strany, ale zväčšujeme ju vždy o 3 kroky. Napríklad pri volaní **iny.stvorec 5** vstup **dlzka** nadobúda postupne hodnoty 5, 8, 13, ... a korytnačka nakreslí špirálu.

Pri experimentovaní s nekonečnou rekurziou sa môže obrázok s pribúdajúcim počtom čiar veľmi rýchlo zneprehľadniť. Vykonávanie príkazov spomalíme, ak medzi ne vsunieme príkaz **cakaj**. *Príkaz očakáva na vstupe číslo, vyjadrujúce čas v milisekundách, na ktorý sa vykonávanie zastaví. Napríklad volanie* **cakaj 1000** *pozastaví výpočet na 1 sekundu.*

Cvičenia

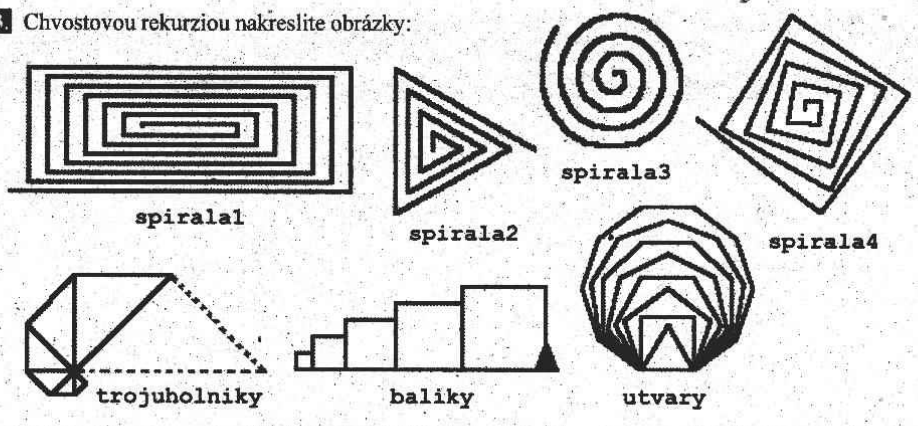
1. Zadefinujte príkazy, ktorými bude korytnačka chodiť donekonečna po obvode trojuholníka, obdĺžnika, kružnice.

2. Naprogramujte blikajúci semafor (stredné svetlo mení farbu striedavo zo žltej na bielu). Pokúste sa navrhnúť riešenie, v ktorom použijete iba jeden príkaz **zmen.farbu.vyplne**.



zmen.fv 14 **zmen.fv 15**

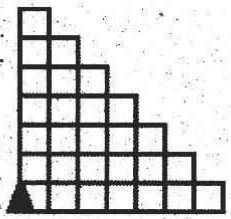
3. Chvostovou rekurziou nakreslite obrázky:



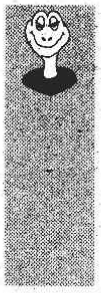
Príklad 2. Chvostovou rekurziou nakreslime schody.

Riešenie:

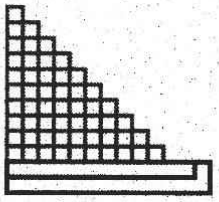
Obrázok schodov sa skladá z obdĺžnikov postupne sa meniacich rozmerov. Ako prvý nakreslíme obdĺžnik s najväčšou výškou. V každom nasledujúcom zmenšíme výšku o 15 bodov a šírku naopak o 15 bodov zväčšíme.



```
viem schody :vyska :sirka
  opakuj 2 [do :vyska vp 90 do :sirka vp 90]
  cakaj 500
  schody :vyska - 15 :sirka + 15
koniec
```



Keďže pri vykonávaní príkazu **schody** hodnota vstupu **vyska** neustále klesá, časom nadobudne záporné hodnoty a korytnačka začne kresliť obdĺžniky na opačnú polrovinu ako doteraz. My však chceme, aby sa kreslenie ukončilo, akonáhle bude hodnota vstupu **vyska** 0 alebo záporné číslo. Doteraz sme to vedeli iba pomocou klávesu **F12** alebo tlačidlom **Stop**. Teraz si ukážeme, ako ukončiť kreslenie programovo. Použijeme na to príkaz **ak**:



```
viem schody :vyska :sirka
  ak :vyska <= 0 [ukonci]
  opakuj 2 [do :vyska vp 90 do :sirka vp 90]
  cakaj 500
  schody :vyska - 15 :sirka + 15
koniec
```

ak *podmienka* [*postupnosť príkazov*]

Podmienka je logický výraz, ktorý je buď pravdivý, alebo nepravdivý. Ak je výraz pravdivý, vykoná sa ***postupnosť príkazov*** uvedená v zátvorkách.

V našom prípade, ak **vyska** nadobudne nulovú alebo zápornú hodnotu, chceme kreslenie ukončiť. Zavoláme preto príkaz **ukonci**. Tento príkaz preruší práve vykonávanú procedúru a vráti výpočet procedúre, ktorá ju volala. Keďže v našom príklade je rekurzívne volanie príkazu **schody** posledným príkazom v procedúre, postupne sa ukončí vykonávanie všetkých príkazov **schody**.

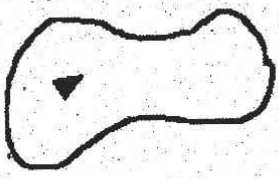
Cvičenie

4. Preprogramujte príkazy z predchádzajúcich cvičení tak, aby sa vykonávanie nekonečnej rekurzie vo vhodnom okamihu zastavilo.

7 Vetvenie výpočtu

Naprogramujme si takýto jednoduchý program: korytnačka sa pohybuje po ploche, ktorá je ohraničená modrou čiarou. Vždy, keď narazí na modrú farbu, náhodne sa otočí a pokračuje v pohybe.

Najskôr nakreslíme obrys plochy. Nakreslíme ho v nejakom grafickom editore, v ktorom ho uložíme na disk ako bitovú mapu (obrázok vo formáte *bmp*). Tento obrázok vložíme do Loga funkciou *Prečítaj plochu* z menu *Súbory*.

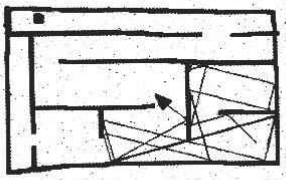


```
viem chod
  do 2
  ak farba.bodu = 9 [vz 2 vp 90 - 180 * nahodne 2]
  chod
koniec
```

Operácia **farba.bodu** je ďalšia zo základných operácií Loga. Jej výslednou hodnotou je farba bodu, na ktorom korytnačka práve stojí.

Príklad 1. Naprogramujme korytnačku ako pávúka, ktorý sa pohybuje v bludisku a „tká" pritom sieť.

Riešenie:

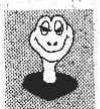


Aj keď sa na prvý pohľad zdá, že riešenie tohto príkladu sa líši od predchádzajúcej úlohy iba tým, že korytnačka má pero dolu, nie je to pravda. Korytnačka totiž pri kresení čiary prekresľuje podklad, a preto bod, na ktorom stojí, je vždy rovnakej farby ako farba jej pera. Tento problém vyriešime tak, že korytnačka najskôr zistí farbu bodu pred sebou (prejde dopredu so zdvihnutým perom). Ak je farba bodu modrá, vráti sa a otočí, inak sa vráti a nakreslí čiaru.

```
viem pavuk
  do 2
  ak farba.bodu = 9 [vz 2 vp 30 - 60 * nahodne 2][vz 2 pd do 2 ph]
  pavuk
koniec
```

V príkaze **pavuk** sme použili úplný príkaz vetvenia:

```
ak podmienka ~
   [čo robiť, ak podmienka platí]~
   [čo robiť, ak podmienka neplatí]
```



Príklad 2. Napíšme príkaz, ktorým korytnačka nakreslí osem guľôčok v rade vedľa seba, náhodne červenou alebo sivou farbou.

Riešenie:

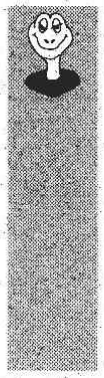
```
viem vygeneruj
  zmaz vp 90 zmen.hp 20
  opakuj 8 [ak nahodne 2 = 1 [zmen.fp 12][zmen.fp 7]~
           pd bodka ph do 25]
koniec
```

Cvičenia

1. Dokreslite do bludiska z príkladu 1 muchu (červenú bodku). Ak pavúk muchu nájde, zje ju (červenú oblasť vyplní farbou pozadia) a program skončí.

2. Naprogramujte hru **Mince**: Korytnačka najskôr nakreslí mince príkazom **vygeneruj**. Potom korytnačkou pohybujeme po minciach a otáčame ich (prefarbujeme guľôčky opačnou farbou, než akou sú nakreslené). Otáčame vždy mincu, na ktorej korytnačka stojí i obe susedné mince.

Príklad 3. Zadefinujme operáciu **pocet.cervenych**, ktorá zistí počet červených guľôčok vygenerovaných príkazom **vygeneruj** z príkladu 2.



Riešenie:

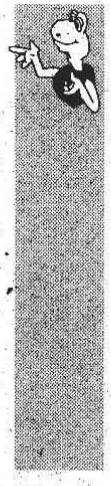
Na postupné sčítanie guľôčok si musíme pamätať ich počet. Potrebujeme na to premennú.

S premennými sme už pracovali. Vstupy do príkazov sú vlastne *lokálne premenné*, ktoré existujú počas výpočtu v tomto príkaze. Hodnotu takejto premennej sme určili na vstupe, a potom sme ju menili pri rekurzívnom volaní:

```
viem spirala :dlzka
  do :dlzka vp 90
  spirala :dlzka + 10
koniec
```

Volaním **spirala 20** nadobudne premenná **dlzka** hodnotu 20. Rekurzívnym volaním **spirala :dlzka + 10** sa hodnota **:dlzka + 10** stane novou hodnotou premennej **dlzka**.

V Logu si však môžeme zadefinovať i *globálne premenné* príkazom **urob "*meno hodnota***. V tomto prípade musíme hneď určiť aj počiatočnú hodnotu premennej. Túto hodnotu môžeme samozrejme kedykoľvek zmeniť:



```
? urob "a 10
? urob "b 3 * :a
? urob "a 2 * :a
? zvys "a
```

Príkaz **zvys "*premenná*** je základný príkaz Loga, ktorý zvýši hodnotu danej ***premennej*** o 1. Tento príkaz má ten istý efekt ako **urob "*premenná* :*premenná* + 1**. Analogicky príkaz **zniz "*premenná*** hodnotu premennej zníži o 1.

Hodnoty premenných **a** a **b** si môžeme pozrieť v okne *Pamäť* alebo ich vypísať do riadkov. Treba si však dávať pozor na to, kedy písať úvodzovky a kedy dvojbodku. Tvar **:a** znamená, že pracujeme s hodnotou premennej **a**, tvar **"a** znamená, že pracujeme s menom premennej **a**. Všimnite si rozdiel:

```
? zobraz :a
21
? zobraz "a
a
```

Teraz už vieme napísať riešenie príkladu:

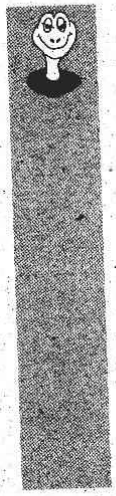
```
viem pocet.cervenych
  ph domov vp 90 urob "cervenych 0
  opakuj 8 [ak farba.bodu = 12 [zvys "cervenych]] do 25]
  vy :cervenych
koniec
```

Všimnite si, že sme v riešení použili úplný príkaz vetvetnia, hoci v prípade, že **farba.bodu <> 12**, *nič nevykonávame. Ak by sme totiž použili neúplný príkaz vetvenia (nenapísali by sme []), Logo by vyhlásilo chybu.*

Predchádzajúci príkaz sčíta červené guľôčky medzi 8 guľôčkami na ploche. Zmeňme tento príkaz tak, aby zistil počet červených guľôčok v rade s ľubovoľným počtom farebných guľôčok.

```
viem pocet.cervenych.inak
  ph domov vp 90 urob "cervenych 0
  pocet.cervenych.inak1
  vy :cervenych
koniec
```

```
viem pocet.cervenych.inak1
  ak farba.bodu = 15 [ukonci]
  ak.farba.bodu = 12 [zvys "cervenych]
  do 25
  pocet.cervenych.inak1
koniec
```



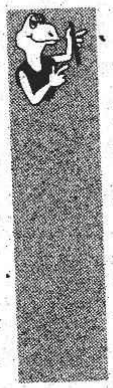
Príklad 4. Napíšme program, v ktorom si počítač náhodne vyberie číslo z intervalu ⟨1, 10⟩ a nechá nás hádať, aké je to číslo.

Riešenie:

```
viem hadaj.cislo
  zo [ Myslim si cislo, uhadni ho! ]
  hadaj 1 + nahodne 10
koniec

viem hadaj :cislo
  urob "tip citaj.slovo
  ak :tip = :cislo ~
     [ zo [ Uhadol si! ] ukonci] ~
     [ zo [ Skus este raz... ] ]
  hadaj :cislo
koniec
```

V riešení príkladu sme použili novú operáciu **citaj.slovo**. Táto operácia najskôr čaká, kým napíšeme do riadkov slovo a stlačíme **Enter**. Výslednou hodnotou je potom toto slovo.



Cvičenia

3. Predefinujte niektoré príkazy z hry **Mince** tak, aby sa v riadkoch vypísal nápis „Červený vyhral“ ak sú všetky mince na ploche červené a nápis „Sivý vyhral“, ak sú všetky mince sivé. Môžete použiť príkazy z predchádzajúcich cvičení.

4. Prerobte príkaz **hadaj** z príkladu 4 tak, aby oznamoval, či je naše číslo väčšie alebo menšie ako to, ktoré máme uhádnuť.

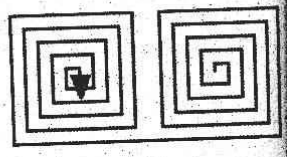
5. Sčítajte počet pokusov potrebných na uhádnutie čísla a podľa tohto počtu rozlíšte, aký text počítač vypíše, ak uhádneme číslo.

6. Preprogramujte príkaz **hadaj** tak, aby dovolil tipovať najviac osemkrát.

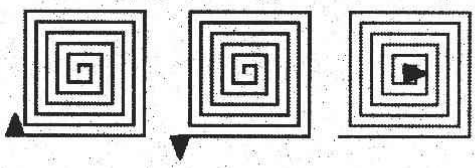
Náročnejšie cvičenia

Príklady, ktoré uvádzame ďalej, presahujú rámec tejto učebnice. Bolo by však škoda neoboznámiť sa s nimi. Uvádzame tu preto aspoň ich riešenia ako námety na ďalšiu prácu.

```
viem spirala1 :dlzka
  ak :dlzka > 100 [vp 180 ukonci]
  do :dlzka vp 90
  spirala1 :dlzka + 5
  vp 90 do :dlzka
koniec
```

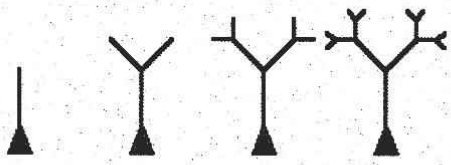


```
viem spirala2 :dlzka
  ak :dlzka > 300 ~
     [vp 90 zmen.fp 14 ukonci]
  do :dlzka vp 90
  spirala2 :dlzka + 3
  do :dlzka vl 90
koniec
```



Binárny strom

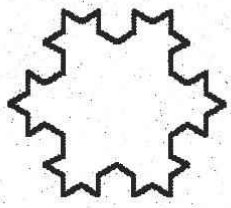
```
viem strom :stupen :dlzka
  ak :stupen = 0 [ukonci]
  dopredu :dlzka  vlavo 45
  strom :stupen - 1 :dlzka / 2
  vpravo 90
  strom :stupen - 1 :dlzka / 2
  vlavo 45  vz :dlzka
koniec
```



stupeň 1 stupeň 2 stupeň 3 stupeň 4

Vločka

```
viem strana :stupen :dlzka
  ak :stupen = 0 [dopredu :dlzka ukonci]
  strana :stupen - 1 :dlzka / 3
  vlavo 60
  strana :stupen - 1 :dlzka / 3
  vpravo 120
  strana :stupen - 1 :dlzka / 3
  vlavo 60
  strana :stupen - 1 :dlzka / 3
koniec
```

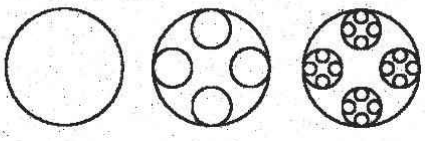


```
viem vlocka :stupen :dlzka
  opakuj 3 [strana :stupen :dlzka ~
            vpravo 120]
koniec
```

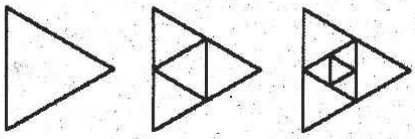
strana stupňa 0 strana stupňa 1 strana stupňa 2

Cvičenie

7. Nakreslite takéto rekurzívne krivky:

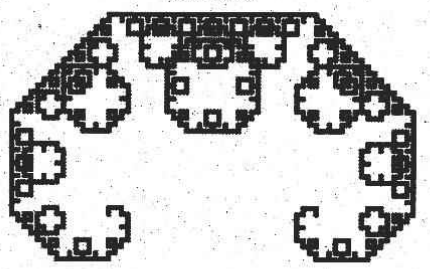


stupeň 0 stupeň 1 stupeň 2



stupeň 1 stupeň 2 stupeň 3

C-krivka



stupeň 0 stupeň 1 stupeň 2 stupeň 3

8 Spracovanie vstupov z klávesnice

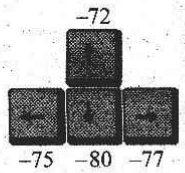
V úvode kapitoly *6 Jednoduchá chvostová rekurzia* sme si naprogramovali šetrič obrazovky. Korytnačka donekonečna kreslila na náhodné pozície na ploche rôzne veľké hviezdy. Doteraz sme ho vedeli prerušiť iba stlačením špeciálneho klávesu na ukončenie **F12**. Doprogramujme teraz šetrič obrazovky tak, aby sa ukončil na stlačenie ľubovoľného klávesu.

```
viem setric.obrazovky
  ak klaves? [ukonci]
  hviezda 5 + nahodne 5
  vp 90 + nahodne 90
  ph do 10 + 25 * nahodne 5 pd
  setric.obrazovky
koniec
```

Prvý riadok príkazu **ak klaves? [ukonci]** presne vystihuje náš zámer. Ak bol stlačený kláves, ukončí sa vykonávanie príkazu. Procedúra **klaves?** je základná logovská operácia, ktorej výsledkom je **ano**, ak bol stlačený nejaký kláves na klávesnici.

V predchádzajúcom príkaze stačilo zistiť, či bol alebo nebol stlačený nejaký kláves. Niekedy však potrebujeme zistiť aj to, ktorý kláves bol stlačený. Používame na to operáciu **klaves**. Táto operácia čaká na stlačenie klávesu a jej výsledkom je číselný kód stlačeného klávesu.

Príklad 1. Napíšme príkaz, ktorý bude vypisovať kódy stlačených klávesov.



Riešenie:

```
viem kody.klavesov
  zo klaves
  kody.klavesov
koniec
```

Po spustení príkazu **kody.klavesov** sa začnú do textovej plochy vypisovať kódy postupne stláčaných klávesov. Napríklad kláves **ESC** má kód 27.

Príklad 2. Preprogramujme príkaz **kody.klavesov** z predchádzajúceho príkladu tak, aby sa jeho vykonávanie po stlačení klávesu **ESC** ukončilo.

Riešenie nesprávne:

```
viem kody.klavesov
  ak klaves = 27 [ukonci]
  zo klaves
  kody.klavesov
koniec
```

Spusťme takto zadefinovaný príkaz a stlačme napríklad medzerovník. Príkaz **klaves** v prvom riadku procedúry čaká na stlačenie klávesu a získaný kód (v tomto prípade 32) porovnáva s číslom 27. Keďže čísla nie sú rovnaké, príkaz **ukonci** sa nevykoná. V druhom riadku príkaz **klaves** znovu čaká na stlačenie klávesu a až kód nového stlačenia je vstupom do príkazu **zobraz**. Kód medzerovníka sme teda stratili. Správne riešenie najskôr raz načíta vstup z klávesnice do premennej, a potom pracuje s hodnotou tejto premennej.

Riešenie správne:

```
viem kody.klavesov
  urob "kl klaves
  ak :kl = 27 [ukonci]
  zo :kl
  kody.klavesov
koniec
```

Príklad 3. Naprogramujme jednoduchý kresliaci program. Po stlačení šípky hore nakreslí korytnačka čiaru dlhú 10 krokov, po stlačení šípky vpravo sa otočí vpravo o 30 stupňov a po stlačení šípky vľavo sa otočí vľavo o 45 stupňov. Program ukončíme stlačením klávesu **ESC**.

Riešenie:

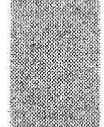
```
viem kreslenie
  urob "kl klaves
  ak :kl = 27 [ukonci]
  ak :kl = -72 [do 10] ; šípka hore
  ak :kl = -75 [vl 45] ; šípka vľavo
  ak :kl = -77 [vp 30] ; šípka vpravo
  kreslenie
koniec
```

Alternatívne riešenie uvedené na obrázku vpravo používa na vyhodnotenie vstupov z klávesnice príkaz **ak.je**. *Pri práci s príkazom* **ak.je** *treba venovať pozornosť správnemu uzátvorkovaniu príkazov.*

Zábavnejšia forma kreslenia je taká, keď korytnačka neustále chodí a my ju iba otáčame:

```
viem aktivne.kreslenie
  ak klaves? [urob "kl klaves][urob "kl 555]
  ak :kl = 27 [ukonci]
  ak :kl = -75 [vl 10] ; šípka vľavo
  ak :kl = -77 [vp 10] ; šípka vpravo
  do 10 cakaj 10
  aktivne.kreslenie
koniec
```

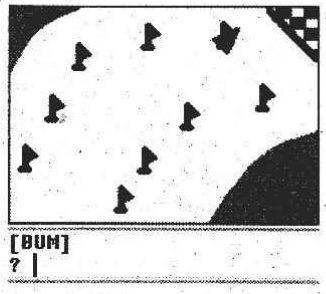
Porovnajme príkazy **kreslenie** a **aktivne.kreslenie**. Pri aktívnom kreslení sa má korytnačka pohybovať stále, nie iba po stlačení šípky. Príkaz **dopredu 10** je preto použitý ako samostatný príkaz (vykonáva sa vždy, a nie iba ako odozva na stlačenie šípky hore). Tiež chceme, aby sa korytnačka hýbala aj v prípade, keď nie je stlačený kláves. Operácia **klaves** však celý výpočet zastaví a čaká na stlačenie klávesu. Kód klávesu preto čítame, iba ak bol stlačený nejaký kláves. Viete prečo sme pri čítaní klávesu použili úplný príkaz **ak**?



Príklad 4. Naprogramujme ovládanie autíčka, ktoré sa pohybuje po dráhe a vyhýba sa čiernym prekážkam. Úlohou hráča je dopraviť autíčko do cieľa (červená linka pred šachovnicou).

Riešenie:

```
viem chod
  ak klaves? ~
     [urob "kl klaves][urob "kl 555]
  ak :kl = -75 [vl 45]
  ak :kl = -77 [vp 45]
  do 1
  ak farba.bodu = 0 [zo [BUM] ukonci]
  ak farba.bodu = 12 [zo [HURA] ukonci]
  chod
koniec
```

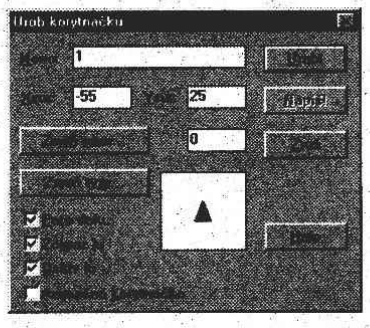


Cvičenia

1. Doprogramujte do príkazu **chod** zrýchľovanie a spomaľovanie pohybu autíčka.
2. Dodefinujte do príkazu **kreslenie** možnosť zmeniť farbu a hrúbku pera.
3. Preprogramujte hru **Mince** z kapitoly *7 Vetvenie výpočtu* tak, aby sme korytnačku ovládali stláčaním klávesov.

9 Práca s viacerými korytnačkami

Naše prvé kroky v Logu začali oboznámením sa s korytnačkou **0** a až dosiaľ sme pracovali iba s ňou. Logo však umožňuje vytvoriť mnoho ďalších korytnačiek, čo pri riešení úloh prináša nové programátorské prístupy.

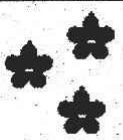


Novú korytnačku vytvoríme kliknutím pravým tlačidlom myši na plochu. V kontextovom menu vyberieme položku *Tu urob korytnačku*. Otvorí sa dialógové okno, v ktorom napíšeme meno korytnačky a potvrdíme tlačidlom *Urob!* Menom môže byť ľubovoľné slovo alebo číslo (napríklad **1**, **Zofka**).

Ostatné údaje o vytváranej korytnačke, ktoré môžeme v tomto dialógu nastaviť, si popíšeme neskôr.

Klikaním na plochu vytvorme dve nové korytnačky. Pomenujme ich **1** a **Zofka**. Napíšme teraz do príkazového riadka príkazy:

```
zmen.hp 10 zmen.fp 2
kvet
```



Všetky tri korytnačky naraz nakreslili zelený kvet. Všetky sú totiž *aktívne*. Korytnačka **0** je aktívna hneď po spustení Loga a zvyšným dvom sme pri ich vytváraní nechali označenú položku *Oslov ju*. Ak chceme nakresliť tri rôznofarebné kvety, musíme každej korytnačke nastaviť inú farbu pera:

```
pre [1][zmen.fp 12]
pre [Zofka][zmen.fp 1]
kvet
```

Každá korytnačka v Logu je *aktívna* alebo *neaktívna*. Zadané príkazy vykonávajú iba aktívne korytnačky. Na *dočasnú* zmenu množiny aktívnych korytnačiek používame príkaz **pre**:

pre [*zoznam korytnačiek*] [*postupnosť príkazov*]

Príkaz **pre** spôsobí, že ***postupnosť príkazov*** uzavretú v hranatých zátvorkách vykonajú iba korytnačky uvedené v ***zozname korytnačiek***.

Príkaz **pre [1][zmen.fp 12]** zmení korytnačke **1** farbu pera na červenú, podobne príkaz **pre [Zofka][zmen.fp 1]** zmení korytnačke **Zofka** farbu pera na modrú. Keďže po vykonaní príkazu **pre** sú aktívne tie korytnačky, ktoré boli aktívne pred jeho volaním, príkaz **kvet** vykonajú všetky tri korytnačky (**0**, **1** i **Zofka**).

Ak chceme osloviť iba jednu korytnačku, zapisujeme to častejšie takto: **pre "*meno* [*zoznam príkazov*]** *(pozor na úvodzovky pred menom korytnačky). V prípade, že menom korytnačky je číslo, úvodzovky písať nemusíme.*
V našom príklade teda **pre 1 [zmen.fp 12]** *a* **pre "Zofka [zmen.fp 1]**.

Okrem klikania na plochu vieme korytnačku vytvoriť i príkazom **urob.korytnacku** (**urob.kor**):

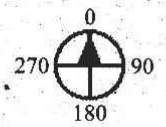
urob.korytnacku "*meno.korytnačky* [*x y smer*]

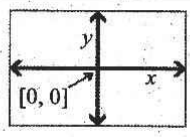
Čísla **x** a **y** sú súradnice bodu na obrazovke, ktorý bude *domovskou pozíciou* korytnačky. ***Smer*** vyjadruje *základný uhol* korytnačky.

Vytvorme ďalšie dve korytnačky:

```
urob.kor 2 [146 -23 0]
urob.kor "Janko [-35 5]
```

Grafickú plochu popisujeme pravouhlou súradnicovou sústavou, známou z matematiky. Základný uhol zodpovedá počiatočnému natočeniu korytnačky. Ak pri vytváraní korytnačky nezadáme základný uhol natočenia, Logo ho nastaví na 0.



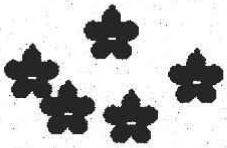


Zrejme ste si všimli, že korytnačky **2** a **Janko** sa na ploche neobjavili. Ak totiž vytvoríme korytnačku príkazom **urob.kor**, automaticky je neaktívna a skrytá. Na aktívnu ju zmeníme príkazom **odteraz**:

odteraz [*zoznam korytnačiek*]

Príkaz urobí korytnačky dané v ***zozname korytnačiek*** aktívnymi. Tieto budú aktívne až dovtedy, kým príkazom **odteraz** neoslovíme iné korytnačky. Zoznam všetkých existujúcich korytnačiek (aktívnych i neaktívnych) vráti operácia **vsetky**. Zoznam aktívnych korytnačiek vráti operácia **kto**.

```
? zobraz vsetky
[0 1 Zofka 2 Janko]
? zobraz kto
[0 1 Zofka]
? odteraz vsetky
? zobraz kto
[0 1 Zofka 2 Janko]
? ukaz zmen.hp 10 kvet
```



Pri práci s viacerými korytnačkami je užitočné vedieť nepotrebné korytnačky zrušiť. Slúži na to príkaz **zrus.korytnacku [*zoznam korytnačiek*]** (skrátene **zrus.kor**).

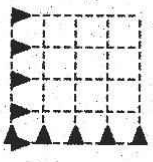
Príklad 1. Nakreslime rad 6 kvetov rôznych farieb.

Riešenie:

```
viem rad.kvetov
  zrus.kor vsetky
  opakuj 6 [urob.kor poc [0 0 90]~
            pre poc [ph do 50*poc vl 90 pd zmen.fp nahodne 15]]
  odteraz vsetky kvet
  opakuj 6 [(urob.kor poc)]
koniec
```

V riešení príkladu sa objavuje nová operácia **poc**. Ide o počítadlo v cykle. Výsledkom operácie **poc** je číslo, ktoré určuje koľkýkrát sa vykonáva príkaz **opakuj**. Pri prvom vykonaní príkazu **opakuj** je hodnota počítadla 1, vytvorí sa preto korytnačka **1**, ktorá prejde 50 krokov. Pri druhom prechode cyklu je už hodnota počítadla 2; vznikne korytnačka **2**, prejde 100 krokov, ... V prípade, že korytnačka "***meno*** už existuje, príkaz **(urob.kor "*meno*)** ju iba predefinuje, čím sa zmení jej aktuálna pozícia na domovskú.

Cvičenia



1. Pomocou viacerých korytnačiek nakreslite štvorcovú sieť rozmerov $n \times n$.

2. Naprogramujte blikajúce slnko. Efekt blikania docielite tak, že budete viacerými korytnačkami striedavo kresliť a mazať (prekreslovať bielou farbou) lúče.

Príklad 2. V príklade 1 sme vytvorili 6 korytnačiek s menami **1**, **2**, ... , **6**. Napíšme príkaz **pohni.sa**, v ktorom budeme tieto korytnačky hýbať klávesmi. Po stlačení klávesu **1** (kód 49) sa korytnačka **1** pohne dopredu o 10 krokov, po stlačení klávesu **2** (kód 50) sa pohne dopredu o 10 krokov korytnačka **2**, ...

Riešenie:

```
viem pohni.sa
  urob "kl klaves
  ak :kl = 27 [ukonci]
  pre :kl - 48 [dopredu 10]
  pohni.sa
koniec
```

V riešení príkladu využívame fakt, že kódy klávesov **1**, **2**, **3**, ... sú za sebou idúce čísla. Ak stlačíme iný kláves ako **1**, **2**, ... , **6**, príkaz hlási chybu. Porozmýšľajte, ako to ošetriť.

Príklad 3. Napíšme príkaz **hyb.sa**, v ktorom budeme pohyb korytnačiek ovládať takto: po stlačení klávesu **4** sa začne korytnačka **4** pohybovať s krokom 1. Po stlačení klávesu **3** korytnačka **4** zastane a pohybovať sa začne korytnačka **3**, ...

Riešenie:

```
viem hyb.sa
  ak klaves? [urob "kl klaves ~
                ak :kl = 27 [ukonci][odteraz :kl - 48]]
  dopredu 1
  hyb.sa
koniec
```

Na začiatku sú všetky korytnačky neaktívne. Kým nestlačíme niektorý z klávesov **1**, **2**, ... , **6**, príkaz **dopredu 1** nebude vykonávať žiadna korytnačka.

Príklad 4. Upravme príklad 3 tak, že sa budú naraz hýbať dopredu s krokom 1 všetky korytnačky. Stlačenie niektorého z klávesov **1**, **2**, ... , **6** spôsobí, že príslušná korytnačka nakreslí na svojej pozícii kvet.

Riešenie:

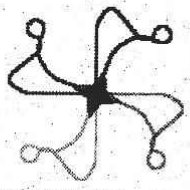
```
viem hybte.sa
  ak klaves? [urob "kl klaves ~
                ak :kl = 27 [ukonci][pre :kl - 48 [kvet]]]
  dopredu 1
  hybte.sa
koniec
```

Pred spustením príkazu **hybte.sa** musia byť všetky korytnačky aktívne.

Cvičenia

1. Preprogramujte príkaz **hybte.sa** z príkladu 4 tak, aby sa po stlačení iného klávesu než **1**, **2**, ... , **6** prestali hýbať všetky korytnačky. Vykonávanie príkazu sa bude dať ukončiť stlačením klávesu **ESC**.

2. Príkazom **aktivne.kreslenie** z kapitoly *8 Spracovanie vstupov z klávesnice* môžeme ovládať všetky aktívne korytnačky. Vytvorte vhodne niekoľko korytnačiek s rôzne nastavenými farbami pera a pomocou príkazu **aktivne.kreslenie** nakreslite takéto obrázky.



3. Naprogramujte hru **motorky**, v ktorej budú dvaja hráči klávesmi ovládať dve korytnačky – motorky. Každá motorka kreslí na ploche čiaru inej farby. Hru prehráva hráč, ktorého motorka narazí na čiaru (súperovu či vlastnú).

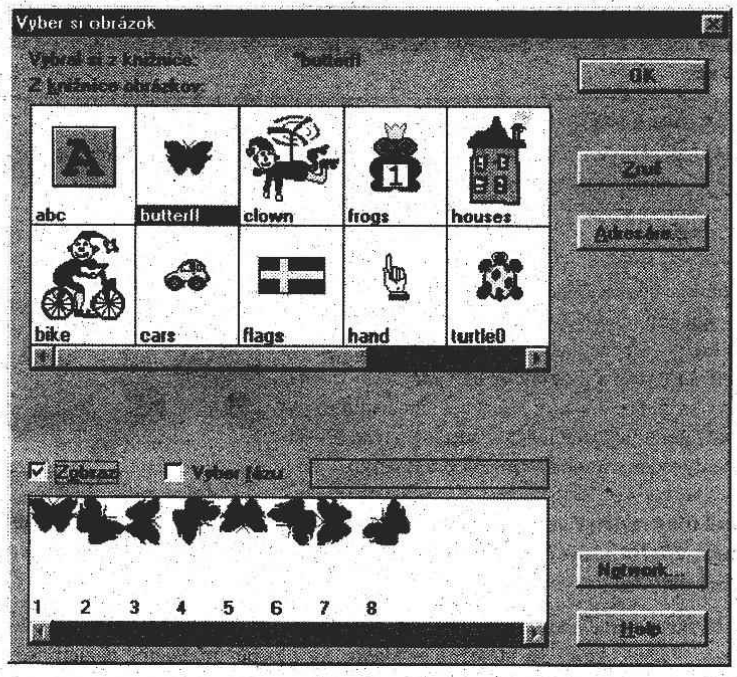
Tvar korytnačky

10

V úlohách, ktoré sme doteraz riešili, nás väčšinou zaujímali pohyby korytnačky a čiary, ktoré kreslila. Korytnačka pre nás predstavovala grafické pero a tvar rovnoramenného trojuholníka vyjadroval jej aktuálne natočenie. Tvar korytnačky je však len jedna z jej vlastností (ako napríklad farba a hrúbka pera) a dá sa meniť.

Tvarom korytnačky môže byť ľubovoľný logovský obrázok. Pod pojmom logovský obrázok rozumieme postupnosť fáz, pričom každá fáza je samostatná bitová mapa, ktorá nie je väčšia ako 255 × 255 grafických bodov. V danom okamihu je korytnačka na ploche zobrazená práve jednou z fáz svojho tvaru, a to v závislosti od jej aktuálneho natočenia.

Najjednoduchší spôsob, ako zmeniť korytnačke tvar, je kliknúť na ňu pravým tlačidlom myši a v kontextovom menu vybrať položku *Korytnačka: zmeň ma...* . Otvorí sa podobný dialóg, aký už poznáme z vytvárania novej korytnačky. Klikneme v ňom na tlačidlo *Zmeň tvar* a v dialógu, ktorý sa následne zobrazí, vyberieme korytnačke nový tvar z knižnice už existujúcich obrázkov. Ak chceme vidieť jednotlivé fázy obrázka, musíme označiť položku *Zobraz*.

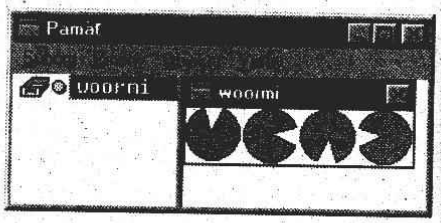


Zmeňme teraz opísaným spôsobom tvar korytnačky na motýľa a sledujme, ktorými fázami bude zobrazený pri rôznych natočeniach. Každej fáze zodpovedá interval natočení, pri ktorých je korytnačka zobrazená práve touto fázou. Od počtu fáz závisí počet a rozsah intervalov, na ktoré sa kruh rozdelí. Na obrázku má korytnačka aktuálny uhol z intervalu (112.5°; 157.5°). Keďže rozsah intervalov je 45 stupňov, príkazom **vp 45** sa korytnačka zobrazí nasledujúcou fázou.

Všetky obrázky, ktoré nám dialóg *Vyber si obrázok* ponúkol, boli vytvorené v grafickom editore Comiedit a uložené ako súbory (obrázky interného formátu *lgw*) na disk. Preto, ak si chceme nakresliť vlastný tvar a ten potom korytnačke priradiť, môžeme ho najskôr editovať v programe Comiedit a uložiť ako samostatný *lgw* súbor. (Ak tento súbor nenahráme do adresára *IMAGE* k ostatným už existujúcim obrázkom, v dialógu *Vyber si obrázok* aktivujeme možnosť prehľadávania disku tlačidlom *Adresáre...*.)

Častejšie však nami vytvorené obrázky nebudeme ukladať na disk ako samostatné súbory, ale budú súčasťou jedného súboru – nášho projektu. Podobne, ako sme si doteraz pamätali v premenných nejaké číselné hodnoty, budeme si aj obrázky pamätať v premenných.

Novú obrázkovú premennú vytvoríme cez menu *Objekty (Pridaj premennú)* v okne *Pamäť*. V dialógu, ktorý sa nám otvorí, zadáme meno premennej a nastavíme, že jej hodnotou má byť obrázok. Po potvrdení tlačidlom **OK** sa spustí Comiedit. Nakreslíme obrázok a grafický editor ukončíme kliknutím na modrú šípku v pravom hornom rohu okna. Tým sa vrátime späť do prostredia Comenius Loga, pričom pamäť bude obsahovať nami práve zadefinovanú obrázkovú premennú. Ak teraz budeme priraďovať korytnačke nový tvar, dialóg *Vyber si obrázok* nám ponúkne okrem obrázkov z knižnice aj všetky obrázkové premenné z pamäte.



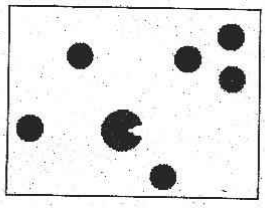
Zatiaľ vieme korytnačke meniť tvar iba v priamom režime. Z programovacích prostriedkov na to slúži príkaz **zmen.tvar**:

zmen.tvar obrázok

Ak namiesto logovského obrázku bude vstupom prázdny zoznam **[]**, tvarom korytnačky sa stane základný tvar (rovnoramenný trojuholník).

```
? zmen.tvar :woormi
```

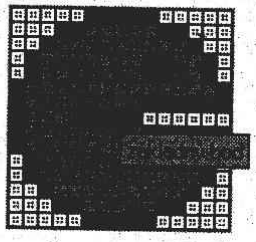
Príklad 1. Naprogramujme takúto jednoduchú hru: Postavička, ktorú ovládame šípkami, chodí štyrmi smermi (vľavo, vpravo, hore a dolu) po ploche, na ktorej sa nachádza potrava (bodky zelenej farby) a jed (bodky červenej farby). Jedením potravy sa sila postavičky zväčšuje, narazením na jed sa sila zmenšuje. Silu postavičky vyjadruje jej tvar. (Postavička má vždy jeden z troch rôzne veľkých tvarov.)



Riešenie:

Najskôr si zadefinujeme tri obrázkové premenné: **maly.tvar**, **stredny.tvar** a **velky.tvar**. Keďže korytnačka bude chodiť vždy iba v štyroch rôznych smeroch, tvary budú štvorfázové obrázky zodpovedajúce smerom hore, vpravo, dolu a vľavo.

Pri kreslení jednotlivých fáz musíme nastaviť vhodný *základný bod*. Je to ten bod fázy (bitovej mapy), ktorý sa stotožní s bodom na ploche určujúcim pozíciu korytnačky. Preddefinovaná hodnota tohto bodu je **[1 1]** (ľavý horný bod bitovej mapy). Ak by sme ju nezmenili, korytnačka by sa pri zisťovaní farby bodu pozerala na bod na ploche ležiaci v ľavom hornom rohu jej tvaru, no prirodzenejšie je odvolávať sa na bod v strede jej tvaru.



Základný bod zmeníme pri editovaní fázy voľbou položky *Voľby (Zákl. bod...)* v hlavnom menu programu Comiedit.

Pozadie zložené z farebných bodiek nakreslíme v grafickom editore a cez *Súbory/Prečítaj plochu* vložíme do Loga, alebo naprogramujeme príkaz, ktorým korytnačka náhodne nakreslí bodky.

Grafickú časť projektu máme za sebou, zostáva nám už len programovať.
Na počítanie sily postavičky použijeme premennú **sila** a vždy, keď sa zmení, nastavíme podľa novej hodnoty tvar korytnačky na jeden z troch obrázkov.

```
viem hra
  ph zmen.fv 15 zmen.tvar :stredny.tvar
  urob "sila 8
  hyb
koniec

viem hyb
  ak klaves? [urob "kl klaves][urob "kl 555]
  ak :kl = 27 [ukonci]
  ak :kl = -72 [zmen.smer 0]
  ak :kl = -80 [zmen.smer 180]
  ak :kl = -75 [zmen.smer 270]
  ak :kl = -77 [zmen.smer 90]
  ak farba.bodu = 12 [zmen.body -1]
  ak farba.bodu = 10 [zmen.body 1]
  do 1
  hyb
koniec

viem zmen.body :zmena
  vypln
  urob "sila :sila + :zmena
  ak :sila < 6 [zmen.tvar :maly.tvar]
  ak :sila > 15 [zmen.tvar :velky.tvar]
  ak zaroven (:sila >= 6) (:sila <= 15) [zmen.tvar :stredny.tvar]
koniec
```

Po stlačení niektorej šípky meníme smer korytnačky príkazom **zmen.smer**:

zmen.smer uhol
Uhol 0 označuje natočenie na sever. Veľkosť uhla stúpa v smere chodu hodinových ručičiek.

V príkaze **zmen.body** sme v poslednom **ak** použili zloženú podmienku. Dve jednoduché podmienky **(:sila >= 6)** a **(:sila <= 15)** sú vstupmi do základnej logovskej operácie **zaroven**, ktorá vráti **ano** iba v prípade, že obe podmienky sú pravdivé. Inak vráti **nie**.
V Logu existuje tiež podobná operácia **alebo**, ktorá rovnako očakáva na vstupe dve podmienky (logické výrazy). Ak je aspoň jedna z nich pravdivá, vráti táto operácia **ano**, inak vráti **nie**.

Cvičenia

1. Doprogramujte predchádzajúci príklad tak, že keď bude mať postavička nulovú silu, alebo vyzbiera všetku potravu, zmení svoj tvar podľa výsledku na veselý alebo smutný a program skončí.

2. Naprogramujte pohyb multifunkčného dopravného prostriedku. Podľa farby podkladu (cesta = 4, obloha = 11, voda = 9), na ktorom sa nachádza, bude meniť svoj tvar na auto, lietadlo alebo loď.

11 Animačné korytnačky

V predchádzajúcej kapitole sme sa naučili meniť korytnačke tvar, v tejto kapitole si ukážeme, ako sa práve vďaka tvaru korytnačky dajú v Logu jednoducho programovať animácie.

Animácia je postupnosť statických obrazov, ktoré premietame nášmu oku v nejakých časových intervaloch. Vieme tiež, že tvarom korytnačky je obrázok zložený z fáz. Ak teda korytnačke povieme, aby postupne menila fázy, ktorými je práve zobrazená na ploche, dostaneme animáciu.

Na to, aby sme vedeli korytnačke priamo určiť fázu, musíme zrušiť závislosť fázy korytnačky od jej aktuálneho natočenia, o ktorej sme si povedali v predchádzajúcej kapitole. Docielime to tým, že ju urobíme *animačnou*.

Každá korytnačka je v Logu *animačná* alebo *neanimačná*. Korytnačku urobíme animačnou príkazom **animacna "ano** a neanimačnou príkazom **animacna "nie**. Neanimačnej korytnačke sa menia fázy automaticky podľa jej aktuálneho uhla, animačnej korytnačke meníme fázy príkazom **zmen.fazu** *číslo*.

Existuje aj iný spôsob, ako určiť, či má byť korytnačka animačná alebo neanimačná. V dialógu Urob korytnačku, resp. v dialógu Zmeň korytnačku túto vlastnosť korytnačke nastavuje položka Animačná.

Urobme teraz korytnačku **0** animačnou a zmeňme jej tvar na lietajúceho klauna. Tento tvar sa skladá z troch fáz. Jednoduchá animácia by teda mohla byť naprogramovaná takto:

```
viem vrt.sa
  opakuj 3 [zmen.fazu poc ~
            cakaj 100]
  vrt.sa
koniec
```

Toto riešenie sa dá vylepšiť použitím operácie **faza**.

faza

Operácia vráti číslo aktuálnej fázy korytnačky.

V našom riešení teda stačí vždy len nastaviť fázu nasledujúcu po aktuálnej fáze. (Fáza nasledujúca po poslednej fáze je prvá.) Takže:

```
viem toc.sa
  zmen.fazu faza + 1
  cakaj 100
  toc.sa
koniec
```

Pridaním príkazu **dopredu** dostaneme animovaný pohyb (nezabudnite pred jeho volaním korytnačku správne natočiť):

```
viem lietaj
  do 5
  zmen.fazu faza + 1
  cakaj 100
  lietaj
koniec
```

Príklad 1. Príklad 1 z predchádzajúcej kapitoly, v ktorom sme šípkami ovládali postavičku, prerobme tak, aby postavička bola pri pohybe animovaná (otvárala a zatvárala ústa).

Riešenie:

Pri pohybe korytnačka cyklicky strieda dve fázy svojho tvaru (otvorené a zatvorené ústa). Tvar budeme korytnačke priraďovať podľa uhla natočenia z obrázkových premenných **hore**, **vpravo**, **dolu** a **vlavo**.

```
viem hra
  ph zmen.smer 0 zmen.tvar :hore
  animacna "ano
  hyb.anim
koniec
viem hyb.anim
  ak klaves? [urob "kl klaves][urob "kl 555]
  ak :kl = 27 [ukonci]
  ak :kl = -72 [zmen.smer 0 zmen.tvar :hore]
  ak :kl = -80 [zmen.smer 180 zmen.tvar :dolu]
  ak :kl = -75 [zmen.smer 270 zmen.tvar :vlavo]
  ak :kl = -77 [zmen.smer 90 zmen.tvar :vpravo]
  do 5
  zmen.fazu faza + 1
  cakaj 100
  hyb.anim
koniec
```

Cvičenie

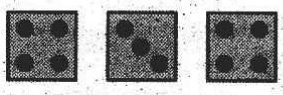
1. Doprogramujte do predchádzajúceho príkladu animovanú konzumáciu jedla a jedu.

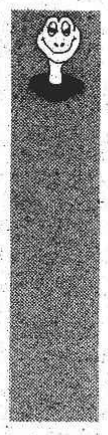
Príklad 2. Naprogramujme hru Videostop.

Riešenie:

Na ploche vytvoríme tri animačné korytnačky, ktorým zmeníme tvar na 6-fázový obrázok z obrázkovej premennej **kocka**. V cykle budeme korytnačkám náhodne meniť fázy. V prípade, že stlačíme iný kláves ako **ESC**, zavoláme príkaz **testuj**, ktorý porovná navzájom fázy korytnačiek, zaktualizuje počet bodov a vypíše body na textovú plochu.

```
viem videostop
  zrus.kor vsetky
  urob.kor 1 [-100 0]
  urob.kor 2 [0 0]
  urob.kor 3 [100 0]
  odteraz vsetky
  zmen.tvar :kocka
  animacna "ano ukaz
  urob "body 0
  videostop1
koniec
viem videostop1
  ak klaves? [ ak klaves = 27 [ukonci][testuj cakaj 1000] ]
  pre 1 [zmen.fazu 1 + nahodne 6]
  pre 2 [zmen.fazu 1 + nahodne 6]
  pre 3 [zmen.fazu 1 + nahodne 6]
  cakaj 500
  videostop1
koniec
```





```
viem testuj
  urob "f1 pre 1 [faza]
  urob "f2 pre 2 [faza]
  urob "f3 pre 3 [faza]
  ak :f1 = :f2 [zvys "body]
  ak :f2 = :f3 [zvys "body]
  ak :f1 = :f3 [zvys "body]
  ak (zaroven (:f1 <> :f2)(:f2 <> :f3)(:f1 <> :f3 )) [zniz "body]
  pis :body
koniec
```

V príkaze **testuj** znižujeme počet bodov, ak sú tvary všetkých troch korytnačiek navzájom rôzne. Tento prípad nastane, iba ak zároveň platia tri podmienky. Keďže operácia **zaroven** očakáva bežne na vstupe dva logické výrazy a my ju voláme až s tromi, musíme ju uzavrieť aj so vstupmi do okrúhlych zátvoriek **()**.

Uveďmeme ešte dva príkazy, ktorými sa nejaký obrázok môže stať súčasťou grafickej plochy:

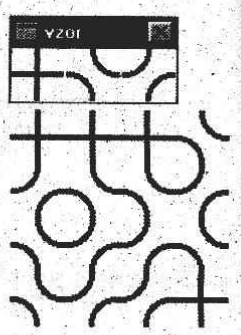
```
otlac.obrazok obrázok
otlac
```

Príkazom **otlac.obrazok** korytnačka položí *do svojej pozície* na plochu prvú fázu vstupného obrázka. Príkazom **otlac** korytnačka otlačí *do svojej pozície* na plochu fázu, ktorou je práve zobrazená.

Príklad 3. Naprogramujme hru Black. Na ploche bude animačná korytnačka, ktorej tvarom je obrázok zložený z fáz-tvarov kamienkov. Korytnačku budeme ovládať klávesmi: šípkami sa bude pohybovať príslušným smerom, stlačením medzerovníka zmení fázu na nasledujúcu a stlačením klávesu **Enter** otlačí svoj tvar na plochu.

Riešenie:

Aby korytnačka kládla kamienky vedľa seba, bude chodiť s krokom rovnajúcim sa rozmeru kamienkov. V našom prípade sú kamienky štvorce s rozmermi 33 × 33.




```
viem black
  urob "kl klaves
  ak :kl = -72 [zmen.smer 0 do 33]
  ak :kl = -80 [zmen.smer 180 do 33]
  ak :kl = -75 [zmen.smer 270 do 33]
  ak :kl = -77 [zmen.smer 90 do 33]
  ak :kl = 32 [zmen.fazu faza + 1]
  ak :kl = 13 [otlac]
  black
koniec
```

Cvičenia

2. Vytvorte v projekte **Videostop** novú animačnú korytnačku **cena** a zmeňte jej tvar na niekoľkofázový obrázok, v ktorom každá fáza bude predstavovať vecnú cenu. V programe potom meňte tejto korytnačke fázy podľa dosiahnutých bodov. V prípade, že hráč bude mať 0 bodov, alebo ak získa najvyššiu cenu, program ukončite.
3. Naprogramujte „živú mozaiku". Jednotlivé kamienky budú animačné korytnačky tvaru obrázkovej premennej **vzor** z príkladu 3 a mozaika bude „žiť" tak, že sa v cykle vždy náhodne určí korytnačka, ktorá zmení svoju fázu na nasledujúcu.

Práca s textom

12

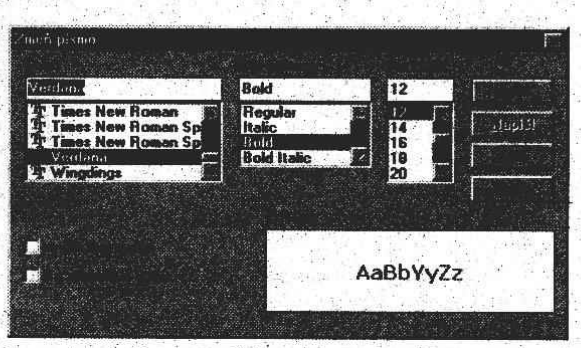
Už niekoľkokrát sme v príkladoch vypisovali do riadkov text príkazom **zobraz**. Napríklad:

```
? zobraz [Žirafa má dlhý krk.]
[Žirafa má dlhý krk.]
```

Texty, ktoré sme doteraz vypisovali, nazývame v Logu *vety*. Veta je zoznam slov uzavretý v hranatých zátvorkách. Napríklad **[Žirafa má dlhý krk.]** je veta, ktorá obsahuje 4 slová. Vety a slová sú údaje (rovnako ako čísla), a preto ich môžeme priraďovať do premenných:

```
? urob "moja.veta [Žirafa má dlhý krk.]
? zo :moja.veta
[Žirafa má dlhý krk.]
```

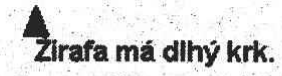
Podobne ako do riadkov, vieme v Logu vypísať text aj na plochu. Urobíme tak príkazom **text**. Zadaný text vypíše korytnačka bez zátvoriek aktuálnou farbou pera v svojej pozícii. Po vypísaní textu korytnačka svoju pozíciu nezmení. Typ písma, ktorým má byť text vypísaný, nastavíme príkazom **zmen.pismo**.




```
? zmen.pismo
```

otvorí sa pomôcka, v ktorej zvolíme typ písma

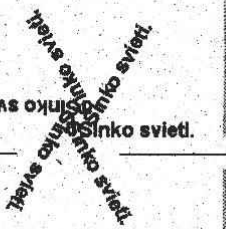
```
? zmen.pismo [Verdana] [12 700 0 0 0]
? text [Žirafa má dlhý krk.]
```



Príklad 1. Napíšme korytnačkou na plochu text **[Slnko svieti.]** v tvare slnečných lúčov.

Riešenie:

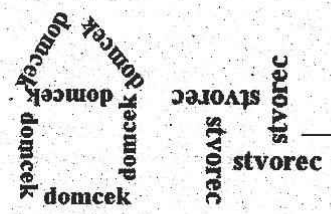
```
viem slnko
  opakuj 6 [text [Slnko svieti.] ~
            vp 360/6]
koniec
```



Príklad 2. Naprogramujme pohybujúci sa text.

Riešenie:

```
viem noviny :nadpis
  vl 90 ph do 5 pd vp 90
  zmen.fp 0 text :nadpis
  cakaj 10
  zmen.fp 15 text :nadpis
  noviny :nadpis
koniec
```



Cvičenia

1. Korytnačkou nakreslite na plochu obrázky:
2. Napíšte príkaz **3D :moj.text**, ktorý vypíše vstupný text 3D-efektom (ten istý text napísaný viackrát inou farbou pera a navzájom posunutý).

3. Naprogramujte jednoduchý šetrič obrazovky, v ktorom sa bude korytnačka posúvať po ploche a náhodnými farbami vypisovať daný text, pokým nestlačíme ľubovoľný kláves.

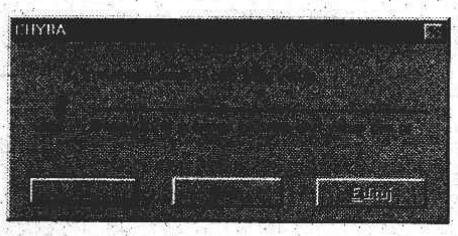
Počet slov vo vete zistíme operáciou **pocet**:

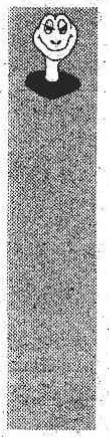
```
? zo pocet [Svet je gombička.]
3
```

Často budeme chcieť zistiť jednotlivé slová vety. Slúžia nám na to nasledujúce operácie:

prvy veta	**pr**	vráti prvé slovo z vety
posledny veta	**po**	vráti posledné slovo z vety
bez.prveho veta	**bez.pr**	vráti vetu bez prvého slova
bez.posledneho veta	**bez.po**	vráti vetu bez posledného slova

```
? urob "moja.veta [Svet je gombička.]
? zo pr :moja.veta
Svet
? zo po :moja.veta
gombička.
? zo bez.pr :moja.veta
[je gombička.]
? zo bez.po :moja.veta
[Svet je]
? zo pr bez.pr :moja.veta
je
? urob "moja.veta bez.pr bez.pr :moja.veta
? zo :moja.veta
[gombička.]
? urob "moja.veta bez.po :moja.veta
? zo :moja.veta
[]
? zo pocet :moja.veta
0
? zo pr :moja.veta
CHYBA: pr neobľubuje [] ako svoj vstup
```





Príklad 3. Vypíšme na plochu postupne sa skracujúcu vetu.

Riešenie:

```
viem napis :vstup
  opakuj pocet :vstup ~
      [text :vstup ~
       urob "vstup bez.pr :vstup ~
       vz 20]
koniec
```

Svet je gombička.

je gombička.

gombička.

Iné riešenie:

```
viem napis :vstup
  ak :vstup = [][ukonci]
  text :vstup
  vz 20
  napis bez.pr :vstup
koniec
```

Operáciami **bez.pr** a **bez.po** sme skracovali vetu. Nové slovo pridáme do vety týmito operáciami:

vloz.prvy *slovo veta*	výsledkom je veta s vloženým ***slovom*** na začiatku
vloz.posledny *slovo veta*	výsledkom je veta s vloženým ***slovom*** na konci

Nezabudnite, že **vloz.prvy** (krátko **vloz.pr**) a **vloz.posledny** (krátko **vloz.po**) sú operácie!

```
? vloz.pr "Naše [Logo je super]
CHYBA: Neviem, čo robiť s [Naše Logo je super]
? zo vloz.pr "Naše [Logo je super]
[Naše Logo je super]
? zo vloz.po "jazyk [Logo je super]
[Logo je super jazyk]
? zo vloz.po "jazyk vloz.pr "Naše [Logo je super]
[Naše Logo je super jazyk]
```

Príklad 4. Napíšme korytnačkou na plochu takýto nápis:

Riešenie:

```
viem napis2 :text
  opakuj pocet :text ~
     [text :text ~
      ph vz 20 ~
      urob "text vloz.posledny  pr :text  bez.pr :text]
koniec
```

Logo je super.
je super. Logo
super. Logo je

Cvičenie

4. Napíšte tieto nápisy:

Logo je super.
super. Logo je
je super. Logo

Logo
je
super.

super.
je
Logo

Užitočnou operáciou pri práci s vetami je operácia **vyber.mi**:

vyber.mi veta
Operácia **vyber.mi** náhodne vyberie jedno slovo z vety.

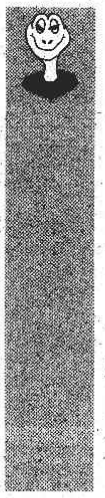
Príklad 5. Napíšme príkaz, ktorý bude generovať náhodné vety.

Riešenie:

```
viem vety
  urob "meno [Anička Zuzka Ferko Júlia Terezka Jožko]
  urob "sloveso [ľúbi neľúbi zbožňuje neznáša]
  urob "jedlo [tortu. palacinky. syr. mlieko. koláče. čokoládu.]
  zmaz
  opakuj 5 [urob "text [] ~
            urob "text vloz.pr vyber.mi :sloveso :text ~
            urob "text vloz.pr vyber.mi :meno ~
                      vloz.po vyber.mi :jedlo :text ~
            text :text vz 30]
koniec
```

Anička neznáša palacinky.
Anička zbožňuje mlieko.
Jožko ľúbi palacinky.
Júlia ľúbi tortu.
Jožko zbožňuje tortu.

Júlia zbožňuje čokoládu.
Ferko neznáša čokoládu.
Zuzka ľúbi syr.
Terezka neľúbi tortu.
Terezka neľúbi syr.



Všetky príkazy a operácie, ktoré sme sa v tejto kapitole naučili, pracujú aj so samotnými slovami.

Napríklad:

```
? zo pr "Ahoj
A
? zo vloz.po "! "Ahoj
Ahoj!
```

13 Práca s myšou

Grafickú plochu popisujeme pravouhlou súradnicovou sústavou, ktorej začiatok prechádza stredom plochy. Vďaka tomu vieme zistiť pozíciu, v ktorej sa korytnačka práve nachádza. Súradnice bodu, v ktorom korytnačka práve stojí, vráti operácia **pozicia** (skrátene **poz**). Samotnú *x*-ovú súradnicu tohto bodu vráti operácia **x.sur**, *y*-ovú súradnicu vráti operácia **y.sur**.
Do konkrétneho bodu presunieme korytnačku príkazom **zmen.poziciu** (krátko **zmen.poz**):

zmen.poziciu [x y]

x je *x*-ová a **y** je *y*-ová súradnica bodu, do ktorého chceme korytnačku presunúť. Iba *x*-ovú súradnicu korytnačke zmeníme príkazom **zmen.x číslo**, podobne *y*-ovú súradnicu zmeníme príkazom **zmen.y číslo**.

Podobne vieme zistiť natočenie korytnačky. Smer korytnačky je výslednou hodnotou operácie **smer**. Príkaz **zmen.smer** na zmenu natočenia korytnačky už poznáme.

Príklad 1. Napíšme príkaz, ktorý spôsobí, že sa korytnačka otočí k danému bodu na ploche a bude týmto smerom donekonečna pochodovať.

Riešenie:

```
viem smer.k.bodu :bod
  zmen.smer smer.k :bod
  pochoduj
koniec
```

```
viem pochoduj
  do 2
  pochoduj
koniec
```

smer.k [x y]
smer.k "meno.korytnačky

Vstupom operácie **smer.k** je bod alebo meno korytnačky. Jej výsledkom je smer, ktorý musí korytnačka zaujať, aby smerovala k danému bodu (ak je vstupom bod **[x y]**) alebo k pozícii danej korytnačky.

Príklad 2. Predefinujme predchádzajúce príkazy tak, aby sa korytnačka zastavila, keď príde k danému bodu.

Riešenie:

```
viem k.bodu :bod
  zmen.smer smer.k :bod
  pochoduj.k.bodu
koniec
```

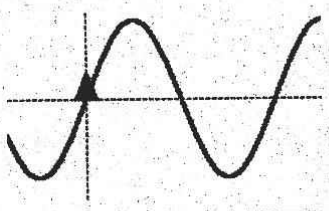
```
viem pochoduj.k.bodu
  do 2
  ak abs poz - :bod < 5 [ukonci]
  pochoduj.k.bodu
koniec
```

Výraz **abs :bod1 - :bod2** *počíta vzdialenosť bodov* **bod1** *a* **bod2**.

Príklad 3. Nakreslime korytnačkou graf funkcie sínus.

Riešenie:

```
viem sinusoida
ph zmen.poz [-500 0] pd
opakuj 1000 [zmen.x x.sur + 1~
             zmen.y 100 * sin x.sur]
koniec
```

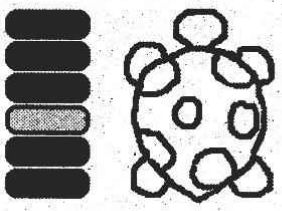


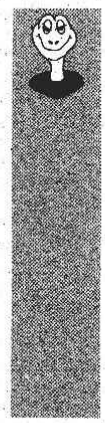
Operáciami **klaves?** a **klaves** sme sa naučili zisťovať, či bol na klávesnici stlačený kláves a rozpoznávať, ktorý kláves bol stlačený. Týmito operáciami môžeme rovnako zistiť i to, či bolo stlačené tlačidlo myši. Výsledkom operácie **klaves** je číslo **0**, ak bolo stlačené ľavé tlačidlo myši a číslo **-2**, ak bolo ľavé tlačidlo pustené. Súradnice bodu, v ktorom sme klikli myšou, sú výsledkom operácie **poz.mysi** (krátko **mys**).

Príklad 4. Naprogramujme si vymaľovávanku.

Riešenie:

```
viem vymaluj
  ak klaves = -2 [ zmen.poz mys ~
                   ak farba.bodu = 15 ~
                      [vypln] ~
                      [zmen.fv farba.bodu] ]
  vymaluj
koniec
```





Obrázok si môžeme pripraviť ako bitovú mapu v niektorom grafickom editore a do Loga vložiť funkciou *Prečítaj plochu* z menu *Súbory*. Tento obrázok vyfarbuje skrytá korytnačka, ktorá zmení pozíciu na pozíciu, kde sme klikli myšou a pozrie sa na farbu bodu pod sebou. Podľa farby bodu rozpoznáva, či sme klikli na paletu (vtedy korytnačka zmení farbu výplne). Ak sme klikli na bielu farbu, korytnačka oblasť vyplní.

Príklad 5. Napíšme príkaz, ktorým budeme vytvárať korytnačky na miestach, na ktoré klikneme myšou. Každej novej korytnačke nastavme náhodne smer, v ktorom sa bude odteraz pohybovať s krokom 2.

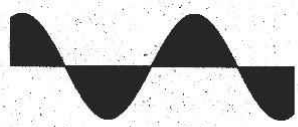
Riešenie:

Korytnačky budeme pomenovávať číslami **1**, **2**, **3**, ... V premennej **ktora** si budeme pamätať meno korytnačky, ktorá bude vytvorená pri najbližšom kliknutí myšou.

```
viem klikanie
  urob "ktora 1
  klikanie.mysou
koniec

viem klikanie.mysou
  ak klaves? [urob "kl klaves][urob "kl 555]
  ak :kl = 27 [ukonci]
  ak :kl = -2 [ urob.kor :ktora mys ~
                pre :ktora [zmen.smer nahodne 360]~
                zvys "ktora ~
                odteraz vsetky  ukaz ]
  do 2
  klikanie.mysou
koniec
```

Cvičenia



1. Nakreslite vyšrafovaný graf funkcie sínus.
2. Zadefinujte si vlastné funkcie (operácie) a nakreslite ich grafy.
3. Prepíšte príkaz **vymaluj** z príkladu 4 tak, aby nebolo možné vypĺňať čiernou farbou.
4. Napíšte príkaz, ktorým sa bude korytnačka pohybovať po ploche vždy smerom k bodu, kam klikneme myšou.

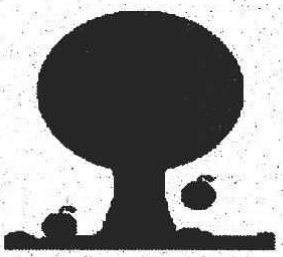
Ak máme na obrazovke ukázaných viac korytnačiek, môžeme myšou klikať nielen na plochu, ale aj na korytnačky. Operácia **zvolena** vráti meno korytnačky, na ktorú sme klikli. Ak sme neklikli na žiadnu korytnačku, operácia **zvolena** vráti prázdny zoznam **[]**.

Príklad 6. Na strome sú jablká (korytnačky tvaru jablka), ktoré chceme oberať myšou. Keď klikneme na jablko na strome, jablko spadne na zem pod stromom.

Riešenie:

```
viem jablka
  ak klaves = -2 ~
     [ak zvolena <> [] [odteraz zvolena ~
                        padaj]]
  jablka
koniec

viem padaj
  ak farba.bodu = 8 [ukonci]
  vz 1 padaj
koniec
```



Príklad 7. Naprogramujme akvárium, v ktorom budú plávať rybky (niektoré v smere zľava doprava, iné v opačnom smere) a smerom nahor budú stúpať vzduchové bubliny. Kliknutím na prázdne miesto na ploche sa vytvorí ďalšia rybka alebo bublina. Klinutie na už existujúcu rybku zmení smer jej plávania na opačný. Kliknutie na bublinu nespôsobí nič.

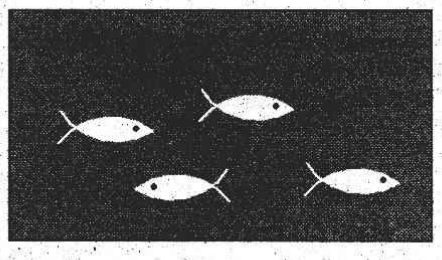
Riešenie:

```
? urob "ktora 0

viem rybicky
  ak klaves? ~
     [urob "kl klaves]~
     [urob "kl 555]
  ak :kl = -2 [ak zvolena = []~
                 [vytvor]~
                 [otoc zvolena]]
  do 2
  rybicky
koniec

viem vytvor
  zvys "ktora
  urob.kor :ktora mys
  pre :ktora ~
      [ak nahodne 100 > 80 ~
          [zmen.tvar :bubliny zmen.smer 0] ~
          [zmen.tvar :ryby zmen.smer 90 + ( nahodne 2 ) * 180 ]
  odteraz vsetky
  ph ukaz
koniec

viem otoc :ktora
  pre :ktora [ak smer <> 0 [vp 180]]
koniec
```

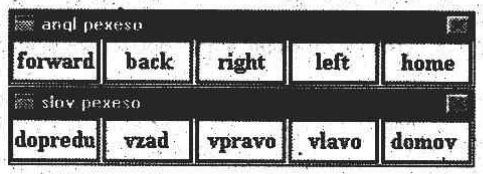


Príklad 8. Naprogramujme pexeso.

Riešenie:

Na ploche vytvoríme 10 animačných korytnačiek (klikaním na plochu alebo vlastným príkazom), budú sa volať `1`, `2`, ... , `10`. Korytnačkám `1`, ... , `5` zmeníme tvar na `angl.pexeso`, korytnačkám `6`, ... , `10` zmeníme tvar na `slov.pexeso`. Samotnú hru nám umožnia hrať nasledujúce príkazy:

```
viem pexeso
  zamiesaj.karticky
  odteraz vsetky ph ukaz
  odteraz []
  hraj.pexeso
koniec
```



Na začiatku sú všetky korytnačky neaktívne. Prvé kliknutie urobí jednu korytnačku aktívnou, pri druhom kliknutí porovnáme fázu aktívnej korytnačky s fázou tej, na ktorú sme klikli. Ak sú fázy zhodné, korytnačky skryjeme (na skryté korytnačky sa nedá kliknúť).

```
viem hraj.pexeso
  ak klaves = -2 [ak zvolena <> [] [vyber.mi.karticku zvolena]]
  hraj.pexeso
koniec

viem vyber.mi.karticku :ktora
  ak kto = [] ~
     [odteraz :ktora  zmen.poz [-275 100]] ~
     [ak kto <> :ktora [testuj :ktora]]
koniec
```

V príkaze `vyber.mi.karticku` testujeme, či kartička, na ktorú sme klikli, je prvá alebo druhá vybraná. Ak je prvá, operácia `kto` vráti prázdny zoznam `[]`. Ak je to druhá v poradí, testujeme fázy.

```
viem testuj :ktora
  pre :ktora [zmen.poz [-55 100]]
  ak faza = pre :ktora [faza] ~
     [skry  pre :ktora [skry]] ~
     [domov pre :ktora [domov]]
  odteraz []
koniec
```

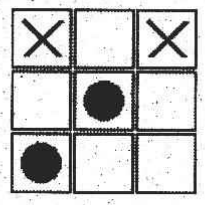
Príkaz `zamiesaj.karticky` *tu neuvádzame, ponúkame aspoň návod: kartičky pexesa pomiešajte napríklad tak, že dvadsaťkrát náhodne vyberiete dve kartičky a navzájom im vymeníte pozície.*

Cvičenia

5. Preprogramujte vymaľovávanku z príkladu 4 tak, že paletu budú tvoriť korytnačky (farebné štvorce). Menami korytnačiek budú čísla farieb, ktoré predstavujú.

6. Naprogramujte známu hru Tic-Tac (Piškvorky 3×3). Každé hracie políčko nech je reprezentované animačnou korytnačkou s tvarom z obrázkovej premennej `znak`. Po kliknutí na ešte neoznačené políčko sa zmení fáza príslušnej korytnačky podľa toho, ktorý hráč je práve na ťahu.

7. Vytvorte na ploche korytnačky, ktorých tvarmi budú obrázky základných geometrických útvarov (kružnica, trojuholník, ...). Potom naprogramujte príkaz, ktorým sa bude ovládať skrytá korytnačka `0` takto: ak klikneme na niektorú z korytnačiek, korytnačka `0` zmení svoj tvar na tvar zvolenej korytnačky a ak klikneme na plochu, korytnačka `0` otlačí svoj tvar na zvolené miesto.



Použitá literatúra

[1] *Abelson, H.:* TI LOGO : Education. Dallas; Texas : Texas Instruments Incorporated, 1984

[2] *Berentes, D.:* Apple Logo : A Complete Illustrated handbook. USA : TAB BOOKS Inc., 1984

[3] *Blaho, A. – Kalaš, I.:* Comenius Logo : tvorivá informatika. Bratislava : CL Group, spol. s.r.o., 1998

[4] *Blaho, A. – Kalaš I. – Máťušová, M.:* Symbolic computations and Logo. Bratislava : Comenius University; Department of Informatics Education Faculty of Mathematics and Physics, 1994

[5] *Blahová, V. a i.:* Comenius LOGO pre Windows. Bratislava : Metodické centrum, 1997

[6] *Blavočienė, T. – Dagienė, V. – Klupšaitė, A.:* LOGO žinynas. Vilnius : Folium, 1996

[7] *Blavočienė, T. – Dagienė, V. – Klupšaitė, A.:* Aš mokausi LOGO (Projekty Knyga mokiniams). Vilnius : Folium, 1997

[8] *Clayson J.:* Visual Modeling with LOGO : A Structual Approach to Seeing. Cambridge; London : The MIT Press, 1988

[9] *Cvik, P. – Tomcsányi, P. – Blaho, A.:* Detské programovacie jazyky I : ŽOFKA. Bratislava : Smena, 1988

[10] *Nikolov, R. – Sendova, E.:* Načala informatiky : Jazik Logo. Moskva : Nauka, 1983

[11] *Nikolov, R. – Sendova, E.:* Ezik i matematika : Logo za vtori klas. Sofia : Jusautor, 1984

[12] *Shimabukuro G.:* Thinking in Logo : A sourcebook for Teachers of Primary Students. Menlo Park : Addison-Wesley, 1988

[13] *Stuur, A. - Turcsányiné Szabó, M.:* Comenius Logo : Játék és Programozás. Budapest : Kossuth Kiadó, 1998

[14] *Tomcsányiová, M.:* Comenius LOGO a Windows. Bratislava : Pytagoras , 1995

[15] *Walat A.:* Wprowadzenie do jezyka i srodowiska Logo Komeniusz. Warszawa : Osrodek edukacji informatycznej i zastosowan komputerów, 1996