

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH
PEDAGOGICKÁ FAKULTA



LOGO a matematika

učební text pro studenty výběrového semináře

PaedDr. Dana Tržilová, CSc.

1993

1 Obsah

1	Obsah	1
2	Úvod	3
3	Filosofie jazyka LOGO	4
3.1	Charakteristika LOGA z hlediska programování	4
3.2	Charakteristika LOGA z hlediska procesu vzdělávání	6
4	Programovací jazyk LOGO	8
4.1	Primitiva jazyka LOGO	8
4.2	Přehled primitiv jazyka LOGO	9
4.2.1	Grafické příkazy a funkce jazyka LOGO	9
4.2.2	Matematické operace a funkce jazyka LOGO	12
4.2.3	Příkazy a operace jazyka LOGO pro práci s textem	13
4.3	Řídící konstrukce	17
4.3.1	Příkaz cyklu	17
4.3.2	Podmíněný příkaz	18
4.4	Procedura	20
4.4.1	Procedura se vstupem	23
4.4.2	Procedury s výstupem	26
4.5	Rekursivnost procedur	26
5	Řešení matematických problémů pomocí LOGA	30
5.1	Planimetrie	30
5.1.1	Pravidelné mnohoúhelníky	31
5.1.2	Parketáž	34
5.1.3	Symetrie geometrických útvarů	40

5.1.4	Geometrické obrazce s konstantním obsahem	48
5.2	Modelování pohybu živočichů	49
5.3	Fraktály	55
6	Literatura	64

2 Úvod

"Mladí lidé se dnes dělí na dvě skupiny: Jedni tráví svůj volný čas u hracích automatů a do omrzení hrají počítačové hry, druzí se baví programováním. Za deset, patnáct let budou ti druzí šéfy těch prvních"

"Teorie programování nás učí, jak metodicky správně analyzovat problém, navrhnout správný algoritmus a napsat přehledný, čitelný a strukturovaný program. Praxe nás učí, jak se vyrovnat s tím, že nic z toho pořádně neumíme. Programování není žádná idylka - je to boj. Boj s problémem, překladačem, operačním systémem, počítačem, nečekanými situacemi, které se nenajdou v učebnicích, a především s vlastní hloupostí, neznalostí a netrpělivostí."

Ivan Kopeček, Jan Kučera: Programátorské poklesky

(Mladá fronta, Praha 1989)

Skriptum je určeno učitelům ZŠ, kteří chtějí podchytit zájem dětí o výpočetní techniku a využít jej k rozvoji jejich tvůrčích schopností a matematických znalostí. Jako prostředek pro komunikaci s počítačem byl vybrán programovací jazyk LOGO, neboť i s malými znalostmi z oblasti programování je zde možno tvořit velmi demonstrativní a atraktivní programy.

Skriptum je rozděleno do dvou částí. První část (kap. 3 a 4) obsahuje popis jazyka LOGO, jejím cílem je uvést do problematiky programování v LOGU. V druhé části (kap. 5) jsou uvedeny příklady matematických poznatků a problémů, které mohou být zkoumány a řešeny použitím LOGA.

3 Filosofie jazyka LOGO

Koncem sedmdesátých let vytvořila skupina vědců vedená Seymour Papertem v laboratoři umělé inteligence na Massachusetts institutu v USA projekt LOGO, jehož myšlenky a prostředky se dnes používají téměř na celém světě pro seznámení dětí se základy programování. Tento projekt představuje určitou filosofii vzdělávání založenou na nových poznatcích vývojové psychologie, matematiky a výpočetní techniky. Projekt klade velký důraz na to, aby bylo prostřednictvím výpočetní techniky vytvořeno adekvátní prostředí pro rozvoj kognitivních dovedností a schopností žáka, aby počítač byl začleněn do vlastního světa dětí. K tomuto účelu projekt využívá nové výukové metody a formy.

3.1 Charakteristika LOGA z hlediska programování

Se zřetelem na cíle projektu skupina navrhla programovací jazyk, který se rovněž nazývá LOGO. Jako základ byl vzat programovací jazyk LISP používaný zejména v oblasti umělé inteligence. Programovací jazyk LOGO má široké možnosti uplatnění:

- obsahuje příkazy pro práci s grafikou počítače, po obrazovce lze kreslit barevné obrázky, kresby mohou dokonce jít mimo obrazovku, neboť grafický prostor je větší než prostor obrazovky;
- ačkoliv LOGO nebylo navrženo pro práci s čísly, lze zde manipulovat se všemi výpočty, které potřebujeme v programu;
- jazyk LOGO je rovněž prostředkem pro zpracování textových informací, tato část LOGA pracující se slovy a seznamy umožňuje např. tisk slov v různém tvaru, sestavování programů vytvářejících věty, popř. programů simulujících rozhovor apod.

Z hlediska programování lze jazyk LOGO charakterizovat těmito vlastnostmi:

a) je procedurální a rozšiřitelný

V LOGU vše probíhá pomocí procedur. Procedura je množina příkazů programovacího jazyka, která definuje posloupnost operací prováděných při zpracování programu. Má své jméno - název procedury. Příkazy v procedurách jsou tvořeny LOGO primitivy (základními příkazy a operátory jazyka LOGO) nebo jinými procedurami. Procedury mohou spolu navzájem komunikovat přes své vstupy a výstupy. Procedurální

podstata jazyka LOGO umožňuje programátorovi sestavovat strukturované programy, tj. rozložit problém do částí a použít tyto části jako stavební bloky.

b) je interaktivní

Primitiva a procedury jazyka LOGO jsou vykonány ihned po zapsání do počítače - zpětná vazba je okamžitá a chyby mohou být opraveny hned jakmile se vyskytnou. Ovládání LOGO editoru (editor slouží k zápisu a opravě procedur) je jednoduché.

c) data jsou v LOGU uspořádána ve formě seznamů

Daty rozumíme údaje zpracovávané programem. Daty mohou být čísla, slova, podmínky nebo stavy popisovaných jevů. Programovací jazyky obvykle odlišují data od vlastního programu, tj. od posloupnosti příkazů, kterými jsou zpracovávána. V LOGU jsou data strukturována seznamy. Seznamy obsahují uspořádanou posloupnost prvků, kterými mohou být čísla, slova nebo jiné seznamy.

d) je rekursivní

Podstatou rekurse jsou jevy vnořené do sebe, vzniká tak struktura o několika úrovních, v níž každá úroveň je reprezentovaná útvarem stejného typu. Použití rekurse v programování lze zjednodušeně charakterizovat tak, že uvnitř nové, právě definované procedury, použijeme název této procedury. Příkazy této procedury se potom opakovaně provádějí až do splnění určité podmínky. Vytvářením rekursivních procedur v LOGU získáváme krátké a elegantní programy, které kreslí poměrně složité obrázky.

Jak bylo výše řečeno skupina vedená Seymour Papert nebyla tvořena pouze týmem počítačových odborníků, ale sdružovala i psychology a pedagogy. Důvodem byla snaha vytvořit pro uživatele přirozené prostředí, které jim umožňuje objevovat, zkoumat a získávat nové znalosti a nedopustit, aby uživatel byl nucen akceptovat omezení na daném stupni výpočetní techniky (rychlost počítače, velikost operační paměti apod), která jsou někdy interpretována jako nutná součást programování.

3.2 Charakteristika LOGA z hlediska procesu vzdělávání

Východiskem vyučovacích metod používaných v LOGU je Papertova filosofie přístupu k počítačovému vzdělávání. Vychází z Piagetovy teorie procesu akomodace a asimilace nové informace do již existujícího schématu jedince. Tj. poznatek, který jedinec získává, je formován a usnadněn poznatky, které již jedinec má. Odtud vyplývá, že je to jedinec, kdo vytváří svůj systém procesu poznávání. Tento proces

může být pouze omezeně urychlen vnějšími činiteli, protože ostatní nesdílejí tutéž strukturu poznávání. Jelikož děti jsou při vyučování spíše "krmeni informacemi" (Papert, S: Mindstorms), místo toho aby byli aktivní při procesu učení, je jim odpírána možnost učit se přirozeným způsobem. Papert proto obhajuje "learning-by-doing" a "self-discovery" přístup k učení, tj. jedinec se učí na základě svého objevování, které probíhá jeho vlastním tempem a způsoby vyhovujícími jeho stylu myšlení. Jako prostředí umožňující aktivní učení byla vytvořena želví geometrie, ve které dítě "učí počítač" kreslit obrázky prostřednictvím zadávání příkazů želvě. Papert předpokládá aktivní učení v průběhu interakce žáka s počítačem, kdy počítač přesně vykoná zadané příkazy. Vychází z předpokladu, že počítačem realizovaný výsledný tvar ukáže všechny udělané chyby a dítě na základě sestavování obrázků a odstraňování vzniklých chyb proniká např. do problematiky geometrie. Otázkou ovšem je, zda proces učení může být efektivní, jestliže je zcela ponechán na procesu samoobjevitelské strategie. V současné době existuje řada výzkumů ukazujících, že důležité pojmy samy o sobě nepříjdou v nestrukturovaném přístupu. Tyto výzkumy zdůrazňují důležitost zásahů učitele do práce žáků. Nesporné jsou však následující vlastnosti jazyka LOGO:

a) prostředí jazyka LOGO je pro žáky velmi motivující

Autoři vyšli z předpokladu, že efektivita učení závisí na hloubce zapojení žáků do řešení problému a využili silných motivačních možností počítačové grafiky (všeobecně je známá např. motivační síla počítačových her). Výzkumy ukazují, že práce v LOGU je pro žáky přitažlivá - vnitřně motivující. LOGO lze použít jako katalyzátoru pro povzbuzení závislosti žáka na učiteli.

b) LOGO obsahuje prvky podobné přirozenému jazyku

LOGO bývá někdy označováno jako **jazyk pro učení**, neboť má některé prvky podobné přirozenému jazyku:

- větší jazykové jednotky jsou vytvářeny z menších jazykových jednotek;
- základními prvky LOGA jsou slova (název LOGO pochází z řeckého slova logos, což znamená slovo nebo myšlenka), slova jsou používána jako názvy pro programy a data, pro vlastnosti a vztahy;
- sestavení programu znamená z hlediska dětí naučit želvu novému slovu.

c) LOGO umožňuje převedení abstraktních pojmů na pojmy konkrétní

Hned od počátku bylo ověřováno použití tohoto jazyka pro zpřístupnění poznatků ze světa počítačů. V této souvislosti se zvláště úspěšná ukázala grafická část LOGA - želví geometrie. Důležitým rysem LOGA z hlediska matematického vzdělávání je jeho schopnost vytvořit **konkrétní, často grafické, modely**

matematických problémů, se kterými může žák manipulovat. Tyto simulace, často nazývané mikrosvěty, ukazují pravidla a zákony určitého systému.

d) LOGO je široce použitelné - umožňuje provádět zajímavé věci jednoduchou formou

Tvůrci LOGA rovněž poukazují na širokou použitelnost tohoto jazyka: je dostatečně jednoduchý pro malé děti a zároveň i dostatečně silný pro pokročilé programátory.

4 Programovací jazyk LOGO

4.1 Primitiva jazyka LOGO

Programovací jazyk LOGO obsahuje asi 200 základních slov, která mají předem nadefinovaný význam (např. CLEAN - vymaž obrazovku). Tato slova nazýváme primitiva. Primitiva jsou tvořena příkazy, operátory a funkcemi.

Příkaz je prvek programovacího jazyka, který způsobuje provedení určité činnosti. Každý příkaz začíná názvem příkazu mnemotechnicky vyjadřujícím činnost příkazu (např. PRINT - tisk), za ním pak následují případné operandy příkazu (např. za názvem příkazu PRINT následuje to, co chceme tisknout). Některé implementace jazyka LOGO připouštějí více příkazů na jedné řádce, příkazy jsou pak od sebe odděleny mezerou. Řádku ukončíme stiskem klávesy ENTER.

napišeme-li PRINT 5 a stiskneme klávesu ENTER objeví se na další řádce 5

Operátorem rozumíme pravidlo určující operace, které je nutno provést s danými operandy. Aritmetické operátory umožňují vykonávat s čísly běžné aritmetické operace (sčítání, násobení apod.), řetězcové operátory umožňují např. spojit části slov, vybrat z daného slova určitou část apod. Protože výsledkem operace zpravidla bývá určitý výstup (např. součet, nové přeuspořádané slovo apod.), používáme je uvnitř příkazů, tj. uvádíme co chceme s tímto výstupem udělat. Chceme-li např. výsledek vytisknout na obrazovce použijeme příkaz PRINT:

```
PRINT 12 + 30  
42
```

Standardní funkce nazýváme některé často se vyskytující funkce, jako např. trigonometrické funkce (sinus, kosinus..), odmocnina, logaritmus apod. Tyto jsou v LOGU předem definované - jsou to primitiva, která vrací jednu hodnotu jako výsledek své činnosti (musíme tedy uvést co chceme s touto hodnotou udělat). Mezi standardní funkce jazyka LOGO patří např. SIN stupně, RANDOM číslo apod. Jako argumenty

standardních funkcí se mohou používat číselné hodnoty celočíselného i reálného typu nebo aritmetické výrazy, které mohou obsahovat i standardní funkce.

V LOGU můžeme používat i jiné než standardní funkce. Ty si však musíme v programu definovat sami (viz kap. 4.4.2)

4.2 Přehled primitiv jazyka LOGO

Následující přehled nejpoužívanějších primitiv jazyka LOGO je rozdělen do skupin podle jejich zaměření, tj. podle oblastí aplikace primitiv.

4.2.1 Grafické příkazy a funkce jazyka LOGO

Pro snadný vstup dětí do problematiky programování je vhodná především grafická část jazyka LOGO. Vytváření programů je zde motivováno učením želvy novým slovům. Proto se pro grafickou část LOGA vžil název "želví geometrie" (Turtle Geometry). Původní želva jazyka LOGO se pohybovala po podlaze a pomocí pera kreslila svou stopu. Ve většině současných verzí LOGA je želva representována malým světelným trojúhelníkem znázorněným na obrazovce počítače.

Základními vlastnostmi želvy jsou:

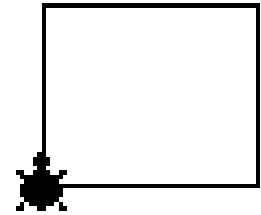
- a) poloha - nachází se na určitém místě podobně jako bod v euklidovské geometrii
- b) orientace - je otočena určitým směrem
- c) dynamičnost - může se pohybovat po obrazovce terminálu

Poloha a orientace želvy určují dohromady její stav. Jestliže je želva uprostřed obrazovky a míří nahoru, říkáme, že je v základním stavu. Stav želvy můžeme měnit pomocí grafických příkazů jazyka LOGO. Patří sem příkazy pohybující se želvou; příkazy ovlivňující zda želva bude či nebude za sebou zanechávat stopu; příkazy nastavující barvu pozadí na obrazovce, popř. barvu želvy a její stopy apod.

Polohu a směr želvy lze měnit dvěma druhy příkazů. První typ příkazů způsobuje **relativní** změny polohy a směru želvy, tj. výsledná změna závisí na předchozím stavu želvy. Polohu želvy měníme příkazy FORWARD (zkratka FD) a BACK (zkratka BK), které jsou následovány číslem popisujícím jak daleko má

želva jít, tj. obsahují jeden vstup. Jednotkou, pomocí níž je určována změna polohy želvy, je "krok želvy" - nejmenší vzdálenost, o kterou se může želva posunout. Směr želvy měníme příkazy RIGHT (zkratka RT) a LEFT (zkratka LT), jejichž vstup určuje o kolik stupňů se má želva otočit. Tyto příkazy umožňují ztotožnit se se želvou (zahrát si na želvu) a přenést zkušenosti ze svého pohybu na ovládání pohybu želvy a zprostředkovaně i na učení se poznatkům geometrie. Jako příklad uvedeme následující posloupnost příkazů pro nakreslení čtverce se stranou 100 želvích kroků (viz obr. 1):

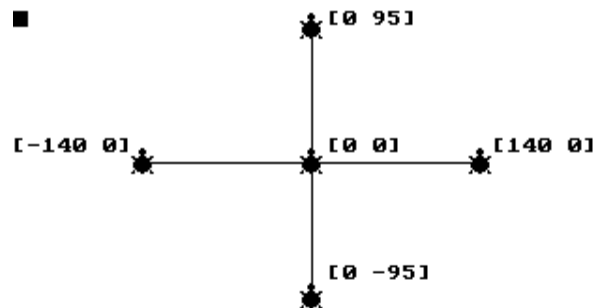
```
FORWARD 100 RIGHT 90
FORWARD 100 RIGHT 90
FORWARD 100 RIGHT 90
FORWARD 100 RIGHT 90
```



obr. 1.

Nakreslený obrázek vymažeme příkazem CLEARGRAPHICS (zkratka CG), želva bude umístěna do počáteční pozice (středu obrazovky) a nasměrována nahoru.

Druhý typ příkazů způsobuje **absolutní** změnu polohy a směru želvy. Využívá kartézský souřadnicový systém, kde poloha želvy je určena dvojicí souřadnic x, y ve tvaru $[x\ y]$ a výchozí poloha $[0\ 0]$ je ve středu obrazovky, želva je nasměrována k hornímu okraji obrazovky (obr. 2). Patří sem např. následující příkazy:



obr. 2. Souřadnicový systém v LOGU

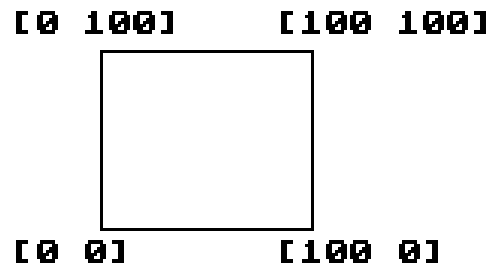
- SETPOS $[x\ y]$, který přesune želvu do pozice o souřadnicích x, y ;
- SETX popř. SETY, které mění pouze jednu souřadnici polohy želvy, obsahují jeden vstup, kterým je žádaná x -ová, popř. y -ová souřadnice (tzn. SETX posune želvu na dané x a y zůstane zachováno);
- příkaz SETHEADING (zkratka SETH) s jedním vstupem, tento příkaz otáčí želvu do žádaného směru (např. SETH 0 natočí želvu k hornímu okraji obrazovky, SETH 180 natočí želvu k dolnímu okraji obrazovky).

Informace o poloze želvy získáme užitím následujících funkcí:

- funkce POS oznámí souřadnice aktuální polohy želvy;
- funkce XCOR, popř. YCOR oznamující x -vou, popř. y -vou souřadnici;
- funkce HEADING oznamující směr želvy ve stupních.

Jako příklad uvedeme posloupnost příkazů pro nakreslení čtverce o straně 100 želvích kroků (obr. 3). Předpokládejme, že se želva nachází ve svém základním stavu.

```
SETPOS [0 100] SETPOS [100 100]
SETPOS [100 0] SETPOS [0 0]
```



obr. 3.

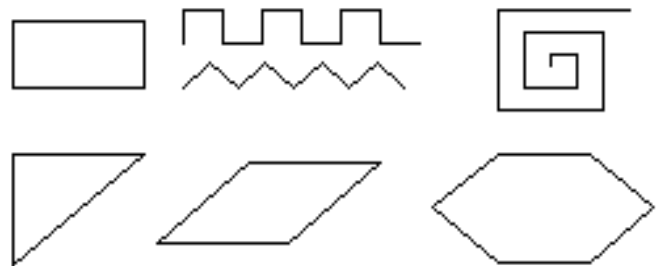
Pomocí výše uvedených příkazů (operátorů změny stavu) ovládáme pohyb želvy po obrazovce terminálu, přičemž želva může za sebou zanechávat stopu svého pohybu. Pero želvy nastavujeme příkazy PENDOWN (zkratka PD; položí pero - nastaví ho ke kreslení), PENUP (zkratka PU; zvedne pero - při pohybu želvy se nic nekreslí), PENERASE (zkratka PE; nastaví pero pro gumování).

Cvičení

1. Použijte pouze příkazy FORWARD číslo, RIGHT 90, RIGHT 45, LEFT 90, LEFT 45 k nakreslení následujících tvarů (obr. 4):

útvary snadné pro kreslení
 obdélník
 cik-cak čára
 spirála

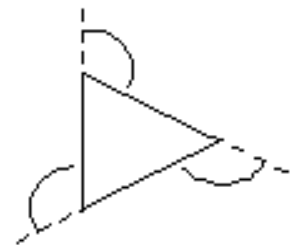
útvary vyžadující více přemýšlení
 trojúhelník
 kosočtverec
 šestiúhelník



obr. 4.

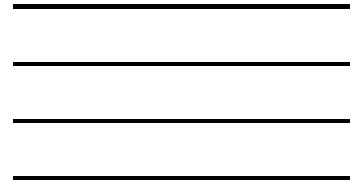
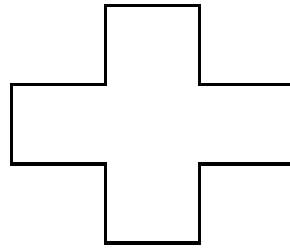
2. Použijte pouze příkazy FORWARD číslo, RIGHT číslo, LEFT číslo k nakreslení následujících obrazců:
 - a) rovnostranný trojúhelník
 - b) pravidelný pětiúhelník

pozn.: při kreslení geometrických obrazců želva "táhne" stopu za sebou, otáčí se tedy o velikost vnějšího úhlu obrazce (viz obr. 5)



obr. 5.

3. Použijte pouze příkazy SETPOS [x y], PENDOWN, PENUP pro nakreslení následujících obrázků (obr. 6). Pozn. pro snazší určení souřadnic je u složitějších obrázků výhodné si je překreslit na čtverečkovaný papír.



obr. 6.

4.2.2 Matematické operace a funkce jazyka LOGO

Nejpoužívanější aritmetické operace a funkce se v jazyku LOGO zapisují následujícím způsobem:

sčítání	+	druhá odmocnina	SQRT číslo
odčítání	-		
násobení	*		
dělení	/		

RANDOM číslo ... vrací náhodné celé nezáporné číslo menší než *číslo*

REMAINDER číslo1 číslo2 ... vrací zbytek po dělení čísla *číslo1* číslem *číslo2*

Aritmetické operace se v LOGU provádí zleva doprava, násobení a dělení má přednost před sčítáním a odčítáním.

Př.

23 . 34 - 25 : 5 zapíšeme v LOGU 23 * 24 - 25 / 5

druhá odmocnina ze 4 zapíšeme v LOGU SQRT 4

Příkaz PRINT ve spojení s aritmetickými operátory umožňuje použít počítač jako kalkulačku, např.: PRINT 5 * 6

30

Musíme však dodržovat následující pravidla:

- v zápisu desetinných čísel používáme desetinnou tečku místo desetinné čárky
- v aritmetických výrazech můžeme použít pouze kulaté závorky
- čísla a aritmetické operace je nutné oddělovat mezerami

Cvičení

1. Přepište uvedený aritmetický výraz do jazyka LOGO
(2 + 6,3) . [(4,5 + 0,8) : 3 + 20.5].
2. Nakreslete obrázek pravoúhlého trojúhelníka s odvěsnami 35 a 63 želvích kroků, výpočet velikosti přepony proveďte v LOGU.

4.2.3 Příkazy a operace jazyka LOGO pro práci s textem

Často je třeba sestavit programy zpracovávající nenumerické údaje. Toto nám umožňuje část jazyka LOGO pracující se slovy a seznamy. Patří sem řetězcové operace, které např. spojují slova nebo seznamy do větších útvarů, rozdělují je na menší části, popř. přeuspořádávají jejich části.

Slovem rozumíme libovolnou posloupnost písmen, číslic nebo speciálních znaků. Jednotlivá slova jsou od sebe oddělena mezerami. Slovo začínající písmeny označujeme uvozovkami (např. "NOVAK"), protože jinak by počítač chápal zápis jako název procedury. Slova obsahující pouze číslice, popř. desetinnou tečku, nazýváme čísla. Tyto není nutné označit uvozovkami. Výše uvedené předvedeme při použití příkazu pro tisk slov (příkaz PRINT):

```
PRINT "ADAM
ADAM
```

```
PRINT 123
123
```

```
PRINT ADAM
I DON'T KNOW HOW TO ADAM
```

V jazyku LOGO existují čtyři operace rozkládající slova:

<u>operátor</u>	<u>český překlad</u>	<u>význam</u>
FIRST	první	vrací první znak slova
BUTFIRST (BF)	bez prvního	vrací slovo bez prvního znaku
LAST	poslední	vrací poslední znak slova
BUTLAST (BL)	bez posledního	vrací slovo bez posledního znaku

Výsledky aplikace těchto operací na slovo IVANA ukážeme při jejich kombinaci s příkazem PRINT:

```
PRINT FIRST "IVANA
I
```

```
PRINT BF "IVANA
VANA
```

```
PRINT LAST "IVANA
A
```

```
PRINT BL "IVANA
IVAN
```

Operátory lze také kombinovat:

```
PRINT BL BL "IVANA
IVA
```

```
PRINT BF BL "IVANA
VAN
```

```
PRINT FIRST BF "IVANA
V
```

pozn. operátory se vyhodnocují
zprava

Jednotlivé části slov lze spojovat pomocí operátoru WORD (slovo):

```
PRINT WORD "DOKTOR "KA
DOKTORKA
```

```
PRINT (WORD "NE "ZVYK "ATI)
NEZVYKATI
```

Počet znaků daného slova zjistíme pomocí operátoru COUNT (spočti):

```
COUNT "TEXT
4
```

```
COUNT 123456
6
```

Libovolný počet slov lze dát dohromady ve formě seznamu. **Seznam** je soubor slov nebo dalších seznamů. Je ohraničen hranatými závorkami. Seznam může obsahovat:

slova	[DNES JE PEKNE POCASI]
čísla	[1 23 56 89]
symboly	[3 * X + (7 / 5)]
příkazy	[FORWARD 50 LEFT 90]
seznamy	[[JAN NOVAK] [EVA POTUCKOVA]]
být prázdný	[]

Pomocí přiřazovacího příkazu **MAKE "jméno výraz** (více viz kap. 4.4.1) lze nadefinovat proměnnou (její jméno je první parametr příkazu) a přiřadit jí hodnotu (druhý parametr - v našem případě seznam)
MAKE "JMENA [[JANA NOVAKOVA] [MIREK VOMACKA] [JOLANA BURIANOVA] [RADEK VOPATA]]

V LOGU lze manipulovat se seznamy podobně jako se slovy. Vybírání částí seznamů zabezpečují operátory **FIRST**, **BUTFIRST**, **LAST**, **BUTLAST**.

```
PRINT LAST :JMENA
RADEK VOPATA
```

```
PRINT FIRST FIRST :JMENA
JANA
```

```
PRINT LAST LAST :JMENA
NOVAKOVA
```

Spojování seznamů, popř. seznamů a slov, dohromady zabezpečují operátory **SENTENCE** a **LIST**.

Operátor **SENTENCE** (věta) vytváří z daných prvků jednoduchý seznam. Pokud vytváříme seznam z více než dvou prvků, musíme celý výraz uzavřít do kulatých závorek.

```
PRINT SENTENCE FIRST :JMENA [JE DOKTORKA]
JANA NOVAKOVA JE DOKTORKA
```

```
PRINT (SENTENCE FIRST :JMENA "A LAST :JMENA [JSOU SOUROZENCI])
JANA NOVAKOVA A RADEK VOPATA JSOU SOUROZENCI
```

Operátor **LIST** (seznam) vytváří z daných prvků strukturovaný seznam, seznam jehož prvky jsou parametry příkazu:

```
SENTENCE [1 2] [3 4]          LIST [1 2] [3 4]
[1 2 3 4]                    [[1 2] [3 4]]
```

Dále jsou zde operátory zajišťující přidání prvku do již existujícího seznamu **FPUT** (vkládá nový prvek na začátek seznamu) a **LPUT** (vkládá nový prvek na konec již existujícího seznamu).

Cvičení

- Pomocí příkazu PRINT a operátorů FIRST, BUTFIRST, LAST, BUTLAST vyberte ze zadaných slov požadované části:

<i>zadané slovo</i>	<i>požadovaná část</i>
MATEMATIKA	MATEMATIK
POKROK	KROK
POKLADNA	KLAD
- Pomocí příkazu PRINT a operátorů WORD, FIRST, BUTFIRST, LAST, BUTLAST vytvořte ze zadaných slov slova nová:

<i>zadaná slova</i>	<i>nové slovo</i>
MYSLIVEC, NA	MYSLIVNA
NE, SMYSL, NY	NESMYSLNY
CHEMIE, K	CHEMIK
PLYN, POMER	PLYNOMER
PRO, VOZIT	PROVOZ
- Pomocí příkazu MAKE vytvořte seznam JMENA, obsahující podstatná jména, a seznam SLOVESA, obsahující slovesa. Napište LOGO řádek, který
 - vybere určitý prvek ze seznamu JMENA a spojí ho do věty s určitým prvkem ze seznamu SLOVESA
 - přidá nový prvek na začátek seznamu JMENA
 - přidá nový prvek na konec seznamu SLOVESA
 - vytiskne počet prvků seznamu JMENA
 - rozšíří větu vytvořenou v bodě b) o další prvek z nově vytvořeného seznamu

4.3 Řídící konstrukce

Pro efektivnější vytváření programů používáme tzv. řídicí konstrukce. Jejich prostřednictvím např. uskutečníme opakování příkazů, popř. přerušíme posloupnost provádění příkazů a stanovíme jinou řádku nebo příkaz, který má být zpracován. Patří mezi ně příkazy pro realizaci cyklu, podmíněné příkazy a rekurse.

4.3.1 Příkaz cyklu

Příkaz cyklu zajišťuje opakované vykonávání v něm obsaženého vnitřního příkazu nebo skupiny příkazů. Má určitou stavbu (strukturu) - označujeme zde co se bude opakovat a za jaké podmínky se to bude opakovat. Podmínkou může být dosažení určitého počtu průchodů - jde pak o cyklus s předem stanoveným počtem opakování. Tento cyklus se v LOGU zapisuje slovem **REPEAT**, za nímž po mezeře následuje číslo označující počet opakování, a dále (opět po mezeře) následuje v hranatých závorkách posloupnost příkazů, které se mají opakovat. Zápis má tedy tvar:

REPEAT n [příkaz₁ příkaz₂ příkaz_x],

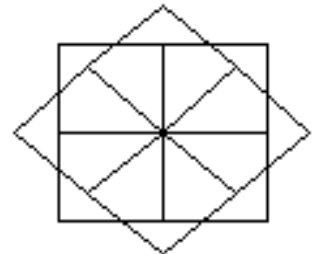
kde **n** je číslo označující počet opakování, **x** je počet příkazů. Jako příklad uvedeme opět posloupnost příkazů pro nakreslení čtverce o straně velikosti 100 kroků želvy:

```
CG
REPEAT 4 [FORWARD 100 RIGHT 90]
```

Seznam příkazů zapsaných v hranatých závorkách může obsahovat i další příkaz REPEAT, např.:

```
CG
REPEAT 8 [REPEAT 4 [FORWARD 50 RIGHT 90] RIGHT 45]
```

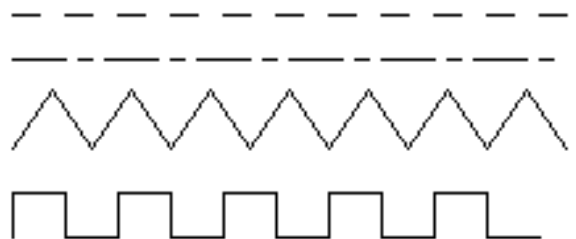
(viz obr. 7), kde vnitřní příkaz REPEAT 4 [FORWARD 50 RIGHT 90] kreslí čtverec o straně 50 kroků želvy.



obr. 7.

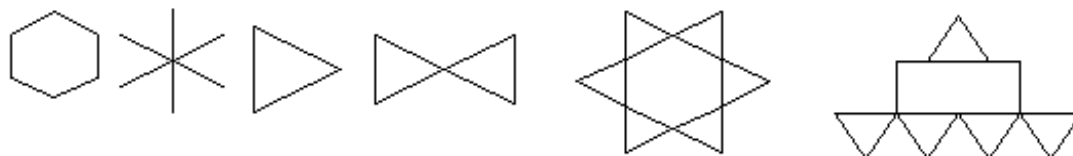
Cvičení

1. Použijte příkazy REPEAT, PU, PD, FD, RT nakreslete následující obrázky (obr. 8).



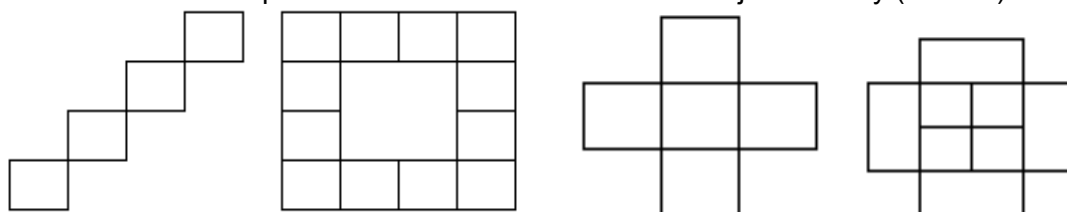
obr. 8.

2. Užitím příkazů REPEAT, FD, RT nakreslete následující obrázky (obr. 9). V některých případech je nutno užít příkaz REPEAT jako jeden z vnitřních příkazů příkazu REPEAT.



obr. 9.

3. Pomocí několikanásobného užití příkazu REPEAT nakreslete následující obrázky (obr. 10).



obr. 10.

4.3.2 Podmíněný příkaz

Vyjádření některých algoritmů vyžaduje použití struktury větvení v proceduře, tj. podmínit určitý postup splněním nějaké podmínky. Toto vyjadřujeme pomocí podmíněného příkazu, který zjišťuje zda je splněna určitá podmínka a v závislosti na výsledku vykoná nebo přeskočí příkazy, které jsou v něm vloženy. Podmínkou může být buď relace ($=$, $>$, $<$) nebo logické funkce vracející TRUE (pravda) nebo FALSE (nepravda):

EQUAL? x y	jsou-li objekty x y stejná slova vrací true, jinak false
EMPTY? x	je-li slovo nebo seznam x prázdný vrací true, jinak false
NUMBER? x	vrací true je-li objekt x číslo, jinak vrací false
LIST? x	vrací true je-li objekt x seznam, jinak vrací false
MEMBER? x y	vrací true je-li objekt x prvkem objektu y, jinak false
KEY?	vrací true je-li stisknuta klávesa, jinak false
NAME? x	je-li objekt x jméno proměnné vrací true, jinak vrací false

V podmínce můžeme také používat logické operátory NOT, AND, OR.

Podmíněné příkazy jsou dvojího druhu: neúplné a úplné. Neúplný podmíněný příkaz má následující tvar:

IF podmínka [seznam1]

Jestliže je podmínka splněna, provedou se příkazy uvedené v seznamu1, v opačném případě se příkazy uvedené v seznamu1 přeskočí. Konkrétní zápis je demonstrován na podmíněném příkazu zjišťujícím zda:

- číslo uložené v proměnné cislo je záporné číslo

```
MAKE "CISLO -25
IF :CISLO < 0 [(PRINT :CISLO [JE ZAPORNE CISLO])]
-25 JE ZAPORNE CISLO
```

```
MAKE "CISLO 25
IF :CISLO < 0 [(PRINT :CISLO [JE ZAPORNE CISLO])]
-
```

- číslo uložené v proměnné c je menší nebo rovno pěti

```
MAKE "C 5
IF OR :C < 5 :C = 5 [PRINT [PODMINKY SPLNENY]]
PODMINKY SPLNENY
```

Úplný podmíněný příkaz zapisujeme takto:

IFELSE podmínka [seznam1] [seznam2]

Jestliže je podmínka splněna, provedou se příkazy uvedené v seznamu1 a pokračuje se dále za seznamem2, v opačném případě (podmínka není splněna) se provedou příkazy uvedené v seznamu2 a pokračuje se dále. Konkrétní zápis je demonstrován a podmíněném příkazu:

- vybírajícím větší číslo z čísel uvedených v proměnných číslo1, číslo2

```
MAKE "CISLO1 25
MAKE "CISLO2 14
IFELSE :CISLO1 > :CISLO2 [PRINT :CISLO1] [PRINT :CISLO2]
25
```

- zjišťujícím zda číslo uvedené v proměnné c je číslo sudé nebo liché

```
MAKE "C 25
IFELSE (REMAINDER :C 2) = 0 [(PRINT :C [JE SUDE])]
[(PRINT :C [JE LICHE])]
25 JE LICHE
```

- zjišťujícím zda slovo EVA se vyskytuje v seznamu A

```
MAKE "A [EVA IVANA JANA]
IFELSE MEMBER? "EVA :A [PRINT [JE V SEZNAMU]]
[PRINT [NENI V SEZNAMU]]
JE V SEZNAMU
```

Cvičení

1. Sestavte neúplný podmíněný příkaz, který zjistí, zda v proměnné cislo je:
 - a) kladné číslo

- b) číslo dělitelné třemi
 - c) prázdné slovo
2. Sestavte úplný podmíněný příkaz, který zjistí zda v obsah proměnné číslo je číslo je kladné nebo záporné

4.4 Procedura

Program v jazyku LOGO nevytváříme jako jednoduchý celek, ale jako soustavu složenou z jednodušších částí, které řeší ucelené dílčí činnosti. Tyto části nazýváme procedury. Jedná se o množiny příkazů programovacího jazyka, které vykonávají určitou posloupnost operací, např. nakreslení čtverce, vypočtení aritmetického průměru apod.

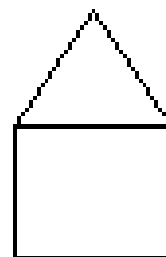
Procedura se skládá z hlavy a těla procedury a je ukončena slovem END. Hlavu procedury tvoří slovo TO, za ním následuje název procedury. Počet znaků v názvu procedury je prakticky neomezený - může být přes dvě řádky obrazovky; prvním znakem musí být písmeno. Název procedury volíme tak, aby co nejlépe vyjadřoval její funkci. Tělo procedury obsahuje popis provedení určité činnosti. Jako příklad uvedeme proceduru, která nakreslí čtverec o straně velikosti 50 kroků želvy.

```
TO CTVEREC
REPEAT 4 [FORWARD 50 RIGHT 90]
END
```

Nově vytvořenou proceduru můžeme používat jako každý jiný příkaz jazyka LOGO, voláme jí příkazem procedury obsahující název dané procedury, tj. napíšeme CTVEREC (viz obr. 11)

```
TO DUM
CTVEREC
FD 50 RT 30
TROJUHELNİK
END

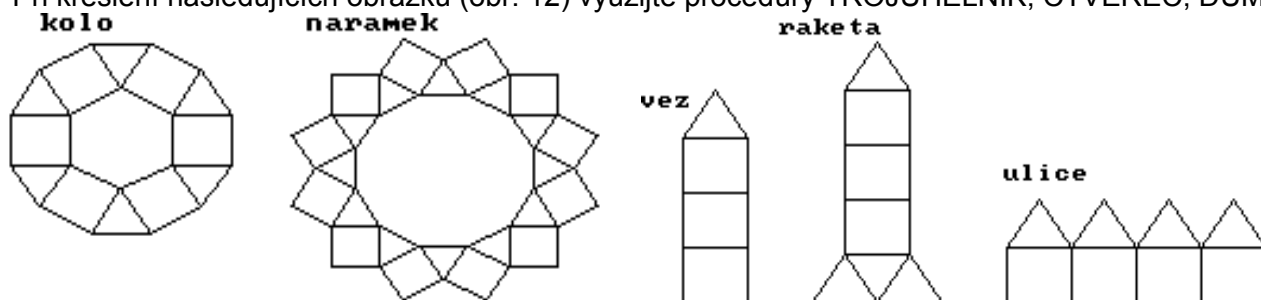
TO TROJUHELNİK
REPEAT 3 [FD 50 RT 120]
END
```



obr. 11.

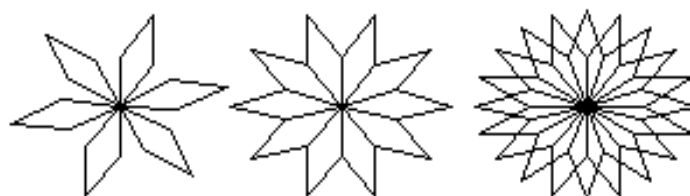
Cvičení

1. Při kreslení následujících obrázků (obr. 12) využijte procedury TROJUHELNÍK, CTVEREC, DUM.



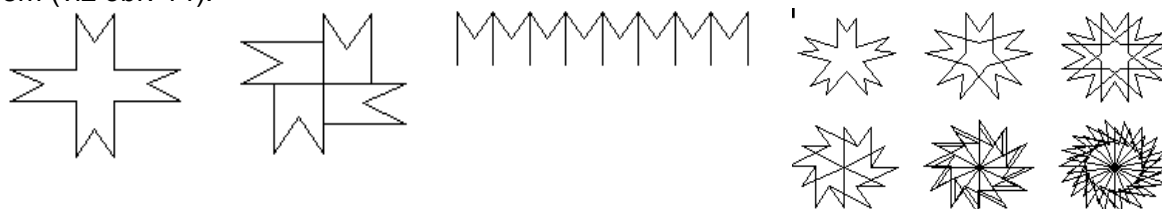
obr. 12.

2. Sestavte proceduru KOSOCTVEREC. Užitím této procedury nakreslete následující obrázky (obr. 13).



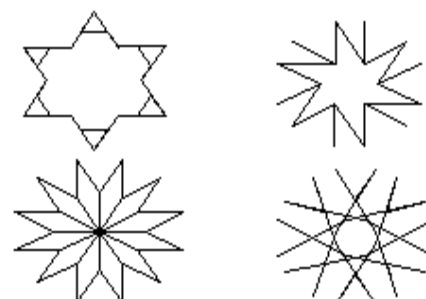
obr. 13.

3. Vyberte si nějaké písmeno a sestavte proceduru, která nakreslí zvolené písmeno v LOGU. Pomocí této procedury sestavte proceduru vytvářející z tohoto písmene řetěz a proceduru otáčející daným písmenem (viz obr. 14).



obr. 14.

4. Rozluštěte pomocí jakého písmene byly nakresleny následující obrázky a sestavte příslušné procedury (obr. 15).



obr. 15.

4.4.1 Procedura se vstupem

Při vytváření procedur se snažíme sestavovat procedury co nejobecnější, abychom je mohli použít v různých programech. Využíváme k tomu procedury s proměnnými. Proměnnou obecně rozumíme symbol, který může nabývat různých hodnot. V programování se proměnnou označuje místo v paměti počítače kam lze vložit hodnotu. Hodnota proměnné může být programem přepisována. V LOGU proměnnou zapisujeme slovem před nímž je uvedena dvojtečka. Slovem rozumíme řadu za sebou napsaných znaků, obsahujících alespoň jeden znak, který není číslicí. V LOGU existují lokální a globální proměnné.

Lokálnost proměnné je v LOGU vyjádřena jejím vypsáním v názvu procedury. Tato proměnná se nazývá parametr. Rozeznáváme formální parametry (označení proměnného údaje) a skutečné parametry (konkrétní hodnoty zadání). V obecně postavené proceduře se formální parametry při vložení do programu nahrazují skutečnými parametry. Volání procedury se potom uskutečňuje pomocí příkazu procedury, který se skládá z názvu procedury za nímž následují skutečné parametry. Jako příklad uvedeme proceduru kreslící čtverec:

```
TO CTVEREC :STRANA
REPEAT 4 [FORWARD :STRANA RIGHT 90]
END
```

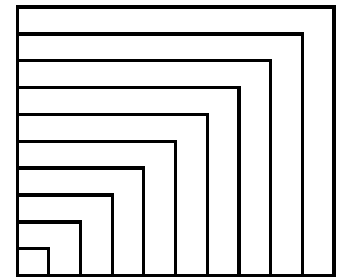
volání procedury CTVEREC 100

Tato procedura může kreslit různé čtverce (viz obr. 16), jejich velikost závisí na dosazené hodnotě parametru :STRANA. Lokální proměnná nabývá hodnoty jen během provádění procedury, v níž je zapsaná jako parametr. Po skončení této procedury je místo v paměti opět uvolněno.

Tedy jestliže například zkusíte vytisknout hodnotu proměnné :STRANA po skončení procedury CTVEREC 100 zjistíte, že je tato proměnná opět bez hodnoty:

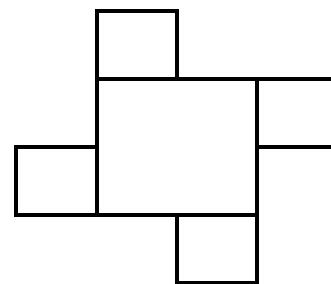
```
CTVEREC 100
PRINT :STRANA
STRANA HAS NO VALUE...(STRANA nemá hodnotu)
```

Skutečnost, že hodnota lokální proměnné je po skončení procedury z daného paměťového místa odstraněna, nám umožňuje použít tentýž název proměnné v různých procedurách, popř. použít jednu proceduru, např. CTVEREC, jako stavební blok v jiné proceduře, aniž bychom se museli starat o to co se děje uvnitř této podprocedury (obr. 17):



obr. 16. Opakované volání procedury CTVEREC se zvětšující se hodnotou parametru STRANA

```
TO CTVEREC.SE.CTVERCI :STRANA
REPEAT 4 [FD :STRANA CTVEREC (:STRANA / 2) RT 90]
END
```



obr. 17.
CTVEREC.SE.CTVERCI
50

Globální proměnná je dostupná v celém programu. Tuto proměnnou vytváříme, popř. měníme její hodnotu, přiřazovacím příkazem MAKE. Příkaz MAKE "STRANA 100 uloží hodnotu 100 do políčka označené STRANA:

```
MAKE "STRANA 100
PRINT :STRANA
100
```

Pozn.: hodnota lokální proměnné se neovlivní hodnotou globální proměnné ani naopak hodnota globální proměnné neovlivní hodnotu lokální proměnné:

```
MAKE "STRANA 100
PRINT :STRANA
100
```

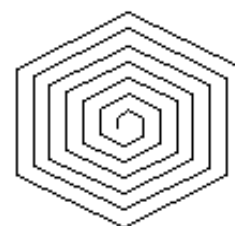
CTVEREC 60 ... nakreslí čtverec o straně 60 kroků želvy

```
PRINT :STRANA
100
```

Použití přiřazovacího příkazu pro vytvoření globální proměnné, popř. přiřazení hodnoty aritmetického výrazu určité proměnné ukážeme v následujících příkladech:

- procedura, která kreslí spirálu (obr. 18):

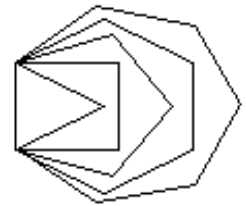
```
TO SPIRALA
MAKE "X 5
REPEAT 40 [FORWARD :X RIGHT 60 MAKE "X :X + 1]
END
```



obr. 18.
SPIRALA

- procedura, která kreslí pravidelný mnohoúhelník (obr. 19):

```
TO MNOHO :POCET :STRANA
MAKE "UHEL 360 / :POCET
REPEAT :POCET [FORWARD :STRANA RIGTH :UHEL]
END
```



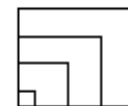
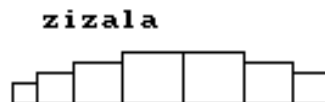
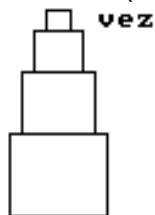
obr. 19.
Procedura
MNOHO
s různými
parametry

Přiřazovací příkazy se od matematických vzorců liší právě použitým symbolem přiřazení (MAKE). Tento symbol neznamena rovnost, ale nahrazení hodnoty proměnné hodnotou výrazu na jeho pravé straně. Např.

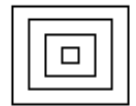
```
MAKE "n :n + 1, což nelze nahradit zápisem n = n + 1
```

Cvičení

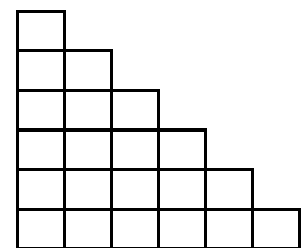
1. S využitím procedury CTVEREC :STRANA a příkazu MAKE sestavte procedury pro nakreslení následujících obrázků (obr. 20).



obr. 20.



2. Sestavte proceduru ROHACEK :POCET pro nakreslení následujícího obrázku (obr. 21). V proceduře využijte opakované volání procedury OBDELNIK :DELKA :SIRKA, kde budete měnit velikost parametrů :DELKA :SIRKA.



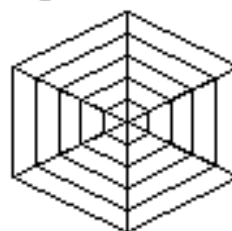
obr. 21. ROHACEK
6

3. Sestavte proceduru s parametrem TROJUHELNIK :STRANA, která nakreslí rovnostranný trojúhelník libovolné velikosti. S využitím této procedury a příkazu MAKE sestavte procedury pro nakreslení následujících obrázků (obr. 22).

rust troj

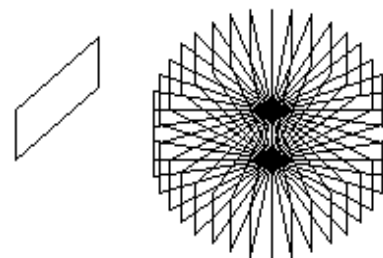


pavucina



obr. 22.

4. Sestavte proceduru s parametrem ROVNOBEZNIK :UHEL, která nakreslí rovnoběžník s konstantní velikostí stran (např. 30 a 60 kroků želvy) a s libovolnou velikostí vnitřních úhlů (parametr UHEL). S využitím této procedury a příkazu MAKE sestavte proceduru, která nakreslí následující obrázek (obr. 23).



obr. 23.

4.4.2 Procedury s výstupem

Jedná se o procedury, které dávají nějaký výsledek - definují hodnotu funkce, proto je nazýváme funkční procedury, popř. funkce. Při vytváření funkcí používáme příkaz **OUTPUT slovo**, který zastaví běh procedury a přiřadí jménu funkce hodnotu *slovo*. Příkladem je funkce dávající nám hodnotu čísla PI:

```
TO PI                                PRINT PI
OUTPUT 3.14159                       3.14159
END
```

funkce počítající druhou mocninu:

```
TO DRUHA.MOCNINA :CISLO             PRINT DRUHA.MOCNINA 16
OUTPUT :CISLO * :CISLO              256
END
```

popř. funkce využívatící opakované sčítání při výpočtu součtinu dvou čísel:

```
TO SOUCIN :A :B
MAKE "VYSLEDEK 0
REPEAT :B [MAKE "VYSLEDEK :VYSLEDEK + :A]
OUTPUT :VYSLEDEK
END
```

Cvičení

1. Sestavte proceduru ARITM.PRUMER :A :B pro výpočet aritmetického průměru dvou čísel a proceduru GEOM.PRUMER :A :B pro výpočet geometrického průměru dvou čísel.
2. Sestavte proceduru NA.CTVRTOU :CISLO, která vypočte čtvrtou mocninu daného čísla. Zobecněte použitý algoritmus opakovaného násobení při sestavení procedury MOCNINA :CISLO :EXPONENT, která vypočte libovolnou mocninu daného čísla.

4.5 Rekursivnost procedur

Podstatou rekurse jsou jevy vnořené do sebe, vzniká tak struktura o několika úrovních, v níž každá úroveň je representovaná útvarem stejného typu. Nižší úroveň je cele obsažena ve vyšší úrovni, tj. po zpracování nižší úrovně je možno vrátit se pouze do nejbližší vyšší úrovně. Aby vkládání dalších úrovní nepokračovalo donekonečna musíme předem stanovit podmínky, za kterých toto vnořování skončí. Podstatou rekurse je možno přiblížit pomocí:

- obrázkové rekurse - obrázek, na kterém je ten samý obrázek, na kterém je ten samý obrázek...
- pohádky o slepičce a kohoutkovi - vnořená přání jsou postupně plněna od posledního k prvnímu
- zajištění si libovolného množství splněných přání na základě nabídky alespoň dvou přání: "Mým druhým přáním je splnění dvou přání".

Rekursivní procedury se nazývají procedury, v jejichž těle se uskutečňuje jejich volání buď přímo jimi nebo prostřednictvím jiné procedury. Vnoření jména procedury do její definice se nazývá rekurzí a situace, kdy v průběhu procedury dojde na její vnořené jméno, se nazývá rekursivním voláním. Použití rekurse v programování lze zjednodušeně charakterizovat tak, že uvnitř nové, právě definované procedury, použijeme název této procedury. V průběhu procedury potom nastává postupné vnořování programu do sebe sama (jeví se jako opakované provádění příkazů této procedury), vnořování se zastaví splněním určité podmínky. Při dokončování postupně vnořených procedur dojde i na příkazy umístěné za rekursivním voláním. Vytvářením rekursivních procedur v LOGU získáváme krátké a elegantní programy, které kreslí poměrně složité obrázky.

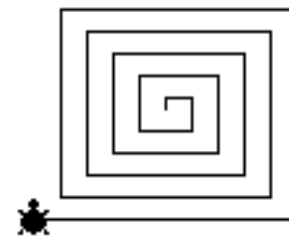
Při řešení úloh pomocí rekurse postupujeme následujícím způsobem:

- 1) najdeme takové počáteční podmínky, abychom úlohu uměli řešit
- 2) na základě algoritmu zjednodušené úlohy navrhne "opakovatelné jádro úlohy", které má následující vlastnosti:
 - pomocí něj přejdeme od složitějšího zadání k zadání jednoduššímu
 - na základě zjednodušeného výsledku najdeme výsledek úlohy původní
 - navržené postupné zjednodušování jednou skončí

Uvedeme dva příklady rekurentní procedury: v prvním příkladu (procedura SPIRALA) je rekursivní volání umístěno na konci procedury, tzn. "jádro úlohy" se bude rekurentně opakovat až do splnění určité podmínky; v druhém příkladu (procedura KRIDLA) je rekursivní volání umístěno uprostřed procedury, tzn. po ukončení vnořování procedur dojde při dokončování postupně vnořených procedur i na příkazy umístěné za rekursivním voláním:

procedura SPIRALA (viz obr. 24):

```
TO SPIRALA :STRANA
IF :STRANA > 100 [STOP]
FD :STRANA RT 90
SPIRALA :STRANA + 5
END
```



obr. 24.

procedura KRIDLA, která nakreslí křídla motýla (viz obr. 25):

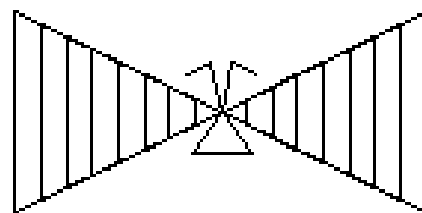
postup ad 1) nakreslení jednoduché mašličky, skládající se ze dvou trojúhelníků
ad 2)

```
REPEAT 3 [FD :VELIKOST RT 120]
REPEAT 3 [BK :VELIKOST RT 120]
```

vykonávání procedury zastavíme při určité minimální velikosti např. 20 kroků želvy

procedura:

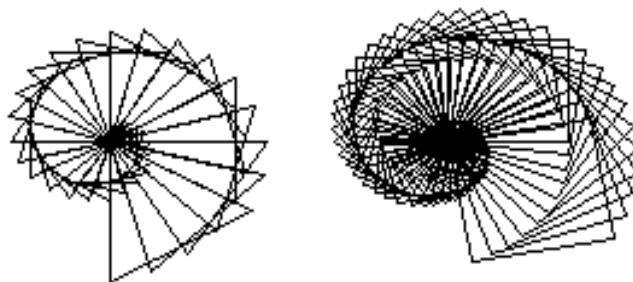
```
TO KRIDLA :VELIKOST
IF :VELIKOST < 20 [STOP]
REPEAT 3 [FD :VELIKOST RT 120]
KRIDLA :VELIKOST - 10
REPEAT 3 [BK :VELIKOST RT 120]
END
```



obr. 25.

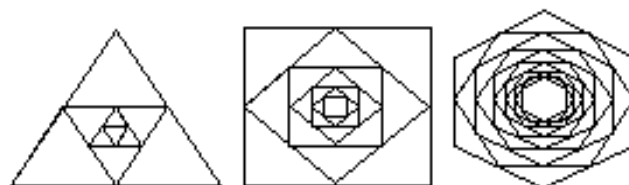
Cvičení

1. Sestavte rekurentní proceduru, která nakreslí obrázek "ulity" vytvořené z rotujících a zvětšujících se trojúhelníků (obr. 26). Jako podmínku pro ukončení kreslení stanovte dosažení určité mezní velikosti strany trojúhelníka. Analogický úkol řešte pro nakreslení "ulity" z rotujících čtverců.



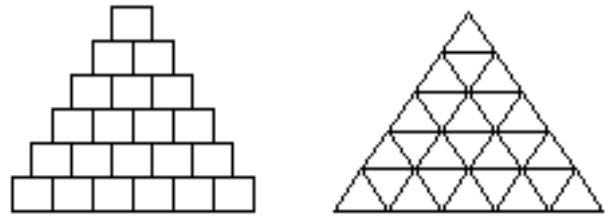
obr. 26.

2. Sestavte rekurentní proceduru, která bude postupně kreslit vepsané trojúhelníky. Postup kreslení zastavte po dosažení určité minimální velikosti strany. Analogickou úlohu řešte pro vepisování čtverců popř. šestiúhelníků (obr. 27).



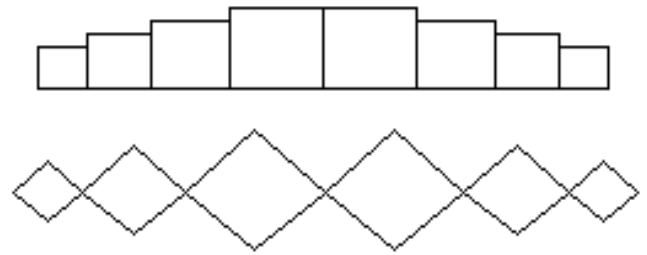
obr. 27.

3. Sestavte rekurentní proceduru pro nakreslení obrázku pyramidy. Procedura by měla obsahovat nakreslení řady n čtverců (zleva doprava), přesun želvy na levý kraj následující řady pyramidy a následné kreslení pyramidy, která má o jednu řadu méně (obr. 28).



obr. 28.

4. Sestavte procedury pro nakreslení následujících pásků (obr. 29). Rekurentní volání umístěte do středu procedury.



obr. 29.

5 Řešení matematických problémů pomocí LOGA

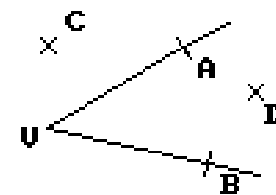
V této kapitole jsou ukázány některé z možností a prostředků výpočetní techniky při prezentaci matematických pojmů (kap. 5.1) a při matematickém modelování přírody (kap. 5.2 a 5.3). Cílem není výuka programovacího jazyka, ale demonstrace spojení matematiky a výpočetní techniky v praxi.

5.1 Planimetrie

Hlavním předmětem vyučování geometrie na ZŠ je vyšetřování vlastností rovinných obrazců. Přínosem LOGA v této oblasti je želví geometrie, která poskytuje velké množství různorodých problémů z geometrie dvourozměrného prostoru (planimetrie). Prostřednictvím ovládnutí pohybu želvy a nastavení pera žák kreslí různé obrázky a je veden ke zkoumání vlastností geometrických útvarů, k experimentování s nimi. Protože želva není tak abstrahována od všech vlastností jako např. bod v euklidovské geometrii, lze ji snáze přirovnat k důvěrně známým objektům. "Želva" tak získává schopnost sloužit jako první presentace formální matematiky, poskytuje žákům dynamickou metaforu (představu) matematických pojmů.

Jako příklad uvedeme LOGEM realizovanou propedeutiku pojmu úhel. V matematice na ZŠ lze úhel definovat různými způsoby, z nichž každý má své opodstatnění:

- úhel jako část roviny omezená dvěma polopřímkami (např. VA , VB) se společným počátkem, tyto polopřímky však vymezují dva různé úhly - konvexní úhel AVB (rovinný útvar se nazývá konvexní, jestliže s každými dvěma různými body X, Y obsahuje celou úsečku XY) a konkávní úhel AVB ; v učebnici pro 4.roč. ZŠ je konkrétní úhel určen bodem dané části roviny (obr. 30 - úhel AVB určený bodem C , úhel AVB určený bodem D)



obr. 30.

- úhel jako průnik dvou polorovin, jejichž hraniční přímky se protínají - takto lze definovat pouze úhel konvexní.

Vlivem neporozumění klasickým definicím pojmu úhel některé děti inklinují k chápání úhlu v pojmech délka a plocha (např. chybné uvažování dětí "čím je větší trojúhelník tím je větší součet jeho vnitřních úhlů"). Tomuto lze zamezit prací dětí s LOGEM, neboť zde je pojem úhlu spojen s otočením želvy. Experimentováním v LOGU žáci získávají zkušenosti s užíváním pojmu úhel (např. otočení se o pravý úhel doleva má tentýž účinek jako třikrát opakované otočení se o pravý úhel doprava, apod.). Musíme však vzít

v úvahu, že ne všechny děti pochopí vztah mezi otočením a úhlem, popř. skutečnost že některé děti používají při práci v LOGU pouze násobky 30 a 45 stupňů. LOGO tedy chápeme pouze jako pomůcku umožňující experimentální práci s matematickými pojmy a možnost ukázat tyto pojmy v jiných souvislostech.

Konkrétní podoba práce v LOGU je uvedena v následujících podkapitolách, každá z nich je věnována jednomu tematickému celku - mikrosvětlu pro experimentování s matematickými pojmy.

5.1.1 Pravidelné mnohoúhelníky

Pravidelný mnohoúhelník je mnohoúhelník, jehož všechny strany a všechny vnitřní úhly mají stejnou velikost. Nejjednodušším pravidelným mnohoúhelníkem je rovnostranný trojúhelník, dále čtverec, pravidelný pětiúhelník, pravidelný šestiúhelník atd. Jednoduchý algoritmus, který popisuje želvě postup nakreslení pravidelného mnohoúhelníka může mít následující tvar:

1. popojdi dopředu o konstantní vzdálenost;
2. otoč se doprava o konstantní úhel;
3. dokud se nedostaneš do výchozího bodu, opakuj body 1. a 2.

Příslušnou proceduru nazveme **MNOHO** :

```
TO MNOHO :STR :UHEL
FD :STR
RT :UHEL
MNOHO :STR :UHEL
END
```

Pokud do této procedury dosadíme za vstupní úhel hodnoty 90, 120, 144, 135, pak dostaneme čtverec, trojúhelník, pěticípou a osmicípou hvězdu (obr. 31 a 32). Jedná se o rekurzivní proceduru, jejíž činnost na rozdíl od výše uvedeného algoritmu není ukončena. Dříve než proceduru upravíme, provedeme si rozbor její činnosti. Ze zápisu procedury a z obr. 31 je zřejmé, že **MNOHO** se vstupním úhlem $360/n$ stupňů nakreslí pravidelný n -úhelník. Formule **počet stran x úhel = 360°** však obecně neplatí - např. neexistuje celočíselný násobek čísla 144 rovnající se číslu 360, ale procedura **MNOHO** se vstupním úhlem 144° kreslí pravidelný mnohoúhelník (viz obr. 32). Skutečnost že $5 \times 144^\circ = 720^\circ$ je násobek 360° vede k následující formuli:

$$(\text{počet stran}) \times (\text{úhel}) = (\text{celé číslo}) \times 360^\circ$$

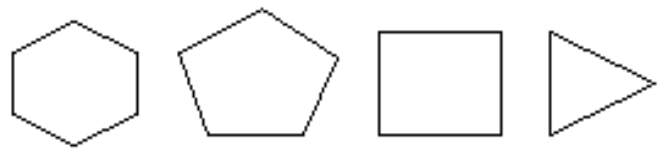
což lze formulovat do věty:

1. věta

Úhel otočení želvy, která se pohybuje po uzavřené křivce (celkové otočení želvy kolem jakékoliv uzavřené dráhy) se rovná celočíselnému násobku 360° .

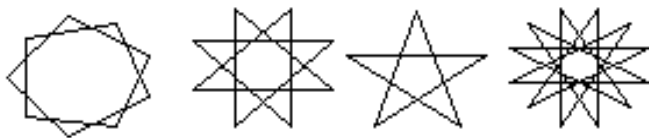
Pravdivost věty je zřejmá. Jestliže je dráha želvy uzavřená a želva započala svou cestu v místě, kde je křivka "hladká" (v bodě, kde má křivka derivaci), musí želva končit svou cestu se stejným nasměrováním a ve stejném bodě, kde cestu začala. Tj. celkové otočení želvy musí být celočíselný násobek 360 stupňů ($U=R \cdot 360$). V závislosti na R (počet otoček želvy) nastávají pro proceduru MNOHO tři možnosti:

1) je-li $R=+1$ nebo $R=-1$ dostáváme konvexní mnohoúhelníky (čtverec, trojúhelník, ...) viz obr. 31;



obr. 31.

MNOHO 30 60, MNOHO 40 72,
MNOHO 50 90, MNOHO 50 120



obr. 32.

MNOHO 40 80, MNOHO 60 135,
MNOHO 60 144, MNOHO 60 150

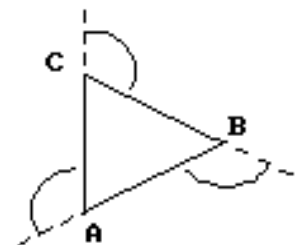
2) je-li $R>1$ nebo $R<-1$ dostáváme hvězdicové mnohoúhelníky (viz obr. 32 pěticípá hvězda, pentagram - tajemný znak pythagorovců, šesticípá hvězda - hvězda krále Davida, osmicípá hvězda, ...),

3) je-li $R=0$ zůstane želva na místě.

Tvrzení 1. věty lze užít k odvození známé věty o velikosti součtu vnitřních úhlů trojúhelníka (obr. 33):

Aplikace věty pro obecný trojúhelník:

$$\begin{aligned} (180^\circ + \alpha) + (180^\circ + \beta) + (180^\circ + \gamma) &= 360^\circ \\ \alpha + \beta + \gamma &= 180^\circ \end{aligned}$$



obr. 33.

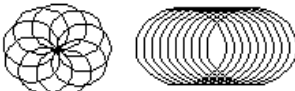
Nyní se však vrátíme k úpravě naší procedury **MNOHO**. Jejím nedostatkem je, že nikdy neskončí svou činnost. Určíme proto podmínku, která by testovala, zda želva dokončila nakreslení prvního útvaru MNOHO:

```
TO MNOHO1 :STR :UHEL
MAKE "OTOC 0
KRESLI :STR :UHEL :OTOC
END
```

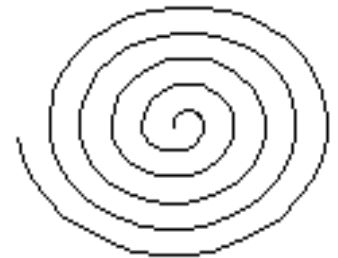
```
TO KRESLI :STR :UHEL :OTOC
FD :STR
RT :UHEL
MAKE "OTOC :OTOC + :UHEL
IF (0 = REMAINDER :OTOC 360) [STOP]
KRESLI :STR :UHEL :OTOC
END
```

Cvičení

- Na základě věty: "Kružnice je obrazec, k němuž se blíží tvar pravidelného mnohoúhelníka, obr. 34, jestliže se délka strany mnohoúhelníka blíží k nule a počet stran tohoto mnohoúhelníka roste nade všechny meze" sestavte proceduru pro nakreslení kružnice. Při sestavování procedury přihlédněte k omezeným rozlišovacím možnostem počítače. Vytvořenou proceduru použijte při kreslení následujících obrázků (obr. 34).



- Sestavte proceduru pro kreslení půloblouku a využijte této procedury k nakreslení spirály viz obr. 35.



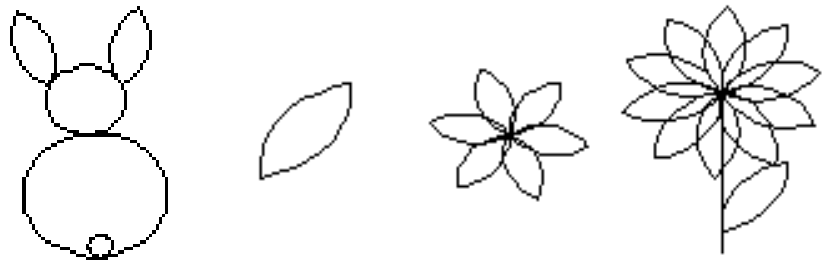
obr. 35.

- Sestavte proceduru pro kreslení čtvrtoblouku a využijte této procedury pro nakreslení paprsku a následovně pro nakreslení slunce viz obr. 36.



obr. 36.

4. Základním stavebním prvkem následujících obrázků (obr. 37) jsou útvary nakreslené procedurou LISTEK. Pro její sestavení využijte procedury pro kreslení čtveroblouku a tvrzení věty 1.



obr. 37.

5. Sestavte procedury pro kreslení následujících kruhů rozdělených na 2,3,4,5,...n dílů (viz obr. 38). Určete vzájemný vztah velikostí těchto částí kruhů. Jaké jsou velikosti obvodů vzniklých částí kruhů?



obr. 38.

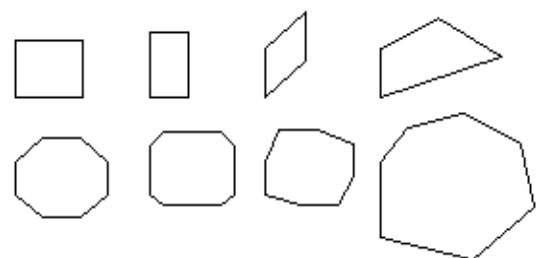
5.1.2 Parquetáž.

Problematika parquetáže řešená pomocí počítače je bohatým zdrojem inspirace pro zkoumání vlastností geometrických útvarů. Jedná se o problém určení rovinných útvarů, které pokryjí rovinu tak, aby žádná část roviny nezůstala nepokryta a aby se žádné dva z těchto rovinných útvarů nepřekrývaly.

Prvním úkolem je sestavit procedury vytvářející základní stavební prvky parquetáže -konvexní mnohoúhelníky.

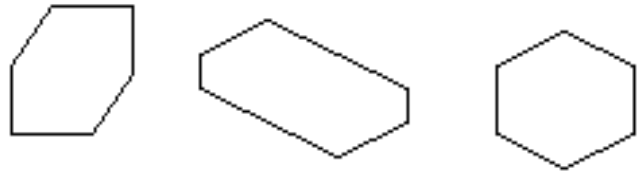
Můžeme je rozdělit do čtyř skupin:

- mnohoúhelníky se shodnými stranami i úhly;
- mnohoúhelníky se shodnými stranami, ale různými vnitřními úhly;
- mnohoúhelníky se shodnými vnitřními úhly ale různými stranami;
- mnohoúhelníky s různými vnitřními úhly i stranami.



obr. 39.

NUHELNIK1 4 35, NUHELNIK1 8 20,
 NUHELNIK2 4 40 20,
 NUHELNIK2 8 30 10,
 NUHELNIK3 4 30, NUHELNIK3 8 20,
 NUHELNIK4 4 30 60,
 NUHELNIK4 8 20 35



obr. 40.

Obrázky vytvořené modifikovanou procedurou SESTIUHELNIK

Příslušné procedury sestavíme např. postupným rozšiřováním procedury NUHELNIK (viz obr. 39):

```
TO NUHELNIK1 :N :STR
MAKE "UHEL 360 / :N
REPEAT :N [FD :STR RT :UHEL]
END
```

```
TO NUHELNIK2 :N :STR1 :STR2
MAKE "UHEL 360 / :N
REPEAT :N [FD :STR1 RT :UHEL
FD :STR2 RT :UHEL]
END
```

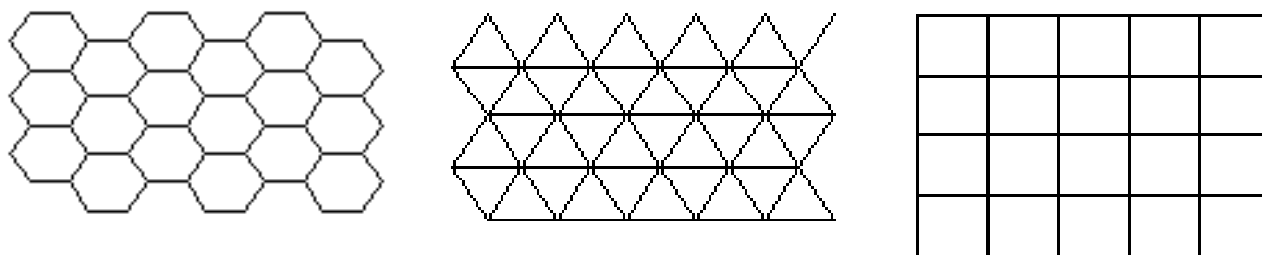
```
TO NUHELNIK3 :N :STR
MAKE "UHEL 360 / (:N * 2)
REPEAT :N / 2 [FD :STR RT :UHEL
FD :STR RT :UHEL * 3]
END
```

```
TO NUHELNIK4 :N :STR :UHEL
MAKE "START POS
REPEAT (:N - 1) [FD :STR RT :UHEL
MAKE "STR :STR+5 MAKE "UHEL :UHEL+5]
SETPOS :START
END
```

Popř. postupným rozšiřováním procedury pro kreslení mnohoúhelníka s daným počtem vrcholů viz obr. 40.

Důležitou složkou parketaže je určení shodných pravidelných n-úhelníků, které mohou pokrýt bez překrývání a bez mezer rovinu. Experimentováním s procedurami pravidelných n-úhelníků zjistíme (viz tabulka), že má-li být okolí určitého bodu a takto pokryto pravidelnými n-úhelníky, můžeme použít čtyři čtverce, tři pravidelné šestiúhelníky nebo šest rovnostranných trojúhelníků.

n-úhelník	počet n-úhelníků	výpočet
trojúhelník	6 trojúhelníků	$6 \times 60 = 360$
čtverec	4 čtverce	$4 \times 90 = 360$
pětiúhelník	-	$3 \times 108 = 324$
šestiúhelník	3 šestiúhelníky	$3 \times 120 = 360$
sedmiúhelník	-	$128,6 \times 3 = 385,7$
osmiúhelník	2 osmiúhelníky + 1 čtverec	$2 \times 135 = 270$ $270 + 90 = 360$
devítiúhelník	-	$2 \times 140 = 280$



obr. 41.
Parquetáž ze shodných
šestiúhelníků, trojúhelníků
a čtverců

Jak totéž odvodíme matematicky? Má-li být okolí určitého bodu a pokryto pravidelnými n -úhelníky, musí být součet velikostí vnitřních úhlů těchto pravidelných n -úhelníků roven 360 stupňů. Protože se jedná o shodné pravidelné n -úhelníky, jsou všechny jeho vnitřní úhly stejně velké a velikost každého z nich je tedy

$$\alpha = \frac{360^\circ}{k}$$

kde k je počet pravidelných n -úhelníků se společným vrcholem A . Velikost vnitřního úhlu pravidelného n -úhelníka lze rovněž odvodit na základě věty o součtu velikostí vnitřních úhlů trojúhelníka, neboť vedeme-li úhlopříčky z jednoho vrcholu mnohoúhelníka ke všem jeho nesousedním vrcholům, rozdělíme daný mnohoúhelník na $n-2$ trojúhelníků s vrcholy totožnými s vrcholy daného n -úhelníka (viz obr. 42):

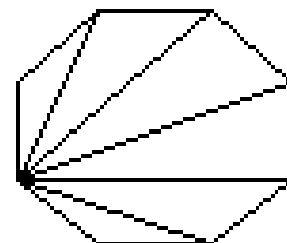
$$\alpha = \frac{(n-2) \cdot 180^\circ}{n}$$

pak platí rovnost:

$$\frac{360^\circ}{k} = \frac{(n-2) \cdot 180^\circ}{n}$$

po úpravách

$$k = 2 \cdot \frac{n}{n-2}$$



obr. 42.

kde k a n jsou přirozená čísla, přičemž n je větší nebo rovno 3, protože neexistuje n -úhelník s menším počtem vrcholů než $n=3$. Z řešení této rovnice v oboru přirozených čísel (dosazujeme postupně čísla $n=3,4,5,6$; pro $n>6$ není řešení v oboru přirozených čísel, protože jmenovatel je větší než číselník) vyplývá, že rovinu můžeme pokrýt jen shodnými rovnostrannými trojúhelníky ($n=3$), čtverci ($n=4$) a pravidelnými šestiúhelníky ($n=6$) viz obr. 41. Sestavení příslušných procedur znamená vytvořit proceduru pro kreslení základního mnohoúhelníka, proceduru sestavující tyto mnohoúhelníky do řady a proceduru spojující jednotlivé řady.

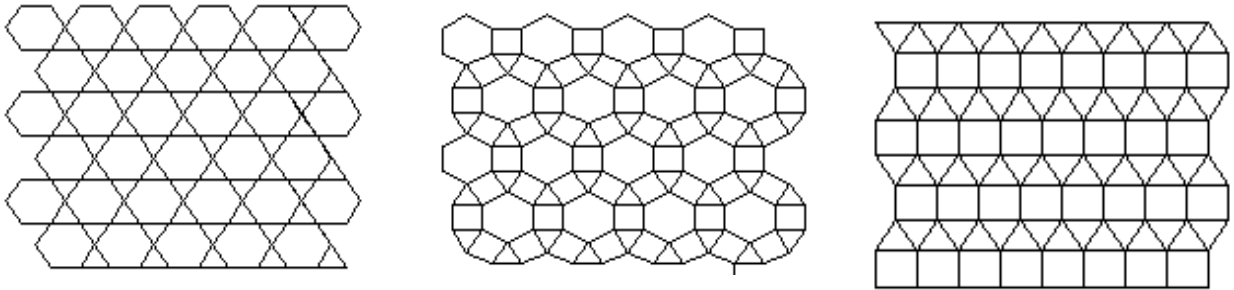
Rozmanitější parquetáže dostaneme, jestliže upustíme od požadavku, aby všechny mnohoúhelníky byly stejného druhu. Označíme-li si x počet rovnostranných trojúhelníků, y počet čtverců, z počet pravidelných

šestiúhelníků pak pro pokrytí okolí bodu a těmito mnohoúhelníky bez mezer a překrývání platí rovnice: $60x + 90y + 120z = 360$

Řešením této lineární rovnice v oboru přirozených čísel dostaneme tyto kořeny

$x = 6$	$y = 0$	$z = 0$
$x = 0$	$y = 4$	$z = 0$
$x = 0$	$y = 0$	$z = 3$
$x = 2$	$y = 0$	$z = 2$
$x = 4$	$y = 0$	$z = 1$
$x = 3$	$y = 2$	$z = 0$
$x = 1$	$y = 2$	$z = 1$

Některé z příslušných mozaik jsou na obr. 43. Sestavení procedur je analogické sestavení procedur pro parketáže ze shodných mnohoúhelníků.



obr. 43. Parketáže tvořené kombinací shodných šestiúhelníků, trojúhelníků a čtverců

Obdobně lze určit všechna pokrytí roviny pravidelnými (ne nutně shodnými) mnohoúhelníky, při kterých, se v každém vrcholu stýkají tři obrazce. Jestliže proměnnými x, y, z označíme počet vrcholů mnohoúhelníků vyplňující okolí bodu A, dostaneme rovnici:

$$\frac{(x+2) \times 180^\circ}{x} + \frac{(y+2) \times 180^\circ}{y} + \frac{(z+2) \times 180^\circ}{z} = 360^\circ$$

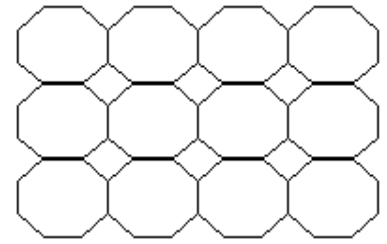
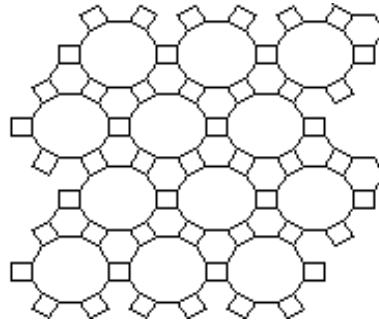
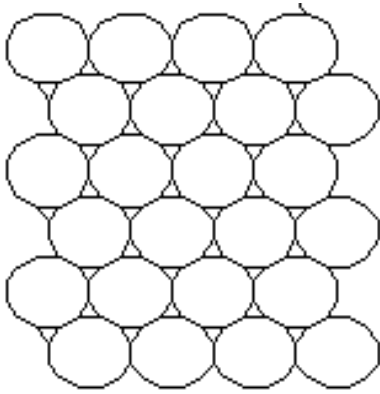
kterou zjednodušíme na tvar:

$$\frac{1}{x} + \frac{1}{y} + \frac{1}{z} = \frac{1}{2}$$

z rovnice lze odvodit tato řešení

$x = 3$	$y = 12$	$z = 12$
$x = 4$	$y = 8$	$z = 8$
$x = 4$	$y = 6$	$z = 12$
$x = 6$	$y = 6$	$z = 6$

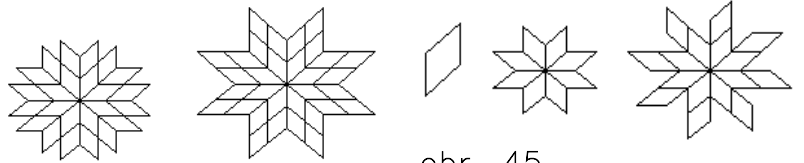
příslušné parketáže viz obr. 44.



obr. 44. Parquetáže, v nichž se v každém vrcholu stýkají tři obrazce

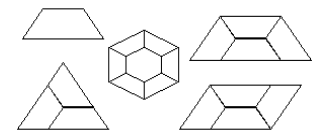
Cvičení

1. Sestavte proceduru KOSOCTVEREC a využijte jí při vytváření procedur následujících mozaik (obr. 45).



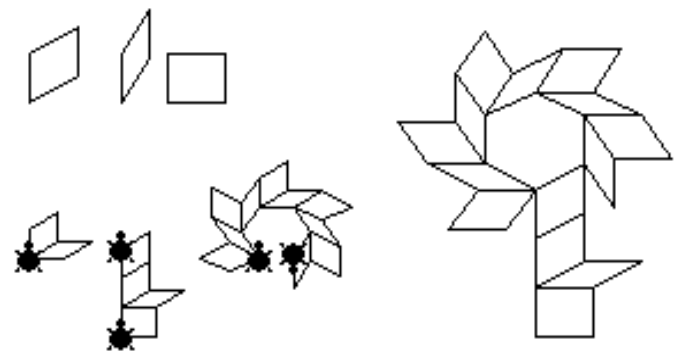
obr. 45.

2. Sestavte proceduru LICHOBEZNIK a využijte jí při stavbě procedur následujících obrázků (viz obr. 46).



obr. 46.

3. Sestavte proceduru KVETINA, která nakreslí ze třech druhů kosočtverců následující obrázek květiny (obr. 47).



obr. 47.

3. Sestavte procedury následujících křivočarých parketáží. Strany těchto obrazců jsou tvořeny čtvrtkružnicemi (obr. 48).



obr. 48.

5.1.3 Symetrie geometrických útvarů

Symetrie je řecké slovo označující souměrnost, pravidelné seskupení předmětů nebo jejich částí podle střední osy. S projevy symetrie se setkáváme v geometrických útvarech, v neživé přírodě (např. krystaly), ve světě rostlin i zvířat (tvar a seskupení listů, rozmístění částí těla). Rozeznáváme souměrnost rotační (rotačně symetrické jsou útvary, které se během otočky kolem svého středu několikrát zobrazí sami na sebe - počet poloh, při nichž se útvar během jedné otočky zobrazí na sebe nazýváme četnost) a souměrnost podle přímky (osově souměrné útvary).

V LOGU vytvoříme souměrné útvary pomocí procedury MNOHO :STR :UHEL (např. procedura MNOHO 30 60 nakreslí rotačně souměrný šestiúhelník s četností šest - otočíme-li ho kolem jeho středu o 360 stupňů zobrazí se šestkrát sám na sebe). Poruší se symetrie kreslených útvarů, jestliže proceduru MNOHO :STR :UHEL obměníme? Abychom soustředili pozornost na vlastní obměny procedury MNOHO :STR :UHEL jsou následující modifikované procedury většinou uvedeny jako "nekonečné" rekursivní programy. Jejich zastavení lze provést přerušením programu pomocí kláves CTRL PAUSE nebo jednoduchou úpravou procedur, která je analogická úpravě procedury MNOHO. Nabízí se tyto základní obměny:

a) modifikace algoritmu (viz obr. 49)

```
TO NOVE.MNOHO :STR :UHEL
FD :STR
RT :UHEL
FD :STR
RT :UHEL*2
NOVE.MNOHO :STR :UHEL
END
```



obr. 49.

```
NOVE.MNOHO 18 135, NOVE.MNOHO 23
144,
NOVE.MNOHO 30 150, NOVE.MNOHO 30
160
```

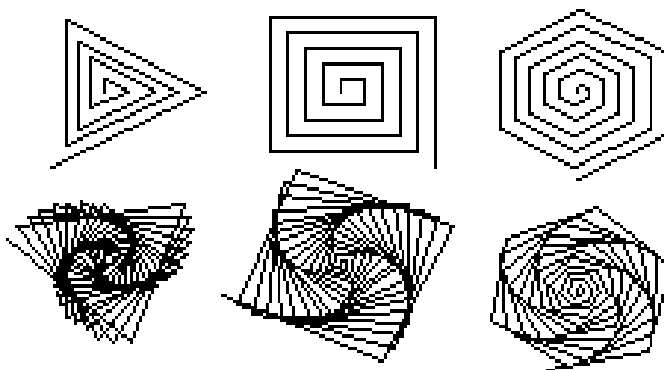
Další obměny získáme např. nahrazením příkazu FD nebo RT procedurou CTVEREC, popř TROJUHELNIK apod.

b) modifikace vstupních parametrů (viz obr. 50 a 51) - tj. rekurentní volání procedury s obměněnými vstupními hodnotami.

- modifikace parametru :STR (změna velikosti posunu vpřed)

```
TO SPIRALA :STR :UHEL
FD :STR
RT :UHEL
SPIRALA :STR + 5 :UHEL
END
```

```
TO SPIRALA2 :STR :UHEL :MEZ
IF :STR > :MEZ [STOP]
FD :STR
RT :UHEL
SPIRALA2 :STR + 5 :UHEL
END
```



obr. 50.
SPIRALA 5 120, SPIRALA 5 117,
SPIRALA 5 90, SPIRALA 5 92,
SPIRALA 1 60, SPIRALA 1 62

Pozn: Zajímavé obrázky dostaneme, budeme-li za vstupní úhel dosazovat hodnoty blízké vstupním úhlům pravidelných n-úhelníků (obr. 50).

- modifikace parametru úhel (změna zakřivení)

```
TO VNITRNI.SPIRALA :STR :UHEL :PRIRUSTEK
FD :STR
RT :UHEL
VNITRNI.SPIRALA :STR :UHEL + :PRIRUSTEK :PRIRUSTEK
END
```

pozn. Při kreslení obrázku procedurou VNITRNI.SPIRALA se křivka střídavě zavinuje (úhel je menší než 180 stupňů) a rozvinuje (úhel je větší než 180 stupňů). Po určitém počtu kroků se vždy dostaneme na původní hodnotu úhlu - tj. $x+k.360$



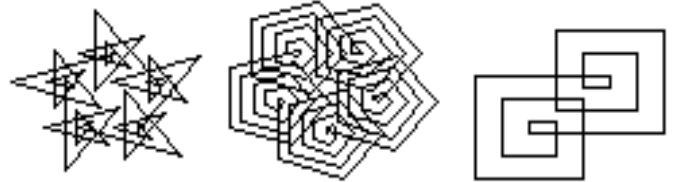
obr. 51.
VNITRNI.SPIRALA 4 4 10,
VNITRNI. SPIRALA 15 2 20,
VNITRNI. SPIRALA 10 2 10,
VNITRNI. SPIRALA 5 10 5

c) modifikace algoritmu i vstupních parametrů (viz obr. 51 a 52)

```
TO ROZLOZ.SPIRAL :STR :UHEL :MAX
MAKE "POC 1
SPIRALA3 :STR :UHEL :MAX
ROZLOZ.SPIRAL :STR :UHEL :MAX
END
```

```
TO SPIRALA3 :STR :UHEL :MAX
IF :POC > :MAX [STOP]
FD :STR * :POC
RT :UHEL
MAKE "POC :POC + 1
SPIRALA3 :STR :UHEL :MAX
END
```

pozn. jedná se o nahrazení dvojice příkazů FD, RT procedurou SPIRALA

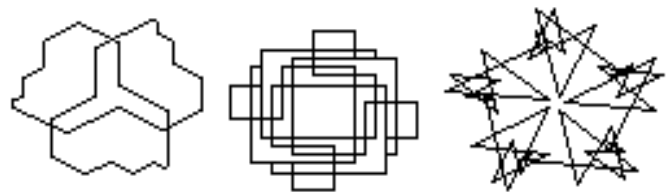


obr. 52.
ROZLOZ.SPIRAL 5 144 8,
ROZLOZ.SPIRAL 2 72 21,
ROZLOZ.SPIRAL 5 90 10

```
TO OBECNE.ROZLOZ.SPIRAL :STR :UHEL :MAX :SEZNAM
MAKE "POC 1
ORIENT.SPIRALA :STR :UHEL :MAX :SEZNAM
OBECNE.ROZLOZ.SPIRAL :STR :UHEL :MAX :SEZNAM
END
```

```
TO ORIENT.SPIRALA :STR :UHEL :MAX :SEZNAM
IF :POC > :MAX [STOP]
FD :STR * :POC
IFELSE MEMBER? :POC :SEZNAM [LT :UHEL][RT :UHEL]
MAKE "POC :POC + 1
ORIENT.SPIRALA :STR :UHEL :MAX :SEZNAM
END
```

pozn. Jedná se o zobecnění procedury ROZLOZ.SPIRAL, které umožňuje želvě otočit se v několika vrcholech doleva místo doprava. Vrcholy, ve kterých se želva otáčí doleva jsou určeny seznamem čísel (tj. jestliže pořadové číslo vrcholu je prvkem seznamu pak se ve vrcholu otočí želva doleva, jinak se otočí doprava)



obr. 53.
OBECNE.ROZLOZ.SPIRAL 2.5 60 10 [2 4 8],
OBECNE.ROZLOZ.SPIRAL 4 90 11 [3 4 5],
OBECNE.ROZLOZ.SPIRAL 5 144 9 [3 4 5]

Při rozboru geometrických útvarů nakreslených modifikovanými procedurami MNOHO

vidíme, že všechny takto vzniklé geometrické útvary jsou podobné mnohoúhelníkům vytvořeným procedurou MNOHO - obsahují základ procedury MNOHO. Pro vysvětlení této skutečnosti použijeme pojem stav. Stav

želvy je dán určením její polohy a jejího nasměrování. Z hlediska stavu želvy jsou potom základní příkazy (FD, BK, LT, RT) operátory změny stavu a programy generující uzavřenou cestu jsou ekvivalentní. Pokud se neustále opakuje určitá posloupnost příkazů, musí vznikat stále stejná změna stavu. Tato změna stavu nastává však vždy ve vztahu k předchozímu stavu želvy a je tvořena úhrnným otočením želvy ve smyčce opakujících se příkazů (což odpovídá vstupnímu úhlu MNOHO) a úhrnným posunutím želvy (což odpovídá parametru posunutí v MNOHO). Tuto skutečnost zformulujeme do následující věty.

2. věta

Každý program přesně opakující nějakou základní smyčku příkazů má strukturu procedury MNOHO se vstupním úhlem T, který je roven celkovému otočení želvy ve smyčce.

Budeme-li podrobně analyzovat tvrzení "mít strukturu procedury MNOHO" dojdeme k názoru, že:

- jestliže úhrnné otočení ve smyčce není rovno násobku 360° , jedná se o opakované dotýkání základního kruhového útvaru;
- jestliže rekurzivní otočení ve smyčce je rovno násobku 360° , jedná se o opakované dotýkání základního přímkovitého geometrického útvaru.

Aplikaci uvedené věty si předvedeme při rozboru několika smyčkových programů, kde ukážeme, že základem geometrického útvaru jakékoliv modifikace procedury MNOHO může být pravidelný pětiúhelník (tj. MNOHO se vstupním úhlem 72°) viz obr. 54.



obr. 54.

MNOHO 30 72,
NOVE.MNOHO 23 144,
VNITRNI.SPIRALA 10 8 40,
ROZLOZ.SPIRAL 5 144 8

procedura NOVE.MNOHO

Pěticipou hvězdu nakreslíme příkazem NOVE.MNOHO 23 144

Celkové otočení ve smyčce $C = 3 \cdot (\text{uhel})$

$$C = 3 \cdot 144 = 432 = 72^\circ \text{ mod } 360^\circ$$

Tj. příslušná pěticipá hvězda má ve skutečnosti strukturu pětiúhelníku a ne strukturu hvězdy procedury MNOHO se vstupním úhlem 144 stupňů. O tom svědčí i skutečnost, že zde nedochází ke křížení čar.

procedura VNITRNI.SPIRALA

Ozdobenou pětícípou hvězdu dostaneme např. příkazem VNITRNI.SPIRALA 10 8 40

Zvětšování hodnoty parametru :UHEL v základní smyčce probíhá následovně:

1. RT 8
2. RT 8 + 1 . 40
3. RT 8 + 2 . 40
- ..
9. RT 8 + 8 . 40
10. RT 8 + 9 . 40 = 360 + 8 příkaz má stejný účinek jako příkaz první
11. ... příkaz bude mít stejný účinek jako příkaz druhý ==> program opakuje prvních devět kroků $C = 9 \cdot 8 + (1 + 2 + \dots + 8) \cdot 40 = 72 + 1440 = 72^\circ \text{ mod } 360^\circ$ což odpovídá pětiúhelníkovému tvaru.

Celkové otočení **ve smyčce**: $C = n \cdot (\text{uhel}) + (1 + 2 + \dots + n-1) \cdot (\text{přírůstek úhlu})$

n ... počet opakování příkazů **FD :STR RT :UHEL** v základní smyčce

procedura ROZLOZ.SPIRAL

Pětiúhelníkový tvar získáme např. příkazem ROZLOZ.SPIRAL 5 144 8

Celkové otočení **ve smyčce**: $C = (\text{úhel}) \cdot (\text{max})$

$$C = 144 \cdot 8 = 1152 = 72^\circ \text{ mod } 360^\circ$$

procedura OBECNE.ROZLOZ.SPIRAL

Pětiúhelníkový tvar dostaneme např. příkazem OBECNE.ROZLOZ.SPIRAL 5 144 9 [3 4 5]

Celkové otočení **ve smyčce**: $C = (\text{max} - L) \cdot (\text{uhel}_{\text{doprava}}) - L \cdot (\text{uhel}_{\text{doleva}}) =$
 $= (\text{max} - 2L) \cdot (\text{uhel})$

(L ... počet prvků v seznamu)

$$C = (9 - 6) \cdot 144 = 432 = 72^\circ \text{ mod } 360^\circ$$

Ukázali jsme si, že symetrie geometrického útvaru jakékoliv procedury ve smyčce příkazů je určena symetrií příslušné procedury **MNOHO :STR :UHEL**. Co ale určuje symetrii mnohoúhelníku této procedury? Je zřejmé, že parametr STR neovlivní tvar mnohoúhelníku, pouze určuje jeho velikost. Podstatou problému je tedy vyřešit otázku vlivu vstupního parametru UHEL na symetrii obrázku procedury MNOHO. Řešení si rozdělíme do dvou částí:

a) Závislost počtu vrcholů (symetrie) mnohoúhelníku procedury MNOHO :STR :UHEL na vstupním parametru UHEL

Z 1. věty vyplývá, že mnohoúhelník procedury MNOHO bude nakreslen právě tehdy, když se želva otočí o celočíselný násobek 360° , tj. můžeme napsat základní rovnici:

$$(1) \quad n \cdot (\text{uhel}) = 360 \cdot R$$

n ... počet stran, popř. vrcholů

uhel ... vstupní úhel

R ... rotační číslo (počet otoček želvy o 360° během nakreslení obrázku)

Rovnice (1) definuje společný násobek hodnoty parametru **UHEL** a 360° . Prvnímu případu, kdy se změna nasměrování želvy rovná celočíselnému násobku 360° odpovídají nejmenší kladná celá čísla vyhovující rovnici (1). Potom číslo **$n \cdot (\text{úhel}) = 360 \cdot R$** představuje nejmenší společný násobek proměnné **UHEL** a 360° .

Odpověď na první otázku je následující:

$$n' = \frac{NSN(UHEL, 360)}{360}, \text{ kde } n \text{ je počet vrcholů odpovídající velikosti vstupního úhlu}$$

$$R' = \frac{NSN(UHEL, 360)}{360}, \text{ R je příslušné rotační číslo}$$

Př. MNOHO 100 144

$$NSN(144, 360) = 720$$

$$n = 720 / 144 = 5 \quad \dots \text{ příslušný MNOHO obrázek má}$$

pětínásobnou symetrii (skládá se z pěti shodných částí spojených dohromady).

Proceduru pro výpočet NSN sestavíme na základě analogie s postupem želvy při kreslení mnohoúhelníku. Tzn. budeme postupně načítávat násobky úhlu a zjišťovat, zda jsme získali násobek 360° .

TO NSN :A :B	TO VYP :A :B
MAKE "NASOBEK :A	IF (0 = REMAINDER :NASOBEK :B) [STOP]
VYP :A :B	MAKE "NASOBEK :NASOBEK + :A
OP :NASOBEK	VYP :A :B
END	END

b) Závislost vstupního parametru UHEL na počtu vrcholů (symetrie) mnohoúhelníku procedury MNOHO

Pro vyvození závislosti vstupního parametru UHEL na počtu vrcholů mnohoúhelníku procedury

MNOHO opět použijeme základní rovnici **$n \cdot (\text{úhel}) = 360 \cdot R$** a odtud:

$$UHEL' = \frac{360(R)}{n}$$

Můžeme však dosadit za **R** jakékoliv celé číslo?

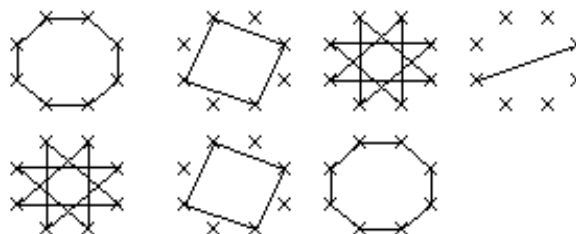
$R = 1$	$U_{HE}L = \frac{360}{N}$	výsledkem bude pravidelný konvexní n-úhelník
$R < 1$ nebo $R > 1$ (např. $R=2, N=10$)	$U_{HE}L = 360 \left(\frac{2}{10} \cdot \frac{360}{5} \cdot 72 \right)$	vypočtený úhel odpovídá pětiúhelníku, nikoliv desetiúhelníku

Aby nebyla snížena násobnost požadované symetrie mnohoúhelníku, musí být zlomek R/n v základním tvaru, tj. čísla R a n musí být nesoudělná. Odpověď na druhou otázku je následující: MNOHO útvar

s n -násobnou symetrií vytvoříme, použijeme-li vstupní úhel $U_{HE}L = \frac{360(R)}{n}$, kde R je libovolné kladné číslo nesoudělné s n .

Podívejme se jak lze interpretovat znalosti činnosti procedury MNOHO. Představme si n vrcholů procedury MNOHO ležících na kružnici a označených od 0 do $n-1$. Abychom vytvořili n -stranné mnohoúhelníky, můžeme spojovat vrcholy aplikací následujících pravidel:

1. Spoj každý vrchol s nejbližším vrcholem.
2. Spoj vrcholy tak, že vždy mezi nimi vynecháš jeden vrchol.
3. Spoj vrcholy tak, že vždy vynecháš dva vrcholy,



obr. 55. Vliv soudělnosti R a n na obrázky kreslené procedurou MNOHO pro $n=8$

atd. Na obr. 55 jsou uvedeny varianty pro $n=8$. Existuje zde $n-1$ možných spojení vrcholů, což odpovídá rotačnímu číslu $R=1,2,\dots,n-1$. Pro libovolný výběr rotačního čísla bude 0 -tý vrchol spojen s R -tým vrcholem, který bude spojen s vrcholem označeným $2R$ atd. Tj. posloupnost indexů spojených vrcholů tvoří násobky $0, R, 2R, \dots, (n-1)R$ modulo n .

- a) Jestliže R a n jsou nesoudělné, potom mnohoúhelník procedury MNOHO bude mít n vrcholů, tj. v našem obrázku bude mnohoúhelník obsahovat všechny vrcholy na kružnici.
- b) Jestliže R a n nejsou soudělné, pak alespoň jeden z vrcholů nebude dosažen (např. pro $n=8$ a $R=2$ projde proces procedury MNOHO sudými vrcholy, pro $n=8$ a $R=4$ projde proces procedury MNOHO vrcholy, jejichž index je násobkem čtyř).

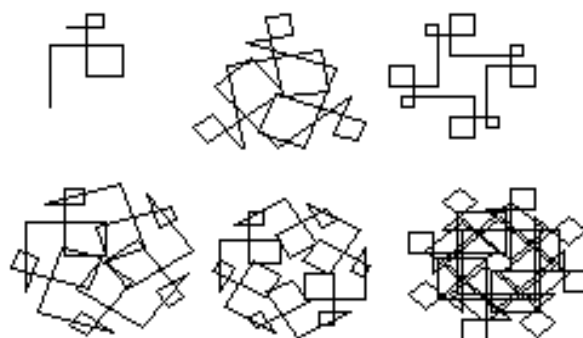
Indexy vrcholů, jimiž proces procedury MNOHO prochází $0, R, 2R, \dots, (n-1)R$ modulo n jsou tedy násobky nějakého celého čísla d . Toto číslo je největším společným dělitelem n a R , $d = \text{NSD}(n, R)$. Největší společný dělitel je ve vztahu k symetrii procedury MNOHO důležitou veličinou:

- jestliže $\text{NSD}(n, R) = 1$, potom R a n jsou nesoudělná a mnohoúhelník procedury MNOHO má n -násobnou symetrii;
- jestliže $\text{NSD}(n, R) < 1$ nebo $\text{NSD}(n, -R) > 1$, potom mnohoúhelník procedury MNOHO má $(n/\text{NSD}(n, R))$ -násobnou symetrii, indexy dosažených vrcholů jsou násobky $\text{NSD}(n, R)$.

Cvičení

1. Vypočítejte celkové otočení T v proceduře PRVEK. Na základě výpočtu určete vstupní úhel do modifikované procedury MNOHO (obměna spočívá v nahrazení příkazu FD procedurou PRVEK) tak, abyste získali útvary s rotační symetrií četnosti 3, 4, 5, 6, 8 apod (viz obr. 56).

```
TO PRVEK :X
FD :X RT 90 FD :X RT 90
FD :X / 2 RT 90 FD :X / 2 RT 90
FD :X RT 90 FD :X / 4 RT 90
FD :X / 4 RT 90 FD :X / 2
END
```

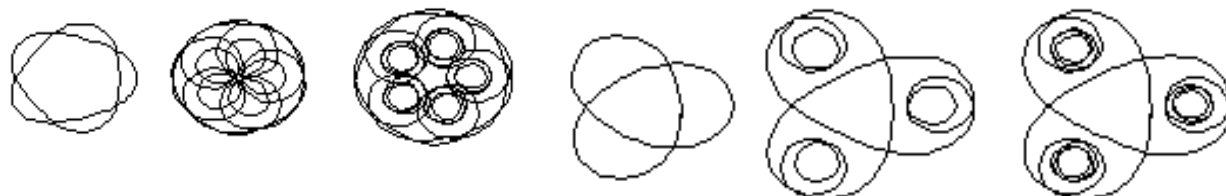


obr. 56.

2. Pomocí proměnných D (dolní hodnota úhlu) a H (horní hodnota úhlu) vyjádřete celkové otočení v následujícím smyčkové proceduře. Nalezený vztah užijte pro nakreslení alespoň tří různých obrázků (libovolné velikosti) se symetrií četnosti 3, 5 (viz obr. 57).

```
TO MNOHO.SMYCKA :STR :DOLNI :HORNI
STOP.SPIRALA :STR :DOLNI :HORNI 1
STOP.SPIRALA :STR :HORNI :DOLNI - 1
MNOHO.SMYCKA :STR :DOLNI :HORNI
END
```

```
TO STOP.SPIRALA :STR :STARTUHEL :CILUHEL :ZMENA
IF :STARTUHEL = :CILUHEL [STOP]
FD :STR LT :STARTUHEL
STOP.SPIRALA :STR :STARTUHEL + :ZMENA :CILUHEL :ZMENA
END
```



obr. 57.

5.1.4 Geometrické obrazce s konstantním obsahem

a) Obdélníky s konstantní plochou

Pokud si zvolíme konstantní obsah obdélníka (např. 800 druhých mocnin želvích kroků) a dosadíme-li tuto veličinu do vztahu pro výpočet obsahu obdélníka ($P=a \cdot b$), dostaneme následující závislost velikosti strany **b** na straně **a**:

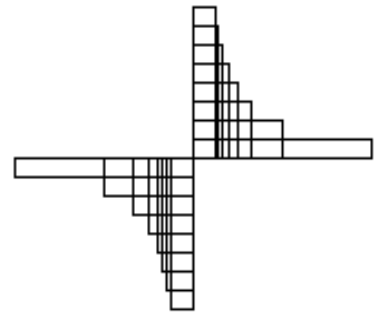
$$b = \frac{800}{a}$$

Procedura pro nakreslení příslušného obdélníku má potom tvar:

```
TO OBDELNIK :A
REPEAT 2 [FD :A RT 90 FD 800 / :a RT 90]
END
```

Budeme-li nyní postupně zvětšovat velikost parametru **A**, dostaneme obrázek připomínající část hyperboly, a proto i proceduru vytvářející jakési "lešení pro hyperbolu" nazveme HYPERBOLA (obr. 58):

```
TO HYPERBOLA
HT MAKE "A 10
REPEAT 8 [OBDELNIK :A MAKE "A :A + 10]
RT 180 MAKE "A 10
REPEAT 8 [OBDELNIK :A MAKE "A :A + 10]
END
```



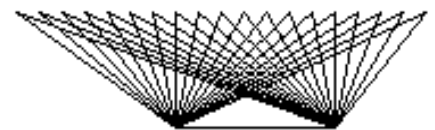
obr. 58. HYPERBOLA

Takto vytvořeného obrázku lze použít jako motivačního prostředku pro zopakování nepřímé úměrnosti, popř. vlastností hyperboly.

b) Planimetrie - "zrcadlení" trojúhelníků

Nyní sestavíme proceduru pro kreslení množiny trojúhelníků se společnou základnou (obr. 59), která je prvkem osy posunutě souměrnosti. Vrcholy trojúhelníků budou ležet na přímce rovnoběžné se základnou a poloha každých spolu sousedních vrcholů se bude lišit posunutím o vektor posunutě souměrnosti:

```
TO ZRCADLENÍ :ZAKLADNA :X :Y :VEKTOR
SETX :ZAKLADNA
SETPOS LIST :X :Y
HOME
IF :X > 100 [STOP]
ZRCADLENÍ :ZAKLADNA :X+ :VEKTOR :Y :VEKTOR
END
```



obr. 59. ZRCADLENÍ 60
-70 60 20

5.2 Modelování pohybu živočichů

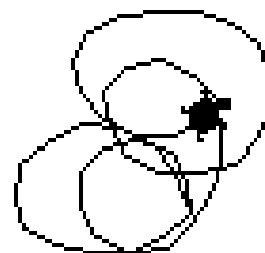
V této kapitole se pokusíme modelovat chování zvířat. Želví geometrie je pro toto modelování výhodná, neboť primitiva želvy (FORWARD, RIGHT, ...) lze chápat jako operátory změny vnitřního stavu želvy.

Nejjednodušším druhem pohybu, který můžeme modelovat pomocí LOGA je **náhodný pohyb** (opakovaný přesun dopředu a otočení se o náhodnou veličinu). Abychom mohli vytvořit příslušnou proceduru, upravíme si generátor náhodných čísel tak, aby generoval náhodné číslo z uzavřeného intervalu $\langle D, H \rangle$.

```
TO NAHODNE :D :H
OP :D + RANDOM (:H - :D)
END
```

Procedura pro náhodný pohyb bude mít následující tvar:

```
TO NAHOD.POHYB :V1 :V2 :U1 :U2
LT NAHODNE :U1 :U2
FD NAHODNE :V1 :V2
NAHOD.POHYB :V1 :V2 :U1 :U2
END
```



obr. 60.
NAHOD.POHYB 10
30 10 50

Tato jednoduchá procedura může sloužit ke zkoumání vlivu omezení parametrů příkazů LEFT a FORWARD na cestu želvy. Dokud např. nezvolíme parametr U1 záporný, bude se želva stále otáčet doleva a její stopa bude připomínat kroužení (obr. 60). Tato procedura náhodného pohybu však často dostane želvu mimo hranice obrazovky.

Sestavení procedury umožňující pohyb želvy pouze v mezích obrazovky vede k modelování chování hmyzu v krabici. Příslušná procedura ZKUS.VPRED má stejnou funkci jako primitivum FORWARD, avšak nedovolí želvě pohyb, jestliže by se dostala mimo určitou vymezenou oblast. Tj. V proceduře budeme provádět se želvou "neviditelný" přesun na novou pozici, procedurou MIMO.MEZ? přezkoušíme, zda je nová pozice mimo vymezenou oblast a pokud ne, provedeme nyní již viditelný přesun želvy.

```
TO ZKUS.VPRED :VZDALENOST
MAKE "STARA.POZICE POS
PU HT
FD :VZDALENOST
MAKE "TEST MIMO.MEZ?
SETPOS :STARA.POZICE
PD ST
```

```

IF :TEST = "FALSE [FD :VZDALENOST]
END

TO MIMO.MEZ?
MAKE "A ABS XCOR MAKE "B ABS YCOR
IFELSE OR (:A > 50) (:B > 50) [OP "TRUE] [OP "FALSE]
END

TO ABS :X
IFELSE OR (:X > 0) (:X = 0) [OP :X] [OP :X * (-1)]
END

TO VZDALENOST :BOD
MAKE "A (XCOR - FIRST :BOD) (MAKE "A :A * :A)
MAKE "B (YCOR - LAST :BOD) (MAKE "B :B * :B)
OP SQRT (:A + :B)
END
    
```

Procedura pro modelování chování želvy v krabici, může mít následující tvar (kdykoliv želva vběhne do hrany krabice, otočíme želvu o 180°) (obr. 61):

```

TO NAHOD.POHYB2 :V1 :V2 :U1 :U2
LT NAHODNE :U1 :U2
ZKUS.VPRED NAHODNE :V1 :V2
IF :TEST [LT 180]
NAHOD.POHYB2 :V1 :V2 :U1 :U2
END
    
```



obr. 61.
 NAHOD.POHYB2 5
 50 -30 30,
 krabice je
 nakreslena
 příkazem PU
 SETPOS [-50 50]
 PD REPEAT 4 [FD
 100 RT 90]

Jinou možností simulace chování želvy u stěny krabice je otáčet se želvou tak dlouho, dokud nebude moci jít opět vpřed, tj. v proceduře NAHOD.POHYB2 zaměníme řádek IF :TEST [LT 180] řádkem IF :TEST [VYTOC]. Procedura VYTOC má následující tvar:

```

TO VYTOC
RT 1
ZKUS.VPRED 1
IF MIMO.MEZ? [VYTOC]
END
    
```



obr. 62.
 NAHOD.POHYB3 5
 50 -30 30

Jak je vidět na obr. 62, zapříčiňuje tato procedura pohyb želvy podél stěn krabice. Podobný

pohyb je možno pozorovat u skutečného hmyzu, kde je toto chování vysvětlováno jako snaha hmyzu dostat

se ven z krabíčky. Naše procedura však vnáší pochybnosti do takového polidšťování chování hmyzu. Není chování hmyzu spíše způsobeno jednoduchou kombinací náhodného pohybu vpřed plus vyhnutí se stěně?

Naši simulaci náhodného pohybu můžeme dále propracovat, jestliže připustíme, že chování želvy bude ovlivněno nějakým stimulem - např. umístíme do krabice potravu. Nyní sestavíme algoritmus umožňující najít potravu "po čichu". Informaci odpovídající schopnosti čichu můžeme želvě poskytnout několika způsoby. Zvolíme např., aby v každém přemístění byla želva schopna určit, zda je vůně potravy silnější nebo slabší.

```
TO VUNE :BOD
MAKE "NOVA.VZDALENOST VZDALENOST :BOD
IFELSE :NOVA.VZDALENOST > :STARA.VZDALENOST [MAKE "VYSL "SLABSI]
[MAKE "VYSL "SILNEJSI]
MAKE "STARA.VZDALENOST :NOVA.VZDALENOST
OP :VYSL
END
```

Procedura využívající informaci o intenzitě vůně pro nalezení potravy má potom následující tvar (obr. 63):

```
TO NAJDI.PO.CICHU :BOD :OTOC
MAKE "STARA.VZDALENOST VZDALENOST :BOD
HLEDEJ :BOD :OTOC
END
```

```
TO HLEDEJ :BOD :OTOC
FD 1
IF (VUNE :BOD) = "SLABSI [LT :OTOC]
HLEDEJ :BOD :OTOC
END
```

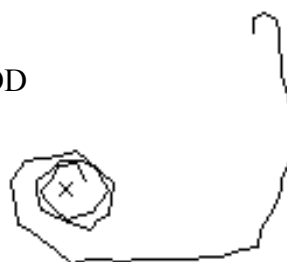


obr. 63.
NAJDI.PO.CICHU
[-50 -50] 40,
potrava je
umístěna v bodě
[-50 -50],
výchozí poloha
želvy je v bodě
[50 0]

Reálnější by však bylo, aby procedura zahrnovala rovněž náhodný pohyb (obr. 64):

```
TO NAJDI.PO.CICHU2 :BOD :V1 :V2
:CICH.OTOC :NAHOD.OTOC
MAKE "STARA.VZDALENOST VZDALENOST :BOD
HLEDEJ2 :BOD :V1 :V2 :CICH.OTOC
:NAHOD.OTOC
END
```

```
TO HLEDEJ2 :BOD :V1 :V2 :CICH.OTOC
:NAHOD.OTOC
FD NAHODNE :V1 :V2
LT NAHODNE :NAHOD.OTOC * (-1)
:NAHOD.OTOC
IF (VUNE :BOD) = "SLABSI [RT :CICH.OTOC]
HLEDEJ2 :BOD :V1 :V2 :CICH.OTOC
:NAHOD.OTOC
END
```



obr. 64.
NAJDI.PO.CICHU2
[-30 -30] 5 10
55 15, potrava je
umístěna v bodě
[-30 -30],
výchozí poloha
želvy je v bodě
[50 50]

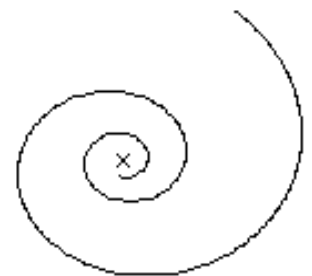
Podobným způsobem, jako jsme nasimulovali čich, bychom mohli nasimulovat zrak. Pro zjednodušení problému bude želva reagovat pouze na intenzitu dopadajícího světla. Rozdíl od simulace čichu spočívá v směrovém zaměření zraku. První model simulace zraku založíme na předpokladu, že jestliže živočich je schopen registrovat světlo, je rovněž schopen obrátit se k němu:

```
TO NASMERUJ :X :Y          TO AZIMUT :X :Y
LT AZIMUT :X :Y           OP (HEADING - TOWARDS LIST :X :Y)
END                         END
```

Nyní je pro želvu jednoduché dostat se ke zdroji světla, tj. obrátit se k němu a zjistit vzdálenost.

Proceduru NASMERUJ lze použít při simulaci létání můry na světlo. Někteří lidé vysvětlují toto chování tím, že hmyz létající v noci řídí své létání dodržováním stále stejného směru k měsíci. Když však dojde k záměně měsíce s nějakým blízkým zdrojem světla, nastane spirálovité přibližování se ke zdroji světla. Toto simuluje procedura NASMERUJ, která zajišťuje pohyb želvy ke zdroji při zachování původního nasměrování (obr. 65):

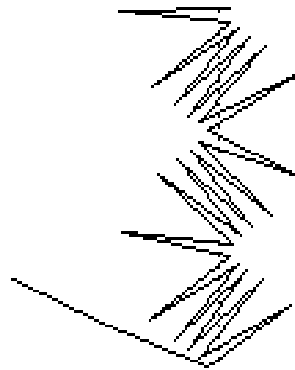
```
TO DRZ.SMER :X :Y :UHEL
NASMERUJ :X :Y
LT :UHEL
FD 1
DRZ.SMER :X :Y :UHEL
END
```



obr. 65.
DRZ.SMER 0 0
80, zdroj světla
je umístěn v bodě
[0 0], výchozí
poloha želvy je
v bodě [50 80]

Během biologického vývoje se vyvinulo u zvířat "řídící ústrojí" umožňující chovat se v dané situaci co možná nejlépe (tzv. adaptabilní, přizpůsobivé řízení). Jednoduchou simulací tohoto řízení si předvedeme na řešení programu "psí křivky". Problém spočívá v realizaci velkého počtu rozhodování v průběhu aproximace následující dráhy psa (obr. 66): pes má mnoho energie a proto odbíhá od svého pána, na pánovo zavolání se pes vrací zpět k pánovi, pán však jde stále po přímce dopředu a pes je tedy nucen při návratu řídit směr svého běhu podle směru polohy pána. Příslušná procedura PES bude obsahovat rekurzivní volání následujících prvků:

- "neviditelný" přesun želvy na pozici pána a označení této pozice (PD FD 0);



obr. 66. PES [0
-100] [-140 0]
5 40

- "neviditelné" přemístění želvy na pozici psa a viditelný přesun želvy směrem k pozici pána (velikost přesunu je dána parametrem RYCHLOST.PSA);
- uložení souřadnic nové polohy psa a vypočtení souřadnice nové polohy pána (zvětšením y-ové souřadnice o rychlost pána).

```

TO PES :POLOHA.PANA :POLOHA.PSA :RYCHLOST.PANA :RYCHLOST.PSA
MAKE "POM LAST :POLOHA.PANA
IF :POM > 70 [STOP]
PU SETPOS :POLOHA.PANA
PD FD 0
PU SETPOS :POLOHA.PSA
SETH TOWARDS :POLOHA.PANA
PD FD :RYCHLOST.PSA
MAKE "POLOHA.PSA POS
MAKE "X FIRST POLOHA.PANA MAKE "Y (LAST :POLOHA.PANA) + :RYCHLOST.PANA
MAKE "POLOHA.PANA LIST :X :Y
PES :POLOHA.PANA :POLOHA.PSA :RYCHLOST.PANA :RYCHLOST.PSA
END
    
```

Dostali jsme se tak k modelování vzájemného působení živočichů. Toto vzájemné působení budeme modelovat ještě na jednom příkladu - jednoduché simulaci chování dravce a kořisti. Nasimulujeme pohyb dravce, který se snaží chytit kořist. Pro zjednodušení problému předpokládejme, že se kořist bude pohybovat stále po kružnici, aniž by věděla o záměru dravce a dravec bude pronásledovat kořist užitím procedury NAJDI.PO.CICHU (obr. 67). Nejprve sestavíme samostatné procedury pro pohyb dravce a kořisti.

```

TO KORIST.KROK :PARAMETRY
FD FIRST :PARAMETRY
RT LAST :PARAMETRY
END
    
```

```

TO DRAVEC.KROK :PARAMETRY
FD FIRST :PARAMETRY
IF (VUNE :KOR.POL) = "SLABSI [LT LAST :PARAMETRY]
END
    
```

Dále sestavíme proceduru LOV, která zajistí střídavé vykonávání pohybu dravce a kořisti. Aby procedury pohybu obou živočichů byly provedeny na správném místě, zavedeme proměnné, v níž budeme uchovávat polohu a nasměrování živočichů.

```

TO LOV :DRAV.POH :DRAV.POL :DRAV.SMER :KOR.POH :KOR.POL :KOR.SMER
PU SETPOS :KOR.POL SETH :KOR.SMER PD
KORIST.KROK :KOR.POH
MAKE "KOR.POL POS
MAKE "KOR.SMER HEADING
PU SETPOS :DRAV.POL SETH :DRAV.SMER PD
DRAVEC.KROK :DRAV.POH
MAKE "DRAV.POL POS
MAKE "DRAV.SMER HEADING
IF :DRAV.POL = :KOR.POL [STOP]
LOV :DRAV.POH :DRAV.POL :DRAV.SMER :KOR.POH :KOR.POL :KOR.SMER
END
    
```

Uvedenou proceduru bychom mohli dále upravovat (např. tvar želvy se bude opakovaně měnit z dravce na kořist). Jak je patrné z obr. 67 procedura NAJDI.PO.CICHU pracuje při pohybu potravy s malou efektivitou. Proto např. vybavíme dravce zrakem a kořisti umožníme unikat před dravcem pomocí procedury NAJDI-
.PO.CICHU.



obr. 67. LOV [10 35] [-90 75] 0 [5
15] [50 30]

5.3 Fraktály

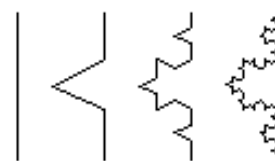
V této kapitole budeme zkoumat složité tvary a struktury přírody. Budou zde ukázány některé zajímavé křivky, dva modely růstu a letmo zmíněna problematika fraktální geometrie.

Přírodě se sice nedostává hladkosti a pravidelnosti, ale nepanuje v ní úplný chaos. Právě harmonie mezi pravidelností a nahodilostí činí přírodu krásnou. Benoit Mandelbrot se úspěšně pokusil tuto harmonii matematicky modelovat. Pravidelné křivky a tělesa, jimiž se klasická geometrie zabývá, bývají symetrické podle osy, roviny nebo středu. V matematice hovoříme o invarianci vůči určitým transformacím, v tomto případě symetriím. Fraktální geometrie se zabývá invariancemi vůči transformacím spočívajícím ve změně měřítka. Studuje útvary, u nichž se stejný vzor opakuje ve stále menším měřítku.

Velmi jednoduchým modelem fraktálu je **křivka Kochové** (obr. 68). Výchozím prvkem je úsečka nazývaná iniciátor. Prostřední třetinu úsečky vyjmeme a nahradíme dvěma úsečkami délky $1/3$, setkávajícími se ve vrcholu rovnostranného trojúhelníka. Vzniklý útvar se nazývá generátor. Nyní každý ze čtyř dílů generátoru nahradíme generátorem zmenšeným v poměru $1:3$. Příslušná procedura **STRANA** kreslící křivku Kochové bude rekurzivní, tj. napíšeme proceduru pro úsečku, která je zhotovena ze čtyř pod-úseček a každá pod-úsečka je zhotovena ze čtyř pod-pod-úseček ... atd. Abychom zajistili styčné body mezi částí a podčástí,

proceduru sestavíme tak, aby přesunula želvu dopředu o určitou vzdálenost bez změny nasměrování. Rovněž zařadíme podmínku pro zastavení generování dalších obrázků.

```
TO STRANA :VEL :ST
IF :ST= 0 [FD :VEL STOP]
STRANA :VEL/ 3 :ST- 1
LT 60
STRANA :VEL/ 3 :ST- 1
RT 120
STRANA :VEL/ 3 :ST- 1
LT 60
STRANA :VEL/ 3 :ST- 1
END
```



obr. 68.
STRANA 80 0,
STRANA 80 1,
STRANA 80 2,
STRANA 80 3

Použijeme-li jako iniciátor pro křivku Kochové rovnostranný trojúhelník, bude výsledkem uzavřená křivka Kochové. Příslušnou proceduru nazveme VLOCKA pro podobnost se sněhovou vločkou (obr. 69).

```
TO VLOCKA :VEL :ST
REPEAT 3 [STRANA :VEL :ST RT 120]
END
```



obr. 69.
Opakované volání procedury VLOCKA s různými parametry a různou výchozí polohou želvy

U této křivky si ukážeme zajímavou vlastnost uzavřené fraktální křivky, tj. ačkoliv je délka uzavřené fraktální křivky nekonečná, je plocha jí uzavřená konečná.

a) Výpočet délky uzavřené křivky Kochové

Tvoříme-li křivku K_n z křivky K_{n-1} vzniknou z každé úsečky na původní křivce K_{n-1} čtyři úsečky délky třikrát menší, tzn.

Řád křivky	0. řád	1. řád	2. řád	...	n. řád
délka křivky	$3 \cdot a$	$3 \cdot a \cdot \left(\frac{4}{3}\right)$	$3 \cdot a \cdot \left(\frac{4}{3}\right)^2$...	$3 \cdot a \cdot \left(\frac{4}{3}\right)^n$

Délku křivky Kochové označíme K ,
$$K = \lim_{n \rightarrow \infty} 3 \cdot a \cdot \left(\frac{4}{3}\right)^n = 4$$

b) Výpočet obrazce, který ohraničuje křivka Kochové

křivka 0.řádu	$P_0 = a^2 \frac{\sqrt{3}}{4}$
---------------	--------------------------------

křivka 1.řádu	$P_1' P_0 \% P_0 \frac{4^0}{9^1} 3$
křivka 2.řádu	$P_2' P_0 \% P_0 \frac{4^0}{9^1} 3 \% P_0 \frac{4^1}{9^2} 3$
...	...
křivka n.řádu	$P_n' P_0 \% P_0 \frac{4^0}{9^1} 3 \% \dots \% P_0 \frac{4^{n&1}}{9^n} 3$

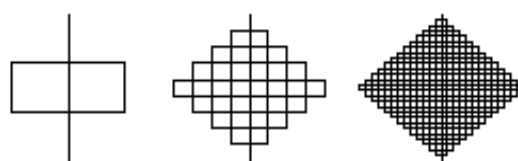
Obsah plochy ohraničené křivkou Kochové je tedy konečná veličina:

$$P' P_0 \left(1 \% 3 \int_{n=1}^4 \frac{4^{n&1}}{9^n} \right)$$

Další skupinu fraktálních křivek tvoří ty, které vyplňují rovinný útvar. Patří sem např. křivka vytvářená podprogramem VYPLN (obr. 70). Myšlenka konstrukce této křivky je následující: předpokládejme, že jsme napsali program, který kreslí něco uvnitř čtverce; nebudeme se zajímat o to, co program dělá, ale soustředíme se na počáteční a koncový stav želvy v programu; budeme požadovat, aby želva vstupovala do čtverce a vystupovala z něj v rozích ležících na úhlopříčce a rovněž aby její počáteční a koncové nasměrování bylo totožné se směrem příslušné úhlopříčky; nyní uvažujme náš čtverec rozdělený na devět shodných čtverců, a vytvořme program procházející stejným způsobem přes nové čtverce; takto můžeme pokračovat dále, na konci dostaneme čtverec, sestávající se z devíti čtverců, každý z nich sestávající se z devíti menších čtverců, atd.

```

TO VYPLN :VEL :ST
IF :ST = 0 [FD :VEL STOP]
VYPLN :VEL / 3 :ST - 1
LT 90
VYPLN :VEL / 3 :ST - 1
REPEAT 3 [RT 90 VYPLN :VEL / 3 :ST - 1]
REPEAT 3 [LT 90 VYPLN :VEL / 3 :ST - 1]
RT 90
VYPLN :VEL / 3 :ST - 1
END
    
```



obr. 70.
VYPLN 80 1, VYPLN 80 2,
VYPLN 80 3

Pokud do parametru ST dosazujeme stále vyšší hodnoty, potom křivka VYPLN prochází podél stále menších čtverců a pro velmi vysokou hodnotu ST si můžeme představit křivku VYPLN procházející každým bodem čtverce.

Snad neznámější křivkou vyplňující prostor je **Hilbertova křivka**. Na obr. 71 je zobrazen postup vytváření této křivky. Jestliže stejným způsobem pokračujeme donekonečna, dostaneme opět křivku procházející

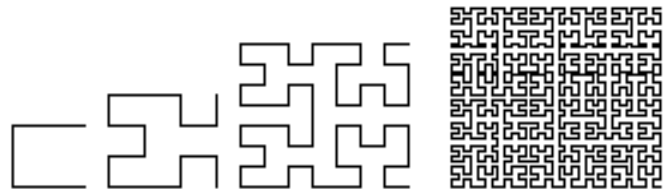
všemi body daného čtverce. Strategii konstrukce příslušné procedury odvodíme z obrázku Hilbertovy křivky 3. stupně: každý n-tý stupeň Hilbertovy křivky je sestaven ze čtyř Hilbertových křivek (n-1) stupně; tyto jsou dvojího druhu, jedna prochází čtvercem příčně doleva a druhá příčně doprava; celý program se skládá ze dvou procedur:

```

TO LHILBERT :VEL :ST
IF :ST= 0 [STOP]
LT 90
PHILBERT :VEL :ST - 1
FD :VEL
RT 90
LHILBERT :VEL :ST - 1
FD :VEL
LHILBERT :VEL :ST - 1
RT 90
FD :VEL
PHILBERT :VEL :ST - 1
LT 90
END

TO PHILBERT :VEL :ST
IF :ST= 0 [STOP]
RT 90
LHILBERT :VEL :ST- 1
FD :VEL
LT 90
PHILBERT :VEL :ST- 1
FD :90
PHILBERT :VEL :ST- 1
LT 90
FD :VEL
LHILBERT :VEL :ST- 1
RT 90
END

```



obr. 71. LHILBERT 30 1, LHILBERT 15 2,
LHILBERT 10 3, LHILBERT 2.8 5

Podobně jako proceduru pro kreslení Hilbertovy křivky lze sestavit proceduru pro kreslení křivky, která dostala pro charakteristický tvar označení drak (obr. 72). Její vznik si můžeme demonstrovat pomocí proužku papíru: dokud to bude možné budeme opakovaně překládat proužek papíru na polovinu a potom jej opatrně rozložíme tak, aby v místech přehybu svíral vždy úhel 90°. Výsledkem bude znázornění zmíněné křivky. Při sestavování algoritmu křivky budeme vycházet z demonstrace křivky pomocí proužku papíru. Bude se jednat o vzájemné rekurzivní volání dvou procedur, jedna bude otáčet křivku doprava, druhá doleva.

```

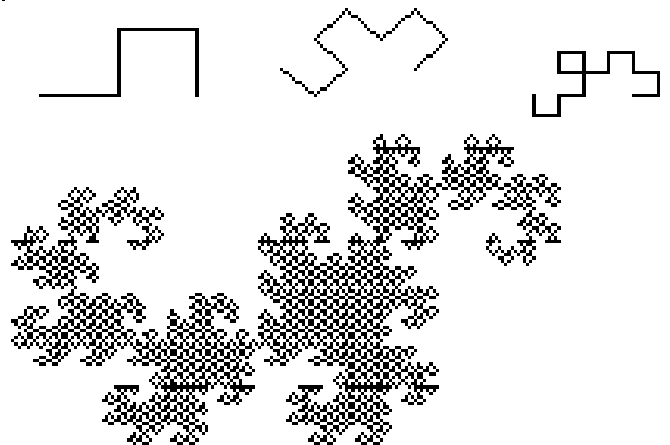
TO LDRAK :VEL :ST
IFELSE :ST = 0 [FD :VEL]
[LDRAK :VEL :ST - 1]
LT 90
IFELSE :ST = 0 [FD :VEL]
[PDRAK :VEL :ST - 1]
END

```

```

TO PDRAK :VEL :ST

```



obr. 72. LDRAK 25 1, LDRAK 15 2,
LDRAK 8 3, LDRAK 2.6 10

```

IFELSE :ST = 0 [FD :VEL]
[LDRAK :VEL :ST - 1]
RT 90
IFELSE :ST = 0 [FD :VEL] [PDRAK :VEL :ST - 1]
END

```

Fraktální plošně větvené struktury mohou vzniknout též z dvourozměrných útvarů. Příkladem je **Serpinského krajka** (obr. 73). Obrázek se skládá z rovnostranného trojúhelníka plus menších kopií tohoto trojúhelníka u každého jeho vrcholu, plus ještě menších kopií trojúhelníka u vrcholů těchto kopií, atd. Při konstrukci obrázku se želva zastaví v každém vrcholu trojúhelníka a vytvoří trojúhelník menší velikosti a pokračuje v kreslení stále menších trojúhelníků až do určité meze.

```

TO SERP.KRAJKA :VEL
IF :VEL < 5 [STOP]
REPEAT 3 [SERP.KRAJKA :VEL / 2 FD :VEL RT 120]
END

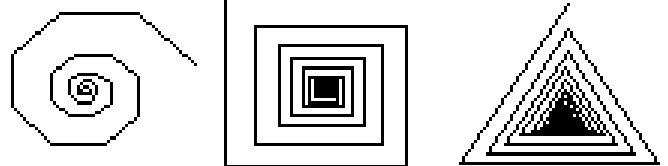
```

Nyní budeme tyto poznatky používat pro modelování růstu biologických tvarů. Nejprve budeme modelovat spirálovitý růst, který je patrný zejména na tvarech ulit a rohů. v kapitole 5.1.3. jsme upravili program MNOHO na program pro kreslení spirál, tj. v každém kroku jsme zvětšovali stranu o konstantní přírůstek. Namísto zvětšování strany přičítáním přírůstku můžeme stranu násobit konstantou.

```

TO EQSPI :STR :UHEL :MIRA
FD :STR
LT :UHEL
EQSPI :STR * :MIRA :UHEL :MIRA
END

```



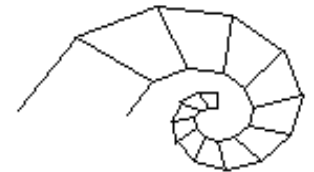
obr. 74.
EQSPI 1 120 1.1,
EQSPI 1 90 1.1,
EQSPI 1 45 1.1

Tato spirála (obr. 74) se nazývá **logaritmická spirála**. Spojíme-li vrcholy spirály s bodem ve

středu, ukáže se, že spirála může být generována posloupností podobných trojúhelníků, každý z nich je postaven na předcházejícím. Tvar logaritmické spirály nacházíme u mnoha výtvarů přírody. Například ulita, v níž žije hlemýžď, se zvětšuje růstem jeho těla, ale zachovává svůj tvar, takže vnitřek hlemýžďova obydlí vypadá stále stejně.

Nyní sestavíme pro želvu program, který napodobuje spirálovitý růst biologických útvarů prostřednictvím hromadění podobných tvarů. Jako základní prvek použijeme čtyřúhelník (komora). Program sestavíme tak, aby každá komora připravila podmínky pro novou komoru. Použitá rekurzivní struktura programu způsobuje, že každá komora generuje novou komoru a program pokračuje stále. Abychom chod programu zastavili, musíme do procedury doplnit testování určité meze (např. velikost základny nové komory, počet komor, apod.) (obr. 75).

```
TO KOMORA :ZAKL :S1 :S2 :U1 :U2
MAKE "START POS
MAKE "USTART HEADING
FD :ZAKL LT :U2 FD :S2
MAKE "X1 XCOR MAKE "Y1 YCOR
PU SETPOS :START SETH :USTART
PD LT :U1 FD :S1
END
```

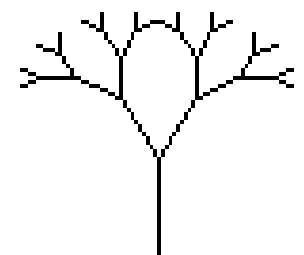


obr. 75.
SPIRAL.RUST 4 8
9 90 50

```
TO SPIRAL.RUST :ZAKL :S1 :S2 :U1 :U2
KOMORA :ZAKL :S1 :S2 :U1 :U2
SETH TOWARDS LIST :X1 :Y1
MAKE "DALSI.ZAKL VZDALENOST :X1 :Y1
MAKE "K :DALSI.ZAKL / :ZAKL
SPIRAL.RUST :DALSI.ZAKL :S1 * :K :S2 * :K :U1 :U2
END
```

Dalším druhem růstového modelu je proces větvení, který charakterizuje růst stromů. Začneme kreslením pravidelného binárního stromu (obr. 76), tj. stromu ve kterém z každé větve vyrůstají dvě další větve. Podstata popisu tohoto stromu je rovněž rekurzivní povahy: větev se skládá z "něčeho" (kmene) a dvou podvětví (věcí se stejnou strukturou jako větev).

```
TO BIN.STROM :DELKA
IF :DELKA < 5 [STOP] ;v tve kratÓí neñ 5 nebudou
;nakreslí hlavní v tev
;natO..ení na v tev niñÓího l ádu
;nakreslení v tve niñÓího l ádu
;natO..ení na pravou v tev
niñÓího l ádu
;nakreslení v tve niñÓího l ádu
;zajiÓt ní, aby byla ñelva po ukon..ení procedury
v té poloze jako na za..átku
;umoñn ní správného napojení v tví niñÓího l ádu
END
```



obr. 76.
BIN.STROM 30

Tento strom však nepřipomíná stromy v přírodě. Budeme ho proto dále modifikovat. Zajistíme:

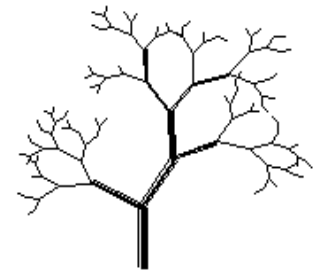
- náhodnou délku větví;
- variace úhlů větvení;
- odlišení kmene od větví, tj. tloušťka větví bude závislá na stupni větve. (obr. 77)

```

TO STROM :DELKA :S1 :S2
IF :DELKA < 5 [STOP]
KMEN :DELKA :DELKA / 10
LT :S1
STROM :DELKA / 2 + RANDOM :DELKA / 2 40 - RANDOM 12 40 - RANDOM 12
RT :S1
RT :S2
STROM :DELKA / 2 + RANDOM :DELKA / 2 40 - RANDOM 12 40 - RANDOM 12
LT :S2
BK :DELKA
END
    
```

```

TO KMEN :A :S
IF :S < 2 [FD :A STOP]
REPEAT :S - 1 [FD :A BK :A RT 90 FD 1 LT 90]
FD :A LT 90
FD :S - 1 RT 90
END
    
```



obr. 77.
STROM 40 60 30

Cvičení

- Zobecněním procedur HRBOL.VPRED1, HRBOL.VPRED2, HRBOL.VPRED3 sestavte proceduru HRBOL.VPRED :ST :D, pomocí které lze nakreslit libovolný z obrázků výše zmíněných procedur (obr. 78).

```

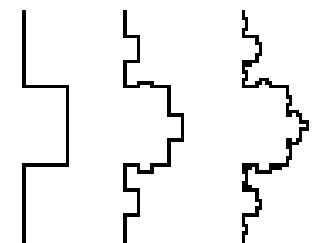
TO HRBOL.VPRED1 :D
FD :D / 3 RT 90
FD :D / 6 LT 90
FD :D / 3 LT 90
FD :D / 6 RT 90
FD :D / 3
END
    
```

```

TO HRBOL.VPRED2 :D
HRBOL.VPRED1 :D / 3 RT 90
HRBOL.VPRED1 :D / 6 LT 90
HRBOL.VPRED1 :D / 3 LT 90
HRBOL.VPRED1 :D / 6 RT 90
HRBOL.VPRED1 :D / 3
END
    
```

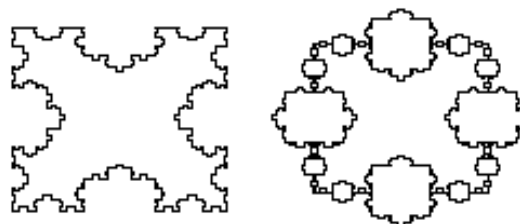
```

TO HRBOL.VPRED3 :D
HRBOL.VPRED2 :D / 3 RT 90
HRBOL.VPRED2 :D / 6 LT 90
HRBOL.VPRED2 :D / 3 LT 90
HRBOL.VPRED2 :D / 6 RT 90
HRBOL.VPRED2 :D / 3
END
    
```



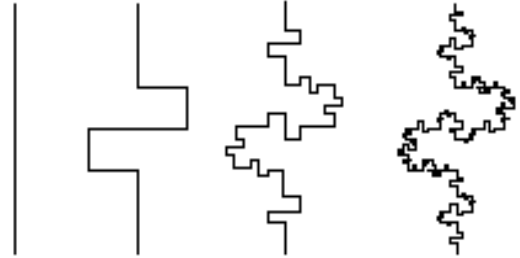
obr. 78.
HRBOL.VPRED1 80,
HRBOL.VPRED2 80,
HRBOL.VPRED3 80

- Pomocí zobecněné procedury HRBOL.VPRED :ST :D nakreslete následující obrázky (obr. 79).



obr. 79.

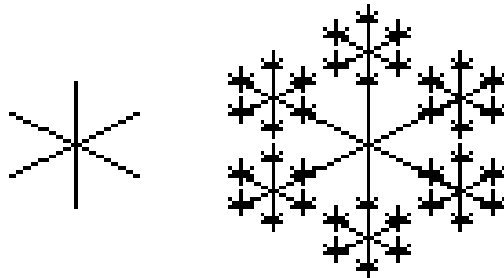
- b) Obměňte proceduru HRBOL.VPRED :ST :D tak, abyste nakreslili následující obrázky (obr. 80).



obr. 80.

```
N.HRBOL.VPRED 0 120,
N.HRBOL.VPRED 1 120,
N.HRBOL.VPRED 2 120,
N.HRBOL.VPRED 3 120
```

2. Obměňte proceduru HVEZDA tak, abyste získali rekursivní proceduru kreslící hvězdu, která má, až do určité



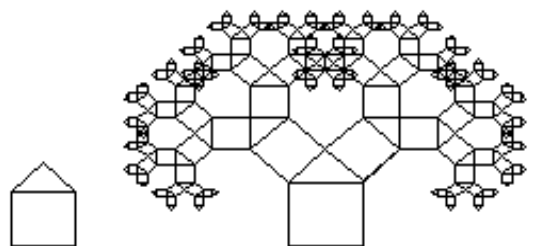
obr. 81. HVEZDA 20, HVEZDA2 30

meze, na každém svém paprsku další hvězdu. Úkolem je tedy vhodně umístit rekursivní volání a zastavit toto volání po dosažení určité meze (např. IF :VEL < 2 [STOP] (obr. 81).

```
TO HVEZDA :VEL
REPEAT 6 [FD :VEL BK :VEL RT 60]
END
```

3. Obměňte proceduru CHALOUPKA tak, aby jste získali rekursivní proceduru, která místo obou odvěsen trojúhelníka tvořícího střechu chaloupky kreslí menší chaloupky (viz obr. 82). Základní chaloupka tedy rekurentně volá na dvou místech same sebe k nakreslení menších chaloupek - toto rekursivní volání se opakuje až po určitou mez. Vytvoření příslušné procedury znamená např. vhodně umístit příkaz IFELSE :VEL < :MEZ [FD :VEL / SQRT 2] [CHALOUPKA :VEL / SQRT 2 :MEZ] a upravit parametry procedury CHALOUPKA.

```
TO CHALOUPKA :VEL
LT 90 REPEAT 4 [FD :VEL RT 90]
FD :VEL RT 45
FD :VEL / SQRT 2
RT 90
FD :VEL / SQRT 2
RT 45 FD :VEL LT 90
```



obr. 82. CHALOUPKA 30, CHALOUPKA2 35 5

END

6 Literatura

- [1] ABELSON, H.: Turtle geometry. The Computer as a Medium for Exploring Mathematics. The MIT Press, Cambridge Massachusetts, 1981.
- [2] Mc DOUGALL, A. - ADAMS, T. - ADAMS, P.: Learning LOGO on the APPLE II. Firemní dokumentace počítače APPLE.
- [3] FRIENDLY, M.: Advanced LOGO a Language for learning. New Jersey, Hillsdale, 1988.
- [4] HOYLES, C.: Culture and computers in the mathematics classroom. University of London, 1985.
- [5] PAPERT, S. A.: Mindstorms. Brighton, The Harvester Presss, 1980.
- [6] PAPERT, S. A.: Počítače a škola. in: Pokroky matematiky, fyziky a astronomie, roč. 29 (1984) č. 1.
- [7] PIAGET, J.: Psychologie dítěte. SPN, Praha 1970, 1. vyd.
- [8] PIAGET, J.: Psychologie inteligence. SPN, Praha 1970, 1. vyd.
- [9] STELLER, E.: Mathematik mit LOGO. Stuttgart, 1986.