6/1 (2008), 209–229
tmcs@inf.unideb.hu
http://tmcs.math.klte.hu

Teaching Mathematics and Computer Science

Shall we use one more representation?

Suggestions about establishing the notion of recursion in teaching informatics in primary schools

Cecília Sitkuné Görömbei

Abstract. Among the most prominent developmental tasks of primary school education one finds increasing pupils' cognitive capacity with especial regard to observing, interpreting, coding and proving skills, which form an integral part of information and communication culture.

Info-technology (problem solving with the tools and methods of informatics), a subject matter within informatics, provides outstanding opportunities to reach the aims outlined above.

This study presents methodological ideas related to the subfield *Algorithmization and data modelling* of *Info-technology.* More specifically, it presents teaching methods to be applied while establishing the notion of recursion in grades 3–8 of primary education, and at the same time it also focuses on various realization possibilities of the prominent developmental tasks mentioned above.

Key words and phrases: primary education, teaching informatics, programming, Logo, recursion.

ZDM Subject Classification: P40, R90, Q60.

Introduction

Among the most prominent developmental tasks of primary school education one finds increasing pupils' cognitive capacity with especial regard to observing, interpreting, coding and proving skills, which form an integral part of information and communication culture. One of the most important tasks of teachers is to strive to improve pupils' reasoning power, first of all systematization, experimental observation, argumentation and problem solving with especial regard to enhancing analysis, synthesis, comparison, generalization, concretization and their everyday use. It is also important that pupils be able to apply the acquired knowledge in novel situations. Hence, forming new ideas, i.e. developing creative thinking comes into prominence. At the same time it is essential to focus on the following areas: decision making, examining various alternatives, comprehensive use of these alternatives, evaluation, argumentation and selecting the best options. [12] *Info-technology (problem solving with the tools and methods of informatics)*, a subject matter within informatics, provides outstanding opportunities to reach the aims outlined above.

This study presents methodological ideas related to the subfield *Algorithmization and data modelling* of *Info-technology.* More specifically, it presents teaching methods to be applied while establishing the notion of recursion in grades 3–8 of primary education, and at the same time it also focuses on various realization possibilities of the prominent developmental tasks mentioned above. My suggestions are based on experience gained while teaching informatics both in classrooms and study circles for grades 3–4 and 5–7 and while preparing pupils for various programming competitions.

Algorithmization and data modelling

Aims, tasks, tools

The aim to be achieved while teaching the subfield Algorithmization and data modelling is that pupils be able to recognize and express the algorithmic particulars appearing in their environments in various forms. In order to attain this goal the National Core Curriculum specifies the following developmental tasks:

Grades 1–4:

- Recognizing, expressing and executing simpler algorithms.
- Interpreting data used in algorithms.
- Describing common data (numbers, texts, drawings…).
- Using a simple development system.

Grades 5–6:

- Computer-aided accomplishment of algorithms in order to solve a given task.
- Determining results on the basis of known data while solving a problem.

- Getting acquainted with tables, diagrams and data organizing.
- Task solving with simple development systems.

Grades 7–8:

- Designing, accomplishing and executing algorithms in order to solve a given problem.
- Tools of algorithmic abstraction, the principle of step by step improvement.
- The relation between the relevant data and the result in problem solving.
- Differentiating and handling datum and complex data.
- Task solving with development systems.

According to the core curriculum various programming languages and development environments can be used as a computational tool for teaching algorithmization. The most preferred one is the Hungarian speaking Comenius Logo and its upgraded version, Imagine, which can be used up to grade 8. (Nevertheless, experience of the National Logo General Competition indicates that it is a popular and effective programming environment not only in grades 3–8, but also in grades 9–10.

Recursion

The term recursion is of Latin origin, it means running backwards, return in time, repetition. In Tótfalusi István's book entitled *Idegenszó-tár* (*Dictionary of Foreignisms*) one finds the following interpretation: "A series of operations comprising repeated steps, where the result always returns to serve as the starting point for further operations." [21]

The notion is used in various scientific fields with specific interpretations. For instance, in literature it surfaces in connection with the almost infinite stories of folk tales, which go there and back,—*Mikkó és a makaróni (Mikko and the Macaroni); A kóró és a kismadár (The Dry Stalk and the Little Bird)*—[8]. The notion may also be connected with belletristic works, for instance it appears in the essays entitled *The Thousand and One Nights* or *Partial Enchantments of the Quxiote* by Luis Borges as a typical example of the literary version of recursion, i.e. the constructional method similar to the principle of Russian "Matryoshkas": numbers of story boxes nested into one another. [2]

Within mathematics the field of recursive functions is the most well-known, its typical examples—mentioning now only the classic ones—are the factorial function, the Fibonacci sequence and the Ackermann function.

The notion of recursion is also used in information science; within the field of programming it means a kind of special repetition, a self-invoking algorithm. One subtype of recursive algorithms realizes computational solutions of recursive mathematical problems, while the other subtype comprises explicit computational problems. For instance certain algorithms of searching and sorting belong to the latter.

Many have tried to define the notion of recursive algorithms in various ways. Obviously, the principle is formulated uniformly, namely, the subprogram (procedure or function) invokes itself; however, there are various explanations related to the special approaches taken by the individual authors. [1, 7, 9, 19]

It is also common in the works mentioned above that the characteristics of recursive algorithms are systemized same important properties are foregrounded.

For instance: If a given task is solved recursively, then

- we look for the simplest case where the solution is obvious,
- we find out how the task can be solved in the following way: via repeated simplifications we get to the simplest case meanwhile supposing that in each simplified case the task has been solved (hence, in each case the solution in question can be used). [9]

Or: In a recursive routine there must be

- a variable which keeps on changing during the calls and in principle it can reach a threshold limit.
- a command which directs the variable towards the threshold limit.
- a halting condition, which decides whether the given variable has reached the threshold limit. If yes, then the recursive calls terminate. [1]

Terminology

A procedure is called *directly recursive* if it invokes itself. It is called *indirectly recursive* if it invokes a procedure which (directly or indirectly) invokes the given procedure.

A procedure is called *simply recursive* if it contains one and only one direct recursive call of itself, and this call is not located within a repetition.

An algorithm is a *multiple recursion* if it contains more than one recursive calls. [9]

A procedure is a *right recursion* if it invokes itself at the very end of the procedure, it is a *left recursion* if it invokes itself at the beginning, and it is a *general recursion* if it invokes itself in the middle. [22]

Step by step recursion

What kinds of tasks shall we choose?

There are various types of tasks that lend themselves to introducing the notion of recursion. In what follows we will consider possibilities that arise while constructing geometrical figures, i.e. when we have pupils draw figures with the help of the turtle.

Even the simplest geometrical figures including repetitions are suitable to gain such experience that facilitates establishing the notion in question. Such figures are for instance regular polygons, spirals and figures which are similar in a mathematical sense. [22, 13]

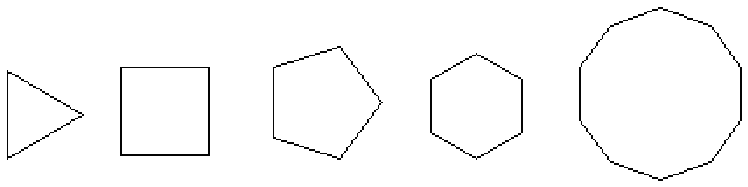

Figure 1. Regular polygons

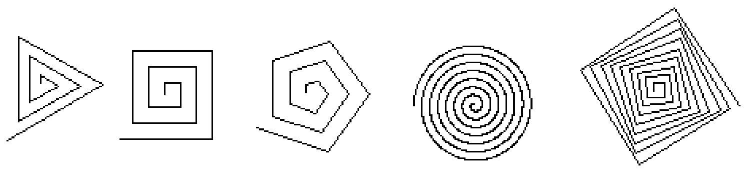

Figure 2. Spiral variants

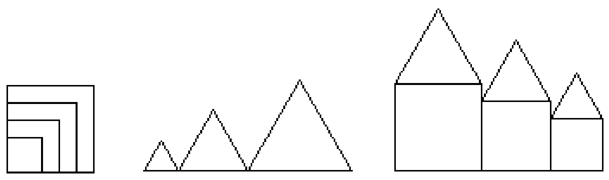

Figure 3. Similar figures

Besides the figures presented above self-similar figures (fractals)—such as the Koch-curve and trees—also provide great possibilities to establish and deepen the notion of recursion. However, construction of the latter requires different and special solutions from a methodological point of view, for that reason we do not deal with them in the present study.

What kinds of steps are involved in the process of forming the notion? How can we facilitate attainment of the objectives of the National Core Curriculum with these?

First,—after having taught the basic commands (Level 1: sequence)—we prepare introduction of recursion via *repetition* (Level 2: iteration) and via forming the notion of *procedure (Level 3: procedure)*; in other words we provide a possibility to acquire preliminary and necessary knowledge. It is recommended that pupils acquire these already in grades 1–4. Pupils familiarize themselves with a simple development system (a version of Logo), they execute simple algorithms and interpret data used in algorithms. Understanding is facilitated in a way which suits best the characteristics of primary school pupils, i.e. using graphic presentation of abstract notions.

Second, pupils’ attention may be directed to recursion while they are drawing the figures mentioned above (Level 4: recursion), but only on the level of referring to the notion, trying it, experimenting with it; simply showing pupils that there is such a possibility in Logo programming language.

Pupils will see that repetition can also be achieved when the procedure invokes itself. Self-calling is not yet bound to a condition, hence, recursion results in an infinite procedure. In grades 1–4 this topic can be raised only in study groups, but in grades 5–6 we can deal with it in course of lessons. The problem in question (recursion is infinite) is extremely suitable for developing thinking and creativity. Graphic presentation still considerably facilitates understanding, and it also helps to reconcile the different functioning of the left and right hemispheres.

Third, the solution of the problem which is related to the answer of the previous problem (recursion is infinite) comes to the foreground (Level 5: regular recursion). This process highlights an important property of recursive algorithms, namely, that their halting must be bound to a condition. At the same time this is a good occasion to introduce the notion of selection. Characterizing the notions in question can be a task in grades 5–8. Examining the alternatives, looking for solutions of various kinds and realizing these solutions develop pupils’ thinking and their evaluation and reasoning capacity well.

Fourth, the acquired knowledge is used to create more difficult figures (Level 6: experimenting with recursive figures). While experimenting with drawing particular figures further notions related to recursion are being formed, for instance the notions of left and right recursion, general recursion, levels of recursion (Level 7: applying experience gained while experimenting in different problem situations). It is especially important to focus on these possibilities in grades 7–8.

How can we put to practice the methodological steps outlined above?

A possible way to accomplish the first three steps will be presented below with the help of two concrete examples (cf. Figures 1 and 2).—The tasks in question are also suitable for introducing the notion of recursion when they are treated independently from one another.—The fourth step is exemplified in the third exercise, with the help of drawing similar figures (cf. Figure 3).

Example 1: drawing regular polygons

While learning the basic commands pupils willingly draw regular polygons. However, if a polygon has too many sides, then one must type quite a lot in order to draw the figure. Thus, pupils themselves demand introducing the command of repetition. Introducing the notion of recursion via drawing polygons is not as spectacular as drawing spirals or fractals; however, because of its simplicity it lends itself particularly well to the given objective.

First step: preparation

We draw a square with the help of sequence and iteration, then the notion of procedure and that of parameter are introduced. More specifically:

Level 1 (sequence):

Only basic commands are used to complete the drawing. The commands of moving are the following: '`forward`', '`back`'. The commands of turning are '`left`', '`right`'.

The actual commands are the following:

```
forward 100 right 90 forward 100 right 90
forward 100 right 90 forward 100 right 90
```

The complete drawing is shown below:

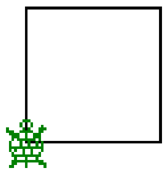

Figure 4. The complete drawing and the position of the turtle before and after drawing

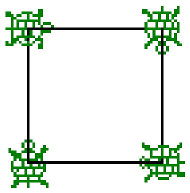

Figure 5. The complete drawing and the positions of the turtle at different stages of drawing

Level 2 (iteration):

Besides the basic commands iteration (cycle organizing) commands are also used. One of the tools to realize the cycle in the specified number of steps is the command '`repeat`'.

The actual commands are the following:

```
repeat 4[forward 100 right 90]
```

Level 3 (procedure):

Procedures are written, first one without a parameter, then a parametric one.

The key word of writing a procedure in Comenius Logo is '`to`', the closing command of the procedure is '`end`'.

a. The procedure without a parameter:

```
to square_1
    repeat 4[forward 100 right 90]
end
```

The procedure is called by '`square_1`', i.e. its name.

b. The recursive procedure when one parameter is used:

```
to square_2 :sidelength
    forward :sidelength right 90
```

```
        square_2 :sidelength
    end
```

The procedure is called by '`square_2 100`', i.e. its name and a specified value for its parameter.

If the connection between the polygon's number of sides and the angle of turning is discovered, then on this level a "universal" polygon drawing procedure can be written, which, with an adequate parameter setting, can draw polygons of any numbers of sides and of any side lengths.

c. The procedure when both number of sides and side length are parameterized:

```
    to polygon_1 :sidelength :sidenumber
        repeat :sidenumber ~
        [forward :sidelength right 360/:sidelength]
    end
```

Second step: the notion of recursion

Level 4: (recursion):

Applying the procedures the possibility of recalling is presented, i.e. recursion.

a. The recursive procedure without a parameter:

```
    to square_3
        forward 100 right 90
        square_3
    end
```

b. The recursive procedure when one parameter is used:

```
    to square _4 :sidelength
        forward :sidelength right 90
        square_4 :sidelength
    end
```

c. The recursive procedure when two parameters are used:

```
    to polygon_2 :sidelength :sidenumber
        forward :sidelength right 360/:sidenumber
        polygon_2 :sidelength :sidenumber
    end
```

NB: Procedure *a* of level 4 can also be applied after procedure *a* of level 3.

The process of drawing can be made more suggestive if the command 'Step by step execution' is chosen from the menu 'Settings'.

The figure becomes more interesting and the infiniteness of the process is even more suggestive when the command of line colouring is used, selecting the colour randomly from the colour palette of Comenius Logo. The figure becomes even more descriptive if a short waiting time is inserted before recalling.

In order to accomplish these aims the procedures given above must be slightly modified, namely, before recalling the following commands are added:

`setpencolor random 15;` and `wait 100`

The recursive solutions applied on level 4 provide a good occasion for interesting observations and they also raise a new problem. The figure is the same (apart from side length) as the ones prepared on level 1, 2 or 3, however, the turtle does not stop, it is "running" round and round on the perimeter of the square (or another type of polygon)—hence, the procedure has become *infinite*.

Third step: the notion of regular recursion, components of "good" recursion

Level 5 (regular recursion):

The problem which has arisen on level 4 can be handled in the following way adhering to the principle of recursion:

- In order to specify the halting condition another parameter is introduced, which serves to signal levels of recursion.
  (This is the variable which keeps on changing during the recursive calls and in theory it can reach a *threshold limit.*)
- The value of this parameter is decreased by 1 on each call.
  (This is the command which directs the parameter mentioned above towards the threshold limit.)
- The resumption condition is determined. If the value of the parameter does not meet the value specified in the resumption condition, then the procedure is finished, there are no more recalls.
  (This is the *halting condition*, which *decides* whether the variable in question *has reached the threshold limit.* If the answer is yes, then there are no more recursive calls.)

a. The procedure with regular recursion, only levels of recursion is parameterized:

```
to square_5 :level
    if :level > 0 [forward 100 right 90 ~
                   square_5 :level-1]
end
```

b. The procedure with regular recursion, size of figure and levels of recursion are parameterized:

```
to square_6 :sidelength :level
    if :level > 0 [forward :sidelength right 90 ~
                   square_6 :sidelength :level-1]
end
```

c. The procedure with regular recursion, size of figure, number of sides and levels of recursion are parameterized:

```
to polygon_3 :sidelength :sidenumber :level
    if :level > 0 [forward :sidelength ~
      right 360 / :sidenumber~
      polygon_3 :sidelength :sidenumber :level-1]
end
```

When the procedures are called it is important how the beginning value of the parameter '`:level`' is set. For instance, in case of drawing a square the call '`square_5 4`' or the call '`square_6 100 4`' is needed in order to get the correct solution. If the value of the parameter '`:level`' is smaller than 4, then the procedure does not draw a square, if it is bigger than 4, then one side may be redrawn more than one times, and this is redundant according to the interpretation of the task.

Example 2: drawing spirals

Spirals often occur in nature, too. Pupils are especially motivated by the movement experienced while drawing the figure. The process is hardly more difficult than drawing regular polygons. Increasing the side length may be problematic, but this can be handled on more than one levels. Let us consider an example:

First step: preparation

Level 1 (sequence):

The commands are the following:

```
forward 10 right 90 forward 20 right 90
forward 30 right 90 forward 40 right 90
```

etc.

The complete drawing is presented below:

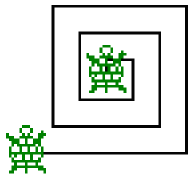

Figure 6. The complete drawing and the positions of the turtle before and after drawing

The first level of concept-formation is similar to that of drawing polygons, however, on the second level a special problem arises. In order to produce figures of varying sizes with repetition one has to handle size variation within repetition itself. To tackle this problem such a parameter is needed that can receive a new value at each step of repetition. The problem in question can also be treated in Logo using a global parameter. However, on the one hand this solution does not fit Logo, which is an automaton based programming language; and on the other hand this solution is also quite difficult from a conceptual point of view, therefore its use is not recommended methodologically in grades 3–6. Nevertheless, construction of the figure may be linked to Level 4 in producing Figure 1. Namely, spirals can also be drawn with recalling, similarly to the case of squares. However, in the case of spirals the procedure is not only called, the value of the parameter is increased on each recall.

Second step: the notion of recursion

Level 4 (recursion):

```
to spiral_5 :s_length
    forward :s_length right 90
    spiral_5 :s_length + 10
end
```

The experience gained is the following: recursion is also infinite here, just like in the case of procedures written on the first example's fourth level, however, infiniteness is quite disturbing here: after a time the figure is not discernible. The reason for this is that because of drawing line segments of increasing length on one another after a time the whole screen is the same colour as the colour of the pen.

Adding the comments marked by semicolons to the procedure—this is the usual way to mark comments in Comenius Logo—; infinity can be demonstrated well, hence we get closer to interpreting the execution of the algorithm.

The procedure is called by its name and the beginning value of the parameter in question; in this case this is '`spiral_5 10`'.

It is worth experimenting further with the procedures mentioned above. Interesting figures can be drawn if not side length, but angle of turning is parameterized and this may be tested with more than one settings. Instead of the traditional spiral more difficult figures are being drawn in such cases, figures which are usually called `inda` 'samentum' or `rózsa` 'rose' in the literature. The external character of the figures also varies depending on the beginning value assigned to the parameter '`:angle`' on the first call and on the measure of its increase on the recall. For further details I refer the reader to Farkas (1994). [3]

b. The procedure using a parameter when angle of turning is parameterized is the following:

```
to spiral_6 :angle
    forward 10 right :angle
    spiral_6 :angle + 10
end
```

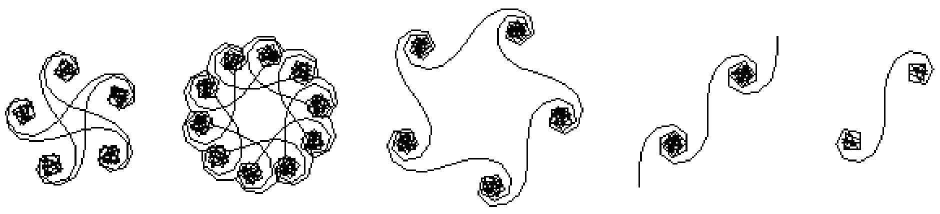

Figure 7. Figures obtained by applying procedure `spiral_6`

Third step: the notion of regular recursion, components of "good" recursion

Level 5 (regular recursion):

The problem arising on the fourth level can be solved in a way which is quite similar to the one presented in the first example. However, there are now two possibilities of regulating recursion:

- One solution—which is more "natural" and therefore pupils can more easily grasp it—is that the condition of resumption is determined as a function of side length.
- The other solution concerns introducing a new parameter which represents levels of recursion. (Cf. the outline of the fifth level in the first example.)

  a. The procedure with regular recursion—halting depends on side length:

  ```
  to spiral_7 :s_length
      if :s_length < 120 [forward :s_length right 90 ~
                          spiral_7 :s_length + 10]
  end
  ```

  The number of the spiral's line segments depends on the beginning value of the parameter designating side length.

  b. The procedure with regular recursion—levels of recursion is also parameterized:

  ```
  to spiral_8 :s_length :level
      if :level > 0 [forward :s_length right 90 ~
                     spiral_8 :s_length + 10 :level - 1]
  end
  ```

  The number of the spiral's line segments depends on the value of the parameter '`:level`'.

Drawing similar figures also lends itself well to introducing the notion of recursion. The first three steps can be realized in a way which is quite similar to the ones of drawing spirals. However, this task, when compared to the previous ones, provides much more opportunities for experimenting and for grasping the working mechanism of recursive algorithms in a more profound manner. The opportunities in question are presented below.

Example 3: similar figures

First and second steps: cf. the examples above

Third step: a solution applying regular recursion

```
to squares_1 :s_length :level
    if :level > 0 [repeat 4 [forward :s_length right 90]˜
                   squares_1 :s_length+10 :level-1]
end
```

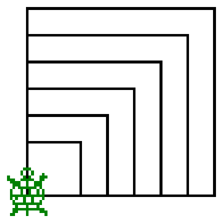

Figure 8. Squares; the figure can be drawn with or without recursion

Fourth step: drawing complex recursive figures

Level 6 (experiments):

The external character of the figure may be different depending on the location of the recall within the procedure. Now the recursive call is located within a repetition; hence, this is not a case of simple recursion. It is recommended to observe the process of drawing with the help of step by step execution.

At this point of the discussion various problems can be raised. Consider the following:

What happens if . . .

- the procedure is recalled while drawing the square, before turning right?

  ```
  to squares_2.a :s_length :level
      if :level > 0 [repeat 4 [forward :s_length ˜
                squares_2.a :s_length :level-1 right 90]]
  end
  ```

- the procedure is recalled while drawing the square, before turning right, and bisecting side length?

  ```
  to squares_2.b :s_length :level
      if :level > 0 [repeat 4 [forward :s_length ˜
              squares_2.b :s_length / 2 :level - 1 right 90]]
  end
  ```

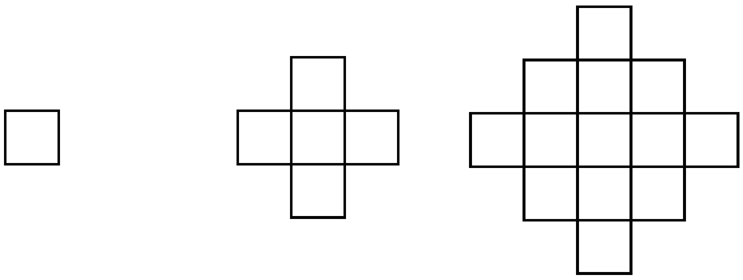
(a) `squares_2.a 20 1` (b) `squares_2.a 20 2` (c) `squares_2.a 20 3`

Figure 9

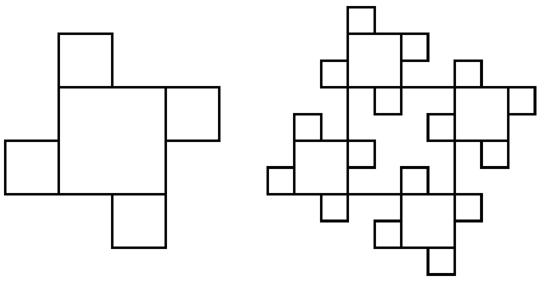
(a) `squares_2.b 40 1` (b) `squares_2.b 40 2` (c) `squares_2.b 40 3`

Figure 10

- the procedure is recalled while drawing the square, after turning right, and bisecting side length?

```
to squares_3.a :s_length :level
    if :level > 0 [repeat 4 [forward :s_length right 90~
                   squares_3.a :s_length / 2 :level - 1]]
end
```

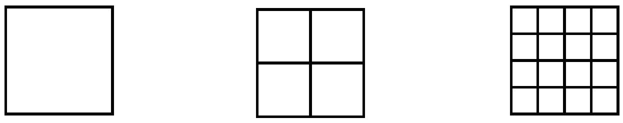
(a) `squares_3.a 40 1` (b) `squares_3.a 40 2` (c) `squares_3.a 40 3`

Figure 11

- the procedure is recalled while drawing the square, after turning right and dividing side length into three parts?

```
to squares_3.b :s_length :level
    if :level > 0 [repeat 4 [forward :s_length right 90~
                to squares_3.b :s_length / 3 :level - 1]]
end
```

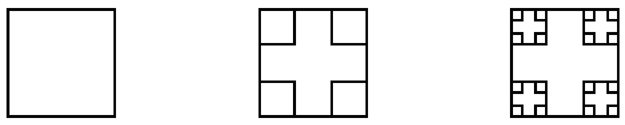
(a) `squares_3.b 40 1` (b) `squares_3.b 40 2` (c) `squares_3.b 40 3`

Figure 12

- the procedure is recalled while drawing the side of the square?

```
to squares_4.a :s_length :level
    if :level > 0 [repeat 4 [forward :s_length/3 ~
                   squares_4.a :s_length/3 :level-1 ~
                   forward :s_length/3 * 2 right 90]]
end
```

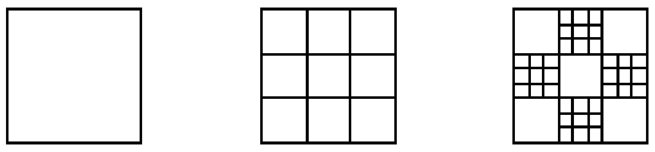
(a) `squares_4.a 50 1` (b) `squares_4.a 50 2` (c) `squares_4.a 50 3`

Figure 13

- the procedure is recalled while drawing the side of the square, but before the recall we turn away?

```
to squares_4.b :s_length :level
    if :level > 0 ~
        [repeat 4 [forward :s_length/2 left 135 ~
               squares_4.b :s_length/3 :level-1 right 135 ~
               forward :s_length/2  right 90]]
end
```

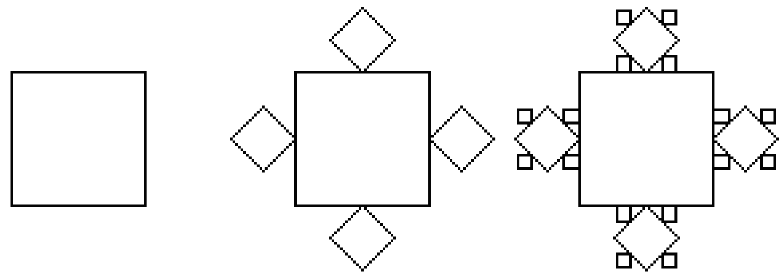
(a) `squares_4.b 50 1` (b) `squares_4.b 50 2` (c) `squares_4.b 50 3`

Figure 14

And so on, the number of possibilities is almost infinite.

Level 7 (applying experiences gained from experimenting)

On this level many types of tasks can be used in order to deepen the acquired knowledge and to apply it in new situations.

Complete figures may be shown to pupils, and their task is to draw these while applying the observed regularities. For instance,

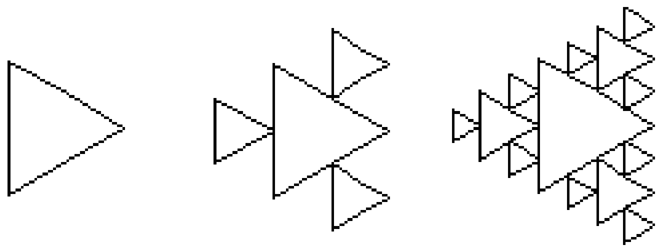
(a) `triangle 50 1` (b) `triangle 50 2` (c) `triangle 50 3`

Figure 15

Tasks which serve to facilitate understanding the role of parameters and that of commands and the working mechanism of recalls can also be devised. For example,

(1) Specify the value of '`ANGLE`' in the procedure below in order to get the following figure:

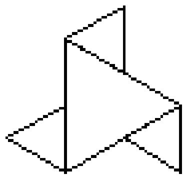

Figure 16. `triangles 50 2`

```
to triangles :length :level
    if :level > 0 [repeat 3 [forward :length right ANGLE1 ~
        triangles :length / 2 :level - 1 right ANGLE2]]
end
```

(ANGLE1=60, ANGLE2=60)

(2) How many circles are drawn if the call '`circles 1 3`' is used?

```
to circles :length :level
    if :level > 0 [repeat 8 [circles :length/2 :level - 1~
                   repeat 45 [forward :length j 1]]]
end
```

($8 \cdot 8 + 8 + 1 = 73$ circles are drawn)

(3) Specify the position of the recursive recall (A, B, C or D) in the procedure below in order to get the following figure:

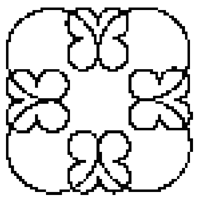

Figure 17. Flower

```
to flower :length :level
    if :level > 0[repeat 4[A repeat 270[forward :length
                  right 1 B] C right 180 D]]
end
```

(C)

Summary

Recursion is perhaps one of those phenomena appearing in nature that are the most difficult to understand; its realization is one of the most complicated programming tasks. However, the notion can be made suggestive and it can be related to concrete activities, hence, it can be taught even in primary schools. It is important to note that “teaching” is not used here in its traditional sense, meaning conveying a given material; it rather means pupils’ active participation in the process of acquiring knowledge. The teacher’s task is to organize this process, to monitor it and to provide the necessary circumstances for individual work.

This study has analysed various realization possibilities of the notion of recursion applying the tools of the Logo turtle-graphics. One methodological solution has been presented out of the two fundamental ones that are available. This solution has many abstraction levels and the presentation also exemplified the principle of gradual building through consecutive steps. The other methodological solution is going to be presented in the forthcoming part of this study.

References

[1] Angster Erzsébet, *Programozás tankönyv I–II. [Programming I–II.]*, 4KÖR Bt., 1999.

[2] Bartha Judit, Három „kínai” útvesztő [Three “Chinese” Labyrinth], *Jelenkor* **10** (2003), 1023–1027.

[3] Farkas Károly, The World of Turtle Roses, *Acta Didactica Universitatis Comeniae Informatics*, Issue, Comenius University Slovakia, Bratislava (1994), 63–67.

[4] Farkas Károly, *How had Logo effected to a pedagogue?*, 2005, `http://eurologo2005.oeiizk.waw.pl/PDF/E2005Farkas.pdf`, last download: 2008. 06. 01.

[5] Heizlerné Bakonyi Viktória, Zsakó László, *Logo versenyfeladatok tára [A collection of Logo competition tasks]*, NJSZT, Budapest, 2003, 2008.

[6] A Logo országos számítástechnikai tanulmányi verseny feladatsorai (1998–2008) [Task sheets of the National Logo Programming Competition (1998–2008)], `http://logo.inf.elte.hu/`, last download: 2008. 06. 01.

[7] Járdán Tamás, Pomaházi Gábor, *Adatszerkezetek és algoritmusok [Data structures and algorithms]*, KTF Líceum Kiadó, Eger, 1998.

[8] Kisgyermekek nagy mesekönyve [The Great Storybook for Little Children], Móra, Budapest, 1975.

[9] C. H. A. Koster, *Programozás felülnézetben [Programming from above]*, Műszaki Könyvkiadó, Budapest 186, 1988.

[10] Kőrösné Mikis Márta, Itt a magyar Comenius Logo! [Here comes the Hungarian Comenius Logo], *Iskolakultúra 1997* **6–7** (1997), 118–120.

[11] Mészáros Tamásné, *Logo-világ [Logo-world]*, Nemzeti Tankönyvkiadó, Budapest, 1997.

[12] A Kormány 202/2007. (VII. 31.) rendelete [Governmental regulation 202/2007. (VII. 31.)] (Nemzeti alaptanterv 2007. [National Core Curriculum 2007.]), Oktatási és Kulturális Minisztérium, 2007,
`http://www.okm.gov.hu/letolt/kozokt/nat_070926.pdf`, last download: 2008. 06. 01.

[13] Seymour Papert, *Mindstorms—Children, Computers and Powerful Ideas*, BASIC BOOKS, Inc., HARPER COLOPHON BOOKS, 1981.

[14] Seymour Papert, *Észrengés – A gyermeki gondolkodás titkos útjai*, SZÁMALK, Budapest, 1988, 61–63, Hungarian translation of the book above.

[15] Rozgonyi-Borus Ferenc, *Informatika [Informatics]. Kerettanterv-rendszer az általános iskolák számára [A system of core curriculum for primary education]*, Mozaik Kiadó, Szeged, 2004.

[16] Rozgonyi-Borus Ferenc, *Imagine I–II–III*, Abax Kiadó, Szeged, 2007.

[17] Robert J. Sternberg, Talia Ben-Zeev, *The Nature of Mathematical Thinking*, Lawrence Erlbaum Associates, Publishers, Mahvah, NJ, 1996.

[18] Robert J. Sternberg, Talia Ben-Zeev, *A matematikai gondolkodás természete*, Vince Kiadó Kft., Budapest, 1998, Hungarian translation of the book above.

[19] Szlávi Péter, Zsakó László, *Módszeres programozás: Rekurzió [Systematic programming: recursion]*, Mikrológia sorozat, ELTE Informatikai Tanszékcsoport, 1991–1997.

[20] Szlávi Péter, Zsakó László, Programozás tanítási módszerek [Programming teaching methods], in: *Informatika a felsőoktatásban 2002 Konferencia tanulmánykötete*, 2002, 1006–1013.

[21] Tótfalusi István, *Idegenszó-tár [Dictionary of Foreignisms]*, Tinta Kiadó, Budapest, 2005.

[22] Turcsányiné Szabó Márta, Zsakó László, *Comenius Logo gyakorlatok [Comenius Logo exercises]*, Kossuth Kiadó, Budapest 4, 1997, 42–45.

[23] Turcsányiné Szabó Márta, *LÓGÓ-s tanulás [Learning Logo]*, 1994,
`http://comlogo.web.elte.hu/publikaciok/hun94.html`, last download: 2008. 07. 24.

CECÍLIA SITKUNÉ GÖRÖMBEI
NYÍREGYHÁZI FŐISKOLA
H–4400 NYÍREGYHÁZA
SÓSTÓI U. 31/B.
HUNGARY

E-mail: `sitkune@zeus.nyf.hu`

(Received April, 2008)