LOGO ELEMENTS IN GEOGEBRA

M. M. Abdurazakov[1], **D. D. Gadjiev**[2], **A. R. Yesayan**

[1] ***The Russian Academy of Education***
119121, Russia, Moscow, ul. Pogodinskaya, 8

[2] ***Indian River State College, Florida State University, USA***
3209, Virginia Avenue, Fort Pierce, FL 34981, United States

Abstract

The presented topic deserves attention because of its extreme usefulness, simplicity of its impact on the initial training in programming. This article discusses the basics of programming and features of work in the Logo environment, focused on the education and use of information technology, includes tools that allow you to display images in the style of commands of the Logo programming language.

The inclusion of the elements of the Logo environment and the recursiveness underlying this language into interactive mathematical environments, for example, in GeoGebra, allow learners to create programs for solving a particular class of problems, which allows us to speak about the object approach when studying the content line of the school informatics course "Basics of algorithms and programming", and the procedural programming approach characteristic of the Logo environment allows the formation of the corresponding features (logical, algorithmic thinking of students). A sufficiently easy to learn programming language in the Logo environment allows students to create programs and demonstrate unlimited possibilities for implementing mnemonics. The examples are series of simple examples of execution of command systems by the virtual executor "turtle" and the results demonstrating the creation of images with the help of a turtle are given.

Keywords: basics of programming, Logo environment, basics of Logo programming language, contractor, system of commands.

DOI: 10.32517/0234-0453-2019-34-4-38-48

For citation:

Abdurazakov M. M., Gadjiev D. D., Yesayan A. R. Logo elements in GeoGebra. *Informatika i obrazovanie — Informatics and Education*, 2019, no. 4, p. 38–48.

Received: February 6, 2019.

Accepted: March 19, 2019.

About the authors

Magomed M. Abdurazakov, Doctor of Sciences (Education), Docent, the Russian Academy of Education, Moscow, Russia; abdurazakov@inbox.ru; ORCID: 0000-0001-8085-8477

Djavanshir D. Gadjiev, Candidate of Sciences (Physics and Mathematics), Professor of Mathematics and Natural Sciences, Indian River State College, Florida State University, USA; dgadjiev@irsc.edu

Albert R. Yesayan, Doctor of Sciences (Education), Professor, Russia

Introduction

The main goal of the school course "Informatics and ICT" is to form the basis of algorithmic thinking in students, which means the ability to solve problems of various origins, requiring the preparation of the action plan from the final set of commands and control structures for the performer to achieve the final result.

The purpose of learning of Algorithmization is to master the students' methods of building algorithms and the formation of the foundations of algorithmic culture.

The success of the students in mastering the topic "Basics of Algorithmization and Programming" depends largely on the general training skills they have acquired, starting with the propaedeutic course of informatics and ICT. Therefore the skills that form the basis of algorithmic thinking should be formed as early as possible, which, means, starting with the propaedeutic course. When studying algorithms in the propaedeutic course, the development aspect is fundamental, and algorithmic thinking is increasingly recognized as a vital skill, as evidenced by many publications [1–6].

Recently, interactive mathematical environments (IME) have become widely used in educational activities around the world as the means of developing ICT competence of students, and for developing research skills in mathematics and computer science lessons, extra classes, electives, and etc., particularly, increasing motivation in the study of mathematics and computer science using IC is a global trend, e.g., such as the basics of modeling, algorithms and programming, creating and researching material and information models.

The scientific and pedagogical literature describes various approaches to learning algorithmization and programming. The role of mathematics in the development of logical thinking is extremely large [7], the use of algorithmic method and the formation of students algorithmic thinking is becoming an actual topic of today, and the skills of using algorithms in practical work are components of computer literacy.

In solving mathematical, engineering and any other problem in practice, there are two approaches to the computer implementation of models and the solution of these problems using application software.

The first approach is based on the fact that:

- to carry out simple calculations, the user must master algorithmization and programming with the help of software tools or a set of educational worlds Logo, Idol, etc.
- algorithmization, learn one or more high-level programming languages, such as Pascal, C, etc.

The second approach is to use specialized software systems for automation of mathematical and engineering calculations, for example, MathCad, MatLab, Mathematica, Maple, MuPAD, GeoGebra, etc., designed for professional users.

Among the programs developed for education in secondary school, we want to highlight the software GeoGebra [8]. The main ideology of the implementation of GeoGebra is to acquire the students the geometric, algebraic, and numeric representations in an interactive mode [9, 10].

The GeoGebra package is a free (free) educational mathematical program that combines geometry, algebra and the beginning of analysis, has a rich ability to work with functions, it can be used to build graphs of functions, solve problems in planimetry and stereometry; it allows you to visualize mathematics, conduct experiments, calculations, etc.; perform the construction of various objects and consider them in dynamics [11].

Didactically reasonable application of software packages in the educational process helps to increase the level of fundamental character of mathematical education [12]. Integration of visualization capabilities of learning outcomes, for example, integration of Logo elements into the GeoGebra package contributes to more effective implementation of the most important didactic principles "from simple to complex" and "maximum visibility and ease of work", and the "friendly interface" of such packages develop and form students' skills of independent learning and cognitive activity necessary for career guidance and further training at the University [13, 14].

As a didactic tool for mastering the basics of programming, it is convenient to use instructional performers of algorithms (virtual performers Turtle, Grasshopper, Draftsman, etc.) and instrumental programming environments or a set of training worlds Logo, Python, CuMir, Game LabKudu, etc. Many prominent scientists, methodology researchers and experts from Russia, European Union and United States believe that the use of performers from a methodological point of view is very effective. The implementation of the principle of visibility is clear from the many virtues of the performer's tasks and increases the interest in the process of solving the problem.

At the laboratory of Artificial Intelligence of the Massachusetts Institute of Technology (MIT, USA), Professor Seymour Papert generated a simple and elegant programming language Logo in 1967. Moreover, he produced an environment for the implementation of Logo programming language, where the learners can easily learn programming, develop thinking, generate creative and research skills, obtain cognitive independence [15]. The name of the language comes from the Greek "logos" — "word, thought, meaning, idea".

There is the performer of commands "turtle" there in the Logo programming, which can move "turtle" forward on the screen for a specified number of steps and change the direction of its movement. Meantime, during its movement, the turtle can leave a mark on the screen in the form of line segments, tracing certain shapes, or it moves without leaving a mark. The set of commands, understood by the performer "turtle", forms the basis of the Logo language. The virtue in the simplicity of the Logo language have made a revolutionary impact on the initial training in programming [16, p. 343–349; 17, p. 130–139; 18–20].

1. Turtle control commands

The children of a primary school, or, moreover, pre-school age learners are able to understand the Logo language. Therefore, there are variety of software environments exist there. These environments are mainly focused on learning and rely on further utilization of information technologies and tools, which are included there. The utilization of information technologies along with programming tools used allow to display images in the style of Logo programming language commands. These Logo programming language command have been described in GeoGebra [21], where the Turtle control was implemented by a set of commands: *Turtle, TurtleBack, TurtleForward, TurtleDown, TurtleUp, TurtleLeft and TurtleRight*, performed via the input line, as well as with the "▶ *Play*" и "⏸ *Pause*" buttons located in the lower left corner of the "*Canvas*" panel. At the same time on the canvas may be several turtles.

We will focus on a brief description of above mentioned commands and, moreover add-on some additional commands.

Turtle initiation and drawing modes.

A. *Turtle[]*
B. *TurtleUp[name]*
C. *TurtleDown[name]*

A) At the command *A*, a bug is installed at the origin with the current possible direction of movement — "to the right". By default, her name is *turtle1 (turtle2, turtle3, ...)*. To assign a custom name to a turtle, the command *A* should be executed in the form *name = Turtle []*. It is permissible to initiate several turtles on one canvas at a time, and in the future each of them can participate in creating of an image independently of each other.

B) At the command of *B*, the drawing mechanism of the turtle is blocked, that is, the mode is activated, in which the turtle leaves no trace behind itself (the pen rises).

C) At the command *C*, the drawing mechanism of the turtle is activated, that is, the mode is activated

in which the turtle leaves a trail when moving (the feather is lowered). This mode is the default.

Move the turtle back and forth.

A. *TurtleForward*[*name*, *dist*] B. *TurtleBack*[*name*,]

A) At the command *A*, the turtle with the name [*name*] is arranged moving forward at a distance [*dist*] in the current direction. At the same time, if the "⏸ *Pause*" button is visible, the bug starts moving right away, and if the "▶ *Play*" button is visible, only after clicking on it with the left mouse button. However, you can start the movement of the turtle using the *StartAnimation*[]. The turtle, whether leaves a mark or not, which is dependent on the current *TurtleDown* or *TurtleUp* mode, e.g. such as, in a case, when clicking *TurtleDown* the drawing is done, but with *TurtleUp* — no. Next, manipulating with a turtle, it is, for the most part, useful to bring a grid to the canvas, which can be done through the "*Settings*" panel.

The image displayed by the turtle is in the form of a trace. The turtle itself is a single object, which, formally, is the *Style* command as output, practically, independent from the user. Through the "*Settings*" panel you can only change the color of the bug's trail. However, this change will be valid only if the trace is selected (active). In addition, you can change the design of the turtle itself by setting the name of the file with its new image on the *Style* tab of the specified panel. The removal of a trail can only be done if it is completely removed.

B) At the command *B*, the movement of a bug with the name [*name*] is generated at a distance [*dist*] back, that is, in the direction opposite to the current one. In other words, the turtle, while is moving, goes backwards, without changing directions. Leaving or not leaving a trace in this case is understood as of the execution of the command *A*. The "⏸ *Pause*"-"▶ *Play*" switch button also acts as in *A*.

Rotate the bug at a given angle.

A. *TurtleRight*[*name*, *angle*] B. *TurtleLeft*[*name*, *angle*]

A) At the command *A*, the turtle turns from the current direction to the right (clockwise) at an angle' measurement in radians. To *angle* measured in degrees, it should be written in the form of *angle*. At the same time, if the "⏸ *Pause*" button is visible, then the *A* turtle starts to rotate immediately, and if the "▶ *Play*" button is visible, then, only after clicking on it with the left mouse button.

B) At the command *B*, the turtle is rotated from the current direction to the left (counterclockwise) at an *angle* in radians. To angle measured in degrees, it should be written as in the form of *angle*. At the same time, if the "⏸ *Pause*" button is visible, then the A turtle begins to turn immediately on A, and if the "▶ *Play*" button is visible, then, only after clicking on it with the left mouse button.

Transferring a bug to a new position.

A. *SetCoords*[*name*, *x*, *y*]

A) At the command *A*, the turtle named [*name*] is transferred to the position (*x, y*) without drawing and without changing a direction.

2. Examples of drawing a turtle

This section contains a series of simple examples demonstrating the creation of images with the help of a bug, including use of single and multiple cycles.

Example 1.

Write the code by which the turtle draws an equilateral triangle with side length of 2.

Decision.

The code, which is required to solve the proposed problem, is shown below at the left. According to it, the turtle with the name *tu* is the first code to set it at the origin with the direction of movement along the abscissa to the right (*tu=Turtle* []). Furthermore, *tu*, moves two units in the specified direction (*TurtleForward* [*tu,* 2]) and then turns left 120° (*TurtleLeft* [*tu,* 120°]). The last two commands are repeated two more times. Afterwards, by executing presented commands and further, by clicking on the "▶ *Play*" button, the image is displayed at the right next to the commands will be drawn (fig. 1).

```
tu=Turtle[]
TurtleForward[tu, 2]
TurtleLeft[tu, 120°]
TurtleForward[tu, 2]
TurtleLeft[tu, 120°]
TurtleForward[tu, 2]
TurtleLeft[tu, 120°]
Click(▶)
```

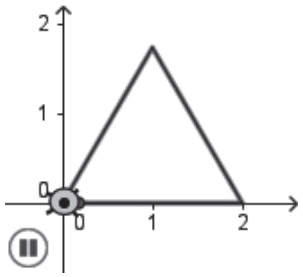


Fig. 1

Organization of cyclic calculations.

A. *Repeat*[*n*, *com*1, *com*2, ..., *comk*] B. *SetValue*[*obj*1, *obj*2] C. *SetValue*[*list*, *n*, *obj*] D. *SetValue*[*bo*, *s*] (*s* = 0, 1)	

A) By the command *A*, the repeated execution of the *com*1, *com*2, ..., *comk* script commands is carried out *n* times. These commands are listed and described in the "*Scripting*" help section. In particular, all

commands for controlling the turtle belong to the script commands. Next, it is important to note that the nesting of *Repeat* commands into each other allows you to organize calculations with multiple cycles.

B) At the command *B*, the existing free object *obj*1 is replaced with the object *obj*2.

C) At the command *C*, the free object, which is the nth element of the list, is replaced with the *obj object* (n = 1, 2, …, *Length*[*list*]).

D) On the *D* command, the existing Boolean variable *bo* takes the Boolean value *true* if s = 1, and *false* if s = 0.

Example 2.

1) $f = x$
SetValue [*f, RandomElement* [{*sin*(*x*), *cos*(*x*), 2, *ln*(*x*), *tan*(*x*)}]]

In this case, at the canvas displays a graph of the function $f(x) = x$. Further, each execution of *SetValue* leads to a pseudo-random replacement of this graph by the graph of one of the functions *sin*(*x*), *cos*(x), 2, *ln*(*x*), *tan*(*x*).

2) $k = 0$
SetValue [*k*, k + 1]

In this case, the variable k takes the value *0*. Further, each execution of *SetValue* results in an increase in the value of k by 1. Such syntax *SetValue* is often used to organize cyclic calculations using the *Repeat* command. At the same time, assignment of the form $k = k + 1$ (and even k = *number*) is not allowed.

Example 3.

Solve the problem of Example 1 using cyclic calculations.

Decision.

A couple of commands *TurtleForward*[*tu,* 2] *and* *TurtleLeft*[*tu,* 120°] in the code of Example 1 is repeated 3 times (fig. 2). You may reduce multiple use of the code by using the command loop *Repeat* to writing it in the following form:

```
tu=Turtle[]
Repeat[3, TurtleForward[tu, 2],
TurtleLeft[tu, 120°]]
StartAnimation[]
```

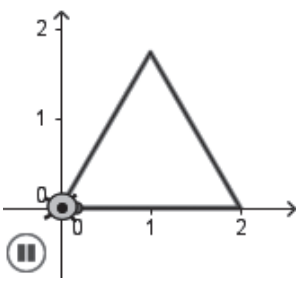


Fig. 2

Example 4. Simple cycle.

Write the code that displays the correct n-gon (n = 3, 4, 5, …) with a side of length a.

Decision.

From the code from the previous example, the solution to this problem can be obtained as follows. We introduce the variable n with an integer value to indicate the number of sides of the polygon and the variable a with a non-negative value for the second argument of the *TurtleForward* command. In order to draw a regular n-gon with side a length, it is required to turn the turtle to the left not by 120°, but by $360°/n$. The latter value is obtained as follows.

Due to the property of the sum of the angles of n-gons (fig. 3), which states that the sum of all internal angles of a regular n-gon is $(n - 2) \cdot \pi$, which corresponds to $(n - 2) \cdot 180$, then one of its inner angles is $180° - 180°(n - 2) / n = 180°(1 - (n - 2) / n) = 360°/n$. From here, the following code can be proposed for solving the problem:

```
n=7
a=2
tu=Turtle[]
Repeat[n, TurtleForward[tu, a],
TurtleLeft[tu, 360°/n]]
Click(▶)
```

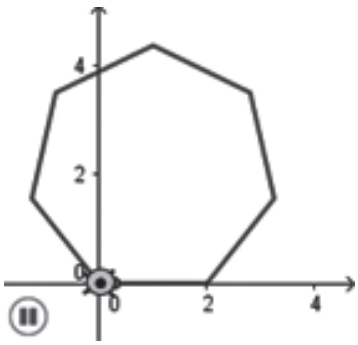


Fig. 3

Comment. According to the code of Example 4, "*circles*" can also be output, setting a sufficiently large n.

Example 5. Double cycle.

Write a code that displays m nested n-gons with a common vertex at the origin and one side which is set on the positive direction of the X-axis (fig. 4).

Decision.

The code and the result of its execution with n = 3 and n = 4 for solving the proposed problem are shown below. The length of the side of the first polygon is a, the next is $a + h$, the third is $a + 2 \cdot h$, and so on. Here, m and n play the role of control variables for the outer and inner cycles, respectively.

```
a=1
h=0.5
m=6
n=3
tu=Turtle[]
Repeat[m,
Repeat[n, TurtleForward[tu, a],
```

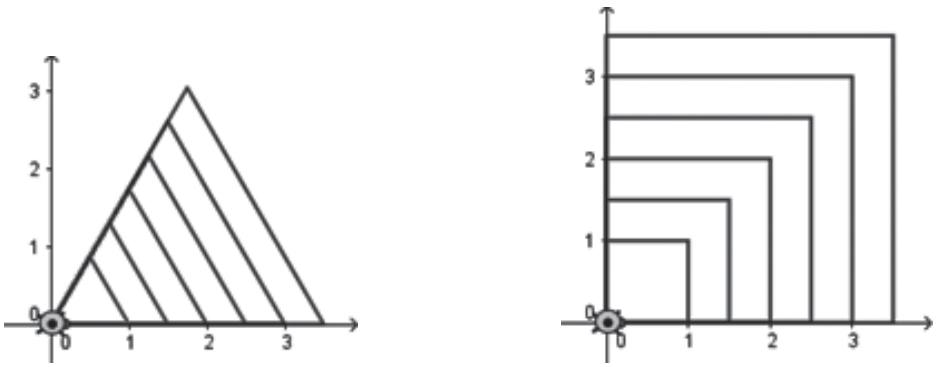


Fig. 4. Output by the code of Example 5 with n = 3 and n = 4

TurtleLeft[*tu*, 360°/*n*]],
SetValue[*a*, *a*+*h*]]
Click(▶)

n = 3
n = 4

Example 6. Triple cycle.

Suppose you want to display images obtained by the code from the previous example with *n* = 3, 4, …, *s*, and the bug should do it with one stroke, that is, for a given *s* at one run.

Decision.

The code shown below solves the problem. In this case, all initial values for variables are entered with a single *Execute* command. There is given below figure 5, which shows the result of the code execution with *n* = 4.

Execute[{"*a*=1", "*h*=0.5", "*m*=6", "*n*=3", "*s*=4"}]
tu=*Turtle*[]
Repeat[*s*,
Repeat[*m*,
Repeat[*n*,
TurtleForward[*tu*, *a*],
TurtleLeft[*tu*, 360°/*n*]],
SetValue[*a*, *a*+0.5]],
TurtleUp[*tu*],
TurtleForward[*tu*, *m*/2+*n*-1],
TurtleDown[*tu*],
SetValue[*a*, 1],
SetValue[*n*, *n*+1]]
Click(▶)

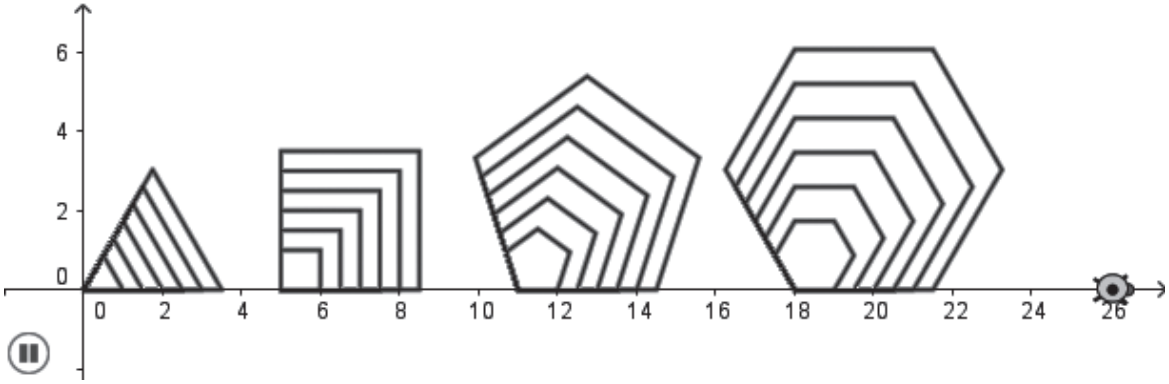


Fig. 5. Output by the code of example 6 with s = 4 (in one stroke)

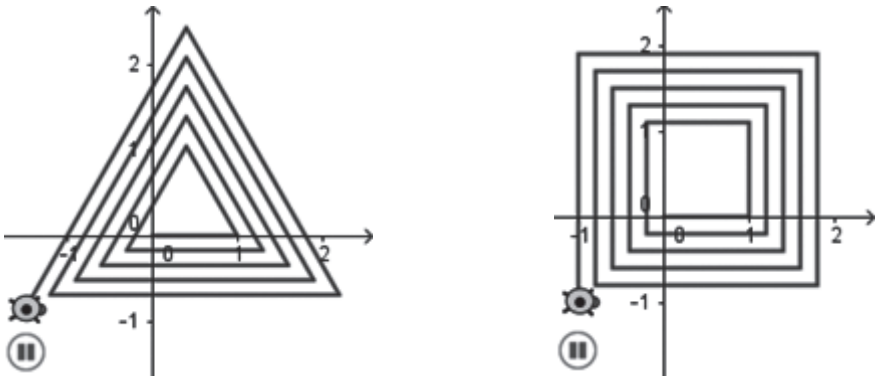


Fig. 6. The output of the spirals according to the code of example 7 with n = 3 and n = 4

Example 7. Spiral.

Write a code to display an image in the form of a triangular, quadrangular, etc. spiral (fig. 6).

Decision.

The following solution shows the possible code and the results of its execution.

In the code: n is the type of spiral (n = 3 is triangular, n = 4 is quadrangular, etc.), a is the length of the initial side of the helix, where h is the increment in the length of the side of the helix with each turn, m is the number of sides of the helix.

```
n=3
m=15
a=1
h=0.2
tu=Turtle[]
Repeat[m,
TurtleForward[tu, a], TurtleLeft[tu, 360°/n],
    SetValue[a, a+h]]
```

n = 3, m = 15, a = 1, h = 0.2
n = 4, m = 20, a = 1, h = 0.1

3. Derivation of correct {*n*/*m*} stars

For the consideration of the following examples, we have defined the concept of a "star".

In order to do so, we have placed around the circle n points at equal distance from each other, that is, to set the correct n-gon. Further, each obtained point is connected by a segment with the m-th ($1 \le m \le n/2$) point of the circle by moving it clockwise. The resulting figure is called a *{n/m} star* or the *star-shaped* form of the original convex n-gon. The vertices of the *{n/m}* star is considered to be the starting points, and the sides are the intervals. The points of intersection of the sides between themselves are not considered to be the tops of the stars. Thus, an *{n/m}* star has n vertices and m sides. A regular convex n-gon may have several stellate forms. Namely, when n is even, it has (n – 4) / 2 star-shaped forms, and when n is odd, it has (n – 3) / 2 star-shaped forms [20]. If, for given n and m, a star exists and, moreover, n and m are co-prime, then the star constitutes to a single object, otherwise it breaks up into several star-shaped forms.

Example 8.

Conclusion of the correct *{n/m} star (n, m* are co-prime*)*. Print the pentagonal, two heptagonal and nine-square regular star-shaped n-gons.

Decision.

To build an *{n/m} star*, one needs to know the magnitude of the rotation angle when changing the direction of movement of the turtle. This angle can be calculated using the formula $\alpha = 360° \cdot m / n$. In fact, if the *{n/m} star* exists, then to build it, the turtle should visit each of the n vertices, and therefore turn around a total angle of $360° \cdot m$.

Hereafter, since the angles of rotation at each vertex are the same, one such angle is equal to $\alpha = 360° \cdot m / n$.

Thus, for a pentagonal {5/2} star, this angle is 144°, for a heptagonal {7/2} star, the angle is 720°/7, for a heptagonal {7/3} star, the angle is 1080°/7 and for a nine-corner star {9/4}, the angle is 160°. The code for outputting the {5/2} star is shown below. If we want to edit the code for other *star*(*s*), then it is obvious that there is only concern exists with the changes in the values of n and m occurred there. The results of the calculations are shown in figure 7.

```
n=5
m=2
a=2
tu=Turtle[]
Repeat[n, TurtleForward[tu, a],
    TurtleLeft[tu, 360°*m/n]]
Click(▶)
```

n = 5, m = 2
n = 7, m = 2
n = 7, m = 3

Example 9. Derivation of correct {11/*m*} stars.

Write a code to build a correct *{11/m} star* with a turtle with m = 2, 3, 4 and 5. With one launch, the output should be carried out at once of all the required stars, that is, be carried out using the "one pen stroke".

Decision.

The code and the result of its implementation are given below (fig. 8).

```
Execute[{«n=11», «m=2», «a=2», «h=4.5»,
    «tu=Turtle[]»}]
Repeat[4],
```

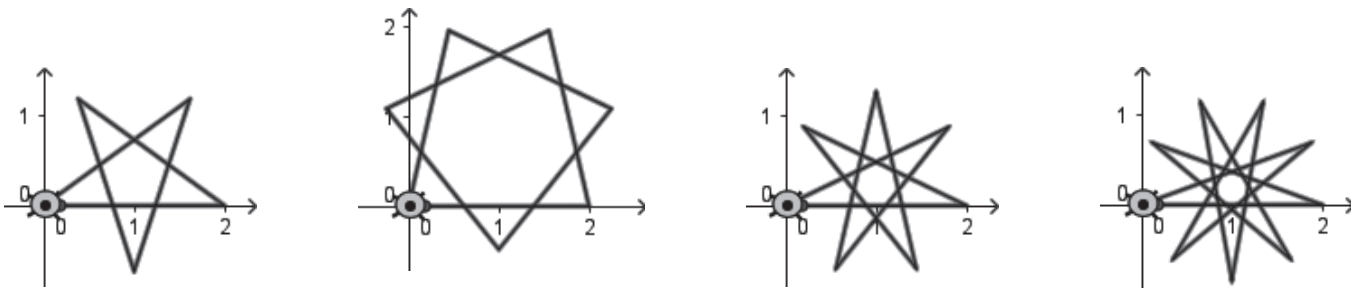


Fig. 7. Output by the code of example 8 for different values of n and m

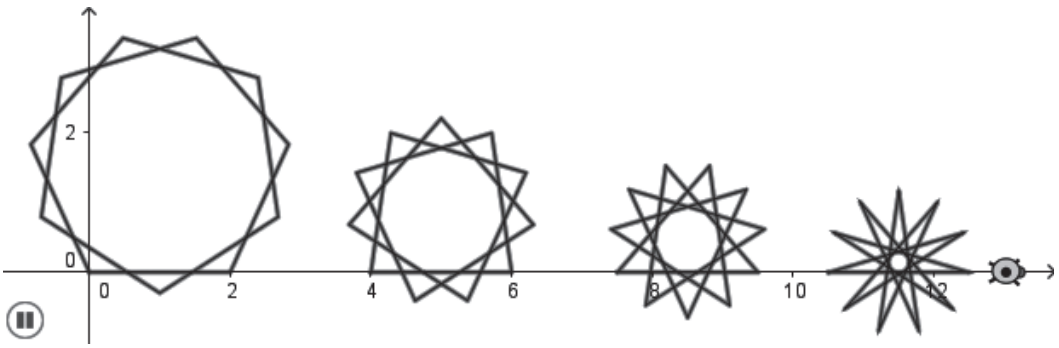


Fig. 8. Conclusion of {11/m} stars with m = 2, 3, 4 and 5

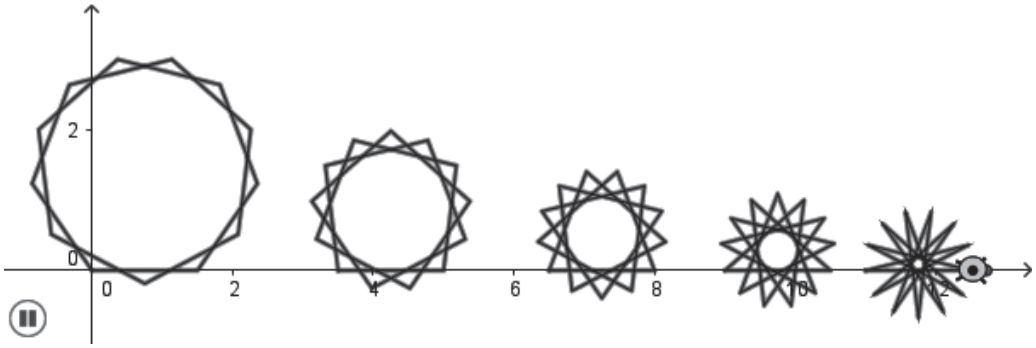


Fig. 9. Conclusion of {13/m} stars with m = 2, 3, 4, 5 and 6

Repeat[*n*, *TurtleForward*[*tu*, *a*],
TurtleLeft[*tu*, 360°**m*/*n*]], *SetValue*[*m*, *m*+1],
SetValue[*h*, *h*-0.5], *TurtleUp*[*tu*],
TurtleForward[*tu*, *h*], *TurtleDown*[*tu*]]
Click(▶)

Example 10. Derivation of correct {13/*m*} stars.

Write a code to build a correct {13/*m*} star with a turtle with *m* = 2, 3, 4, 5 and 6. At one launch, the output should be carried out at once of all the required stars, that is, to be carried out with "one stroke of a pen".

Decision.

The code for solving the proposed problem is easily obtained by cosmetic editing of the code from the previous example. The resulting code and the result of its implementation are given below (fig. 9).

Execute[{"*n*=13", "*m*=2", "*a*=1.5", "*h*=4.5",
"*tu*=*Turtle*[]"}]
Repeat[5,
Repeat[*n*, *TurtleForward*[*tu*, *a*],
TurtleLeft[*tu*, 360°**m*/*n*]], *SetValue*[*m*, *m*+1],
SetValue[*h*, *h*-0.5], *TurtleUp*[*tu*],
TurtleForward[*tu*, *h*], *TurtleDown*[*tu*]]
Click(▶)

4. Drawing a turtle with the help of controls

It is quite simple by managing the "Turtle" (fig. 10). commands entered through the input line. However, it is even easier to perform this, if you activate the corresponding commands with the help of control elements. We described below how we handle this work at the presented mini-application there. The set of controls for such application should look like the one shown in figures 11–15 and the individual controls should act as follows:

- in a blank document, the initiation of a turtle *(Turtle [] command)*, that is, placing it at the origin and indicating the direction of movement to the "*right*" should be done by clicking the left mouse button on the general purpose button

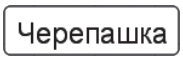


Fig. 10

- setting "*Draw*" mode is to leave a trace when moving by a bug and "*Do not draw*" is to move a turtle without drawing (*TurtleDown and TurtleUp commands*) should be done with one *bo* slider with two logical values *0 and 1.* When *bo* = 0, the mode should be "*Do not draw*", and when *bo* = 1 is "*Draw*". On the canvas, this element may look as follows

Fig. 11

- moving the turtle forward or backward to a distance *dist (TurtleForward and TurtleBack* commands) should be implemented by an associated pair of controls: a slider and a general-purpose button. On the canvas, these elements may look like this.

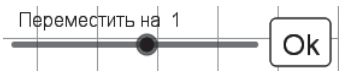


Fig. 12

This bundle of elements should work as follows. The slider is used to set the displacement amount *dist* with the update *dist* above the slider. Each click of the left mouse button on the "*Ok*" button triggers one of the *TurtleForward* or *TurtleBack* commands using the current *dist* value. More precisely, if *dist* > 0, then TurtleForward is moving the turtle to a distance *dist* ahead in the current direction, and if *dis t* < 0, then *TurtleBack* is moving the turtle to a distance |*dist*| in the opposite direction to the current;

- the turtle turns to the left, that is, counterclockwise, or to the right, that is, clockwise to the angle α (*TurtleLeft and TurtleRight* commands) must be implemented by an associated pair of control elements: a slider and a general-purpose button. On the canvas, these elements may look like as this:

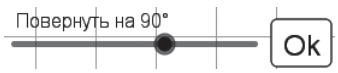


Fig. 13

This bundle of elements should work as follows. A slider sets the angle of rotation α with the update α above the slider. Each click of the left mouse button on the "*Ok*" button is activated by one of the *TurtleLeft* or *TurtleRight* command by using the current command value. More precisely, if α > 0, then *TurtleLeft* works by turning the turtle at an angle α counterclockwise, and if α < 0, then *TurtleRight* works by turning the turtle at an angle |α| clockwise;

- the rapid movement of the turtle to the position (*x, y*) of the canvas (*SetCoords* command) must be achieved by the associated triple of control elements: two sliders and a general-purpose button. On the canvas, these elements may look like this:

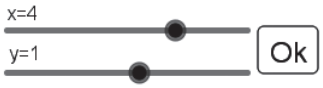


Fig. 14

This bundle of elements should work as follows. Sliders set a specific point (*x, y*), and clicking the left mouse button on the "*Ok*" button triggers the *SetCoords* command, which moves the turtle to the (*x, y*) position without drawing and changing the current direction of movement.

To create the required application, it is significant to show how to form the above-mentioned control elements and make them work exactly as described above.

1) Черепашка button. Use the "OK Button" tool to display a general-purpose button on the canvas. Using the context menu, open the "*Settings*" panel, on the "*Basic*" tab, create the "*Turtle*" heading, and on the "*Scripts*" tab, assign the "*On Click*" event handler as a *geogebra* script consisting of one command *tu = Turtle* []. This will ensure the execution of this script when you click the "*Ok*" button with the left mouse button, which means it will initialize the turtle with the name *tu* and output *it* in the current position with the initial direction of movement "to the right". Note that in a blank document, the default starting position is the origin.

2) Slider with the inscription "Не рисовать Рисовать". Using the "a=2 Slider" tool, we will display on the canvas a control element with the same name as the name *bo*. Using the context menu, make the slider to be invisible label, and through the "*Settings*" panel, set the smallest value for it as 0, the largest value as 1, and step is 1.

Thus, the *bo* slider will be appointed to two values as 0 and 1, which will determine the drawing modes "*Draw*" and "*Do not draw*". Since we refused to display the slider's label, instead of it, using the "*Text*" tool, we will output the corresponding text message "*Do not draw Draw*".

3) A bunch of slider and "Переместить на 1 Ok" button. Using the "a=2 Slider" and "OK Button" tools,

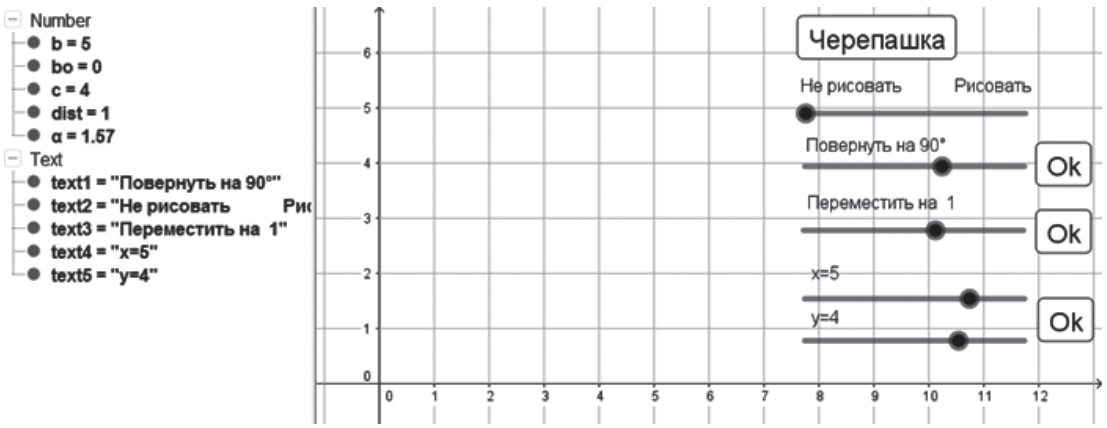


Fig. 15. A set of controls for manipulating a bug

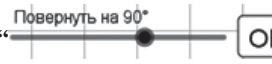
we will display on the canvas a slider with the name *dist*, a general purpose button with the title "*Ok*" and place them one after the other.

For a *dist* slider through its context menu, its label is invisible. Using "*Settings*" panel we define for *dist* the smallest value as –5, the largest value as 5 and the step — 0.1. With this slider we will set the magnitude and direction of movement of the turtle. Namely, when you click the general purposed button "*Ok*" with the left mouse button, the movement is assumed: by distance |*dist*|: in the current direction, if *dist* > 0, and in the opposite direction, if *dist* < 0. Since we have refused to display the slider's label, instead of it, using the "ABC Text" tool, we will display a text inscription with constant and variable parts in the form "*Move to* [*dist*]". If we create an inscription, its variable part must be formed in the bordering box.

For the general purpose click the button "*Ok*", open the "*Settings*" panel and on the "*Scripts*" tab, assign the "*On Click*" event handler a *geogebra*-script as:

If[*bo*==1, *TurtleDown*[*tu*], *TurtleUp*[*tu*]]
TurtleForward[*tu*, *dist*]

Then each time you click the left mouse button on the "*Ok*" button, this script will be executed, that is, the bug will move a distance |*dist*| in the desired direction, and with or without drawing, as required by the value of the *bo* slider.

4) A bunch of slider and " 

" button. Using the "a=2 Slider" and "OK Button" tools, we will bring up the slider button, the general purposed button with the title "*Ok*" on the canvas and place them one after the other.

For the slider α, via the context menu, to make its label invisible, and on the "Settings" panel we define the smallest value as –360°, the maximum value as 360° and the step — 5°. This slider will set the current direction of movement of the turtle. Since we refused to display the slider's label, instead of it, using the "ABC Text" tool, we will display a text inscription with constant and variable parts in the form "Rotate to [α]". If we want to create an inscription, its variable part must be formed in the bordering box.

For the general purposed button "*Ok*", open the "*Settings*" panel and on the "*Scripts*" tab, assign the "*On Click*" event handler as a *geogebra* script from one *TurtleLeft*[*tu,* α] command. Then each time you click the left mouse button on the "Ok" button, this script will be executed, that is, the bug will turn to the angle |α|: counterclockwise, if α > 0, and clockwise, if α < 0.

5) You can move the turtle to any position of the canvas by using the *dist* slider and the associated "*Ok*" button. But there is no control over the speed of movement of the turtle. Therefore, to quickly move the turtle without drawing you need to create a different mechanism. And it can rely on the *SetCoords*[*tu, x, y*] command previously described. We show how this can be done. We will display on the canvas two sliders b and c with values *b* and *c* from –10 to 10 in increments of 0.1 and the associated general purposed button "*Ok*". The first slider will set the value of the abscissa *x*, and the second is the value of the ordinate *y* at the new position (*x, y*) of the turtle. Hide the marks of the sliders, and instead of them we will display text labels with constant and variable parts of the form "*x*=[*b*]" and "*y*=[*c*]". Moving the turtle to position (*x, y*) will be by clicking the left mouse button on the "*Ok*" button, to which the *SetCoords script*[*tu, x, y*] must be assigned to the "*On Click*" event.

6) Through the context menu, the positions of the sliders and text labels will be (*Absolute Position On Screen*).

An application for manipulating a turtle with the help of control elements has been created and you can get drawing of associated work (see fig. 16). Note that all controls of the bug could be brought to "*Cloth 2*".

Comment. Since a single object is drawn by a turtle, it is impossible to remove or correct its incorrectly drawn parts. Using the *Ctrl+z* key, the entire trail of the turtle is removed from the canvas, and through the "*Objects*" panel you can delete the turtle itself. It is impossible to restore the image using the *Ctrl+y* key. It is also impossible, having saved the trail, to delete or make an invisible bug. In addition, if you write an image created by a turtle into a *ggb*-file, then when you open it, the trace will disappear. You can save such a picture by writing it, for example, in the form of a *png*-file.

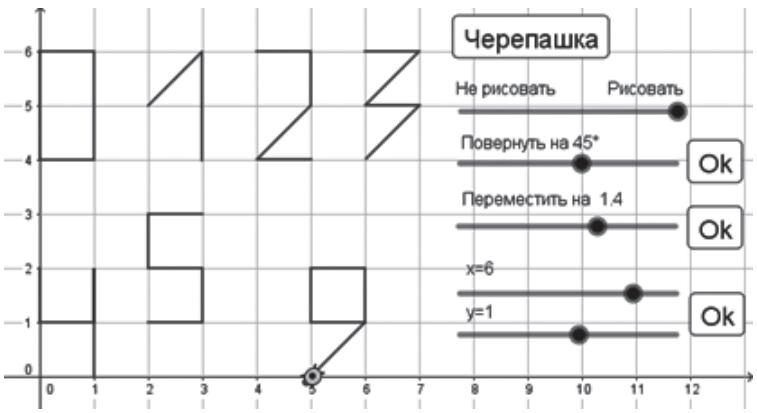


Fig. 16. Example of drawing a turtle with the help of controls

Finding

The main idea of transferring the Logo elements to the programming plane in the GeoGebra computer environment is to discover the fascinating programming power for learners. Such knowledge in programming allows the learners to be well-familiar with the computer tools, too. Moreover, acquired programming power alongside with the utilized computer tools authorize the learners to build new knowledge of the concepts of the objects along with their properties and operations on the objects. Meantime, the learners are covering the mainstream of tasks by creating images, models and sketches through integration of the interface of the computer environment with the elements of the programming tools. This approach of using computer programs in training allows you to:

Firstly, the environment of GeoGebra with the elements of the *Logo* allows for a new assessment of the role of new technologies in education, which consists of not only in demanding new competencies from teachers, but also in using an integrated environment to expand the capabilities of students' research skills.

Secondly, computer technology and computer environment with integrated programming elements allow us to improve the quality in the field of educational and cognitive activity, which are subjected to the training purposes. Such training or professional development creates the opportunity to further betterment of cognitive activity, where learners cultivate research spirit to implement the didactic principle of visibility to visualize the results by solving various mathematical problems.

References

1. *Gebauer H., Hromkovič J., Keller L., Kosírová I., Serafini G., Steffen B.* Programming in LOGO. Available at: http://abz.inf.ethz.ch/wp-content/uploads/unterrichtsmaterialien/primarschulen/logo_heft_en.pdf

2. Informatics in Schools. Fundamentals of Computer Science and Software Engineering. *Proc. 11th Int. Conf. on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer International Publishing, 2018. 396 p. DOI: 10.1007/978-3-030-02750-6

3. *Hromkovic J.* Introduction to Programming with LOGO Spanish edition, translated from German to Spanish by Angelica Herrera Loyo. Mexico, Cimat Newton Editing and Educational Technology, 2015.

4. *Hromkovic J.* Einführung in die Programmierung mit LOGO. Springer Vieweg, 2014. 259 p. DOI: 10.1007/978-3-658-04832-7

5. *Staub J.* xLogo online — a web-based programming IDE for Logo. ETH Zurich, 2016. DOI: 10.3929/ethz-a-010725653

6. *Hromkovic J., Kohn T., Komm D., Serafini G.* Combining the Power of Python with the Simplicity of Logo for a Sustainable Computer Science Education. Springer, Cham, 2016. DOI: 10.1007/978-3-319-46747-4_13

7. *Knut D. E.* Algoritmicheskoe myshlenie i matematicheskoe myshlenie [Algorithm thinking and mathematical thinking]. (In Russian.) Available at: http://kph.npu.edu.ua/!e-book/clasik/data/math/knut.html

8. GeoGebra. Available at: http://www.geogebra.com

9. *Aleksanyan G. A., Chernyaeva E. P.* Primenenie vozmozhnostej programmy GeoGebra pri izuchenii temy "Prostejshie preobrazovaniya grafikov" [Application of the possibilities of the GeoGebra program in the study of the theme "Simple transformation of schedules"]. *Sovremennye problemy nauki i obrazovaniya — Modern Problems of Science and Education*, 2017, no. 5, p. 244. (In Russian.) Available at: http://science-education.ru/ru/article/view?id=26815

10. *Kazakova E. V.* Vvedenie v sredu GeoGebra [Introduction to the GeoGebra]. *Materialy IV Vserossijskoj nauchno-metodicheskoj konferentsii s mezhdunarodnym uchastiem. V. R. Majer "Informatsionnye tekhnologii v matematike i matematicheskom obrazovanii" [Proc. IV All-Russ. Sci. and Method. Conf. with international participation. V. R. Mayer "Information Technologies in Mathematics and Mathematical Education"]*. Krasnoyarsk, Krasnoyarskij gosudarstvennyj pedagogicheskij universitet im. V. P. Astaf'eva, 2015, p. 28–30. (In Russian.)

11. *Makarova N. V., Kochurova E. G., Nikolaichuk G. S., Nilova Yu. N., Titova Yu. F.* Informatika i IKT 8-9 klassy [Informatics and ICT 8-9 classes]. Saint Petersburg, Piter, 2010. 416 p. (In Russian.)

12. *Pushkaryeva T. P., Stepanova T. A., Kalitina V. V.* Didakticheskie sredstva razvitiya algoritmicheskogo stilya myshleniya studentov [Didactic tools for students' algorithmic thinking development]. *Obrazovanie i nauka — The Education and Science Journal*, 2017, no. 9, p. 126–143. (In Russian.) DOI: 10.17853/1994-5639-2017-9-126-143

13. *Abdurazakov M. M.* Puti optimizatsii soderzhaniya obshheobrazovatel'nogo kursa informatiki v osnovnoj shkole [Ways to optimize the content of general education informatics course in basic school]. *Sovremennye informacionnye tehnologii i IT-obrazovanie — Modern Information Technologies and IT-Education*, 2015, vol. 11, no. 1, p. 284–290. (In Russian.)

14. *Abdurazakov M. M., Esayan A. R., Monahov V. M.* Prognosticheskij potentsial optimizatsionnoj metodologii i tekhnologii proektirovaniya metodicheskoj sistemy obucheniya s napered zadannymi svojstvami [The prognostic potential of the optimization methodology and technology of designing a methodical system of education with predetermined properties]. *Sovremennye informacionnye tehnologii i IT-obrazovanie — Modern Information Technologies and IT-Education*, 2016, vol. 12, no. 3-2, p. 6–10. (In Russian.) Available at: http://sitito.cs.msu.ru/index.php/SITITO/article/view/114

15. *Papert S.* Perevorot v soznanii: Deti, komp'yutery i plodotvornye idei [Revolution: Children, computers, and fruitful ideas]. Moscow, Pedagogika, 1989. 224 p. (In Russian.)

16. *Yesayan A. R., Chubarikov V. N., Dobrovolsky N. M., Yakushin A. V.* Tekhnicheskoe risovanie v Tikz [Technical drawing in Tikz]. Tula, Izdatel'stvo TGPU im. L. N. Tolstogo, 2014. 402 p. (In Russian.)

17. *Yesayan A. R., Yakushin A. V.* O realizatsii Logo v LaTeX [On the implementation Logo in LaTeX]. *Čebyševskij sbornik — Chebyshevskii Sbornik*, 2014, vol. 15, no. 3, p. 131–140. (In Russian.) DOI: 10.22405/2226-8383-2014-15-3-131-140

18. *Nikolov R., Sedova E.* Nachalo informatiki. Yazyk Logo [The beginning of informatics. Logo language]. Moscow, Nauka, 1989. 176 p. (In Russian.)

19. Tutorial:GeoGebra Turtle. Available at: https://wiki.geogebra.org/en/Tutorial:GeoGebra_Turtle

20. Stellation. Available at: https://en.wikipedia.org/wiki/Stellation

21. *Abdurazakov M. M., Esayan A. R., Nimatulaev M. M.* Dinamicheskie modeli i ehksperimental'naya proverka gipotez v GeoGebra [Dynamic models and experimental hypothesis testing in GeoGebra]. *Sbornik statej uchastnikov Mezhdunarodnoj nauchno-prakticheskoj konferentsii "Sovremennye obrazovatel'nye web-tekhnologii v sisteme shkol'noj i professional'noj podgotovki" [Proc. Int. Sci. and Practical Conf. "Modern educational web-technologies in the system of school and vocational training"]*. Natsional'nyj issledovatel'skij Nizhegorodskij gosudarstvennyj universitet im. N. I. Lobachevskogo, Arzamasskij filial, 2017, p. 265–271. (In Russian.)

ЭЛЕМЕНТЫ LOGO В GEOGEBRA

М. М. Абдуразаков[1], Д. Д. Гаджиев[2], А. Р. Есаян

[1] Российская академия образования
119121, Россия, г. Москва, ул. Погодинская, д. 8

[2] Индиан Ривер Колледж, Государственный университет штата Флорида, США
3209, Virginia Avenue, Fort Pierce, FL 34981, United States

Аннотация

Заявленная тема заслуживает внимания в силу своей чрезвычайной полезности, простоты своего влияния на начальное обучение программированию. В данной статье рассматриваются основы программирования и особенности работы в среде Logo, ориентированной на обучение и использование информационных технологий, включаются средства, позволяющие выводить изображения в стиле команд языка программирования Logo.

Включение элементов среды Logo в интерактивные математические среды, например в GeoGebra, и лежащие в основе этого языка рекурсивность дают возможность обучающимся создавать программы для решения конкретного класса задач, что позволяет говорить об объектном подходе при изучении содержательной линии школьного курса информатики «Основы алгоритмизации и программирования», а характерный для среды Logo «процедурный подход программирования» позволяет формировать соответствующие стили мышления обучающихся (логическое мышление, алгоритмическое мышление). Достаточно простой для усвоения язык программирования в среде Logo позволяет обучающимся создавать программы и демонстрировать неограниченные возможности реализации мнемоники. В статье приведены серии простых примеров выполнения систем команд виртуальным исполнителем «черепашка» и результаты, демонстрирующие создание изображений с помощью «черепашки».

Ключевые слова: основы программирования, среда Logo, основы языка Logo, исполнитель, система команд.

DOI: 10.32517/0234-0453-2019-34-4-38-48

Для цитирования:

Абдуразаков М. М., Гаджиев Д. Д., Есаян А. Р. Элементы Logo в GeoGebra // Информатика и образование. 2019. № 4. С. 38–48. (In English.)

Статья поступила в редакцию: 6 февраля 2019 года.

Статья принята к печати: 19 марта 2019 года.

Сведения об авторах

Абдуразаков Магомед Мусаевич, доктор пед. наук, доцент, Российская академия образования, г. Москва, Россия; abdurazakov@inbox.ru; ORCID: 0000-0001-8085-8477

Гаджиев Джаваншир Джебраил, канд. физ.-мат. наук, профессор математики и естественных наук, Индиан Ривер Колледж, Государственный университет штата Флорида, США; dgadjiev@irsc.edu

Есаян Альберт Рубенович, доктор пед. наук, профессор, Россия

НОВОСТИ

Ростех разработал первый суперкомпьютер на базе процессоров «Эльбрус»

ИНЭУМ им. И. С. Брука, входящий в концерн «Автоматика» госкорпорации Ростех, совместно с группой компаний РСК разработал первый суперкомпьютер на российских восьмиядерных микропроцессорах «Эльбрус-8С». Он предназначен для организации высокопроизводительных вычислений, обработки больших данных и решения задач, требующих обеспечения максимального уровня информационной безопасности.

Суперкомпьютерное решение состоит из стоек, содержащих компактные четырехпроцессорные блейд-серверы с жидкостным охлаждением. Такой способ теплоотведения позволил снизить габариты модуля, что дает возможность разместить до 153 вычислительных узлов в одной серверной стойке. Их суммарная вычислительная мощность составляет до 75 терафлопс двойной точности — одна стойка может выполнять до 75 трлн операций с плавающей запятой за секунду. При этом итоговая мощность суперкомпьютера практически не ограничена, так как стойки можно объединить в единый вычислительный кластер.

Каждое ядро восьмиядерного микропроцессора «Эльбрус-8С» с отечественной архитектурой «Эльбрус» способно исполнять до 25 операций за один такт. Архитектура процессора позволяет значительно наращивать производительность прикладных программ за счет их оптимизации.

«Четвертое поколение отечественных микропроцессоров “Эльбрус” подходит для выполнения сложных математических расчетов и моделирования нейронных сетей, которые являются основой технологий искусственного интеллекта. Основное отличие наших ИТ-систем — высокий уровень информационной безопасности. “Эльбрусы” гарантированно не содержат “закладок”, позволяющих удаленно влиять на работу системы и незаконно снимать информацию. Это позволяет использовать наш суперкомпьютер в чувствительных сферах, связанных с обработкой конфиденциальной информации», — сообщил исполнительный директор Ростеха Олег Евтушенко.

Отличительным свойством разработанного суперкомпьютера является высокая энергоэффективность. На его охлаждение расходуется менее 6 % всего потребляемого электричества. Подсистема электропитания обеспечивает возможность эксплуатации в сложных условиях с расширенным диапазоном напряжений и повышенным уровнем помех. Серверы имеют встроенный функционал для удаленной диагностики и управления.

Возможно применение различных технологий коммутации, в том числе использование отечественной сети межмашинного обмена. Конструктив стойки позволяет заменять вычислительные узлы, блоки питания и модули гидрорегулирования в режиме «горячей замены» без прерывания работоспособности комплекса.

(По материалам, предоставленным пресс-службой госкорпорации Ростех)