1.Знак

Знакът е или цифра от 0 до 9 или главна буква от A до Z и от А до Я, или малка буква от a до z и от а до я, или е препинателен знак !, <, ?, & и др. Последователност от знакове образуват дума в Лого. някои знаковетрябва да се използват внимателно, тъй като имат специално значение. Когато за вход на процедура се използва знак се смята, че това е дума с дължина 1.

2. Букви главни и малки

По подразбиране Лого не прави разлика между малки и главни думи (ПЕТЪР и петър са идентични). За да се направи промяна, те. да се направи разлика между малки и главни букви се използва командата SETLOGO с вход [CASE 1]. (Команда SETLOGO трябва да се използва внимателно.

3. Променливи

Величина, която може да приема различни стойности, се нарича променлива. Всяка променлива има име и стойност.

а) Името на всяка променлива е последователност от знаци, несъдържащи интервали. Такава последователност а Лого се нарича дума. Когато се задава име на променлива, непосредствено пред него се поставя водещ знак кавички ("), за да се отличава от имената на процедурите.
Например: "ПИ, "Размер 13, "A&B и т.н. Самият водещ знак не е съставна част от името.

б) Когато трябва да се използва стойността на една променлива пред името й поставяме водещ знак двоеточие.
Например: "A - име на променлива;
:A - стойност на променливата

4. Задаване стойност на променлива

MAKE ; НАПРАВИ; НПР — команда с 2 входа
MAKE дума произволен_вход
(MAKE име стойност)
Дефинира променлива с име дума *и* стойност произволен вход, ако не е дефинирана променлива с такова име. В противен случай (променлива с име дума е дефинирана) изменя текущата стойност на"зададената променлива. j
Първият вход е име на променлива, а вторият - стойност за тази променлива.
Например, след изпълнението на командата:
MAKE"СТРАНА 10
променливата СТРАНА ще име стойност 10.
Вторият вход може да бъде записан и във вид на израз, който се пресмята преди изпълнението на командата
MAKE "СТРАНА 10
MAKE "ПЕРИМЕТЪР 4*:СТРАНА,
променливата ПЕРИМЕТЪР ще има стойност 4*10 =40.

Възможно е новата стойност да зависи от старата стойност:
MAKE "СТРАНА :СТРАНА/2

5.Извеждане на стойност на променлива

PRINT; PR ИЗВЕДИ — команда с 1 или п входа
PRINT произволен_вход
(PRINT произволен_вход1 произволен_вход2 ...)
(произволен вход= израз)
Ако входът на командата е списък от думи и числа, те се заграждат в квадратни скоби й се извеждат на един ред, разделени с по един интервал.
PRINT [ЗДРАВЕЙ, КАК СИ?]
ЗДРАВЕЙ, КАК СИ?
Ако входовете са повече от един командата PRINT, заедно с входовете й се загражда в кръгли скоби.
MAKE "LICE 100
(PRINT [ЛИЦЕТО НА КВАДРАТА Е:] :LICE)
ЛИЦЕТО НА КВАДРАТА Е: 100

6. Процедури

Операции с числа - Това са процедури, които приемат като входове едно или повече числа. Стойността, която връщат, също е число. Тази стойност се използва от Други операции или команди. Например, командите от костенурковата геометрия. *,+,-, /, ABS, ARCTAN, COS, DIFFERENCE, DIV, EXP, INT, LN, MOD, PRODUCT, QUOTIENT, RANDOM, RANDOMIZE, ROUND, SIN, SQRT, SUM, TAN

Предикати с числа - Операции, които приемат два входа. Стойността, която връщат е TRUE или FALSE в зависимост от това, дали е изпълнено определено условие. Тази стойност се използва от други процедури. Например, условните процедури. <, <=, >, >=, <>, =, EQUAL?, GREATER?, LESS?

Операции с думи и списъци - Тези операции приемат като входове думи и списъци. Стойностите, които се получават след изпълнението им се използват от други операции (с числа, с думи и списъци, с логически изрази) или от команди (за работа с файлове, за дефиниране и използване на променливи, структури контрол). <, <=, >, >=, <>, =, ASCII, BUTFIRST; BF, BUTLAST; BL, BUTMEMBER, CHAR, COUNT, EMPTY?, EQUAL?, FIRST, FPUT, FROMMEMBER, GREATER?, ITEM, LAST, LESS?, LIST, LIST?, LPUT, MEMBER, MEMBER?, NUMBER?, PICK, REPLACE, SENTENCE; SE, SHUFFLE, WORD, WORD?, REVERSE

Списъци от свойства - Няколко атрибута, наречени свойства, могат да бъдат обединени в именувана група - списък от свойства, който представлява специална структура от данни в Comenius Logo. Тази група включва процедури за работа със списъци от свойства. Има възможност за редактиране и изтриване на свойства. Допуска се променлива и списък от свойства да имат едно и също име, тъй като представляват независими

обекти в работната памет. BURIED, BURY,
BURYALL, EDIT; ED. ERALL, ERASE; ER, GPROP, PLIST, PPROP, PPS,
PROPS, REMPROP, SETBURY, SETPLIST, UNBURY

Предикати с логически изрази - Тази група включва специалните думи TRUE и FALSE, които означават логическите стойности вярно и невярно, и операции, чиито входове са логически изрази. Тези операции връщат стойност TRUE или FALSE, която се използва от други процедури Например, структурите за контрол. AND, FALSE, NOT, OR, TRUE

Дефиниране и използване на променливи - Тази група съдържа операции и команди, които дават възможност за дефиниране, модифициране, изтриване и "архивиране" на променливи. В много случаи се налага стойностите на променливите да се използват като входове на други операции или команди. В езика Logo няма различни типове променливи в зависимост от различните типове данни. BURIED,
BURY, BURYALL, DEC, EDIT; ED, ER , ERALL, ERASE; ER, INC, LET,
LOCAL, MAKE, NAME, NAME?, NAMES, SETBURY, THING, UNBURY

Структури за контрол - Всяка потребителска процедура се състои от няколко действия (стъпки). Често те не се изпълняват последователно: някои от тях се повтарят многократно, други се изпълняват само при определено условие. Вградените процедури, чрез които се задава в какъв ред да се изпълняват отделните стъпки, се наричат структури за контрол. Техните входове са думи и/или списъци от Logo инструкции.
APPLY, CASE, CATCH, DEBUG, ERROR, FILTER, FOR, FOREACH,
GENERATE, IF, IFF, IFFALSE; IFF, IFTRUE; I FT, IGNORE, MAKEERROR,
MAP, OUTPUT; OP, PAUSE, EPC, REPCOUNT; REPC, REPEAT, RUN,
STOP, TEST, THROW, TOPLEVEL, WAIT, WAITUNTIL, WHILE

Дефиниране и редактиране на процедури - Тази група съдържа операции и команди за дефиниране, редактиране, изтриване, "архивиране" на потребителски процедури. Има възможност за извеждане дефинициите (или само заглавията) на процедурите и за класифициране на процедурите - вградени или потребителски. Ефектът от изпълнението на повечетб процедури от тази група може да се постигне като се използва менюто на Прозореца на паметта. BURIED,BURY, BURYALL, DEFINE, DEFINED?, EDIT; ED,
END, ERALL, ERASE;
ER, PO, POTS, PRIMITIVE?, PROCS, TEXT, TO, PRIMITIVES

Костенуркова геометрия - Операциите и командите от тази група позволяват да се създават костенурки, да се указва как и какво да чертаят или пишат върху графичния екран, да се зареждат готови изображения, да се изтрива съдържанието на екрана, да се променя състоянието на моливите на костенурките (цвят, широчина, вид на следата). ALLTURTLES;
ALL, ASK, BACK; BK, CLEAN, CLEARSCREEN;CS/, DOT, DOTCOLOR, DRAW, EACH,
ERALL, ERASETURTLE;
ERTtlRTLE, FILL, FILLCOLOR; FC, FILLPATTERN; FP, FORWARD; FD,
GETIMAGE, GETSHAPE, GETWINDOW, HEADING, HIDETURTLE; HT,
HOME, INSIDE?, LEFT; LT, MAKETURTLE, OUTSIDE?, OVERLAP?,
OVERLAPPED, PATTERN, PEN, PENCOLOR; PC, PENDOWN; PD,
PENERASE; PE, PENREVERSE; PX, PENSTATE, PENUP; PU,
PENWIDTH; PW, PHASE, POLYGON, POLYGONLINE, POS,
PUTIMAGE,READLISTTURTLE; RLT, REORDER, RIGHT; RT, RLT,
SETFILLCOLOR;SETFC, SETFILLMODE, SETFILLPATTERN; SETFP,
SETHEADING;SETH, SETPATTERN, SETPEN,
SETPENCOLOR;SETPC,SETPENSTATE, SETPENWIDTH; SETPW, SETPHASE,
SETPHASEMODE,SETPOS, SETPW, SETSHAPE, SETSHAPECOLOR, SETTTEXT;
SETTT,SETX, SETXY, SETY, SHAPECOLOR, SHAPEPOS, SHOWN?,SHOWTURTLE;
ST, STAMP, TELL, TEXTSIZE, TOWARDS, TTEXT; TT,TURTLESTATE, WHO,
WINDOW, WRAP, XCOR, YCOR

Работа с образи • Процедурите от тази група дават следните възможности: за получаване на образи, като се прави "снимка" на част от .съдържанието на графичния екран; за четене или запазване на образи във файлове; за зареждане на изображения на екрана.
EMPTYIMAGE, GETIMAGE, GETSHAPE, HOTSPOT, IMAGE, IMAGE?,
IMAGESIZE, LOADBITMAP, LOADIMAGE, PATH, PHASE, PUTIMAGE,
SAVEBITMAP, SAVEIMAGE, SETHOTSPOT, SETIMODE, SETPATH,
SETPHASE, SETPHASEMODE, SETSHAPE, SETSHAPECOLOR,
SHAPECOLOR, SHAPEPOS
Следващите процедури, включени в групата Операции с думи и списъци, могат да се използват и при работа с образи: =, BL, BUTFIRST;
BF, BUTLAST; BL, COUNT, EMPTY?, EQUAL?, FIRST, FPUT, ITEM.
LAST, LIST, LIST?, LPUT, PICK, REPLACE, SENTENCE; SE, SHUFFLE.
REVERSE

Потребителски интерфейс - Тази група включва процедури, които определят вида на работната област на Основния прозорец. Дали да са показани иконите. Дали да е видим само графичния екран, само текстовия екран или и двата (смесен екран). Изтриване съдържанието на двата екрана. Какви да са техните размери и цветове - цвят на фона за графичния екран, цвят на фона и на текста за текстовия екран.
BACKGROUND; BG, CLEAN, CLEARSCREEN; CS, CLEARTEXT; CT,
CURSOR, DOTCOLOR, DRAW, GETGS, GETTS, GRAPHICSCREEN; GS,
HIDESPEEDBAR, LOADSCREEN, LOGOSIZE, PRINTG, PRINTGRAPHICS,
PRINTG, PRINTTEXT; PRINTT, RESETPAL, RESETPALETTE;
RESETPAL, SAVESCREEN, SETBG, SETBACKGROUND, SETCURSOR,
SEETGS, SETLOGOSIZE, SETMOUSECURSOR; SETMC, SETPROMPT,
SETREADLISTPROMPT; SETRLPROMPT, SETTC, SETTS,
SHOWSPEEDBAR, SPLITSCREEN; SS, TEXTSCREEN; TS,
SETMOUSEPOS, WINDOWSPOS

Диалогови прозорци и бутони - Тази група включва команди и операции, които дават възможност да се отварят и затварят специални диалогови прозорци, да се показват и скриват иконите.
BUTTONS,BUTTONSSIZE, CHOOSER, CLOSEPROBLEM,
HIDEBUTTONS,HIDESPEEDBAR, LOADPROBLEM, SETBUTTONS,
SETBUTTONSSIZE,SHOWBUTTONS, SHOWSPEEDBAR

Въвеждане и извеждане на данни - С помощта на командите и операциите от тази група се въвеждат данни от графичния екран, от текстовия екран, чрез мишката, от клавиатурата или от файл или се извеждат данни на графичния екран, на текстовия екран или във файл. Има възможност за извеждане дефинициите (или само заглавията) на

процедурите. CHAR, CLIPBOARD, KEY?, MESSAGEBOX, PL, PO, POTS, PR, PRINT; PR, PRINTG, PRINTGRAPHICS, PRINTG, PRINTLINE; PL, PRINTM, PRINTMEMORY; PRINTM, PRINTT, PRINTTEXT; PRINTT, PRINTTO, RC, READCHAR; RC, READFROM, READKEY, READLIST; RL, READLISTDIALOG; RLD, RLT, READLISTTURTLE, READWORD; RW, RL,RLD, RLT, RW, SETFORMAT, SETPROMPT, SETREADLISTPROMPT; SETRLPROMPT, SETRLPROMPT, SH, SHOW; SH, TYPE

Работа с файлове - Операциите и командите от тази група дават следните възможности: да се зареждат и запазват файлове, съдържащи побитови изображения (bitmaps), образи, библиотеки, и текстови файлове; да се пренасочва входа и изхода на данни (т.е. да се установява файл за четене и файл за писане); да се задават специалните поддиректории на Comenius Logo. APPEND, CLIPBOARD, DIR, EDFILE, EDITFILE; EDFILE, EOF?, ERASEFILE; ERFILE, ERFILE, FILE?, LOAD, LOADBITMAP, LOADIMAGE, LOADLIB, LOADLIBRARY; LOADLIB, LOADPATH, LOADSCREEN, PATH, PRINTTO, READFROM, SAVE, SAVEBITMAP, SAVEIMAGE, SAVELIB, SAVELIBRARY; SAVELIB, SAVESCREEN, SETPATH

Управление на мишката - Comenius Logo предлага възможност за отчитане действията, които потребителят извършва с мишката - натискане и отпускане на ляв или десен бутон на мишката, влачене на мишката при натиснат ляв бутон и др. CODEAREA, MOUSE, MOUSESTATE, READKEY, SETMC, SETMOUSECURSOR, KEY?, TOUCHED, .SETMOUSEPOS

Музика и мултимедия - Тези команди позволяват да се генерират звуци, да се интерпретират мелодии, да се контролират мултимедийни устройства. CLOSEVIDEO, MCI, OPENVIDEO, PATH, PLAY, PLAYW,PLAYWAVE; PLAYW, SETPATH, SOUND,

Управление на работната памет - Единствената команда, включена в тази група, служи за оптимизиране (т. нар. garbage collection) на работната памет. CLEANMEM

Характеристики на системата - Тази група съдържа вградени процедури, които осъществяват връзка със системата Windows. Например, за да се получи информация за текущите време и дата се използва командата SETLOGO, като се задават стойности на специалните променливи, които определят основни за системата Comenius Logo характеристики. DATE, DATEW, DATEWORD; DATEW, LOGO, RUNPROG, RUNPROGRAM; RUNPROG, SETLOGO, TIME, SYSTEMMETRICS, TIME, TIMEW, TIMEWORD; TIMEW

Изход от системата - Единствената команда, включена в тази група, предизвиква изход от Comenius Logo. BYE

7. Команди (инструкции) и операции

Процедури, които връщат стойност, се наричат операции, а тези, които нямат тези свойства - команди. Командите имат някакъв ефект, например извеждат текст на екрана, движат костенурка, променят някакви характеристики и др.

Акцентът в описанието на операциите е, че връщат стойност, с други думи, след изпълнението на операциите се получава резултат. Всяка стойност, получена като резултат от изпълнението на една операция трябва да се използва като вход за друга операция или команда. Операциите, както и инструкциите могат да бъдат вградени или пък дефинирани от потребителя.

Примери за вградени операции

print sqrt 2.56 print sqrt 3+6

round - при вход произволно десетично число дава като резултат най-близкото цяло число

ср. гъстота на населението -

```
to sr.gast :plost :naselenie
( print [srednata gastota na naselenieto e] ~
round :naselenie / :plost fjitel na kw.km])
end
```

изпълняваме с sr.gast 110900 8846000

pick число - резултатът е случайно число между 0 и входа - за да се имитира зарче трябва да запишем в някоя инструкция операцията 1+pick 6

PICK произволен_вход, където вход не е празната дума, празният списък или празният образ.

Ако входът е дума, резултатът е случайно избран знак от думата.
Ако входът е списък, резултатът е случайно избран елемент от списъка.
Ако входът е образ, резултатът е случайно избран кадър от образа.

SH PICK [1 2 3 4 5 6 7 8 9]
8

Примери за дефиниране на нови операции - a"

степен -

```
to stepen :a :n
make "stepen 1
repeat :n [make "stepen :stepen * :a]
print :stepen
end
```

8. Графични команди в Comenius Logo, цикъл, процедури без и с параметри

redraw - подготовка за чертане, т. е. преминаване в графичен режим.Екранът се изчиства и в средата му се появява костенурката, обърната на горе (на север). Трябва да не се забравя мястото, където се намира костенурката и посоката в която е обърната,

cs изчиства екрана
forward брой стъпки (напред),
back брой стъпки (назад),
left бройградуси (наляво),
right брой_градуси(надясно)

repeat брой_повторения [списък_от_инструкции] (повтори)
Една стъпка = разстоянието между две светещи точки на екрана

пример 1.

```
fd 50
right 90
fd 50
right 90
fd 50
right 90
fd 50
right 90
```

задачи: да се начертае квадрат, като се използват инструкциите fd и left, back и left, back и right; да се напишат буквите А, Б, В

пример 2

```
repeat 4 [fd 50 right 90]
```

пример 3

За чертане на равностранен триъгълник костенурката се обръща на 120 градуса (външните ъгли на триъгълника)

```
repeat 3 [fd 50 right 120]
```

пример 4 - правилен шестоъгълник със страна 40

```
repeat 6 [fd 40 right 60] и др фигури
cs   repeat 8 [fd 20 right 45] или
cs   repeat 12 [fd 10 right 30] или
cs   repeat 180 [fd 1 right 2]
```

- **влагане на цикли**

```
[repeat 4   [fd 50 right 90]                    repeat 18
                                                right 20]
```

- **процедури без параметър**

```
квадрат -   to kvadrat
            repeat 4 [fd 50 right 90]
            end
триъгълник - to tri
            repeat 3 [fd 50 right 120]
            end
къща    - to house
            kvadrat
            .  fd 50 right 30
            tri
            end
```

- **процедури с параметър**

```
квадрат - to kvadrat :strana
        repeat 4 [fd :strana right 90]
end
kvadrat 80
kvadrat 40
kvadrat 20 и т.н.
```

Входът може да бъде и аритметичен израз (+, -,\ /) напр. fd 5*4+6

```
триъгълник    to tri :strana
                  repeat 3 [fd rstrana right 120]
                  end
```

пример 8 - процедура с параметър, която вика други процедури с параметър

```
къща   - to house :razmer
           kvadrat :razmer
           fd :razmer right 30
           tri :razmer
       end
       house 50,
       cs,
           house 100 и т.н.
```

(необходими са процедурите kvadrat и tri)

село - изпълняване няколко пъти с различни входове

(придвижваме се в начално положение)

```
to house :razmer
kvadrat :razmer
fd :razmer right 30
tri :razmer
```

(с pendown и репир)

```
to house :razmer
pendown
kvadrat :razmer
```

```
left 210
fd :razmer right 90
fd :razmer right 90
end
```

,

```
tri .razmer
left 210
fd :razmer right 90
penup
fd '.razmer right 90
end
```

пример 9 - процедури с повече параметри

```
правоъгълник -    to prav :visochina :osnova
repeat 2 [fd :visochina right 90 fd :osnova right 90]
                  end
```

изпълняваме с различни входове - prav 100 prav 50 и т. н.

пример 10 - кула

```
to kula
kvadrat 80
fd 80
kvadrat 40
fd40
kvadrat 20
fd20
```

```
kvadrat 10
fd 10
end
```

Това не е най-елегантния начин за изчертаване на кула (страната е два пъти по-малка)

пример 11 - задаване стойност на променлива

```
to nova.kula .strana
repeat 5 [kvadrat :strana fd :strana make "strana :strana/2]
end
```

9. Рекурсивни процедури - процедури, които в описанието си съдържат обръщение към себе си - удобни са за моделиране на периодично движение

```
to krag.orbita
fd 1 right 1
krag.orbita
end
```

За изпълнение:

```
draw
krag.orbita,
```

а за прекъсване: ctrl/break

За дефиниране на по-разнообразни орбити, можем да въведем подходящи параметри -

```
to orbita '.razmer :agal
fd :razmer right :aga
orbita :razmer :agal
end
```

За изпълнение:

```
draw
orbita 50 144
orbita 30 108
orbita 50 160
```

експеримент 1 -

```
to :razmer :agal
fd : razmer right :agal
eksperiment :razmer + 5 :agal
end
```

изпълнява се с входове:

```
eksperiment 1 90
eksperiment 2 89
eksperiment 1 72
eksperiment 2 75
```

експеримент 2 -

```
to eksperiment2 :razmer :agal
fd: razmer
right :agal
eksperiment2 :razmer :agal + 5
end
```

изпълнява се с входове:

```
eksperiment2 3 2
eksperiment2 10 1.
```

експеримент 3

```
to eksperiment3 :razmer :agal :uvel1 :uvel2
fd .razmer  right :agal
eksperiment2 :razmer + :uvel1 :agal + :uvel2 :uvel1 :uvel2
end
```

За получаване на различни фигури изпълняваме например с: eksperiment3 5 90 3 3 и т.н.

10. Логически изрази

а) Логически стойности

Един логически израз има една от двете стойности: вярно (TRUE), когато съответното условието е удовлетворено и невярно (FALSE), в противен случай. Тези две стойности се наричат логически (вероятностни). Логическа функция е функция, която на всеки елемент от дадено множество съпоставя една от двете логически стойности вярно или невярно.

б) Съждения и предикати

Израз, за който може да се каже, че е верен или неверен, в математическата логика се нарича *съждение.* Истинността или неистинността, приписвана на дадено съждение, се нарича негова верностна стойност.

Израз, който съдържа променлива величина и става вярно или невярно съждение в зависимост от стойността, която даваме на променливата величина, се нарича *предикат.* Предикатът всъщност е логическа функция, защото на всеки елемент от дадено множество (множество от числа, хора и др.) се съпоставя една от двете логически стойности - вярно или невярно,

x - 2=0;

x + 3 <0

Решението на едно уравнение или на едно неравенство е множество от онези стойности на x, за които съответният предикат се превръща във вярно съждение.

в) Логически изрази

И така, съжденията и предикатите са логичеки изрази. В Лого числените стойности могат да се сравняват, като се използват знаците за сравнение: >, <, =. Логическите стойности се означават с думите ВЯРНО (TRUE) и НЕВЯРНО (FALSE).

5 = 0 - Резултат: невярно;

10 > 0 - Резултат: вярно

г) Операции предикати

Операция, резултатът от която е логическа стойнот, се нарича операция - предикат. Предикатите в Лого, също както и другите операции, могат да бъдат вградени или дефинирани от потребителя. Използват се главно в условни инструкции.

Имената на вградените предикати завършват с ? знак.
Number? - дава резултат True, ако входът й е число и False - в противен случай:
print number? 5 - отговор true;
Print number? [5] - отговор false
WORD? - дава резултат TRUE, ако входът е дума, и FALSE в противен случай:
? SH WORD? "CynEP.LOGO - отговор TRUE
SH WORD? LAST [AX OX EX] - отговор TRUE ;
Забел. За разлика от Print , Show извежда най-външните скоби на списъците (може да се използва и командата Type, която действа както командата Print, но курсорът остава на същия ред.)
LIST? - дава резултат TRUE, ако входът е списък, и FALSE - в противен случай.
? SH LIST? [А Б В] - отговор TRUE ;
? SH LIST? [ ] - отговор TRUE ;
? SH LIST? FIRST [А Б В] - отговор FALSE ;
? SH LIST? "ЯНА - отговор FALSE
Може да се дефинират операции-предикати и от потребителя - добре е"да се спазва уговорката за ? знак.

11. Условни команди и операции в Лого

IF логически_израз списък1
IF логически_израз списък1 списък2
където списък1, списък2 са списъци от инструкции.
Команда: Проверява се условието, дадено като първи вход. Ако стойността на логическия израз е TRUE, се изпълняват инструкциите от списък1 и изпълнението на IF завършва. Ако логическият израз е FALSE, в първата форма (не е даден вход списък2), нищо не се случва, т.е, изпълнението на командата няма никакъв ефект. Във втората форма се изпълняват инструкциите от дадения списък2.

```
TO PROBA.1 :CHISLO
      IF MOD :CHISLO 2 = 0 [PR "ЧЕТНО] [PR "НЕЧЕТНО]
END
? ПРОБА. 1 22
ЧЕТНО
? ПРОБА. 1 23
НЕЧЕТНО
```

Операция: IF логически_израз [израз1] [израз2]
Ако условието е изпълнено, резултатът е стойността на израз! В противен случай (условието не е изпълнено) резултатът е стойността на израз2.

```
TO PROBA.2 :CHISLO
OP IF MOD :CHISLO 2 = 0 ["ЧЕТНО] ["НЕЧЕТНО]
END
```

Изпълнение на процедурата

```
? SH PROBA.2 22
ЧЕТНО
? SH PROBA.2 23
НЕЧЕТНО
```

Внимание! Инструкцията, съдържаща IF, трябва да е разположена на един логически ред и, ако не е единствената инструкция на реда, трябва да се загради в кръгли скоби.
Ефектът от изпълнението на IF може да се постигне на няколко стъпки като се използват командите TEST, IFTRUE и IFFALSE. (виж Help)
Забележка:
OUTPUT; OP ИЗХОД — команда с 1 вход
OUTPUT произволен_вход
Прекратява изпълнението на операцията, в чиято дефиниция се среща. Резултатът (изходът) от тази операция е произволен вход.
Командата OUTPUT се използва само в дефинициите на потребителски операции. За разлика от операциите, изпълнението на командите (вж. типове процедури) се прекратява чрез командата STOP).

```
TO suma.kwadr :A :B
OP : A *  :A + :B * :B
END
```

Пример1: Отгатни число: да се намисли едно число, да се сравни намисленото число със зададеното от нас число - предположение и в зависимост от резултата да ни поздрави или да ни съобщи, че не сме познали, (числото ще бъде в границите 0 до 20)

```
to shans :chislo
make "sluchajno pick 21
if :chislo = :sluchajno [pr [brawo pozna]][pr [jalko, ne pozna]]
pr [hajde da igraem pak]
end
```

[4] *Пример 2:* Дефиниране на операции-предикати - дали дадено число е
четно или нечетно

```
to chetno? :x
if ( mod :x 2 ) = 0 [print [true]] [print [false]]
end
```

12. Логически операции в ЛОГО

Сред операциите-предикати има специален вид операции, които не само
дават като резултат логическа стойност, но изискват входовете им да са също логически стойности.
NOT НЕЕ — операция с 1 вход
NOT логически_израз - Резултатът е TRUE, ако стойността на входа е FALSE, и FALSE - в противен случай.
? SH NOT "TRUE отг. FALSE
? SH NOT NUMBER? FIRST "A55 отг. TRUE
OR НЯКОЕОТ — операция с 2 или n входа
OR логически_израз1 логичес'ки_израз2

(OR логически_израз ...) - Резултатът е FALSE, ако стойностите на всички входове са FALSE. В противен случай (поне един вход има стойност TRUE) резултатът е TRUE. Ако има повече от два входа (или само един вход), инструкцията, съдържаща OR, трябва да се постави в кръгли скоби.

? SH OR "FALSE "TRUE отг. TRUE
? SH OR NUMBER? FIRST [[2 3]] NUMBER? LAST [[2 3]] отг. FALSE
? IF OR EQUAL? 2 3 EQUAL? 4 5 [PR "AAA] [PR "БББ] отг. БББ

AND ВСИЧКИ — операция с 2 или п входа
AND логически израз логически израз
(AND логически израз ...) - Резултатът е TRUE, ако стойностите и на двата входа са TRUE. В противен случай резултатът е FALSE. Ако има повече (или по-малко) входове от два, резултатът ще бъде TRUE, само ако всички входове са TRUE.

? SH AND NUMBER? FIRST [2 3] NUMBER? LAST [2 3] отг. TRUE
? IF AND (COUNT [1 2 3] = 3) MEMBER? 2 123 [PR "ДА] отг. ДА
? SH (AND NUMBER? 5 NUMBER? 6 NUMBER? "АБВГ) отг. FALSE

13. Обекти в ЛОГО. Вградени операции с думи и списъци

а) обекти в Лого

В Лого има три основни обекта - числа, думи и списъци.

Думата е последователност от знаци без интервал между тях. За подчертаване, че даден обект е дума, непосредствено пред него се поставя знакът (") - например, "HELLO, "ЗДРАВЕЙ .

Числата могат да бъдат цели или десетични. Пред тях не е необходимо да се поставя знакът".

Списъкът е последователност от числа, думи или други списъци, оградени от квадратни скоби ([ ]).

Примери за списъци:

PRINT [АЗ СЪМ САМО ЕДИН КОМПЮТЪР]
REPEAT 3 [FD 20 LEFT 90]
Елементите на списъка се отделят с интервал:
[до ре ми фа] 1 2 3 4 [ааа бббб] [зала 405] ~
[точка а1 [3d 40] точка в1 [10-20]]

За компютъра е ясно, че елементите на даден списък са или думи или списъци й затова не е необходимо да се посочва с водещ знак ("), че даден елемент от списъка е дума.

Пример: Присвояваната стойност е дума
make "duma "Logo
print :duma отг. Logo

Пример: Присвояваната стойност е списък
make "spisak [обичате ли Моцарт?]
print :spisak отг. обичате ли Моцарт?

б) вградени операции с думи и списъци

В Лого съществуват вградени операции - предикати, с които може да се проверява какъв е типът на даден обект.

За работа с отделни части от обектите се използват вградените в Лого операции, които могат да се прилагат към дума или списък.

Butlast *обект* (безпоследен) - Ако стойността на обект е дума, резултатът е дума, която съдържа всички знакове на обект без последния, а ако е списък, резултатът е списък, който съдържа всички елементи на обект без последния.

Butfirst *обект* (безпърви) - Ако стойността на обект е дума, резултатът е дума, която съдържа всички знакове на обект без първия, а ако е списък, резултатът е списък, който съдържа всички елементи на обект без първия.

First обект (първи) - Резултатът е първият елемент на входния обект, който може да бъде дума или списък.

Last *обект* (последен) - Резултатът е последният елемент на входния обект, който може да бъде дума или списък.

Примери:

print First "азбука отг. А
Print Butfirst "азбука отг. Збука
Print Butlast "азбука отг. азбук

Make "азбука [А Б В Г Д]
Print First .азбука отг. А
Print First [[Иван Иванов] е учител по информатика] отг. Иван Иванов

Ако трябва да се отрежат наведнъж няколко елемента от входа трябва да се приложи последователно операцията Butfirst и (или) Butlast:

Print Butfirst Butfirst [аз ти той тя то] отг. той тя то или
Print Butlast Butlast [аз ти той тя то] отг. аз ти той

Ако се отрежат повече елементи отколкото има в списъка
Print Butfirst Butfirst [аз ти] отг.[ ]
(Butlast не приема [ ] като вход.)
[ ] - е празен списък,
празна дума е дума, която не съдържа нито един знак. Празна дума не може да бъде вход на тези операции-предикати.

в) Обединяване на списъци в думи

Sentence *обект1 обект2* (обединение) или *(Sentence обект1 обект2 ...обеКтN)* - входът е списъци или думи, а резултатът е списък, получен чрез обединяване на елементите на входовете.

Print Sentence [аз ти той] [ние вие те] отг. аз ти той ние вие те

List *обект1 обект2* или *(List обект1 обект2 ... обектИ)* (списък) - има подобно действие, но резултатът е списък, чийто елементи са самите входове.

Print List [аз ти той] [ние вие те] отг. [аз ти той] [ние вие те]

Ако входовете на Sentence и List са думи, резултатът е един и същ.

Print List "Здравей "Приятел отг. Здравей Приятел
Print Sentence "Здравей "Приятел отг. Здравей Приятел

Word *дума1 дума2* или *(Word дума1 дума2 ... думаИ)* (дума)- за обединяване на входовете в посочения ред (в една дума). Когато

входовете са повече от два те трябва да се заградят заедно с името на операцията в кръгли скоби.

```
Print Word "теле "фон                    отг. Телефон
Print (Word "ин "фор "ма "тика)          отг. информатика
```

Word? *обект* - Резултатът е думата True, ако входният обект е дума и False в противен случай.

List? *обект* - Резултатът е - True, ако входният обект е списък и False в противен случай.

14. Входни операции

а) въвеждане на знаци

READCHAR; *RC ВХОДЗНАК — операция без входове*

Резултатът е дума от един знак, прочетен от клавиатурата или от текущо установения файл за четене (вж. READFROM).

Ако се чете от клавиатурата, системата изчаква докато потребителят натисне клавиш (по-точно, изчаква докато предикатът KEY? Върне стойност TRUE). Въведеният знак не се извежда на текстовия екран.

```
? SH RC          ако потребителят натисне клавиша A,
A                A се извежда от SHOW, а не от READCHAR
? SH (SE RC RC RC RC RC RC)
                 ако потребителят натисне П, Р, И, В, Е, Т
[ П Р И В Е Т]
```

За да се провери дали е натиснат управляващ клавиш (стрелка, Insert, Delete, Home, End, PgUp, PgDn и др.), може да се използва и READCHAR, но по-удобна е READKEY.

```
? IF RC = CHAR 0 [SH ASCII RC]; ако потребителят натисне
83    ; клавиша Delete
```

б) програма художник - за упражнение на Readchar

За рисуване чрез натискане на определени клавиши ще създадфм програмата hudojnik, а преди това програмата komandi.

художник - без спирачка

```
(f - напред; b - назад;
l - наляво; r- надясно)
to hudojnik
komandi
hudojnik
end
to komandi
make "komanda
readchar
if :koman.da = "f
[forward 10]
if :komanda = "b [back
10]
if .-komanda = "l [left 10]
end
```

художник - със спирачка

```
(f - напред; b - назад;
l - наляво; r - надясно s - стоп)
to hudojnik
komandi
if :komanda = "s [stop]
hudojnik
end
to komandi
make "komanda readchar
if :komanda = "f [forward 10]
if :komanda = "b [back 10]
if :komanda = "r [right 10]
```

По същия начин с отделни клавиши може да се кодира изчертаването на някои геометрични фигури - квадрат, триъгълник окръжност и др.

в) въвеждане на дума

READWORD; RW ВХОДДУМА — *операция без входове*

READWORD

Резултатът е дума, прочетена от клавиатурата или от текущо установения файл за четене (вж. READFROM).

Ако се чете от клавиатурата, на текстовия екран се извежда промптът на READLIST, изчаква се докато потребителят въведе дума и натисне клавиша Enter. Въведената дума е резултатът от изпълнението на операцията. Ако се въведат повече от една думи, резултатът е първата от тях.

```
? MAKE "ТВОЕ.ИМЕ RW
: АСЕН                   ако потребителят въведе такава дума
? SH :ТВОЕ.ИМЕ
АСЕН
```

в) въвеждане на текст

) READLIST; RL ВХОДСПИСЪК — операция без входове

Резултатът е списък, прочетен от клавиатурата или от текущо установения файл за четене (вж. READFROM), във вид на списък. Ако се чете от клавиатурата, на текстовия екран се извежда промптът на READLIST, изчаква се докато потребителят въведе текст и натисне клавиша Enter. Въведеният текст е резултатът от изпълнението на операцията.

```
? SH READLIST
: МНОГО СТРАННА СТРУКТУРА
[МНОГО СТРАННА СТРУКТУРА ]
```

READLIST се използва за създаване на диалогови програми, т.е. програми, при изпълнението на които се осъществява диалог с потребителя.

```
?MAKE "СП READLIST
: PRINT [ПРИВЕТ]
?RUN :СП
ПРИВЕТ   ; вж. процедурата RUN
```

г) четене на данни от файл

READFROM ВХОДОТ — команда с 1 вход

READFROM файл

READFROM [ ]

Пренасочва входа на данни, т.е. установява файл за четене, който се използва при изпълнение на операциите READCHAR, READWORD и READLIST - те ще четат данни от зададения файл вместо от клавиатурата. Файлът ще бъде потърсен в работната директория или в директорията, зададена в началото на посоченото име на файл. Ако не се зададе разширение, по подразбиране се използва .LGO. Ако файл с даденото име не съществува, възниква грешка:

? READFROM "FILE55
Грешка: Имам проблеми с файла FILE55
Ако е достигнат края на текущо установения файл за четене, предикатът EOF? връща стойност TRUE.
READFROM [ ] възстановява входа на данни да бъде от клавиатурата.
\д) *Примери:* - Извеждане и въвеждане на данни във и от файл
Извеждане на данни във файл: (PRINTTO - виж подточка и .)
? PRINTTO "TRY.TXT
? PR "ТОВА
? (PR "СТАВА "НАЧАЛО.)
? PR [МНОГО ДЪЛЪГ СПИСЪК]
? SH [МНОГО ДЪЛЪГ СПИСЪК]
? PRINTTO [ ] ; файлът се затваря и се възстановява изходът на данни да бъде на текстовия екран

Създаден е текстовият файл *TRY.TXT, който изглежда по следния начин:*

ТОВА
СТАВА НАЧАЛО.
МНОГО ДЪЛЪГ СПИСЪК
[МНОГО ДЪЛЪГ СПИСЪК]

Въвеждане на данни от файла *TRY.TXT:*

? READFROM "TRY.TXT
READWORD
? SH RW прочита първата дума ТОВА
? PR RC С
READCHAR прочита един знак от втория ред ? PR RW
ТАВА
? SH READLIST
[НАЧАЛО.] останалата част от реда ? SH RL
[МНОГО ДЪЛЪГ СПИСЪК]
? SH RL
[[МНОГО ДЪЛЪГ СПИСЪК]]

е) въвеждане на клавиш

READKEY *ВХОДКЛАВИШ* *операция без входове* READKEY

Изчаква докато потребителят или натисне клавиш от. клавиатурата – в този случай резултатът е ASCII кода на натиснатия клавиш, или извърши някое от следните действия с мишката, ако указателят сочи графичния екран:

- натискане на левия бутон на мишката резултатът е 0,
- отпускане на левия бутон на мишката резултатът е -2,
- влачене на мишката при натиснат ляв бутон резултатът е -1.
  - само ако е дефиниран списък от свойства с име специалната дума ODEAREA, щракване с левия бутон на мишката (натискане и отпускане на бутона) върху зададена правоъгълна област от графичния екран- резултатът е съответният на областта код.

Когато чрез READKEY се отчитат действията на мишката в областта на графичния екран, може да се използва операцията MOUSE, която връща позицията на указателя на мишката при последното извършено действие - натискане, отпускане на бутон или влачене на мишката.

? SH SE READKEY READKEY ; ако потребителят натисне б и Б
[225 193]

(за KEY? - виж **подточка** к .)

```
        ТО ИЗЧАКАЙ
        IF KEY? [ДОКЛАД STOP]
        ИЗЧАКАЙ                    ; цикълът се реализира чрез READKEY
END
 ТО ДОКЛАД

        SH READKEY
        SH MOUSE
 END
 ? ИЗЧАКАЙ      ; изчаква докато потребителя натисне 0
 ; левия бутон на мишката в областта на
 [-10187]          ; графичния екран
```

?

За да се провери дали е натиснат управляващ клавиш (стрелка, Insert, Delete, Home, End, PgUp, PgDn и др.)> може да се използва и операцията READCHAR, но по- удобна е REDKEY.

```
? IF READKEY = -79 [PR [Това е клавиша End] ]
; ако потребителят    ; натисне клавиша End
Това е клавиша End
```

ж) въвеждане на списък

READLIST; RL ВХОДСПИСЪК — операция без входове
READLIST
Резултатът е списък, прочетен от клавиатурата или от текущо установения файл за четене.

Ако се чете от клавиатурата, на текстовия екран се извежда промптът на READLIST, изчаква се докато потребителят въведе текст и натисне клавиша. Въведеният текст е резултатът от изпълнението на операцията.

```
? SH READLIST
        : САМО ДВЕ ДУМИ   ; ако потребителят въведе такива данни
[САМО ДВЕ ДУМИ]
? SH READLIST
: МНОГО [СТРАННА [СТРУКТУРА] ]
[МНОГО [СТРАННА [СТРУКТУРА] ] ]
```

READLIST се използва за създаване на диалогови програми, т.е. програми, при изпълнението на които се осъществява диалог с потребителя.

```
? MAKE "СП READLIST
: PRINT [ПРИВЕТ]
? RUN :СП
ПРИВЕТ   ; вж. процедурата RUN
```

з) Програма интервю - за упражнение на Readlist

```
to interv
print [Бихте ли се представили на нашите читатели? Как се казвате?]
make "otgowor readlist
; това е коментар за мене
( print [Здравейте] :otgowor [Какво мислите за компютрите?])
make "otgowor readlist
print sentence [Съгласен съм.че] :otgowor
( print [Вашето становище, че] :otgowor -
[е вярно, но според мен
компютърът може само толкова, -   останалата част от думата
колкото може и човекът, който работи с него. А Вие как мислите?])
make "otgowor readlist
          print [Мисля, че времето ни за интервю изтече. -
Моля, кажете ми Вашия девиз.]
make "otgowor readlist
print [Прекрасно!]
print sentence [Ще запомня тези думи -] :otgowor
print [Искате ли да продължим нашия разговор друг път?]
make "otgowor readlist
if :otgowor = [qa][print [Заповядайте отново утре.]]
print [Довиждане]
end
```

и) PRINTTO ИЗВЕЖДАЙНА -- команда с 1 вход

PRINTTO файл
PRINTTO [ ]
Пренасочва изхода на данни, т.е. установява файл за писане, който се използва при изпълнение на командите PRINT, PRINTLINE, SHOW и TYPE - те ще извеждат данни в зададения файл, а не на текстовия екран.
Файлът ще бъде потърсен в работната директория или в директорията, зададена в началото на посоченото име на файл. Ако не се зададе разширение, по подразбиране се използва .LGO. Ако файл с посоченото име не съществува, такъв се създава. В противен случай (съществува), съдържанието на стария файл се изтрива.

Инструкциите, които потребителят задава от командния ред, и съобщенията за грешки винаги се извеждат на текстовия екран, т.е. не могат да се изведат във файл.
Инструкцията PRINTTO [ ] възстановява изхода на данни да бъде на текстовия екран.

к) - KEY? КЛАВИШ? — операция без входове

Резултатът е TRUE или когато се натисне клавиш от клавиатурата, или когато се извърши някое от следните действия с мишката, ако указателят сочи графичния екран:
- натискане на левия бутон на мишката,
- отпускане на левия бутон на мишката,
- влачене на мишката при натиснат ляв бутон.

В противен случай резултатът е FALSE.

15. Локални и глобални променливи

15.1. Локални променливи

Локалните променливи съществуват само докато съответната процедура се изпълнява. След като завърши работата си, процедурата освобождава заетата от нея локална памет и създадените там променливи се унищожават.

Пример1 процедура с един параметър - удвояване на дадено число

```
to udvo :chislo
Make "chislo :chislo*2
Print :chislo
end
```

При обръщение към процедура с параметри най-напред се отделя локална (собствена за процедурата) памет, като за всеки параметър се съхранява там по една локална променлива със стойност съответния му вход. За примера udvo 10 се създава една локална променлива chislo със стойност 10, след което стойността на локалната променлива става 20 и именно тази стойност се извежда на екрана.
Ако след изпълнението на процедурата се зададе инструкцията
Print xhislo излиза съобщение, че няма такава променлива.

Пример2: една процедура с вход се обръща към друга процедура с вход.

```
to primer :chislo
Print xhislo
udvo :chislo
Print xhislo
end
```

При изпълнение на primer 1 се получава резултат: 1 (от primer); 2 (от udvo); 1 (от primer)
При извикването на процедурата udvo се създава и за нея една локална променлива с име chislo (в нейната собствена локална памет).
Процедурата udvo може да използва и променя само своята локална променлива chislo, но не и променливата chislo от локалната памет, отделена за процедурата primer. С други думи променливата chislo на процедурата primer е невидима за процедурата udvo. Така процедурата udvo удвоява само своето число и след завършване на изпълнението си го унищожава, процедурата primer извежда 1 - стойността на своята локална променлива. Наличието на локални променливи с едни и същи имена в различни процедури не объркава компютъра.
Когато една процедура трябва да използва променлива с дадено име, тя търси най-напред в своята локална памет. Ако не я намери я търси в локалната памет на извикалата я процедура, а ако не може да я намери и там, ще я потърси в процедурата извикапа извикалата и т.н. по веригата на вложените извиквания.

Пример3; дадена процедура използва променлива, която не е локална за самата нея (процедурата udvo ще бъде без параметри),

```
to udvo
```

```
        Make "chislo :chislo*2
    Print xhislo
    end ( процедурата не може да бъде използвана самостоятелно)
```
Ще я включим като част от друга процедура в която променливата chislo получава стойност.
```
    to primerl xhislo
    Print xhislo
        Udvo
        Print xhislo
    end
```
В този случай udvo няма локална променлива и търси стойноста на число в локалната памет на извикалата я процедура primerl. След като изпълнението на udvo завърши, стойността на chislo става 2. (при primerl 1)

15.2 Гпобални променливи

Да проследим какво ще стане, ако в една процедура се опитаме да присвоим стойност на променлива, която не е локална нито за нея нито за някоя от извиканите процедури (ако има такива).

Пример:
```
        to primer2 xhislo
    Print xhislo
    udvo
        Print xhislo
        make "novo_chislo 3.14
    Print :novo_chislo
    End
```
при изпълнение на primer2 10 отговор 10, 20, 20, 3.14

Маке създава нова променлива, която не е локална за никоя от активните в момента процедури. Такива променливи могат да се използват и изменят от всички процедури (стига те да не притежават локални променливи със същите имена) Тези променливи съществуват, докато не бъдат изтрити от работната памет. Наричат се глобални променливи.

Пример*:* на променливата се присвояват не числа, а думи.
```
    to opit :whod
    Print :whod
    Make "whod "lokalna
    Make "nowa "globalna
    Print :whod
    Print :nowa
    end
```
При изпълнение Print :whod и Print :nowa преди извикването на процедурата opit- няма променлива с това име. Действително и двете променливи нямат стойност. Те получават стойност при изпълнение на процедурата opit със зададен вход. opit "да -резултатът е да; lokalna; globalna. След това отново да изведем стойностите на whod и nowa - Print :whod - няма стоност; Print :nowa - globalna;

С първата инструкция make на процедурата opit се изменя стойността на локалната променлива whod, която се губи след края на изпълнението.

С втората инструкция make се създава глобална променлива – тя съществува и след края на изпълнението.

Глобални променливи можем да създаваме и в команден режим, т.е. на глобално ниво: make "chislo 10. След тази инструкция е възможно да се изпълни последния вариант на udvo самостоятелно:
```
udvo                отг. 20.
```
erase име_на_променлива - Информацията за променливата се изтрива от работната памет.

ERASE; ER БЕЗ — команда с 1 или п входа

ERASE вход1

(ERASE вход1 вход2 ...)

където вход е или потребителски_обект,
или списък_от_потребителски_обекти.

Изтрива зададените потребителски обекти.

Често като входове на командата ERASE се използват списъците, връщани от PROCS, NAMES и PROPS.

Резултати от изпълнението на PROCS, NAMES и PROPS...
```
? ERASE NAMES     ; изтрива всички променливи
? (ERASE [F G] [:ПРОМ1 ПРОМ2] PROPS)
; изтрива процедурите F, G, променливите ПРОМ1 и ПРОМ2, и
всички списъци от свойства
? (ERASE "F ":ПРОМ ""P)   ; изтрива процедурата F,    променливата
:ПРОМ и списъка от свойства "P
? ERASE BURIED     ; изтрива "архивираните" обекти
```
Алтернативна възможност:

команда Изтрий избрания обект от меню Обекти в Прозореца на паметта Изтрива избраните обекти.

16. Отново Рекурсия

С термина рекурсия в програмирането се означава възможността да бъдат дефинирани процедури, които съдържат в описанието си обръщение към самите себе си.

Пример*:* Да се дефинира рекурсивна процедура, която чертае произволна композиция от квадрати от следния тип: в центъра е разположен най-малкият квадрат с размер на страната 10 стъпки. От него към краищата размерът на квадратите се увеличава, като разликата в дължините на страните на всеки два съседни квадрата е също 10 стъпки, (изпълнява се kompoz 30 , kompoz 40)
```
    to kompoz :razmer
    kwadrat:razmer
    if :razmer = 10 [stop]
    kompoz :razmer-10
    kwadrat :razmer
    end
    to kwadrat :strana
    repeat 4 [fd :strana right 90]
```

```
        right 90 fd :strana left 90
        end
```

DEBUG [списък_от_инструкции]

Зададените инструкции се изпълняват постъпково. Системата ще изчаква потребителят да потвърди изпълнението на всяка следваща стъпка. Отваря се диалогов прозорец "Стъпка по стъпка", който предлага различни варианти на постъпково изпълнение.

Ако е отворен Прозорецът на паметта, в Областта за преглед, автоматично се отваря прозорецът на текущата процедура и се осветява редът от нея, който предстои да бъде изпълнен.

DEBUG може да се изпълнява само от командния ред и трябва да бъде единствената команда в реда.

За примера изпълняваме: debug [kompoz 30]

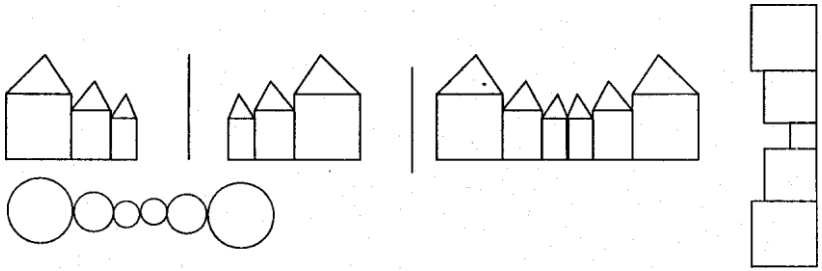
STOP

Прекратява изпълнението на командата, в чиято дефиниция се среща. Управлението се предава на извикващата я процедура.

STOP се използва само в дефинициите на потребителски команди. За разлика от командите, изпълнението на операциите (вж. Команди и операции) се прекратява чрез командата OUTPUT.

Задачи:

Редактирайте процедурата kompoz , така че да чертае фигурите:



17. Рекурсивна обработка на думи

Ще направим словесен вариант на задачата - композиция от квадрати, като заменим квадрата с думата ЛОГО. Ако се приеме, че скъсяването на думата ЛОГО съответства с намаляването на размера на квадратите, то аналогията с квадратната композиция е пълна.

```
to komp :d              ;ЛОГО
print :d                ;ЛОГ
if ( butlast :d ) = "   ;ЛО
[stop]
komp butfirst :d        ;Л
print :d                ;ЛО
end                     ;ЛОГ
                                ;ЛОГО
```

извиква се с: komp "ЛОГО

Шаблон - рекурсивна обработка на думи

```
        to име_на_процедура :whod
        if :whod="    [stop]
        действие_с :whod
        име_на_процедура butlast :whod
        end
```

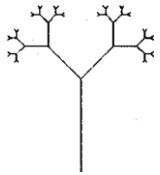
съществуват различни варианти на този шаблон - вместо butfirst може да се използва butlast. В някои случаи се прави нещо не с :whod, а с firstwhod и т.н.

Задачи: да се напишат рекурс. процедури, за обработка на входната дума по сл. начин:

ЛОГО	Л	лого	О	лого
ЛОГ	ЛО	ого	го	ого
ЛО	ЛОГ	го	ого	го
Л	ЛОГО	О	лого	О
				го
				ого
				лого

18. Разклоняване и рекурсия

а) двоично дърво - алгоритъм



Рекурсията се използва и за описване на модели, съдържащи части, които имат структурата на целия модел. Типичен случай е двоичното дърво.

Задача: Да се състави процедура за чертане на двоично дърво, т.е. дърво, в което всеки клон ражда по два нови клона, като дължината на всеки нов клон е половината от дължината на родителя, а ъгълът на ртклонение е 45 градуса. Алгоритъмът е:

За начертаване на ДЪРВО с размер :размер
- начертаваме стъбло с дължина :рзмер
- начертаваме ляво поддърво
- начертаваме дясно поддърво

Условието за спиране е: размерът на стъблото на елементарното дърво (стъбло) да стане по-малък от 2 см.

б) процедура за чертане на двоично дърво (от тип инструкция)

Ляво и дясно поддърво може да се начертае, като се извика същата процедура tree, с двойно по-малък вход (рекурсивно обръщение). Непосредствено Преди рекурсивното обръщение трябва костенурката да

се завърти на наляво или надясно на 45 градуса, за да се получи съответно ляво или дясно поддърво.

```
to tree: razmer
if razmer <2 [stop]
fd: razmer
left 45
tree :razmer/2
right 90
tree :razmer/2
left 45
back :razmer*
end
```

След изпълнение на tree с даден размер, костенурката се връща в основата на дървото с този размер. в) варианти на двоични дървета При редактиране на процедурата tree, чрез промяна напр. на ъгъла на завъртане или коефицианта с който се намалява размера на стъблото на поддървото, ще се получават различни варианти на дървото. Но макар и красиви те са неестветвени. За по-добро имитиране на природата ще се внесат елементи на асиметрия - напр. всички леви и всички десни клони имат постоянна дължина, нр левите клони са два
пъти по-дълги от десните и ъгълът между два клона е 60 градуса. Тъй като дължината на клоните не намалява се променя условието за спиране. Включва се брояч, който отчита броя на поколенията клони (нивото на рекурсия)

```
to new_tree :razmer :br
if :br=0 [stop]
left 30
fd :razmer
newjree :razmer :br-1
back :razmer
right 60
fd : razmer/2
new_tree :razmer :br-1
back :razmer 12
left 30
end
```

Изпълняваме процедурата с входове 40 и 6 и след изпълнение на процедурата прибавяме стъбло с командата back 100
Задачи: 1. Редактирайте процедурата new_tree, като въведете втори параметър *ъгъл,* който задава ъгъла на отклонение за дървото.

2. Съставете процедура за чертане на дърво, от всеки клон на което да излизат по три нови разклонения.

19. Дефиниране на рекурсивни операции с числа

Рекурсивният подход може да се използва и при дефиниране на операции. Операциите са аналог на функциите. Напр. функцията факториел е класически пример на математическа функция, която се реализира с рекурсия. Рекурсивното определеие на N! е: N!=N(N-1)!

```
to faktoriel :n
if :n =1 [output 1]
output :n*faktoriel :n-1
end
```

За изпълнение напр.

```
make "a faktoriel 5
print :a
```

OUTPUT произволен_вход

Прекратява изпълнението на операцията, в чиято дефиниция се среща. Резултатът (изходът) от тази операция е произволен вход. Командата OUTPUT се използва само в дефинициите на потребителски операции. (За разлика от операциите, изпълнението на командите се прекратява чрез командата STOP).

20. Рекурсивни операции с думи и списъци

Операциите, които ще разгледаме са приложими както за думи, така и за списъци. Обединяваме и двете възможности в операция предикат, която определя дали даден списък (дума) е празен,

```
to prazen? :obekt
        if or (:obekt = " ) (:obekt =[ ]) [output "true]   [output "false]
end
```

изпълнение:
```
print prazen?"          true
        print prazen? "aaa      false
print prazen? [ ]       true
        print prazen? [a a a]     false
```

а) **операция lench -** операцията получава като вход списък и дава като резултат броя на елементите на този списък.

```
to lench :spis
If :spis=[ ] [output 0]
output 1+lench butfirst :spis
end
```

изпълнение:
```
print lench [ ]                 0
        print lench [aaa]   1
        print lench [aaa bbb ccc]     3
```

б) **операция-предикат** element? - тя има два входа, вторият от които е списък и дава като резултат true false, в зависимост от това, дали първият вход е елемент на този списък или не.
Нека за определеност считаме, че първият вход на операцията е дума. За да определим дали тази дума е елемент на зададения списък, трябва да я сравним последователно с неговите елементи. Достъп до елемент на списък ни дават операциите first и last - нека използваме first. И така на всяка стъпка от обработката ще сравняваме думата - първи вход с първия елемент от списъка. Ако те съвпадат, изходът от операцията ще е true. В противен случай търсенето трябва да продължи в останалата част от списъка (без първият му елемент) и т.н.В случай, че списъкът е изчерпан

(редуциран до празен), без да сме срещнали търсената дума,
то изходът от операцията ще бъде false.

```
to element? :elem :spis
if :spis=[ ] [print "false]
if :elem = (first :spis) [print "true]]print element? :elem (butfirst :spi$)
end
```

Като разполагаме с предиката element? може да напишем операцията
broj_glasni, която преброява гласните в зададена дума.

```
to broj_glasni :D
    if glasna? (first :D) [output 1+broj_glasni]
     .  butfirst :D
                output broj_glasni butfirst :D
     end
```

където операцията предикат проверява дали входът й е гласна буква,

```
to glasna? :B
        output element? :B [а е.и о у ю я]
end
```

Процедурата broj_glasni обработва думата-вход буква по буква. При всяко рекурсивно извикване (на всяка стъпка) се изследва първата буква от думата. Ако тя е гласна, то резултатът от това рекурсивно извикване е равен на 1 плюс броя на гласните в останалата част от думата. Ако първата буква на думата не е гласна, то резултатът съвпада с броя на гласните в останалата част от думата.
Съществува и стандартна процедура
(ЕЛЕМЕНТ?)
MEMBER? произволен_вход списък
MEMBER? дума1 дума2
В първата форма резултатът е TRUE, ако първият вход е елемент на втория, т.е. на списъка, и FALSE - в противен случай.
Във втората форма резултатът е TRUE в случай, че първата дума се съдържа във втората дума, и FALSE - в противен случай.

```
? SH MEMBER? "АБВ [А АБ АБВ АБВГ ]          отг. TRUE
? SH MEMBER? "АБВ [ [А АБ АБВ АБВГ ] ]      отг. FALSE
;списъкът има само един елемент, който е списък [А АБ АБВ АБВГ]
? SH MEMBER? [АБВ] [ [А] [АБ] [АБВ] ]        отг. TRUE
? SH MEMBER? "ОР "ГЕОРГИ                     отг. TRUE
```

в) операция ELEMENT е най-полезната за работа със списъци. Тяслужи за извличане на елемент от списъка. Операцията има два входа -цяло число N и списък, и дава като резултат N-тия елемент на зададения списък.
Когато N е единица, резултатът ще бъде първият елемент на зададения списък. Това е елементарният случай. Ако N е по-голямо от 1, можем да редуцираме задачата до по-проста задача от същия вид, като вземем предвид, че N-тия елемент на зададения списък съвпада с (N-I)-bvm елемент на същия списък, без първия му елемент. Така резултатът от текущото рекурсивно извикване ще съвпадне с резултата на същата операция с първи вход N-1 и втори вход - списъка, без първия му елемент.

```
to element :n :spis
if :n=1 [ output first :spis]
output element (:n-1) (butfirst :spis)
end
```

Съществува и стандартна процедура (ЕЛЕМЕНТ)
ITEM число произволен_вход
В първата форма (първият вход е число), ако вторият вход е дума, резултатът е съответният на числото знак на думата, ако вторият вход есписък, резултатът е съответният на числото елемент на списъка, ако вторият вход е образ, резултатът е съответният на числото кадър на образа.. Ако зададеното число не е цяло, то се закръгля - прилага се операцията INT.

```
? SH ITEM 3 [А Б В Г Д]    •          отг. В
? SH ITEM 3 [ [А] [Б] [В] [Г] ]       отг. [В]
? SH ITEM 4 "ПЕТЪР                    отг. Ъ
? SH ITEM 2.73 [ [аа] [бб] [вв] [гг] ]    отг [бб]
```

21. Работа със списъци

а) процедура за въвеждане елементите на един списък (например за въвеждане на първите участници в едно състезание)

```
to nach_klasirane
print [въведете на един ред началното класиране]
make "br_klasirane readlist
end
```

Напр. ако челната група се състои от четирима ще въведем при изпълнение на процедурата *nach_klasirane* списък от четири имена: Иван Мария Грета Петър и глобалната променлива br_klasirane ще получи като стойност списък, съдържащ четирите имена.

б) извеждане на пръв и последен елемент от списъка - операциите First и Last
Например: Print First :br_klasirane
или Print Last :br_klasirane.

в) включване на елемент в списъка
за включване на елемент, като първи в списъка се използва операцията Fput
Напр.за включване на Румен в списъка отпред ще се запише:

```
make "br_klasirane Fput "Румен :br_klasirane
```

за включване на елемент като последен в списъка се използва операцията Lput
Напр. за включване на Ана в списъка отзад ще се запише:

```
make "br_klasirane Lput "Ана :br_klasirane
```

за включване на елемент от списъка на произволно избрано място ще напишем процедура, която ще вмъква дадения елемент пред посочен елемент от списъка: *obekt*- елементът, който се вмъква, *elem* – елемента от списъка пред който ще се вмъква новият елемент; spis - списъкът^, в случая br_klasirane.

```
to wm_pred :elem :obekt :spis
if :spis =[ ] [output sentence :obekt[ ]]
```

```
if first :spis =:elem [output sentence :obekt :spis]
output sentence (first :spis ) (wm_pred :elem :obekt butfirst :spis)
end
```

Дефинираната операция ще се използва за актуализиране на класирането - да се вмъкне Кати пред Румен в списъка.

```
make "br_klasirane wm_pred "Румен "Кати :br_klasirane
Print :br_klasirane
```

За по-голямо удобство ще напишем процедура, от тип инструкция, която да вмъква произволно избран обект пред определен елемент от списъка, *koj* - обектът, който се вмъква; *kogo* - обектът от списъка пред който се вмъква.

```
to wmaknLpred :kogo :koj
make "br_klasirane wm_pred :kogo :koj :br_klasirane
end
```

За включване на Пепа пред Кати:

```
wmakni_pred "Кати "Пепа
```

а за да видим пълния списък

```
print :br_elementi
```

в) отстраняване на елемент от списъка

За целта дефинираме нова операция.

```
to otstranen :elem :spis
if :spis =[ ] [output [ ] ]
if first :spis =:elem [output butfirst :spis]
output sentence (first :spis ) otstranen :elem butfirst :spis)
end
```

Ще дефинираме процедура от тип инструкция, с която ще отстраняваме елементи от списъка.

```
to otstrani :ime
make "br_klasirane otstranen :ime :br_klasirane
end
```

За да отстраним "Кати:

```
otstrani "Кати
```

За да видим списъка

```
print :br_klasirane
```

г) търсене на обект в списъка

Ще дефинираме процедурата sprawka, в която ще използваме дефинираната по-рано операция предикат element? (или вградената операция предикат member?)

```
to sprawka :koj
if member? :koj :br_klasirane [( print :koj [е в списъка]) stop]
( print :koj [не е в списъка])
end
```

например: sprawka "Кати

Задачи:

1. Напишете операция за класиране на елемент, който има даден пореден номер в списъка.
2. Като редактирате дадената операция, получете оперция за замяна на елемент с посочен пореден номер с друг.

```
to repl :n :el2 :spis
if :spis=[ ] [output :spis]
if :n=1 [output sentence :el2 :spis]
sentence (first :spis) (repl :n :el2 :spis
end
```

3. Напишете процедура за размяна местата на два елемента от списъка.

22. Данни

Данните в Лого се представят чрез думи или списъци. Те се свързват с определен тип - числов, текстов, логически и др. За всеки тип са определени операциите, които могат да се извършват с тях. напр. числовият тип се свързва с аритметичните операции, текстовият с обработката на думи или списъци и т.н. За разлика от езика Паскал типовете данни в Лого не са строго разграничени, напр. думата True може да се възприеме като всяка друга дума и може да се обработва с операциите First, Last и т.н.

23. Структури от данни. Опашка, стек, едномерен масив

23.1. Опашка

Тя се представя със списък, за който са забранени част от допустимите за него операции. Както и на обикновена опашка пред магазин обслужването става по реда на пристигането, т. е. обслужва се този, който е най-отпред, след което се отстранява от опашката, т.е. правилото за работа с опашка е *първи влязал, първи излязъл.* В Лого опашка може да се реализира, ако при работа със списъци се използват само операциите First, Butfirst, Lput.

23.2. Стек

Правилото за работа със стек е *последен влязъл, първи излязъл.* Стекът може да се представи със списък, при който включването и изключването на елементи става само от единия му край. За работа със стек трябва да се изберат за работа една от тройките: First, Butfirst и Fput или Last, Butlast и Lput.

За програмата художник - ще се използва стек, за да се отмени последната въведена команда. За целта ще се запомня последователността от въведените команди и ако искаме да отменим последната изпълнена команда, може да изтрием рисунката, да отстраним от списъка последната въведена команда и да изпълним списъка от команди отново.

В процедурата komandi и prechertaj списъкът chertej се използва като опашка, а в процедурите komandi и otmeni_komanda - като стек.

л- ляво д- дясно к-край на работа
н- напред; о- отмени команда в- вдигни молив
з- назад; с- спусни молив ч- пречертай отново

```
to hudojnik
draw
```

```
        nachalo
        komandi
        end
                to nachalo
                make "dopust_kom [н д л з в с к ч о]
                make "chertej [ ]
                end
to komandi
make "komanda readchar
if member? :komanda :dopust_kom [make "chertej lput :komanda xhertej]
izpalni_komanda :komanda
   komandi
end
            to izpalni_komanda :komanda
            if :komanda = "н [forward 10]
            if :komanda = "з [back 10]
            if .komanda = "д [right 10]
            if :komanda = "л [left 10]
            if :komanda = "в [penup]
            if :komanda = "с [pendown]
            if :komanda = "к [throw "TOPLEVEL]
            if :komanda = "ч [draw prechertaj butlast xhertej]
            if :komanda = "о [draw otmeni_komanda]
            end
                to prechertaj :chertej
                if xhertej = [ ][stop]
                make "komanda first :chertej
                izpalni_komanda :komanda

                prechertaj butfirst :chertej
                end
        to otmenLkomanda
        make "chertej butlast xhertej
        prechertaj .-chertej
        end
        (финал)
THROW "TOPLEVEL
```

Прекратява се изпълнението на всички текущи процедури, т.е. прекратява се изпълнението не само на процедурата, в която инструкцията, съдържаща думата TOPLEVEL, се използва, но и на тази процедура, която я извиква и т.н. за всички процедури, чието изпълнение не е завършило. Казва се, че се реализира нелокален изход, за разлика от командите STOP и OUTPUT, които предизвикват изход само от процедурата, в която се използват - локален изход.

23.3. Едномерен масив

Наредена съвкупност от еднотипни данни, всяка от които се определя еднозначно чрез своя пореден номер, се нарича едномерен масив.

Масивите не могат да променят своя размер, т.е. броя на елементите си. Основни операции за работа с масив са: достъп до елемент с определен номер и замяна на един елемент с друг.

Да се дефинира операцията Elem и Zamenen

```
to elem :nomer :masiw
            if :masiw -[ ] [output [няма елемент с такъв номер]
            if :nomer=1 [output First :masiw]
            output elem :nomer-1 (butfirst :masiw)
            end
        to zamenen :nomer :element :masiw
        if :masiw = [ ][output [няма елемент с такъв номер]]
        if :nomer = 1 [output fput :element butfirst :masiw]
        output fput first :masiw zamenen :nomer -1 :element butfirst :masiw
        end
```

За пример:

```
        make "masiw [3 0 4 5 2 ]
        make "masiw zamenen 2 6 :masiw
        print :masiw                    отг. 3 6 4 5 2
        make "masiw zamenen 5 13 :masiw
        print :masiw                    отг. 3 6 4  5 13
```

Елементите на даден масив могат да бъдат не само числа, но и произволни други еднотипни данни - дума, списъци, точки, прави и др.

24. Работа с костенурки

Върху графичния екран живеят една или повече костенурки (костенурката представлява абстрактен обект в Лого - малка фигура в графичния екран), които представляват малки графични курсори. Comenius Logo предлага богат избор на вградени процедури за работа с костенурки. Например, за движение на костенурките по графичния екран. Ако моливът на една костенурка е спуснат, когато тя се движи върху екрана остава следа и по този начин се създава графично изображение.

Лик на костенурка

За да се представи една костенурка на графичния екран, се използва произволен образ. Ликът на костенурката не е част от съдържанието на графичния екран, костенурката "живее малко над екрана". Това означава, че тя може да се движи без да изтрива с лика си начертаното върху екрана.

Характеристики на костенурка

Всяка костенурка има следните характеристики:

-*име,* може да е произволна дума;

-*начална позиция,* т.е. позицията, в която е създадена;

-*начална посока,* т.е. посоката, която костенурката е имала, непосредствено след нейното създаване;

-*текуща позиция* е точка [X Y], координатите се отчитат спрямо координатна система с начало [0 0] в центъра на графичния екран;

-*текуща посока* е число от 0 до 360, т.е. ъгъл, отчетен в градуси;

-*състояние на молива* - дали моливът е вдигнат, спуснат, дали е

алтернативен или е в режим на изтриване - цвят на молива, широчина на молива, вид на следата;
-*лик,* който представя костенурката на графичния екран;
-дали костенурката е скрита или е показана;
-дали е активна или е пасивна;
-как се използват кадрите на лика: дали се сменят при промяна посоката на костенурката (по подразбиране) или когато потребителят укаже, независимо от текущата посока на костенурката;
-шрифт на текста, който костенурката може да изведе на графичния екран,
-работна област на костенурката и нейното поведение, когато достигне границите на областта.

Команди за управление на костенурка

Предлагат се следните команди за инициализиране и изтриване на костенурка:
MAKETURTLE - създава нова костенурка,
SETSHAPE - дава нов лик на костенурка,
ERASETURTLE - изтрива костенурка,
Най-използваните команди за управление на костенурка са:
-за промяна текущата позиция на костенурка - FORWARD, SETX, SETY, SETXY, SETPOS, HOME;
-за промяна «текущата посока на костенурка - LEFT, RIGHT, SETHEADING;
-за промяна състоянието на молива на костенурка - PENUP, PENDOWN, PENERASE, PENREVERSE, SETPEN, SETPENCOLOR, SETPENWIDTH, SETPATTERN;
-за да се покаже или скрие костенурка - SHOWTURTLE, HIDETURTLE;
-за текста, който костенурката извежда на графичния екран - SETTTEXT, TTEXT;
-за задаване работна област на костенурка и определяне нейното поведение, когато достигне границите на областта - WRAP, WINDOW.

цяло число в интервала [0,15], съответстващо на плътен цвят, п пътните

2 3 1 B B 7 10 11 12 13 14 15

Операции за характеристиките на костенурка

Освен команди за управление, при работа с костенурка се използват операции, които дават информация за текущото състояние на костенурката, т.е. за нейните характеристики. Стойностите, които тези операции връщат показват какви са текущите позиция и посока на костенурката, цвят на молива, широчина на молива и др.
За да се получи информация за текущото състояние на молива, може да се използват следните операции: PEN, PENCOLOR, PENWIDTH, PATTERN, PENSTATE.
За да се разбере дали костенурката е показана или е скрита, се използва предиката SHOWN?.
За информация относно текущите позиция и посока: POS, XCOR, YCOR, HEADING. POLYGON списък_за_многоъгълник

Всяка активна костенурка начертава на графичния екран запълнен многоъгълник, видът на който се определя от зададения списък за многоъгълник. Входът може да е от вида:
[число1 число2 ...]
или от вида
[число списък_за_многоъгълник].
Когато входът е от първия вид - [число1 число2 ...] - всяка активна костенурка се придвижва число1 стъпки напред, завърта се число2 градуса надясно и т.н. за всяка двойка числа от списъка.
Когато входът е от втория вид - [число писък_за_многоъгълник] - всяка активна костенурка изпълнява зададения списък за многоъгълник толкова пъти, колкото е числото.
Контурът на многоъгълника се начертава с цвета на молива.
Вътрешността се запълва в зависимост от текущите вид на пълнежа и цвят на пълнежа.
Ако списъкът за многоъгълник не задава затворен контур, т.е. ако при изчертаването на контура на многоъгълника костенурката не завършва там, откъдето е започнала, POLYGON автоматично затваря контура.
Пример 1 - запълнен квадрат
? POLYGON [100 90 100 90 100 90 100 90]
Пример 2 - начертава се същия запълнен квадрат, какъвто и в пример 1.
? POLYGON [4 [100 90]]
Пример 3 - две звезди (комнтар по-късно)

```
? MAKETURTLE 0 [0 0 0]
? MAKETURTLE 1 [0 0 180]
? TELL [0 1]
? PU
? SETPC 4
? SETFC 8
? POLYGON [5 [100 144]]
? RT 90
```

? POLYGON [11 [23 134 39 83 5 [45 99]**Д**
SETPENCOLOR цвят (SETPC)
Установява цвят на молива на всички активни костенурки. Изпълнението на командата предизвиква и промяна и на
цвета на запълване на всички активни костенурки, той става идентичен с цвета на молива.
Операцията PENCOLOR връща цвета на молива на първата активна костенурка.
1. Задача:
а) Да се нарисуват звезди със зададен размер чрез командара polygon (Първите две процедури са написани от деца)

```
to star6 :size
  polygon list 5 list :size 144
  end
```

```
to star7 :size :n
  polygon list :n list .size 360 / :n * 2
 end
```

б) Да се включи и задаване на цвят

```
 to star6color :size :col
setpc :col
```

```
to star7color :size :n :col
setpc :col
```

```
polygon list 5 list :size 144
end
```

```
polygon list :n list :size 360 / :n * 2
end
```

SETPOS точка

Всяка активна костенурка се премества в посочената точка. Текущата посока не се променя. Ако моливът на костенурката е спуснат (PENDOWN), когато се придвижва в новата позиция, върху графичния екран остава следа.

POS - Операцията POS връща точката, която е текуща позиция на първата активна костенурка. За да се промени текущата позиция иа всяка активна костенурка се използват и командите SETX, SETXY или SETY.

MOUSE

Връща позицията - списък от вида [X Y] - на последното извършено действие с мишката в областта на графичния екран, т.е. когато указателят на мишката сочи графичния екран, операцията отчита натискане на левия бутон на мишката, отпускане на левия бутон на мишката и влачене на мишката при натиснат ляв бутон.

В повечето случаи MOUSE се използва заедно с операцията READKEY, която връща

0, ако е натиснат бутон,

-2, ако е отпуснат левия бутон на мишката, и

-1 при влачене на мишката.

PENDOWN (PD)

Спуска молива на всяка активна костенурка. Когато моливът на костенурка е спуснат, при нейното движение върху графичния екран остава следа. По подразбиране моливът на всяка костенурка е спуснат (вж. командата MAKETURTLE). За да се промени режима на молива на първата активна костенурка, се използват командите SETPEN, PENUP, PENERASE и PENREVERSE.

PENUP (PU)

Вдига молива на всяка активна костенурка. Когато моливът на костенурка е вдигнат, при нейното движение върху графичния екран не остава следа.

в) Да се чертае звездата на място посочено с мишката,

```
to click :size
if readkey = 0 [penup setpos mouse pendown star6 :size]
end
```

Работа с много костенурки

В Comenius Logo могат да се създадат до 4000 костенурки. По подразбиране съществува само една костенурка 0. Чрез командите MAKETURTLE и ERASETURTLE съответно се създават нови и се изтриват вече ненужни костенурки. В даден момент една костенурка е или активна, или пасивна (неактивна). Следните операции дават информация кои костенурки съществуват и кои от съществуващите са активни:

ALLTURTLES - списък на всички костенурки,

WHO - списък на активните костенурки.

Когато се работи с много костенурки, трябва да се спазват следните две правила:

- всяка команда за управление на костенурка (например за движение върху екрана) се изпълнява едновременно от всички активни костенурки,

- всяка операция, даваща информация за характеристика на костенурка (например за позиция, цвят на молива), се изпълнява от първата активна костенурка. Ако няма активна костенурка, възниква грешка.

При работа с много костенурки се използват следните команди:

TELL променя списъка на активните костенурки,

ASK независимо дали са активни, избрани костенурки изпълняват някакви инструкции,

EACH последователно активните костенурки изпълняват зададени инструкции.

CLEARSCREEN (CS)

Изтрива начертаното върху графичния екран. Всяка костенурка застава в началната си позиция и се обръща в началната си посока (началните позиция и посока се определят при създаването на костенурката - команда MAKETURTLE).

GRAPHICSCREEN (GS)

Графичният екран заема цялата работна област на Основния прозорец. Текстовият екран е скрит, но неговото съдържание не се изтрива и той остава активен, т.е. могат да бъдат въведени инструкции от клавиатурата.

SHOWTURTLE (ST)

Всички активни костенурки се появяват на графичния екран. Чрез предиката SHOWN? може да се провери дали първата активна костенурка е скрита. Когато една костенурка е видима, чертае по-бавно, затова в някои случаи е уместно костенурките да се скрият.

HIDETURTLE (HT)

Всички активни костенурки се скриват. Когато една костенурка е скрита, нейният лик не се вижда на **г']' Г';** графичния екран.

2.Задача: Да се създаде процедура за изчертаване на '-.i^[1]- i..- дадената фигура,

```
to fig
cs graphicscreen hideturtle
pendown
setpc 5
setpattern 4
figi
end
```

```
to figl
repeat 6 [repeat 2 [fd 25 rt 60 fd 25 rt 120] ~
rt 60]
penup fd 50 pendown rt 120
repeat 6 [fd 50 rt 60]
end
```

MAKETURTLE дума [ ]

MAKETURTLE дума [число1 число2 число3]

MAKETURTLE дума [число1 число2 число3 думи... лик]

където

- *думи...* е едно или повече имена измежду следните имена на вградени команди PU, PD, PE, PX, PENUP, PENDOWN, PENUP, PENERASE,

PENREVERSE, ST, HT, SHOWTURTLE, HIDETURTLE, TELL и SETPHASEMODE,
- *лик* е или образ, или "LGW_файл, или :променлива със стойност образ.

Създава се нова костенурка с име дума, ако такава не съществува. Характеристиките (позиция, посока, цвят на молива и др.) на новата костенурка зависят от останалите входове.
MAKETURTLE дума [ ] има аналогичен ефект на MAKETURTLE дума [0 0 0]. Ако вторият вход не е празният списък, първите две числа - число1 и число2 - са координатите на началната позиция, а число3 е началната посока на костенурката. Ако вторият вход не съдържа повече елементи, другите характеристики на новата костенурка по подразбиране са: цвят на молива - 0 (черен), широчина на молива -1, състояние на молива - PENDOWN, костенурката е скрита - HIDETURTLE, костенурката не е активна; ликът на костенурката е малък триъгълник (24 кадъра).

Ако вторият вход съдържа елементи думи (една или повече команди) и лик те задават съответни характеристики на новата костенурка (входовете думи и лик имат смисъл, само ако задават характеристики, които са различни от тези по подразбиране).
След стартиране на Comenius Logo, съществува само костенурка 0. Тя е активна (вж. процедурите WHO и TELL) и има начална позиция [0 0] и начална посока 0.
Ако костенурка с име дума съществува, командата MAKETURTLE променя характеристиките на костенурката, за които са дадени съответни входове. Това не -е в сила за началните позиция и посока – те се задават само веднъж, при създаването на костенурката.
ERASETURTLE костенурка (ERTURTLE)
ERASETURTLE списък_от_костеурки
Изтрива зададената костенурка (костенурки). След като една костенурка е изтрита, нейното име вече не е елемент на списъка, връщан от операцията ALLTURTLES. За да се отстрани име на костенурка само от списъка на активните костенурки, се използва командата TELL.
Възможно е костенурка 0 също да бъде изтрита. Командите DRAW и ERALL я възстановяват в нейното начално положение.
ALLTURTLES (ALL)
Връща списък от всички текущо съществуващи костенурки, т.е. костенурки, които са били създадени чрез командата MAKETURTLE. (възможно е само някои от тях да са активни в момента.) По-точно, операцията връща списък от имената на костенурките. WHO
Резултатът съдържа списък от имената на всички текущо активни костенурки (възможно е само някои от текущо съществуващите костенурки да са активни в момента). В зависимост от броя им, една или повече, резултатът е едно име (дума) или списък от имена.
Редът на костенурките в списъка, връщан от WHO, съответства на реда, в който са били създадени, и може да се промени чрез командата REORDER.
Редът на костенурките в списъка, връщан от ALLTURTLES, съответства на реда, в който са били създадени, и може да се промени чрез командата REORDER.
TELL костенурка
TELL списък_от_костенурки
Посочените костенурки стават активни. След стартиране на Comenius Logo, съществува само една костенурка - костенурка 0, която е активна.
3. Задача: Да се създадат две костенурки с различни начални позиции и посоки и едновременно да се начертае фигурата от зад. 2.

```
to gen
        erturtle allturtles
        maketurtle "a [0 0 0]
        maketurtle "b [120 70 15]
        ell [a b]
        end
```

-за изпълнение :

```
draw
gen
fig
```

RANDOM число, където число е неотрицателно. Връща случайно избрано цяло число в интервала [0,число-1]. Ако число не е цяло, първо към него се прилага операцията ROUND (закръгля се),
команда: ASK костенурка списък_от_инструкции
ASK списък_от_костенурки списък_от_инструкции
операция: ASK костенурка [израз]
ASK списък_от_костенурки [израз]
Ако ASK е *команда,* посочените костенурки изпълняват едновременно списъка от инструкции. Ако са посочени костенурки, които не са активни, те се активират за времето, през което се изпълняват инструкциите. Ако вторият вход съдържа израз, неговата стойност се пресмята и става резултат от изпълнението на ASK. В този случай ASK е *операция.* Ако са посочени костенурки, които не са активни, те се активират за времето, през което се пресмята стойността на израза.
4. Задача: Да се дефинира процедура, която чертае дадения квадратен елемент. Като се използва създадената процедура, да се дефинира . друга - рекурсивна процедура, която чертае стълба. Да се създадат п на брой костенурки с различни начални посоки, но с една и съща начална позиция - в центъра на екрана. Всяка от костенурките да начертае стълба:

```
to kwadrat :a
repeat 3 [fd :a rt 90]
bk10rt90
end
```

```
to stalba :a
if :a<3 [stop]
kwadrat :a
stalba :a-5
```

```
end

to rabota :n                                          stalba 20
if :n=0 [stop]                                        stalba 20
maketurtle :n[ ] ask :n [rt random      setpc random 15 pd]
rabota :n-1
end
```

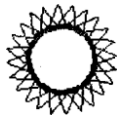
За изпълнение:

```
cs rabota 35
```

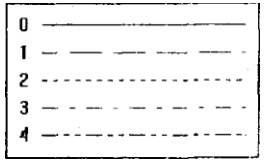
25. Координатна графика

LIST произволен_вход1 произволен_вход2
(LIST произволен_вход1 ...)
Резултатът е списък, получен чрез обединяване на входовете в посочения ред.
ITEM число произволен_вход
ITEM подмножество произволен_вход
В първата форма (първият вход е число), ако вторият вход е дума, резултатът е съответният на числото знак на думата, ако вторият вход е списък, резултатът е съответният на числото елемент на списъка, ако вторият вход е образ, резултатът е съответния на числото кадър на образа. Ако зададеното число не е цяло, то се закръгля - прилага се операцията INT.
Във втората форма (първият вход е подмножество) резултатът е частта от втория вход, съответстваща на подмножеството. Ако е зададено подмножество с 0 елементи (например [3,0]), резултатът е празната дума, празният списък или празният образ в зависимост от типа на втория вход.
STAMP
Всяка активна_костенурка отпечатва (щампова) лика си на графичния екран. По този начин текущият кадър на образа, който се използва за лик на костенурката, става част от съдържанието на графичния екран.
За разлика от STAMP, командата PUTIMAGE отпечатва на екрана произволен образ.
Нека е активна само костенурка 0, чийто лик е малък триъгълник (по подразбиране):

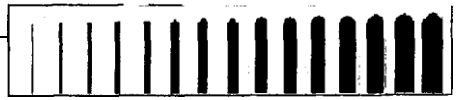
```
        ? TELL 0
        ? PENUP
        ? ST
? REPEAT 24 [FD 45 STAMP RT 165]
? RT 90 FD 110
```



SETHEADING ъгъл (SETH)-
Всяка активна костенурка се обръща в посока, която сключва зададения ъгъл с посоката север.
За разлика от командите LEFT и RIGHT изпълнението на SETHEADING не зависи от текущата посока на костенурката.
SETXY число1 число2
Всяка активна костенурка се премества в точка с координати число1 и число2. Текущата посока не се променя. Ако моливът на костенурката е спуснат (PENDOWN), когато се придвижва в новата позиция, върху графичния екран остава следа.
SETX число
Всяка активна костенурка се премества хоризонтално в точка с абсциса посоченото число. Текущата посока не се променя. Операцията XCOR връща абсцисата на точката, която е текуща позиция на първата активна костенурка.
SETY число
Всяка активна костенурка се премества вертикално в точка с ордината посоченото число. Текущата посока не се променя. Операцията YCOR връща ординатата на точката, която е текуща позиция на първата активна костенурка.
SETPATTERN следа
Установява вида на следата, който моливите на всички активни костенурки оставят върху графичния екран при движението на костенурките.



По подразбиране видът на следата на всяка костенурка е 0, т.е. непрекъсната следа.
Операцията PATTERN дава като резултат вида на следата на първата активна костенурка.
SETPENWIDTH
широчина_на_молив (setpwj^
Установява широчина на моливите на всички активни костенурки. По подразбиране широчината на молива на всяка костенурка е 1 (вж. командата MAKETURTLE).



Операцията PENWIDTH
дава като резултат широчината на молива на първата активна костенурка.
TOWARDS точка
TOWARDS костенурка
Резултатът е ъгълът между посоката север и посоката, в която трябва да се обърне първата активна костенурка, за да се насочи, в зависимост от входа, или към дадената точка, или към точката, която е текущата позиция на дадената костенурка.
Резултатът обикновено се използва като вход на командата SETHEADING.
5. Задача: Четири коли се разполагат в срещуположни писти. Започват да се движат едновременно. Всяка от колите се насочва към съседната. При такова движение става катастрофа - нито една от колите не може да продължи да се движи. Да се създаде следната програма.

а) процедура, която присвоява начални стойности на четири променливи, определящи позициите на четирите коли на графичния екран.

```
to init.pos :a
make "pos1 list :a / 2 :a / 2
make "pos2 list :a / 2 ( -  :a / 2 )
make "pos3 list - :a / 2 ( -  :a / 2 )
```

б) при условие, че съществува променлива от тип образ, съдържаща изображенията на четирите коли, да се дефинира процедура, която сг.здава четири костенурки - четирите коли, разполага ги в началните позиции и ги прави активни.

```
to init.cars
erturtle all
maketurtle 1 ( sentence :pos1 item 1 :shape)
maketurtle 2 ( sentence :pos2 item 2 :shape )
maketurtle 3 ( sentence :pos3 item 3 :shape )
maketurtle 4 ( sentence :pos4 item 4 -.shape )
tell allturtles
showturtle stamp
end
```

в) четири процедури, осъществяващи по една стъпка от движението на четирите коли, първата се насочва към втората, втората към третата, тя към четвъртата, а четвъртата към първата.

```
to 1to2
penup
setpos :pos1
seth towards :pos2
pendown fd 4
make"posl.old :pos1
make "pos1 pos
end
```

```
to
2to3
penu
p
        setpos
:pos2
seth towards
:pos3
```

```
to 3to4
penup
        setpos
:pos3
seth towards
:pos4
pendown fd 4
make "pos3 pos
```

```
to 4to1
penup
setpos :pos4
seth towards :pos1 .old
pendown fd 4
make "pos4 pos
end
```

г) процедура, осъществяваща пълното движение на колите, която извиква процедурите (в).

```
to go
    if near? [stop]
    ask 1 [1to2]
    ask 2 [2to3]
    ask 3 [3to4]
    ask 4 [4to1]
    go
    end
```

д) предикат, който дава резултат true, когато колите се сблъскват, и false, когато не са близо една до друга.

```
to near?
if abs :pos1 - :pos2 < 10 [output "true]
output "false
end
```

е) главна процедура на програмата с параметър за разстоянието между началните позиции на колите.

```
to cars :d
cs
        init.pos :d
init.cars
        go
        end
```

За изпълнение:

Еднократно се изпълнява следната последователност:

-В команден режим се изпълнява командата
make "shape run chooser "loadimage
появява се диалоговият прозорец.Образ и от него избираме рисунката с име cars, като задаваме нужните параметри;
- В прозореца на паметта се появява променливата shape.
Едва сега може да се стартира процедурата cars 200

26. Контрол на клавиатурата и мишката. Комуникация с мишката

CASE произволенвход [елементП елемент2 ...],
където всеки елемент на втория вход може да е:
- стойност - списък_от_инструкции,
- стойност - [израз],
- само ако е последният елемент на втория вход, списък_от_инструкции,
- само ако е последният елемент на втория вход, [израз]

Стойността на произволен вход се сравнява последователно с всяка стойност, която се съдържа във втория вход. Сравняването се извършва като се прилага предиката EQUAL? дотогава, докато за някоя стойност се получи резултат TRUE. Ако след тази стойност в списъка, даден като втори вход, следва списък от инструкции, те се изпълняват. В този случай CASE е команда. Операция е тогава, когато след стойност следва израз, той се пресмята и получената стойност става резултат от изпълнението на CASE.

Ако стойността на първия вход не е еквивалентна на никоя стойност, съдържаща се във втория вход, изпълнението на процедурата продължава. Ако вторият вход съдържа последен елемент, на който не съответства стойност, в зависимост от вида му - списък от инструкции или израз - той се изпълнява или пресмята и изпълнението съответно на командата или операцията CASE завършва.
Когато вторият вход не съдържа последен елемент, на който не съответства стойност, са възможни два случая. Ако в инструкцията, от която процедурата CASE се извиква, се предвижда тя да върне стойност, т.е. CASE се използва като операция, възниква грешка. Ако CASE се използва като команда, нищо не се случва, т.е. изпълнението на командата няма никакъв ефект.
Само ако стойността на произволен вход е дума, тогава се допуска елементите на втория вход да са от вида:
дума1 дума2 ... **flyMaN** списък_от_инструкции
или
дума1 дума2 ... **flyMaN** [израз].
В такъв случай стойността на първия вход се сравнява с всяка от дума1, дума2, **flyMaN,** вместо само с една стойност.
RUN списък_от_инструкции - изпълняват се инструкциите от дадения като вход списък от инструкции.
операция:RUN [израз] - пресмята се изразът, който е елемент на входа. Получената стойност става резултат от изпълнението на RUN.
CHOOSER вграденапроцедура - където вградена процедура трябва да има диалогов прозорец. Дава възможност да се отварят диалогови прозорци от потребителски процедури. Отваря се съответният диалогов прозорец, потребителят може да въведе данни в полетата на прозореца, след което са възможни два случая:
- прозорецът да се затвори като се натисне клавиша Enter или бутона Направи!, в този случай резултатът е списък, съдържащ инструкция - вградената процедура и входовете за нея, въведени с помощта надиалоговия прозорец,
- прозорецът да се откаже като се натисне клавиша Esc или бутона Откажи, в този случай резултатът е празният списък.
SETPHASE число, където число е цяло и е в интервала [-32767,32767]. Ликовете на костенурките са образи, но образите представляват последователност от кадри. В даден момент само един от кадрите на образа, (който се използва за лик на костенурка), може да се види на графичния екран (да представя костенурката) - нарича се текущ кадър на лика. PHASE установява текущ кадър на лика на всяка активна костенурка, такъв ще бъде съответният на даденото число кадър. Командата има смисъл, само ако се използва SETPHASEMODE "TRUE!
SETPHASEMODE логически_израз
SETPHASEMODE указва как да се използват кадрите на лика на всяка активна костенурка.
Ако стойността на входа е FALSE (по подразбиране), текущият кадър на лика ще зависи от текущата посока на костенурката. Текущият кадър се променя, когато костенурката сменя посоката си.
Ако стойността на входа е TRUE, текущият кадър на лика няма връзка с посоката на костенурката. Текущият кадър се променя като се използва командата SETPHASE. По такъв начин могат да се постигнат интересни анимационни ефекти.
Допуска се SETPHASEMODE (само името на командата) да се използва като вход на командата MAKETURTLE, например:

```
MAKETURTLE "СТЕФАН [0 0 180 PU SETPHASEMODE ST]
```

указва СТЕФАН да бъде анимационна костенурка, т.е. за тази костенурка се задава SETPHASEMODE "TRUE.
SETSHAPECOLOR цвят
SETSHAPECOLOR [ ]
Установява цвят на лика на всяка активна костенурка. Инструкцията SETSHAPECOLOR [ ] указва активните костенурки да нямат цвят| на лика. Операцията SHAPECOLOR връща цвета на лика на първата активна костенурка или [ ].
SPLITSCREEN След изпълнението на командата в работната област на Основния прозорец са видими и графичния, и текстовия екрани. Броят на видимите редове от текстовия екран може да се промени като се използва командата SETTS.
SETTS число (set text screen), където число е цяло. и е в интервала [2,25].
Задава колко редове от текстовия екран да се виждат тогава, когато в работната област на Основния прозорец са показани и графичния, и текстовия екрани, т.е. екранът е смесен (команда SPLITSCREEN).

6. Задача: Да се дефинира процедура, която осъществява рисуване с мишката върху графичния екран.

```
to free, hand
cs graphicscreen
free.handl
end

to free.handl
case readkey[
        0 [penup setpos mouse pendown] ~
        -1 -2 [setpos mouse] -
        99 [run chooser "setpc] ~
        27 [stop]]
free.handl
end
```

Забележка:
Ако искаме да променим цвета преди изпълнение на процедурата - изпълняваме командите:

```
make "n 5
setpc :n
free.hand
```

7. Задача: Да се дефинира процедурата Modify.free.hand с вход положително число A, която дава възможност да се рисува с мишката (т.е. с костенурка 0) върху графичния екран така, както се рисува с молив върху хартия.
Друга костенурка - костенурка 1 - да извършва същото движение, но модифицирано чрез умножение с коефициента A. По време на изпълнение на процедурата да е възможно да се сменят цветът и дебелината на молива на костенурка 1.

Задачата илюстрира една възможна връзка между средата Comenius Logi и аналитичната геометрия.

Анализ на задачата:

- Кривата, която потребителят рисува върху графичния екран, може да се разглежда като съставена от множество вектори. Всеки вектор има за начало точката от графичния екран, в която се намира костенурка 0, и край - точката в която се намира указателят на мишката. Костенурка 0 се насочва към указателя на мишката и при придвижването си към него начертава съответния вектор. Костенурка 1 трябва да чертае същият вектор, но с модифицирана дължина;
- Всяка точка от графичния екран може да се разглежда като втори край на радиус-вектор, чието начало е центърът на графичния екран. Координатите на радиус-вектора съвпадат с координатите на точката, която е негов втори край. Следователно, векторите, от които е съставена кривата, начертана от потребителя, се явяват разлики от такива радиус-вектори;
- Операцията LENGTH с вход координати на вектор, връща като резултат дължината на вектора.
- Процедурата MOVE (с първи вход *vect* - списък, съдържащ координати на вектор и втори вход pos - списък, съдържащ координати на точка) насочва костенурка 0 към точката, запомнена в *pos* и я придвижва напред толкова стъпки, колкото е дължината на вектора във *vect.* Костенурка 1 се насочва по посока на костенурка 0 и се придвижва напред толкова стъпки, колкото е дължината на вектора във *vect,* умножен по коефициента A;
- Управлението на процеса се реализира чрез процедурата WRITE, която следи натиснат ли е бутон на мишката или клавиш от клавиатурата. При натискане на левия бутон на мишката (резултат от операцията readkey е 0), моливите на двете костенурки се спускат;
- При плъзгане на мишката (резултатът от операцията readkey е -1), се извиква процедурата MOVE с първи вход - координатите на вектора с начало точката, в която се намира костенурка 0, и край - точката в която се намира указателят на мишката. Като втори вход на процедурата MOVE се подава точката, в която се намира указателят на мишката. Костенурки 0 и 1 начертават отсечки със съответната дължина;
- При отпускане на левия бутон на мишката (резултатът от операцията readkey е -2), моливите на двете костенурки се вдигат;
- При натискане на клавишите w и c, съответно с входове 119 и 99, потребителят може да задава дебелина и цвят на молива на костенурка 1.
- При натискане на клавиш ESC рисуването се прекратява;
- Ако не е натиснат бутон на мишката или клавиш, така че костенурки 0 и 1 да се движат едновременно с указателя на мишката, се изпълнява процедурата MOVE с входове, получени по същия начин, както в случая на плъзгане на мишката, като позицията на указателя на мишката се получава чрез операцията mousestate.
- В предложеното решение за улеснение графичният екран се рзделя на две, като с мишката (респ. костенурка 0) може да се рисува в горната половина, а в долната половина костенурка 1 следва движението на костенурка 0. За тази цел се използва командата window за всяка от костенурките.

Дефинира се променливата *pen* от тип образ и костенурка 1 получава за лик този образ.

```
to MODIFY.FREE.HAND :a
draw hideturtle penup setpos [-310 195] rt 90 pendown
fd 620 rt 90 fd 340 rt 90 fd 620 rt 90 fd 340 '
bk170rt90fd620penup
setpos [-250 100]
( window [[-310 195][310 25]])
maketurtle 1 [-250 -100 showturtle penup :pen]
ask 1 [( window [[-310 25][310 -145]])]
ask 1 [setpenwidth 2 setshapecolor 7]
splitscreen setts 3
print [Рисувайте с мишката!]
print [Натиснете < c > за избор на цвят и < w > за дебелина на молива!]
WRITE
end

        to MOVE :vect :pos
        ask 0 [setheading towards :pos fd LENGTH :vect]
        ask 1 [setheading ask 0 [heading] fd LENGTH (:vect * :a)]
        end

                to LENGTH :v

to WRITE
if key? [case readkey ~
                  [0 [ask [0 1][pendown]] ~
```

```
                -1 [MOVE ( mouse - (ask 0 [pos])) mouse] -2 [ask [0
        1][penup]] ~
        119 [ask 1 [run chooser "setpenwidth]] ~
        99 [ask 1 [run chooser "setpencolor]] ~
        27 [stop]]] ~
          [MOVE ( ( last mousestate ) - ( ask 0 [pos])) (last mousestate )]
      WRITE
end
```

За изпълнение:

- Еднократно се изпълняват следните команди:

1. Стартира се графичният редактор на образи и се начертава едно моливче;
2. От меню Файл се избира командата "Запази като" за да се запази рисунката под името PEN.LGW в директорията на Comenius Logo, (може и в друга директория), след което се излиза от графичния редактор на образи;.
3. В команден режим се изпълнява командата make "pen run chooser "loadimage появява се диалоговият прозорец Образ и от него избираме рисунката с име pen, като задаваме нужните параметри;
4. В прозореца на паметта се появява променливата *pen,*

- Едва сега може да се стартира процедурата modify.free.hand.

OUTSIDE? правоъгълник
(OUTSIDE? точка правоъгълник)
(OUTSIDE? костенурка правоъгълник)
В първата форма (един вход - правоъгълник) резултатът е TRUE, ако първата активна костенурка не се намира във външна за правоъгълника точка. В противен случай - първата активна костенурка е в зададената правоъгълна област или на някоя от границите на областта – резултатът е FALSE..

Във втората форма (входове точка и правоъгълник), резултатът е TRUE, ако точката е външна за правоъгълника. В противен случай резултатът е FALSE.

В третата форма (входове костенурка и правоъгълник), резултатът е TRUE, ако посочената костенурка се намира във външна за правоъгълника точка. В противен случай резултатът е FALSE. В тази форма OUTSIDE? можа да се използва за неактивни костенурки или за скрити костенурки.

PHASE

Ликовете на костенурките са образи, но образът представлява последователност от кадри. В даден момент само от един от кадрите на образа, който се използва за лик на костенурка, може да се види на графичния екран (да представя костенурката) - нарича се текущ кадър на лика. PHASE връща цяло число, съответстващо на текущия кадър на лика на първата активна костенурка.

Ако ликът на костенурката е празният образ, резултатът е 0. Изпълнението на операцията не зависи от това дали костенурката е видима или скрита (SHOWTURTLE или HIDETURTLE). Как се сменят кадрите на ликовете Всеки образ може да бъде лик на костенурка. Има два начина за смяна
кадрите на ликовете. Текущият кадър на лика ще зависи от текущата посока на костенурката. Текущият кадър се сменя, когато костенурката променя посоката ри. например, ако ликът има 24 кадъра, всеки кадър съответства на ъгъл! 15 градуса.
Текущият кадър на лика няма връзка с посоката на костенурката.
Текущият кадър се сменя като се използва командата setphase. По такъв начин могат да се получат интересни анимационни ефекти. По подразбиране се използва първия от двата начина. В този случай кадрите на лика се сменят при изпълнение на командите right, left или setheading. За да се използва втория начин, се изпълнява командата setphasemode с вход "true. В този случай кадрите на лика се сменят чрез командата setphase.

REPCOUNT (REPC)

Може да се използва само в рамките на командите REPEAT, FOR, WHILE и FOREACH, т.е. там, където има циклично изпълнение на инструкции. Дава като резултат цяло число, което показва за кой пореден път се изпълнява съответния цикъл.

WINDOW

(WINDOW правоъгълник) или (WINDOW [ ])
Установява работна област на всяка активна костенурка и определя поведението на костенурката, когато достигне границите на областта. В работната област костенурката може да се покаже, да чертае линии или многоъгълници (команди POLYGON и POLYGONLINE), да извежда текст (команда TTEXT), да запълва (команда FILL) и др.
В първата форма (без входове) се запазва последна, установена за съответната костенурка, работна област. При условие, че за костенурката не е била задавана работна област, такава става целият графичен екран. Ако е даден вход правоъгълник, работна става областта, която той задава. Когато командата се използва в третата форма (вход празният списък), работна област става графичния екран.
Когато костенурката пресече някоя от границите на работната си област (например, при изпълнение на командата FORWARD), тя излиза вън от областта. Работната област е част от безкрайна графична равнина.
По подразбиране (вж. командата MAKETURTLE) работната област на всички костенурки е графичният екран и те могат да го напускат (вж. команда WRAP).

LET дума1 произволен_вход1
(LET дума1 произволен_вход1 дума2 произволен_вход2 . . .)
В първата форма (два входа) се дефинира *локална* за текущо изпълняваната процедура променлива с име дума и стойност произволен вход. Когато командата се използва във втората форма (п входа), се дефинират повече от една локални променливи.За разлика от LET, командата LOCAL създава локални променливи, но не присвоява начални стойности.

INC име_на_променлива

(INC име_на_променлива число), където стойността на променливата е число.
Увеличава с 1 стойността на зададената променлива, в случай че командата има само един вход. Във втория случай (два входа) стойността на променливата се увеличава с толкова, колкото е даденото число.
COUNT произволен_вход
Ако входът е дума, резултатът е броят на знаковете в думата. Ако входът е списък, резултатът е броят на елементите му. Ако входът е образ, резултатът е броят на кадрите му.
) LOCAL дума1
LOCAL списък_от_думи
(LOCAL дума1 ...)
В първата форма (един вход) се създава локална за текущо изпълняваната процедура променлива с име зададената дума. Командата не присвоява начална стойност (това трябва да се направи допълнително чрез някоя от командите MAKE или NAME). Когато се използват втората и третата форма, се създават повече от една локални променливи. За разлика от LOCAL, командата LET създава локални променливи и присвоява на начални стойности.
WORD дума1 дума2
(WORD дума1 ...)
Резултатът е дума, получена чрез обединяване на входовете в посочения ред.
8. Задача:
Да се дефинира процедура tri.helik със следния ефект: три клоуна - хеликоптери да "летят" върху графичния екран и **fpfcr** се виждат като сянка. Чрез щракване с мишката да се извика ЩяеТ помощника Правоъгълник и да се зададе правоъгълна област от графичния екран. Когато някой от клоуните навлезе в тази област, сянката му да се изменя в пълния му - ' . . цветен образ. Когато клоунът напусне областта, отново да става сянка.
В задачата се съчетава работа с множество костенурки.
Анализ на задачата:
- Първоначално се създават три костенурки - сенки и три костенурки-клоуни. Така костенурките - клоуни ще са по-млади и ликовете им ще застъпват ликовете на костенурките - сенки. Чрез командата window се задават прозорци: за костенурките-сенки се задава целия графичен екран, а за костенурките клоуни - областта, която потребителят задав|а чрез помощника Правоъгълник. Така извън зададената област ще се виждат само костенурките - сенки;
- Всичките шест костенурки са анимационни и кадрите им ще се сменят чрез командата setphase;
- Допълнително е зададен правоъгълник [-200 150] [200 -150], за да бъде движението на костенурките обозримо за потребителя;

```
to tri.helik
cs
erturtle allturtles
setts 2
print [Като плъзгате мишката задайте правоъгълна област!]
maketurtle "shadowl [-165 44 :shadow setphasemode showturtle penup tell]
maketurtle "shadow2 [-27 -91 :shadow setphasemode showturtle penup tell]
maketurtle "shadow3 [127 3 :shadow setphasemode showturtle penup tell]
maketurtle "clounl [-165 44 :cloun setphasemode showturtle penup tell]
maketurtle "cloun2 [-27 -91 :cloun setphasemode showturtle penup tell]
maketurtle "cloun3 [127 3 :cloun setphasemode showturtle penup tell]
ask [shadowl shadow2 shadow3][window]
ask [clounl cloun2 cloun3][( window last chooser "inside? )]
tri.helikl
end

to tri.helikl
if key? [if readkey = 27 [stop]]
repeat 3 [ ask ( sentence word "cloun repc word "shadow repc) ~
              [setphase phase + 1 fd 2 rt 10 - random 21 ~
              if outside? [[-200 150][200 -150]][rt 180 fd 2]]]
tri.helikl
```

За изпълнение:
- Еднократно се изпълняват следните команди:
1 .В команден режим се изпълнява командата
make "cloun run chooser "loadimage
появява се диалоговият прозорец Образ и от него избираме рисунката с име *cloun,* като задаваме нужните параметри. В прозореца на паметта се появява променливата *cloun;*
2.Отваря се графичният редактор на образи. Зарежда се чрез командата "Отвори" графиката cloun.lgp, като и трите кадъра се променят в сиво.
3. Запазват се кадрите под името shadow (или в поддиректория image или в произволна директория и се излиза от графичния редактор.
4. В команден режим се изпълнява командата
make "shadow run chooser "loadimage
появява се диалоговият прозорец Образ и от него избираме рисунката с име *shadow,* като задаваме нужните параметри. В прозореца на паметта се появява променливата *shadow;*
- Едва сега може да се стартира процедурата tri.helik