

Logo in Lifelong Learning

Janka Pekárová, *pekarova@fmph.uniba.sk*

Dept of Informatics Education, Comenius University, Bratislava, Slovakia

Andrea Hrušecká, *hrusecka@fmph.uniba.sk*

Dept of Informatics Education, Comenius University, Bratislava, Slovakia

Abstract

In our paper we introduce curricula of on-line courses of programming in Imagine Logo environment. Courses were meant mainly for IT science teachers with small or no experience in programming.

We characterize successful participation in the course by following aims: Programmer

- defines and uses own commands for the turtle; recognizes situations where using parameter in command is appropriate,
- creates more turtles; changes shape of the turtle to the image or describes the shape by simple drawing-list,
- changes the behaviour of the turtles; turtles react on events or changing properties of their surroundings,
- controls turtles by launching processes,
- uses buttons, sliders and multimedia objects for enriching own projects.

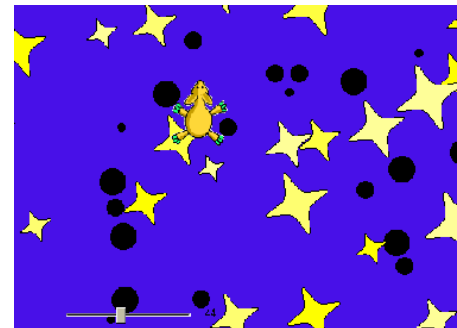


Figure 1 Milky Way project

We developed prototype of the project which contains all aims mentioned above – *Milky Way project*.

We describe ways that we have chosen to fulfil educational aims. We discuss several examples that we used for constructing basic programmer skills in Imagine – turtle geometry, repetition of commands, definition of own commands, reactions on events and processes for controlling turtle's movement. We tried to develop skill to read and understand unknown code, to analyse solution and to find principles on which solution is based. We consider these skills as essential for teaching of programming.

Latter we search for propaedeutic of advanced techniques of programming in Imagine Logo in our courses, especially recursion and object-oriented approach as described in Blaho and Kalaš (2002). We examine final projects of several participants and design alternatives for program. We explore knowledge authors of the programs missed that could help them to reduce their solution.

We deduce a few recommendations for teachers' training in Logo programming from our experiences.

Keywords

Imagine Logo, life-long learning, programming, analyse, object-oriented approach

Course background

We worked as designers and tutors of online courses in Logo programming in last two years. Courses were meant for the IT science teachers which had had small or no programming skills before. However, they used computers daily and acted as flexible learners in discovering new ways how to use computer – using computer to **design** own projects, to **share ideas** about own projects and to realize projects – **create** small programs in Imagine Logo programming environment. We used distance learning form – participants of the course used learning management system to access study material, tutors or other course members. This form of study fitted many employed teachers - they could study when they wanted and where they wanted.

Course content refers to Slovak version of Imagine Logo Primary Workbook by Blaho and Kalaš (2004), the textbook for pupils and extends it into introduction to advanced programmer's skills (e.g. recursion, object – oriented programming).

Units Order	Content of Unit	Level
1 - 6	<ul style="list-style-type: none"> ▪ Basic turtle commands ▪ Cycle, repetition of several commands ▪ [Polygon] ▪ Definition of own commands ▪ Using own commands to create new ones ▪ Commands with one parameter ▪ Reactions on click, on drag ▪ Creating new turtle using toolbar ▪ Addressing concrete turtle 	Beginner Imagine Logo Primary Book
7	Revision	
8 - 10	<ul style="list-style-type: none"> ▪ Image as shape of the turtle ▪ Frames, animated shape ▪ Launching process by every command ▪ [Naming and cancelling process] ▪ Simple conditions, testing colour of the page 	Beginner Imagine Logo Primary Book
11	Revision	
12	Creating turtle by command	Advanced
13	Drawing-lists as shape of the turtle	
14	Private variables of the turtle	
15	Revision	

Our role as tutors and designers in the course consisted of these responsibilities:

- defining educational aims of the course,
- design and implementation of study material to cover defined aims,
- checking progress of the participants,
- ensuring the quality of study material and quality of on-line study,
- development of learning and profiting community.

Educational aims of the course

We characterize successful participation in the course by following aims: Programmer

- defines and uses own commands for the turtle; recognizes situations where commands with one parameter are appropriate to use,
- creates more turtles; changes shape of the turtle to image or describes the shape by simple drawing-list,
- changes the behaviour of the turtles; turtles react on events or various properties of their surroundings,
- controls turtles by defining processes,
- uses buttons, sliders and multimedia objects for enriching own projects.

We merged defined skills into prototype of the final project – project *Milky Way*. Main hero of the game is yellow lamb – space pilgrim who is picking up the stars. Dangerous Black Holes are appearing from time to time in the sky – the lamb can lose in them.

We can find any of educational aims implemented in the project.

Programmer defines own command for creating Black Holes of various sizes.

Execution of the command is controlled by process.

Stars are turtles with their shape described by drawing-list. User can set their amount by slider at the very beginning of the game.

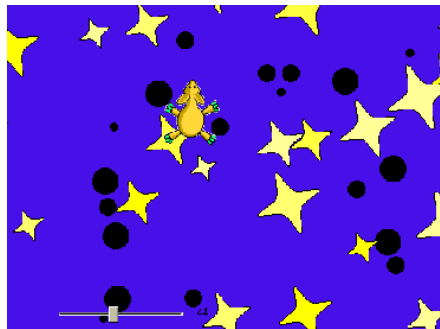


Figure 1 Milky Way project

Space pilgrim is the turtle with image used as its shape. Motion process controls figure's movement. User can change direction of the pilgrim by clicking it or by using direction buttons. When pilgrim meets the star, star will disappear (object will be destroyed). When pilgrim enters black point (piece of Black Hole), he will be destroyed and game will end (all user-defined processes stop).

In following part we describe ways we used to form programmer skills during on-line course of programming in Imagine Logo. They contain not only programs, but also stimuli for discussion among learners.

First ten units of the course cover and deepen content of Imagine Logo Primary Workbook. Teacher develops skills similar to skills pupils will have if the successfully work with the book. He becomes programmer – beginner in Imagine Logo. What do we offer him in our course?

Beginner level of programming

In the first half of the course we concentrate on development of skill to create and understand Imagine programs. Main activities include defining own commands, joining them to more complicated ones and design of first small games where turtle is controlled by buttons. Ability to

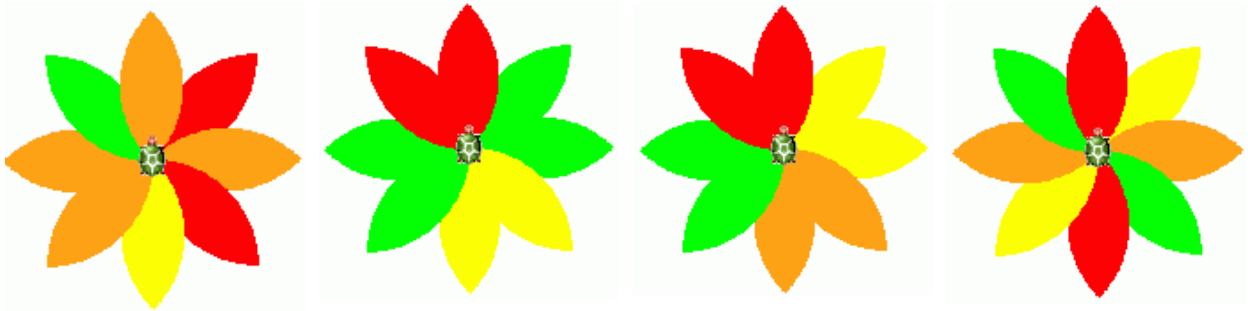
compose program from small pieces belongs to good practice among programmers and it is essential to build it from the first experiences with programming.

Teacher of programming shall dispense not only by knowledge of commands and several programming techniques, but also by perfectly-developed ability to analyse solution of pupil and recognize errors. We mean not only syntactic errors, but mainly errors in program logic.

We designed many test-like tasks where teachers could prove their comprehension:

- some tasks were concentrated on **skill to understand program**,

Which programs generate following drawings? Assign program to drawing it produces.



```
Repeat 2 [
  setPC "yellow
  petal right 45
  setPC "orange
  petal right 45
  setPC "green
  petal right 45
  setPC "red
  petal right 45
]
```

```
repeat 4 [
  setPC ?
  repeat 2 [
    petal
    right 45
  ]
]
```

```
repeat 8 [
  setPC ?
  petal
  right 45
]
```

```
setPC "yellow
repeat 2 [petal
right 45]
setPC "orange
repeat 2 [petal
right 45]
setPC "green
repeat 2 [petal
right 45]
setPC "red
repeat 2 [petal
right 45]
```

Teacher shall also analyse and accomplish simple programs:

We defined command `bedflower :B` – turtle will draw `:B` colourful dots. Number of dots is given by `:B`. Distance between two following dots is 40 points.

Complete the command `garden` – turtle will draw 4 bedflowers of various colours. Distance between bedflowers is 40 points. Turtle will return to start position after drawing all bedflowers.

```
to garden
  cs
  pu right 90
  repeat ...[
    setPC any
    bedflower ...
    back ...
    left ...
    forward ...
    right 90
  ]
  right 90 forward 4 * 40 right 180
end
```



Figure 2 Bedflowers

We also designed fill-in type of task in chapter *More turtles, more opportunities*. Teachers should fill in names of turtles according picture:

```
askEach [ _ _ ] [setPC "green]
askEach [ _ _ ] [
  setPC _ left 45
  repeat 4 [fd 50 bk 50 right 90]
]
ask _ [
  point 100 setPC "white point 70
]
ask _ [
  fd 50 right 90
  repeat 180 [fd 1 right 1]
  right 90
]
```



Figure 3 Addressing turtles

We wanted also to profit from specific character of children programming languages: from openness, visual attractiveness, clearness and talkativeness. Therefore we started to develop small games as soon as possible - after managing basic turtle's commands. We supposed that developing games would fasten teachers' motivation – they could create useful and funny products for children.

Easter – egg project is a kind of painting book where children choose one of variety of patterns, drag it and stamp it on blank Easter – egg following own fantasy. Teachers studying Logo programming should examine functionality of the project and suggest improvements/changes so that project better serves children painters' intentions.

Easter – egg project represents special kind of task – **explore** unknown project, notice scientifically its goals and functionality, then **express** opinion – design changes and finally **exchange** ideas with other members of learning community – three Xs which Harel (2003) finds necessary for children to thrive. In *Easter – egg activity* we find idea of three Xs applied into the world of adults. Fisher (2003) mentions: "When other learners are engaged in answering questions, learners explore and examine concepts more, **think** more and **share** more".

What do beginning programmers learn in this kind of activity? They synthesise pieces of own knowledge, judge the program and try to find solutions of problems observed.

Participants of the course designed valuable modifications of the project:

- adding new button to the project. User can set different colour of blank Easter-egg by pushing the button;
- editing patterns so that user can drag whole patterns also by holding transparent part of patterns;
- patterns should be stamped only on the egg, not in its surroundings.

Inclusion of third suggestion into project leads to the need to distinguish colours – colour of the egg and its surroundings. Project can serve as introduction to simple conditions – situations in which turtle changes its behaviour according to colour of the page.

That's why we re-used and extended *Easter - egg activity* in chapter focused on "deciding" – turtle's reactions to various situations and designed another small project – project *Design Your Clown*. Idea of the project is the same – user can drag and stamp some pattern (eyes, nose, butterfly) to the clown's face and clothes. User can stamp part of face (eyes, mouth, nose) only to beige face of clown. If he tries to stamp it elsewhere in place with different colour, pattern will automatically return to green stamp bar – to home position.



Figure 4 Design Your Clown project

Introduction to advanced programming skills

Object – oriented Approach

Blaho and Kalaš (2002) designed several steps to develop understanding of object oriented metaphor:

1. Elementary Logo for Single Turtle.
2. Elementary Logo Activities for Multiple Turtles.
3. Multiple Turtles with Private Information.
4. Clones of Objects.
5. Instances of Instances.
6. Hierarchy of Several Layers.
7. User-defined Classes.

We implemented first four steps of mentioned approach in the whole curricula of the course and designed several projects with the goal to support using of object-orientedness of Imagine Logo.

[Elementary Logo for Single Turtle] *My own boardgame.*

Using one turtle, programmer can draw plan of boardgame. Extending command for drawing the plan by length parameter causes easy scalability of the plan.

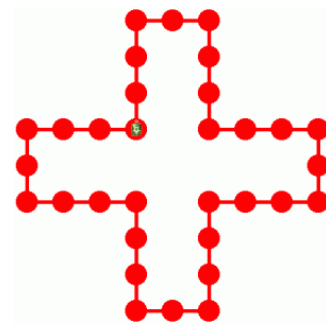


Figure 5 Plan of the boardgame

[Elementary Logo Activities for Multiple Turtles] *Living Picture* project. Programmer creates three turtles named **settler**, **kelpie** and **forester** by direct manipulation (using New turtle button from the tool-pane). Each turtle reacts on click in special manner – forester plants (i.e. stamps) trees, kelpie stamps fish or frogs and settler creates stones and people.



Figure 6 Living Picture project

[Clones of Objects] Project *Typewriter*. Programmer creates one turtle as prototype – letter. He sets automatic dragging to it, sets its pen up and changes turtle's behavior – stamping and returning to home position. Then he creates several copies of the prototype – clones with same behavior. After changing frames of clones typewriter is ready for use.



Figure 7 Typewriter project

[Multiple Turtles with Private Information] Project *My Tones*. After loading the project several notes discover on the page. Each note stands for one tone. When clicking the note, note is enlarged and its tone is played.

Each note represents member of the special family of the turtles. The family is characterized by special note-like shape of turtles and reaction on click – resizing the shape.

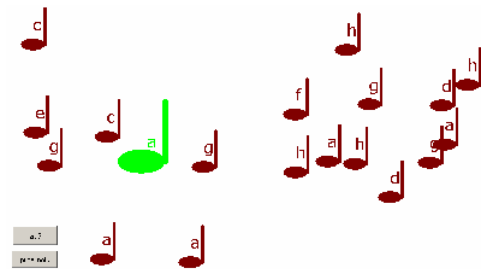


Figure 8 My Tones project

Concept of private information which turtle stores should be quite familiar also to beginners. As soon as they work with more turtles, they learn that each turtle has its own settings – pen, position, shape. We can deepen awareness of private information by using them in extraordinary way. Project *Frogs* illustrates this idea:

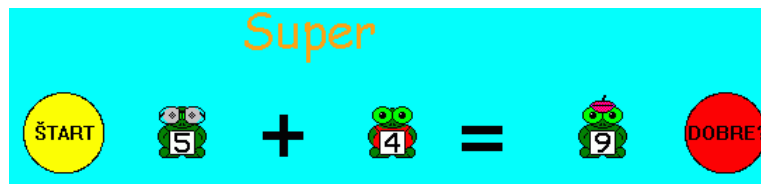


Figure 9 Frogs project

Project contains three frogs with random numbers on their T-shirts. User changes number of first and second frog by clicking it. His task is to achieve number of the third frog as the sum of the first two frogs' numbers.

How to find the sum? Clue lies in using number of frame of frog shape. Image of the frog consists of nine frames – each frame contains frog with different number. Frame number equals number on T-shirt of the frog. We can therefore provide checking sum very simply:

```
ifelse frog1'frame + frog2'frame = frog3'frame [label "Super"][label "Nooo.]
```

Project *Frogs* inspired math and IT science teacher to extension to more analytic type of task:



Figure 10 Extension of Frogs project

User shall click on each flower which is equal to the sum of any two numbers of the frogs in the pond. In the pond there are four frogs living. When user clicks incorrect flower, flower gets red, otherwise it disappears.

Teacher thought out simple solution – she enumerated all combinations of frames of frogs when clicking the flower. We find this solution appropriate to teacher's knowledge of programming and relatively small number of combinations. If she wanted to find more clever solution, she would possibly work with list where all possible combinations would be stored. This list shall generate when creating frogs. Other solution is going through list of frogs every time when any flower is clicked and sequential check of the sum (we would at first check all possible sums containing frog number 1 and another frog, then frog number 2 and another frog etc.)

Although participants of the course didn't learn anything about classes and instances of instances, we tried to prepare them for future understanding of these concepts. In project *Sort Numbers* child differs between two types of numbers: odd and even. Task of child is to put all even numbers into special box. If he tries to put odd number to the box, number jumped back.

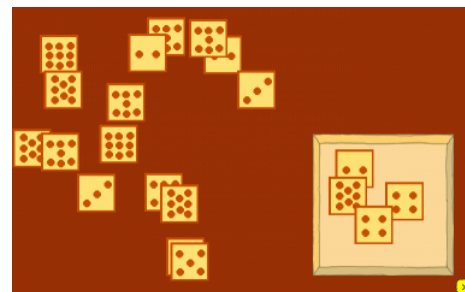


Figure 11 Sort Numbers project

In programmer's point of view: we create two “families” of objects, group of odd numbers which react on dragging by obstinate return to home position and group of even numbers which distinguish whether they are in the box or not (if not, they jump to home position).

Project can be re-programmed to object-oriented approach easily – families will be instances of two types of object – one object represents odd number and the other even number.

Recursion

Recursion comprises very powerful tool in Logo world, for instance

- combination of recursion and turtle geometry can produce magnificent effect because programmer can easily describe complicated shapes as fractals (as Foltynowicz and Walat (2005) show).
- recursion can be used to process some information, as method of special data structure – we can describe family of turtle having special properties and settings in lists and then compile the information by simple recursion:

```
make "myfamily [[Stan [0 180] 90 "taupe] [Hilda [0 90] 180 "green] [Clare [0
0] 270 "orange]]

to createNewFamily :family
  if empty? :family [stopme]
  new "turtle ![
    name (first first :family)
    pos (item 2 first :family)
    pencolour (last first :family)
    heading (item 3 first :family)
    shape [fd 50 point 30]
  ]
  createNewFamily butFirst :family
end
```

We – tutors discussed inclusion of recursion into course content. In our opinion deep understanding of recursion as well as design of own data structures for projects goes beyond beginning programmer's level. For this reason as well as limited number of units and our decision to cover the content of Imagine Logo Primary Book we haven't created unit engaged specially to recursion. Another reason is that solution of the problem by simple form of recursion -tail recursion can be easily replaced by processes in Imagine Logo.

However, we used several examples which can be used propaedeutically in understanding to recursion. These examples aim at simple form of recursion – tail recursion. Although concept of recursion is not mentioned in the assignments, discussed solution contains anticipation of developing this concept latter. Popular examples of this kind, also used in distance course are so called “carpet patterns” which Hrušecká, Kalaš (2006) introduces:

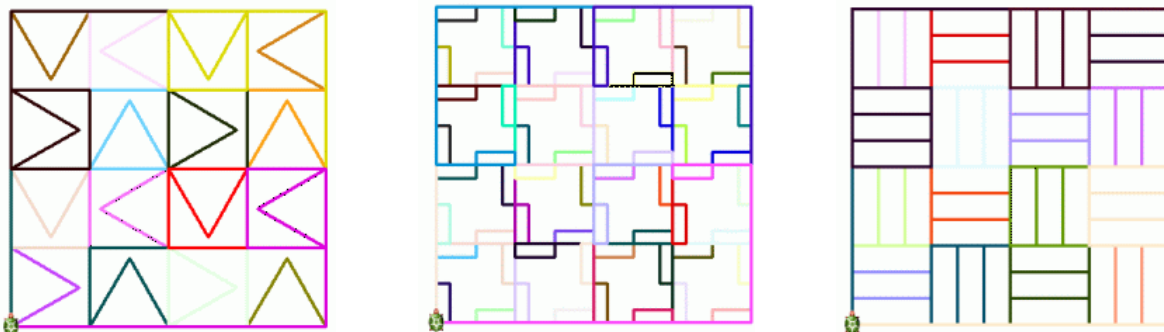


Figure 12 Carpet patterns

One pattern creates basis of each carpet in the picture. Pattern is repeated 4-times and forms small square. Turtle moves and forms greater square by drawing four small squares.

Pattern of first carpet consists of square and triangle of random colour:

```
to myPattern
  setPc any
  square 60
  triangle 60
end
```

Four patterns form small square...

```
to smallSquare
  repeat 4 [
    myPattern
  ]
  fd 120
```

...and four small squares form bigger one.

```
to bigSquare
  repeat 4 [
    smallSquare
  ]
  fd 240
```

```

      rt 90
    ]
end
      rt 90
    ]
end

```

Beginner can notice application of same rule in drawing big square as in small square. The only difference is the distance the turtle moves after drawing the pattern – it's always size of square multiplied by two. Definition of giant square consisting of 4 big squares should be very simple.

How to describe our solution without need to define new and new commands which differ only by distance in forward command? We can discuss re-using same procedure with different parameter and in this way discover tail-recursion as simplest form of recursive calling.

Participants of the course evaluated carpet patterns in different ways:

"I admit these patterns shocked me a bit at first but I managed all three patterns according to solved example."

"I consider carpet patterns task most difficult of all. I don't like drawing such complicated shapes." Same student adds: "I liked all tasks in unit". Difficulty doesn't obviously eliminate attractiveness of the task. We believe that course participants will naturally discover power of recursion in future.

Becoming Game-maker

Milky Way project became base of *Ecology Project* – final project of K5 teacher of nature science and technical training. Her game represents food chain – stork catches mice which are fed by grain – some of grain is poisoned by pesticides and some not. Stork – disguised space pilgrim - doesn't know which mouse will die because of eating poisoned grain. Its task is to pick up all mice. Mice are moving all the time and pick grain; player cannot affect their choice of grain. That's why player never knows result of the game – whether stork survives or not.

Teacher tried to design all figures of the game – grain, mice and stork – as turtles. She soon discovered serious obstacle of her access – when stork tests overlapping with other object, programmer needs to distinguish whether it is grain or mouse. In this point she needs some data structure to remember type of object. One solution hides in user – defined classes *Grain* and *Mouse*. When using object-oriented access, testing of overlapping is quite simple:

```

if stork'overlap? alof "mouse [
  ; stork remembers count of eaten mice
  stork'setmouse mouse + 1
  eraseObject alof stork'overlapList alof "mouse
]

```

However, teacher as beginning programmer could not define own classes and found recognition of types as difficult problem with her knowledge. Another solution of the problem is storing names of mice and grain objects in lists as basic data structure of Logo. Work with lists as abstract data types in contrast with direct manipulation with turtles exceeds beginner level of programming. So does applying a procedure or procedural object – programmer can store information about type of object in private variable and use standard Imagine Logo `map` procedure to distinguish between objects.



Figure 13 Ecology Project

These reasons led us to combination of turtle's drawing and objects of one type displayed on page. Teacher applied our access successfully in *Ecology Project* – red poisoned and healthy yellow grain are coloured points and stork can safely test overlapping with all turtles – the other turtles living on page are all mice.

Design of *Milky Way* project illustrates limits of course content. Teachers use “families” of turtles with similar properties and behaviour, but don't know effective way how to recognize between different families. Their own game-making and practice in Imagine programming will perhaps bring the need for advanced object – oriented level of programming in future.

Conclusion

Which skills and abilities determine advanced level of programming? Where do bounds between beginning and advanced programmer lie? We try to find partial answer to this question:

- all programmers should understand at least own code of program,
- all programmers should compose their program from smaller part,
- design of data structure brings new possibilities for extending own program. Still, with no knowledge of manipulation with data structures we can design attractive and constructivist microworlds in Logo programming environment,
- object-oriented approach seems as natural solution to variety of problems. However, deep understanding of this approach requires some time. We evaluated one of numerous ways how to start with the concept of object.

Mastery of teacher does not mean to show all we know, but also in secret: “I know special tricks how to solve your problem quickly and more effectively...but would my student understand it just now?” Understanding of the children isn't given all the time; learning is process of developing understanding.

That's why we would recommend to split course content into two levels. Last five units can serve as starting point for building advanced programmer skills in Imagine Logo world.

References

- Blaho A., Kalaš I. (2002) *Object Metaphor Helps Create Simple Logo Projects*. In Proceedings of EuroLogo 2001, A Turtle Odyssey. Linz, August 2001. pp. 55 – 65.
- Blaho A., Kalaš I. (2004) *Imagine Logo Primary Workbook*. Logotron, Cambridge
- Foltynowicz I., Walat A. (2005) *Fractal Variations*. In Proceedings of EuroLogo 2005. Warsaw, August 2005. pp. 33 – 43.
- Fisher M. (2003) *Designing Courses and Teaching on the Web*. Scarecrow Education, Oxford.
- Harel I. (2003): *Learning Skills for the Millennium*. The Three Xs. Online http://www.mamamedia.com/areas/grownups/new/21_learning/three_xs.html
- Kalaš I., Hrušecká A. (2006): *Programming in Imagine environment* (in Slovak), Metodicko-pedagogické centrum, Bratislava.