Výroční zpráva

IX/88

ZPRAVODAJ
ATARI
Klub Praha

Vydává 487. ZO Svazarmu — ATARI KLUB v Praze 4.
Šéfredaktor a vedoucí redakční rady JUDR. Jan Hlaváček.
Zástupce šéfredaktora ing. Stanislav Borský.
Obálku navrhl RNDr. J. Tamchyna.
Adresa redakce:
487. ZO Svazarmu - ATARI KLUB Praha
REDAKCE
poštovní přihrádka 51
100 00 Praha 10
Řídí redakční rada: V. Bílek, ing. J. Biskup, RNDr. J. Bok, CSc., ing. S. Borský, ing. V. Friedrich, ing. O. Hanuš, RNDr. L. Hejna, CSc., Z. Lazar, prom. fyz., CSc., F. Tvrdek, ing. M. Vavrda.
Technická redakce Otilie Strnadová.
Otisk povolen se souhlasem redakce při zachování autorských práv a s uvedením pramene. Rukopisy nevyžádané redakcí se nevracejí. Za původnost a věcnou správnost ručí autor.
Vychází šestkrát ročně. Neprodejné. Členům klubu distribuováno zdarma. Nepravidelné přílohy na objednávku jsou kompenzovány zvláštním klubovým příspěvkem.
Rozsah čísla 140 stran. Neprošlo jazykovou úpravou.
TISK CTK REPRO
Do tisku předáno v lednu 1989

Vydávání schváleno OV Svazarmu Praha 4 a OŠK ONV Praha 4.
Evidenční číslo UVTEI 86 042.

příloha IX

LOGO PRO DĚTI I DOSPĚLÉ

Ing. Vojtěch Sedláček
Ing. Václav Friedrich

PŘEDMLUVA

Do rukou uživatelů počítačů Atari se konečně dostává dlouho očekávaná příručka programování v jazyce Logo. Nebylo nikterak lehké vytvořit publikaci, která by nebyla jenom populárním povídáním, ale také ani nezáživnou učebnicí. Stále jsme totiž museli mít na paměti, že s jazykem Logo přijdou do styku děti i dospělí, začátečníci i pokročilí.

Příručka se proto skládá z několika rozdílně pojatých částí. V první části, nazvané "Výuka hrou", se čtenář může seznámit s projektem LOGO a několika různými pohledy na jazyk Logo. Druhá část, "Od želvy k rekurzi", se snaží přiblížit základní funkce jazyka Logo všem kategoriím uživatelů, tedy i těm nejmenším. V této části jsou formou populárního výkladu a vhodně volených příkladů představovány jednotlivé příkazy a konstrukce jazyka, bez nároků na úplnost a přesné definice. Třetí část příručky, nazvaná "Logo převážně vážně", bude uživatele seznamovat s jednotlivými příkazy a konstrukcemi podrobněji. I v této části naleznete mnoho názorných příkladů a ukázek. Do poslední části, nazvané "Zajímavé projekty", jsme zařadili několik větších programových celků, které tak názorně demonstrují širokou oblast aplikací jazyka Logo.

Doufáme, že naše příručka alespoň trochu pomůže uživatelům počítačů Atari při programování v jazyce Logo a tím se stane jedním z prvních příspěvků v oblasti, která je u nás dosud nepříliš známá a tudíž i málo rozšířená.

Autoři

A. VÝUKA HROU

1. Projekt LOGO a jazyk Logo

V poslední době se stále více hovoří o informační explozi a nutnosti získávání počítačové gramotnosti u co nejširší oblasti lidí. Současný rozvoj lidské společnosti je charakterizován vysokým nárůstem informací, které již nelze zpracovávat a vyhodnocovat stávajícími metodami. Ke slovu se dostává výpočetní technika, která tak začíná zasahovat do všech oborů lidského poznání, včetně humanitních a společenských věd. Počítače vytlačují kartotéky, archívy, psací stroje i kalkulačky, počítačové sítě nahrazují telefon a dálnopis.

Pro člověka současné epochy představují tyto tendence nutnost seznámit se s výpočetní technikou a zvládnout její obsluhu jako uživatel. Výuka počítačové gramotnosti se stává běžnou součástí výchovného procesu ve všech vyspělých zemích světa. Počítačová gramotnost znamená dokázat využít výhody výpočetní techniky všude, kde je to možné. Proto se ve výchovném procesu klade velký důraz na logické, kombinatorické a algoritmické myšlení.

Ve výuce počítačové gramotnosti dosahují velkých úspěchů projekty založené na využití prostředků umělé inteligence. Do této kategorie patří i československá akce Karel, ze zahraničních patří mezi nejúspěšnější a nejznámější projekt "Logika jako programovací jazyk" profesora Kowalského z londýnské Imperial College, který využívá jazyk Prolog, a americký výukový projekt LOGO z MIT, založený na jazyce LISP.

Projekt LOGO z massachussetské university zaznamenal ve světě obrovský úspěch. Jazyk Logo se stal velice populárním programovacím prostředkem, a to i u nejmladší generace. Tzv. želví grafika (turtle graphics), rekurzivní programování a další principy využívané v Logu se dají s úspěchem aplikovat i ve většině vyšších programovacích jazyků, například v Pascalu. Návyky získané při práci s jazykem Logo se tak stávají použitelnými zcela obecně.

Autor jazyka Logo, profesor Seymour Pappert z MIT, vytvořil skutečně živý programový prostředek schopný plynulé adaptace podle potřeb svých uživatelů. Programování v Logu přestává být nežádoucnou rutinou, ale stává se zajímavou a názornou hrou. Logo si tak získává oblibu u dětí sotva školou povinných stejně jako u dospělých, u počítačových začátečníků stejně jako u profesionálů.

2. Principy práce s jazykem Logo

Jazyk Logo vznikl odvozením z jazyka LISP, který byl výchozím programovacím prostředkem pro celý projekt LOGO. Logo je plnohodnotný programovací jazyk, který umožňuje řešit úlohy grafického, numerického i nenumerického charakteru (například práce s textem). Přitom byl navrhován s ohledem na jednoduché osvojení a širokou adaptabilitu.

Nejzajímavější a zpočátku jistě nejatraktivnější součástí Loga je želví grafika. Logo je však vybaveno i všemi základními logickými a aritmetickými operacemi a funkcemi, včetně goniometrických. Pro aplikace nenumerického charakteru obsahuje Logo skupinu efektivních procedur pro zpracování seznamu. V této oblasti použití je nejvíce vidět příbuznost Loga s jazykem LISP.

Základními programovými konstrukcemi v Logu jsou větvení a rekurze. Pomocí rekurzivních procedur je možno často velice jednoduše řešit úlohy, které by jinak byly řešitelné jen obtížně. Rekurzí lze také opsat všechny programové konstrukce známé z vyšších jazyků, tedy i skoky a cykly.

Jazyk Logo umožňuje také pracovat s datovými strukturami, konstantami a proměnnými. Složité datové struktury opisuje pomocí hierarchicky uspořádaných seznamů obdobně jako v jazyce LISP. Velmi obecně je chápán také pojem hodnoty.

Logo je procedurální jazyk. To znamená, že všechny úlohy v tomto jazyce se řeší formou návrhu nových procedur, které tak rozšiřují základní slovník jazyka. Při řešení problémů v Logu tedy nevytváříme kompaktní program, ale množinu nových procedur: příkazů a funkcí. Tuto množinu budeme pro jednoduchost (a kvůli odlišení od pojmu program) nazývat projektem.

Tato koncepce jazyka umožňuje jednoduché a přehledné řešení i jinak poměrně složitých úloh. Na rozdíl od jiných programovacích jazyků také nemusí uživatel ihned zvládnout rozsáhlý slovník klíčových slov. Zpočátku a mnohdy i na delší dobu vystačí vlastně pouze s několika základními slovy, pravidly a konstrukcemi.

Vytvořením projektu se rozšiřuje slovník jazyka o nová slova, nové konstrukce. Tímto způsobem vlastně vzniká z Loga nový jazyk, přizpůsobený a orientovaný na potřeby programátora nebo uživatele. Schopnost jazyka rozšiřovat a modifikovat sebe sama je typickou vlastností jazyka Logo a činí ho použitelným prakticky zcela obecně.

3. Logo a logické myšlení

Jazyk Logo byl vytvořen jako součást výukového projektu, který ma za úkol zformovat a posílit logické myšlení budoucího uživatele výpočetní techniky. Pracovat úspěšně s výpočetní technikou totiž neznamená umět naprogramovat několik triviálních úloh v některém z vyšších programovacích jazyků, ale zejména umět logicky myslet.

Tímto směrem je veden i uživatel jazyka Logo. Při řešení libovolné úlohy je nucen osvojit si proces analýzy a syntézy problému, metodu rozkladu úlohy na jednodušší podúlohy a jejich samostatné řešení. Procedury jazyka, standardní i nově vytvořené, se pak používají jako stavební kostky k vytváření složitějších struktur. Tento přístup k řešení úloh na počítači je znám jako strukturované programování a představuje formu strukturovaného logického myšlení.

Jazyk Logo rozvíjí i další formy myšlení, například učí hledat a odstraňovat chyby ve vlastních úsudcích. Chybná úvaha při řešení úlohy vede vždy k nesprávné činnosti navrhované

procedury, což umožňuje její rychlé a názorné odhalení. Odstraňováním chyb si uživatel vytváří a tříbí své trasovací a ladící schopnosti, naučí se rozlišovat chyby podstatné, logické, od nepodstatných, které jsou spíše "kosmetického" charakteru.

Je známým faktem, že aktivní absolventi kursu jazyka Logo se mnohem lépe přizpůsobují novým trendům v informatice, také mnohem lépe zvládají výuku vyšších programovacích jazyků. Vytříbené logické myšlení dokáží aplikovat i v běžném životě, to znamená i v situacích, které s výpočetní technikou bezprostředně nesouvisí.

4. Oblasti využití jazyka Logo

Vlastnosti jazyka Logo předurčují tento programovací jazyk k využití v různých oblastech programování a aplikaci výpočetní techniky.

Jazyk Logo byl původně vytvořen jako logický výukový prostředek, který lze využít prakticky ve všech oblastech logického programování. Od jednoduchých kreseb a matematických úloh lze přitom postupně přecházet k abstraktnějším problémům, jako je analýza textu, zpracování dat, řešení různých logických problémů a hlavolamů (např. myš v bludišti).

Na Logo lze však hledět i jako na ucelený grafický systém. Obsahuje základní grafické příkazy pro bodovou i tzv. želví grafiku a není složité vytvořit další (např. pro kreslení kružnice, obdélníku apod.). Lze například vytvořit jednoduchý grafický editor spolupracující s grafickou tabulkou, zobrazovat grafy zadaných aritmetických funkcí nebo vytvářet různé typy diagramů ze vstupního datového souboru.

Logo je však také plnohodnotným programovacím jazykem. Umožňuje pracovat s celočíselnými, reálnými i textovými hodnotami, obsahuje základní programové struktury a příkazy, aritmetické, logické a znakové funkce. Chybějící příkazy cyklu "for", "repeat" a "while" lze opsat pomocí rekurzivního volání nebo přímo naprogramovat. V jazyce Logo lze tedy řešit i úlohy numerického charakteru.

Jazyk Logo lze také využít pro zpracování seznamů, a to i víceúrovňových. Na seznamy lze převést prakticky všechny datové struktury používané ve vyšších programovacích jazycích (pole, záznam, strom, dynamický seznam apod.). To umožňuje řešit v Logu úlohy na zpracování textu nebo dat, i úlohy z oblasti umělé inteligence, jako jsou například expertní a dialogové systémy.

Interaktivní a interpretační charakter jazyka Logo však s sebou přináší i určité nevýhody. Malá rychlost zpracování a omezená velikost pracovní paměti RAM neumožňují obvykle nasadit tento jazyk v profesionálních aplikacích (např. jako databázi nebo textový editor). Principy získané při návrhu a studiu jednoduchých aplikačních úloh z dané oblasti však lze s úspěchem využít i u specializovaných profesionálních programů, a to je jedna z největších předností, která Logu zajistila trvalou popularitu mezi uživateli všech věkových i profesních skupin.

B. OD ŽELVY K REKURZI

Tato část příručky je úvodním kursem jazyka Logo. Je určena především začátečníkům, ale přečtou si ji jistě i ti pokročilejší. Každá kapitola obsahuje řadu příkladů a je ukončena slovníčkem, který shrnuje všechny nové pojmy.

5. Jak vyvolat Logo

5.1 Zavedení jazyka do počítače

Co potřebujeme k tomu, abychom mohli pracovat s jazykem Logo? Samozřejmě počítač, televizní přijímač a programovací jazyk Logo. Ten je buď v zásuvné jednotce (cartridge), na kazetě nebo na disketě. Podle toho pak ještě potřebujeme kazetový magnetofon nebo disketovou jednotku. Na kazetě či disketě můžeme také uchovávat naše programy.

Vlastní zacházení s počítačem je dostatečně popsáno v příručce, která se s počítačem dodává. Proto si řekneme, co je třeba udělat k zavedení jazyka Logo do našeho počítače:

a) Zásuvná jednotka:

1. Zapneme televizní přijímač nebo monitor, počítač ponecháme vypnutý.

2. Máme-li připojenou disketovou jednotku, zapneme ji a vložíme disk s operačním systémem DOS.

3. Do počítače vložíme zásuvnou jednotku (cartridge) s programovacím jazykem Logo a zapneme jej.

b) Kazetový magnetofon:

1. Zapneme televizní přijímač nebo monitor, počítač ponecháme vypnutý.

2. Stiskneme současně klávesy START a OPTION a zapneme počítač. Ozve se akustická návěst. Na magnetofonu stiskneme PLAY, pak stiskneme nějakou klávesu (např. mezerník) a motor magnetofonu se rozběhne.

c) Disketová jednotka:

1. Zapneme televizní přijímač nebo monitor, počítač ponecháme vypnutý. Zapneme rovněž disketovou jednotku a počkáme, až zhasne kontrolka.

2. Do jednotky zasuneme disketu s operačním systémem (DOS) a jazykem Logo.

3. Stiskneme klávesu OPTION a zapneme počítač. Po zavedení

operačního systému do paměti načteme obvyklým způsobem soubor s jazykem Logo. Vlastní způsob zavedení programu závisí od použitého operačního systému.

Ve všech třech případech se po načtení programu objeví hlavička:

```
(C) 1983 LCSI ALL RIGHTS RESERVED
WELCOME TO ATARI LOGO.
?_
```

(Všechna práva vyhrazena, vítá vás ATARI LOGO.)

LCSI je zkratka firmy LOGO COMPUTER SYSTEMS INC., která vytvořila jazyk Logo na počítače Atari a Apple.

Otazník na začátku následujícího řádku znamená, že můžeme psát, podtržením je znázorněn kurzor, tedy místo, kam budeme psát.

*** POZOR ***
Není-li tomu tak, zkontrolujeme, zda je počítač s televizním přijímačem či monitorem správně propojen, zda je správně vložena zásuvná jednotka nebo zda při načítání nedošlo k chybě. Zopakujeme případně předcházející kroky.

5.2 S čím Logo pracuje?

Klávesnice, kterou je počítač ATARI vybaven, je podobná klávesnici psacího stroje. Na zkoušku napíšeme nějaké slovo, ačkoliv mu Logo pravděpodobně nebude rozumět. Například:

```
?AHOJ
```

Stiskneme klávesu RETURN a Logo nám odpoví:

```
I DON'T KNOW HOW TO AHOJ      (Nevím si rady s AHOJ)
```

Můžeme psát, co chceme. Počítači ani programu Logo tím neublížíme. V nejhorším případě můžeme počítač vypnout a zavést Logo od začátku.

Písmena (A,B,C az Z), číslice (0 až 9) a všechny ostatní znaky (např. ,?,*) jsou znakové klávesy. Pomocí těchto kláves můžeme psát texty. Oproti tomu funkční klávesy mají speciální význam:

RETURN

V jazyce Logo má klávesa RETURN programovou funkci ve významu: nyní proveď, co jsem napsal! Logo přijme naši instrukci, teprve po stisku této klávesy.

MEZERNÍK

Mezerníkem píšeme sice neviditelný, ale velmi důležitý znak, a to mezeru. Používá se jako oddělovač slov. Napíšeme-li

například TOJESLOVO, Logo tomu bude rozumět jako slovu jednomu, zatímco když napíšeme TO JE SLOVO, pak to bude Logo pokládat za slova tři.

SHIFT

Pokud je nějaká klávesa stisknuta spolu s klávesou SHIFT (přemykač), pak se uplatní její druhý význam. Například hranatou závorku napíšeme pouze s pomocí tohoto přemykače.

Hranaté závorky jsou v jazyce Logo, na rozdíl od kulatých, velmi důležité. Klávesu SHIFT stiskneme vždy jako první, pak teprve další.

CONTROL

Klávesa CONTROL (CTRL) dává stisknuté klávese další funkční význam. Samotná klávesa CTRL nic neznamená. Na obrazovce se nemusí pokaždé něco napsat, ale Logo tomu bude rozumět, např.:

CTRL * posune kurzor o jednu pozici vlevo,
CTRL * posune kurzor o jednu pozici vpravo.

Klávesy se šipkami jsou užitečné hlavně při editování. Umožňují nám pohybovat kurzorem, aniž bychom ovlivnili samotný text.

*** POZOR ***

Pomocí CTRL klávesy s příslušnou šipkou můžeme pohybovat kurzorem nahoru a dolů. To však platí pouze v editoru.

Text můžeme vkládat tak, že ho prostě píšeme do místa, kam je nastaven kurzor. Podobně můžeme text znak po znaku rušit.

DELETE

Touto klávesou posunujeme kurzor doleva, znak na pozici kurzoru se zruší.

BREAK

Klávesou BREAK zastavíme vše, co Logo právě dělá. Můžeme tak opustit i editor a tím zabránit, aby se provedly změny. Po stisknutí této klávesy Logo vypíše:

```
?STOPPED!          (Zastaveno!)
?_
```

a můžeme psát další příkazy.

ESC

Tato klávesa slouží k opuštění editoru. Podrobněji se o této i dalších speciálních editovacích funkcích zmíníme v následující kapitole.

ATARI KEY

Tuto klávesu najdeme v pravém dolním rohu klávesnice. Je označena symbolem Atari nebo dvoubarevným čtverečkem. Po stisku klávesy budou všechny znaky vystupovat invertovaně (tzn. tmavé znaky na světlém pozadí). Druhé použití klávesy vede zpět k normálnímu zobrazení.

CAPS

Po zapnutí počítače je vše psáno velkými písmeny. Po stisknutí této klávesy se zobrazují malá písmena. Všechna klíčová slova a příkazy jazyka Logo však musí být psány pouze velkými písmeny, jinak jim Logo nebude rozumět.

SHIFT + CAPS

Klávesa SHIFT spolu s klávesou CAPS přepne zpět na velká písmena.

RESET

Při práci s jazykem Logo tuto klávesu obvykle nepoužíváme. Vymazali bychom si totiž obsah operační paměti, tj. všechny dosud vytvořené konstrukce.

*** POZOR ***
U některých verzí jazyka Logo (např. disketové) dojde při stisknutí RESET i k vymazání jazyka Logo. Musíme vypnout počítač a program zavést znovu

6. Psaní a příkazy tisku

6.1 Příkaz PRINT (PR)

Zkusme napsat pozdrav:

PRINT [DOBRY DEN]

Na obrazovce se instrukce sice objevila, ale Logo ji přijme teprve až stiskneme klávesu RETURN. Logo pak na dalším řádku odpoví:

DOBRY DEN

Zkusme jako další zprávu třeba tvrzení:

JA JSEM NEJVETSI

Při psaní jsme ale udělali chybu a napsali místo toho:

PRINT [JA JSEM NEJMENSI]

Co teď? Nepoužijeme klávesu RETURN, kterou bychom příkaz odeslali, ale klávesu DELETE, a to tolikrát, až zůstane:

```
PRINT [JA JSEM NEJ
```

a nyní dopíšeme správně zbytek textu:

```
PRINT [JA JSEM NEJVETSI]
```

Stiskneme RETURN a instrukce se provede:

```
JA JSEM NEJVETSI
```

Funkce DELETE je jednou z několika funkcí jazyka Logo, kterou můžeme změnit to, co jsme napsali, aniž bychom museli psát celý řádek od začátku. S dalšími takovými funkcemi se seznámíme v následujících kapitolách.

Podobně můžeme zkusit vytisknout i jiné věty. Text, který za příkazem PRINT následuje, musí být uzavřen v hranatých závorkách.

6.2 Vymazání obrazovky

Užitečným příkazem, který vyčistí obrazovku a umístí kurzor do výchozí polohy (vlevo nahoře), je příkaz:

```
CT (+RETURN)
```

6.3 Psaní procedur

Definováním procedur můžeme vytvořit zcela nové příkazy. Slouží k tomu klíčové slovo TO. Na témže řádku ještě musíme uvést jméno procedury. Pak řekneme, co má vlastně procedura dělat. Definujme například proceduru POZDRAV, a to tak, že kdykoliv napíšeme příkaz POZDRAV, tak nám Logo odpoví:

```
DOBRY DEN
NASHLEDANOU
?
```

Začneme tím, že napíšeme TO a po mezeře jméno procedury:

```
?TO POZDRAV
```

Na začátku řádku však Logo nyní nebude vypisovat otazník (?), ale symbol >, a to proto, aby nám připomínalo, že píšeme proceduru, a ne příkazy, které by se měly hned vykonávat.

```
>PRINT [DOBRY DEN]
>PRINT [NASHLEDANOU]
>END
?_
```

Příkazem END psaní procedury ukončíme.

*** POZOR ***
Nadále si již nebudeme připomínat, **že za** každým řádkem máme stisknout RETURN nebo že musíme používat mezer mezi slovy. Budeme to pokládat za samozřejmé.

Chceme-li nyní pozdravit tisíckrát, pak napíšeme:

```
?REPEAT 1000 [POZDRAV]
```

Zdravení můžeme zastavit stisknutím klávesy BREAK. Vypíše se:

```
STOPPED IN POZDRAV (Zastaveno v proceduře POZDRAV)
?_
```

6.4 Slovníček

Seznámili jsme se s těmito klíčovými slovy :

PRINT (PR)	tiskni
CT	vymaž obrazovku
TO	definuj proceduru
END	konec procedury
REPEAT	opakuj

Též jsme použili několik zvláštních kláves:

[	levá hranatá závorka
]	pravá hranatá závorka
BREAK	zastav
DELETE	vyřaď
RETURN	zpět (slovo má počítač)

7. Setkání se želvou

V této části se začneme učit programovat pomocí počítačového tvora-želvy. Když se jazyk Logo vyvíjel, používal místo želvy robota, který se pohyboval na kolečkách. K počítači byl "připoután" dlouhým kabelem a kreslil jím jakoby po podlaze.

Naše želva žije na obrazovce. Má kreslící pero a může s ním po obrazovce kreslit. Logo má mnoho příkazů, pomocí kterých můžeme želvu ovládat. S některými nejdůležitějšími se nyní seznámíme.

7.1 Vyvolání želvy

Želva se na obrazovku vyvolá příkazem:

```
ST (SHOW TURTLE)
```

Uprostřed obrazovky se objeví malá želvička. Má takový tvar, aby bylo patrné, kam směřuje. Umístění, neboli stav želvy, je dáno jednak její polohou, jednak jejím natočením (orientací). Nejdůležitější příkazy pro želvu jsou právě ty, které mění její stav.

Na začátku želva směřuje vzhůru. Kurzor se pohybuje pouze ve spodní části obrazu. Příkazy se nyní objevují na posledních pěti řádcích obrazovky, v tzv. "příkazovém oknu".

7.2 Změna umístění želvy

Ke změně umístění želvy slouží čtyři základní příkazy: FORWARD, RIGHT, LEFT a BACK.

7.2.1 Pohyb dopředu

K pohybu želvy dopředu slouží příkaz FORWARD, za kterým následuje číslo. Takové číslo nazýváme vstupem a zde znamená, o kolik kroků se želva posune. Napíšeme následující příkaz:

FORWARD 50

Želva se posune, změní svoji polohu, ale nezmění svoji orientaci.

Jako vstup jsme zvolili 50, ale stejně tak jsme mohli zvolit jakékoliv jiné číslo. Mezera mezi příkazem FORWARD a číslem je samozřejmě důležitá, odlišuje totiž slovo FORWARD od slova FORWARD50. Další mezery mezi slovy jazyk Logo ignoruje.

*** POZOR ****
Při práci s jazykem Logo budeme možná zprvu často chybovat. Mnohé z chyb bývají chyby při psaní. snad nejčastější z nich je právě vynechání mezery mezi příkazem a jeho vstupy. například FORWARD je příkaz, který jako vstup očekává nějaké číslo. FORWARD patří do slovníku jazyka Logo. Takže FORWARD 50 způsobí posun želvy o 50 kroků vpřed, zatímco slovo FORWARD50 jako takové není definováno (pokud jsme si jej sami nedefinovali).

Rozdíl je tedy pouze v mezeře mezi slovy. Podobně rozdíl mezi FRWARD a FORWARD je jenom písmeno O, ale pro Logo je to rozdíl podstatný.

Napíšeme-li:

FRWARD 50

pak Logo odpoví:

I DON'T KNOW HOW TO FRWARD (Nevím, co se slovem FRWARD.)

7.2.2 Rotace doprava

Ke změně natočení (orientaci) želvy směrem doprava slouží příkaz RIGHT. Za příkazem musí být uvedeno číslo, které vyjadřuje počet stupňů, o které se želva otočí. Můžeme zadat otočení želvy i o více než 360 stupňů. Chceme-li například otočit želvu o 90 stupňů, napíšeme:

RIGHT 90

Želva se nyní otočí doprava o 90 stupňů. Želva na obrazovce změnila pouze svoji orientaci, nikoliv svoji polohu.

7.2.3 Rotace doleva

Příkazem LEFT, podobně jako příkazem RIGHT, měníme orientaci želvy, ale opačným směrem, tedy doleva:

```
LEFT 45
```

Želva se otočí o 45 stupňů doleva. Svoji polohu nemění. Účinek natočení bude vidět jasněji, když nyní želvu necháme popojít o 25 kroků:

```
FORWARD 25
```

7.2.4 Pohyb dozadu

Příkaz BACK způsobí, že se želva posune zpět, změní svoji polohu:

```
BACK 30
```

Číslo 30 jsme zase zvolili pouze jako příklad. Mohli bychom samozřejmě zvolit jakékoliv číslo jiné.

7.2.5 Mazání obrazovky

Občas potřebujeme vymazat obrazovku a začít znovu. Udělá to příkaz CS, který vymaže všechny želví stopy a želvu umístí do výchozí polohy s orientací ve středu obrazovky. Příkaz píšeme samotný, nevyžaduje žádné vstupy:

```
CS
```

7.3 Slovníček

V této kapitole jsme se seznámili s novými příkazy jazyka Logo:

BACK (BK)	zpět
CS	vymaž obrazovku
FORWARD (FD)	vpřed
LEFT (LT)	vlevo
RIGHT (RT)	vpravo
SHOW TURTLE (ST)	nastav želvu

8. Učíme želvu kreslit

8.1 Kreslíme čtverec

Nakreslit například čtverec může želva snadno pomocí příkazů FORWARD, RIGHT nebo LEFT. Často budeme místo klíčových slov používat jejich zkratek (patří též mezi klíčová, tj. vyhrazená slova). Napíšeme tedy:

```
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
```

Můžeme přitom nahradit číslo 30 například číslem 50, pak želva nakreslí čtverec větší.

Definujme nyní příkaz, kterým želva nakreslí celý čtverec najednou. Nové procedury, které napíšeme, se automaticky stávají novými příkazy jazyka Logo. Pokaždé pak, když budeme chtít čtverec nakreslit, použijeme naši novou proceduru, místo toho, abychom opět vypisovali jednotlivé příkazy.

Chceme-li definovat nový příkaz, musíme pro něj nejprve vybrat jméno. My zvolíme jako jméno slovo ČTVEREC. Mohli bychom samozřejmě zvolit i jméno jiné. Pomocí příkazu TO budeme definovat CTVEREC, podobně jako jsme to udělali při definování procedury POZDRAV.

Nejprve tedy napíšeme TO CTVEREC, pak jednotlivé příkazy a nakonec END. Tato metoda je však dobrá pouze tehdy, jsme-li v psaní pečliví.

8.2 Úvod do editoru Atari Logo

Jiný způsob, jak definovat proceduru, spočívá v použití editoru, kterým je jazyk Logo vybaven. Když totiž uděláme chybu, můžeme ji v editoru snadno opravit. Při použití editoru se však ukáží také jeho nevýhody. V editoru se dají příkazy pouze opravovat, a ne spouštět. Obrázky, které dosud želva nakreslila, jsou přepsány editovaným textem a tedy ztraceny.

Na druhé straně, pokud definujeme proceduru pouze příkazem TO, nemůžeme opravovat předcházející řádky, jak to umí editor.

Příkazem EDIT, nebo jeho zkratkou ED sdělíme jazyku Logo, že budeme editovat. Za příkazem musí následovat jméno procedury, kterou hodláme editovat. Bezprostředně před tímto jménem musíme vždy napsat uvozovky. Mezi uvozovkami a jménem nesmí být mezera. Tedy:

```
EDIT "CTVEREC
```

Stiskneme RETURN a můžeme začít s editováním.

*** POZOR ***
Pokud bychom na uvozovky zapomněli a napsali bychom pouze:

```
EDIT CTVEREC
```

odpoví nám Logo:

```
I DON'T KNOW HOW TO CTVEREC
```

tedy, že našemu požadavku nerozumí.

Jakmile začneme editovat, jak jsme již uvedli, kresba na obrazovce zmizí. Editor začne tím, že na první horní řádek obrazovky vypíše titul:

```
TO CTVEREC
```

Příkaz TO říká, že následuje definice procedury, CTVEREC je jméno vybrané procedury.

Kurzor je na začátku titulního řádku. V editoru se můžeme orientovat pouze podle kurzoru. Vzhledem k tomu, že titulní řádek měnit nechceme, stiskneme CTRL = a kurzor poskočí na následující řádek. Nyní již můžeme psát příkazy procedur CTVEREC:

```
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
```

Zdůrazněme, že vše, co v editoru píšeme, je řetězec znaků Chceme-li k řetězci někde něco přidat, pak tam stačí přesunou kurzor:

CTRL + posouvá kurzor zpátky
CTRL * posouvá kurzor dopředu

Vše, co napíšeme, se stává součástí řetězce. Pokud udělám chybu, přesuneme pomocí CTRL + kurzor k místu chyby. Znaky přes které kurzor přechází, se nemění.

Znak vymažeme tak, že na něj nastavíme kurzor a pak použijeme klávesu CTRL DELETE. Znak zmizí.

Je-li kurzor na konci textového řádku, pak stisknutím CTRL DELETE se přesune následující řádek za kurzor:

```
FD 30_
RT 90
```

pak CTRL DELETE způsobí:

```
FD 30_RT 90
```

Takto lze spojovat několik řádků v jediný.

Na konci editování napíšeme poslední řádek END a editor opustíme stisknutím klávesy ESC. Objeví se opět normální obrazovka a Logo nám odpoví:

```
CTVEREC DEFINED      (CTVEREC definován)
```

*** POZOR ***
Zapomeneme-li uvést poslední příkaz END, může dojít k zhroucení celého systému a musíme jazyk Logo natáhnout znovu.

8.3 Používání nových příkazů

Náš nový příkaz vyzkoušíme. Napíšeme:

```
CTVEREC
```

a želva nakreslí čtverec. Ještě jednou napíšeme:

```
CTVEREC
```

Nyní želva svoji dráhu jednoduše zopakuje.
Když nejprve želvu natočíme a poté napíšeme příkaz CTVEREC, objeví se nová kresba:

```
RIGHT 45
CTVEREC
```

8.4 Použití příkazu REPEAT

Příkazem REPEAT můžeme zadat opakování jednoho či více příkazů. Příkaz REPEAT vyžaduje dva vstupy. Například:

```
REPEAT 3 [RT 45 CTVEREC]
```

První vstup určuje počet opakování. Druhý vstup je seznam instrukcí, které se mají opakovat. Instrukce musí být uzavřeny v hranatých závorkách. Závorky zde fungují jako obálka. Tedy znovu:

```
REPEAT 3 [RT 45 CTVEREC]
```

Zkusme nyní pomocí předcházející procedury udělat proceduru novou. Nazveme ji CTVEREC.HVEZDA:

```
EDIT "CTVEREC.HVEZDA
```

Jsme v editoru, na obrazovce se objeví titulní řádek:

```
TO CTVEREC.HVEZDA
```

Kurzor je na začátku řádku. Protože titulní řádek měnit nechceme, stiskneme CTRL = a pokračujeme v psaní:

```
REPEAT 8 [CTVEREC RT 45]
END
```

Editaci ukončíme klávesou ESC.

Novou proceduru hned vyzkoušíme. Nejprve ale nastavíme želvu do výchozí pozice. Vzpomeňme na příkaz CS. Napíšeme jej:

```
CS
```

a pak:

```
CTVEREC.HVEZDA
```

Želva zůstala uprostřed kresby. Nechceme-li, aby nám její přítomnost kresbu rušila, napíšeme příkaz:

```
HT
```

Tímto příkazem uděláme želvu neviditelnou. Zpět ji zviditelní příkaz, který již známe:

```
ST
```

*** POZOR ***
Pokud čtverce vypadají spíše jako obdélníky, není závada v jazyce LOGO, ale v televizním přijímači či monitoru. Geometrii zobrazení můžeme však změnit příkazem .SETSCR. Zkusme:

```
.SETSCR 0.85
```

Pak napíšeme:

```
CS CTVEREC
```

Je-li čtverec ještě horší, zkusme například:

```
.SETSCR 1.1
CS CTVEREC
```

případně vyzkoušíme další hodnoty, až budeme se zobrazením spokojeni.

8.5 Další využití procedury CTVEREC

Jednou definovanou proceduru můžeme používat podobně jako ostatní slova jazyka LOGO, tedy jako FD, BK, LT, RT apod. Stejně může takovou proceduru využívat i jiná procedura. To je jedna z významných vlastností jazyka LOGO. Proceduru CTVEREC můžeme používat různými způsoby, například k vytvoření následujících procedur VLAJKA a VLAJKA.ZPET. Procedury se budou lišit především tím, že zanechají po nakreslení obrázku želvu v různých polohách. Procedura VLAJKA ponechá želvu v jiné poloze, než v jaké byla před kreslením, zatímco procedura VLAJKA.ZPET vrátí želvu do původní polohy. Orientaci želvy obě procedury po ukončení nemění. Důsledky těchto rozdílů vidíme v procedurách KRIZ a VLAJKY:

```
TO VLAJKA
FD 30
```

```
CTVEREC
END

TO KRIZ
REPEAT 4 [VLAJKA RT 90]
END

TO VLAJKA.ZPET
VLAJKA
BK 30
END

TO VLAJKY
REPEAT 4 [VLAJKA.ZPET RT 90]
END

TO VLAJKY.VLAJKY
VLAJKY
RT 45
VLAJKY
END
```

Když nyní počítač vypneme, všechny procedury, které jsme napsali, ztratíme. V následující kapitole si však řekneme něco o jejich trvalém uložení.

8.6 Slovníček

Poznali jsme několik nových příkazů:

EDIT (ED)	vyvolej textový editor
HT	zneviditelní želvu
.SETSCR	nastav geometrii zobrazení

Dále jsme se seznámili s několika novými speciálními klávesami:

BREAK	zastav
CTRL *	posun doprava
CTRL +	posun doleva
CTRL -	posun nahoru
CTRL =	posun dolů
CTRL DELETE	vymaž znak
ESC	ukonči textový editor

9. Jak ukládat a načítat procedury

Procedury, které definujeme, ukládá LOGO do své pracovní oblasti. Pracovní oblast je část operační paměti počítače a její obsah se uchovává jen po dobu zapnutí počítače. Trvale můžeme procedury uložit pouze na kazetu nebo disketu. K uložení slouží příkaz SAVE, k opětnému vyvolání příkaz LOAD.

*** POZOR ***

K uložení a načtení procedur samozřejmě potřebujeme kazetový magnetofon nebo disketovou jednotku.

9.1 Práce s kazetou

Chceme-li ukládat procedury na kazetu, vložíme nejprve do kazetového magnetofonu prázdnou kazetu a převineme ji na začátek. Vynulujeme počítadlo odvinutého pásku a napíšeme:

```
SAVE "C:
```

Symbol C: je označení kazetového magnetofonu. Sdělujeme tím počítači, že ma ukládat na kazetu. Když klávesou RETURN příkaz SAVE "C: odešleme, zazní dvakrát akustická signalizace. Na magnetofonu stiskneme současně PLAY a RECORD a poté na klávesnici počítače ještě jednou RETURN. Nyní se nahraje soubor všech vytvořených procedur. Po nahrání se na obrazovce opět objeví kurzor a my můžeme počítač vypnout. Na tutéž stranu kazety můžeme uložit i více různých souborů. Stačí, když si poznamenáme vždy na konci nahrávání stav počítadla. Je dobré si poznamenat i jméno nebo stručný popis každého souboru. Mezi jednotlivými soubory na kazetě je vhodné vytvořit odvinutím pásku krátké mezery. Stačí odvinout asi 5 až 10 čísel podle počítadla.

Při načtení zpět z kazety nastavíme pásek podle stavu počítadla na začátek požadovaného souboru a napíšeme:

```
LOAD "C:
```

Opět příkaz odešleme klávesou RETURN, zazní akustická signalizace, na magnetofonu stiskneme tlačítko PLAY a na počítači ještě jednou klávesu RETURN. Vše, co bylo předtím uloženo, se nyní načte zpět do pracovní oblasti. Po úspěšném načtení se objeví na obrazovce jména načtených procedur a kurzor.

9.2 Práce s disketou

9.2.1 Zápis a čtení procedur

Na disketu se ukládá celá pracovní oblast jako jediný diskový soubor, který musí být označen jménem. Soubory na disketě musí mít různá jména. Jména mohou být dlouhá jeden až osm znaků s volitelným tříznakovým rozšířením. Prvním znakem ve jméně musí být písmeno. Všechna použitá písmena musí být velká. Je-li použito ve jméně volitelné rozšíření, je nutné ho oddělit od jména tečkou. Například chceme uložit na disketu soubor, který se má jmenovat NAVRH s rozšířením 001. Napíšeme:

```
SAVE "D:NAVRH.001
```

Jméno souboru musí být jedno slovo, kterému bezprostředně

předchází kód zařízení (v našem případě D:) a uvozovky. Příkaz odešleme klávesou RETURN. Až bude soubor uložen, objeví se na obrazovce kurzor a my můžeme počítač vypnout.

*** POZOR ***
Ukládání souboru je možné jenom na diskety, které byly předem naformátovány. Podrobnosti viz diskový manuál dodávaný s disketovou jednotkou.

K prečtení zpět z diskety slouží příkaz LOAD, který je obdobou příkazu SAVE:

```
LOAD "D:NAVRH.001
```

Vše, co jsme předtím uložili do souboru NAVRH.001, se přesune do pracovní oblasti paměti počítače.

*** POZOR ***
Pokud jsme však zapnuli počítač dříve než disketovou jednotku, odpoví nám Logo:

```
I CAN'T OPEN D:NAVRH.001
```

(Soubor NAVRH.001 nemohu otevřít.)

9.2.2 Seznam uložených souborů

Potřebujeme-li zjistit, které soubory jsou uloženy na disketě, použijeme příkazu CATALOG s parametrem typu zařízení, např.:

```
CATALOG "D:
```

Na obrazovce se vypíše seznam jmen všech uložených souborů.

9.2.3 Zrušení souboru

Soubory uložené na disketě je možné zrušit příkazem ERF. Jeho vstupem je jméno souboru, kterému předchází "D: (podobně jako u příkazu LOAD nebo SAVE):

```
ERF "D:NAVRH.001
```

Příkaz způsobí vymazání souboru, který se jmenuje NAVRH.001.
Příkazem ERF však nelze vymazávat soubory uložené na kazetě. To lze jenom tak, že na totéž místo na kazetě uložíme jiný soubor.

*** POZOR ***
Ve jménu souboru nelze použít zástupné znaky ? a *.

9.3 Vypsání pracovní oblasti na tiskárně

Pokud máme k počítači připojenou tiskárnu, můžeme si nechat vypsat obsah pracovní oblasti. V tom případě napíšeme:

SAVE "P:

a všechny procedury z pracovní oblasti se vytisknou.

9.4 Slovníček

V této kapitole jsme se seznámili s novými klíčovými slovy jazyka Logo:

CATALOG	seznam souboru
ERF	zruš soubor na disketě
LOAD	načti do pracovní oblasti
SAVE	ulož pracovní oblast

10. Želva a text

Dokud neoslovíme nějakým příkazem želvu, slouží celá obrazovka jen pro textové zprávy. Jakmile však želvu vyvoláme, rozdělí se obrazovka na dvě podélné části; větší pole pro želvu, menší pro text. Pro texty je vyhrazeno pět řádků v dolní části obrazu.

Příkazem TS můžeme vyhradit pro text opět celou obrazovku. Příkazem SS pak obrazovku opět rozdělíme na pole pro želvu a pětiřádkové pole pro text. Žádný z těchto příkazů neruší původní obsahy polí. Mění pouze jejich viditelnost. Můžeme si to vyzkoušet a zároveň se přesvědčíme, že téhož lze dosáhnout speciálními klávesami. Místo TS můžeme ve stejném významu použít CTRL T, místo SS pak CTRL S.

Podobně můžeme vyhradit celou obrazovku jenom pro želvu, a to příkazem FS. Text viditelný nebude, i když zůstane zachován. Místo FS můžeme se stejným účinkem použít CTRL F.

Příkazy CTRL T, CTRL S, CTRL F můžeme použít i tehdy, když běží nějaká procedura. Zkusme:

CTRL F
CTRL S

Nyní přejdeme mezi textovou a smíšenou, tj. normální obrazovkou:

CTRL T
CTRL S

*** POZOR ***
Textový editor používá pole pro želvu, proto je toto pole vždy po editování vymazáno.

Příkazy CTRL S a CTRL F fungují vždy až poté, co byl použit nějaký příkaz pro želvu. Nefungují tedy ihned po opuštění editoru nebo zapnutí počítače.

10.1 Slovníček

V této kapitole jsme se seznámili s těmito klíčovými slovy:

FS	celá obrazovka pro želvu
SS	část pro želvu, část pro text
TS	celá obrazovka pro text

Též jsme použili další speciální klávesy v podobném významu:

CTRL P	celá obrazovka pro želvu
CTRL S	část pro želvu, část pro text
CTRL T	celá obrazovka pro text

11. Kreslící pero a barvy

Když se želva pohybuje, například na základě našich příkazů FD nebo BK, zanechává za sebou viditelnou stopu. Je to tím, že je vybavena kreslícím perem a jím píše. Želva se může pohybovat i bez psaní. Musíme jí ale sdělit, že má kreslící pero zvednout. Změnou barvy pera můžeme také změnit barvu kreslené stopy. V této kapitole si vysvětlíme, jak lze pera použít.

11.1 Příkazy pro kreslící pero

Ke zvednutí kreslícího pera slouží příkaz PENUP (nebo jen PU). Kreslení se obnoví příkazem PENDOWN (PD).

Zkusme si několik pokusů s těmito příkazy a pozorujme, jak pracují:

```
FD 15
PENUP
FD 15
PENDOWN
FD 20
```

Použitím příkazů zvednutí (PU) nebo spuštění (PD) pera měníme stav, ve kterém se pero nachází. Stav pera mění také příkazy PE a PX.

Příkazem PE se pero změní v mazací gumu. Sleduje-li želva stopu, kterou předtím nakreslila, pak ji vymaže. Vyzkoušíme si to na příkladě:

```
CS
PENDOWN
CTVEREC
```

a nyní:

```
PE
CTVEREC
```

Co želva nejdříve nakreslila, to pak vymazala. Všimneme si, že přitom neudělala žádnou novou čáru.

*** POZOR ***
Pokud Logo napíše:

I DON'T KNOW HOW TO CTVEREC

pak si ověříme, zda byla procedura CTVEREC skutečně definována. Zřejmě není v naší pracovní oblasti.

Příkaz PX spojuje příkazy PD a PE. Jeho použití způsobí, že želva kreslí pouze, když prochází přes prázdné pozadí. Jestliže prochází přes již dříve nakreslenou čáru, vymaže tu její část, přes kterou přejde. Lze tím docílit některých pěkných efektů. Například:

PX
CTVEREC
CTVEREC
CTVEREC

Příkaz PENDOWN (PD) vrátí pero do původního stavu pro kreslení. Tedy:

PENDOWN

11.2 Použití barev

Nyní popíšeme možnosti, které nám nabízí počítač Atari v oblasti barevné grafiky. Počítač může zobrazovat ve 128 barvách, přesněji řečeno v 16 barvách po 8 odstínech. Barvy jsou číselně kódovány takto:

0 - 7	šedá
8 - 15	žlutá
16 - 23	oranžová
24 - 31	červená
32 - 39	růžová
40 - 47	purpurová
48 - 55	fialová
56 - 63	modrá
64 - 71	modrá
72 - 79	světle modrá
80 - 87	tyrkysová
88 - 95	azurová
96 -103	zelená
104-111	žlutozelená
112-119	hnědá
120-127	zlatá

*** POZOR ***
Jak barvy ve skutečnosti vypadají záleží též na TV přijímači a jeho nastavení.

V uvedeném číselném rozpětí pro každou barvu nižší číslo znamená tmavší odstín a vyšší světlejší. Například:

0 = černá
7 = bílá

Existují celkem tři typy změn barev. Můžeme změnit barvu pozadí, barvu kreslícího pera želvy a konečně i barvu želvy samé. O poslední možnosti si řekneme více v kapitole 20.

11.2.1 Změna barvy pozadí

ATARI Logo začíná s modrou barvou pozadí (74). Tuto barvu můžeme změnit příkazem SETBG, přičemž je nutno zadat číslo barvy jako vstup příkazu. Zkusíme to:

```
SETBG 1
SETBG 40
SETBG 120
```

Vytvoříme si proceduru, která bude cyklicky měnit barvy pozadí. Aby se barvy neměnily příliš rychle, použijeme příkazu WAIT. Tento příkaz pozastaví běh procedury do vykonání následujícího příkazu po dobu, kterou určuje zadané číslo jako vstup. Příkaz WAIT 50 bude čekat právě jednu sekundu. Proceduru pojmenujeme např. ZMENA:

```
TO ZMENA
SETBG 0 WAIT 20
SETBG 35 WAIT 20
SETBG 48 WAIT 20
SETBG 60 WAIT 20
SETBG 98 WAIT 20
SETBG 126 WAIT 20
END
```

Proceduru ZMENA necháme proběhnou vícekrát, napíšeme proto:

```
REPEAT 4 [ZMENA]
```

Jaké je momentální číslo barvy pozadí se můžeme snadno dozvědět. Je totiž obsaženo v proměnné BG, kterou si můžeme nechat kdykoliv vytisknout:

```
PRINT BG
```

a Logo nám odpoví:

```
126
```

Nyní nastavíme původní barvu pozadí:

```
SETBG 74
PRINT BG

74
```

11.2.2 Změna barvy kreslícího pera

Želva může ke kreslení použít vždy jedno ze tří různých per. Pera jsou očíslována 0, 1 a 2. Logo začíná vždy s perem číslo 0 a jeho barva je žlutá (číslo barvy 15). Pokud jsme barvu pera nezměnili, následující příklad nakreslí čtverec žlutě:

```
CS
REPEAT 4 [FD 30 RT 90]
```

Číslo pera, které želva právě používá, je uloženo v proměnné označené PN. Vytiskneme ji:

```
PRINT PN

0
```

Pero lze změnit příkazem SETPN. Pomocí třech různých per může želva kreslit třemi různými barvami. Pero číslo 1 kreslí purpurově (číslo barvy 47) a pero číslo 2 kreslí červeně (číslo barvy 121). Použijeme pero 1 a 2 k nakreslení purpurového a červeného čtverce:

```
SETPN 1
REPEAT 4 [FD 30 RT 90]
LT 180
SETPN 2
REPEAT 4 [FD 30 RT 90]
```

*** POZOR ***
Může se stát, že změníme-li barvu pozadí, nebude barva pera přesně odpovídat barvě uvedené v tabulce barev. To záleží i na nastavení televizního přijímače.

Na barvu pera se můžeme též zeptat. Je totiž uložena v proměnné PC:

```
PR PC 0

15

PR PC 1

47

PR PC 2

121
```

ATARI Logo dovoluje také změnit barvu kresby želvy. Docílí se toho příkazem SETPC. Příkaz vyžaduje dva vstupy. Prvním vstupem je číslo pera, jehož barvu budeme měnit, druhým vstupem je pak číslo nové barvy. Následujícím příkazem:

```
SETPC 0 40
```

změníme barvu prvního čtverce (předtím nakresleného žlutě) na purpurovou. Druhý čtverec si ponechá barvu původní. Je to proto, že byl nakreslen jiným perem. Nadále bude pero číslo 0 kreslit purpurově. Vyzkoušíme si to:

```
SETPN 0
FD 50
```

Na obrazovce se nakreslí purpurová čára.

11.3 Slovníček

Naučili jsme se nové příkazy:

PE	pero jako guma
PENDOWN (PD)	spusť pero
PENUP (PU)	zvedni pero
PX	invertující pero
SETBG	nastav barvu pozadí
SETPC	nastav barvu pera
SETPN	nastav číslo pera
WAIT	počkej

Seznámili jsme se také s těmito funkcemi:

BG	barva pozadí
PC	barva pera
PN	číslo pera

12. Další možnosti editace

12.1 Editor Atari Logo

Pomocí editoru můžeme kdykoliv změnit již dříve definované procedury nebo proceduru právě definovanou. Důvodem změny může být třeba to, že procedura nepracuje tak, jak má.

Pokusme se například definovat proceduru KOSOCTVEREC, ale nepřesně:

```
TO KOSOCTVEREC
CTVEREC
RT 45
END
```

Když proceduru zkusíme, nakreslí místo kosočtverce jenom čtverec. Někde je chyba: příkaz RT 45 by měl být před příkazem, který kreslí čtverec. Abychom chybu odstranili, musíme proceduru znovu editovat:

```
EDIT "KOSOCTVEREC
```

Nyní se na obrazovce vypíše celý text naší procedury:

```
TO KOSOCTVEREC
CTVEREC
RT 45
END
```

Kurzor se nachází v levém horním rohu na písmenu T ve slově TO. Musíme ho nejprve přemístit na místo, které chceme změnit. Použijeme k tomu kláves CTRL + a CTRL - .

Editovat začneme tedy tím, že kurzor nastavíme na konec řádku:

```
TO KOSOCTVEREC_
```

Nyní stikneme RETURN a napíšeme:

```
RT 45
```

Dále přesuneme kurzor na předposlední řádek před END a to tak, že dvakrát stiskneme CTRL =:

```
RT 45_
```

Nyní tiskneme DELETE, až řádek vymažeme.

Práci s editorem již můžeme ukončit. Stikneme klávesu ESC a Logo nám sdělí:

```
KOSOCTVEREC DEFINED     (definován)
```

*** POZOR ***
Pokud se během editování rozhodneme, že proceduru opravovat nechceme, stačí stisknout klávesu BREAK. Logo editor opustí a změnu neprovede. Procedura zůstane tak, jak byla před editací.

12.2 Přehled příkazů editoru

K ovládání editoru používáme následující řídící klávesy (příkazy):

CTRL =	kurzor na následující řádek
CTRL -	kurzor na předcházející řádek
CTRL +	kurzor o jedno místo doleva
CTRL *	kurzor o jedno místo doprava
CTRL A	kurzor na začátek řádku
CTRL E	kurzor na konec řádku
RETURN	kurzor a následující text na začátek dalšího řádku
DELETE	výmaz znaku nalevo
SHIFT DELETE	výmaz zbytku řádku

*** POZOR ***
Příkazy pro pohyb kurzoru se uplatní pouze tehdy, je-li na obrazovce text.

12.3 Editování mimo editor

Většinu editačních příkazů můžeme používat i mimo editor. Platí však právě vždy jen pro jeden řádek. Zde stojí za zmínku příkaz CTRL Y, kterým poslední řádek můžeme zkopírovat a pak případně dál opravovat.

13. Pracovní oblast

Při práci s jazykem Logo vytváříme nová klíčová slova procedury. Logo je ukládá do části operační paměti, kterou nazýváme pracovní oblast. Je několik způsobů, jak lze pracovní oblast prohlédnout. Můžeme si například vytisknout jména všech našich procedur, jejich definic apod.

13.1 Vytištění procedur

K vytištění jmen všech procedur uložených v pracovní oblasti použijeme příkazu POTS:

```
POTS
```

a Logo odpoví:

```
TO KOSOCTVEREC
TO CTVEREC
```

atd.

K vytištění definic procedur uložených v pracovní oblasti slouží příkaz POPS:

```
POPS
```

a Logo odpoví:

```
TO KOSOCTVEREC
RT 45
CTVEREC
END

TO CTVEREC
REPEAT 4 [FD 30 RT 90]
END
```

atd.

K vytištění pouze určité procedury slouží příkaz PO:

```
PO "CTVEREC.HVEZDA
```

a Logo odpoví:

```
TO CTVEREC.HVEZDA
REPEAT 8 [CTVEREC RT 45]
END
```

Příkazem PO lze vytisknout též více procedur podle seznamu. Například:

```
PO [CTVEREC.HVEZDA KOSOCTVEREC]
```

a bude následovat výpis uvedených procedur. Přitom nezapomeňme vyhradit celou obrazovku pro text příkazem TS nebo CTRL T.

13.2 Vymazání z pracovní oblasti

Některé procedury zpravidla potřebujeme z pracovní oblasti vymazat. Jsou to například ty, které již nebudeme používat nebo nechceme uložit na disketu nebo kazetu. Vždy se ale ještě jednou přesvědčíme, zda vymazáváme skutečně jenom to, co chceme.

Pro tyto účely je Logo vybaveno několika příkazy. Nejpoužívanější je ERASE (ER), za kterým následuje jméno procedury:

```
ERASE "KOSOCTVEREC
```

Uvedená procedura je z pracovní oblasti vymazána. Jmen můžeme uvést i více:

```
ERASE [CTVEREC CTVEREC.HVEZDA]
```

a uvedené procedury jsou vymazány.

Celou pracovní oblast uvolníme, tedy všechny procedury vymažeme příkazem:

```
ERPS
```

13.3 Slovníček

Seznámili jsme se s novými příkazy Loga:

POTS	vytiskni jména procedur
POPS	vytiskni definice všech procedur
PO	vytiskni definice uvedených procedur
ERASE (ER)	vymaž uvedené procedury
ERPS	vymaž všechny procedury

14. Kreslíme pavouka

Představme si pavouka se čtyřma nohama na každé straně. Jednu nohu tvoří dvě čáry, které svírají úhel 90 stupňů. Nejdříve tedy zkusíme jednu pravou nohu:

```
TO PRAVA.NOHA
FD 30
RT 90
FD 30
END
```

a hned ji vyzkoušíme:

```
PRAVA.NOHA
```

Jenže želva skončila tam, kde kreslení další nohy nemůže začít. Teď by totiž naše procedura nakreslila spíše schody než nohy.

Abychom měli vždy jistotu, kde se želva po skončení procedury nachází, měla by správně každá procedura vrátit želvu tam, kde byla, když procedura začala. Pomocí editoru proceduru upravíme:

```
EDIT "PRAVA.NOHA
```

Editor nám proceduru vypíše a my ji doplníme o tři nové příkazy takto:

```
TO PRAVA.NOHA
FD 30
RT 90
FD 30
BK 30
LT 90
BK 30
END
```

a můžeme proceduru vyzkoušet:

```
CS
PRAVA.NOHA
```

Kresba je stejná, želva je však zpět ve výchozí poloze. Pokračujme v návrhu procedury, která by nakreslila všechny čtyři nohy pravé strany:

```
TO PRAVA.STRANA
RT 90
REPEAT 4 [PRAVA.NOHA LT 20]
LT 10
END
```

Posledním příkazem v proceduře dosáhneme toho, že želva bude mít stejnou polohu i orientaci jako na začátku. Podle dobré zásady: "Zánech želvu, jak jsi ji našel".

Podobně to bude i s levou nohou:

```
TO LEVA.NOHA
FD 30
LT 90
FD 30
BK 30
```

```
RT 90
BK 30
END
```

Vyzkoušíme ji a použijeme ji v proceduře pro levou stranu:

```
TO LEVA.STRANA
LT 90
REPEAT 4 [LEVA.NOHA RT 20]
RT 10
END
```

Konečně napíšeme proceduru pro celého pavouka:

```
TO PAVOUK
LEVA.STRANA
PRAVA.STRANA
FD 10 BK 10
HT
END
```

15. Kreslíme trojúhelníky

Želva může samozřejmě kreslit i různé trojúhelníky. Nejprve si ale zkusíme trojúhelník rovnostranný. Jeho strany, podobně jako u čtverce, jsou stejně dlouhé. Zvolíme i v tomto případě délku strany 30 kroků:

```
FD 30
```

První strana je hotová a teď nastává velké rozhodování. O kolik stupňů se má želva otočit, aby takový trojúhelník správně nakreslila? Ve škole jsme se učili, že rovnostranný trojúhelník má velikost úhlu 60 stupňů. Podívejme se proto, co se stane, když se želva o 60 stupňů otočí:

```
RT 60
FD 30
RT 60
```

Zajímavé, ale trojúhelník to není. Budeme tedy pokračovat, dokud se obrazec neuzavře:

```
FD 30
RT 60
FD 30
RT 60
FD 30
RT 60
FD 30
RT 60
```

Obrázek má šest stran a ne tři. Je to pravidelný šestiúhelník. Aby to byl trojúhelník, musí se želva na každém rohu otočit ne o 60, ale o 120 stupňů. Proč? Odpověď je jednoduchá. Musíme se na věc dívat očima želvy. Na svém trojúhelníkovém

výletě se musí naše želva otočit celkem o 360 stupňů, aby se vrátila zpět na začátek. Neboť se otáčí v každém vrcholu, tedy celkem třikrát, musí se vždy otočit o 360 / 3 = 120 stupňů. Tedy:

```
CS
FD 30
RT 120
FD 30
RT 120
FD 30
RT 120
```

a je to trojúhelník.

Víme už, jak definovat proceduru TROJUHELNIK. Použijeme k tomu editoru:

```
EDIT "TROJUHELNIK
REPEAT 3 [FD 30 RT 120]
END
```

Chvíli si s procedurou pohrajeme:

```
REPEAT 3 [TROJUHELNIK RT 120]
REPEAT 6 [TROJUHELNIK RT 60]
REPEAT 100 [TROJUHELNIK RT 30]
```

V posledním případě želva svoji dráhu opakuje mnohokrát. Můžeme ji zastavit klávesou BREAK.

Počet opakování v příkazu REPEAT lze také vypočítat. Otočí-li se želva například vždy o 30 stupňů, pak počet různých opakování je 12 (360 / 30 = 12). Logo aritmetické operace umí, takže je může spočítat za nás:

```
REPEAT 360 / 30 [TROJUHELNIK RT 30]
```

Využijeme proceduru TROJUHELNIK a nakreslíme třeba stan:

```
CS
TROJUHELNIK
```

Jenže stan musí být otočený základnou dolů. Želvu tedy otočíme:

```
CS
RT 90
TROJUHELNIK
```

Teď je zase stan vzhůru nohama. Nezapomeňme, že vnitřní úhly v našem trojúhelníku jsou 60 stupňů. Jestliže tedy otočíme želvu o 90 stupňů a pak o 60, nemůže být správně nastavena. Proto ji musíme natočit právě jen o 30 stupňů:

```
TO STAN
RT 30
TROJUHELNIK
END
```

Nakreslíme pomocí této procedury třeba strom:

```
TO STROM
STAN
RT 60
FD 15
RT 90
FD 15
RT 180
END
```

Stromů můžeme na obrazovce nakreslit i více:

```
CS
STROM
RT 90
PU
FD 30
LT 90
PD
STROM
```

Dobrým programátorským zvykem je používat zvláštních procedur, kterými se nastavují počáteční stavy. Takovou proceduru, která připraví želvu k nakreslení nového stromu, nazveme NASTAV.STROM:

```
TO NASTAV.STROM
RT 90
PU FD 30
LT 90 PD
END
```

Procedury můžeme samozřejmě používat opakovaně:

```
REPEAT 3 [STROM NASTAV.STROM]
```

Pokud bychom chtěli změnit vzdálenosti mezi stromy, pak editorem opravíme v proceduře NASTAV.STROM hodnotu u příkazu FD.

Nakonec využijeme toho, že želva umí nakreslit jak čtverec, tak trojúhelník, a nakreslíme dům:

```
CS
CTVEREC
FD 30
TROJUHELNIK
```

To ale není dům. Je tu chyba a nebude ji těžké opravit. Střechu nakreslíme pomocí procedury STAN:

```
CS
CTVEREC
FD 30
STAN
```

Dům je nyní skoro k nerozeznání od skutečného.

16. Proměnné

16.1 Zavedení proměnných

Chceme-li nakreslit čtverce o různých stranách, například 60, 50, 100, 10 apod., můžeme vytvořit pro každý takový čtverec vlastní proceduru: CTVEREC60, CTVEREC50, CTVEREC100,... Jednodušší by však bylo, kdybychom vytvořili proceduru jednu, a té zadávali velikost strany jako vstupní hodnotu obdobně jako příkazům FORWARD, LEFT, RIGHT a BACK, tedy takto:

```
CTVEREC 60
CTVEREC 50
CTVEREC 100
CTVEREC 10
```

Proceduru, která dokáže kreslit různě velké čtverce, nazveme například KRABICEP, aby nám její název připomínal, co bude kreslit (P, neboť kreslí čtverce doprava).

Nejrychlejší bude, když novou proceduru nebudeme psát od začátku, ale editováním opravíme vhodnou proceduru již napsanou, a to CTVEREC:

```
EDIT "CTVEREC
```

Editor proceduru vypíše, kurzor bude opět na prvním znaku. Změníme nejprve jméno procedury. Přesuneme pomocí CTRL E kurzor na první mezeru za slovem CTVEREC a funkcí DELETE slovo vymažeme. Místo něj napíšeme KRABICEP:

```
TO KRABICEP
```

Stiskneme ESC a Logo odpoví:

```
KRABICEP DEFINED
```

Původní procedura CTVEREC stále existuje. V této chvíli jsou definovány dvě stejné procedury, a to CTVEREC a KRABICEP. Nyní začneme druhou proceduru měnit:

```
EDIT "KRABICEP
```

Editor proceduru vypíše:

```
TO KRABICEP
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
FD 30
RT 90
END
```

Chceme-li měnit délku strany čtverce, musíme změnit vstupy všech příkazů FD (FORWARD). Ale jak to provést? Vyřešíme to tak, že místo čísla 30 uvedeme na vstupu nějaké jméno, například STRANA. Tomu pak přiřadíme hodnotu. Nyní již můžeme psát příkaz takto:

```
FD :STRANA
```

Dvojtečka, která bezprostředně jménu předchází, jej odlišuje od příkazu. Nyní již zapíšeme proceduru KRABICEP se vstupním parametrem STRANA:

```
TO KRABICEP :STRANA
FD :STRANA
RT 90
FD :STRANA
RT 90
FD :STRANA
RT 90
FD :STRANA
RT 90
END
```

Takto upravené proceduře můžeme zadávat různé délky stran:

```
KRABICEP 60
KRABICEP 50
KRABICEP 100
```

Právě jsme si ukázali význam důležité myšlenky využívané hlavně v matematice, a to používání jmen místo čísel. Obdobně jako v matematice jim i v Logu budeme říkat proměnné. Místo kouzelného X pro proměnnou, jak to často dělá školní algebra, jsme zvolili smysluplnější STRANA, protože je patrnější, co znamená.

Při definování této procedury jsme chtěli, aby želva nakreslila čtverec, ale nevěděli jsme ještě, jak bude velký. Prostě si přejeme čtverce různých velikostí. Když napíšeme příkaz FORWARD (FD), víme, že musí mít též vstup. Nemůžeme napsat FD bez vstupu. Musíme napsat FD a ještě něco. To něco jsme pojmenovali jako STRANA. V jazyce Logo to znamená: vždy musí být něco v košíku, který se jmenuje STRANA. Začne-li Logo příkaz FORWARD :STRANA provádět, už musí košík něco obsahovat.

Košík naplníme, když při volání procedury napíšeme:

```
KRABICEP 10
```

nebo

```
KRABICEP 25
```

Jakmile Logo příkaz přijme, naplní žádanou vstupní hodnotou košík, který se jmenuje STRANA. Procedura sama však může použít obsah košíku i někdy později.

*** POZOR ***
Možné chyby:
-místo STRANA jsme napsali STANA, nebo něco jiného,
-zapomněli jsme na dvojtečku před jménem proměnné,
do procedury KRABICEP jsme vložili nesprávný příkaz,
-podobně jsme mohli nějaký příkaz náhodou vymazat,
-mezi dvojtečkou a proměnnou jsme napsali mezeru, nebo jsme umístili dvojtečku před číslem,
-před dvojtečkou jsme zapomněli jednu mezeru.

Dvojtečka před slovem v jazyku Logo, jak jsme se již zmínili, říká, že se jedná o jméno košíku neboli proměnné. Proměnná může obsahovat číslo, jiné slovo, seznam slov nebo dokonce seznam seznamů.

Příklady použití procedury KRABICEP:

```
TO CTVERCE
KRABICEP 10
KRABICEP 20
KRABICEP 30
KRABICEP 40
END

TO KOSOCTVERCE
RT 45
REPEAT 4 [CTVERCE RT 90]
END

TO VLAJKAP :STRANA
PD :STRANA
KRABICEP :STRANA
BK :STRANA
END

TO 6VLAJEK :STRANA
REPEAT 6 [VLAJKAP :STRANA RT 60]
END
```

Vidíme, že jméno může obsahovat i začínat číslicí.

```
TO VLAJKOLO
6VLAJEK :STRANA
6VLAJEK :STRANA-20
END
```

Vidíme, že vstupem (hodnotou proměnné) může být i aritmetický výraz.

Procedury vyzkoušíme:

```
VLAJKA 30

6VLAJEK 30

VLAJKOLO 30
```

Tím, že se procedury mohou modifikovat pomocí vstupu, a tak nastavovat tvary obrázků, stávají se užitečnějšími i zajímavějšími.

16.2 Velké a malé trojúhelníky

Podobně můžeme upravit i procedury pro kreslení trojúhelníku, a to tak, aby měly proměnné vstupy:

```
ED "TROJUHELNIK
```

Editor proceduru vypíše:

```
TO TROJUHELNIK
REPEAT 3 [FD 30 RT 120]
END
```

Proceduru upravíme a přejmenujeme:

```
TO TROJUHELNIKP :STRANA
REPEAT 3 [FD :STRANA RT 120]
END
```

a proceduru zkusíme v řadě aplikací jako:

```
TO TROJUHELNIKY
TROJUHELNIKP 10
TROJUHELNIKP 20
TROJUHELNIKP 30
TROJUHELNIKP 40
END

TO TROJ.HVEZDY
REPEAT 10 [TROJUHELNIKY RT 36]
END

TO SMRK :STRANA
RT 30 TROJUHELNIKP :STRANA
RT 60 FD :STRANA/2
LT 90 BK :STRANA/2
END

TO LES
SMRK 30
SMRK 40
SMRK 50
END
```

16.3 Aritmetika

Jak je patrno z některých předcházejících příkladů, Logo umí i počítat. Vyzkoušíme si to:

```
PR 5+3
```

Logo odpoví:

```
8
```

nebo:

```
PR 345-32
```

```
313

PR 4*23

92

PR 25/5

5
```

Atari Logo umí počítat i s čísly v pohyblivé desetinné čárce. Zde jsou další příklady:

```
PR 25/6

4.16666666

PR 4*2.3

9.2

PR 25/2

12.5
```

16.4 Slovníček

V této kapitole jsme se seznámili s následujícími klíčovými znaky:

:	příznak proměnné
+	operátor sčítání
-	operátor odčítání
*	operátor násobení
/	operátor dělení

17. Kruhy a oblouky

17.1 Želva kreslí kruh

Želva nemusí kreslit jenom rovné čáry, ale také křivky. Křivky kreslí po malých krůčcích. Po každém krůčku se pootočí.

Pokud má želva nakreslit celý kruh, musí se otočit celkem o 360 stupňů. Zkusme to:

```
REPEAT 360 [FD 1 RT 1]
```

Kruh nevypadá špatně, ale kreslí se příliš dlouho. To proto, že želva opakuje příkazy v závorce 360 krát, tedy tolikrát, jako kdyby nakreslila 90 čtverců!

Při menším kompromisu můžeme nakreslit kruh rychleji:

```
REPEAT 36 [FD 10 RT 10]
```

Kruh sice není tak dokonalý, je to vlastně 36-ti úhelník, ale želva ho nakreslí 10 krát rychleji než kruh předešlý.

Proč jsme také vstup u příkazu FD změnili na 10? Následující procedura nám to ukáže. Bude umět totiž kreslit kruhy různých velikostí:

```
TO KRUH :KROK
REPEAT 36 [FD :KROK RT 10]
END
```

Vyzkoušíme ji pro různé vstupy:

```
KRUH 1
KRUH 5
KRUH 10
```

Skutečně, velikost kruhu závisí na hodnotě zadaného vstupu. Není to nic divného, protože velikost kroku je příkazu FD předána v proměnné KROK. A tento krok určuje velikost obvodu kruhu.

17.2 Kruh s poloměrem

Často je výhodnější zadávat velikost kruhu jeho poloměrem, tedy vzdáleností od středu ke kružnici. Abychom však nemuseli vypočítávat poloměr při každém volání procedury KRUH, napíšeme si takovou proceduru, která si ho spočítá sama. Nazveme ji KRUHR a použijeme v ní naši proceduru KRUH:

```
TO KRUHR :POLOMER
KRUH 2*3.14*:POLOMER/36
END
```

Povědomý výraz 2*3.14*:POLOMER je vlastně 2.pí.r a vypočítá obvod kruhu. Vlastní obvod se kreslí pomocí 36 příkazů FD. Abychom dostali velikost kroku, musíme tedy obvod vydělit číslem 36.

Proceduru vyzkoušíme:

```
KRUHR 30
RT 90 FD 30
FD 30 HT
```

Pomocí této procedury můžeme nakreslit květ, terč nebo hlavu panáka.

17.3 Želva kreslí oblouky

Často potřebujeme nakreslit jenom část kružnice, oblouk. Můžeme toho dosáhnout poměrně jednoduše. Vyvoláme naši proceduru KRUH a klávesou BREAK kresbu přerušíme.

S tímto způsobem je však spojena jedna nesnáz. Abychom dosáhli pokaždé požadované velikosti oblouku, musíme být mimořádně šikovní. Lepší proto bude, když vybavíme proceduru dalším vstupem, který bude velikost oblouku určovat. Tento vstup bude udávat, kolik má želva udělat kroků a pootočení.

Vyjdeme z procedury KRUH a novou proceduru nazveme OBLOUK:

```
EDIT "KRUH
```

Editor proceduru vypíše:

```
TO KRUH :KROK
REPEAT 36 [FD :KROK RT 10]
END
```

a my ji upravíme takto:

```
TO OBLOUK :KROK :KOLIKRAT
REPEAT :KOLIKRAT [FD :KROK RT 10]
END
```

Nyní ji můžeme vyzkoušet:

```
OBLOUK 10 36
OBLOUK 10 18
OBLOUK 10 9
```

Šikovnější ale bude zadávat velikost oblouku ve stupních. Proto proceduru OBLOUK ještě editorem upravíme do této podoby:

```
TO OBLOUK :KROK :STUPNE
REPEAT :STUPNE/10 [FD 10 :KROK RT 10]
END
```

17.4 Použití oblouku

Uvedenou proceduru OBLOUK můžeme využít ke kresbám různých obrazců a ornamentů. Nakreslíme třeba smyčku:

```
OBLOUK 6 120
RT 120
OBLOUK 6 120
```

nebo rybu:

```
OBLOUK 6 90
RT 90
OBLOUK 6 90
RT 90
```

Nyní napíšeme proceduru, která umí kreslit květy. Použijeme k tomu ještě proceduru OBLOUKP, která je uvedena v kapitole 50 (Oblouky a kružnice).

```
TO KVET :VELIKOST
OBLOUKP :VELIKOST 90 RT 90
```

```
OBLOUKP :VELIKOST 90 RT 90
END
```

A nyní ji zkusíme:

```
KVET 40 KVET 30
REPEAT 8 [KVET 40 KVET 30 RT 45]
REPEAT 4 [KVET 30 RT 45 KVET 40 RT 45]
```

*** POZOR ***
Pokud se příkazy nevejdou na jeden řádek, pokračuje Logo automaticky na řádku následujícím a na konci prvého vypíše šipku.

Pokusíme se nakreslit něco složitějšího, třeba labuť. Budeme k tomu potřebovat opět několik procedur, které jsou uvedeny v kapitole 50. Jedná se o procedury OBLOUKP a OBLOUKL. Ty pak dále používají procedur OBLOUK.PRAVY, OBLOUK.LEVY, OBLOUKL1 a OBLOUKP1. Použijeme také naši proceduru KVET:

```
TO TELO :VELIKOST
RT 45
KVET :VELIKOST
LT 45
END

TO KRK :VELIKOST
LT 45
OBLOUKP :VELIKOST 90
OBLOUKL :VELIKOST 90
RT 45
END

TO HLAVA :VELIKOST
LT 135
KVET :VELIKOST
RT 135
END
```

Vlastní procedura, která nakreslí labuť, by pak mohla vypadat třeba takto:

```
TO LABUT :VELIKOST
TELO :VELIKOST
KRK :VELIKOST/2
HLAVA :VELIKOST/4
END
```

Vyzkoušíme ji:

```
LABUT 50
```

18. Pole želvy

18.1 Orientace želvy

Želva, jak jsme se již zmínili, má svoji polohu a orientaci. Orientace želvy je dána podobně jako u kompasu ve stupních. Sever je 0 stupňů, je to horní okraj obrazovky. Východ je 90 stupňů, jih 180, západ pak 270 stupňů:

```
                sever
                  0

      západ                  východ
       270                     90

                 jih
                 180
```

18.1.1 Funkce HEADING

Želva má na začátku vždy orientaci 0 stupňů. Stejnou orientaci nastaví i příkaz CS. Momentální orientaci želvy poskytuje funkce (operace) HEADING:

```
CS
RT 90
PR HEADING
```

Logo odpoví:

```
90
```

Funkce HEADING (orientace) nám ukáže směr, do kterého je želva právě nastavena. HEADING patří mezi klíčová slova jazyka Logo. Liší se však třeba od slov jako PRINT nebo FORWARD. Není to totiž příkaz, ale funkce. Na rozdíl od příkazu je výsledkem funkce nějaká hodnota, číselná nebo jiná. V této kapitole se zmíníme ještě i o jiných funkcích.

18.1.2 Příkaz SETH

Do určité orientace můžeme želvu nastavit příkazem SETH. Příkaz se od nám známých příkazů RT nebo LT liší tím, že výsledná orientace želvy nezáleží na jejím předcházejícím natočení:

```
SETH 90
RT 90
SETH 90
```

18.2 Poloha želvy

18.2.1 Funkce POS

Poloha želvy na obrazovce je určena dvěma čísly, která udávají její vzdálenost od středu obrazovky. Hodnoty polohy nám poskytuje funkce POS. Na začátku je želva ve středu obrazovky, hodnota POS je dvojice 0 0. První číslo je x-ová souřadnice, tedy vzdálenost od středu na vodorovné ose. Je-li želva od středu západně, pak je toto číslo záporné. Druhé číslo pak udává vzdálenost želvy od středu na svislé ose, je to y-ová souřadnice. Je-li želva od středu jižně, pak je toto číslo záporné, tj. má znaménko mínus (-).

18.2.2 Funkce XCOR a YCOR

Každý bod obrazovky, po které se želva pohybuje, je tedy určen dvěma souřadnicemi. X-ová souřadnice (XCOR) běží vodorovně, y-ová souřadnice (YCOR) svisle. Ve středu obrazovky je jak XCOR, tak YCOR rovno nule. Klíčová slova XCOR a YCOR jsou tedy dalšími funkcemi jazyka Logo, se kterými se seznamujeme. Mezní hodnoty souřadnic jsou dány takto (verze v cartridgi):

```
                120

     -158         0          161

               -119
```

*** POZOR ***
Mezní hodnoty souřadnic se však mohou u jednotlivých verzí jazyka Logo poněkud lišit. Mění je také nastavení příkazem .SETSCR.

Například provedeme příkazy:

```
CS
LT 90
FD 30
PR POS
```

a Logo nám odpoví:

```
-30 0
```

což znamená, že želva je 30 kroků západně od středu podél vodorovné osy. A dále:

```
BK 60
PR POS
```

Logo odpoví:

30 0

Nyní je želva 30 kroků východně od středu opět podél vodorovné osy. Na každou souřadnici se můžeme ptát i jednotlivě:

PR XCOR

30

PR YCOR

0

Budeme dále pokračovat v pohybu:

RT 90
FD 52
PR XCOR
PR YCOR

Nyní bude odpověď:

30
52

Želva se nyní nachází 30 kroků východně a 52 kroků severně vzhledem ke středu obrazovky.

*** POZOR ***
Mějme na paměti, že příkaz CS vrací vždy želvu do výchozí polohy na 0 0. Pokud jej tedy během našeho zkoušení použijeme, mohou být vypisované hodnoty poloh jiné.

18.2.3 Příkaz SETPOS

Přímo na určité místo můžeme želvu umístit příkazem SETPOS. Příkaz se odlišuje od příkazu FD nebo BK v tom, že výsledný efekt na předchozí poloze želvy nezávisí. Též nemění orientaci želvy:

SETPOS [50 -52]

Želva se přesunula na novou pozici. Prvním vstupem je x-ová souřadnice, druhým y-ová souřadnice.

18.2.4 Příkazy WRAP a WINDOW

Režimy WRAP a WINDOW vymezují akční prostor želvy na obrazovce. Volbou některého z těchto režimů je určeno chování želvy, pokud opustí stanovený rozsah obrazovky (například po příkazu FD 400).

Normálně je želva nastavena do režimu WRAP, což znamená, že kdykoliv by měla překročit jednu stranu obrazovky, objeví se na druhé, protější straně. Směr jejího pohybu zůstane zachován:

```
CS
FD 400
PR POS
```

Logo odpoví:

```
0 16
```

Želva tedy není 400 kroků od středu, ale jenom 16.

Příkaz WINDOW naopak způsobí, že želva klidně opustí obrazovku, neobjeví se na protější straně, jak tomu bylo u režimu WRAP. Tím se nám může stát želva neviditelnou, ač vše dělá podle nás dál. Souřadnice v tomto případě mohou být obrovské:

```
WINDOW
FD 400
PR POS
```

a Logo odpoví:

```
0 400
```

Želva je skutečně 400 kroků od středu a tím je z dohledu. Příkaz CS nám ji vrátí zpět do středu, ať je kdekoliv.

Zpět do režimu WRAP se vrátíme tak, že napíšeme příkaz WRAP. Po napsání příkazů WRAP nebo WINDOW se obrazovka vždy vymaže a želva se uloží do výchozí polohy ve středu obrazovky.

18.3 Kreslíme pomocí příkazu SETPOS

Existuje poměrně jednoduchý způsob, jak nakreslíme pravoúhlý trojúhelník, známe-li délky odvěsen. Nejprve si někde poznamenáme výchozí polohu želvy. Použijeme k tomu nového příkazu MAKE:

```
CS
MAKE "START POS
```

Příkaz MAKE udělá dvě věci. Uloží výstup z POS do pracovní oblasti a dá mu jméno START. Vznikne tak nová proměnná (košík) se jménem START. Pokud se tedy nyní zeptáme:

```
PR :START
```

pak nám Logo odpoví:

```
0 0
```

Dvě ramena svírající pravý úhel želva nakreslí snadno:

```
FD 33
RT 90
```

```
FD 42
```

a zpět do výchozí polohy dostaneme želvu příkazem SETPOS se vstupem START takto:

```
SETPOS :START
```

Želva se přesunula do polohy z proměnné START a nakreslila přeponu. Celá procedura, kterou nazveme TRI pak bude vypadat takto:

```
TO TRI :STRANA1 :STRANA2
MAKE "START POS
FD :STRANA1
RT 90
FD :STRANA2
SETPOS :START
END
```

Vyzkoušíme ji:

```
CS
TRI 40 50
SETH 0
TRI 75 20
```

18.4 Slovníček

V této kapitole jsme se seznámili s novými příkazy jazyka Logo:

SETH	nastav orientaci
SETPOS	nastav polohu
WINDOW	režim neohraničený
WRAP	režim ohraničený
MAKE	přiřaď

Dále jsme použili nové funkce:

HEADING	orientace
POS	poloha
XCOR	x-ová souřadnice
YCOR	y-ová souřadnice

19. Mnohoúhelníky a spirály

Podobně, jako jsme měnili v předcházejících procedurách počet kroků, které želva měla vykonat, můžeme měnit i její orientaci (úhel otočení). Plynulou změnou těchto dvou složek můžeme docílit krásných a překvapivých obrazů. Ukážeme si to v následující proceduře, která používá dvou vstupů. Jeden určuje počet kroků a druhý velikost natočení:

```
TO VICE :KROK :UHEL
FD :KROK
RT :UHEL
VICE :KROK :UHEL
END
```

Když proceduru vyvoláme:

```
VICE 30 90
```

tak se želva nezastaví, protože příkazy pohybu se stále opakují. Proceduru VICE a tedy i želvu zastaví pouze klávesa BREAK. Logo pak odpoví:

```
STOPPED! IN VICE
```

Uvedeme některé další příklady použití procedury VICE. Mezi jednotlivými kresbami použijeme příkazu CS:

```
VICE 30 120
VICE 30 60
VICE 30 72
VICE 30 144
VICE 30 40
VICE 30 160
```

Procedura VICE je rekurzivní. To znamená, že volá sebe samu. V jazyce Logo má každá procedura své jméno a je zcela samostatná. Každá procedura může volat jakoukoliv jinou proceduru, dokonce i sama sebe, a právě takové proceduře pak říkáme rekurzivní.

19.1 Kreslení spirál

Želva v proceduře VICE kreslí pouze uzavřené obrazy. Vrací se totiž vždy na místo, odkud začala kreslit. Vyjímku by tvořil případ, kdy by se želva měla otočit při každém kole o 0 nebo 360 stupňů (nebo samozřejmě o násobek 360). Pak by se pohybovala rovně.

Pokud má želva nakreslit například spirálu, nesmí se vrátit do svého výchozího stavu. Při každém kole (kolem rozumíme jeden průchod procedurou) by měla zvýšit počet kroků o které popojde, a tak by se postupně vzdalovala od místa, kde začala.

Dosáhneme toho nejlépe tak, že při každém volání procedury VICE trochu zvětšíme proměnou KROK. Takovou úpravou získáme proceduru, která nakreslí spirálu. Proceduru nazveme SPI a získáme ji například editováním procedury VICE:

```
TO SPI :KROK :UHEL
FD :KROK
RT :UHEL
SPI :KROK + 6 :UHEL
END
```

Proceduru vyzkoušíme s různými vstupy:

```
SPI 5 90
SPI 5 120
SPI 5 60
SPI 5 144
SPI 5 125
SPI 5 160
```

Přírůstek k proměnné KROK jsme v proceduře SPI zvolili napevno jako 6. Lepší by bylo, kdybychom ho též zadávali jako vstup. Proceduru tedy ještě jednou upravíme:

```
TO SPI :KROK :UHEL :ZMENA
FD :KROK
RT :UHEL
SPI :KROK + :ZMENA :UHEL :ZMENA
END
```

a vyvoláme ji:

```
SPI 5 75 1
SPI 5 75 2
```

Ať je želva kdekoliv, můžeme ji zastavit klávesou BREAK.

20. Další možnosti želví grafiky

Atari Logo nabízí poměrně bohaté možnosti želví graficky. Existují například celkem čtyři želvy. Želva, o které jsme dosud hovořili, byla pouze jedna ze čtyř. Všechny želvy mohou kreslit současně. Mohou se však také jenom pohybovat.

V této kapitole si ukážeme, jak lze ovládat každou želvu zvlášť a co všechno želvy dovedou.

20.1 Několik želv

Želvy jsou očíslovány 0, 1, 2 a 3. Které želvy budou vykonávat naše příkazy, určuje příkaz TELL. Zkusme všechny želvy seřadit do řady. Začneme takto:

```
ST
FD 40
```

Na začátku Logo používá želvu číslo 0, a ta vykonává všechny příkazy, dokud neurčíme želvu jinou:

```
TELL 1
```

Ve středu obrazovky se objevila želva číslo 1.

*** POZOR ***
Může se stát, že se želva číslo jedna neobjeví. Je tomu tak proto, že příkaz TELL zobrazí želvu pouze při prvním použití. Pokud se tedy neobjeví, je schována. Objeví se po příkazu ST.

```
FD 20
TELL 2
BK 20
TELL 3
```

Příkazem TELL můžeme určit i více želv najednou, použijeme k tomu hranatých závorek:

```
TELL [0 1 2 3]
RT 90
FD 20
```

Všechny želvy se otočily vpravo a postoupily dopředu.

20.2 Želvy v pohybu

Želvu (nebo želvy) lze uvést do pohybu příkazem SETSP:

```
SETSP 20
```

Příkaz způsobil, že se všechny želvy daly do pohybu rychlostí 20. Větší číslo znamená větší rychlost pohybu, menší číslo menší rychlost. Nula má za následek opětné zastavení.

Želvy za sebou zanechávají stopu, neboť jsou jejich pera nastavena na kreslení. Můžeme však z nich udělat vymazávače (gumy):

```
PE
```

Želvy se nyní pouze pohybují, stále však jedním směrem. Změníme-li jejich orientaci, změníme také směr jejich pohybu.

Zkusíme to u želvy číslo 1:

```
TELL 1
RT 180
```

Vidíme zde jeden podstatný rozdíl od grafiky, s jakou jsme pracovali až dosud. Zatímco se želvy pohybují, může totiž Logo vykonávat další příkazy.

Vrátíme se k výchozímu stavu želv:

```
TELL [0 1 2 3]
CS
```

Všechny želvy se naskládaly jedna na druhou. Schováme je:

```
HT
```

pak vyvoláme želvu 0:

```
TELL 0 ST
```

a spustíme jí kreslící pero:

```
PD
```

Tak jsme docílili opět výchozího stavu.

20.3 Barva želvy

Když jsme si želvy vyvolali, všimli jsme si, že má každá jinou barvu. Barvu želvy můžeme změnit příkazem SETC:

```
TELL 0
SETC 20
```

Barva želvy číslo 0 se změnila na žlutou. Čísla barev jsou jednotná a jsou uvedena v 11. kapitole.

20.4 Tvar želvy

20.4.1 Změna tvaru želvy

Tvar každé želvy můžeme celkem libovolně změnit, třeba na kosmickou loď, na hvězdu nebo na ptáčka. Najednou dokonce můžeme definovat až 15 různých tvarů. Umožňuje nám to editor tvaru.

Editor tvaru vyvoláme příkazem EDSH. Jako vstup použijeme číslo tvaru od 1 do 15. Všechny tvary jsou zprvu prázdné:

```
EDSH 1
```

Na obrazovce se objevila mřížka políček. Jsou uspořádány do 16 řad a 8 sloupců. Všechna políčka jsou prázdná. Kurzorem, který se objevil v levém horním rohu, můžeme pohybovat pomocí kláves CTRL - (nahoru), CTRL = (dolů), CTRL + (doleva), CTRL * (doprava). Samotným pohybem obsah políček neměníme.

Políčko, na kterém se právě nachází kurzor, vyplníme stiskem mezerníku. Nový tvar vytváříme postupným vyplňováním určitých políček v dané mřížce. Je-li kurzor na prázdném políčku, pak jej mezerníkem vyplníme, je-li kurzor na plném políčku, pak mezerníkem políčko vymažeme. Mezerníkem polohu kurzoru neovlivníme. To lze pouze pomocí CTRL s příslušnou šipkou.

*** POZOR ***
Některé verze jazyka Logo používají ke kreslení ve znakovém editoru jiné klávesy:

INSERT (>)	kreslení bodu
CLEAR (<)	mazání bodu

Zkusme si udělat tvar číslo 1 podle naší představy (třeba jako sluníčko). Editor tvaru opustíme podobně jako editor textový klávesou ESC.

Nyní příkazem SETSH vyměníme tvar želvy 0 za nově definovaný tvar:

```
CS
TELL 0
ST
SETSH 1
```

a tvar želvě zase vrátíme:

```
SETSH 0
```

Želva, které jsme dali nový tvar, bude dělat vše jako želva původní s jedním malým rozdílem. Když změníme naší nové "želvě" orientaci, nenatočí se její kresba do příslušného směru jako u želvy původní, i když pohybovat se tímto směrem samozřejmě bude.

20.4.2 Uložení tvaru

Tvar, který jsme nově vytvořili, se uloží hned poté, co opustíme tvarový editor (ESC), do pracovní oblasti. Není tam však, jako koneckonců i vše ostatní, uložen trvale. Pokud počítač vypneme a pak znova zapneme, bude všech 15 tvarů opět prázdných.

Abychom náš nový tvar uchovali, musíme ho nejprve pojmenovat. Slouží k tomu příkaz GETSH. Ukážeme si to na příkladě:

```
MAKE "SLUNCE GETSH 1
```

Nyní se náš nový tvar, který jsme definovali pod číslem 1, bude jmenovat SLUNCE. Způsobil to příkaz MAKE, který přiřadil proměnné SLUNCE tvar číslo 1.

Teď již můžeme nahrát pracovní oblast na kazetu či disketu příkazem SAVE, náš tvar SLUNCE se nahraje automaticky se všemi procedurami.

Jestliže později nahrajeme danou pracovní oblast do počítače příkazem LOAD, můžeme proměnnou SLUNCE opět přiřadit tvaru 1. K tomu slouží příkaz PUTSH, jako vstup vyžaduje číslo příslušného tvaru:

```
PUTSH 1 :SLUNCE
```

Tím je tvar z proměnné SLUNCE uložen zpět do tvaru číslo 1.

Dále již stačí, když napíšeme:

```
TELL 0
SETSH 1
```

a želva číslo 0 na sebe vezme opět podobu sluníčka.

20.5 Zjišťování střetu

Jazyk Logo je vybaven pozoruhodnou možností. Můžeme totiž snadno zjišťovat, kdy se některá želva střetne s kresbou nebo

jinou želvou. Typy možných střetů jsou očíslovány od 0 do 21, a to podle toho, kterých želv nebo kreseb se týkají.

Logo rozlišuje celkem tři druhy kreseb, záleží na tom, kterým perem byly nakresleny. Kresby jsou očíslovány stejně jako příslušná pera, to jest 0, 1 a 2.

Tabulka typu střetu je uvedena v příloze 5. Pomocí těchto kódů a příkazu WHEN pak určujeme typ střetu, který má být testován.

Jak budeme s příkazem WHEN pracovat, si hned ukážeme. Nejdříve však vymažeme obrazovku a vrátíme želvám jejich původní podoby:

```
TELL [0 1 2 3]
CS
SETSH 0
HT
```

Vyvoláme želvu 0 a necháme ji nakreslit vodorovnou čáru:

```
TELL 0 ST
SETPN 0
PD
RT 90
FD 50 BK 100
```

Vyvoláme želvu 1 a rozjedeme ji:

```
TELL 1
ST
PU
SETSP 20
```

Nyní máme připraven střet. Chceme, aby želva 1, kdykoliv se střetne s čárou, změnila směr svého pohybu. Mohli bychom to udělat třeba tak, že bychom vždy, když želva narazí na čáru, napsali příkaz RT 180. Lepší jistě bude, udělá-li to Logo za nás:

```
WHEN 4 [RT 180]
```

Číslo 4 zde symbolizuje střet mezi želvou 1 a kresbou pera 0.

Příkaz WHEN vyvolává takzvané hlídače, které celý svůj čas věnují pozorování příslušného střetu. Jakmile ke střetu dojde, hlídač vydá želvě příkazy podle seznamu. Potom se vrátí zpět k pozorování. Z této činnosti ho vysvobodí buď příkaz CS nebo:

```
WHEN 4 []
```

Během doby, kdy hlídač pozoruje příslušné želvy a jejich střety, můžeme zadávat další příkazy.

20.6 Slovníček

V této kapitole jsme se seznámili s novými příkazy:

EDSH	vyvolej editor tvaru

PUSH	ulož tvar z proměnné do vzoru
SETC	nastav peru barvu
SETSH	dej želvě daný tvar
SETSP	nastav rychlost pohybu
TELL	určí želvu
WHEN	nastav hlídače podmínky

a funkci:

GETSH	obsahuje číslo tvaru

21. Navrhujeme hru

V této kapitole si ukážeme, jak využít pohybu želv ve hře. Jako příklad jsme si vybrali jednoduchou "navigační" hru. Na obrazovce se objeví cíl, několik překážek a želva. Úkolem hráče je dovést želvu kolem překážek až do cíle, a to co nejmenším počtem tahů.

Jako cíl může posloužit nějaká želví kresba nebo to může být i želva sama. My jako cíle použijeme želvu číslo 0. Želvu číslo 1 pak použijeme jako navigátora. Hlídače z příkazu WHEN pak necháme hlídat, kdy se obě želvy střetnou.

Takovou úlohu je snadnější a jednodušší řešit postupně, po krocích. Zpočátku nepoužijeme překážek. Želvu 1 uvedeme do pohybu a hráč se bude snažit pomocí příkazu Logo, například RT 45 nebo LT 30, dovést želvu až do cíle. V další variantě hru vylepšíme tím, že budeme želvu ovládat určitými klávesami nebo ovladačem. Nakonec hru obohatíme o překážky.

21.1 Nastavení hry

Nejprve musíme nastavit cíl a potom želvu. To uděláme jedinou procedurou, kterou pojmenujeme NASTAV. Procedura náhodně umístí želvu někde na obrazovce. Ponechá však želvě základní orientaci, tzn. 0 stupňů:

```
TO NASTAV
PENUP HOME
RT RANDOM 360
FD RANDOM 80
SETSH 0
END
```

Funkce RANDOM poskytne náhodné číslo. Toto číslo je vždy menší než číslo, které je ve funkci RANDOM uvedeno jako vstup.

V proceduře NASTAV se tedy želva po příkazu RT natočí doprava o úhel nejméně 0 a nejvíce 359 stupňů. Konkrétní číslo je po každém volání RANDOM "vypočítáno" znovu.

Vstupem do příkazu FD je též náhodné číslo. Zde však toto číslo bude vždy menší než 79 a současně bude větší nebo rovno nule.

Poslední příkaz v NASTAV orientuje želvu severně.

Proceduru NASTAV použijeme nejen k nastavení želvy a cíle, ale také k nastavení překážek. Následující procedura NASTAV.HRU umístí náhodně na obrazovce jak želvu, tak cíl. Použije přitom dvakrát proceduru NASTAV:

```
TO NASTAV.HRU
CS
TELL 0 ST
NASTAV
TELL 1 ST
NASTAV
END
```

Nyní příkaz NASTAV.HRU vyzkoušíme. Hra však bude přehlednější, když cíl nebude vypadat stejně jako želva. Zkusme proto na jeho místě použít tvar sluníčka, který jsme vytvořili jako tvar 1 v minulé kapitole. Dosáhneme toho tím, že do NASTAV.HRU doplníme příkaz SETSH 1:

```
TO NASTAV.HRU
CS
TELL 0 ST
NASTAV
SETSH 1
TELL 1 ST
NASTAV
END
```

NASTAV.HRU použijeme v další proceduře, kterou nazveme HRA. V proceduře vyvoláme též hlídače příkazu WHEN a želvu uvedeme do pohybu:

```
TO HRA
NASTAV.HRU
SETSP 30
WHEN 19 [SETSP 0]
END
```

Poslední příkaz procedury zastaví želvu 1, když se dotkne cíle. Číslo 19 symbolizuje střet mezi želvou 0 a želvou 1.

Hru několikrát vyzkoušíme, např.:

```
HRA
RT 45
LT 10
```

21.2 Ovládání hry klávesou

Je mnoho způsobů, jak realizovat ovládání hry, kterou jsme právě napsali. Lze se například ptát slovy nebo celými větami a Logo bude podobně odpovídat. My však dáme přednost ovládání pomocí kláves. Pomůže nám k tomu nová funkce RC:

```
PR RC
```

Logo čeká na stisknutí nějaké klávesy. Stiskneme A. Funkce RC tento znak přijme a předá ho příkazu PRINT, který znak vytiskne:

```
A
```

Na další znak pak už Logo nečeká, pokračuje hned dál.

Procedura RC podobně jako HEADING nebo POS patří mezi funkce, a proto může být použita pouze jako vstup nějakého příkazu nebo jiné funkce. Můžeme, například, přiřadit výstup z RC proměnné pomocí příkazu MAKE:

```
MAKE "KLAVESA RC
```

Nyní napíšeme nějaký znak (například Z). Znak se po napsání na obrazovce neobjeví. Uloží se do proměnné KLAVESA:

```
PRINT :KLAVESA
```

a Logo odpoví:

```
Z
```

tj. znak, který jsme napsali.

V následující proceduře využijeme právě uvedeného postupu. Proceduru nazveme POSLYS. Bude reagovat na stisknutí kláves L nebo P, a to takto:

P - otočení o 15 stupňů doprava

L - otočení o 15 stupňů doleva

```
TO POSLYS
MAKE "ODPOVED RC
IF :ODPOVED = "P [RT 15]
IF :ODPOVED = "L [LT 15]
END
```

V proceduře POSLYS je výstupem pro příkaz RC proměnná ODPOVED. Příkazem IF si procedura otestuje obsah proměnné ODPOVED. Příkaz IF vyžaduje dva vstupy. Prvním je vlastní podmínka testu (buď je splněna nebo ne), druhým je seznam příkazů. Je-li podmínka splněna, příkazy ze seznamu se provedou. V testu jsme použili další funkci jazyka Logo a to znaménko rovnosti (=). Porovnává shodnost dvou vstupů. Jsou-li oba shodné, podmínka je splněna, v opačném případě ne. Procedura POSLYS však běží bez ohledu na to, jestli něco napíšeme či nikoliv.

Jinak by tomu bylo, kdybychom použili jinou funkci, a to KEYP. Tato funkce nastavuje podmínku jako splněnou už tím, že je stisknuta jakákoliv klávesa. Není-li stisknuta žádná klávesa, podmínka splněna není. Pomocí KEYP můžeme napsat proceduru, která vyvolá proceduru POSLYS pouze tehdy, bude-li stisknut nějaký znak. Proceduru označíme jako HRAJ:

```
TO HRAJ
IF KEYP [POSLYS]
HRAJ
END
```

Procedura HRAJ je rekurzivní, posledním řádkem volá sebe samu. Poběží, i když se želva dávno zastavila. Proceduru HRAJ zastavíme klávesou BREAK. Zkusíme ji s procedurou HRA:

```
HRA
HRAJ
```

Želvu ovládáme stiskem P nebo L.

*** POZOR ***
Pokud želva na stisk kláves P nebo L nereaguje, zřejmě nemáme přepnuto na správnou sadu znaků. Procedura totiž vyžaduje pouze velká písmena. Použijeme proto SHIFT a CAPS.

21.3 Další možnosti

Pokusíme se naši hru dále vylepšit. Nejprve přidáme do procedury NASTAV.HRU příkaz k nakreslení překážek:

```
TO NASTAV.HRU
CS
TELL 0 ST
REPEAT 3 [NASTAV PD FD 20]
NASTAV
SETSH 1
TELL 1 ST
NASTAV
END
```

Příkazem REPEAT ve třetím řádku nakreslí procedura tři krátké čáry v náhodném rozmístění.

Proceduru HRA dále obohatíme o proceduru POKYNY:

```
TO HRA
POKYNY
NASTAV.HRU
SETSP 10
WHEN 19 [SETSP 0]
WHEN 4 [BK 6]
HRAJ
END

TO POKYNY
SS
PRINT [RIDTE ZELVU TAK, ABY DOSAHLA CILE]
PRINT [K JEJIMU OVLADANI POUZIJTE KLAVES P NEBO L]
END
```

Proceduru vyzkoušíme:

```
HRA
```

Je o hodně lepší, i když se dá ještě dále vylepšovat.

Procedura HRAJ běží dál, i když se želva zastavila. Zastavíme ji klávesou BREAK. Upravme proceduru tak, aby se zastavila sama. Jak ale pozná, že hra skončila? Jedna z možností by byla sledovat rychlost želvy 1. V proceduře HRA hlídač příkazů WHEN hned poté, co se želvy střetnou, želvu zastaví příkazem SETSP 0. Procedura HRAJ bude proto kontrolovat, zda želva stojí. Pokud tomu tak bude, procedura se zastaví:

```
TO HRAJ
IF SPEED = 0 [STOP]
IF KEYP [POSLYS]
HRAJ
END
```

Použili jsme zde novou funkci SPEED, která obsahuje okamžitou rychlost želvy.
Konečnou verzí hry ještě vyzkoušíme:

```
HRA
```

Želva skutečně na klávesy P a L reaguje. Jestliže dosáhne cíle, procedura se zastaví a na obrazovce se objeví kurzor .
Hru můžeme samozřejmě dále upravovat a její techniky využít i v jiných podobných hrách. Můžeme například, nahradit želvu O jiným, vhodnějším tvarem nebo doplnit hru o zobrazení skóre třeba tak, že hráč ztratí bod, kdykoliv se dotkne jeho želva překážky.

21.4 Slovníček

V této kapitole jsme se seznámili s těmito novými příkazy:

IF	testuj
STOP	zastav

a novými funkcemi:

KEYP	test klávesnice
RANDOM	náhodné číslo
RC	čte znak
SPEED	rychlost želvy
=	test rovnosti

22. Rekurzivní procedury

22.1 Nekonečná rekurze

Členění úloh do procedur patří mezi nejsilnější stránky jazyka Logo. Každá procedura má své jméno a je zcela nezávislá. Může vyvolávat a využívat jakoukoliv jinou proceduru a též může být sama volána jinými procedurami. Některé procedury, jak jsme již viděli, volají samy sebe. Takovým říkáme rekurzivní. Použili jsme je již v 19. kapitole, kde procedury VICE a SPI byly rekurzivní:

```
TO VICE :KROK :UHEL
FD :KROK
RT :UHEL
VICE :KROK :UHEL
END
```

```
TO SPI :KROK :UHEL :ZMENA
FD :KROK
RT :UHEL
SPI :KROK+:ZMENA :UHEL :ZMENA
END
```

Ve své definici VICE volá zase VICE, podobně SPI volá SPI.

O co zde vlastně jde? Představme si, že Logo má neomezený počet jakýchsi pomocníků. Pokaždé, kdy je vyvolána nějaká procedura, je povolán příslušný pomocník, aby prohlédl definici procedury. Začne provádět postupně jednotlivé instrukce a k tomu si povolává další pomocníky. K provedení celé procedury je zpravidla zapotřebí několik takových pomocníků.

Například je-li vyvolána procedura VICE, musí nejdřív její pomocník povolat pomocníka příkazu FD a poté pomocníka příkazu RT. Až ti svoji práci skončí, povolá se další, již druhý pomocník procedury VICE. Tento druhý pomocník pak opět povolá pomocníky příkazu FD a RT, a pak třetího pomocníka procedury VICE. Zatím první a druhý pomocník čekají. Tak jsou povoláváni stále další pomocníci procedury VICE, a to do té doby, dokud nestiskneme klávesu BREAK.

V tomto případě se první pomocník procedury VICE nikdy nedočká splnění úkolu od druhého pomocníka, neboť ten marně čeká na třetího pomocníka, a tak dále. Taková rekurze se nazývá nekonečná.

To však neznamená, že takto donekonečna musí pracovat každá rekurzivní procedura. Může být totiž navržena tak, aby se po čase zastavila sama. Ve skutečnosti jsou právě podmínky zastavení v rekurzivních procedurách tím nejdůležitějším. Jinak řečeno: každá správná rekurzivní procedura se musí umět zastavit.

22.2 Zastavení rekurzivních procedur

22.2.1 Zastavení při stisku klávesy

Zastavit rekurzivní proceduru můžeme různými způsoby. Například použitím funkce KEYP, kterou jsme zavedli v předcházející kapitole:

```
TO SPI :KROK :UHEL :ZMENA
IF KEYP [STOP]
FD :KROK
RT :UHEL
SPI :KROK + :ZMENA :ZMENA
END
```

Výstup z funkce KEYP je testován. Procedura SPI se zastaví, je-li stisknuta libovolná klávesa.

V tomto případě se tedy poslední pomocník procedury SPI zastaví při stisknutí libovolné klávesy (provede příkaz STOP) a předá řízení svému předchůdci. Tomu však zbývá provést pouze závěrečný příkaz END, čili vlastně také pouze vrací činnost svému předchůdci. Jednotliví pomocníci si tedy vzájemně předávají řízení až do té doby, dokud jej nepřevezme první

pomocník procedury SPI, který celou činnost ukončí a řízení se vrací zpět k uživateli, o čemž nás vyrozumí i otazník a kurzor na novém textovém řádku.

22.2.2 Jiný způsob zastavení

Elegantnější řešení samozřejmě je, pokud se procedura zastaví úplně sama. Budeme například testovat hodnotu proměnné KROK. Procedura se pak zastaví, bude-li hodnota proměnné KROK větší než 15. K tomu editorem změníme druhý řádek procedury:

```
TO SPI :KROK :UHEL :ZMENA
IF :KROK > 15 [STOP]
FD :KROK
RT UHEL
SPI :KROK + :ZMENA :UHEL :ZMENA
END
```

Upravenou proceduru vyzkoušíme:

```
SPI 1 10 .1
```

V proceduře VICE použijeme k jejímu zastavení malého triku. Po ukončení kresby se totiž želva vrátí do původní polohy. To znamená, že musela udělat úplnou otáčku, tedy otočit se o 360 stupňů, případně o celý násobek 360. Stačí proto vědět, jaká byla její orientace ve chvíli, kdy začala kreslit. Pak po každém natočení otestujeme její novou orientaci. Nejprve však musíme zařídit, aby si Logo někde výchozí orientaci želvy pamatovalo, nazveme takovou proměnnou START:

```
MAKE "START HEADING
```

Test v proceduře musíme ale umístit až za příkaz RT. Proč? Kdyby byl totiž před příkazem RT, procedura by se zastavila dříve, než by želva stačila cokoliv nakreslit:

```
TO VICE :KROK :UHEL
FD :KROK
RT :UHEL
IF HEADING = :START [STOP]
VICE :KROK :UHEL
END
```

Proceduru vyzkoušíme:

```
VICE 10 10
```

Nesmíme však zapomenout před voláním procedury na přiřazení její počáteční orientace do proměnné START. Jinak procedura nebude pracovat. Nejlepší bude, když tento příkaz vyčleníme přímo do procedury. Přejmenujeme VICE na VICE1 a přidáme START jako další vstup:

```
TO VICE1 :KROK :UHEL :START
```

```
FD :KROK
RT UHEL
IF HEADING = :START [STOP]
VICE1 :KROK :UHEL :START
END
```

Vytvoříme novou proceduru VICE, která použije funkci HEADING a předá její hodnotu proceduře VICE1:

```
TO VICE :KROK :UHEL
VICE1 :KROK :UHEL HEADING
END
```

Příklad:

```
VICE 10 10
```

Nyní vše obstarává procedura VICE. HEADING je funkce, proto není uvozena dvojtečkou, jak je tomu u názvu proměnných.

22.3 Místo slovníčku

V této lekci jsme si vysvětlili pojem rekurze a naučili se tento mocný prostředek jazyka Logo používat. Nezaváděli jsme již žádné nové příkazy ani funkce, což však neznamená, že jsme slovník jazyka zcela vyčerpali. Mnohé procedury jazyka Logo teprve čekají, až se s nimi seznámíme. Jsou to zejména příkazy a funkce, které pracují se slovy, řetězci a seznamy.

Touto kapitolou však končí náš kurs Loga věnovaný obzvláště začátečníkům. Seznámili jsme se v něm se všemi základními vlastnostmi jazyka, jeho grafickými možnostmi a způsobem, jak rozšiřovat jeho slovník. Pomocí želvičky již umíme nakreslit například domek se zahrádkou, ale také navrhovat rozmanité obrazce a ornamenty. Dokážeme rovněž sestavit jednoduchou hru, ovládanou z klávesnice nebo joystickem.

Následující část příručky bude jakousi vyšší školou jazyka Logo. Některé kapitoly budou opakováním nebo prohlubováním již známých skutečností, v jiných se setkáme s novými příkazy a funkcemi a na příkladech si vysvětlíme jejich použití. Pokud jste tedy dočetli naši příručku až sem a máte chuť pokračovat, zveme Vás na další procházku jazykem Logo. Stačí otočit list

C. LOGO PŘEVÁŽNĚ VÁŽNĚ

Tato část příručky se snaží seznámit čtenáře s jazykem Logo poněkud podrobněji a více systematicky než část předchozí. Předpokládá alespoň povšechnou znalost práce s počítačem Atari, proto se nezabývá například zavedením jazyka do počítače nebo způsobem psaní textu. Také uváděné příklady jsou popisovány méně důkladně než dosud.

23. Základní příkazy

Jazyk Logo využívá ke komunikaci s uživatelem dvou obrazovek: textové a grafické. Textová obrazovka se uplatňuje zejména v úlohách numerického a nenumerického charakteru, želví grafika využívá obrazovku grafickou. Následující kapitoly se týkají práce s grafickou obrazovkou, o textové obrazovce budeme mluvit později.

Dominantním prvkem grafické obrazovky je obrazec ve tvaru želvy. Umístění želvy na obrazovce je jednoznačně dáno její polohou (pozicí) a směrem natočení (orientací). Tyto parametry lze změnit pomocí příkazu pro želví grafiku. Při každé změně polohy nakreslí želva čáru z původní do nové pozice. Tímto způsobem lze tedy vytvářet kresby tvořené přímými čárami.

K pohybu želvy po obrazovce slouží 5 základních příkazů:

```
HOME
FORWARD n
BACK n
RIGHT n
LEFT n
```

Příkaz HOME nastavuje želvu do tzv. základní pozice. Je přitom umístěna doprostřed obrazovky a orientována směrem nahoru. V základní pozici je želva automaticky i po vyvolání jazyka Logo.

Příkaz FORWARD n (zkratka FD n) posune želvu o n kroků dopředu. Je-li parametr n záporný, pohybuje se želva dozadu. Jak již bylo uvedeno, při tomto pohybu kreslí želva čáru. Opačný účinek má příkaz BACK n (zkratka BK n), při kterém se naopak želva pohybuje o n kroků dozadu. Platí tedy, že příkazy:

```
FORWARD n
BACK -n
```

jsou ekvivalentní. Oba příkazy mění pouze polohu želvy, orientace se přitom nemění.

Příkaz LEFT n (zkratka LT n) otočí želvu o n stupňů doleva. V případě záporné hodnoty n se želva otáčí doprava. Opačný účinek má příkaz RIGHT n (zkratka RT n), který otáčí

želvou o n stupňů doprava. Opět tedy platí, že příkazy:

```
LEFT n
RIGHT -n
```

jsou ekvivalentní. Poloha želvy se těmito příkazy nemění.

Chceme-li tedy nakreslit pomocí želvy čtverec o straně 50 kroků, stačí zadat následující posloupnost příkazů:

```
FORWARD 50
RIGHT 90
FORWARD 50
RIGHT 90
FORWARD 50
HOME
```

Symbol želvy je samozřejmě pouze orientační a přebírá v Logu pozici kurzoru. Viditelnost želvy lze přitom potlačit příkazem:

```
HT
```

a znovu obnovit příkazem:

```
ST
```

Viditelnost želvy lze kdykoliv testovat pomocí funkce SHOWNP, která vrací hodnotu TRUE v případě, že je želva viditelná, a hodnotu FALSE v případě, že je želva neviditelná.

Hodnoty TRUE a FALSE slouží v Logu k vyjádření pravdy a nepravdy. Mají tedy význam logických konstant, jsou výsledkem funkcí, které mají charakter testu, nebo logických výrazů (viz kapitola 38).

Vlastnosti funkce SHOWNP si můžeme ověřit aplikací následující posloupnosti příkazů:

```
HT
PRINT SHOWNP
ST
PRINT SHOWNP
```

24. Grafická obrazovka

Grafickou obrazovku inicializujeme příkazem CS. Po provedení tohoto příkazu je nastaven základní tvar grafické obrazovky s textovým oknem ve spodní části. Obrazovka je přitom vymazána, želva umístěna do výchozí polohy.

K pouhému vymazání grafické obrazovky slouží příkaz CLEAN. Tímto příkazem se vymažou pouze nakreslené čáry, poloha a orientace želvy se však nemění.

Rozměry grafické obrazovky (v krocích želvy) se liší verze od verze, uživatel může tyto parametry částečně ovlivnit direktivou .SETSCR (viz kapitola 49). Rozměry obrazovky se mohou, ale nemusí shodovat s akčním rozsahem želvy. Ten je nastavován příkazy WRAP a WINDOW.

V základním režimu zobrazení WRAP je pohyb želvy omezen rozměry obrazovky. Zadáme-li příkaz, který by měl za následek pohyb želvy ven z obrazovky, objeví se želva na protilehlém okraji.

V režimu WINDOW je naopak prostor pro pohyb želvy prakticky neomezený. Z tohoto prostoru je však viditelná pouze část obrazovky kolem vychozí pozice. Zadáme-li nyní příkaz, který by měl za následek pohyb želvy ven z obrazovky, želva zmizí.

Rozdíl mezi oběma režimy lépe pochopíme, provedeme-li následující dvě posloupnosti příkazů a srovnáme jejich důsledky:

```
CS
WRAP
LT 30 FD 5000

CS
WINDOW
LT 30 FD 5000
```

V prvním případě skončí želva svůj pohyb ve viditelné části obrazovky, v druhém případě obrazovku opustí.

25. Pero

Dosud jsme předpokládali, že při každé změně polohy želvy je nakreslena úsečka spojující původní a novou pozici želvy. Tuto stopu pohybu želvy přisuzujeme tzv. peru neboli písátku, kterým je želva "vybavena" a umožňuje tak dokumentovat svůj pohyb.

Výchozí a základní funkcí pera je funkce kreslící, kterou nastavujeme příkazem PENDOWN (zkratka PD). V tomto režimu pero skutečně kopíruje každý pohyb želvy, při každé změně polohy želvy je nakreslena další úsečka.

Pero však má i další funkce, které lze nastavit speciálními příkazy. Příkaz PE převádí pero do režimu gumy, ve kterém jsou všechny kresby při pohybu želvy mazány. Příkaz PX provádí inverzi všech bodů ve stopě želvy, tj. kresby maže, prázdná místa vyplňuje.

Příkaz PENUP (zkratka PU) zcela vyřazuje všechny funkce pera. V tomto režimu se želva pohybuje, aniž by přitom zanechávala jakoukoliv stopu.

Stav písátka želvy lze zjistit pomocí funkce PEN. Hodnotou funkce mohou být slova PD (kreslící režim), PE (režim gumy), PX (režim inverze) a PU (pero vyřazeno).

Změny funkce pera lze samozřejmě využít i při kreslení na grafické obrazovce. Například k pouhému přesunu želvy o 50 kroků (bez kresby) lze použít posloupnost příkazu:

```
PENUP
FORWARD 50
PENDOWN
```

nebo zkráceně:

```
PU FD 50 PD
```

V jazyce Atari Logo má každá želva k dispozici tři písátka, která se liší barvou. Změnou právě použitého pera lze tedy měnit i barvu kresby. K volbě pera slouží příkaz:

```
SETPN n
```

kde n je číslo pera (0 až 2). Číslo aktuálního písátka udává funkce PN a lze ji zjistit například příkazem:

```
PRINT PN
```

Co nakreslí následující posloupnost příkazu?

```
CS
SETPN 0
RT 30 FD 50
SETPN 1
RT 120 FD 50
SETPN 2
HOME
SETPN 0
```

I bez ověření na počítači by měl každý poznat rovnostranný trojúhelník o délce strany 50 kroků, který má každou stranu jiné barvy.

26. Souřadnice

Umístění želvy na obrazovce (bez ohledu na její viditelnost) je dáno její pozicí a orientací. Pozice želvy je uspořádaná dvojice [X Y], která udává vodorovnou a svislou souřadnici želvy vzhledem k výchozímu postavení (HOME) v krocích želvy. Orientací želvy rozumíme úhel, o který je želva natočena vzhledem k výchozímu postavení (ve stupních). Želva natočená nahoru má tedy orientaci 0, želva natočená doprava má orientaci 90, želva natočená dolů má orientaci 180, želva natočená doleva má orientaci 270, apod.

Po příkazu HOME nebo CS, kdy se želva dostane zpět do výchozího postavení, má tedy polohu [0 0] a orientaci 0.

Okamžité umístění želvy udávají dvě funkce: POS a HEADING. Hodnotou funkce POS je dvouprvkový seznam x-ové a y-ové souřadnice želvy (v krocích), hodnotou funkce HEADING je orientace želvy (ve stupních).

Například po provedení příkazu:

```
CS
FORWARD 50
LEFT 90
```

budou souřadnice želvy [0 50] a orientace 270 stupňů, o čemž se lze přesvědčit dvojicí příkazů:

```
PRINT POS
PRINT HEADING
```

K zjištění pouze jedné z obou souřadnic polohy želvy slouží funkce XCOR a YCOR.

Změnit umístění želvy na grafické obrazovce můžeme nejen použitím základních příkazů želví grafiky, ale také příkazy pro nastavení pozice a orientace želvy. Tyto příkazy jsou absolutní, vztažené pouze vůči výchozímu umístění, na rozdíl od základních příkazů, které jsou relativní, vztažené vůči okamžitému umístění želvy.

K nastavení nové polohy želvy na obrazovce slouží příkaz:

```
SETPOS [X Y]
```

kde X a Y jsou souřadnice nové polohy želvy. Tento příkaz mění polohu želvy obdobně jako FORWARD nebo BACK, proto aktivuje i činnost pera. Chceme-li provést pouhý přesun želvy bez kreslení (resp. mazání), musíme nejprve vypojit činnost písátka, např.:

```
PU
SETPOS [X Y]
PD
```

Ke změně pouze jedné ze souřadnic polohy želvy slouží příkazy:

```
SETX X
SETY Y
```

kde X, resp. Y je nová hodnota měněné souřadnice. Pomocí těchto příkazů se želva pohybuje pouze vodorovným, resp. svislým směrem. Dvojice příkazů:

```
SETX X
SETPOS [X YCOR]
```

resp.:

```
SETY Y
SETPOS [XCOR Y]
```

jsou ekvivalentní. Příkazy SETPOS, SETX a SETY nemění orientaci želvy.

K nastavení nové orientace želvy slouží příkaz:

```
SETH n
```

kde n je nová orientace želvy ve stupních. Například příkaz:

```
SETH 45
```

natočí želvu úhlopříčně vpravo. Příkaz SETH nemá vliv na polohu (souřadnice) želvy.

Příkazy želví grafiky (relativní) a příkazy pro nastavení nového umístění želvy (absolutní) lze vzájemně kombinovat podle potřeby. Lze tedy například psát:

```
CS
FORWARD 50
```

```
SETPOS [50 50]
SETH 180
FORWARD 50
HOME
```

Tato posloupnost příkazů nakreslí čtverec obdobně jako příklad uvedený v kapitole 23.

Kromě kreseb vytvořených pohybem želvy umožňuje Atari Logo využít i příkazy pro kresbu jednotlivých bodů o daných souřadnicích. K tomu slouží příkaz:

```
DOT [X Y]
```

který nakreslí bod o souřadnicích X a Y. Barva bodu a charakter kresby jsou dány aktuálním písátkem. Příkaz DOT nemění umístění želvy.

Doplníme-li tedy výše uvedenou posloupnost příkazu pro kresbu čtverce příkazem:

```
DOT [25 25]
```

zobrazí se ve středu čtverce bod. Želva přitom zůstává ve výchozím postavení.

27. Barvy

Při práci s grafickou obrazovkou rozlišujeme celkem 5 barev: barvu pozadí obrazovky, barvu želvy a barvy per- 0 až 2. Hodnoty těchto barev (v číselném kódu) udávají následující funkce:

```
BG    - pro barvu pozadí obrazovky
COLOR - pro barvu želvy
PC n  - pro barvu pera (n=0,1,2)
```

Výchozí hodnoty můžeme zjistit pomocí příkazů:

```
PRINT BG
PRINT COLOR
PRINT PC 0
PRINT PC 1
PRINT PC 2
```

Vztah mezi kódy a skutečnými barvami udává tabulka v příloze 4.

Všech 5 barev na grafické obrazovce můžeme změnit příslušnými příkazy:

```
SETBG C    - pro barvu pozadí obrazovky
SETC C     - pro barvu želvy
SETPC n C  - pro barvu pera (n=0,1,2)
```

0-15

Hodnota C udává kód nově nastavené barvy. Například příkaz:

```
SETBG 48
```

nastaví barvu pozadí na tmavě modrou.

Uživatelé, kteří nemají barevný televizor nebo monitor, těžko rozliší mezi vychozími barvami jednotlivých per. Pro tyto případy je vhodné změnit odstíny per příkazy:

```
SETPC 0 0
SETPC 1 4
SETPC 2 7
```

Pero 0 bude nyní kreslit černě, pero 1 šedě a pero 2 bíle.

28. Definice tvaru želvy

Jazyk Atari Logo neumožňuje pouze měnit jednotlivě používané barvy, ale dokonce dovoluje modifikovat a vytvářet nové tvary želvy. Ve standardním provedení má želva skutečně stylizovaný tvar želvy, s krunýřem, nohama, hlavičkou a ocáskem. Tuto podobu grafického kurzoru však lze změnit a místo želvy použít například robota, křížek nebo jiný symbol, který si sami vytvoříme.

Do paměti počítače můžeme uložit až 15 různých návrhů tvaru želvy (kromě standardního) a z nich podle potřeby vybrat vhodný tvar jako aktuální. K definici tvaru želvy lze použít dvě metody:

a) návrh pomocí grafického editoru,
b) návrh pomocí číselného seznamu.

28.1 Návrh pomocí editoru

Grafický editor vyvoláme příkazem:

```
EDSH n
```

kde n je číslo navrhované předlohy (1 až 15). Po vyvolání editoru se objeví tzv. editační obrazovka, v jejímž levém horním rohu je umístěn rastr 8x16 bodů. Pomocí kurzorových šipek se po rastru můžeme pohybovat a klávesou INSERT (>) zakreslovat jednotlivé body nového tvaru želvy. Pomocí klávesy CLEAR (<) naopak můžeme již zvolené body mazat.

V některých verzích Loga zastupuje funkci kláves INSERT a CLEAR mezerník. Stisknutím mezerníku se mění zobrazení daného bodu rastru z neviditelného na viditelné a také naopak.

Po ukončení návrhu opustíme editor stisknutím klávesy ESC. Navrhovaný tvar želvy se uloží do paměti a můžeme jej kdykoli vyvolat. Návrh můžeme ukončit též stisknutím klávesy BREAK, v tomto případě se však do paměti počítače neukládá.

28.2 Návrh pomocí seznamu

Návrh pomocí číselného seznamu předpokládá, že jsme předlohu vytvořili odděleně od jazyka Logo, například pomocí programu Ultrafont. Tento program dokonce umožňuje přímo získat osmici hodnot vyjadřující tvar jednotlivých znaků. Želva v Logu je tvořena 8x16 body, tj. jakoby dvěma znaky umístěnými nad

sebou. Definuje ji tedy 16 hodnot (0 až 255, každý řádek představuje 1 hodnotu).

(Kódování tvaru znaku do 8 jednobytových hodnot předpokládá znalost dvojkové soustavy a lze se o něm dočíst například v příručce Pavla Dočekala "ABC o počítačích Atari", kapitola 3.)

Máme-li číselný seznam vytvořený, můžeme ho Logu zadat příkazem:

PUTSH n l

kde n je číslo předlohy (1 až 16) a l je seznam 16 hodnot uzavřených v hranatých závorkách. Například příkaz:

PUTSH 2 [24 24 24 24 24 24 66 66 60 60 90 90 90 90 153 153]

definuje želvu ve tvaru firemního znaku Atari a ukládá ji do paměti jako předlohu číslo 2.

Pro libovolnou definovanou předlohu 1 až 15 můžeme získat seznam definujících hodnot prostřednictvím funkce GETSH n, kde n je číslo předlohy, např.:

PRINT GETSH 2

Funkci lze samozřejmě použít i na předlohy definované pomocí editoru.

Želvě lze již definovanou předlohu přiřadit příkazem:

SETSH n

kde n je číslo předlohy. Číslo 0 je určeno pro standardní předlohu (skutečnou želvu), čísla 1 až 15 pro předlohy uživatelské. Například výše definovanou předlohu ve tvaru znaku Atari lze vyvolat příkazem:

SETSH 2

Číslo aktuální předlohy (standardní nebo uživatelské) je hodnotou funkce SHAPE a můžeme jej tedy zjistit pomocí příkazu:

PRINT SHAPE

Oproti standardní želvičce má uživatelsky definovaná želva jednu nevýhodu. Zatímco standardní želva svou polohou skutečně vystihuje orientaci a při změně orientace se otáčí, tvar uživatelsky definované želvy zůstává stále stejný a nelze tedy pouhým pohledem odhadnout, jaká je její skutečná orientace. To může působit určité problémy, a proto doporučujeme ve stadiu výuky nebo návrhu využívat převážně želvičku standardní.

29. Volba aktuální želvy

Některé dialekty jazyka Logo umožňují zároveň pracovat i s více želvami najednou. Tuto možnost má i Atari Logo, které dovoluje používat až 4 želvy. Jejich základní charakteristiky jsou vyjádřeny následující tabulkou:

Želva č.	0	1	2	3
SHOWNP	TRUE	FALSE	FALSE	FALSE
COLOR	7	20	44	68
PN	0	0	0	0
PEN	PD	PD	PD	PD
SHAPE	0	0	0	0

Standardně tedy pracujeme pouze s želvou 0.

Volbu aktuální želvy provádíme příkazem:

```
TELL n
```

kde n je číslo želvy (0 až 3). Tato volba je globální, tj. všechny procedury nyní použité budou prováděny zvolenou aktuální želvou.

Příkaz TELL umožňuje naráz aktualizovat i více želv. V tom případě je parametrem příkazu seznam čísel všech aktualizovaných želv (v hranatých závorkách), např.:

```
TELL [0 1 2 3]
```

V tomto případě budou všechny zadané procedury provádět všechny 4 želvy současně.

Číslo aktuální želvy (nebo seznam aktuálních želv) získáme jako hodnotu funkce EACH, např.:

```
PRINT EACH
```

Kromě globální definice aktuální želvy můžeme použít i lokální definici, která má tvar:

```
ASK n l
```

kde n je číslo (popř. seznam čísel) aktuální želvy a l je seznam procedur, které má aktuální želva provést. Po provedení seznamu l, bude opět platit nastavení dané posledním příkazem TELL.

Tak například příkaz:

```
ASK [0 1] [RT 45 FD 50]
```

provede se želvami 0 a 1 posloupnost příkazů:

```
RIGHT 45
FORWARD 50
```

a se želvami 2 a 3 neprovede nic. Aktualizace želv nastavená posledním příkazem WHO zůstane nezměněna.

U každé ze 4 želv lze měnit tyto parametry:

polohu,
orientaci,
viditelnost,
barvu, tvar,
aktuální písátko,

funkci písátka.

Barvy per 0, 1 a 2 jsou pro všechny želvy společné.
K ovládání více želv najednou lze použít i příkaz EACH, který má tvar:

```
EACH l
```

kde l je seznam příkazů (obdobně jako u příkazu ASK). Příkaz EACH provede zadaný seznam příkazů pro všechny aktuální želvy najednou. Objeví-li se v seznamu l funkce WHO, dosadí se za její hodnotu vždy číslo želvy, s kterou je daná posloupnost příkazů prováděna.
Například příkaz:

```
EACH [RT WHO * 90 FD 50]
```

provede za předpokladu, že jsou aktivní všechny čtyři želvy, totéž jako posloupnost:

```
ASK 0 [RT 0 FD 50]
ASK 1 [RT 90 FD 50]
ASK 2 [RT 180 FD 50]
ASK 3 [RT 270 FD 50]
```

Za funkci WHO se postupně dosadí hodnoty 0,1,2 a 3.
Obdobně příkaz:

```
EACH [PRINT SHAPE]
```

vytiskne čísla předloh všech aktuálních želv. Samotný příkaz:

```
PRINT SHAPE
```

by totiž vytiskl pouze hodnotu tvaru aktuální želvy s nejnižším číslem (tedy tvar želvy 0).

30. Nastavení rychlosti želvy

Dosud jsme v našich úvahách předpokládali, že pokud želvu neuvedeme do pohybu, samovolně svoje umístění nemění. Pro každou želvu však lze nastavit rychlost, se kterou se bude po obrazovce pohybovat samovolně, tj. nezávisle na našich instrukcích. Příkaz pro nastavení rychlosti má tvar:

```
SETSP n
```

kde n je rychlost želvy (přibližně v krocích za sekundu). Tento příkaz nastavuje rychlost všech aktuálních želv najednou.
Každá želva se může pohybovat jinou rychlostí, lze tedy například použít příkazu:

```
EACH [SETSP WHO * 5 + 5]
```

Želva se pohybuje stanovenou rychlostí dopředu (v případě záporného parametru dozadu) ve směru daném její orientací.

Rychlost samovolného pohybu každé želvy můžeme zjistit pomocí funkce SPEED. Například rychlost druhé želvy zjistíme příkazem:

```
ASK 2 [PRINT SPEED]
```

31. Vytváření nových příkazů

Dosud jsme se seznámili s grafickými příkazy jazyka Logo a naučili se je používat. Těžištěm programování v Logu však není přímá aplikace jednotlivých příkazů, ale vytváření nových procedur, příkazů a funkcí, a obohacování základního slovníku jazyka.

31.1 Definice nových příkazů

Každý nový příkaz vytváříme jako posloupnost příkazů již dříve definovaných. Výjimku tvoří pouze příkazy rekurzivní, které mohou obsahovat samy sebe. Musíme však pomocí programových konstrukcí zabezpečit, aby nedošlo k rekurzi nekonečné.

K definici nového příkazu je nutno přepnout počítač do režimu definice. K tomu slouží příkaz TO ve tvaru:

```
TO s
```

kde s je posloupnost znaků tvořící jméno nového příkazu. Jako jméno příkazu nelze použít jméno jiného standardního nebo již definovaného příkazu, zápis čísla nebo posloupnost znaků obsahující mezeru.

Například chceme-li nadefinovat novou proceduru CTVEREC, použijeme příkaz:

```
TO CTVEREC
```

Po zadání příkazu TO se počítač přepne do definičního režimu, což se projeví změnou prvního znaku na řádku z "?" (otazník) na ">" (větší než). Nyní již můžeme zadávat jednotlivé příkazy, které budou tvořit námi definovanou proceduru. Tyto příkazy se nebudou v průběhu zadávání provádět, ale budou se pouze ukládat do paměti počítače. Definici nového slova zakončíme a definiční režim opustíme příkazem END.

Například definice nového příkazu CTVEREC, který nakreslí čtverec o straně 50 kroků, bude mít tvar (včetne prvních znaků na řádku, které se samozřejmě nezadávají):

```
?TO CTVEREC
>FORWARD 50
>RIGHT 90
>FORWARD 50
>RIGHT 90
```

```
>FORWARD 50
>RIGHT 90
>END
CTVEREC DEFINED
?
```

Nyní již máme příkaz CTVEREC nadefinovaný, o čemž nás informuje i zpráva CTVEREC DEFINED, která se objeví ihned po zadání příkazu END, a můžeme jej tedy použít jako kterýkoliv jiný příkaz jazyka, například v posloupnosti:

```
CS
CTVEREC
LEFT 90
CTVEREC
LEFT 90
CTVEREC
LEFT 90
CTVEREC
HOME
```

která nakreslí 4 čtverce symetricky rozmístěné kolem bodu [0 0].

31.2 Editace příkazů

Již nadefinované příkazy můžeme kdykoliv modifikovat nebo opravit s využitím textového editoru jazyka Logo, který vyvoláme příkazem EDIT (zkratka ED) ve tvaru:

```
EDIT l
```

kde l je seznam všech příkazů, které chceme modifikovat. Tento příkaz vyvolá textový editor a na obrazovce zobrazí stávající definice uvedených příkazů. Například k úpravě příkazu CTVEREC použijeme příkaz:

```
EDIT [CTVEREC]
```

V případě, že chceme editovat pouze jeden příkaz, můžeme uvést jméno příkazu bez hranatých seznamových závorek, ale s uvozovkou na začátku, tedy:

```
EDIT "CTVEREC
```

Pro pohyb po obrazovce editoru a opravy lze použít kurzorových šipek a běžných funkcí obrazovkového editoru jako v jazyce Basic. Dále lze využít některých speciálních funkcí (CTRL je zkratka pro klávesu CONTROL):

CTRL A	přesun kurzoru na začátek řádku,
CTRL E	přesun kurzoru na konec řádku,
CTRL X	přesun kurzoru na začátek textu,
CTRL Z	přesun kurzoru na konec textu,
CTRL V	přesun kurzoru na následující stránku,
CTRL W	přesun kurzoru na předcházející stránku,
SHIFT DEL	vymaž řádek (uložení do bufferu).

CTRL Y vyvolání bufferu,
CTRL 3 vyvolání naposledy napsaného znaku.

Funkce CTRL V a CTRL W se uplatní v případě, že editovaný text je delší než jedna stránka obrazovky. Funkce SHIFT DEL a CTRL Y se používají k přesunu textového řádku na jiné místo v textu, popřípadě na jeho rozmnožení.

Funkce CTRL A, CTRL E, CTRL 3 a CTRL Y lze použít i mimo textový editor, přičemž v tomto případě funkce CTRL Y kopíruje naposledy zadaný příkaz (řádek).

Po ukončení všech úprav lze editor opustit klávesou ESC. O provedení oprav je vydána zpráva stejně jako při vytváření nových příkazů (... DEFINED). Musíme však dát pozor, zda jsou všechny editované příkazy ukončeny slovem END. V opačném případě může dojít k zhroucení celého systému. Editor lze opustit i stisknutím klávesy BREAK, v tomto případě se však žádné úpravy neprovádějí.

Editor lze použít i k definici nových příkazů, tato metoda je často výhodnější než přímá definice (zejména díky možnosti pohybu kurzoru po celé obrazovce). V tomto případě lze použít příkaz EDIT i bez parametru, který vyvolá prázdnou editační obrazovku.

32. Větvení a cykly

I jazyk Logo obsahuje programové konstrukce, které umožňují vytvářet nelineární rozvětvené definice nových příkazů.

32.1 Příkaz větvení IF

Základní programovou konstrukcí v Logu je příkaz větvení IF, který obdobně jako ve vyšších programovacích jazycích má dvě formy: jednoduchou a složenou.

Příkaz jednoduchého větvení má tvar:

```
IF p l
```

kde p je podmínka, tj. logický výraz nebo funkce (s hodnotami TRUE a FALSE) a l je seznam příkazů který je proveden pouze v případě, že je podmínka splněna (má hodnotu TRUE).

Příkaz složeného větvení má tvar:

```
IF p l1 l2
```

kde l1 a l2 jsou dva seznamy příkazů. Je-li splněna podmínka p (má hodnotu TRUE), provede se seznam příkazů l1, jinak (p má hodnotu FALSE) je proveden seznam příkazů l2.

Jako ukázku použití příkazu IF můžeme uvést definici procedury XT, která změní viditelnost želvy (z viditelné na neviditelnou a opačně):

```
TO XT
IF SHOWNP [HT] [ST]
END
```

32.2 Příkaz cyklu REPEAT

Další strukturou používanou v jazyce Logo je cyklus REPEAT. Příkaz REPEAT má tvar:

```
REPEAT n l
```

kde l je seznam příkazů tvořící tělo cyklu a n je počet opakování těchto příkazů. Pomocí příkazu REPEAT můžeme například zjednodušit definici příkazu CTVEREC:

```
TO CTVEREC
REPEAT 4 [FD 50 RT 90]
END
```

Cyklus REPEAT odpovídá obecnému cyklu "for" z vyšších programovacích jazyků. Ostatní cykly je nutno řešit pomocí rekurze, například:

```
TO JIH
IF HEADING = 180 [STOP]
RT 1
JIH
END
```

Příkaz STOP slouží k ukončení provádění dané procedury. Obvykle se používá právě k ukončení rekurzivního procesu. Výše uvedenou proceduru JIH lze opsat slovně jako: "dokud není želva orientována na jih, natáčej ji o 1 stupeň doprava". (Stejný účinek má samozřejmě jediný příkaz SETH 180).

Komu chybí v Logu klasické pascalské cykly "for", "while" a "repeat", může využít jejich definice z projektu v kapitole 51.

33. Proměnné a jejich hodnoty

Obdobně jako u vyšších programovacích jazyků můžeme i v Logu vytvářet proměnné a přiřazovat jim hodnoty. Hodnotou proměnné může být číslo, logická konstanta, odkaz na jinou proměnnou nebo seznam libovolných hodnot.

Abychom rozlišili proměnné od procedur, používáme následující zápisy:

X	označuje jméno příkazu X
"X	označuje jméno proměnné X,
:X	označuje hodnotu proměnné X.

Napíšeme-li tedy:

```
PRINT SUMA
```

chápe Logo slovo SUMA jako jméno procedury a snaží se ji provést. Příkaz:

```
PRINT "SUMA
```

vytiskne jméno proměnné SUMA, čili text:

```
SUMA
```

Skutečnou hodnotu proměnné SUMA vytiskneme příkazem:

```
PRINT :SUMA
```

Pojem proměnné umožňuje rozšířit definici nových příkazů o parametry. Každý příkaz může mít libovolný počet parametrů, které uvádíme za jméno příkazu. Tím zobecníme definice nových příkazů, například již uvedený příkaz CTVEREC můžeme upravit na:

```
TO CTVEREC :STRANA
REPEAT 4 [FD :STRANA RT 90]
END
```

Chceme-li nyní nakreslit čtverec o straně 50, zadáme příkaz:

```
CTVEREC 50
```

který dosadí do parametru STRANA hodnotu 50 a vykoná danou proceduru. Parametry příkazu jsou proměnné lokální, tj. jejich hodnoty platí pouze v rámci daného příkazu.

Uvedený příklad bychom mohli dokonce zobecnit i na libovolný n-úhelník:

```
TO NUHELNIK :POCET :STRANA
REPEAT :POCET [FD :STRANA RT 360 / :POCET]
END
```

a definici příkazu CTVEREC uvést jako:

```
TO CTVEREC :STRANA
NUHELNIK 4 :STRANA
END
```

Pro přiřazení hodnot proměnným slouží příkaz:

```
MAKE jm val
```

kde jm je jméno proměnné a val hodnota proměnné. Příkladem přiřazení mohou být příkazy:

Logo	Pascal
MAKE "B 4.23	B:=4.23
MAKE "P "TRUE	P:=TRUE
MAKE "S [1 2 3]	S:=[1 2 3]
MAKE "A :B	A:=B
MAKE "A "B	A^:=B

Vysvětlení si zaslouží poslední dvě přiřazení. Příkaz:

```
MAKE "A :B
```

přiřazuje proměnné A hodnotu proměnné B, tj. proměnné A i B budou mít stejnou hodnotu, což lze znázornit graficky jako:

```
A          B
:          :
4.23       4.23
```

Naproti tomu příkaz:

```
MAKE "A "B
```

přiřazuje proměnné A jméno proměnné B, což lze graficky znázornit jako:

```
A
:
B
:
4.23
```

Vzniká tedy hierarchická struktura připomínající dynamický seznam z jazyka Pascal. Takové struktury budeme nazývat vertikální na rozdíl od seznamů, které budeme nazývat strukturami horizontálními.

Pro zjišťování hodnot ve vertikálních strukturách slouží funkce THING, která jménu dané proměnné vrací její hodnotu:

funkce	hodnota
THING "A	"B
THING "B	4.23
THING :A	4.23

Zápis THING "X má stejnou funkci jako zápis :X, tj. označuje hodnotu proměnné X.

V textovém editoru lze prohlížet a popřípadě měnit hodnoty všech proměnných. K tomu slouží příkaz:

```
EDNS
```

který vyvolá editační obrazovku se zápisem všech proměnných a jejich hodnot ve formě příkazu MAKE, např.:

```
MAKE "A "B
MAKE "B 4.23
```

Editor opustíme stisknutím klávesy ESC (změny se provedou) nebo BREAK (změny se neprovedou).

Ke zjištění, zda dané jméno představuje jméno proměnné (čili zda daná proměnná má definovanou hodnotu), slouží logická funkce NAMEP, např.:

```
PRINT NAMEP "A
```

Hodnotou proměnné může být i seznam, a to i seznam příkazů. Povolenou konstrukcí je tedy zápis:

```
MAKE "PRIKAZ [FD 30 RT 60 FD 50]
```

K provedení posloupnosti příkazů ve tvaru seznamu slouží příkaz RUN, který má tvar:

```
RUN l
```

kde l je daný seznam příkazů. V našem případě lze tedy použít příkaz:

```
RUN :PRIKAZ
```

a provede se posloupnost:

```
FD 30 RT 60 FD 50
```

34. Příkazy a funkce

Všechny procedury jazyka Logo můžeme rozdělit do dvou skupin: příkazy a funkce. Funkce je taková procedura, jejímž výsledkem je (kromě provedení příslušných akcí) určitá hodnota, která musí být dále zpracována, např. přiřazena proměnné, vytištěna, apod. Na rozdíl od funkcí, příkazy žádné hodnoty nevracejí.

Příkladem příkazů mohou být zápisy:

```
FORWARD 30
MAKE "A :B + 1
ASK 2 [SETSP 3]
NUHELNIK 3 20
```

Příkladem funkcí jsou zápisy:

```
WHO
SPEED
PC 2
POS
```

Dosud jsme si říkali, jak lze rozšířit slovník jazyka Logo o nové příkazy. K definici nových funkcí můžeme použít naprosto stejný postup, je však nutno označit hodnotu, která má být výsledkem. K tomu slouží příkaz výstupu funkce.

Příkaz výstupu funkce OUTPUT (zkratka OP) má tvar:

```
OUTPUT x
```

kde x je výstupní hodnota definované funkce. Tento příkaz ukončuje činnost daného bloku (funkce), proto musí být uveden jako poslední. Například zápis:

```
TO MOCNINA :X
OUTPUT :X * :X
END
```

definuje funkci MOCNINA, která slouží k výpočtu druhé mocniny číselné hodnoty. Lze tedy psát:

```
PRINT MOCNINA 5
```

Tento příkaz vytiskne hodnotu 25.

I v definicích funkčních hodnot lze použít různé programové konstrukce. Jednou z nich je podmíněna funkce IF, která má tvar:

```
IF p l1 l2
```

kde p je podmínka (logický výraz nebo funkce) a l1, l2 jsou seznamy vyjadřující zápisy funkcí. Je-li splněna podmínka p (p má hodnotu TRUE), bude výsledkem hodnota funkce l1, jinak (p má hodnotu FALSE) hodnota funkce l2.

Jako příklad použití funkce IF si uvedeme návrh funkce VAL, která nalezne hodnotu ve vertikální struktuře:

```
TO VAL :STR
OP IF NAMEP :STR [VAL THING :STR][:STR]
END
```

Tato funkce je navržena pomocí rekurze. Dokud platí, že hodnotou dané proměnné je jméno proměnné, opakujeme volání funkce VAL pro tuto hodnotu. V opačném případě je výsledkem hodnota proměnné.

K ověření činnosti funkce STR si vytvoříme nějaký vertikální seznam, např. pomocí příkazů:

```
MAKE "PROM1 "PROM2
MAKE "PROM2 "PROM3
MAKE "PROM3 "PROM4
MAKE "PROM4 5.234
```

Vytvořený seznam si můžeme znázornit jako:

```
PROM1
 :
PROM2
 :
PROM3
 :
PROM4
 :
5.234
```

Použijeme-li nyní příkaz:

```
PRINT VAL "PROM1
```

vytiskne nám hodnotu:

```
5.234
```

K provedení funkce ve tvaru seznamu slouží funkce RUN, která je obdobou příkazu RUN a má tvar:

```
RUN l
```

kde l je seznam tvořící zápis funkce. Funkci RUN můžeme s výhodou použít například v numerických úlohách, ve kterých je vstupním parametrem zápis aritmetické funkce (např. kreslení grafu zadané funkce).

35. Procedury vstupu a výstupu

Každý programovací jazyk obsahuje příkazy nebo procedury pro vstup a výstup hodnot. Jejich prostřednictvím komunikují vytvořené programy s uživatelem. Také jazyk Logo obsahuje funkce a příkazy pro vstup a výstup hodnot.

35.1 Procedury vstupu

Pro vstup hodnot slouží celkem 3 funkce. První z nich, KEYP, je logická funkce, která pouze testuje, je-li stisknuta některá z kláves klávesnice.

Funkce RC slouží k načtení jednoho znaku z klávesnice. Tento znak je také výstupem funkce (jako jednoznakový řetězec).

Funkce RL čte z klávesnice celý vstupní řádek až po znak RETURN. Tento řádek je interpretován jako seznam znaků a je výstupní hodnotou funkce RL.

Lze tedy například psát:

```
MAKE "X RL
```

Tento příkaz načte z klávesnice celý řádek a přiřadí jej proměnné X.

35.2 Procedury výstupu

K výstupu hodnot na obrazovku slouží opět 3 příkazy. Základní z nich je příkaz PRINT (zkratka PR), který má tvar:

```
PRINT x
```

kde x je libovolná zobrazitelná hodnota (číselná, logická, jméno proměnné, výraz apod.). Správnými zápisy příkazu PRINT jsou například:

```
PRINT FALSE
PRINT 1 + 1
PRINT "Ahoj
PRINT :A
PRINT :X > 5
PRINT SPEED
```

Je-li výstupem příkazu PRINT seznam, např.:

```
PRINT [1 2 3]
```

tiskne se bez vnějších závorek, tj. jako:

```
1 2 3
```

Obdobnou funkci jako PRINT má i příkaz SHOW. Jediným rozdílem mezi oběma příkazy je, že příkaz SHOW tiskne seznamy včetne vnějších závorek, tj. příkaz:

```
SHOW [1 2 3]
```

vytiskne:

```
[1 2 3]
```

Jak příkaz PRINT, tak i příkaz SHOW tisknou každou hodnotu na nový řádek. Chceme-li vytisknout více hodnot na jeden řádek, použijeme příkaz TYPE. Například sekvence:

```
TYPE "Jak
TYPE "_se
PRINT "_mas?
```

vytiskne jediný řádek:

```
Jak_se_mas?
```

Příkazy PRINT, SHOW a TYPE patří mezi příkazy s proměnným počtem parametrů. Má-li takový příkaz více než 1 parametr, udává se včetně těchto parametrů do kulatých závorek. Například příkaz:

```
(PRINT :A :B :A + :B)
```

má stejnou funkci jako trojice příkazů:

```
TYPE :A
TYPE :B
PRINT :A + :B
```

36. Textová obrazovka

36.1 Obrazovky jazyka Logo

Jazyk Atari Logo obsahuje pro komunikaci s uživatelem 3 nezávislé obrazovky: editační, grafickou a textovou. Do této chvíle jsme hovořili převážně o obrazovce grafické, případně editační.

Textovou obrazovku vyvoláme příkazem:

```
TS
```

Textová obrazovka slouží pouze pro práci s textem, ke zpětnému vyvolání grafické obrazovky slouží příkaz:

SS

K vyvolání grafické obrazovky dojde také při prvním zadání grafického příkazu nebo funkce, např. CS.

Standardní grafická obrazovka obsahuje ve své spodní části část obrazovky textové. Tuto část můžeme vyřadit příkazem:

FS

čímž se celá plocha obrazovky vyhradí pro účely grafiky.

Kromě příkazu SS, TS a FS můžeme k přepínání obrazovek použít i speciální příkazy editoru:

CTRL S pro grafickou obrazovku,
CTRL T pro textovou obrazovku,
CTRL F pro grafickou obrazovku s vyřazeným textovým oknem.

Příkazy editoru lze použít kdykoliv, tedy i během činnosti procedury (například k odkrytí části kresby na grafické obrazovce, kterou zakrývalo textové okno).

36.2 Práce s textovou obrazovkou

V dalších odstavcích se budeme zabývat příkazy pro práci s textovou obrazovkou. Jedním z nich je příkaz pro mazání textové obrazovky, který má tvar:

CT

Tento příkaz lze použít i v grafické obrazovce k vymazání textového okna.

Funkci želvy přebírá v textové obrazovce klasický textový kurzor. Jeho souřadnice získáme jako hodnotu funkce:

CURSOR

která vrací dvouprvkový seznam [X Y]. Znak o souřadnicích [0 0] je umístěn v levém horním rohu textové obrazovky.

K nastavení kurzoru do dané polohy slouží příkaz:

SETCURSOR [X Y]

který má stejnou funkci jako příkaz:

POSITION X,Y

v jazyce Atari Basic.

Textovou obrazovku využijeme zejména při řešení úloh numerického, popřípadě nenumerického charakteru.

37. Aritmetické operace

Jazyk Logo umí pracovat s číselnými hodnotami tří typů:

hodnoty celočíselné (např. 23),
hodnoty desetinné (např. 3.14159)
hodnoty s exponentem (např. 2.5E-06).

Zápis 2.5E-06 představuje číslo 2.5*10^-6, kde symbol ^ nahrazuje znak mocniny (který však v Logu neexistuje). To znamená, že 2.5E-06 je zápis čísla 0.0000025.

S číselnými hodnotami lze provádět v Logu aritmetické výpočty obdobně jako ve vyšších programovacích jazycích. Uživatel má přitom k dispozici následující aritmetické operace:

n+m součet
n-m rozdíl
n*m součin
n/m podíl

a dále uvedené aritmetické funkce:

COS n kosinus
SIN n sinus
SQRT n druhá odmocnina

Úhel ve funkcích SIN a COS se zadává ve stupních.

S využitím těchto operací, funkcí a kulatých závorek lze vytvářet aritmetické výrazy. Parametry těchto výrazů mohou být číselné konstanty (např. 5.23), proměnné (např :SOUCET) i funkce (např. SPEED). Při vyhodnocování aritmetických výrazů platí stejná pravidla jako v matematice, včetně priority operací.

Číselným aritmetickým výrazem může být například zápis:

```
1 + SIN (:X - :Y) * COS (:X + :Y)
```

který by v běžné matematické notaci mohl být zapsán jako:

```
1+sin(x-y).cos(x+y)
```

Goniometrické funkce lze použít například ke kreslení kružnic, elips a oblouků. Ukážeme si definici příkazu, který nakreslí kružnici se středem [X Y] a poloměrem R:

```
TO KRUZNICE :X :Y :R
PENUP
SETPOS [:X :Y + :R]
PENDOWN
MAKE "FI 0
REPEAT 36 [SETPOS [(:X + :R * SIN :FI)(:Y + :R * COS :FI)]
MAKE "FI :FI + 10]
END
```

Kromě základních operací obsahuje Logo i další funkce, které lze využít v aritmetických výpočtech. Jsou to jednak

funkce součtu a součinu, zaokrouhlovací funkce a funkce náhodných hodnot.

Funkce součtu a součinu mají stejný význam jako odpovídající aritmetické operace, tj.:

```
SUM n m       je totéž jako  n+m
PRODUCT n m   je totéž jako  n*m
```

Tyto funkce mohou mít i více než dva parametry, v tomto případě se zápis funkce včetně parametrů uzavírá do závorek, například:

```
(SUM 1 :X :Y)
```

je totéž jako:

```
1 + :X + :Y
```

K zaokrouhlování desetinných čísel na celá slouží dvě funkce. Funkce INT n zaokrouhluje číslo n vždy dolů, oproti tomu funkce ROUND n zaokrouhluje podle běžných pravidel zaokrouhlování. To znamená, že:

```
INT 3.1 --> 3     ROUND 3.1 --> 3
INT 3.7 --> 3     ROUND 3.7 --> 4
```

K vypočtům s celými čísly slouží i funkce:

```
REMAINDER n m
```

která vypočte zbytek po celočíselném dělení n/m. Například příkaz:

```
PRINT REMAINDER 22 7
```

vytiskne hodnotu:

```
1
```

Funkci celočíselného dělení lze definovat jako:

```
TO DIV :N :M
OP INT :N / :M
END
```

Funkce náhodných hodnot RANDOM má tvar:

```
RANDOM n
```

kde n je celé kladné číslo. Tato funkce slouží ke generování celých čísel v rozsahu 0 až n-1. Například ke generování čísel 1 až 6 lze použít funkcí:

```
TO KOSTKA
OP 1 + RANDOM 6
END
```

Příkaz RERANDOM (bez parametrů) slouží k nastavení výchozích podmínek generátoru. Použití příkazu RERANDOM zaručuje vždy stejný průběh generovaných hodnot.

S pomocí uvedených operací a funkcí a dalších konstrukcí jazyka Logo můžeme definovat další aritmetické funkce, které Atari Logo standardně nemá. Uvedeme si jako příklad definici funkce POWER (zkratka PW), která slouží k výpočtu nezáporné celočíselné mocniny (x na n-tou):

```
TO POWER :X :N
IF :N = 0 [OP 1]
OP PRODUCT :X POWER :X :N - 1
END
TO PW :X :N
OP POWER :X :N
END
```

K výpočtu byl použit rekurzivní vztah:

x^n=x*x^(n-1)

kde znak ^ zastupuje funkci mocniny.

Některé uživatele možná zklame skutečnost, že Logo neobsahuje funkce LOG (logaritmus), EXP (exponenciální funkce) a ATN (arkustangenta). Tyto funkce nenajdeme prakticky u žádného dialektu jazyka Logo, tedy ani u Atari Loga. Důvod je zřejmě v tom, že původní záměr jazyka Logo se opíral zejména o želví grafiku, kde jsou tyto funkce prakticky nevyužitelné.

38. Logické operace

Jak již bylo uvedeno dříve, jazyk Logo obsahuje dvě konstanty k vyjádření logických hodnot:

TRUE	pro vyjádření pojmu "pravda".
FALSE	pro vyjádření pojmu "nepravda".

Logické hodnoty jsou výsledkem srovnání číselných hodnot (např. :X>3) a různých logických funkcí (např. SHOWNP, NAMEP aj.).

V jazyce Logo můžeme použít všech 6 typů srovnání číselných hodnot:

m=n	rovná se
m<>n	nerovná se
m>n	větší než
m<n	menší než
m>=n	větší nebo rovno
m<=n	menší nebo rovno

Pomocí logických operací můžeme vytvářet z jednoduchých srovnání a logických funkcí složitější logické výrazy. Logo obsahuje tři logické operátory funkce NOT (negace vyjadřuje vztah "není pravda, že"), AND (logický součin vyjadřuje vztah "a současně") a OR (logický součet vyjadřuje vztah "nebo").

Význam všech 3 funkcí vyjadřují následující tabulky:

:X	NOT :X
TRUE	FALSE
FALSE	TRUE

:X	:Y	AND :X :Y
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

:X	:Y	OR :X :Y
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Například k vyjádření vztahu 3<x<5 lze použít funkci AND:

```
AND 3<:X :X<5
```

Logické operace AND a OR lze použít i jako funkce s proměnným počtem parametrů. V tomto případě musí být celý zápis funkce uzavřen v závorkách, např.:

```
(AND :X < 5 :Y > 3 SHOWNP)
```

Logické funkce a operace a z nich vytvořené logické výrazy se používají převážně v příkazech a funkcích IF k vyjádření podmínek větvení, například:

```
IF (AND :X<5 :Y>3 SHOWNP) [HOME]
```

Obdobně jako u funkcí aritmetických můžeme vytvářet i vlastní funkce logické. Například funkce XOR (nonekvivalence), která je definována jako:

```
x xor y = (x and not y) or (not x and y)
```

může být vytvořena jako:

```
TO XOR :X :Y
OP OR (AND :X NOT :Y)(AND NOT :X :Y)
END
```

39. Slova a seznamy

Všechny objekty, se kterými jazyk Logo pracuje, lze rozdělit do dvou skupin:

a) slova (též atomy),
b) seznamy.

Slova jsou všechny jednoduché údaje, tj. jména proměnných, číselné a textové hodnoty, apod. Seznamy jsou uspořádané struktury, tvořené slovy nebo opět seznamy (tzv. víceúrovňové seznamy).

39.1 Slova

Slova, pokud se vyskytují samostatně, tj. ne jako položky seznamu, označujeme na začátku uvozovkami (viz jména proměnných), abychom je odlišili od názvu procedur. Uvozovky jsou nepovinné pouze u číselných údajů a několika jazykových konstant: TRUE, FALSE, PU, PD, PX, PE. Příklady správně zapsaných slov jsou:

```
TRUE
12.76
"ATOM
"D:PRIKLAD.LOG
3.14159E-12
```

39.2 Seznamy

Seznamy uzavíráme do závorek. Pokud jsou položkami seznamu slova, uvádíme je vždy bez počátečních uvozovek. Seznamy mohou mít libovolný počet položek i úrovní, tzv. prázdný seznam neobsahuje žádnou položku. Příklady správně zapsaných seznamů jsou:

```
[]
[A]
[[[[ATOM]]]]
[1 2 3 4 5]
[ZELVA ROBOT SOVA]
[1 [a b] TRUE]
```

Třetí a poslední z uvedených seznamů jsou víceúrovňové, první seznam je prázdný.

Počet prvků seznamu (na nejvyšší úrovni) udává funkce COUNT, například:

```
PRINT COUNT [a b c d]
```

vytiskne hodnotu 4.

39.3 Práce se seznamy

Pro operace se seznamy je vyhrazena celá skupina funkcí jazyka Logo. Můžeme je rozdělit do 3 skupin:

a) přístupové funkce,
b) spojovací funkce,
c) testovací funkce.

39.3.1 Přístupové funkce

Přístupové funkce umožňují vybrat ze seznamu příslušnou hodnotu nebo část seznamu. Jsou analogií funkcí cdr a car jazyka Lisp:

FIRST s	první položka seznamu s
LAST s	poslední položka seznamu s
BUTFIRST s (zkratka BF)	seznam s bez první položky
BUTLAST s (zkratka BL)	seznam s bez poslední položky

Například platí:

```
FIRST [A B C]           -> "A
BUTFIRST [A B C]        -> [B C]
LAST [A B C]            -> "C
BUTLAST [A B C]         -> [A B]
FIRST BUTFIRST [A B C] -> "B
```

a také:

```
LAST [1 [2 [3]]]            -> [2 [3]]
LAST LAST [1 [2 [3]]]       -> [3]
LAST LAST LAST [1 [2 [3]]] -> 3
```

Pro přístup k libovolné n-té položce seznamu slouží v některých dialektech jazyka Logo funkce ITEM. Neboť Atari Logo tuto funkci nemá, nadefinujeme si ji jako:

```
TO ITEM :KOLIKATY :SEZNAM
IF :KOLIKATY = 1 [OP FIRST :SEZNAM]
OP ITEM (:KOLIKATY - 1) (BF :SEZNAM)
END
```

Pak příkaz:

```
PRINT ITEM 4 [a b c d e f g]
```

vytiskne hodnotu:

```
d
```

39.3.2 Spojovací funkce

Spojovací funkce slouží k vytváření seznamu z jednotlivých položek nebo částí. Jazyk Atari Logo má tyto spojovací funkce:

```
SENTENCE o1 o2
LIST o1 o2
FPUT o s
```

Funkce SENTENCE (zkratka SE) vytváří z daných objektů o1 a o2 seznam tak, že oba objekty zřetězí. Například:

```
SE "A [B C]      -> [A B C]
SE TRUE FALSE    -> [TRUE FALSE]
SE [1 2][3 4]    -> [1 2 3 4]
```

Funkce LIST rovněž vytváří z daných objektů seznam, uvažuje je však jako položky nového seznamu. Například:

```
LIST "A [B C]      -> [A [B C]]
LIST TRUE FALSE    -> [TRUE FALSE]
LIST [1 2][3 4]    -> [[1 2][3 4]]
```

Funkce FPUT přiřadí objekt o jako první položku seznamu s, například:

```
FPUT "A [B C]      -> [A B C]
FPUT [1 2][3 4]    -> [[1 2] 3 4]
```

Funkce SENTENCE a LIST mohou mít i více než 2 parametry v tom případě však musí být funkce včetně parametru uzavřena do závorek, např.:

```
PRINT (SE "A "B "C [D E F] "G)
MAKE "LST (LIST :X :Y :Z)
```

39.3.3 Testovací funkce

Testovací funkce slouží k zjištění vlastností daných objektů. V Atari Logu jsou tyto funkce:

```
WORDP o          test, zda daný objekt je slovo,
NUMBERP o        test, zda daný objekt je číslo,
LISTP o          test, zda daný objekt je seznam,
EMPTYP s         test na prázdný seznam,
EQUALP o1 o2     test shodnosti dvou objektů,
MEMBERP o s      test, zda objekt o je položkou seznamu s.
```

Všechny testovací funkce jsou logického typu, tj. jejich výsledkem je logická hodnota TRUE (v případě splnění testu) nebo FALSE (v případě nesplnění testu). Například:

```
WORDP [A B C]          -> FALSE
NUMBERP 3.14           -> TRUE
LISTP "ATARI           -> FALSE
```

```
EMPTYP []                -> TRUE
EQUALP "A FIRST [A B] -> TRUE
MEMBERP "A [[A] [B]]  -> FALSE
```

39.3.4 Návrh uživatelských funkcí

Jako příklad návrhu složitější funkce pro práci se seznamy si uvedeme funkci JETAM?, která zjistí, zda daný objekt CO je prvkem seznamu KDE, a to na libovolné úrovni vřazení:

```
TO JETAM? :CO :KDE
IF EMPTYP :KDE [OP FALSE]
IF MEMBERP :CO :KDE [OP TRUE]
IF LISTP  FIRST :KDE [OP OR (JETAM? :CO FIRST :KDE)(JETAM?
:CO BF :KDE)]
OP JETAM? :CO BF :KDE
END
```

Tato funkce je rekurzivní. Její vyčíslení může být, zejména u složitějších struktur, poněkud zdlouhavé. To je způsobeno omezenou rychlostí zpracování procedur v jazyce Logo.

Například příkaz:

```
PRINT JETAM? "A [[A][B]]
```

vytiskne hodnotu:

```
TRUE
```

příkaz:

```
MAKE "TEST JETAM? [A B] [[A][B]]
```

přiřadí proměnné TEST hodnotu FALSE.

40. Slova jako řetězce

40.1 Slova a řetězce

Pokud jsme v minulé kapitole předpokládali, že slova jsou objekty dále nedělitelné, není to zcela pravda. Na slova můžeme totiž hledět jako na řetězce znaků a provádět s nimi obdobné operace jako se seznamy.

Řetězcem budeme nazývat v Logu libovolnou posloupnost znaků s vyjímkou mezery. Samostatně stojící řetězce budeme označovat předponou " (uvozovky). Vyjímku tvoří řetězce vyjadřující číselné hodnoty a konstanty jazyka Logo (TRUE, FALSE, PU, PD, PE, PX).

Délkou řetězce budeme chápat počet znaků, ze kterých se skládá (bez znaku "). Řetězce mohou mít libovolnou nezápornou délku, řetězec délky 0 nazýváme prázdný a budeme ho označovat:

Délku řetězce udává funkce COUNT, např. příkaz tisku:

```
PRINT COUNT "Atari
```

vytiskne hodnotu 5.

40.2 Operace s řetězci

K výběru jednotlivých znaků nebo částí řetězce používáme přístupové funkce FIRST, BUTFIRST, LAST a BUTLAST stejně jako u seznamu. Platí:

```
FIRST r     - první znak řetězce r
LAST r      - poslední znak řetězce r
BUTFIRST r  - řetězec r bez prvního znaku
BUTLAST r   - řetězec r bez posledního znaku
```

Například:

```
FIRST "Atari      ->  "A
LAST "Atari       ->  "i
BF "Atari         ->  "tari
BL "Atari         ->  "Atar
FIRST BF "Atari   ->  "t
BF BL "Atari      ->  "tar
```

K zřetězení dvou řetězců v jediný (ke spojení dvou slov v jediné) slouží funkce WORD ve tvaru:

```
WORD r1 r2
```

kde r1 a r2 jsou spojované řetězce. Funkce WORD je obdobou funkce SENTENCE u seznamu. Rovněž může mít i více než dva parametry, takže například příkaz:

```
PRINT (WORD "Atari 800 "XL)
```

vytiskne:

```
Atari 800XL
```

K testování řetězců slouží dvě testovací funkce které se rovněž používají pro práci se seznamy:

```
EMPTYP r      -  test na prázdný řetězec r,
EQUALP r1 r2  -  test na shodnost řetězců r1 a r2.
```

Platí tedy:

```
EMPTYP "                ->  TRUE
EMPTYP "Atari           ->  FALSE
EQUALP "Atari "ATARI    ->  FALSE
EQUALP "PU PU           ->  TRUE
```

Jako příklad použití uvedených funkcí si můžeme ukázat definici funkce OBRAT, která k danému řetězci vytvoří řetězec

převrácený (tj. napsaný pozpátku):

```
TO OBRAT :SLOVO
IF EMPTYP :SLOVO [OP :SLOVO]
OP WORD OBRAT BF :SLOVO FIRST :SLOVO
END
```

Pak příkaz:

```
PR OBRAT "Atari
```

vytiskne slovo:

```
iratA
```

Funkce OBRAT je rekurzivní a převrácený řetězec vytiskne najednou. Postupně, tzn. znak po znaku ho bude tisknout procedura OTOC:

```
TO OTOC :SLOVO
IF EMPTYP :SLOVO [PR [] STOP]
TYPE LAST :SLOVO
OTOC BL :SLOVO
END
```

Napíšeme-li:

```
OTOC "Logo-je-bajo
```

vytiskne se postupně:

```
ojab-ej-ogoL
```

40.3 Řetězce a znaky

Pro práci s řetězci slouží také další dvě funkce, které vyjadřují vztah mezi znakem a jeho ATASCII kódem. Jsou to funkce ASCII (znak -> kód) a CHAR (kód -> znak). Funkce ASCII r vrací jako hodnotu ATASCII kód prvního znaku řetězce r, funkce CHAR n vrací jednoznaký řetězec tvořený znakem o ATASCII kódu n. Například:

```
ASCII "Atari        ->  65
ASCII "A            ->  65
CHAR 65             ->  "A
CHAR ASCII "Atari   ->  "A
```

Následující funkce VELKE využívá funkci ASCII a CHAR a mění všechna malá písmena v řetězci na velká:

```
TO VELKE :SLOVO
IF EMPTYP :SLOVO [OP :SLOVO]
OP WORD (ZMENA FIRST :SLOVO) (VELKE BF :SLOVO)
```

kde ZMENA je pomocná funkce definovaná jako:

```
TO ZMENA :ZNAK
IF (ASCII :ZNAK > 96) AND (ASCII :ZNAK < 124) [OP CHAR
(ASCII :ZNAK 32)]
OP :ZNAK
END
```

Příkaz:

```
PRINT VELKE "Atari
```

pak vytiskne text:

```
ATARI
```

41. Ovladače

S dosud uvedenými příkazy a funkcemi se můžeme setkat prakticky u každé verze jazyka Logo. Dialekt Atari Logo však obsahuje i další procedury, jejichž popis bude obsahem této a následujících kapitol.

Jazyk Atari Logo například umožňuje spolupracovat s ovladači. Dále uvedené funkce umožňují v našich projektech používat joysticky, ovladače typu paddle a světelné pero.

41.1 Joystick

K načtení stavu křížových ovladačů (joysticků) slouží funkce JOY, která má tvar:

```
JOY n
```

kde n je číslo ovladače (0 nebo 1). Kódování polohy joysticku je rozdílné od jazyka Basic (funkce STICK), jednotlivé polohy mají následující hodnoty:

```
 7 0 1
  \:/
6-(-1)-2
  /:\
 5 4 3
```

V klidové poloze ovladače je tedy hodnota funkce JOY rovna -1.

Logická funkce JOYB n testuje stisknutí tlačítka (triggeru) u joysticku s číslem n. Stisknuté tlačítko vrací hodnotu TRUE, nestisknuté FALSE. V závislosti na stavu tlačítka ovladače lze tedy vytvářet větvení typu:

```
IF JOYB 0 [....][....]
```

Použití ovladače typu joystick demonstruje následující příklad:

```
TO POHYB
IF JOYB 0 [CS]
IF JOY 0 <> -1 [SETH 45 * JOY 0]
FD 1
POHYB
END
```

41.2 Paddle

Stav ovladače typu paddle (například grafické tabulky, potenciometru, apod.) určuje funkce PADDLE, která má tvar:

```
PADDLE n
```

kde n je číslo ovladače (0 až 3). Funkce může nabývat hodnot 0 až 228 a shoduje se se stejnojmennou funkcí v jazyce Basic.

Logická funkce PADDLEB n testuje stisknutí tlačítka na ovladači typu paddle. Význam této funkce je zcela obdobný jako u funkce JOYB.

K nakreslení bodu o souřadnicích nastavených pomocí grafické tabulky lze například použít příkaz:

```
IF PADDLEB 0 [DOT (PADDLE 0 - 114) (PADDLE 1 - 114)]
```

41.3 Světelné pero

Majitelé světelného pera mohou ve svých projektech, zejména grafických, využít i tento typ ovladače. K zjištění souřadnic světelného pera slouží v některých dialektech Loga funkce PEN, která má tvar:

```
PEN 0     pro zjištění x-ové souřadnice,
PEN 1     pro zjištění y-ové souřadnice.
```

Například chceme-li pomocí světelného pera nastavit pozici želvy, stačí zadat příkaz:

```
SETPOS [PEN 0 PEN 1]
```

42. Zvukový výstup

42.1 Příkaz TOOT

Jazyk Atari Logo umožňuje vytvářet zvuky pomocí dvou zvukových generátorů, které se nastavují pomocí příkazu TOOT. Tento příkaz má tvar:

```
TOOT n1 n2 n3 n4
```

kde:

n1 číslo zvukového kanálu (0 nebo 1)
n2 výška tónu (v 5/6 Hz)
n3 hlasitost (0 až 15)
n4 délka tónu (v 1/50 s)

Chceme-li např. vytvořit tón komorní a (a1=440 Hz) v délce 5 sekund, zadáme příkaz:

```
TOOT 0 440 * 5 / 6 10 5 * 50
```

Oproti jazyku Basic nelze u Loga volit zabarvení tónu, určité možnosti v tomto směru však poskytuje použití obou kanálů současně. Například kombinace:

```
TOOT 0 440 * 5 / 6 7 5 * 50
TOOT 1 880 * 5 / 6 7 5 * 50
```

vytvoří tón s měkčím zabarvením, příkazy:

```
TOOT 0 440 * 5 / 6 7 5 * 50
TOOT 1 441 * 5 / 6 7 5 * 50
```

lze docílit tzv. "chorus efekt".

42.2 Příkaz SETENV

Další možnosti ve vytváření zvuku poskytuje příkaz SETENV, který slouží k nastavení dozvuku. Příkaz má tvar:

```
SETENV n1 n2
```

kde n1 je číslo kanálu a n2 udává, za kolik 1/50 s má být snížena hlasitost o jednotku.
Například volba:

```
SETENV 1 50
```

nastaví dozvuk, při kterém se tón s hlasitostí 10 ztlumí na nulu za 10 sekund.

42.3 Programování hudby

I s takto omezenými možnostmi lze v Logu se zvukem pracovat, např. při výuce intonace. Vytvořme například tabulku tónů a frekvencí jednočárkované oktávy:

```
TO OKTAVA
MAKE "c 261.626
MAKE "c+ 277.183
MAKE "d- 277.183
:
MAKE "b- 466.164
MAKE "h 493.883
```

```
MAKE "C 523.251
END
```

Nyní lze vytvořit příkaz pro zahrání jednoho tónu:

```
TO TON :T
TOOT 0 :T * 5 / 6 10 50
END
```

a příkaz k zahrání celé melodie:

```
TO MELODIE :S
IF EMPTYP :S [STOP]
TON FIRST :S
MELODIE BF :S
END
```

Nyní stačí inicializovat tabulku příkazem:

```
OKTAVA
```

a můžeme začít hrát. Například stupnici C-dur zahrajeme jako:

```
MELODIE [c d e f g a h C]
```

a obdobně harmonickou stupnici c-moll:

```
MELODIE [c d e- f g a h C]
```

43. Přerušovací systém

Přerušení je stav, při kterém počítač (procesor, program) zastaví svoji dosavadní činnost a spustí zvláštní podprogram, který se nazývá obslužný. Po ukončení běhu tohoto podprogramu pokračuje počítač v původní činnosti.

Přerušení je velmi běžný jev v činnosti počítače. Zobrazení, zvukový výstup, čtení klávesnice, komunikace s vnějšími zařízeními, to vše se provádí prostřednictvím přerušení. Tato přerušení nemůže uživatel ovlivnit, v některých programovacích jazycích však existuje možnost, aby si uživatel definoval svá vlastní přerušení včetně způsobu jejich obsluhy.

V Logu lze přerušení demonstrovat například na příkazu SPEED. Po nastavení nenulové hodnoty parametru tohoto příkazu se dá želvička do pohybu, a to nezávisle na našich příkazech. Přestože želvičce zadáváme příkazy, po určité době (1/50 s) přeruší jejich vykonávání a posune se dopředu (v závislosti na parametru příkazu SPEED).

Příkaz SPEED není příkladem ryze uživatelského přerušení. Můžeme sice ovlivňovat rychlost želvičky, ale obslužný podprogram zůstává v podstatě stále stejný pohyb želvy dopředu.

Jazyk Atari Logo však umožňuje vytvářet přerušení skutečně uživatelská, kdy se v závislosti na vzniklé situaci vykoná příkaz nebo posloupnost příkazů, zadaných uživatelem. K vyvolání přerušení můžeme využít těchto podmínek:

-dotyk dané želvy s čárou dané barvy,
-dotyk (kolize) dvou želv,
-změna stavu joysticku (páky nebo tlačítka),
-časové přerušení (každou sekundu).

Každa podmínka má svůj kód, tyto kódy udává přehledně tabulka v příloze 5.

K nastavení uživatelského přerušení slouží příkaz WHEN, který má tvar:

```
WHEN k l
```

kde k je kód podmínky a l je seznam příkazů definující obslužný podprogram. Kdykoliv nyní dojde k situaci s daným kódem, přeruší se původní činnost a provede se seznam příkazů l. Poté se dále pokračuje v původní činnosti.

K zrušení přerušení s kódem k lze použít příkaz:

```
WHEN k []
```

Všechna nastavená přerušení rovněž ruší příkaz CS.

Takto lze například nakreslit na grafické obrazovce uzavřenou čáru, jakousi "ohrádku", do ní umístit želvu a nastavit přerušení typu želva 0 čára 0 (tj. s kódem 0):

```
WHEN 0 [RIGHT 180]
```

Uvedeme-li nyní želvu do pohybu příkazy:

```
PENUP
SETSP 5
```

bude se želva v uzavřeném prostoru pohybovat, aniž by ho opustila. Při kontaktu s ohrádkou se prostě otočí o 180 stupňů a bude v pohybu pokračovat. Pohyb můžeme ukončit stisknutím klávesy BREAK, přerušení zrušit příkazem CS.

Uvedené podmínky nemusíme využívat pouze ve spojení s přerušovacím systémem. Logická funkce COND, která má tvar:

```
COND k
```

nabývá hodnoty TRUE v okamžiku, kdy je splněna podmínka s kódem k. Takto lze i bez přerušení vyřešit předchozí úlohu:

```
TO OHRADKA
IF COND 0 [RIGHT 180 FD 1][FD 1]
WAIT 10
OHRADKA
END
```

Želvu nyní uvedeme do pohybu příkazem OHRADKA. Příkaz WAIT 10 (bude vysvětlen v kapitole 45) slouží k lepší synchronizaci činnosti příkazu OHRADKA.

Rozdíl mezi oběma řešeními je zřejmý. První řešení se opíralo o přerušení (SETSP a WHEN), pohyb želvy nebránil uživateli dále programovat (např. provádět výpočty, definovat

nové příkazy apod.). Druhé řešení se opírá o nekonečnou rekurzi bez přerušení. V tomto případě je interpret Logo plně vytížen a nelze již tedy provádět nic jiného.

*** POZOR ***
Nesnažme se používat současně obě podmínky (WHEN a COND). Obvykle dojde k chybné synchronizaci, a tím k zhroucení činnosti Loga.

44. Vstupní a výstupní zařízení

Jazyk Atari Logo obsahuje i příkazy pro práci s vnějšími zařízeními, vstupními i výstupními. Umožňuje jednak ukládat a načítat vytvořené procedury a hodnoty proměnných, jednak definovat zařízení pro vstup a výstup při komunikaci s jazykem Logo.

44.1 Čtení a zápis procedur

Pro zápis procedur a proměnných na vnější zařízení slouží příkaz SAVE který má tvar:

```
SAVE fn
```

kde fn je řetězec definující výstupní zařízení, resp. soubor. Správný zápis je tedy například:

```
SAVE "D:FUNKCE.LOG
```

Příkaz SAVE provede uložení všech uživatelsky definovaných procedur a proměnných. Můžeme jej rovněž použít pro výpis na tiskárnu jako:

```
SAVE "P:
```

Ke čtení procedur a proměnných z vnějšího zařízení slouží příkaz LOAD, který má tvar:

```
LOAD fn
```

kde fn je řetězec definující vstupní zařízení, resp. soubor. Správný zápis je například:

```
LOAD "C:
```

Jména všech načtených procedur se vypíší na obrazovku obdobně jako při jejich definici, např.:

```
PUNKCE DEFINED
```

44.2 Práce s disketovou jednotkou

Pro práci s disketovou jednotkou jsou určeny další dva příkazy CATALOG a ERF. Příkaz CATALOG má tvar:

CATALOG fn

kde fn je jméno zařízení. Příkaz slouží k výpisu adresáře disku, například:

CATALOG "D:

vypíše všechny soubory na disku 1.

Příkaz ERF má tvar:

ERF fn

kde fn je jméno souboru. Příkaz zruší na disku soubor s daným jménem, např.:

ERF "D8:DATA.PIC

zruší soubor DATA.PIC na ramdisku počítače 130 XE (256 XT).

44.3 Vstupní a výstupní soubory

Standardním vstupním i výstupním zařízením v Logu je obrazovkový editor (E:). Toto zařízení má jako vstup klávesnici a jako výstup obrazovku. Uživatel má však možnost kdykoliv zvolit jako vstup nebo výstup jiné zařízení, popř. soubor.
Standardní vstup můžeme změnit příkazem:

SETREAD fn

kde fn je specifikace nového vstupu. Například příkaz:

SETREAD "D:VSTUP.DAT

nastaví jako vstup diskový soubor VSTUP.DAT. Z tohoto souboru budou nadále čteny všechny údaje (např. pomocí příkazů RC, RL). Uzavření vstupního souboru a návrat k editoru (klávesnici) provádí příkaz:

SETREAD []

*** POZOR ***
Zadáme-li v přímém režimu nové vstupní zařízení, odpojí se klávesnice a počítač se stane neovladatelným.

Nové výstupní zařízení, resp. soubor volíme příkazem:

SETWRITE fn

kde fn je bližší specifikace výstupu. Například pro výstup na tiskárnu volíme příkaz:

```
SETWRITE "P:
```

Výstupní soubor uzavřeme a editor (obrazovku) zpět nastavíme příkazem:

```
SETWRITE []
```

Následující příkaz MOCNINY vytiskne na zadané výstupní zařízení druhé mocniny všech čísel od N do M:

```
TO MOCNINY :N :M :FN
SETWRITE :FN
SQUARE :N :M
SETWRITE []
END
```

kde pomocný příkaz SQUARE je:

```
TO SQUARE :N :M
PRINT :N * :N
IF :N = :M [STOP]
SQUARE :N + 1 :M
END
```

U jmen diskových souborů nelze používat zástupné znaky * a ? ("wild-cards") jako v jazyce Basic nebo v DOSu. Toto omezení se vztahuje na všechny příkazy uvedené v této kapitole.

45. Časové zpoždění

Jak již bylo uvedeno v kapitole 43, obsahuje Atari Logo příkaz WAIT pro časové zpoždění, které je realizováno jako pasívní čekání. Příkaz má tvar:

```
WAIT n
```

kde n je číselná hodnota vyjadřující zpoždění v násobcích 1/50 sekundy. Narazí-li interpret na příkaz WAIT, provádění příkazu se zastaví na dobu danou parametrem. Například ke zpoždění výpočtu v délce 4 sekund je možno použít příkaz:

```
WAIT 200
```

46. Výpisy

Pro lepší orientaci v uživatelsky vytvořených procedurách, proměnných a přerušeních slouží příkazy pro jejich výpisy. Jsou to:

PO r	výpis definice procedury se jmenem r
POALL	výpis všech definic a hodnot proměnných
POTS	výpis názvu všech procedur
POPS	výpis definic všech procedur
PONS	výpis hodnot všech proměnných
POD n	výpis nastaveneho přerušení s kódem n

PODS výpis všech nastavených přerušení

Posloupnost příkazů:

```
PO r1
PO r2
:
PO rn
```

lze nahradit jediným příkazem:

```
PO [r1 r2 ... rn]
```

Parametry r1, r2 az rn jsou jména procedur, například:

```
PO [CTVEREC FUNKCE KRUH]
```

Výpisy definic procedur, resp. hodnot proměnných se provádějí ve stejné formě jako v textovém editoru, tj. jako příkazy TO, resp. MAKE. Výpisy nastavených přerušení se provádějí ve formě příkazu WHEN. Není-li nastaveno žádné přerušení, výpis se neprovádí.

Následující příkaz TISK slouží k výpisu definice zvolené procedury, resp. procedur na daném zařízení:

```
TO TISK :F :S
SETWRITE :F
PO :S
SETWRITE []
END
```

Například k výpisu definice příkazu CTVEREC na tiskárnu použijeme zápis:

```
TISK "P: "CTVEREC
```

47. Rušení

Opakem příkazu výpisu jsou příkazy rušení. Slouží k zrušení definic uživatelských procedur nebo hodnot proměnných. V Atari Logu jsou tyto příkazy rušení:

ERASE r (ER)	zrušení definice procedury se jménem r
ERALL	zrušení všech definic a proměnných
ERPS	zrušení definic všech procedur
ERN v	zrušení hodnoty proměnné v
ERNS	zrušení hodnot všech proměnných

Více příkazů ERASE, resp. ERN lze nahradit jediným, použijeme-li jako parametr seznam parametrů všech jednotlivých názvů (řetězců), například:

```
ERN [A B C]
```

je totéž jako:

```
ERN "A
ERN "B
ERN "C
```

*** POZOR ***
Zrušit nelze primitiva jazyka Logo, tj. standardní, uživatelem nedefinované příkazy a funkce.

48. Práce s volnou pamětí

Při zpracování příkazů nebo funkcí jazyka Logo je k předávání parametru mezi příkazy, ukládání hodnot při rekurzi, apod. využívána volná paměť RAM. Tato část paměti je organizována do tzv. uzlu (nodes). Nemá smysl vysvětlovat v této příručce strukturu a význam volné paměti, neboť pro uživatele nejsou tyto informace nijak důležité.

Při provádění procedur jsou jednotlivé uzly volné paměti obsazovány, dokud nedojde k jejich úplnému vyčerpání. Počet právě volných (neobsazených) uzlů udává funkce NODES a lze je zjistit příkazem:

```
PRINT NODES
```

Dojde-li k úplnému vyčerpání volné paměti, nastane tzv. recyklace. Všechny uzly volné paměti se přitom projdou a ty, jejichž obsah již není aktuální, se opět uvolní. Recyklace paměti trvá asi 1 sekundu, na tuto dobu se také zastaví činnost příkazů nebo funkce.

Recyklaci lze provést i kdykoliv mimo tuto situaci příkazem:

```
RECYCLE
```

Při nesprávném návrhu rekurzivních procedur se může stát, že dojde k nekonečné rekurzi. Obsahuje-li rekurzivní procedura parametry, dojde k jejich postupnému ukládání, dokud není zaplněna celá volná paměť. V takovém případě je recyklace samozřejmě neúčinná (všechny uzly nesou aktuální informace) a dojde k zahlcení paměti.

(Těm z vás, kteří se s uvedenými informacemi o volné paměti nespokojili, doporučujeme přečíst kapitolu "Algoritmy čištění paměti" v příručce Speciální programovací jazyky od Josefa Koláře, kterou vydalo ediční středisko ČVUT.)

49. Direktivy

Speciálními procedurami jazyka Logo jsou tzv. direktivy (také řídící nebo systémové procedury). Jejich název vznikl od toho, že se obvykle používají pouze v přímém režimu (direct access), tj. mimo definice jiných procedur. Direktivy jsou od ostatních procedur odlišeny i tím, že jejich názvy začínají tečkou.

49.1 Výpis všech primitiv

K výpisu jmen všech primitiv slouží příkaz:

.PRIMITIVES

Primitiva jsou standardní, dále nerozložitelné příkazy nebo funkce (např. PRINT, SENTENCE, ST apod.).

49.2 Práce s pamětí

K přímému přístupu k paměťovým buňkám slouží funkce .EXAMINE a příkaz .DEPOSIT. Funkce .EXAMINE slouží ke zjištění hodnoty paměťové buňky. Je obdobou funkce PEEK z jazyka Basic a má tvar:

.EXAMINE adr

kde adr je adresa buňky. Příkaz .DEPOSIT slouží ke změně hodnoty paměťové buňky. Je obdobou příkazu POKE z jazyka Basic a má tvar:

.DEPOSIT adr val

kde adr je adresa buňky a val nová dosazovaná hodnota. Například basický příkaz:

POKE 710,PEEK(709)

nahradíme v Logu příkazem:

.DEPOSIT 710 .EXAMINE 709

Příkaz (direktiva):

.CALL adr

vyvolá podprogram ve strojovém kódu na adrese adr. Je tedy obdobou basického příkazu:

I=USR(adr)

49.3 Nastavení rozměru obrazovky

Poslední dvě direktivy jazyka Logo slouží k nastavení svislého rozměru grafické obrazovky (v krocích želvy). Vodorovný rozměr přitom zůstává stále stejný. Změnou svislého rozměru se snažíme docílit toho, aby nakreslený čtverec byl skutečně čtvercem a ne obdélníkem, aby kružnice nebyla elipsou, apod.

Svislý rozměr nastavujeme příkazem:

.SETSCR n

kde n je číslo, které vyjadřuje vztah mezi svislým rozměrem obrazovky v bodech (tj. 192) a v krocích:

$$n = \frac{\text{počet bodů}}{\text{počet kroků}}$$

Standardní hodnota je 0.8 (odpovídá 240 krokům) a lze ji kdykoliv zjistit jako hodnotu funkce .SCRUNCH, například příkazem:

```
PRINT .SCRUNCH
```

V diskové verzi jazyka LOGO (a též upravené kazetové) jsou mezi rozměry X, Y a hodnotou .SCRUNCH tyto vztahy:

.SCRUNCH	X	Y
0.8	320	240
1.0	320	192
2.0	320	96
0.5	320	384

Záporné hodnoty parametru .SCRUNCH způsobí, že vertikální osa bude orientována opačně, tj. orientace 0 (ve směru rostoucí osy y) bude svisle dolů.

*** POZOR ***
Uvedené hodnoty odpovídají diskové verzi jazyka. Jiné verze mohou mít rozměry obrazu a tím i hodnoty příkazu .SETSCR a funkce .SCRUNCH rozdílné od uvedených.

D. ZAJIMAVÉ PROJEKTY

50. Oblouky a kružnice

Tento projekt obsahuje návrhy vhodných procedur pro kreslení kružnic a jejich částí, tj. oblouku. Je vlastně rozšířením skupiny příkazů definovaných v kapitole 17.

```
TO OBLOUK.PRAVY :POLOMER :STUPNE
OBLOUKP1 .174532 * :POLOMER :STUPNE / 10
IF 0 = REMAINDER :STUPNE 10 [STOP]
FD .174532 * :POLOMER / 10 * REMAINDER :STUPNE 10
RT REMAINDER :STUPNE 10
END

TO OBLOUKP :POLOMER :STUPNE
OBLOUK.PRAVY :POLOMER :STUPNE
END
```

```
TO OBLOUK.LEVY :POLOMER :STUPNE
OBLOUKL1 .174532 * :POLOMER :STUPNE / 10
IF 0 = REMAINDER :STUPNE 10 [STOP]
FD .174532 * :POLOMER / 10 * REMAINDER :STUPNE
LT REMAINDER :STUPNE 10
END

TO OBLOUKL :POLOMER :STUPNE
OBLOUK.LEVY :POLOMER :STUPNE
END

TO OBLOUKP1 :KROK :KOLIKRAT
REPEAT :KOLIKRAT [RT 5 FD :KROK RT 5]
END

TO OBLOUKL1 :KROK :KOLIKRAT
REPEAT :KOLIKRAT [LT 5 FD :KROK LT 5]
END

TO KRUHL :POLOMER
OBLOUKL1 .174532 * :POLOMER 36
END

TO KRUHP :POLOMER
OBLOUKP1 .174532 * :POLOMER 36
END
```

Pozn.: uvedené procedury kreslí ve skutečnosti 36-ti úhelníky. Konstanta 0.174532 = 2*3.1416/36.

51. Cykly v Logu a jejich použití

Jazyk Logo neobsahuje ve skutečnosti klasické programové cykly, neboť je nahrazuje rekurzí. Chceme-li přesto cykly používat (například pro potřeby výuky), musíme si je definovat sami.

V teorii algoritmu rozlišujeme dva typy cyklu "while" s podmínkou na začátku a "repeat" s podmínkou na konci. Pascalský zápis:

```
while P do S
```

značí: dokud je splněna podmínka P, opakuj příkaz S. Podobně zápis:

```
repeat S until P
```

vyjadřuje: opakuj příkaz S do té doby, než je splněna podmínka P.

Oba dva cykly vyjádříme v Logu pomocí rekurze:

```
TO CYKLUS.WHILE :P :S
IF NOT RUN :P [STOP]
RUN :S
CYKLUS.WHILE :P :S
END
```

```
TO CYKLUS.REPEAT :S :P
RUN :S
IF RUN :P [STOP]
CYKLUS.REPEAT :S :P
END
```

Ještě si dodefinujeme k oběma příkazům zkratky:

```
TO CW :P :S
CYKLUS.WHILE :P :S
END

TO CR :S :P
CYKLUS.REPEAT :S :P
END
```

Parametry S (příkaz) a P (podmínka) zadáváme ve formě seznamu.

Nyní si ukážeme použití obou cyklů. Cyklus "while" použijeme k nerekurzivní definici funkce faktoriál:

```
TO FACT :N
MAKE "I 1
MAKE "F 1
CW [:I < :N][MAKE "I :I + 1 MAKE "F :F * :I]
OP :F
END
```

Pro N > 2 bychom mohli použít i cyklus "repeat":

```
TO FACT2 :N
MAKE "I 1
MAKE "F 1
IF :N < 2 [OP :F]
CR [MAKE "I :I + 1 MAKE "F :F * :I][:I = :N]
OP :F
END
```

V algoritmických jazycích se setkáváme i s cyklem typu "for", jehož zápis:

```
for I:=A to B do S
```

znamená: pro všechna čísla (celá) od A do B proveď příkaz S. Cyklus "for" je zvláštním případem cyklu "while":

```
TO CYKLUS.FOR :I :A :B :S
MAKE "I :A
CW [:I <= :B][RUN :S MAKE "I :I + 1]
END

TO CF :I :A :B :S
CYKLUS.FOR :I :A :B :S
END
```

Funkce faktoriál by v tomto případě vypadala takto:

```
TO FACT3 :N
MAKE "I 1
MAKE "F 1
CP "I 2 :N [MAKE "F :F * :I]
END
```

Pro srovnání na závěr uveďme rekurzivní definici:

```
TO FACT4 :N
IF :N < 2 [OP 1]
OP :N * FACT :N 1
END
```

Závisí již na programátorovi, které formě definice dá přednost. Vytříbenému logickému myšlení nejlépe odpovídá definice rekurzivní, čili poslední.

52. Autodráha – návrh hry

V následujícím projektu využijeme možnosti jazyka ATARI Logo, které nám nabízí v oblasti animované grafiky. Dva automobily budou závodit na dráze. Jejich rychlost i zvuk budou řízeny ovladači.

Nejprve pomocí editoru tvaru vytvoříme vzory vozů:

```
EDSH 1

...00...
.0.00.0.
.000000.
.0.00.0.
...00...
..0..0..
..0000..
..0000..
..0000..
..0000..
..0..0..
00.00.00
00.00.00
00000000
00.00.00
00.00.00

EDSH 2

..0000..
...00...
.0.00.0.
.000000.
.0.00.0.
...00...
..0000..
..0..0..
```

```
..0..0..
..0000..
..0000..
00.00.00
00000000
00000000
00.00.00
..0000..
```

Příkazem MAKE přiřadíme uvedené tvary proměnným AUTO1 a AUTO2, tím se nám také automaticky uloží na výstupní médium, až použijeme příkazu SAVE:

```
MAKE "AUTO1 GETSH 1
MAKE "AUTO2 GETSH 2
```

Na disketu je můžeme uložit hned:

```
SAVE D:AUTA
```

A zpět je pak vyvoláme příkazem:

```
LOAD "D:AUTA
```

Celá hra automobilových závodů sestává z následujících procedur:

Procedura ZAVODY je hlavní procedurou našeho programu. Volá přímo či nepřímo všechny ostatní procedury:

```
TO ZAVODY
NASTAV.DRAHU
NASTAV.STRET
JEDEME
END
```

Procedura NASTAV.CARU vytvoří jízdní dráhu a umístí vozy na startovní čáru:

```
TO NASTAV.DRAHU
TELL O HT
FS
ZAVODNIK1
ZAVODNIK2
ZAVODNI.DRAHA
CILOVA.PASKA
VYMALUJ.AUTA
END
```

Procedura ZAVODNIK1 dá první želvě tvar prvního vozu a umístí ji:

```
TO ZAVODNIK1
TELL 1
PUTSH 1 :AUTO1
PU ST
SETSH 1
```

```
SETPOS [-30 -95]
END
```

Následující procedura ZAVODNIK2 udělá totéž pro druhého závodníka:

```
TO ZAVODNIK2
TELL 2
PUTSH 2 :AUTO2
PU ST
SETSH 2
SETPOS [30 -95]
END
```

Nyní je uvedena procedura ZAVODNI.DRAHA, která nakreslí středovou čáru závodní dráhy:

```
TO ZAVODNI.DRAHA
SETBG 0
TELL 0
SETSP 150 WAIT 80 SETSP 0
END
```

a procedura CILOVA.PASKA, která nakreslí cíl:

```
TO CILOVA.PASKA
TELL 3 PU HT
SETPOS [-50 95]
PD
RT 90 SETPN 0
FD 100
SETH 0
END
```

Procedura VYMALUJ.AUTA auta vybarví. Zvolili jsme bílou barvu pro první a červenou pro druhý vůz:

```
TO VYMALUJ.AUTA
TELL 1
SETC 7
TELL 2
SETC 27
END
```

Nastavení hlídačů střetu některého z aut a cílové pásky provede procedura NASTAV.STRET. Jakmile se tak stane, barva pozadí se změní:

```
TO NASTAV.STRET
MAKE "BARVA BG
WHEN 4 [MAKE "BARVA :BARVA + 1 SETBG :BARVA IF BG = 127
SETBG 0]
WHEN 8 [MAKE "BARVA :BARVA + 1 SETBG :BARVA IF BG = 127
SETBG 0]
END
```

Následující procedura JEDEME řídí pohyby vozů a jejich zvuky:

```
TO JEDEME
PEDAL1
HLUK1
PEDAL2
HLUK2
JEDEME
END
```

Procedura PEDAL1 pohybuje prvním vozem. Jeho rychlost je pětinásobkem hodnoty, nastavené prvním ovladačem:

```
TO PEDAL1
TELL 1
MAKE "KVALT JOY 0
IF :KVALT < 0 [MAKE "KVALT :KVALT * -1]
SETSP :KVALT * 5
IF JOYB 0 [SETPOS -30 -95]
END
```

Hluk motoru prvního vozu určuje procedura HLUK1:

```
TO HLUK1
TOOT 0 (JOY 0) * 10 + 30 15 4
END
```

Obdobně jsou procedury PEDAL2 a HLUK2 pro druhý vůz:

```
TO PEDAL2
TELL 2
MAKE "KVALT JOY 1
IF :KVALT < 0 [MAKE "KVALT :KVALT * -1]
SETSP :KVALT * 5
IF JOYB 1 [SETPOS 30 -95]
END

TO HLUK2
TOOT 1 (JOY 1) * 10 + 30 15 4
END
```

A závody mohou začít. Každý hráč odstartuje tlačítkem svého ovladače. Pak jím řídí rychlost svého vozu.

53. Grafy zadaných funkcí

Tento projekt slouží k zobrazení grafu zadaných funkcí na grafické obrazovce pomocí jazyka Logo. Projekt se volá příkazem GRAF, který je navržen bez parametrů jako interaktivní.

```
TO GRAF
TS CT
VSTUP
```

```
NORMA
FS CS HT
OSY
NAKRES
END
```

Příkaz VSTUP slouží k zadání vstupních parametrů:

```
TO VSTUP
PR [Graf funkce y = f(x).]
PR
TYPE [f(x)=] MAKE "FX RL
INPUT [xmin=] "X1
INPUT [xmax=] "X2
INPUT [krok=] "DX
PR
INPUT [ymin=] "Y1
INPUT [ymax=] "Y2
END
```

K uložení číselných hodnot do proměnných slouží příkaz INPUT:

```
TO INPUT :TXT :VAL
TYPE :TXT
MAKE "VAL RL
MAKE "VAL FIRST :VAL
END
```

Příkaz NORMA určí koeficienty lineárních transformací mezi hodnotami a kroky:

```
TO NORMA
MAKE "A 300 / (:X2 - :X1)
MAKE "C 200 / (:Y2 - :Y1)
MAKE "B 150 - :A * :X2
MAKE "D 100 - :C * :Y2
END
```

Příkaz OSY nakreslí souřadnicové osy (v případě, že jsou viditelné):

```
TO OSY
IF :X1 * :X2 < 0 [OSAX]
IF :Y1 * :Y2 > 0 [OSAY]
END

TO OSAX
PU
SETPOS [-150 :D]
PD SETPN 1
SETPOS [150 :D]
END
TO OSAY
PU
SETPOS [:B -100]
PD SETPN 1
```

```
SETPOS [:B 100]
END
```

Vlastní kresbu křivky provede příkaz NAKRES:

```
TO NAKRES
PU
MAKE "X :X1
PLOT
PD SETPN 0
KRESLI
END

TO KRESLI
MAKE "X :X + :DX
IF :X > :X2 [STOP]
PLOT
KRESLI
END
```

Příkaz PLOT slouží k vykreslení jednoho kroku (bodu) průběhu funkce.

```
TO PLOT
MAKE "Y RUN :FX
SETPOS [(:A * :X + :B)(:C * :Y + :D)]
END
```

V tomto projektu si procedury předávaly hodnoty výhradně prostřednictvím globálních proměnných.

54. Množinové operace

Tento projekt slouží k názorné demonstraci práce s dvěma množinami. Jako množina se přitom chápe jednoúrovňový seznam libovolných prvků (tj. prvky nesmí být opět seznamy). Projekt se může využít například ve výuce matematiky. Volá se příkazem MNOZINA, který je opět jako v předešlém případě navržen jako interaktivní.

```
TO MNOZINA
IF NOT ZADANI [PR [Zadani neni regulerni.]][OPERACE]
END
```

Logická funkce zadání slouží k načtení obou množin. Pokud obě množiny jsou regulérně zadány, vrací funkce hodnotu TRUE, jinak vrací FALSE.

```
TO ZADANI
PRINT [Zadejte obe mnoziny.]
TYPE "X=
MAKE "X RL
TYPE "Y=
MAKE "Y RL
IF NOT STRUKTURA :X [OP FALSE]
IF NOT STRUKTURA :Y [OP FALSE]
```

```
ZJEDNODUSENI
OP TRUE
END
```

Funkce, která nevyhodnocuje pouze daný výraz, ale provádí i některé příkazy, se nazývá funkce s vedlejším efektem. V tomto případě je vedlejším efektem např. načtení obou množin.

Funkce STRUKTURA testuje daný vstup, zda jde skutečně o jednoúrovňový seznam.

```
TO STRUKTURA :S
IF NOT LISTP :S [OP FALSE]
IF EMPTYP :S [OP TRUE]
IF LISTP FIRST :S [OP FALSE]
OP STRUKTURA BF :S
END
```

Příkaz ZJEDNODUSENI vytváří a tiskne tzv. jednoduchý zápis množin X a Y, ve kterém se každý prvek množiny vyskytuje nejvýše jednou. Například množina zapsaná jako [1 2 3 2 3 1 3] má jednoduchý zápis například [1 2 3]. Vlastní operaci zjednodušení provádí funkce SINGLE.

```
TO ZJEDNODUSENI
MAKE "X SINGLE :X
MAKE "Y SINGLE :Y
PR [Jednoduchy tvar mnozin:]
(PR "X= :X)
(PR "Y= :Y)
END

TO SINGLE :S
IF EMPTYP :S [OP :S]
IF MEMBERP FIRST :S BF :S [OP SINGLE BF :S]
OP PPUT FIRST :S SINGLE BF :S
END
```

Příkaz OPERACE (bude proveden v případě regulérního zápisu obou množin) provede s množinami X a Y všechny základní množinové operace.

```
TO OPERACE
IF EQUAL :X :Y [PR [Mnoziny X a Y jsou shodne.]]
IF INCL :X :Y [PR [X je podmnozinou Y.]
IF INCL :Y :X [PR [Y je podmnozinou X.]
(PR "X+Y= UNION :X :Y)
(PR "X.Y= INTER :X :Y)
(PR "X-Y= MINUS :X :Y)
(PR "Y-X= MINUS :Y :X)
END
```

Množinové operace jsou realizovány těmito funkcemi:

EQUAL	test shodnosti množin
INCL	test podmnožiny
UNION	sjednocení množin
INTER	průnik množin

MINUS rozdíl množin (doplněk)

```
TO EQUAL :A :B
OP AND (INCL :A :B)(INCL :B :A)
END

TO INCL :A :B
IF EMPTYP :A [OP TRUE]
IF MEMBERP FIRST :A :B [OP INCL BF :A :B]
OP FALSE
END

TO UNION :A :B
IF EMPTYP :A [OP :B]
IF MEMBERP FIRST :A :B [OP UNION BF :A :B]
OP FPUT FIRST :A UNION BF :A :B
END

TO INTER :A :B
IF EMPTYP :A [OP []]
IF NOT MEMBERP FIRST :A :B [OP INTER BF :A :B]
OP FPUT FIRST :A INTER BF :A :B
END

TO MINUS :A :B
IF EMPTYP :A [OP []]
IF MEMBERP FIRST :A :B [OP MINUS BF :A :B]
OP FPUT FIRST :A MINUS BF :A :B
END
```

55. Třírozměrná grafika

Tento projekt umožňuje zobrazovat z různých pohledů třírozměrné objekty tvořené rovnými čárami bez zjišťování viditelnosti. Projekt vznikl úpravou článku TRZECI WYMIAR z časopisu Bajtek 1/86.

Princip zobrazení třírozměrné grafiky do roviny spočívá v transformaci tří prostorových souřadnic do dvou rovinných. Transformace je jednoznačně dána orientací jednotlivých os (KATX, KATY, KATZ) a poměry, v kterých se jednotlivé souřadnice budou promítat do roviny (PX, PY, PZ). Pro nastavení orientace os slouží příkaz AXES, pro nastavení promítacích poměrů příkaz PROJECT.

```
TO AXES :X :Y :Z
MAKE "KATX :X
MAKE "KATY :Y
MAKE "KATZ :Z
END

TO PROJECT :X :Y :Z
MAKE "PX :X
MAKE "PY :Y
MAKE "PZ :Z
END
```

Příkaz LOOK v závislosti na parametru automaticky nastavuje jeden z několika základních druhů promítání:

N normální (rovnoběžné),
a axonometrické,
H horní Mongeovo,
B boční Mongeovo,
P přední Mongeovo,
T tříčtvrteční.

```
TO LOOK :N
IF MEMBERP :N [N NORMAL] [AXES 90 0 225 PROJECT 1 1 0.5 CS
HT STOP]
IF MEMBERP  :N [A AXONOMETRIE] [AXES 120 0 240 PROJECT 1 1
1 CS HT STOP]
IF MEMBERP :N [H HORNI] [AXES 90 0 180 PROJECT 1 0 1 CS HT
STOP]
IF MEMBERP  :N [B  BOCNI] [AXES 0 0 90 PROJECT 0 1 1 CS HT
STOP]
IF MEMBERP :N [P PREDNI] [AXES 90 0 0 PROJECT 1 1  0 CS HT
STOP]
IF MEMBERP  :N [T  TRICTVRTE] [AXES  100 0 235 PROJECT 1 1
0.65 CS HT STOP]
PRINT [Neznamy pohled!]
END
```

Pro třírozměrnou grafiku je nutno vybudovat nové příkazy pro kreslení základních čar. V projektu je kromě obecné čáry (příkaz VEKTOR :X :Y :Z, kde parametry označují přírůstky v jednotlivých osách) možno kreslit přímo čáry ve směru jednotlivých os:

VPRAVO :N - pohyb vpravo o N kroků,
VLEVO :N - pohyb vlevo o N kroků,
NAHORU :N - pohyb nahoru o N kroků,
DOLU :N - pohyb dolů o N kroků,
DOPREDU :N - pohyb dopředu o N kroků,
DOZADU :N - pohyb dozadu o N kroků.

```
TO VEKTOR :X :Y :Z
MAKE "OLD POS
HOP [VPRAVO :X VLEVO :Y DOPREDU :Z]
MAKE "NEW POS
HOP [SETPOS :OLD]
SETPOS :NEW
END

TO VPRAVO :L
SETH :KATX
FD :L * :PX
END

TO VLEVO :L
VPRAVO -1 * :L
END
```

```
TO NAHORU :L
SETH :KATY
FD :L * :PY
END

TO DOLU :L
NAHORU -1 * :L
END

TO DOPREDU :L
SETH :KATZ
FD :L * :PZ
END

TO DOZADU :L
DOPREDU -1 * :L
END

TO HOP :CO
PU RUN :CO PD
END
```

Příkaz HOP slouží k přesunu písátka (kreslení se zdviženým perem). Tento příkaz je použit v definici procedury VEKTOR, může však být použit i samostatně.

K zjednodušení definic slouží další 4 příkazy pro kreslení obdélníku (PLOCHA.XY, PLOCHA.XZ, PLOCHA.YZ) a kvádru (KVADR.XYZ) orientovaných ve směru souřadnicových os.

```
TO PLOCHA.XY :X :Y
NAHORU :Y VPRAVO :X
DOLU :Y VLEVO :X
END

TO PLOCHA.XZ :X :Z
DOPREDU :Z VPRAVO :X
DOZADU :Z VLEVO :X
END

TO PLOCHA.YZ :Y :Z
NAHORU :Y DOPREDU :Z
DOLU :Y DOZADU :Z
END

TO KVADR.XYZ :X :Y :Z
PLOCHA.XY :X :Y
PLOCHA.XZ :X :Z
HOP [DOPREDU :Z]
PLOCHA.XY :X :Y
HOP [DOZADU :Z NAHORU :Y]
PLOCHA.XZ :X :Z
HOP [DOLU :Y]
END
```

Pro kreslení obrazců orientovaných mimo souřadnicové osy je možno použít příkaz ROTACE, pro zvětšení nebo zmenšení

obrazu příkazy ZVETSENI a ZMENSENI:

```
TO ROTACE :N
MAKE "KATX :KATX + :N
MAKE "KATY :KATY + :N
MAKE "KATZ :KATZ + :N
END

TO ZVETSENI :K
PROJECT :K * :PX :K * :PY :K * :PZ
END

TO ZMENSENI :K
ZVETSENI 1/:K
END
```

Ve zmíněném článku jsou uvedeny i náměty na další zajímavé příkazy, které si může uživatel vytvořit podle své potřeby. Následující příklady představují jen zlomek z možností využití zmíněného projektu pro třírozměrnou grafiku:

```
TO PRIKLAD1
HOP [HOME]
SETPN 0
VPRAVO 50 HOP [HOME]
SETPN 1
NAHORU 50 HOP [HOME]
SETPN 2
DOPREDU 50 HOP [HOME]
END

TO PRIKLAD2
REPEAT 10 [PLOCHA.XZ 30 50 HOP [NAHORU 8]]
END

TO PRIKLAD3
SETPN 0
KVADR.XYZ 30 70 50
HOP [NAHORU 70]
SETPN 1
KVADR.XYZ 30 15 50
HOP [DOLU 70 VPRAVO 30]
SETPN 2
KVADR.XYZ 40 45 50
END

PRIKLAD4
SETPN 0
KVADR.XYZ 50 50 50
SETPN 1
ROTACE 45
KVADR.XYZ 50 50 50
SETPN 2
ROTACE -45
ZVETSENI 1.5
KVADR.XYZ 50 50 50
ZMENSENI 1.5
END
```

Před kreslením je nutno nejprve nastavit koeficienty transformace, poté teprve začít vlastní kreslení, např.:

```
LOOK "T
PRIKLAD1
```

nebo:

```
AXES 0 90 235
PROJECT .7 .8 1
PRIKLAD 3
```

56. Jídelní lístek

Projekt jídelního lístku je ukázkou manipulace s řetězci a je převzat z knihy P. Goodyeara: Logo.

Předpokládejme, že jídlo se skládá ze tří položek:

polévka nebo předkrm (START),
hlavní jídlo (HLAVNI),
zákusek (DESERT).

Na vzajemné kombinace nebudeme klást žádná omezení, rovněž nebudeme přihlížet k vlastnostem jednotlivých pokrmů (kalorická hodnota, obsah cukru apod.).

Základem projektu jsou tři seznamy START, HLAVNI a DESERT, které obsahují jména všech nabízených jídel, resp. jejich složek. K vytvoření náhodně sestaveného menu použijeme příkaz:

```
TO MENU
TYPE "Predkrm:
PR VYBER START
TYPE "Hlavni:
PR VYBER HLAVNI
TYPE "Zakusek:
PR VYBER DESERT
PR [Dobrou chut!]
END
```

Funkce VYBER slouží k náhodnému výběru položky ze seznamu:

```
TO VYBER :X
OP ITEM (RANDOM COUNT :X) :X
END
```

kde ITEM je výběrová funkce definovaná jako:

```
TO ITEM :N :L
IF :N = 1 [OP FIRST :L]
OP ITEM (:N - 1) (BF :L)
END
```

Seznamy START, HLAVNI a DESERT můžeme vytvořit pomocí příkazu MAKE, například:

```
MAKE "START [[hovezi polevka][slepici polevka][sunka
s okurkou]]
MAKE "HLAVNI [[veprova knedlik zeli][znojemska hovezi
ryze][sekana brambor]]
MAKE "DESERT [[puding][sachr se slehackou]]
```

Elegantnější řešení však předpokládá použití editoru, který by měl mít tyto funkce:

výpis všech položek (LIST),
přidání položky (INSERT),
zrušení položky (DELETE).

Všechny tři procedury by měly být konverzačního (dialogového) typu. Jejich možný návrh uvádí následující část projektu:

```
TO LIST
PR [Ktery seznam chcete vypsat?]
MAKE "NAME RL
PR [Tento seznam obsahuje:]
VYPIS :NAME
END

TO VYPIS :S
IF EMPTYP :S [STOP]
PRINT FIRST :S
VYPIS BF :S
END

TO INSERT
PR [Do kterehо seznamu chcete pridat?]
MAKE "NAME RL
PR [Kterou polozku chcete pridat?]
MAKE "POL RL
IF MEMBERP :POL :NAME [PR [Polozka jiz existuje.] STOP]
PPUT :POL :NAME
END

TO DELETE
PR [Z kterehо seznamu chcete vyjmout?]
MAKE "NAME RL
PR [Kterou polozku chcete rusit?]
MAKE "POL RL
IF NOT MEMBERP :POL :NAME [PR [Polozka neexistuje.] STOP]
MAKE "NAME VYJMOUT :POL :NAME
END

TO VYJMOUT :X :S
IF :S = [] [OP []]
IF :X = FIRST :S [OP BF :S]
OP FPUT (FIRST :S)(VYJMOUT :X BF :S)
END
```

Tento projekt by bylo možno rozšířit o výběrové procedury, které by prováděly výběr na základě vlastností jednotlivých pokrmů (např. obsah cukru, tuku, kalorická hodnota,apod.). Tyto

vlastnosti by mohly tvořit vertikální seznam, na jehož vrcholu by byl uveden symbolicky název pokrmu a jeho hodnotou by byl seznam vlastností, například:

```
HP
:
[[Hovezi polevka] FALSE TRUE 356]
```

HP je jméno proměnné, které bychom ukládali jako prvek seznamu START. Hodnotou proměnné HP je seznam, který obsahuje postupně:

```
Hovezi polevka - přesný název,
FALSE          - obsah cukru,
TRUE           - obsah tuku,
356            - kalorická hodnota.
```

Takto obohacený projekt se již svými parametry blíží programům v jazycích Lisp a Prolog a spadá již plně do oblasti umělé inteligence.

PŘÍLOHY

1. Abecední seznam příkazů a funkcí

```
n       číselný parametr (např. 62, 4.2E+4)
s       slovo, řetězec (např. TRUE, 3456, "X)
l       seznam (např. [], [A B], [[1 2][]])
p       logická hodnota (TRUE, FALSE)
->      výsledná hodnota funkce
nsl     parametr může být více typů
(PRIKAZ par ...) příkaz o proměnném počtu parametrů
```

AND p p -> p
(AND p ...) -> p
Logický součin dvou a více parametrů.

ASCII s -> n
Kód ASCII znaku s.

ASK nsl l2
Lokální volba želvy. Příkazy určené seznamem l2 se provedou se všemi želvami, které jsou uvedeny v prvním parametru. Aktuální želva se tímto příkazem nemění.

BACK n BK n
Pohyb želvy směrem vzad o n kroků. V případě záporného parametru se želva pohybuje dopředu.

BG -> n
Aktuální hodnota barvy pozadí (0 až 127).

BUTFIRST sl -> sl BF
Seznam, který vznikne odebráním prvního prvku ze vstupního seznamu nebo řetězec bez prvního znaku.

BUTLAST sl -> sl BL
Seznam, který vznikne odebráním posledního prvku ze vstupního seznamu nebo řetězec bez posledního znaku.

CATALOG s
Adresář zařízení určeného parametrem.

CHAR n -> s
Znak o kódu n. Inverzní funkce k funkci ASCII.

CLEAN
Vymazání obrazovky. Pozice želvy se nemění.

COLOR -> n
Barva aktuální želvy.

COND n -> p
Je-li nastavena podmínka o kódu n, je hodnotou funkce TRUE, jinak FALSE.

COS n -> n
Standardní aritmetická funkce (jako v BASICu). Úhel je vyjádřen ve stupních. Parametr funkce nemusí být v závorce, např.:
PRINT COS 90

COUNT sl -> n
Počet prvků daného seznamu.

CS
Inicializace grafické obrazovky. Kombinace příkazů CLEAN a HOME. Nemění stav pera, viditelnost želvy a režim (WRAP/-WINDOW).

CT
Příkaz k vymazání textové obrazovky. Grafická obrazovka zůstává beze změn.

CURSOR -> l
Souřadnice textového kurzoru. Funkce vrací seznam ve tvaru [X Y].

DOT l
Příkaz k nakreslení bodu o daných souřadnicích. Parametr je seznam dvou hodnot [X Y]. Příkaz DOT nemění umístění želvy.

EACH l
Oslovení všech želv najednou. Příkazy uvedené v seznamu l provedou naráz všechny želvy.

EDIT sl ED
Volání editoru příkazů (seznamu příkazů). Všechny editované příkazy musí končit slovem END. Editor opustíme klávesou <ESC>.

EDNS
Vyvolání editoru hodnot proměnných. Editor opouštíme klávesou <ESC>.

EDSH n
Volání editoru tvaru želvy. Po vytvoření nového tvaru návrat pomocí <ESC>. Parametr může být z rozsahu 1 až 16.

EMPTYP sl -> p
Test, zda daný parametr je prázdný seznam.

END
Konec definice nového příkazu.

EQUALP nsl nsl -> p
Test shodnosti dvou objektů (čísel, slov, seznamů).

ERALL
Zrušení všech definic a proměnných.

ERASE sl ER sl
Zrušení vybraných definic.

ERF s
Zrušení souboru určeného parametrem s, např.:
ERF "D:PUNKCE.LOG

ERN sl
Zrušení vybraných proměnných.

ERNS
Zrušení hodnot všech proměnných.

ERPS
Zrušení všech definic.

FIRST sl -> sl
První prvek daného seznamu nebo první znak řetězce.

FORWARD n FD n
Pohyb želvy směrem vpřed o n kroků. V případě záporného parametru se želva pohybuje dozadu.

FPUT nsl l -> l
Funkce, která přidá daný objekt jako první prvek seznamu. Např.:
FPUT "A [B C] -> [A B C]

FS
Nastavení režimu grafické obrazovky.

GETSH n -> l
Seznam hodnot odpovídající tvaru želvy dané hodnoty.

HEADING -> n
Funkce určující orientaci želvy ve stupních (nahoru=0, vpravo=90, dolů=180, vlevo=270).

HOME
Přesun želvy do výchozího postavení, tj. doprostřed obrazovky s otočením směrem nahoru.

HT
Potlačení zobrazení (viditelnosti) želvy. Všechny grafické procedury se vykonávají beze změn, ale vyšší rychlostí.

IF p l
Příkaz jednoduchého větvení. Je-li splněna podmínka p, provede se seznam příkazů l.

IF p l1 l2
Příkaz dvojitého větvení. Je-li splněna podmínka p, provede se seznam příkazů l1, jinak se provede seznam příkazů l2.

IF p l1 l2 -> nsl
Podmíněná funkce. Seznamy l1 a l2 vyjadřují funkce (stejného

výsledného typu). Je-li splněna podmínka p, bude výsledkem hodnota funkce l1, jinak hodnota funkce l2.

INT n -> n
Celá část čísla n (zaokrouhluje vždy dolů).

JOY n -> n
Poloha joysticku číslo n. Polohy jsou kódovány takto:

```
 7 0 1
  \:/
6-(-1)-2
  /:\
 5 4 3
```

JOYB n -> p
Test stisknutí tlačítka joysticku n.

KEYP -> p
Test stisknutí klávesy. Je-li klávesa stisknuta, je hodnota funkce TRUE.

LAST sl -> sl
Poslední prvek daného seznamu nebo poslední znak řetězce.

LEFT n LT n
Rotace želvy vlevo o n stupňů. V případě záporného parametru se želva otáčí vpravo.

LIST nsl nsl -> l
(LIST nsl ...) -> l
Funkce, která z daných objektů vytvoří seznam. Jsou-li objekty seznamy, budou tvořit prvky nového seznamu. Např.:

```
LIST [1 2][3 4] -> [[1 2][3 4]]
```

LISTP nsl -> p
Test, zda daný parametr je seznam.

LOAD s
Čtení příkazů, funkcí a hodnot proměnných ze zařízení určeného parametrem s, např.:

```
LOAD "D:FUNKCE.LOG
```

MAKE s nsl
Vytvoření proměnné dané hodnoty (dosazovací příkaz). Např.:

```
MAKE "X "A
MAKE "A "B
MAKE :A 4.28
```

V zápisech představuje " označení jména, : označení hodnoty. Výše uvedená přiřazení vytvořila strukturu:

```
X
:
A
:
B
```

4.28

MEMBERP nsl l -> p
Test, zda daný objekt je prvek daného seznamu l.

NAMEP s -> p
Test, zda dané slovo je jménem nějaké proměnné (má-li hodnotu).

NODES -> n
Velikost volné paměti v uzlech stavového grafu.

NOT p -> p
Logická negace.

NUMBERP nsl -> p
Test, zda daný parametr je číslo (vyjadřuje číselnou hodnotu).

OR p p -> p
(OR p ...) -> p
Logický součet dvou a více parametrů.

OUTPUT nsl OP
Příkaz výstupu funkce. Parametr příkazu je výstupní hodnotou funkce definované pomocí TO. Tento příkaz ukončuje činnost daného bloku (funkce), proto musí být uveden jako poslední, např.:

```
TO POW :X
OUTPUT :X * :X
END

TO PYTHAGOR :A :B
OUTPUT SQRT (POW :A + POW :B)
END
```

PADDLE n -> n
Poloha paddle číslo n (hodnota 0 až 228).

PADDLEB n -> p
Test stisknutí tlačítka paddle n.

PC n -> n
Aktuální hodnota barvy pera (parametr je číslo pera 0 až 2).

PE
Pero v režimu gumy. Všechny kresby jsou stopou želvy mazány.

PEN -> s
Funkce určující stav želvy. Možné funkční hodnoty: PD, PE, PX, PU.

PEN n -> n
Poloha světelného pera. Parametr udává souřadnici (0=x, 1=y).

PENDOWN PD
Základní (výchozí) pozice pera. Při pohybu želvy vzniká barevná stopa.

PENUP PU
V tomto režimu nevzniká při pohybu želvy žádná kresba.

PN -> n
Hodnota aktuálního pera (0 až 2).

PO sl
Výpis definic daných parametrem.

POALL
Výpis všech definic a hodnot proměnných.

POD n
Výpis nastavených přerušení s kódem n.

PODS
Výpis všech nastavených přerušení.

PONS
Výpis všech proměnných a jejich hodnot.

POPS
Výpis všech definic.

POS -> l
Funkce určující umístění želvy ve formě seznamu [X Y].

POTS
Výpis všech názvů definic.

PRINT nsl PR
(PRINT nsl ...)
Příkaz tisku. Při tisku seznamu se vynechávají vnější závorky. Více příkazů tisku na jediný řádek za sebou lze nahradit jediným:

```
     TYPE :A
     TYPE :B
     PRINT "Ahoj
```

lze psát jako:

```
     (PRINT :A :B "Ahoj).
```

PRODUCT n n -> n
(PRODUCT n ...) -> n
Součin čísel daných parametry.

PUTSH n l
Definice tvaru želvy pomocí seznamu (l je seznam 16 čísel od 0 do 255).

PX
Pero v režimu inverze. Kresby se mažou, prázdná místa vyplňují.

RANDOM n -> n
Generátor náhodného celého čísla z intervalu <0,n-1>.

RC -> s
Čtení znaků z klávesnice. Příkaz čeká na první stisknutou klávesu.

RECYCLE
Příkaz k obnově volné paměti (jinak se obnova provádí automaticky při malé hodnotě NODES).

REMAINDER n n -> n
Zbytek po celočíselném dělení (modulo).

RERANDOM
Nastavení generátoru náhodných čísel do výchozích podmínek.

REPEAT n l
Příkaz cyklu. Seznam příkazů l se opakuje n krát. Odpovídá cyklu typu FOR. Ostatní typy cyklu je nutno řešit rekurzí.

RIGHT n RT n
Rotace želvy vpravo o n stupňů. V případě záporného parametru se želva otáčí vlevo.

ROUND n -> n
Zaokrouhlení čísla n na celé (podle pravidel zaokrouhlování).

RL -> l
Čtení seznamu znaků z klávesnice. Seznam je ukončen stisknutím klávesy <RETURN>.

RUN l
Interpretace obecného seznamu jako příkazu.

RUN l -> nsl
Interpretace obecného seznamu jako funkce.

SAVE s
Zápis příkazů, funkcí a hodnot proměnných na zařízení určené parametrem, např.:
    SAVE "C:

SENTENCE nsl nsl -> l SE
(SENTENCE nsl ...) -> l
Funkce, která z daných objektů vytvoří seznam. Jsou-li objekty seznamy, jde o jejich zřetězení. Např.:
    SE [1 2][3 4] -> [1 2 3 4]

SETBG n

Nastavení barvy pozadí (hodnoty od 0 do 127 0=černá, 7=bílá).

SETC n
Nastavení barvy želvy.

SETCURSOR l
Nastavení textového kurzoru (jako POSITION). Parametr je seznam ve tvaru [X Y].

SETENV n1 n2
Nastavení dozvuku kanálu n1. Parametr n2 udává, za kolik 1/50 s má být snížena hlasitost o jednotku.

SETH n
Nastavení želvy do daného směru (parametr ve stupních nahoru=0, vpravo=90, dolů=180, vlevo=270).

SETPC n n
Nastavení barvy (2. par.) pro příslušné pero (1. par.).

SETPN n
Volba aktuálního pera (0 až 2). Pera se liší barvou, předešlé příkazy platí pro libovolné aktuální pero.

SETPOS l
Nastavení želvy na dané místo na obrazovce (směr želvy se nemění). Parametr má tvar [X Y].

SETREAD s
Nastavení nového vstupního zařízení (namísto klávesnice) otevření kanálu pro vstup. Kanál uzavřeme operací:
SETREAD []

SETSH n
Nastavení tvaru želvy (0 az 16).

SETSP n
Určení rychlosti pohybu aktuální želvy. Želva se pohybuje nezávisle na příkazech z klávesnice. Při záporném n se želva pohybuje dozadu.

SETWRITE s
Nastavení nového výstupního zařízení (současně s obrazovkou) otevření kanálu pro výstup, např.:
SETWRITE "P:

SETX n
Posun želvy ve vodorovném směru tak, aby x-ová souřadnice měla hodnotu n. Druhá souřadnice se nemění.

SETY n
Posun želvy ve svislém směru tak, aby y-ová souřadnice měla hodnotu n. Druhá souřadnice se nemění.

SHAPE -> n
Aktuální hodnota tvaru želvy (0=standard, 1 až 16=uživatelský).

SHOW nsl
(SHOW nsl ...)
Příkaz tisku (jako PRINT). Seznamy se tisknou i s vnějšími závorkami.

SHOWNP -> p

Funkce, která zjišťuje viditelnost želvy. Pokud je želva viditelná, je výsledek TRUE, v opačném případě je výsledek FALSE.

SIN n -> n
SQRT n -> n
Aritmetické funkce (jako v BASICu). Úhel je vyjádřen ve stupních. Parametr funkce nemusí být v závorce, např.:

```
PRINT SIN 90
```

SPEED -> n
Rychlost pohybu aktuální želvy.

SS
Nastavení režimu smíšené (výchozí) obrazovky tj. grafické s textovým okénkem.

ST
Nastavení zobrazení (zviditelnění) želvy. Je-li obrazovka v textovém režimu, přepíná se automaticky do smíšeného.

STOP
Předčasné ukončení provádění příkazu nebo funkce. Využívá se zejména k ukončení rekurzí.

SUM n n -> n
(SUM n ...) -> n
Součet čísel, daných parametry.

TELL nsl
Volba aktuální želvy (globální). Parametr n je v rozmezí 0 až 3. Standardní výchozí hodnota je 0. Parametr ve formě seznamu slouží k aktualizaci více želv naráz, např.:

```
TELL [0 2 3]
```

THING s -> nsl :
Aktuální hodnota proměnné. Zápis:

```
THING "ALPA
```

lze zkrátit jako:

```
:ALFA.
```

TO s par1 par2 ...
Hlavička definice nového slova (příkazu). Atom s vyjadřuje jméno příkazu, par1 ... jsou parametry. Definici ukončujeme slovem END. Např.:

```
TO RECTANGLE :X :Y
FD :X RT 90
FD :Y RT 90
FD :X RT 90
FD :Y RT 90
END
```

TOOT n1 n2 n3 n4
Příkaz zvukového výstupu:
n1 číslo kanálu (0 nebo 1)
n2 výška tónu (v 5/6 Hz)

n3 hlasitost
n4 délka (v 1/50 s)

TS
Nastavení režimu textové obrazovky.

TYPE nsl
(TYPE nsl ...)

Příkaz tisku (jako PRINT). Po ukončení příkazu nedochází k přechodu na nový řádek.

WAIT n
Zastavení činnosti algoritmu na n*1/50 s, např.:
WAIT 200

WHEN n l
Nastavení přerušení s kódem podmínky n. Kdykoliv je podmínka splněna, vykoná se seznam příkazů l. Přerušení se ruší příkazem CS nebo WHEN n [].

WHO -> nl
Aktuální želva (želvy).

WINDOW
Režim zobrazení: Prostor pro pohyb želvy není prakticky omezen, zobrazena je však pouze část prostoru kolem nulového bodu ([0 0]).

WORD s s -> s
(WORD s ...) -> s
Funkce, která z daných slov vytvoří nové slovo (zřetězením). Např.:
WORD "Ah "oj -> "Ahoj

WORDP nsl -> p
Test, zda daný parametr je slovo (řetězec).

WRAP
Režim zobrazení: Prostor pro pohyb želvy je omezen velikostí obrazovky. Při překročení některého z rozměrů se želva objeví znovu na opačném konci obrazovky.

XCOR -> n
Aktuální x-ová souřadnice želvy.

YCOR -> n
Aktuální y-ová souřadnice želvy.

n + n -> n
n - n -> n
n * n -> n
n / n -> n
(n) -> n

Standardní aritmetické operace. Struktura aritmetického výrazu

se standardními operacemi odpovídá jazyku BASIC, např.:

```
     PRINT 3 * 2 + :X / 3
```

```
n = n   -> p
n > n   -> p
n < n   -> p
n >= n  -> p
n <= n  -> p
n <> n  -> p
```

Standardní logické relace (srovnání dvou čísel). Zápis relací odpovídá konvencím jazyka BASIC, např.

```
     IF :A <> 0 [PR "Ne]
```

.CALL n
Vyvolání strojového podprogramu na adrese n.

.DEPOSIT
Ekvivalent příkazu POKE, např.:

```
     .DEPOSIT 710 123
```

.EXAMINE n
Ekvivalent funkce PEEK, např.:

```
     MAKE "X .EXAMINE 756
```

.PRIMITIVES
Výpis jmen všech primitivních příkazů a funkcí (tj. dále nerozložitelných).

.SCRUNCH -> n
Aktuální rozměr obrazovky.

.SETSCR n
Nastavení rozměru obrazovky.

2. Přehled řídících kláves

BREAK	přeruší probíhající činnost, zruší posledně zadaný řádek
CTRL+	posuv kurzoru doleva
CTRL*	posuv kurzoru doprava
CTRL-	posuv kurzoru nahoru
CTRL=	posuv kurzoru dolů
CTRL 1	zastaví výpis na obrazovce, po druhém stisknutí ho opět spustí
CTRL a	nastaví kurzor na začátek řádku
CTRL CLEAR	vymaže zbytek řádku napravo od kurzoru
CTRL DELETE	vymaže znak, na kterém je kurzor
CTRL E	přemístí kurzor na konec řádku
CTRL F	vyhradí celou obrazovku pro grafiku
CTRL INSERT	vytvoří prázdný řádek, zbytek se posune nahoru
CTRL S	rozdělí obrazovku na část textovou a grafickou
CTRL T	vyhradí celou obrazovku pro text
CTRL V	nová stránka na obrazovce
CTRL W	předchozí stránka
CTRL X	kurzor na začátek programu

CTRL Y	vyvolá obsah mazací vyrovnávací paměti
CTRL Z	přesune kurzor na konec editovaného programu
DELETE	vymaže znak nalevo od kurzoru
ESC	ukončí práci editoru
RETURN	ukončí editovaný řádek, nastaví kurzor na začátek dalšího řádku
SHIFT DELETE	vymaže zbytek řádku napravo od kurzoru
SHIFT INSERT	vloží mezeru na místo kurzoru
RESET	studený start jazyka Logo, obsah pracovní paměti se vynuluje

3. Seznam chybových hlášení

Vysvětlující české texty nejsou doslovnými překlady anglických chybových hlášení. Značky <a>, popř. <b> nahrazují hodnoty nebo jména procedur.

FILE <a> NOT FOUND
Soubor <a> není nalezen, daný disk jej neobsahuje.

I CAN'T OPEN <a>
Soubor <a> nemůže být otevřen, např. chybí-li dané zařízení, je-li chráněn proti zápisu apod.

I DON'T KNOW HOW TO <a>
Procedura <a> nebyla dosud definována.

NOT ENOUGH INPUTS TO <a>
Procedura <a> má nedostatečný počet vstupních parametrů.

NUMBER TOO BIG
Příliš vysoká číselná hodnota, např. při dělení nulou.

OUT OF SPACE
Plná paměť RAM. Oblast paměti vyhrazená pro uživatele je zcela zaplněna. Obvykle nastává při nekonečné rekurzi.

TOO MUCH IN ()'S
Uvedený výraz nemůže být uzavřen do závorek, popř. je příliš složitý.

UNEXPECTED ')'
Neočekávaná zavírací závorka, např. chybí-li otevírací.

YOU'RE AT TOPLEVEL
Na nejvyšší úrovni volání (popř. rekurze) byl proveden pokus o návrat do vyšší úrovně.

YOU DON'T SAY WHAT TO DO WITH <a>
Hodnota <a> nebyla zpracována, např. použijeme-li funkci namísto příkazu.

<a> DIDN'T OUTPUT TO <b>
Hodnota <a> nemůže být výstupem funkce <b>.

<a> DOESN'T LIKE <b> AS INPUT
Hodnota <b> nemůže být vstupem procedury <a>.

<a> HAS NO VALUE
Proměnná <a> nemá dosud definovanou hodnotu (obvykle při použití :<a> namísto "<a>).

<a> IS a BAD FILE NAME
Řetězec <a> nemůže být jméno souboru, např. je chybně vytvořen.

<a> IS a PRIMITIVE
Procedura <a> je standardní a nemůže být změněna, např. editací.

<a> IS ALREADY DEFINED
Procedura <a> již byla definována a nelze ji předefinovat příkazem TO.

<a> IS NOT TRUE OR FALSE
Hodnota <a> není logického typu (TRUE nebo FALSE).

<a> STOPPED!
Běh procedury <a> byl zastaven klávesou BREAK.

..... IN <a>
Daná chybová situace nastala při zpracování procedury <a>, např.:

X HAS NO VALUE IN SCITANI

4. Tabulka barev

Nejnižší hodnota představuje nejtmavší odstín, nejvyšší nejsvětlejší.

0	černá
1 - 6	šedá
7	bílá
8 - 15	žlutá
16 - 23	oranžová
24 - 31	červená
32 - 39	růžová
40 - 47	fialová
48 - 55	modrofialová
56 - 63	modrá
64 - 71	modrá
72 - 79	světle modrá
80 - 87	tyrkysová
88 - 95	modrozelená
96 -103	zelená
104-111	žlutozelená
112-119	hnědá
120-127	zlatá

5. Tabulka kódů příkazu WHEN

kód	podmínka
0	želva 0 - pero 0
1	želva 0 - pero 1
2	želva 0 - pero 2
3	stisknutý trigger
4	želva 1 - pero 0
5	želva 1 - pero 1
6	želva 1 - pero 2
7	každou sekundu
8	želva 2 - pero 0
9	želva 2 - pero 1
10	želva 2 - pero 2
11	>>> nepoužito <<<
12	želva 3 - pero 0
13	želva 3 - pero 1
14	želva 3 - pero 2
15	změna joysticku
16	želva 0 - želva 3
17	želva 1 - želva 3
18	želva 2 - želva 3
19	želva 1 - želva 0
20	želva 2 - želva 0
21	želva 2 - želva 1

6. Srovnávací tabulky příkazů

V následujících odstavcích je srovnáván jazyk Atari Logo se třemi dalšími rozšířenými verzemi jazyka a se standardem TPN (Turtle Procedure Notation):

ATR ... Atari Logo
TPN ... standardní verze
SIN ... Sinclair Logo (ZX Spectrum)
TER ... Terrapin Logo (Apple, Commodore)
TIL ... Texas Logo (TI-99)

Přehled zahrnuje rozdíly v zápisech základních příkazů a funkcí Loga, přičemž si nedělá nárok na úplnost.

Grafika:

```
ATR:  FORWARD, BACK,      RIGHT, LEFT
TPN:  FORWARD, BACKWARD,  RIGHT, LEFT
SIN:  FORWARD, BACK,      RIGHT, LEFT
TER:  FORWARD, BACK,      RIGHT, LEFT
TIL:  FORWARD, BACK,      RIGHT, LEFT

ATR:  PENUP, PENDOWN, PENERASE
TPN:  PENUP, PENDOWN, RUBBER
SIN:  PENUP, PENDOWN, PENERASE
TER:  PENUP, PENDOWN, PENCOLOR 6
TIL:  PENUP, PENDOWN, PENERASE
```

```
ATR:  SETPC,      SETBG
TPN:  PENCOLOUR,  BACKGROUND
SIN:  SETPC,      SETBG
TER:  PENCOLOR,   BACKGROUND
TIL:  SETCOLOR,   BACKGROUND

ATR:  HT,          ST,          CLEAN, HOME
TPN:  HIDE,        REVEAL,      CLEAR, HOME
SIN:  HIDETURTLE,  SHOWTURTLE,  CLEAN, HOME
TER:  HIDETURTLE,  SHOWTURTLE,  CLEAN, HOME
TIL:  HIDETURTLE,  SHOWTURTLE,  -----, HOME

ATR:  SETX, SETY, SETH
TPN:  SETX, SETY, SETH
SIN:  SETX, SETY, SETHEADING
TER:  SETX, SETY, SETHEADING
TIL:  SX,   SY,   SETHEADING

ATR:  BG,            PC
TPN:  BACKGROUND?,   PENCOLOUR?
SIN:  BACKGROUND,    PENCOLOR
TER:  TURTLESTATE,   ---------
TIL:  -----------,   COLOR
```

Struktury:

Standard TPN používá strukturované příkazy REPEAT...AGAIN a IF...THEN...ELSE ve smyslu:

```
REPEAT
 szn
AGAIN
```

namísto:

```
REPEAT [szn]
```

a:

```
IF podm
THEN
 szn1
ELSE
 szn2
```

namísto:

```
IF podm [szn1][szn2]
```

Vstup a výstup:

```
ATR:  RC,             RL,        PRINT, TYPE
TPN:  READCHAR,       READ,      PRINT, ----
SIN:  READCHAR,       READLIST,  PRINT, TYPE
TER:  READCHARACTER,  REQUEST,   PRINT, PRINT1
TIL:  READCHAR,       READLINE,  PRINT, TYPE
```

Seznamy:

ATR:	FPUT,	SENTENCE,	COUNT
TPN:	PUTFIRST,	JOIN,	COUNT
SIN:	FPUT,	SENTENCE,	COUNT
TER:	FPUT,	SENTENCE,	-----
TIL:	FPUT,	SENTENCE,	-----

Funkce:

Některé verze Loga mají jako poslední znak logických funkcí znak "?" namísto "P", například:

WORD?, NUMBER?, KEY?

namísto:

WORDP, NUMBERP, KEYP

7. Použitá a doporučená literatura

[1] Friedrich, V., Sedláček, V.: Logo vyúka hrou. In: Mikropočítačova technika a výuka mládeže. Sborník ČSVTS, Praha 1987.

[2] Goodyear, P.: Logo a guide to learning through programming. Ellis Horwood, London 1984.

[3] Kolář, J.: Speciální programovací jazyky. FEL ČVUT, Praha 1976.

[4] Waligorsky, M.: Logo slownik minimum. In: Bajtek 3-7/86.

[5] Waligorsky, M.: Programowac moze kazdy. In: Bajtek 86-87 (prubezne).

[6] Waligorsky, M.: Trzeci wymiar. In: Bajtek 1/86.

[7] Apple Logo Reference Manual. Apple, 1982.

[8] Logo. BYTE 8/1982 (účelově zaměřené číslo).

[9] LSCI Atari Logo User's Manual. Atari, 1983.

ZÁVĚR

Kdo dočetl příručku až do konce a problém logického programování ho natolik zaujal, že by se o něm rád dozvěděl více, může se již nyní těšit na publikaci od LOGA K LISPU, která se bude zabývat dalšími problémy této zajímavé oblasti programování včetně aplikací umělé inteligence (např. symbolické derivování, úpravy matematických výrazů, překladače živých jazyků, expertní systémy, ...) a bude současně i příručkou programování v jazyce Atari Lisp 1.5.

OBSAH

Publikované zo súhlasom - viď Prohlášení představitelů AK Praha.

Igi/2021