ISSN 2676-9425

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2

Number 1

2020

Eötvös Loránd University
Faculty of Informatics
Budapest, Hungary

Contents

The Canvas LMS Systems' Advanced Possibilities in Course Design and the Usage of Evaluation in Formal, Non-formal and Informal Education

ABONYI-TÓTH Andor

Abstract. From the 2016's academic year, the Eötvös Loránd University, in addition to the Moodle framework, also enabled teachers to use the OpenSource version of the Canvas LMS, not only for courses related to formal education but for course-based or even open courses (MOOC). An essential aspect of choosing the framework was to have built-in features which support advanced approaches to the course management and evaluation (e.g., teamwork, peer review, advanced module organization, student differentiation, collaborative opportunities, outcome-based education, rubrics). In my article, besides presenting the usage of the mentioned functions in education, I also summarize the statistics of the usage of this system as well as the teacher's feedback.

Keywords: Canvas LMS, differentiation, learning management, grading schemes.

1. Introduction

ELTE is Hungary's most prestigious university with the richest traditions and the highest international rankings in the country, where tradition and innovation go hand-in-hand[1]. The field of e-learning also has an important role in our institution, we support our courses with different learning management systems (LMS). From the 2016/17's academic year (besides the previously started Moodle system) we provide the usage of the Canvas system to our instructors and students.

With the introduction of the Canvas system, we intended to support not only just formal but also the informal education as well. Because of technical reasons we choose to run two versions of Canvas, the first one (canvas.elte.hu) is to support the formal education, while the second one (mooc.elte.hu) had been created for open courses, further teacher training, etc.

The interaction with the Neptun Education system which is used in the Hungarian higher educational institutions was solved in January 2017, allowing the instructors to initiate the creation of e-learning courses from the education system. Instructors and students are automatically given access to the courses that have been created, they can enter into the system with the Neptun ID which they are already familiar with. In the Canvas system that is used in formal education, statistics are available for each semester (see Table 1.)

Semester	Number of courses	Number of instructors	Number of students
2016/17/2	212	104	3686
2017/18/1	276	124	4642
2017/18/2	301	168	5106
2018/19/1	463	281	8449
2018/19/2	606	375	9766

Table 1: Statistical data on the Canvas environment for formal education

[1] ELTE Brochure: https://www.elte.hu/file/ELTEkiadvany2015.pdf

The popularity of the Canvas framework is visibly constantly increasing, and now colleagues are launching three times more courses than three years ago. Table 2 shows how they started courses are distributing between the Canvas and Moodle systems.

Semester	Number of all the courses	Number of Canvas courses	Number of Moodle courses
2016/17/2	1171	212 (18%)	959 (82%)
2017/18/1	1338	276 (21%)	1062 (79%)
2017/18/2	1287	301 (23%)	986 (77%)
2018/19/1	1840	463 (25%)	1377(75%)
2018/19/2	1880	606 (32%)	1274 (68%)

Table 2**:** Distribution of all formal courses between the two frameworks

One of the reasons for Moodle's popularity that it was introduced at our university before Canvas, so most of the instructors were reluctant for switching to a new framework. Besides this, there is also a group of instructors who are keen to experiment with the functions of this new framework and use its advanced course management and evaluation capabilities. It is clearly visible that almost one-third of the courses are launching in the Canvas system, and that more and more instructors are setting up e-learning courses to support learning.

In a Canvas copy which supports informal and non-formal courses, instructors can request the creation of courses by filling in a form, after the verification of the data, the course is created manually. In this framework, a total of 2169 courses have been announced[2], most of which are such test courses in which instructors can test the capabilities of the system during the course, and experiment with certain functions. We have 510 courses in which there is real training, which means that students are also enrolled in.

For this system, we did not examine the number of courses in a breakdown by semester, because here the start and end dates of the courses can be set arbitrarily, so short (a few weeks) or long-term (multiple semesters long) courses can be started.

2 Possibilities for organizing and evaluating the Canvas course

In the Canvas framework, the first page of the course – which appears at first after entering the course – can be varied based on the choice of the instructor: the syllabus, the front page edited by the instructor, the assignments list, the course activity stream, and the modules list [1].

In the 2016/17/2 semester, we examined what instructors prefer to set as their home page. As a result, we got to know that most of the instructors chose the course activity stream as their home page (52%), followed by the modules (27%), the syllabus (13%), the content page (8%), and finally the assignments list (1%) setting [2].

[2] According to the March 31, 2019 status

2.1 Module Organizational options, experiences

In the Canvas framework, modules allow to group content according to a specific aspect, whether it is a variety of different topics or even teaching weeks. For each module, precondition can be set, as well as the conditions required for the module to be considered complete. The conditions may not only include hand-ins to be submitted, but also that students need to intervene in editing the page or writing a forum post. These options allow a wide range of course management solutions.

Among the published courses in the examined semester, 74 were found (35%) where the instructors created forums, while the number of comments was low, only 8% of the courses included more than hundreds of forum posts [2].

The way students have completed each module can also be easily queried with the „Module Progress" function. Here we can see which modules have been completed and which elements are still to be completed.

In the case of smaller courses with fewer thematic units, it is not necessary to design the modules, the topic itself or the syllabus can illustrate the structure and requirements of the course. But over a certain complexity, module management becomes essential. Regarding the 212 launched courses in the examined semester, we can state that there were no modules in just over half of the courses.

2.2 Consultation, organization of the exam work presentation

In blended education, it is common for students to access the curricula and exercises online, but they must defend their examinations personally at a given time. Even in the case of online courses, which only take place on the electronic interface, it may be important for students to be able to consult online with the instructors in a particular interval. In this case, the most ideal would be if students could register for the appointments offered by the instructor. This feature is supported default on the Canvas LMS by selecting the scheduler in the calendar, after which students can specify the characteristics of each consultation time.

2.3 Supporting teamwork and collaboration

Canvas's framework supports the set of groups at different levels. The so-called „Sections" should be used in case if we present a course parallel (or shifted in time) for multiple groups, and we want to differentiate the individual groups according to that the same tasks have different end lines. In the case of these sections, we can set that students are allowed to communicate with students from different sections, with their own groupmates or with everybody.

We can also set up groups when we are setting the tasks. The set of groups created to implement a particular project is called group set. When we create a group setting, we can declare whether we to allow self-organizing groups to be created, or that, students can apply themselves to a particular group.

We can also set whether group members should be in the same section or not. Canvas also creates a group work interface for each group, where students can collaborate virtually with each other (editing pages, creating forums, collectively editable documents, current announcements, etc.) so they do not necessarily have to meet in person.

Groups can be created randomly after we have specified the number of groups, but we can arrange students either manually, by using the drag and drop technique. In the groups, we can also manually select a group leader, but this can also happen automatically, which is going to be either the first joined student or chosen by the random selection. The group leader allowed to change the name of the group, which can be useful when the members themselves decide which project to implement.

When we are setting the tasks, we can also declare the task to be done in group work. We can either assign the task to an existing group set or assign it to the students individually.

During group work, it is essential that students can communicate with each other and work together. For this, the Canvas framework provides forums, internal correspondence, and shared real-time document management by integrating the Google Docs or EtherPad service.

2.4 Advanced evaluation options

To evaluate the tasks, the framework provides many convenient features. One of these is the „Speedgrader", which shows the uploaded files sorted by the students (in many cases these are viewable online), and on the interface, it is possible to add scores, evaluations, comments instantly, even based on rubrics, which we are going to discuss later on in more details. We can export the scores or import them from an external source into the framework.

The different types of tasks (e.g., teamwork, individual work, attendance, smaller classroom tests, hand-ins, micro-presentations) can be organized into so-called task-groups, and the achieved scores can be weighted (Figure 1.) according to these groups [1].

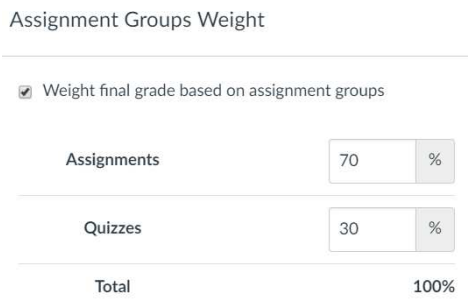


Figure 1: Setting assignment groups weight

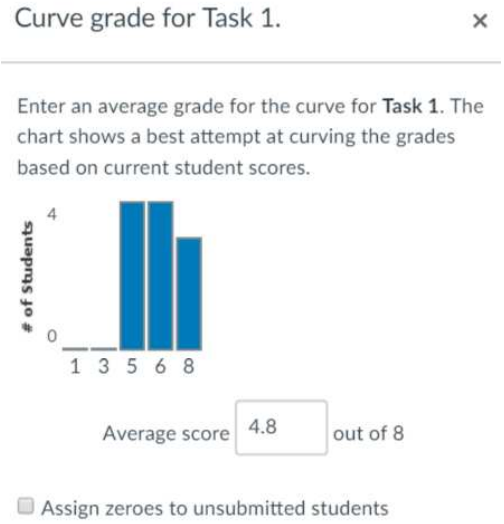


Figure 2: Curving grades for an assignment

In the case of published assignments, quizzes, surveys, it is worth noting that we can order extra chances and time for students. This option is suitable for remedying possible technical errors (e.g., computer crashes) just like enforcing the rights of disabled students (e.g., extra preparation time).

Due to the nature of the tasks, we can apply criteria-oriented or norm-oriented assessment [3]. In the last case, the performance of the students can be compared to each other or to the achieved average by a reference group. Throughout this, we can determine that the student's performance is in which performance area. Therefore, in such evaluations, it would be important to compare student's performance to the achieved average by the group (or reference group). This function of the Canvas LMS (Figure 2) is suitable for this to correct the scores achieved by students.

The system allows the evaluation of tasks in the form of peer reviews. Several advantages are known for using peer reviews [4]:

- Peer review builds student investment in writing and helps students understand the relationship between their writing and their coursework in ways that undergraduates sometimes overlook.
- Making the writing process more collaborative through peer review gives students opportunities to learn from one another and to think carefully about the role of writing in the course at hand.
- Studies have shown that even strong writers benefit from the process of peer review: students report that they learn as much or more from identifying and articulating weaknesses in a peer's paper as from incorporating peers' feedback into their own work.
- Peer review provides students with contemporary models of disciplinary writing.
- Peer review allows students to clarify their own ideas as they explain them to classmates and as they formulate questions about their classmates' writing.
- Peer review provides a professional experience for students having their writing reviewed.
- Peer review minimizes last-minute drafting and may cut down on common lower-level writing errors.

Peer reviews can be set during the process of setting the individual tasks, after that, we can set manually that who evaluate whose work (even in an anonymous way) or we let the system to assign the tasks.

In order for the evaluation to take place from a specific angle, we can also associate the tasks with a so-called rubric (Figure 3), which helps to allocate the partial points easily. In the absence of this, the evaluation of the tasks will be highly subjective and unfair, because it greatly depends on the evaluator's own aspects, and students will not know the reason why they lost points.

Blog writing

Criteria	Ratings					Pts
Quality of Writing	4.0 pts Expert *very informative or deeply reflective; informational post: synthesizes learned content and constructs new meaning; well organized*	3.0 pts Accomplished *some new information on the topic or reflective; informational post: attempts to synthesize information and form new meaning; well organized*	2.0 pts Capable *gives some new information on the topic; informational post: has trouble with integrating read or learned information and mostly repeats without construction of new meaning; poorly organized*	1.0 pts Beginner *gives no new information on the topic; nformational post: there is little to no evidence of other readings or information in order to form new meaning; poorly organized*	0.0 pts No Marks	4.0 pts
Presentation	4.0 pts Expert *all words spelled correctly; no grammar errors formatting makes the post more interesting and easier to read*	3.0 pts Accomplished *few spelling errors; few grammar errors; some formatting to help make the post easier to read*	2.0 pts Capable *several spelling errors; several grammar errors; formatting makes post difficult to follow or read*	1.0 pts Beginner *many words misspelled; many grammar errors; formatting makes post difficult to follow or read*	0.0 pts No Marks	4.0 pts

Figure 3: An example Rubric about the blog writing

Therefore, to determine grades based on student results, we need to create a so-called course grading scheme in which we specify the intervals for each grade according to the percentage of the results they achieved. We can set the grading schemes at the system level, so the instructors only have to choose from the predefined schemas and they only need to create new ones if they want to grade based on their own schema.

Besides the determination of the grades, students' learning outcomes can be specified in more detail (or it is maybe even recommended to do so), to achieve this in the Canvas framework there is a menu called „Learning outcomes" [1]. Learning outcomes are „*statements that describe the knowledge or skills students should acquire by the end of a particular assignment, class, course, or program, and help students understand why that knowledge and those skills will be useful to them*"[3].

In the Canvas LMS system, learning outcomes can be included among the criteria of the rubrics for the test forms used in the course, and scores can be assigned to each level of skills [5].

2.5 Statistics, analytics

The instructor's work is supported by built-in statistical and analytical modules. In the statistics which are available for quizzes and surveys, there are different metrics (achieved minimum, maximum, standard deviation, average score, average fill time) available, and we can see the results even sorted by questions, where in addition to the achieved results the discrimination index (D) is available too (Figure 4).

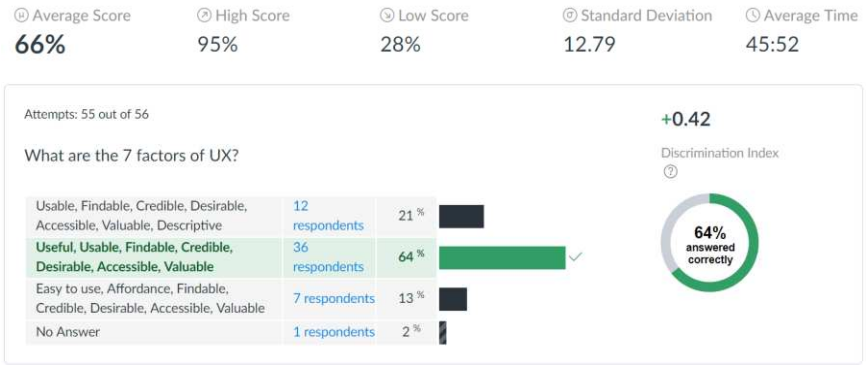


Figure 4: Results including the Discrimination Index

The item discrimination index is a „measure of how well an item is able to distinguish between examinees who are knowledgeable and those who are not, or between masters and non-masters[4]". For the quizzes to differentiate appropriately between students, take a look at the achieved result, so we can improve the surveys based on those results. Questions with an index lower than 0.20 are worth reviewing, and the ones with the value of negative D are suggested to be deleted, or maybe it is advisable to check the key that there is no error in it.

[3] What Are Learning Outcomes? – Centre for Teaching Support & Innovation. University of Toronto https://bit.ly/2IPU5fU

[4] Professional Testing, Inc | Test Topics. http://www.proftesting.com/test_topics/steps_9.php

Statistics are available for the entire course, including several levels. In a base level, we can see the number of forums/forum posts, tasks, uploaded hand-ins, quizzes, the number of students, we can query the names of the most recently enrolled students, as well as the information about the usage of the storage. More sophisticated statistics are provided by the so-called „analytics module" (Figure 5). Here we can see how page views and other activities have evolved over the past period, and we can track the status of each submissions status (on time/late/missing tasks).

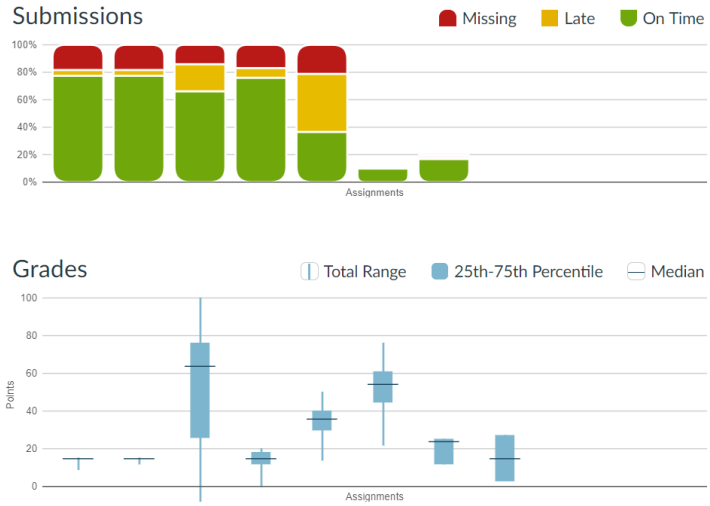


Figure 5: Course Analytics

For the tasks that have already been submitted and evaluated, a Box-Whisker diagram shows the most important features.

A box and whisker plot is a graph that presents information from a five-number summary. It's ideal for comparing distributions because the center, spread, and overall range are immediately apparent. In a box and whisker plot: the ends of the box are the upper and lower quartiles, so the box spans the interquartile range, the median is marked by a horizontal line inside the box, and the whiskers are the two lines outside the box that extend to the highest and lowest observations[5].

Under the analytical charts, we can also see the full list of the names of the courses, where each student has the following data: page view, participation, submissions, on-time/late/missing submissions, and the current score. For example, based on these data, we can easily identify drop-out students and give them targeted additional support materials or even consultation times.

3 Launch open courses (MOOC) in the Canvas framework

As part of ELTE's e-learning strategy, we are not only launching courses that support formal education, but also open courses both in Hungarian and English for which anyone can register. The first stage of achieving this goal was the „Open Course Designing"[6] MOOC course we launched in October 2018, which will be followed by new courses in a wide range of topics in the near future.

[5] Statistics: Power from Data! Box and whisker plots; https://www150.statcan.gc.ca/n1/edu/power-pouvoir/ch12/5214889-eng.htm

[6] https://mooc.elte.hu/courses/451

The course was facilitated from October 2018 to early December, with 1096 participants participating in it, and a total of 1552 forum posts were created in the 88 forums.

One of the specialties of our course was that we provided access not only to one course, but we have also created a test course for each participant, who was able to test the possibilities of Canvas LMS with an instructor and prepare for their exam.

The evaluation of the examination task was carried out in the form of self-evaluation and peer review, in which the (public) learning journals led by the participants were given an important role, where they could reflect on what they had learned in each module.

4 Instructor feedback

Following the introduction of the Canvas, feedbacks were gathered from the instructors about the experience of using the system [2]. The instructors had a need for the use of template scenes, so when they start a course, they can start with different types of templates. Although this is not directly supported by the system, there is no obstacle to offer sample packages for instructors which can be imported into the system at the beginning of the semester, so they can fill it up with content.

The usefulness of the student view (course view, testing it with a student test user account) has been highlighted by several people, as it is the easiest way to test how the course is presented to the students.

Quizzes and surveys are used by many instructors, but the question of the cloning option is often denounced by many, just like importing a question bank from another system which causes many problems. Several instructors indicated that it is very easy to assign extra time for filling surveys/quizzes for those students who got special educational needs; it is also a very useful feature. It is also easier for many instructors to limit the types of the files uploaded by students to each task, such as when we expect a PDF file, so students cannot upload a file with a DOC extension, which makes the corrections process easier. Some of the instructors would highly appreciate the option of a mark to indicate that the task should be revised, now it is only possible to give notes, instructions, and opinions in a free text block beside the scoring.

Currently, student attendance can be recorded in the same way as the assignments, which according to the instructors should be separated from the assignments.

5 Conclusion

We can see that in the Canvas LMS system, there are a number of functions that fit into the modern pedagogical toolkit, the advanced course management, and evaluation solutions can be used in both mixed and online educational forms. However, in order to maximize the use of these opportunities for instructors, it is necessary to initiate methodological training.

The Canvas LMS's functionality can be expanded to include LTI (Learning Tools Interoperability) tools, including gamification, absences management, content sharing, and custom assessment examination functions, in addition to the features described in this article.

Bibliography

1. Canvas Doc Team: Canvas Instructor Guide. 2019. [on-line] https://community.canvaslms.com/docs/DOC-10460
2. Abonyi-Tóth, Andor; Tóth-Mózer, Szilvia: A Canvas LMS használatának tapasztalatai az ELTE képzéseiben. In: Nádasi, András (szerk.) Agria Media 2017 : „A digitális átállás a tanulást élménnyé teszi" = „Digital transformation as a key to experience - based learning" Eger, Magyarország : EKE Líceum Kiadó, (2018) pp. 49-57. , 9 p
3. Hambleton, R.K., Swaminathan, H., Algina, J., & Coulson, D.: Criterion referenced testing and measurement: a review of technical issues and developments. Review of Educational Research, 1978, 48(1), 1-47.
4. Southwestern University: Benefits of Peer Review. [on-line] https://bit.ly/2WiK7ux
5. Linda Crocker: Introduction to Measurement Theory (in Handbook of Complementary Methods in Education Research, Routledge, 2012, 371-384)

The course "Designing Open Courses" was created in the framework of the EFOP-3.4.3-16-2016-00011

Authors

ABONYI-TÓTH Andor

Eötvös Loránd University, Faculty of Informatics, Department of Media and Educational Informatics, Budapest, Hungary, e-mail: abonyita@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 1. 2020.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.1.419

License

Didactic Options and Experience with 3D Printing Technology and Lego Mindstorms Kits

Igor ČERNÁK, Michal ROJČEK, Patrik SITIARIK

Abstract. The article discusses didactic possibilities and experiences with the use of 3d printing technology and Lego Mindstorms. 3d printing we use to teach computer graphics. It is essential that the teacher has a knowledge of how technology works and how it can implement it into teaching. The same is true of the Lego Mindstorms, which we use to teach programming and robotics. The article focuses on both technologies in terms of their capabilities and practical experience in teaching information subjects.

Keywords: 3D printing, Lego Mindstorms, RepRap, Robotics, Programming.

1 Introduction

Today time is marked by consumism and this is also true in the realm of education. Students are taught more to receive content (consume it) than to create it. It's hard to take them because they've already watched thousands of websites and videos, and they're hard to surprise or amaze. The way it could be changed is to teach students that even they can create something. Show them the potential of their own creativity, imagination or artistic creation. Involve not only the head, but also the hands. Teach them to think and create. Look at things in a comprehensive perspective. Analyze and systematize. Use new options to do this. Some of them like 3D printing and Lego Mindstorms kits will be presented in the following chapters.

2 3D Printing

3D printing is a phenomenon that has been greatly spread over recent years to schools, households and small businesses, especially since 3D printers have become affordable for these groups of users as well. In 2007, RepRap was given a project of a self-closing printer, which apparently had little contribution to the massive expansion of this technology. RepRap is an international community project of 3D printers developed on the principle of open hardware. The RepRap 3D printer is composed predominantly of many plastic parts that can be printed on another RepRap printer. The name RepRap is the abbreviation of Replicating Rapid Prototyper, which means that it is capable of self-replication and rapid prototyping. The entire documentation needed to compile the hardware and the operation of its own RepRap, including firmware and control software, is released under the GNU (General Public License) license, under which free software is issued [1, 2]. Thanks to its overall openness and affordability, RepRap has become a very popular project for the worldwide community of domestic champions. In some mail stores it is possible to buy all the parts on RepRap at favorable prices. A simple 3D printer can be bought already in order for a few dozen euros. Not only the open hardware community but also established companies that deal with the development and sale of professional 3D printers offer cheap models for home use.

Figure 1: 3D printer XYZ da Vinci Mini w

The affordability of this technology opens up new possibilities in teaching in the development of information and technological thinking, and the artistic sentiment of pupils and students. It provides a platform through which pupils and students can make their ideas more tangible. They have a choice of different software tools for designing 3D models that are free. From simpler tools designed for children and beginners e.g. Tinkercad [3], to more complex, designed for advanced and professionals e.g. FreeCAD [4] – parametric modeling, or Blender [5] for general 3D graphics. All changed tools can export 3D objects to the STL (stereolithography) format, which is suitable for 3D printers.

Students learn to move around in 3D space when creating a 3D model. They detect that the object should be viewed from multiple pages for editing. They learn what is a trait, bokorys, floor plan, orthographic and Perspective view. Basic graphic transformations such as shift, rotate, and scale change. They learn a variety of modelling techniques, such as parametric modelling, modelling based on the application of the Boolean operation of the association and the difference, or their combination. For parametrical Modelling, you specify the object's exact numeric parameters such as height, width, depth, number of polygons, rotation angle, magnification percentage, etc. When modeled using Boolean, the unification operation creates new objects by pooling multiple bodies into a single body. In the operation of the Booleovský difference, they "cut off" from the solids using other bodies. The TINKERCAD environment is also handled by children of first instance. They can draw their own toy and print them on a 3D printer with the help of the teacher. Is there more motivation for children than this?

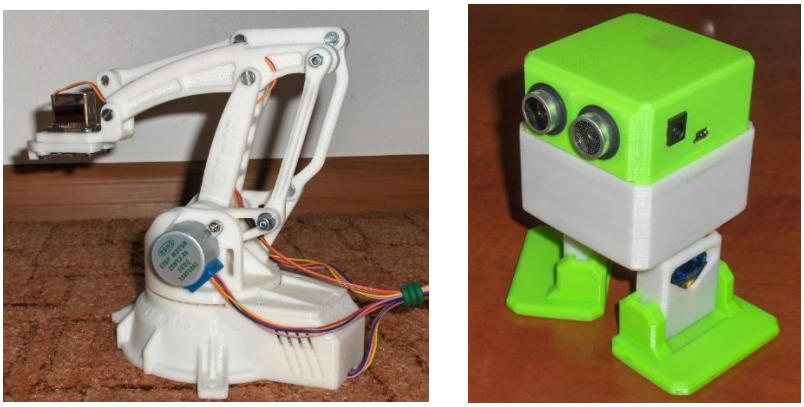
If the teacher wants to reach for the finished 3D models, they are available in large databases. For all I think very popular www.thingiverse.com, where 3d models are of different types or specialized Database of technical 3d models www.grabcad.com, or search engine 3D models www.yeggi.com, which looks for 3d models in multiple databases at once.

If the object is difficult to modeling, and in the 3D model databases (a typical example can be a human head of a particular person), it remains only to purchase or produce a 3D scanner that can be scanned by the object. From affordable models there is not so much choice. In this case, we recommend that you focus on quality software because it depends on how the missing pixels will be counted. Most of these scanners leave the scanned 3D models of holes, which must subsequently be "implicate".

If you already have pupils, whether students prepared a 3D model, followed by the phase of 3D printing. Here we recommend to clearly use the wholesome printing material (filament) of the PLA (PolyLactic Acid) plastic. This plastic is characterized by the fact that when the press is not lit, it does not release the harmful fumes when heated and is wholesome. In addition, the 3D printer may not have a heated washer as in ABS plaste. To snap a model, just stick glue (you need to try more brands). The printing process is time-consuming, so you need to schedule your lessons to print your model every time. For example, at the end of the lesson you can print models (preferably more at once), depending on the size of the print space and the size of the models. We recommend that you introduce restrictions on the size of the object due to the parallelism of the printing, but also the consumption of the print filament. Then pupils and students can take home their product and boast to their parents or friends with their creations.

With us at the Department of Informatics PF KU in Ružomberok, we use 3D printer and 3D scanner especially in the subject of computer graphics, where we teach different methods of 3D modeling. 3D printing can also be used in other subjects: for example, computer system architectures or the basics of electrotechnics for informatics, where we use the Arduino electronic kits. Just 3D Printing and Arduino (or other type of microcontroller) can be used to design and produce more complex mechatronic constructions and equipment see. Figure. 2. In the last time, we have dealt with the production of robotic hands in the department of EEZYbotArm MK3, which consists of 21 plastic parts, which we pushed approx. 5 working days. Electronics form the Arduino microcontroller with a four-axis stepper motor driver, 3 Step Motors and one servo motor to handle. The Model implements reverse kinematics.

Another model is Otto robot, which can be variously modified, skinned and programmed eg. in child prog. language Scratch. It consists of six plastic parts and electronics formed again by the Arduino microcontroller, USG sensor, 4 micro servos, piezo buzzer, bluetooth module and other optional components. The Robot can be used as a didactic hint for Scratch.



a) Robotical hand EezybotArm MK3 b) OTTO educational robot

Figure 2: Combining 3d printing and electronics.
The products we did in the Department of Informatics PF KU in Ružomberok.

Both models can be found on the page www.thingiverse.com, where they can be downloaded freely. The Open hardware platform is very promising and monetization of this platform is acceptable [6].

Another prospective platform is the Lego Mindstorms kit, which will be presented in the next chapter.

3 Lego Mindstorms kids

Robotics and cybernetics are often mentioned terms in professional articles and scientific publications. These terms bind with the automation of the activity that was previously performed manually. The process of automation thus leads to greater efficiency and productivity. Recently, however, the field of robotics is increasingly emerging on the campus of universities and secondary schools.

The area of robotics and automation is also expanding into education, especially on the grounds that the robot can cause attention and interest to the pupil. Increased attention and interest also results in increased motivation to work and learn something new. Lego Mindstorms are an excellent means of first familiarise pupils in pre-school age with the world of robotics. These kits are used not only in high schools but also in secondary schools. There are even common cases where children first become acquainted with this building at elementary School [8].

The Lego Mindstorms kit features a classic Lego cube, a microcontroller module that is also a basic control unit, motors and sensors. History indicates 3 base types of these cubes microcontroller. RCX, NXT 2.0 and EV3. Currently, for the purposes of studies, other uses are mainly used by NXT 2.0 and EV3 microcontrollers.

Microcontrollers NXT and EV3, i.e. the latest generation of these smart cubes are sold together with the Lego kits. In the kit with NXT 2.0 we can find 3 Step Motors, 4 sensors and the Lego parts themselves. The basic sensors that are supplied to the kits are touch, light, acoustic and ultrasonic sensors. Each of these sensors performs different tasks. The touch sensor responds to pressure; This means that when you press it, it performs a task according to which it is programmed. Depending on the program, we can, for example, control the motor movement, react to any other sensor or many other instructions by pressing the sensor. The light sensor recognizes colors and can react to changing lightness also. Again, as in the previous case, it can be used to interact differently with engines or other sensors. The last two sensors that we can find in such a kit are sound and ultrasonic sensors. The audio sensor responds to the sound that is in the vicinity and, depending on it, performs functions by program. The ultrasonic sensor is specific in that it can measure distance and, accordingly, at certain defined values of this distance, change its characteristics of the movement or running of the programme [11], [12].

Figure 3: The NXT Robot Model [17]

For developers and those sensors that are not enough, there is a possibility to purchase various additional sensors such as accelerometer, compass or RFID sensor.

There are some notable differences between the NXT and EV3 microcontrollers. The NXT cube is controlled by a 48 MHz processor with 64K memory while the EV3 cube (see fig. 4.) controls 300 MHz processor with 64 MB of memory. Additionally, when NXT is a cube, the memory space is limited to 256 KB, which may be a big problem for more complex programs. Ev3 microcontroller has an internal memory space much richer + there is a possible extension of memory via USB or SD card to a few GB. The differences between the processors and the memory in these two microcontrollers are reflected in particular in the execution of complex computational operations and the conversion of records per second.

Basic communication on both Lego cubes is via a USB cable that connects directly to the computer in which the program is already ready. The NXT Cube offers more comfort and communication via Bluetooth. The EV3 cube in addition to USB and Bluetooth also offers the ability to control and connectivity via Wifi, which is a great advantage especially for computers that do not have a Bluetooth interface in the base construction.

Figure 4: Options for plugging sensors and motors on a microcontroller EV3 [18]

We can programme such smart NXT and EV3 cubes in several ways [9]. The most frequently used are the environment LEGO MINDSTORMS NXT 2.0 and LEGO MINDSTORMS EV3 Home Edition [10], [11], [12]. These two environments are iconic, that is, they are fully graphic and pupils do not need to write long lines of code. They simply store individual graphical blocks of execution in succession and then upload them to the assembled robot. Figure. 5.

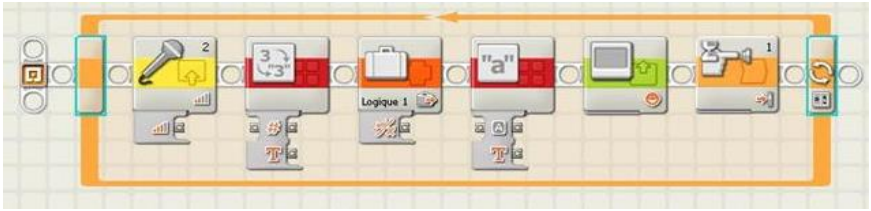


Figure 5: Example of a program in THE NXT-G development environment

In addition to the iconic or graphic programming of robots, there is also the possibility to program the robot directly on a brief microcontroller or programmed in a BricX environment [13], [14], [15]. In the BricX environment, you can write long lines of code and over the icon programming here, students have the opportunity to create a more sophisticated and accurate program. The EV3 cube also supports a higher C programming language and can therefore compile the code in any C compiler.

The basic use of Lego Mindstorms is mainly in teaching. In the course of lessons measured for programming and robotics, these kits can be use in particular as a means by which pupils are actually physically familiar with the possibility of programming robots. In the teaching hours, pupils

first become acquainted with the hardware of the robots and then with individual development environments also. Subsequently, students can start working on the robot's own project and program various tasks and functions. Students so of course can take advantage of their imagination as a result of the various interesting projects you see. Figure. 6.

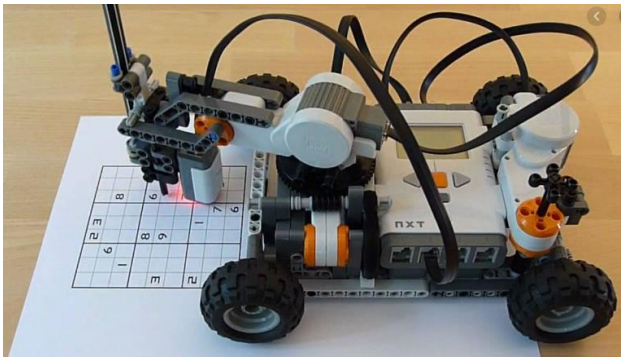


Figure 6: NXT Robot enabling the evaluation and registration of SUDOKU [19]

In addition to the classic lessons, such a kit can be use for other forms of teaching or events also. The University of the Catholic University of Ružomberok is organized annually by the Children's College, which is dedicated to kids from 7 to 12 years, where the workshops from informatics are presented with such Lego Mindstorms. Figure. 7. The children can be familiar with the robot for the first time and have the ability to program them directly on a microcontroller's cube, where they try to work with sound, ultrasonical (ultrasonic) and light sensors. Working with the pupil's bots is very interested in gaining new knowledge and skills from this area. In the children's University was organized a workshop from informatics, which lasted two lessons. In this time, we have been able to present a colleague with colleagues, and what it consists of. The beginning of this workshop was developed in the spirit of the representation of individual pupils. In this section, the pupils gave us their names, where they are, their hobbies and whether they have been met from robotics [16]. Most of these children have already met from robotics just at the workshop, which took place in the previous year. We also introduced the EV3 and NXT cubes and presented a detailed description of what they are composed of. Students could see the wiring of the sensors and the engines also. Then we divided the pupils into groups of three and each group gave a pre-prepared robot, which was constructed according to the basic NXT manual. Pupils gradually programmed this robot directly on a microcontroller's cube. In this part of the lesson pupils programmed the robot a total of 4 times. The programmes were mainly related to the programming of sensors namely touch, ultrasonic, light and acoustic. After this activity, we have finally demonstrated a demonstration of more complex robots to see what all the pupils can build and program through this Lego Mindstorms kit.

Figure 7: photograph of pupils at the workshop from the Catholic University in Ružomberok

We start working on lessons with our teaching science students by explaining the details of what is robotics and cybernetics. We also introduce well-known companies dealing with this issue. Among the most popular company includes Boston Dynamics. This company has seen great advances over the past year in the field of robotics. We divided the students into groups and each of the groups chose one robot to be present for the next hour. After these presentations, we joined the hardware and software, which has the Lego Mindstorms kit. We have described a specific processor, memory, connectivity and compatibility options. We have mentioned what different types of cube can be programmed. In the next part, students build a basic robot according to NXT instructions and try how to program the robot directly on a microcontroller's cube. After these activities, students already know what a NXT robot is and how to program it at a glance and what part is folded.

The next lesson is followed by programming in the environment of Lego Mindstorms NXT 2.0 G and also programming in the Advanced online website makecod.mindstorms.com [14]. Within these two environments, pupils will meet from a more complex programming, which cannot be achieved by programming on a microcontroller cube. The page makecode.com [14] is a wide variety of programming options for a newer EV3 cube with a wide range of tutorials. The student may display a blockprogramming environment similar to the child's Scratch programming language, but can also convert it to a Java code that the cube EV3 understands. The students are ready to create their own project after analyzing the specific environment. This project is characterised by the fact that the students in pairs will build their own robot according to their own status and create a separate code for it. They will use all of their previous knowledge in this way, resulting in interesting projects presented either in the trial or in the end of the teaching unit [13], [16].

4 Conclusion

From practical experience, it can be noted that the technology presented to students is very motivable, as they open up the well-known perspective of materializing their ideas by means of 3D printing, the development of motor skills and Technical-Informative thinking using Lego Mindstorms, and both gives a space of creativity and imagination. After all, what is more motivating for man than self-realization and the subsequent award?

References

1. REPRAP.ORG - RepRapWiki [online] reprap.org/wiki/Main_Page .
2. HUMMING, M. 3d Printer RepRap: Open-source lubricants that make your life easier. *Linuxexpres*, 2012-09-10 [online] www.linuxexpres.cz/hardware/3d-tisk.
3. TINKERCAD - From mind to design in minutes [online] www.tinkercad.com .
4. FREECAD - Your Own 3D Parametric Modeler [online] www.freecadweb.org .
5. BLENDER - Open Source 3D creation. Free to use for any purpose, forever [online] www.blender.org .
6. GIBB, A - Building Open Source Hardware: DIY Manufacturing for Hackers and Makers, Addison-Wesley: New York, pp. 253–277 (2015).
7. Robotics in informal teaching of informatics. DidInfo 2019, 03.04.2019-05.04.2019, Slovakia, Didinfo 2019: International Conference on Computer Science teaching: 25. Annual conference: Matej Bel University in Banská Bystrica, 2019. ISBN (online) 978-80-557-1533-9. -ISSN (online) 2454-051X, p. 77-81. - io.fpv.umb.sk/didinfo/Zbornik_Didinfo_2019.pdf.

8. Developing informative thinking through 3d press and LEGO robots [electronic document] innovative trends in trade didacties linking theory and Practice teaching strategies to critical and creative thinking, date, Slovak/rec. prof. PhDr. Martin Bílek, Ph.D. doc. paeddr. Juraj chamber, PhD, Nitra (Slovakia): University of Constanina philosopher in Nitra, 2019. -ISBN (online) 978-80-558-1408-7. www.pf.ukf.sk/images/docs/projekty/2017/pC-Cp/konferencie/2019/APVV%20Zborn%C3%ADk%202019.pdf.
9. WIKI. ROBOTIKA.SK-building LEGO mindstorm NXT in the classroom [online] wiki.robotika.sk/robowiki/index.php?title=Stavebnice_LEGO_MINDSTORMS_NXT_vo_vyu%C4%8Dovan%C3%AD.
10. EDUCATION. LEGO.COM - MINDSTORMS SOFTWARE [online] education.lego.com/en-us/downloads/mindstorms-ev3/software.
11. HAVELKA, M. . – STOFFOVÁ, V.: Robotika - Stavba a programování robotů (LEGO Mindstorms NXT a EV3) 1. vyd. Olomouc : Pedgogická fakulta UP v Olomouci, 2017. 85. s
12. STOFFOVÁ, V.–.HAVELKA, M.: Práca s robotickými stavebnicami na 2. stupni ZŠ - Zbierka riešených úloh. 1. vyd. Olomouc : Pedgogická fakulta UP v Olomouci, 2018. 66. s.
13. STOFFOVÁ, V. – TAKÁČ, O.: Robotické stavebnice v príprave učiteľov informačnej výchovy (Robot kits in teachers preparation for information education). In Havelka, M., Chráska, M., Klement, M., Serafín, Č. (ed.): *Trendy ve vzdělávání 2013*. Olomouc : agentura GEVAK s.r.o., 2013. 315-322. s. ISBN 978-80-86768-52-6 / ISSN 1805-8949
14. Microsoft.com-Lego Mindstorms Education EV3 [online] makecode.mindstorms.com.
15. SOURCEFORGE.NET - Bricx Command Center [online]
16. STOFFOVÁ, V. – ZBORAN, M.: Hravá forma stavby a programovania robotov na základnej škole. In: *Trendy ve vzdělávání*, 11, 2018, č. 2, s. 130 − 139. ISSN 1805-8949
17. Figure 3 https://docplayer.nl/docs-images/70/63006544/images/16-2.jpg
18. Figure 4 https://a.allegroimg.com/s1440/035a32/ab060569413584c4506ecf66230a
19. Figure 6 https://miro.medium.com/max/2000/0*ZzhYR2ZYQKri27Kz

Authors

Igor ČERNÁK

Catholic University in Ružomberok, Faculty of Education, Slovakia, e-mail: igor.cernak@ku.sk

Michal ROJČEK

Catholic University in Ružomberok, Faculty of Education, Slovakia, e-mail: michal.rojcek@ku.sk

Patrik SITIARIK

Catholic University in Ružomberok, Faculty of Education, Slovakia, e-mail: patrik.sitiarik@ku.sk

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 1. 2020.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.1.416

License

Interactive Teaching of Programming Language Theory with a Proof Assistant

BERECZKY Péter, DONKÓ István,
HORPÁCSI Dániel, KAPOSI Ambrus, NÉMETH Dávid János

Abstract. Teaching of programming language theory has a long track record at ELTE Faculty of Informatics. Traditionally, formal semantics and type systems of programming languages, similarly to other theory-oriented subjects, were taught with the pen and paper method. However, modern proof assistants call for replacing this old-fashioned way of teaching with novel and interactive methods that bring deeper understanding, provide better learning experience and build technical skills in applying formal methods. The authors have launched practice classes for two programming language theory subjects and carefully developed course material based on executable and verifiable definitions formalised in the Coq proof assistant. In this paper, we share our experiences regarding the design and implementation of the new material, we outline the pros and cons of using a proof assistant in the courses, and we describe how the presented method may be adapted to other courses.

Keywords: formal semantics, type systems, proof assistant, Coq, interactive teaching

1. Introduction

Teaching mathematically precise formalisation techniques is a key part of university level Computer Science education. The courses titled *Formal Semantics* and *Type Systems for Programming Languages* are compulsory in the Software Technology specialisation of the Computer Science master program at Eötvös Loránd University (ELTE). The aim of these courses is that students acquire the skills necessary to apply mathematical methods when describing programming languages. During these courses the students have to digest and obtain intuition for a large number of abstract concepts and formal notations. This is especially hard for students with less mathematical affinity. By sort of "wrapping" the raw mathematical contents into detailed, practical examples, we can help the students grasp the constructions and obtain understanding. What helps even more is giving the students tools to freely experiment with which gives immediate feedback. To achieve this, we started employing computer proof assistant systems in our practical sessions. The current article describes the experiences gained from teaching with this new methodology.

2. Proof assistants in education

Proof assistants (interactive theorem provers) are software which give the user the ability to:

- Define detailed abstract models
- Construct statements over previously defined models
- Interactively prove statements using mathematical methods, the correctness of which can be checked automatically by computers

Several such systems were considered during the planning phase of our courses, comparing them based on previous experience in the literature regarding teaching and their compatibility with the previously outlined goals.

The Haskell programming language [6] is strictly speaking not an interactive theorem prover, but its purely functional nature and simple syntax makes it suitable for modelling programming languages. The abstract syntax can be represented with algebraic data types and the semantics can be described using executable functions in a denotational style. Haskell was previously used to

teach programming language semantics [11], but as it lacks dedicated theorem proving functionality, we decided against it.

Agda [1] is another functional programming language with a type system stronger than Haskell's. It supports dependent types [14], that is, types that depend on values. This strength allows representing mathematical statements as types, and constructive proofs as executable functions that implement the given type. This is usually called the propositions as types principle or Curry-Howard isomorphism [15]. Agda was also used before as a tool to teach programming language theory [10]. It uses the same notation for proofs and programs which makes it simple but confusing at the same time because paper-based proofs look very different from Agda proofs.

The Isabelle/HOL theorem prover [7] meets these requirements and its educational potential has also been demonstrated [2], but its type system does not support dependent types. This makes it more verbose, by requiring the user to manually handle information in certain cases that could otherwise be encoded into the types themselves.

We chose to use the Coq proof assistant [5], as it has a dependently typed specification language (*Gallina*) that is similar to Agda in expressive power, but it also comes with a separate tactic system that can be used to construct imperative style proofs which are similar to pen and paper proofs. Its widespread use in education [3], detailed auxiliary materials [9] and active support [4] have further strengthened its leading position.

3. The Formal Semantics course

In contrast to natural languages, programming languages are artificially created by humans for the purpose of efficiently controlling computers. Because of this, the meaning of their statements needs to be clear without any chance of ambiguity, so they can be interpreted by a computer. *Formal Semantics* deals with the task of describing the behaviour of programming languages and through this process defining the meaning of programs written in them.

The course has always played an important role in the curriculum of the Computer Science master programs at ELTE. It provides insight into the mathematical methods involved in formalisation of the meaning of programming languages. Different ways of describing the syntax are described, as well as static and dynamic semantics, including operational, denotational and axiomatically given semantics. Well known real world (mostly imperative) programming languages serve as examples in the formalisation process. This makes it easier to connect this new information with previous studies and helps deepening the understanding of the already acquired knowledge.

3.1 Aim of the course

The course has multiple goals, the most important of which is to popularise the formal definition of programming languages by clarifying the role of this process in the understanding, comparison and analysis of programs, as these are the key steps in proving correctness of a program. Students create their own mathematically precise descriptions of several programming language constructs they are already familiar with, e.g. branches, loops or exception handling. When doing this they need to carefully examine the behaviour of these constructs and thus gain a better understanding of their interactions, limits, the similarities and differences between their different implementations in various programming languages.

Throughout the classes several concepts from previous courses (such as syntax definitions from the *Compilers* course) are used which helps in refreshing, reinforcing and expanding their already established knowledge. We introduce methods for defining operational semantics (small-step and big-step) through the notions of configurations and inductively defined transition relations between them. Denotational semantics is explained using executable semantic functions and the principle of compositionality.

Furthermore, many of the mathematical concepts that the students encounter for the purpose of applying them in formal semantics are from set theory or algebra, thus applicable in their future studies. Examples are inductive definitions, pattern matching, induction, or parts of fixed-point theory.

3.2 Previous renditions of the course

The initial version of the course in the more mathematically focused program was made up of lectures and practical lessons (both 90 minutes each week for one semester), during which students could practice the use of the mathematics discussed in the lectures, but no interactive techniques were employed, the classes were based on the pen and paper method. From 2008 onwards the practical sessions were removed, only the lectures remained part of the course (with the same amount of time). This affected the effectiveness and popularity of the course negatively, as students had a hard time fully understanding the abstract concepts without adequate practice.

3.3 The new practical lab classes

In 2018 the course was renewed by including 90 minutes of practical session per week in addition to the 90-minute lectures. The syllabus had to be constructed in a way which was suitable for all students in their second semester of their master's studies. The instructors have agreed that giving the students a way in which they can experiment themselves is highly beneficial and being able to do so interactively through a computer is even better. As the goal was to not only formalise the definitions, but also prove theorems about them, a system with such capabilities was needed.

3.4 Main principles

The primary task of the practical sessions is to help the students understand the lecture material. This means formalising definitions, expressions, programs, theorems and examples. Unfortunately, most of the students are not familiar with proof assistants, therefore they need to learn how to use Coq without losing focus: this course should be on formal semantics and not the technical details of Coq's implementation. Our experience was that the following educational principles were helpful in this task.

1. **Understanding at the lectures, coding at the practical sessions.** The lecture material is presented sometimes through very complex examples in order to model real programming language constructions closely. These examples are discussed in detail in the lecture, but their machine-checked formalisation is too difficult for a beginner proof assistant user to implement. We could simply share the source code of such formalisations with the students (as it was the case with source code for executable semantics in previous renditions of the course), but one goal of the new practical sessions is to involve the students even more. In the practical sessions most of the time is spent on formalising

informal or half-formal definitions from scratch or using the material of the previous lessons. The students implement the definitions and prove theorems independently, rather than reading or modifying other existing formalisations. This way the time is not spent on understanding and discussing complex problems, but on practicing formalisation using smaller examples. Therefore, the sessions are focused on programming language semantics while at the same time the students gain experience in using Coq.

2. **First understand, then code.** The previous principle should not imply that students should blindly do coding before understanding the definitions, theorems and proofs. In fact, the practical session has the additional task to separate the understanding of the theoretical materials from the formalisation in Coq. During these sessions, the students try to solve small problems in complete detail, but only after they have understood the theory in detail. For this reason, whiteboard and paper is used when it is necessary, but these "analog" exercises get less emphasis than during the exercise sessions for other mathematical courses.
3. **Graduality in familiarising with the proof assistant.** Coq allows its users to create formalisations conveniently with a large number of language features. However, if we aimed to teach the students how to write idiomatic Coq code, we would end up not teaching formal semantics, but rather Coq through formal semantics examples. To keep the focus on the topic of the course, the language elements and functions of the theorem prover are introduced step-by-step; three-four elements (commands, tactics, etc.) per practical session. This style is similar to that of the book Programming Language Foundations [9]. The result is that students gain knowledge of the concepts and language of Coq progressively without getting lost in its details.
4. **Graduality in the complexity of the theorems.** The students usually neither have sufficient knowledge of theorem provers, nor of the process of theorem proving. Because of this reason, some proof theory has to be taught at the Formal Semantics course, while discussing the method of formalising proofs. In order to keep the focus and enthusiasm of the students, the complexity of proofs should be increased with caution. According to previous experience, the master's students have no problems with formalising simple functions with case distinction, however, that is not true for the transformation of derivation rules to inductive definitions. With adequate preparation we can maintain a continuous sense of achievement during the semester, which makes the usage of proof assistants a positive experience. This can be an important milestone in the students' professional development.
5. **Continuous work and short tests.** The requirements of the course were planned so that the students have to study every week. There is an optional homework assignment every week and each practical session starts with a short assignment similar to the homework. The results of these short tests determine the final grade for the practical sessions (students obtain a grade separately for the lecture). Every assigned task has to be solved in Coq to accelerate the acquisition of the proof assistant. The assignments are submitted in an online e-learning platform and the students get immediate feedback on the correctness of their submissions. This setup helps the students stay motivated throughout the semester.

3.5 Syllabus

The syllabus of the practical sessions is primarily based on the lecture materials and the aforementioned book [9]. Fortunately, the book gives an excellent basis for the swift introduction

to Coq, while the formal semantics examples discussed in it are often very similar to the ones presented during the lectures, because both sources deal with the concepts of imperative programming. The syllabus of the practice was assembled to follow and formalise the materials of the book during the first half of the semester and the materials of the lectures during the second half.

Below we describe the contents of the practical sessions for each week. To illustrate the increasing complexity of the material, we include four example Coq source codes snippets. We also list the usual informal proofs for comparison (although not always exactly these theorems are discussed during the lectures).

1. Introduction of the proof assistant by formalising bool type, then defining functions (e.g. and, or, not) followed by lemmas and theorems and their proofs (e.g. commutativity of and) by case distinction.
2. Inductive definition of natural numbers. Introduction of structural recursion (and pattern matching) by defining recursive functions (e.g. addition) and structural induction to prove properties about these functions in the form of lemmas, theorems.

```
Inductive Nat : Type :=
| O
| S : Nat -> Nat.

Fixpoint plusn (n m : Nat) : Nat :=
match n with
| O => m
| S n' => S (plusn n' m)
end.

Notation "n + m" := (plusn n m)
 (left associativity, at level 50).

Theorem plusn_rid : forall n : Nat,
n + O = n.
Proof.
  intros.
  induction n.
  * simpl. reflexivity.
  * simpl. rewrite IHn. reflexivity.
Qed.
```

Assume that natural numbers are defined based on the Peano-axioms with the 0 constant and the S successor function.

We define the addition in the following way, and denote with "+":

$$add(n,m) = \begin{cases} m & n = 0 \\ add(n',m) & \exists n' : n = S(n') \end{cases}$$

Theorem: $n + 0 = n$.

This statement can be proven by induction on n.

- Base case $= 0 : 0 + 0 = 0$ is true according to the definition of *add*.
- Induction hypothesis: $n + 0 = n$.

The statement to prove: $S(n) + 0 = S(n)$.
By the definition of *add*: $S(n) + 0 = S(n + 0)$.
By the induction hypothesis: $S(n + 0) = S(n)$.

3. Formalisation of binary trees followed by the expression language syntax with inductive definitions (deep embedding). Static semantics: mappings, functions with the domain of trees or expressions (e.g. the number of leaves in a tree, the number of literals, operations in one expression). Formalisation of the denotational semantics for the given expression language. Simple inductive proofs about the trees and expressions.
4. Transformation of expressions to equivalent expressions (optimisation mappings). Proofs about the meaning preservation of these transformations in the denotational semantics.
5. Introduction of states (variable environment) and the extension of the expression syntax and semantics with variables followed by lemmas and proofs about the behaviour of states.

```
Inductive aexp : Type :=
| ALit (n : nat)
| AVar (x : ident)
| APlus (a1 a2 : aexp).

Definition state : Type :=
```

We define the syntax of expressions with BNF ("n" denotes a natural number, "x" a string):

$$a \in Aexp ::= n \mid x \mid a_1 + a_2$$

```
ident -> nat.

Fixpoint aeval (a: aexp) (s : state) :
nat :=
match a with
| ALit n => n
| AVar x => s x
| APlus a1 a2 =>
aeval a1 s + aeval a2 s
end.

Definition update (s:state) (x:ident)
(n:nat) : state :=
fun y => if eqb x y
then n
else s y.

Lemma update_onlyx:
forall s:state, forall x x':ident,
forall n:nat,
~(x = x') -> (update s x n) x' = s x'.
Proof.
  intros. unfold update.
  rewrite <- eqb_neq in H.
  rewrite H. reflexivity.
Qed.
```

A state is a function from variable identifiers to natural numbers:

$$s \in State = ident \to N$$

The denotational semantics is a recursive function on expressions:

$$A: Aexp \to (State \to N)$$

$$A[\![n]\!]s = n$$

$$A[\![x]\!]s = s(x)$$

$$A[\![a_1 + a_2]\!]s = A[\![a_1]\!]s + A[\![a_2]\!]s$$

We will use $s[y \to n]$ to denote that s' state which is the same as s except $s'(y) = n$.

Theorem: for any s state, if $x \neq x'$ then $s[x \to n](x') = s(x')$.

The proof is simple. The $s[y \to n]$ and s are different only in what values that are mapped to x. So, for any x' that is different from x, these values will be the same.

6. Static analysis: free and bound variables, function and inductive definition (relation) about an expression being closed followed by lemmas and proofs about closed expressions.
7. Formalisation of small-step semantics for expressions. Practicing the formalisation of the inference rules to inductive definitions. Evaluation of example expressions in the presented semantics.
8. Formalisation of big-step semantics for expressions. Proof of equivalence with other (denotational or small-step) semantics.

```
Reserved Notation "c -=> c'" (at
level 50).

Inductive eval_bigstep : aexp * state
-> nat -> Prop :=
| eval_lit n s:
  (ALit n, s) -=> n
| eval_var x s:
  (AVar x, s) -=> s x
| eval_plus a1 a2 n m s:
  (a1, s) -=> n ->
  (a2, s) -=> m
 ->
  (APlus a1 a2, s) -=> (n + m)

where "c -=> c'" :=
  (eval_bigstep c c').

Theorem denot_iff_bigstep :
  forall a:aexp,
  forall s:state,
```

The big-step semantics can be described with inference rules between configurations of an expression and a state and a natural number:

$$\frac{}{\langle n, s\rangle \Rightarrow n}$$

$$\frac{}{\langle x, s\rangle \Rightarrow s(x)}$$

$$\frac{\langle a_1, s\rangle \Rightarrow n \quad \langle a_1, s\rangle \Rightarrow m}{\langle a_1 + a_2, s\rangle \Rightarrow n + m}$$

Theorem: The equivalence of the big-step and denotational semantics, i.e. for every expression a and state s and natural number n, $A[\![a]\!]s = n \leftrightarrow \langle a, s\rangle \Rightarrow n$.

We provide proof for the forward direction in this paper. This way, the hypothesis is$A[\![a]\!]s = n$.We use induction by a.

- We must prove that $\langle n', s\rangle \Rightarrow n$, i.e. $n = n'$. But according to the hypothesis, $A[\![n']\!]s = n$, that is only possible when $n = n'$.

```
  forall n:nat,
  aeval a s = n <-> (a,s) -=> n.
Proof.
  split.
  * generalize dependent n.
    induction a.
 - intros. subst. apply eval_lit.
    - intros. subst. apply eval_var.
    - simpl. intros. subst. apply
eval_plus.
      + apply IHa1. reflexivity.
      + apply IHa2. reflexivity.
  * intros. generalize dependent n.
    induction a.
    - simpl. intros. inversion H.
      reflexivity.
    - simpl. intros. inversion H.
      reflexivity.
    - intros. simpl. inversion H.
      rewrite (IHa1 n0 H4).
      rewrite (IHa2 m H5).
      reflexivity.
Qed.
```

- We must prove that $\langle x, s\rangle \Rightarrow n$, i.e. $s(x) = n$. According to the hypothesis $A[\![x]\!]s = n$, we know that $s(x) = n$, because of the definition of the denotational semantics.
- Induction hypotheses:
  - $\forall n: A[\![a_1]\!]s = n \leftrightarrow \langle a_1, s\rangle \Rightarrow n$
  - $\forall n: A[\![a_2]\!]s = n \leftrightarrow \langle a_2, s\rangle \Rightarrow n$

According to the hypothesis $A[\![a_1 + a_2]\!]s = n$, that means $A[\![a_1]\!]s + A[\![a_2]\!]s = n$. To $\langle a_1 + a_2, s\rangle \Rightarrow n$ it is sufficient to prove that $\langle a_1, s\rangle \Rightarrow A[\![a_1]\!]s$ and $\langle a_2, s\rangle \Rightarrow A[\![a_2]\!]s$ which are exactly the induction hypotheses with choosing $n = A[\![a_1]\!]s$ in the first case, and $n = A[\![a_2]\!]s$ in the second.

9. Introduction of imperative programming statements. Formalisation of their syntax and denotational semantics. Fixpoint theory, the question of termination.
10. Extension of the previous big-step semantics with statements followed by example program evaluation proofs.
11. Formalisation of additional examples, the extension of the semantics with other statements (e.g. counting loop as an inductive rule or as syntactic sugar). Proofs about the equivalence between big-step and denotational semantics.
12. The equivalence of program patterns (e.g. while loop can be unfolded to a conditional statement containing a similar loop).

```
Lemma while_unfold (b:bexp) (s:stmt)
(st st':state):
  (SWhile b s, st) -=> st' <->
  (SIf b (SSeq s (SWhile b s))
         SSkip, st) -=> st'.
Proof.
  split.
  * intros. inversion H.
    - subst.
      apply eval_if_true.
      apply (eval_seq st'0 _ _ _ _
H3 H5).
      exact H6.
    - apply eval_if_false.
      + apply eval_skip.
      + exact H4.
  * intros. inversion H.
    - subst. inversion H5. subst.
      apply (eval_while_true st'0).
      + exact H4.
      + exact H7.
      + exact H6.
```

Theorem:
$\langle while\ b\ do\ S, s\rangle \Rightarrow s' \leftrightarrow$
$\langle if\ b\ then\ S; while\ b\ do\ S\ else\ skip, s\rangle \Rightarrow s'$.

The hypothesis $\langle while\ b\ do\ S, s\rangle \Rightarrow s'$ could only be gotten by two different ways: either b evaluates to true or false.

If b evaluates to false, then according to the semantics of the loop, the original state s will be the result. So, to prove the original statement, it is sufficient to prove (because of the condition of the **if** statement evaluated to false) that $\langle skip, s\rangle \Rightarrow s$ which is exactly the semantics of skip.

If b evaluates to true, then according to the semantics of the loop, we get two new hypotheses: $\langle S, s\rangle \Rightarrow s_1$ and $\langle while\ b\ do\ S, s_1\rangle \Rightarrow s'$. In this case it is sufficient to prove (because of the condition of the **if** statement evaluated to true) that $\langle S; while\ b\ do\ S, s\rangle \Rightarrow s'$. In order to prove this statement, the inference rule of the sequence can be used. We need to provide an intermediate s_2 state,

`- subst. inversion H5.`
`apply eval_while_false.`
`rewrite <- H0. exact H6.`
`Qed.`
	that conforms the next two statements: $\langle S, s\rangle \Rightarrow s_2$ and $\langle while\ b\ do\ S, s_2\rangle \Rightarrow s'$. Fortunately, s_1 is such a state.

13. Extraction of correct Coq code to Haskell. Formal semantics in the practice, industry.

Because of the structure of this step-by-step material, the code is usually reused from previous session (as is the case for the denotational semantics or the syntax of expressions in the example codes above). In order not to spend time on redefining these always from scratch, the necessary source code is shared with the students before the session. In addition, it is also useful to make the code discussed in the lesson accessible which can be used by students later when they solve the homework or practice for the next test.

3.6 Deep understanding of complex concepts

As mentioned before, there are concepts and methods that are challenging to understand, even for master's students. The practical sessions have a very important role in this perspective, which is to introduce these complex, abstract concepts through small, simple examples. Therefore, during the semester these are practiced on gradually larger and harder examples. According to our previous experience, the teaching of the following concepts became easier due to the practical sessions:

- *Derivation rules (inference rules) and derivations*: It is not always clear for the students why do we formalise the operational semantics in a concrete way; what can be written over and under the line (which separates the premises and conclusion) of the inference rule. The students are forced by the system to follow the rules of the formalism given by inductive types in Coq. As a consequence, the students grasp faster and easier why, how and where derivation rules can be applied. Probably this is also due to the pattern matching mechanism in Coq, which is familiar to students who previously studied functional programming (which is a compulsory course in our bachelor's programme).
- *Structural induction*: The students learned about mathematical induction (for natural numbers), but they are not familiar with the concept of structural induction for arbitrary inductively defined sets. So, the scheme used for inductive proofs is not always understandable for them in the case of structurally complex types; which hypotheses should be used, which statements are not correct implying an incorrect assumption, what should the induction be based on, etc. Usually, paper-based proofs are more readable, however, they can be faulty or only partial. On the other hand, the proving process in Coq is interactive (using its integrated development environment, *CoqIde*), so that the students are led by the proof assistant through the process step-by-step. In every step, Coq reports the current hypotheses and the statements to be proven, and it does not allow to take faulty steps. Last but not least, each proof goal has to be proven, including trivialities.
- *Compositionality*: The principle of compositionality is usually just learned word by word by the student, but they do not understand its essence and its potential in practical applications. However, when proving a statement by structural induction on expressions the proof assistant shows clearly, that the induction hypotheses are the same as the statements to be proven about complex structures thanks to the compositionality. Similarly, Coq's termination checker accepts compositional recursive definitions but rejects those where recursive calls are not on structurally smaller arguments.

3.7 Results

The effectiveness of the new practical course can be measured objectively by looking at the grades received at the end of the semester. Here we only evaluate the exams based on the syllabus of the lectures as these had the same requirements as in the previous years. This was the first year of the practical sessions, so there is not enough data to draw far-reaching conclusions.

The oral exam aims to measure the lexical knowledge as well as the understanding of the material. Because of the large number of students there is also a written exam which helps in filtering out those who are not sufficiently prepared and also gives a chance to those who are not really interested in the subject to get a passing grade. If someone successfully completes that, they can choose to continue with the oral exam to obtain a better grade.

3.7.1 Exam results

The reintroduction of the practical course has clearly improved the results of the exams. This is most likely due to the students not only memorizing the theoretical subject matter, but actually learning and understanding it more. In the following we compare the results of the 2018 Spring (57 students) and 2019 Spring (61 students) semesters.

- **Improvement of the average mark.** In Hungary, the grades range from 1 (failed) to 5 (excellent). When looking at all the participants of the exam, the average of all received marks raised from 2.6 to 3.14 without significant changes to the written pre-exam and having the same people conduct the exams (see Figure 1).

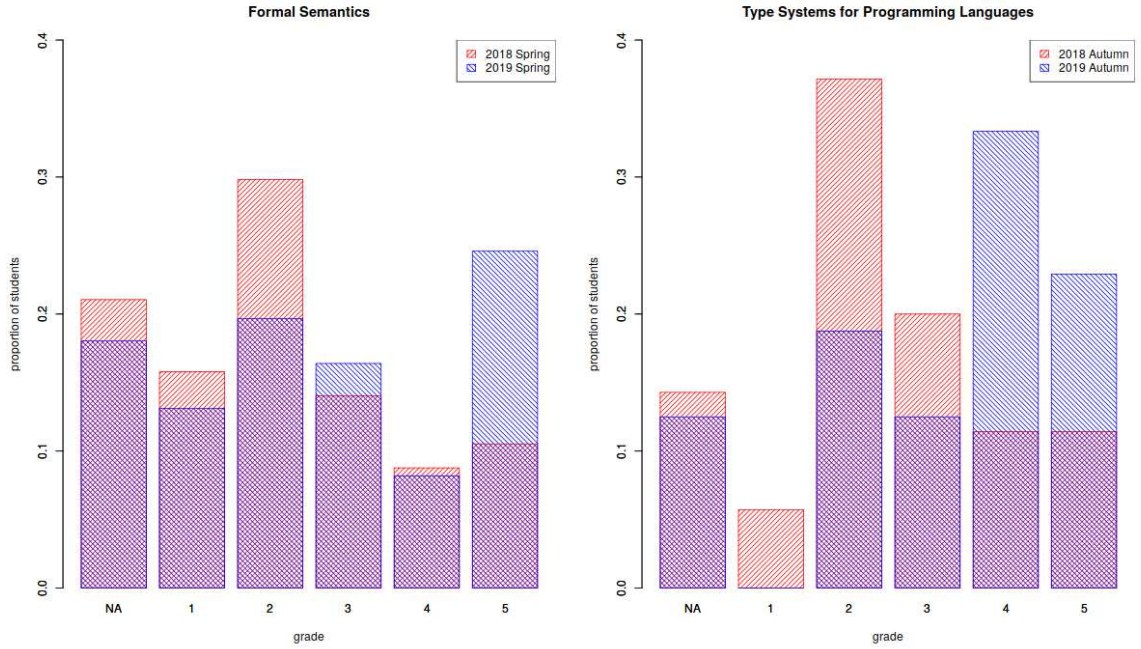


Figure 1: Exam results in the two investigated courses comparing the semesters without (red) and with (blue) practical sessions.

- **Increased participation in the oral exam.** Although students have the option to leave with a passing grade (2) after a successful written pre-exam, this year the ratio of those who chose to take the opportunity to continue increased from 33% to 48%. This shows that

they were more confident in their knowledge and almost half of them aimed for a higher mark instead of just around a third of them as the year before.

- **Improvement of the average mark at the oral exam.** The oral exam requires students to demonstrate their detailed knowledge and understanding of a specific part of the subject. This year they performed much better when asked to explain their assigned topic, which further confirms that they have internalized it more. The average mark of those who took the option of the oral examination raised from 3.9 to 4.2.

3.7.2 Subjective results

From the perspective of the teachers, the improvement was apparent when looking at the fluency of the exams and the confidence of the students. They seemed to understand the concepts, the steps and the general structure of proofs better as well. The improvement seen numerically in the results was perceivable subjectively too.

Positive feedback has also arrived from the students. Thanks to the interactive methods in the practical sessions they approached the subject with more enthusiasm. In the long term, this approach seems to be capable of delivering usable knowledge to average students who would otherwise not be interested enough to invest time in understanding hard mathematical concepts on their own.

4. The Type Systems for Programming Languages course

4.1 Aim and main principles of the course

In the 2019 Autumn semester the Coq proof assistant was also utilised for teaching practical sessions in another course titled *Type Systems for Programming Languages*. This is a third semester master's course which similarly to Formal Semantics, did not have practical sessions before. The structure and grading of these classes were very similar to the format of Formal Semantics outlined above. There was a small assignment every week that helped practice the new language constructs, notions and concepts introduced during the preceding session. These were followed by a short test at the beginning of every session that was similar to that week's homework, so that those who spent time on it at home had a significant advantage and thus were motivated to do so. In later phases of the course the students were encouraged to prepare their own reusable tactics at home and bring their code to the classes. This opportunity was taken only by a few of them, but those who chose to do so clearly showed that trying to look at the problem from a general perspective instead of only concentrating on solving one specific proof helped them gain a deeper understanding of the recurring patterns often used in proofs. The results of these small tests determined the final grade for the practical course. At the end of the semester every student also got a bigger homework in the form of a formalised language with a type system and had to prove a theorem for it. The languages and theorems were randomly assigned in a way that ensured a unique task for each student.

4.2 Syllabus

Instead of the imperative constructs discussed in *Formal Semantics* (assignment, branching, loops, etc.), this course uses an expression language for demonstration purposes that is closer to the functional paradigm. There is an operational semantics defined for this language, and it is extended

with typing relations. The arc of the whole semester was defined by following Parts I—VII of Harper [12] and Hungarian lecture notes based on them [13]. The first few sessions were mostly rehearsing the basics of Coq through small illustrative examples with terms and types, such as a recursively defined type checker on simple inductively defined terms, because not all students encountered the language before. After this, basic inductive proofs were introduced, and the soundness and completeness of the previously defined type inference algorithm were proven. This was followed by creating an inductive typing relation and an inductive transition relation which opened the opportunity to discuss operational semantics and type systems separately as well as their interactions. Further into the course the notion of contexts and well-formedness were presented, and multiple important theorems were proven that connected the previous concepts, such as the *Unicity of Typing*, the *Substitution Lemma*, the *Lemma of Decomposition*, the *Theorem of Determinism*, the *Theorem of Progress* and the *Theorem of Type Preservation*. The following snippets illustrate a few smaller pieces of code, as the full sources would be unsuitable for incorporation into the article because of their lengths.

Partial definition of the typing relation:

```
Inductive TypeJudgement : Con -> Tm -> Ty -> Prop :=
[...]
  | TJ_plus {G : Con} {t t' : Tm} : a
      (G |- t : Nat) -> (G |- t' : Nat) ->
      (G |- (t + t') : Nat)
[...]
where "G |- tm : ty" := (TypeJudgement G tm ty).
```

$$\frac{\Gamma \vdash t : \mathrm{Nat} \quad \Gamma \vdash t' : \mathrm{Nat}}{\Gamma \vdash t + t' : \mathrm{Nat}}$$

Figure 2: Coq formalization (left) and traditional notation (right) of an inductive type judgment constructor definition stating that if two terms (`t` and `t'`) can both be typed as `Nat` in a certain context, their sum can be proven to be of type `Nat` as well in the same context.

Partial definition of the small-step transition judgment:

```
Inductive OneStepTransitionJudgement : Tm -> Tm -> Prop :=
| OSTJ_sum {n1 n2 n : nat} :
    ((n1 + n2)%nat = n) ->
    num n1 + num n2 |--> num n
[...]
| OSTJ_plus_left {t1 t1' t2 : Tm} :
    (t1 |--> t1') ->
    t1 + t2 |--> t1' + t2

| OSTJ_plus_right {t1 t2 t2' : Tm} :
    t1 val -> (t2 |--> t2') ->
    t1 + t2 |--> t1 + t2'
[...]
where "t |--> t'" := (OneStepTransitionJudgement t t').
```

$$\frac{n_1 + n_2 = n}{num\ n_1 + num\ n_2 \longmapsto \mathrm{num}\ n}$$

$$\frac{t_1 \longmapsto t_1'}{t_1 + t_2 \longmapsto t_1' + t_2}$$

$$\frac{t_1\ val \quad t_2 \longmapsto t_2'}{t_1 + t_2 \longmapsto t_1 + t_2'}$$

Figure 3: Coq formalization (left) and traditional notation (right) of an inductive small-step operational semantics definition concerning the evaluation of the addition of two terms.

Partial proof for the Theorem of Progress:

```
Theorem progress {t : Tm} {A : Ty} :
  ( * |- t : A ) ->
  t val \/ (exists (t' : Tm), t |--> t').
Proof.
  intros. remember * as G. induction H.
[...]
  - destruct (IHTypeJudgement1 HeqG).
    + destruct (IHTypeJudgement2 HeqG).
      * pose (n1 := progress_helper_Nat H H1).
        inversion n1. rewrite H3.

        pose (n2 := progress_helper_Nat H0 H2).
        inversion n2. rewrite H4.

        right. eexists.
        refine (OSTJ_sum _). reflexivity.

      * inversion H2. right. eexists.
        exact (OSTJ_plus_right H1 H3).

    + inversion H1. right. eexists.
      exact (OSTJ_plus_left H2).
[...]
Qed.
```

Induction is initiated on `H`, the hypothesis claiming that the term `t` has type `A` in the empty context.

In case the typing judgment was constructed using `TJ_plus` (see Figure 1.) the inductive hypotheses state that the theorem holds for the left and right operands separately. There are two branches based on the first hypothesis:

- If `t` is already a value we further split the proof based on the second hypothesis:
  - `t'` is already a value as well, in which case they must both be in the form of `num n`, which means that the sum rule can be applied
  - `t'` can be rewritten, which means that the rule that rewrites the right-hand side can be applied
- In case `t` can be further rewritten we simply apply the rule that rewrites the left-hand side of the addition

Figure 4: Coq formalization (left) and explanation (right) of a branch from the inductive proof of the *Theorem of Progress*. It states that if a term can be properly typed in a certain context, then it is either already a value, or can be further evaluated using the small-step operational semantics.

4.3 Results

This way of incorporating computers into the teaching had some advantages, like the ability to automatically grade assignments by just type checking them, but also had brought some disadvantages, such as the practical lesson being a lot slower and after a while lagging behind the lecture because of the strict nature of the interaction with a proof assistant that did not allow the students to progress further without getting all the details right. To alleviate this, the skeletons of files (that contained most of the definitions and theorems) were prepared for every class and the students only had to fill in the missing parts of the definitions and proofs themselves. Because of this some students were able to advance more quickly and get through the file before the end of the class. Creating some harder optional tasks for further exercise at the end of the files was a great success and resulted in the retention of the attention of the faster students as well.

The new practical sessions were introduced in the 2019 Autumn semester. Comparing the results of the 2018 Autumn (35 students) and 2019 Autumn (48 students) exams (see Figure 1), we observe that the average went up from 2.8 to 3.7. The style and difficulty of the exams were the same in the compared semesters (written exam in the exam period).

5. Coq in other courses

Apart from the already mentioned two courses, there are several other subjects that could potentially benefit from the introduction of computer-based proof verification in the topic of programming theory, logic and mathematics. In the last semester work on a formalisation of the syllabus of *Distributed Systems* begun by a student [8]. In the long term it seems convincing that the introduction of proof assistants spanning across several subjects could greatly improve the proficiency of the students in several fields and thus increase the quality of the Computer Science program. Other courses which might apply proof assistants include *Logics* and *Computability theory*.

6. The challenges of using theorem provers

Among the desirable traits of the proof assistants that facilitate the learning process there happen to be some that also pose pedagogical challenges. Some of these arise because of design choices made during the development of Coq, while some others are consequences of one fundamental difference between proofs written on paper and those that can be verified by computers.

For example, while functional programming is now a compulsory part of the bachelor's program, the syntax of Coq is still strange for most of the students, as it follows the OCaml style [16] with which most of them are not familiar. Showing them the same definition in multiple languages – in object-oriented alternatives as well – can help in overcoming this obstacle.

The aforementioned difference from the usual blackboard reasonings they are used to is that a computer can only check the correctness of a proof if every little technical detail is given for all the branches created by different cases that need to be dealt with, lots of which are often omitted in lectures and textbooks because they can easily be seen intuitively. Luckily this can be simplified by creating reusable tactics that are generic enough to cover the repetitive sections of proofs, thus reducing the verbosity and increasing the similarity to their respective paper-based counterparts.

7. Summary

In this article we showed how interactive practical sessions using computer proof assistants were launched for courses on programming language theory that had only been taught using pen and paper methods before. The motivation behind this was to support the abstract concepts in these courses with tangible examples that help understanding the material and lead to building skills in applying formal methods. Several interactive theorem provers were considered which support executable and verifiable definitions, but Coq proved to be outstanding with its separated specification and proof languages, expressive type system as well as the abundantly available literature.

We have identified guiding principles in applying interactive theorem proving in our master's classes, such as "First understand, then code", "Graduality" and "Continuous work and short tests". These ensure that the programming language theory classes are adequately augmented by the practical sessions and the students gain a better understanding of nontrivial concepts such as deduction, induction or compositionality. Introducing the new system naturally came with technical and pedagogical challenges. Students needed to familiarise themselves with machine-assisted theorem proving and the programming language of Coq, but this initial investment payed off by the end of the semester.

The effectiveness of the lab sessions was evaluated based on an objective comparison between the exam results of the old and the new system. Significant improvements were observed in both courses: the average grades improved by 0.54 and 0.9 grades in the two courses respectively (on a scale from 1 to 5). In one of the courses, the participation rate at the oral exam also increased. Encouraged by these achievements there are plans for introducing computer proof assistants in other courses.

The high-level precise modelling of programming languages is not only an essential part of proving program correctness but also helps Computer Science students gain proficiency in programming languages in general and improve their abstraction skills. Hopefully, the experiences shared in this paper can provide useful advice on how to apply proof assistants in theoretical courses, making them more accessible to students.

Bibliography

1. The Agda programming language (from 2007)
   https://wiki.portal.chalmers.se/agda/pmwiki.php
2. Tobias Nipkow, Gerwin Klein: *Concrete Semantics with Isabelle/HOL* (from 2014)
   http://concrete-semantics.org/
3. Coq In The Classroom (from 2017)
   https://github.com/coq/coq/wiki/CoqInTheClassroom
4. GitHub repository of the Coq proof assistant (from 2007)
   https://github.com/coq/coq
5. The Coq proof assistant (from 1989)
   https://coq.inria.fr/
6. The Haskell programming language (from 1990)
   https://www.haskell.org/
7. The Isabelle proof assistant (from 1986)
   https://isabelle.in.tum.de/
8. Donkó István: orsi-formalization. Student formalization of material from the Distributed Systems course at ELTE. (2019)
   https://github.com/Isti115/orsi-formalization
9. Benjamin C. Pierce et al.: *Programming Language Foundations*. Software Foundations series, volume 2. Electronic book. (2018)
   http://www.cis.upenn.edu/~bcpierce/sf
10. Philip Wadler: *Programming Language Foundations in Agda*. Conference material, 21st Brazilian Symposium (SBMF 2018). Salvador, Brazil, November 26–30, 2018.
11. O. Johansson: *Programming language semantics*. Course materials. Umeåi University, Umeå, Sweden (1993-1998)
    http://oldwww.cs.umu.se/local/kurser/TDBC05/
12. Robert Harper. 2012. *Practical Foundations for Programming Languages*. Cambridge University Press, USA.

13. Ambrus Kaposi. 2019. Nyelvek típusrendszere. Lecture notes https://bitbucket.org/akaposi/tipusrendszerek/src/master/

14. Per Martin-Löf. 1982. *Constructive Mathematics and Computer Programming.* In: Editor(s): L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, Klaus-Peter Podewski. Studies in Logic and the Foundations of Mathematics, Elsevier, Volume 104, 1982, Pages 153-175.

15. Philip Wadler. 2015. *Propositions as types. Commun. ACM* 58, 12 (November 2015), 75–84. DOI: https://doi.org/10.1145/2699407

16. Xavier Leroy & Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy and Jérôme Vouillon. 2020. *The OCaml system release 4.10: Documentation and user's manual.*

Authors

BERECZKY Péter

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary, e-mail: berpeti@inf.elte.hu

DONKÓ István

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary, e-mail: isti115@inf.elte.hu

HORPÁCSI Dániel

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary, e-mail: daniel-h@elte.hu

KAPOSI Ambrus

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary, e-mail: akaposi@inf.elte.hu

NÉMETH Dávid János

Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary, e-mail: ndj@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 1. 2020.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.1.470

License

Overview of Repetition

MENYHÁRT László Gábor
(ORCID: 0000-0002-1574-4454)
ELTE Eötvös Loránd University, Budapest, Hungary
Faculty of Informatics, 3in Research Group, Martonvásár, Hungary

Abstract. There are differences between the everyday concept of repetition, the usage in informatics and the different implementations. This article collects the different levels, paradigms and language options and it presents a performed measurement and finally it evaluates the results didactically. So, it is useful for students preparing for a programming competition and their informatics teachers.

Keywords: repetition, recursion, iteration, programming languages, benchmark

1. Introduction

In the ordinary sense, repetition is a natural phenomenon: Think of regular alternations of days, weeks, seasons, etc. Nevertheless, it appears in IT in several ways. We can think differently, they occur differently in paradigms, they have different syntax and implementation in programming languages, so they run is different, so their run time can be very different. These observations and impressions have in-spired me to gather current opportunities and compare them.

This article reviews the conceptual definitions, measurement of running different implementations and didactic examination of the results. The presented method and the current results can be useful for students preparing for a programming competition and their informatics teachers.

2. The concept of repetition

Repetition is completion or execution of something several times. I collected some definition from different places, even though everyone knows or feels what it means.

Hungarian Synonym dictionary (Magyar Szinonima kéziszótár) contains the words: again (ismét) and repeat (ismétel):

„***ismét*** *újra, megint | szintén, ismételten*" [1/140. o.]

„***ismétel*** *mondogat, hajtogat, csépel| szajkóz | megismétel, elismétel, visszamond| reprodukál, megrepetál | próbál, gyakorol | folytat*" [1/140. o.]

In New Hungarian Dictionary (Új Magyar Lexikon) the definition of repetition is this: „***ismétlés*** *:* ***1.*** *(nev) oktatási eljárás ; … Osztály~: …* ***2.*** *(nyelvt) … A magyarban jelölheti a cselekmény hosszan tartó voltát (pl. ment-ment) …* ***3.*** *a stilisztikában …*" [2/448. o.]

The Hungarian ethnographic lexicon (Magyar néprajzi lexikon) [3] provides only a linguistic definition to emphasize something to say as a poetic expression.

„***ismétlés*** *(fn.) … Cselekvés, melynél fogva valamit ismét, azaz újra, még egyszer teszünk. …*" can be found in [4].

Sulinet knowledgebase at Informatics subject contains definition of repetition as 'multiple process' („Többszöri végrehajtás"). [5]

In WikiSzótár [6] one of the definitions of repetition is 'Redoing an action when we do the same thing we did before.' („**2.** Egy **cselekvés újra végzése**, amikor ugyanazt tesszük, amit korábban már megtettünk.").

In Cambridge dictionary **repeat** is defined as „*to happen, or to do something, more than once*" [7], while **iteration** is defined as „*the process of doing something again and again, …"* [8].

3. Repetition on levels of informatics

When we have a specific problem to solve, we just know that something will have to be done several times. At the next abstract level of our thinking, we decide whether an operation that we would like to perform several times will be a function and it calls itself, which is recursion [10], or will be repeated cyclically. This distinction is already present as a paradigm, since functional programming provides only the former.

Running of loops can be distinguished as conditional and specified number of times.

There are two groups at this specified number of times execution. First has a loop variable as a counter which changes at repetition, while the other has a given variable with the item from the series. With the first version to get the items from a series is possible with a counter which will be an index of the elements, the second version gives back immediately a given element from the series to process.

Conditional loops can be divided into two groups, first version checks the conditional at first, second checks it at the end. The difference is that the statements will be processed at least once when the checking is at last, while when the checking is at the beginning it possible that the statements are not processed. Both versions' condition is a logical expression which defines whether the statements must be processed (again) or not. Loops can be grouped by happening based on the condition. Namely whether the execution will happen or not at true expression. So, it can define the condition of staying inside or exit.

<table>
<tr><th colspan="6">Repetition</th></tr>
<tr><td rowspan="4">Recursion</td><td colspan="5">Iteration</td></tr>
<tr><td colspan="4">Conditional</td><td colspan="2">Running a certain number of items</td></tr>
<tr><td colspan="2">Check first</td><td colspan="2">Check last</td><td rowspan="2">Counter</td><td rowspan="2">Iteration on elements of series</td></tr>
<tr><td>staying inside</td><td>exit cond.</td><td>staying inside</td><td>exit cond.</td></tr>
</table>

Table 1.: Overview table of repetition

Syntax can modify this clear picture, because most programming languages do not support any of the above options. For example, exit condition must be defined in Pascal for the conditional loop where the condition will be checked at last, but in C-like languages staying inside condition must be defined. Usually staying inside condition must be defined in those conditional loops where the condition will be checked at first, but shell script in Linux there is syntax for defining exit condition. Indeed, all four types of condition must be defined. In C the `for` keyword can be used as loop which checks the condition at first, and there are `break` and `continue` keywords to end earlier the `for` loop which runs theoretically a certain number of items. Of course, with multiple nested `for` loops you have to use tricks to, so you should avoid this option. There is programming language

(like Logo), where loop-variable is not present generally, it can be used only if it required (in REPEAT the variable REPCOUNT).

Not only keywords and syntax can be different in high-level programming languages, but the compiler can change the actually running byte code and its implementation. So, their runtime can be different. I have already implemented the solution of the same tasks in different languages [11, 12, 13, 14, 15, 16] and the runtime was diverse, it occurred to me to make a more accurate measurement.

4. Description of measurement

I wrote code for five-six different loop implementation in six programming languages. Then I collected the measured run times in milliseconds.

Programming languages were C++, C#, Java, JavaScript, Pascal and Python.

I implemented recursion, loops with checking conditional at first, checking it at last, iteration with loop variables as counter and as item if the language supported. It was skipped in case of not supported. Sometimes iteration with items can be implemented in more ways, then these were merged into a group.

4.1. Presentation of the environment

The measurement ran on a computer with Intel® Core™ i7-8750H 2.20GHz processor and 32GB memory, which has 64 bit Windows 10 Pro operating system.

The C++ compiler was mingw32-g++.exe with version 5.1.0 which is default in the Code::Blocks 17.12.

I used .NER Framework's compiler for C# found on Windows 10, it is „Microsoft (R) Visual C# Compiler version 4.8.3752.0".

Java compiler was the currently most recent OpenJDK published on 6th of October 2019.

```
>java -version
openjdk version "13.0.1" 2019-10-15
OpenJDK Runtime Environment (build 13.0.1+9)
OpenJDK 64-Bit Server VM (build 13.0.1+9, mixed mode, sharing)
```

JavaScript codes ran on Node v10.16.0.

„Free Pascal Compiler version 3.0.4 [2017/10/06] for i386" was used for Pascal 32 bit.

Interpreter for Python codes was „Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:30:26) [MSC v.1500 64 bit (AMD64)]".

4.2. Presentation of implemented algorithm

Repetition was implemented in all six programming languages and possible syntaxes based on the next algorithm.

As found, number of steps is limited in recursion in most programming languages, at first the maximum number of steps was determined. Accurate values are in paragraph 4.3 in Table 2. This

maximum value is not enough to measure all methods because it is too fast. So, codes must be run multiple times, (`for`) loop was chosen for this.

Elapsed time was calculated from the difference of start and end timestamp. Empty loop, namely loop without statements are not useable, so a simple assignment was present. But I was interested only in elapsed time of the loop, so I had to remove the time of the assignment. So, I implemented the same with two assignments as well. Abstract algorithm:

```
time1
loop
  assignment1
time2
loop
  assignment1
  assignment2
time3
```

I calculated the total time devoted to an assigment (1) and I assumed that time of assignments are equals (2). I calculated (3) the specific time of the given loop:

$$T_{a2} = (t_3 - t_2) - (t_2 - t_1) \quad (1)$$

$$T_{a1} = T_{a2} \quad (2)$$

$$T_l = t_2 - t_1 - T_{a1} \quad (3)$$

Implemented codes can be found in Appendix A.

4.3. Results of measurement

At first the maximum number of steps was determined based on the implementation of recursion which is good in all programming languages. The recursive function calls happen at the beginning of the function to avoid the optimization of tail-recursion.

Programming language	**MaxCnt (In this environment, with these program codes)**
C++	56988
C#	15918
Java	11420
JavaScript	8940
Pascal	65134
Python	4194

Table 2.: Maximum number of recursive function calls

Finally, 4000 repetition happened 10.000 times in an iteration with simple for syntax to get measurable and comparable results. Later it was detected that it was good choice because this is the fastest and that is why it does not cause confusion in comparison. So double times 40.000.000 loops were measured.

4.4. Evaluation of measurement

Appendix B. contains raw data of measurement from which average and relative deviation was determined with Excel. The time (T_i) in milliseconds concern to the first 40 million loops. Deviation is given in relative to average in percentage, so it is easier to understand and compare. Next table contains the results:

Programing language	While	For	ForEach [+lambda]	ForEach +named fnc	Recursion
C++	7 (127%)	6 (113,9%)	1142 (8,8%)	1141 (12,7%)	135 (17,7%)
C#	20 (46,8%)	24 (55,1%)	117 (43%)	113 (69,2%)	90 (10,2%)
Java	54 (42,2%)	50 (21,6%)	234 (22%)* 170 (18,7%)	2020 (12,4%)	109 (15,9%)
JavaScript	36 (37,5%)	41 (68,1%)	254 (18,6%)	242 (14,7%)	267 (9,2%)
Pascal	65 (10,1%)	89 (36,5%)	565 (8,6)*	notSupported	138 (11,3%)
Python	2054 (28,8%)	1156 (55,3%)	460 (95,4%)*	3589 (20,8%)	9876 (15,4%)

Table 3.: Results of measurement: average (deviation relative to the average in %)

4.5. Opinions of the results

You can see in the table that the specified number of steps with loop variable (for) and the conditional loop with checking at first (`while`) are the fastest repetition methods only there is deviation in Python. Recursion is the next which is slower double times in Pascal and Java, four-five times slower in C#, JavaScript and Python, and twenty times slower in C++, as the averages of previous two. Previous average times are fraction of a second but in Python it is around 10 seconds. Similar time was measured for iteration over the items of series with C# and JavaScript to the time of recursion calls, but it is nine times slower in C++, and two times or eighteen times slower in Java depends on usage of anonymous or named functions at iteration. While it is "only" three times slower in Python compared to for loop, but it is faster than recursion. For loop in C++ is the fastest, and recursion is Python is the slowest.

5. Didactic consideration

If I had to choose a programming language for education based on the table above, I would choose C#, because the measured times are consistent. Pascal does not support every method, so it should be skipped. Python is too slow. So, my order would be C#, JavaScript, C++ and Java based on the average runtime and its deviation.

Of course, all methods must be taught to the students and should be shown how they can choose between paradigms and syntaxes.

There are cases where the readability of the code is more important, but sometimes fast runtime is necessary. For instance, it was determined [9], that recursion calls usually less effective then iterations, but we often get a simpler, easier-to-read code. Such a comparative table can also be useful in programming competitions where time limits are used, and the competitor can choose

the language. However, the statement is not only an assignment, and other language elements can affect the total runtime.

The knowledge and measurement results from this article can help you to choose the right implementation for a given task. The described technique can be helpful later if Benchmark [17] would be performed again because of processor development and appearance of newer compilers may change the times described here. Because newer compiler versions can analyse the codes and create more efficient binary codes with varying changes of codes. Like Python has version 3.8, and there are alternate implementations [18, 19] as well, which would be worth comparing.

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

References

1. Bék Gerzson, Csifféry Tamás: *Magyar szinonima kéziszótár*, Könyvmíves Könyvkiadó, Budapest (2004)
2. Akadémiai kiadó szerkesztősége: *Új magyar lexikon*, 3 G-J, Nyolcadik, változatlan lenyomat, Akadémiai Kiadó, Budapest (1962)
3. *Magyar néprajzi lexikon*, Akadémiai Kiadó, Budapest (1977-1982)
4. Czuczor Gergely, Fogarasi János: *A magyar nyelv szótára*, Emich Gusztáv at Hungarian academic printer Pest (1862) https://czuczor.oszk.hu/kereses.php?kereses=ism%C3%A9tl%C3%A9s (last viewed: 2019.11.11.) and https://mek.oszk.hu/05800/05887/pdf/3kotet_1.pdf (page 74., last viewed: 2019.11.11.)
5. *The concept of repetition in the Sulinet knowledgebase* (2016) https://tudasbazis.sulinet.hu/hu/informatika/informatika/informatika-2-evfolyam/algoritmusok-a-hetkoznapokban-tevekenysegek-elemekre-bontasa-helyes-sorrend-megkeresese/ismetles-szerepe (last viewed: 2019.11.11.)
6. *WikiSzótár* Dictionary (2012) https://wikiszotar.hu/ertelmezo-szotar/Ism%C3%A9tl%C3%A9s (last viewed: 2019.11.11.)
7. *Repeat in Cambridge Dictionary* (2019) https://dictionary.cambridge.org/dictionary/english/repeat (last viewed: 2019.11.11.)
8. *Iteration in Cambridge Dictionary* (2019) https://dictionary.cambridge.org/dictionary/english/iteration (last viewed: 2019.11.11.)
9. Rónyai L., Ivanyos G., Szabó R.: *Algoritmusok*, Typotex (2004) page 37.
10. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest.: *Introduction to Algorithms*, MIT Press (1990) (Hungarian version: Algoritmusok, Műszaki Könyvkiadó 1997)
11. *C++ reference* (2019) https://en.cppreference.com/w/ (last viewed: 2019.11.11.)
12. *C# reference* (2017) https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/ (last viewed: 2019.11.11.)

13. *Java documentation* (2019) https://docs.oracle.com/javase/tutorial/java/ (last viewed: 2019.11.11.)

14. *JavaScript reference* (2019) https://developer.mozilla.org/hu/docs/Web/JavaScript/Reference (last viewed: 2019.11.11.)

15. *Free Pascal reference guide* (2017) https://www.freepascal.org/docs-html/ref/ref.html (last viewed: 2019.11.11.)

16. *The Python Language Reference* (2019) https://docs.python.org/2.7/reference/index.html (last viewed: 2019.11.11.)

17. *Meaning of Beenhmark* https://dictionary.cambridge.org/dictionary/english/benchmark (last viewed: 2019.12.26.)

18. *Alternative Python Implementations* https://www.python.org/download/alternatives/ (last viewed: 2019.11.11.)

19. *Cython* https://cython.org/ (last viewed: 2019.11.11.)

Appendix

A. Source codes

A.1. C++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <array>
#include <chrono>

using namespace std;

int a;
int b;

void recursiveFor(int &i, const int &N, std::array<int,65135> &v) {
    if (i<N) {
        i++;
        recursiveFor(i,N,v);
        a=v[i-1];
    }
}

void recursiveFor2(int &i, const int &N, std::array<int,65135> &v) {
    if (i<N) {
        i++;
        recursiveFor2(i,N,v);
        a=v[i-1];
        b=v[i-1];
    }
}

int main()
{
```

```
    auto timer_start= std::chrono::high_resolution_clock::now();
    auto timer_mid= std::chrono::high_resolution_clock::now();
    auto timer_end= std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed;
    std::chrono::duration<double> elapsed2;

    const int MaxN=65135;
    int N=4000;
    int times=10000;

    std::array<int,MaxN> v;
    for (int i=0; i<N; i++)
    {
        v[i];
    }

    auto func=[](const int&x)
    {
        a=x;
    };

     auto func2=[](const int&x)
    {
        a=x;
        b=x;
    };

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            i++;
        }
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            b=v[i];
            i++;
        }
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;while;cnt:"<< times*N <<";lt:" << elapsed.count()*1000-
(elapsed2.count()*1000-elapsed.count()*1000) << "\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        do
        {
            a=v[i];
            i++;
        } while (i<=N);
    }
```

```
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        do
        {
            a=v[i];
            b=v[i];
            i++;
        } while (i<=N);
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;do-while;cnt:"<< times*N <<";lt:" << elapsed.count()*1000-
(elapsed2.count()*1000-elapsed.count()*1000) << "\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
        }
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
            b=v[i];
        }
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;for;cnt:"<< times*N <<";lt:" << elapsed.count()*1000-
(elapsed2.count()*1000-elapsed.count()*1000) << "\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(),
                      [](const int&x)
        {
            a=x;
        });
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(),
                      [](const int&x)
        {
            a=x;
            b=x;
        });
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;ForEach + lambda;cnt:"<< times*N <<";lt:" <<
elapsed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) << "\n";
```

```
    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(), func);
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        std::for_each(v.begin(), v.end(), func2);
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;ForEach + named function;cnt:"<< times*N <<";lt:" <<
elapsed.count()*1000-(elapsed2.count()*1000-elapsed.count()*1000) << "\n";

    timer_start = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor(i,N,v);
    }
    timer_mid = std::chrono::high_resolution_clock::now();
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor2(i,N,v);
    }
    timer_end = std::chrono::high_resolution_clock::now();
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    std::cout << "Cpp;Recursive;cnt:"<< times*N <<";lt:" << elapsed.count()*1000-
(elapsed2.count()*1000-elapsed.count()*1000) << "\n";

    return 0;
}
```

A.2. C#

```
using System;

public class Program
{
    static int a;
    static int b;

    static void recursiveFor(int i, int N, int[] v) {
        if (i<N) {
            i++;
            recursiveFor(i,N,v);
            a=v[i-1];
        }
    }

    static void recursiveFor2(int i, int N, int[] v) {
        if (i<N) {
            i++;
            recursiveFor2(i,N,v);
            a=v[i-1];
            b=v[i-1];
        }
```

```
    }

    public static void Main()
    {
    DateTime timer_start= DateTime.Now;
    DateTime timer_mid= DateTime.Now;
    DateTime timer_end;
    TimeSpan elapsed;
    TimeSpan elapsed2;

    int N=4000;
    int times=10000;

    int[] v=new int[N];
    for (int i=0; i<N; i++)
    {
        v[i]=i;
    }

    Action<int> func = x => {
        a=x;
    };
    Action<int> func2 = x => {
        a=x;
        b=x;
    };

    timer_start= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
        }
    }
    timer_mid= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
            b=v[i];
        }
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    Console.WriteLine("cs;for;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);

    timer_start= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            i++;
        }
    }
    timer_mid= DateTime.Now;
    for (int j=0; j<times; j++)
    {
        int i=0;
```

```
            while ( i<N)
            {
                a=v[i];
                b=v[i];
                i++;
            }
        }
        timer_end = DateTime.Now;
        elapsed = timer_mid - timer_start;
        elapsed2 = timer_end - timer_mid;
        Console.WriteLine("cs;while;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);

        timer_start= DateTime.Now;
        for (int j=0; j<times; j++)
        {
            Array.ForEach(v, (int x) =>
            {
                a=x;
            });
        }
        timer_mid= DateTime.Now;
        for (int j=0; j<times; j++)
        {
            Array.ForEach(v, (int x) =>
            {
                a=x;
                b=x;
            });
        }
        timer_end = DateTime.Now;
        elapsed = timer_mid - timer_start;
        elapsed2 = timer_end - timer_mid;
        Console.WriteLine("cs;ForEach + lambda;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);

        timer_start= DateTime.Now;
        for (int j=0; j<times; j++)
        {
            Array.ForEach(v,func);
        }
        timer_mid= DateTime.Now;
        for (int j=0; j<times; j++)
        {
            Array.ForEach(v,func2);
        }
        timer_end = DateTime.Now;
        elapsed = timer_mid - timer_start;
        elapsed2 = timer_end - timer_mid;
        Console.WriteLine("cs;ForEach + named
function;cnt:{1};lt:{0}",elapsed.Milliseconds-(elapsed2.Milliseconds-
elapsed.Milliseconds),N*times);

        timer_start= DateTime.Now;
        for (int j=0; j<times; j++)
        {
            int i=0;
            recursiveFor(i,N,v);
        }
        timer_mid= DateTime.Now;
        for (int j=0; j<times; j++)
        {
            int i=0;
            recursiveFor2(i,N,v);
```

```
    }
    timer_end = DateTime.Now;
    elapsed = timer_mid - timer_start;
    elapsed2 = timer_end - timer_mid;
    Console.WriteLine("cs;Recursive;cnt:{1};lt:{0}",elapsed.Milliseconds-
(elapsed2.Milliseconds-elapsed.Milliseconds),N*times);
}
}
```

A.3. Java

```
import java.util.Date;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.function.Consumer;

public class ciklus{

    static Integer a;
    static Integer b;

    public static void recursiveFor(int i, int N, Integer[] v) {
        if (i<N) {
            i++;
            recursiveFor(i,N,v);
            a=v[i-1];
        }
    }

    public static void recursiveFor2(int i, int N, Integer[] v) {
        if (i<N) {
            i++;
            recursiveFor2(i,N,v);
            a=v[i-1];
            b=v[i-1];
        }
    }

    public static void main(String[] args) {
        Date timer_start=new Date();
        Date timer_mid=new Date();
        Date timer_end;
        long elapsed;
        long elapsed2;

        int MaxN=65135;//100000000;
        int N=4000;//100000000;
        int times=10000; //30;

    Integer[] v=new Integer[MaxN];
        for (int i=0; i<N; i++)
        {
            v[i]=i;
        }

        Consumer<Integer> func = (x) ->
        {
            a=x;
        };
        Consumer<Integer> func2 = (x) ->
        {
```

```
            a=x;
            b=x;
        };

    timer_start = new Date();
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
        }
    }
    timer_mid = new Date();
    for (int j=0; j<times; j++)
    {
        for (int i=0; i<N; i++)
        {
            a=v[i];
            b=v[i];
        }
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;for;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-
elapsed)));

    timer_start = new Date();
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            i++;
        }
    }
    timer_mid = new Date();
    for (int j=0; j<times; j++)
    {
        int i=0;
        while (i<N)
        {
            a=v[i];
            b=v[i];
            i++;
        }
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;while;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-
elapsed)));

    timer_start = new Date();
    for (int j=0; j<times; j++)
    {
        for (Integer x : v)
        {
            a=x;
        }
    }
    timer_mid = new Date();
    for (int j=0; j<times; j++)
```

```
    {
        for (Integer x : v)
        {
            a=x;
            b=x;
        }
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;forEach;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-
elapsed)));

    List<Integer> vl=Arrays.asList(v);
    timer_start =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach((x)-> {
            a=x;
        });
    }
    timer_mid =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach(x-> {
            a=x;
            b=x;
        });
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;ForEach + lambda;cnt:"+times*N+";lt:"+(elapsed-
(elapsed2-elapsed)));

    timer_start =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach(func);
    }
       timer_mid =new Date();
    for (int j=0; j<times; j++)
    {
        vl.forEach(func2);
    }
    timer_end = new Date();
    elapsed = timer_mid.getTime() - timer_start.getTime();
    elapsed2 = timer_end.getTime() - timer_mid.getTime();
    System.out.println("Java;ForEach + named
function;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-elapsed)));

    timer_start = new Date();
    try {
    for (int j=0; j<times; j++)
    {
        int i=0;
        recursiveFor(i,N,v);
    }
    } catch(Exception ex) {
    }
    timer_mid = new Date();
    try {
    for (int j=0; j<times; j++)
    {
```

```
            int i=0;
            recursiveFor2(i,N,v);
        }
        } catch(Exception ex) {
        }
        timer_end = new Date();
        elapsed = timer_mid.getTime() - timer_start.getTime();
        elapsed2 = timer_end.getTime() - timer_mid.getTime();
        System.out.println("Java;Recursive;cnt:"+times*N+";lt:"+(elapsed-(elapsed2-
elapsed)));
        }
}
```

A.4. JavaScript

```
let N=4000;
let times=10000;
let a;
let b;
let v=[];
for (let i=0; i<N; i++)
{
    v.push(i);
}

function recursiveFor(i, N, v) {
    if (i<N) {
        i++;
        recursiveFor(i,N,v);
        a=v[i-1];
    }
}

function recursiveFor2(i, N, v) {
    if (i<N) {
        i++;
        recursiveFor2(i,N,v);
        a=v[i-1];
        b=v[i-1];
    }
}

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    for (let i=0; i<N; i++)
    {
        a=i;
    }
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    for (let i=0; i<N; i++)
    {
        a=i;
        b=i;
    }
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
```

```
console.log("Js;for;cnt:",(times*N),";lt:",elapsed-(elapsed2-elapsed));

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    while (i<N)
    {
        a=i;
        i++;
    }
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    while (i<N)
    {
        a=i;
        b=i;
        i++;
    }
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;While;cnt:",(times*N),";lt:",elapsed-(elapsed2-elapsed));

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach((item, index, arr)=>{
        a=item;
    });
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach((item, index, arr)=>{
        a=item;
        b=item;
    });
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;ForEach + lambda function;cnt:",(times*N),";lt:",elapsed-
(elapsed2-elapsed));

function func(item, index, arr) {
  a=item;
}
function func2(item, index, arr) {
  a=item;
  b=item;
}
var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    v.forEach(func);
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
```

```
    v.forEach(func2);
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;ForEach + named function;cnt:",(times*N),";lt:",elapsed-
(elapsed2-elapsed));

var timer_start = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    recursiveFor(i,N,v);
}
var timer_mid = Date.now();
for (let j=0; j<times; j++)
{
    let i=0;
    recursiveFor2(i,N,v);
}
var timer_end = Date.now();
elapsed=timer_mid-timer_start;
elapsed2=timer_end-timer_mid;
console.log("Js;Recursive;cnt:",(times*N),";lt:",elapsed-(elapsed2-elapsed));
```

A.5. Pascal

```
Program ciklus;

Uses sysutils;

  var
    timer_start:TDateTime;
    timer_mid:TDateTime;
    timer_end:TDateTime;
    elapsed:TDateTime;
    elapsed2:TDateTime;
    N:integer;
    times:integer;
    v:array[1..65135] of integer;
    i:integer;
    j:integer;
    a:integer;
    b:integer;
    w:integer;

procedure recursiveFor(var i:integer; const N:integer; var v:array of integer);
begin
    if (i<=N) then begin
        i:=i+1;
        recursiveFor(i,N,v);
        a:=v[i-1];
    end
end;

procedure recursiveFor2(var i:integer; const N:integer; var v:array of integer);
begin
    if (i<=N) then begin
        i:=i+1;
        recursiveFor2(i,N,v);
        a:=v[i-1];
        b:=v[i-1];
```

```
    end
end;

BEGIN
    N:=4000;
    times:=10000;

    for i:=1 to N do begin
        v[i]:=i;
    end;

    timer_start:=Now;
    for j:=1 to times do begin
        for i:=1 to N do begin
            a:=v[i];
        end;
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        for i:=1 to N do begin
            a:=v[i];
            b:=v[i];
        end;
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;for;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

    timer_start:=Now;
    for j:=1 to times do begin
        i:=1;
        while (i<=N) do begin
            a:=v[i];
            i:=i+1;
        end;
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        i:=1;
        while (i<=N) do begin
            a:=v[i];
            b:=v[i];
            i:=i+1;
        end;
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;while;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

    timer_start:=Now;
    for j:=1 to times do begin
        for w in v do begin
            a:=w;
        end;
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        for w in v do begin
            a:=w;
            b:=w;
```

```
        end;
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;forEach;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

    timer_start:=Now;
    for j:=1 to times do begin
    end;
    timer_end:=Now;
    elapsed:=timer_end-timer_start;
    WriteLn('Pas;ForEach + lambda;cnt:',times*N,';lt:Not supported');
    timer_start:=Now;
    for j:=1 to times do begin
    end;
    timer_end:=Now;
    elapsed:=timer_end-timer_start;
    WriteLn('Pas;ForEach + named function;cnt:',times*N,';lt:Not supported');
    timer_start:=Now;
    for j:=1 to times do begin
        i:=1;
        recursiveFor(i,N,v);
    end;
    timer_mid:=Now;
    for j:=1 to times do begin
        i:=1;
        recursiveFor2(i,N,v);
    end;
    timer_end:=Now;
    elapsed:=timer_mid-timer_start;
    elapsed2:=timer_end-timer_mid;
    WriteLn('Pas;Recursive;cnt:',times*N,';lt:',(elapsed-(elapsed2-
elapsed))*100000000:6:0);

END.
```

A.6. Python

```
import sys
from datetime import datetime

#print sys.getrecursionlimit()
sys.setrecursionlimit(100000)
#print sys.getrecursionlimit()

N=4000;
times=10000;
v=[]
for i in range(1,N):
  v.append(i);

def func(x):
  a=x

def func2(x):
  a=x
  b=x

def recursiveFor(i,N,v):
  if i<N-1:
    i=i+1
```

```
    recursiveFor(i,N,v);
    a=v[i-1]

def recursiveFor2(i,N,v):
  if i<N-1:
    i=i+1
    recursiveFor2(i,N,v);
    a=v[i-1]
    b=v[i-1]

timer_start = datetime.now()
for j in range(1,times):
  for i in range(1,N):
    a=v[i-1]
timer_mid = datetime.now()
for j in range(1,times):
  for i in range(1,N):
    a=v[i-1]
    b=v[i-1]
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print
'Py;For;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/1000)-
((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  i=0
  while i<N-1:
    #std::cout << v[i] << std::endl;
    a=v[i]
    i=i+1
timer_mid = datetime.now()
for j in range(1,times):
  i=0
  while i<N-1:
    a=v[i]
    b=v[i]
    i=i+1
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print
'Py;While;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/1000)-
((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  for x in v:
    a=x
timer_mid = datetime.now()
for j in range(1,times):
  for x in v:
    a=x
    b=x
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print 'Py;ForEach [+
lambda];cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/1000)-
```

```
((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  for x in v:
    func(x)
timer_mid = datetime.now()
for j in range(1,times):
  for x in v:
    func2(x)
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print 'Py;ForEach + named
function;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/1000)-
((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))

timer_start = datetime.now()
for j in range(1,times):
  i=0
  recursiveFor(i,N,v);
timer_mid = datetime.now()
for j in range(1,times):
  i=0
  recursiveFor2(i,N,v);
timer_end = datetime.now()
elapsed=timer_mid-timer_start
elapsed2=timer_end-timer_mid
print
'Py;recursive;cnt:',N*times,';lt:',(elapsed.seconds*1000+elapsed.microseconds/100
0)-((elapsed2.seconds*1000+elapsed2.microseconds/1000)-
(elapsed.seconds*1000+elapsed.microseconds/1000))
```

B. Raw data of measurement

```
Cpp;while;cnt:40000000;lt:0.056
Cpp;do-while;cnt:40000000;lt:0.658
Cpp;for;cnt:40000000;lt:31.361
Cpp;ForEach + lambda;cnt:40000000;lt:1076.23
Cpp;ForEach + named function;cnt:40000000;lt:1087.14
Cpp;Recursive;cnt:40000000;lt:124.673
cs;for;cnt:40000000;lt:22
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:92
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:85
Java;for;cnt:40000000;lt:56
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:213
Java;ForEach + lambda;cnt:40000000;lt:153
Java;ForEach + named function;cnt:40000000;lt:1901
Java;Recursive;cnt:40000000;lt:98
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 249
Js;ForEach + named function;cnt: 40000000 ;lt: 229
Js;Recursive;cnt: 40000000 ;lt: 263
Pas;for;cnt:40000000;lt:    97
Pas;while;cnt:40000000;lt:    58
Pas;forEach;cnt:40000000;lt:   549
```

```
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   130
Py;For;cnt: 40000000 ;lt: 1140
Py;While;cnt: 40000000 ;lt: 1872
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 265
Py;ForEach + named function;cnt: 40000000 ;lt: 3176
Py;recursive;cnt: 40000000 ;lt: 9022

Cpp;while;cnt:40000000;lt:11.553
Cpp;do-while;cnt:40000000;lt:0.589
Cpp;for;cnt:40000000;lt:22.609
Cpp;ForEach + lambda;cnt:40000000;lt:1095.92
Cpp;ForEach + named function;cnt:40000000;lt:1125.63
Cpp;Recursive;cnt:40000000;lt:110.243
cs;for;cnt:40000000;lt:31
cs;while;cnt:40000000;lt:8
cs;ForEach + lambda;cnt:40000000;lt:117
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:91
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:48
Java;forEach;cnt:40000000;lt:199
Java;ForEach + lambda;cnt:40000000;lt:157
Java;ForEach + named function;cnt:40000000;lt:1961
Java;Recursive;cnt:40000000;lt:138
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 30
Js;ForEach + lambda function;cnt: 40000000 ;lt: 235
Js;ForEach + named function;cnt: 40000000 ;lt: 227
Js;Recursive;cnt: 40000000 ;lt: 251
Pas;for;cnt:40000000;lt:   101
Pas;while;cnt:40000000;lt:    94
Pas;forEach;cnt:40000000;lt:   546
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   118
Py;For;cnt: 40000000 ;lt: 1204
Py;While;cnt: 40000000 ;lt: 2665
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 270
Py;ForEach + named function;cnt: 40000000 ;lt: 3197
Py;recursive;cnt: 40000000 ;lt: 8994

Cpp;while;cnt:40000000;lt:24.209
Cpp;do-while;cnt:40000000;lt:1.789
Cpp;for;cnt:40000000;lt:4.774
Cpp;ForEach + lambda;cnt:40000000;lt:1095
Cpp;ForEach + named function;cnt:40000000;lt:1109.37
Cpp;Recursive;cnt:40000000;lt:126.22
cs;for;cnt:40000000;lt:47
cs;while;cnt:40000000;lt:2
cs;ForEach + lambda;cnt:40000000;lt:90
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:50
Java;while;cnt:40000000;lt:66
Java;forEach;cnt:40000000;lt:228
Java;ForEach + lambda;cnt:40000000;lt:157
Java;ForEach + named function;cnt:40000000;lt:1938
Java;Recursive;cnt:40000000;lt:112
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 238
Js;ForEach + named function;cnt: 40000000 ;lt: 231
```

```
Js;Recursive;cnt: 40000000 ;lt: 251
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    62
Pas;forEach;cnt:40000000;lt:   560
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   159
Py;For;cnt: 40000000 ;lt: 1042
Py;While;cnt: 40000000 ;lt: 2833
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 235
Py;ForEach + named function;cnt: 40000000 ;lt: 3176
Py;recursive;cnt: 40000000 ;lt: 9232

Cpp;while;cnt:40000000;lt:4.446
Cpp;do-while;cnt:40000000;lt:3.817
Cpp;for;cnt:40000000;lt:0.397
Cpp;ForEach + lambda;cnt:40000000;lt:1093.52
Cpp;ForEach + named function;cnt:40000000;lt:1108.86
Cpp;Recursive;cnt:40000000;lt:127.066
cs;for;cnt:40000000;lt:13
cs;while;cnt:40000000;lt:8
cs;ForEach + lambda;cnt:40000000;lt:125
cs;ForEach + named function;cnt:40000000;lt:82
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:58
Java;while;cnt:40000000;lt:31
Java;forEach;cnt:40000000;lt:211
Java;ForEach + lambda;cnt:40000000;lt:166
Java;ForEach + named function;cnt:40000000;lt:1922
Java;Recursive;cnt:40000000;lt:113
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 247
Js;ForEach + named function;cnt: 40000000 ;lt: 231
Js;Recursive;cnt: 40000000 ;lt: 271
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   556
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1131
Py;While;cnt: 40000000 ;lt: 2009
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 277
Py;ForEach + named function;cnt: 40000000 ;lt: 2980
Py;recursive;cnt: 40000000 ;lt: 7478

Cpp;while;cnt:40000000;lt:3.886
Cpp;do-while;cnt:40000000;lt:0.068
Cpp;for;cnt:40000000;lt:2.29
Cpp;ForEach + lambda;cnt:40000000;lt:1114.82
Cpp;ForEach + named function;cnt:40000000;lt:1160.19
Cpp;Recursive;cnt:40000000;lt:134.357
cs;for;cnt:40000000;lt:20
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:91
cs;ForEach + named function;cnt:40000000;lt:93
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:65
Java;while;cnt:40000000;lt:61
Java;forEach;cnt:40000000;lt:222
Java;ForEach + lambda;cnt:40000000;lt:160
Java;ForEach + named function;cnt:40000000;lt:2018
Java;Recursive;cnt:40000000;lt:117
```

```
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 252
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 270
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   564
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   137
Py;For;cnt: 40000000 ;lt: 1190
Py;While;cnt: 40000000 ;lt: 1832
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 300
Py;ForEach + named function;cnt: 40000000 ;lt: 3291
Py;recursive;cnt: 40000000 ;lt: 9734

Cpp;while;cnt:40000000;lt:5.952
Cpp;do-while;cnt:40000000;lt:3.779
Cpp;for;cnt:40000000;lt:5.6
Cpp;ForEach + lambda;cnt:40000000;lt:1136.82
Cpp;ForEach + named function;cnt:40000000;lt:1141.19
Cpp;Recursive;cnt:40000000;lt:133.593
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:89
Java;for;cnt:40000000;lt:38
Java;while;cnt:40000000;lt:54
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:177
Java;ForEach + named function;cnt:40000000;lt:2018
Java;Recursive;cnt:40000000;lt:110
Js;for;cnt: 40000000 ;lt: 40
Js;While;cnt: 40000000 ;lt: 40
Js;ForEach + lambda function;cnt: 40000000 ;lt: 253
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 265
Pas;for;cnt:40000000;lt:    81
Pas;while;cnt:40000000;lt:    66
Pas;forEach;cnt:40000000;lt:   565
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   131
Py;For;cnt: 40000000 ;lt: 101
Py;While;cnt: 40000000 ;lt: 1431
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 1225
Py;ForEach + named function;cnt: 40000000 ;lt: 5895
Py;recursive;cnt: 40000000 ;lt: 14090

Cpp;while;cnt:40000000;lt:5.244
Cpp;do-while;cnt:40000000;lt:0.978
Cpp;for;cnt:40000000;lt:3.862
Cpp;ForEach + lambda;cnt:40000000;lt:1139.48
Cpp;ForEach + named function;cnt:40000000;lt:1143.29
Cpp;Recursive;cnt:40000000;lt:133.426
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:94
cs;Recursive;cnt:40000000;lt:89
Java;for;cnt:40000000;lt:38
Java;while;cnt:40000000;lt:48
```

```
Java;forEach;cnt:40000000;lt:232
Java;ForEach + lambda;cnt:40000000;lt:172
Java;ForEach + named function;cnt:40000000;lt:2012
Java;Recursive;cnt:40000000;lt:114
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 36
Js;ForEach + lambda function;cnt: 40000000 ;lt: 247
Js;ForEach + named function;cnt: 40000000 ;lt: 237
Js;Recursive;cnt: 40000000 ;lt: 272
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   565
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   139
Py;For;cnt: 40000000 ;lt: 1654
Py;While;cnt: 40000000 ;lt: 2797
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 265
Py;ForEach + named function;cnt: 40000000 ;lt: 3279
Py;recursive;cnt: 40000000 ;lt: 10467

Cpp;while;cnt:40000000;lt:5.164
Cpp;do-while;cnt:40000000;lt:2.288
Cpp;for;cnt:40000000;lt:4.173
Cpp;ForEach + lambda;cnt:40000000;lt:1149.63
Cpp;ForEach + named function;cnt:40000000;lt:1146.72
Cpp;Recursive;cnt:40000000;lt:134.877
cs;for;cnt:40000000;lt:24
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:111
Java;for;cnt:40000000;lt:59
Java;while;cnt:40000000;lt:52
Java;forEach;cnt:40000000;lt:247
Java;ForEach + lambda;cnt:40000000;lt:107
Java;ForEach + named function;cnt:40000000;lt:2035
Java;Recursive;cnt:40000000;lt:114
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 259
Js;ForEach + named function;cnt: 40000000 ;lt: 244
Js;Recursive;cnt: 40000000 ;lt: 280
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   646
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   152
Py;For;cnt: 40000000 ;lt: 1313
Py;While;cnt: 40000000 ;lt: 1905
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 268
Py;ForEach + named function;cnt: 40000000 ;lt: 3216
Py;recursive;cnt: 40000000 ;lt: 9942

Cpp;while;cnt:40000000;lt:0.999
Cpp;do-while;cnt:40000000;lt:2.048
Cpp;for;cnt:40000000;lt:0.98
Cpp;ForEach + lambda;cnt:40000000;lt:1216.67
Cpp;ForEach + named function;cnt:40000000;lt:1147.93
Cpp;Recursive;cnt:40000000;lt:139.67
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:95
```

```
cs;ForEach + named function;cnt:40000000;lt:94
cs;Recursive;cnt:40000000;lt:89
Java;for;cnt:40000000;lt:41
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:279
Java;ForEach + lambda;cnt:40000000;lt:193
Java;ForEach + named function;cnt:40000000;lt:1988
Java;Recursive;cnt:40000000;lt:44
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 241
Js;ForEach + named function;cnt: 40000000 ;lt: 228
Js;Recursive;cnt: 40000000 ;lt: 272
Pas;for;cnt:40000000;lt:    81
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   583
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 809
Py;While;cnt: 40000000 ;lt: 2704
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 234
Py;ForEach + named function;cnt: 40000000 ;lt: 3207
Py;recursive;cnt: 40000000 ;lt: 10193

Cpp;while;cnt:40000000;lt:1.675
Cpp;do-while;cnt:40000000;lt:1.435
Cpp;for;cnt:40000000;lt:4.495
Cpp;ForEach + lambda;cnt:40000000;lt:1117.01
Cpp;ForEach + named function;cnt:40000000;lt:1149.84
Cpp;Recursive;cnt:40000000;lt:127.438
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:7
cs;ForEach + lambda;cnt:40000000;lt:118
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:57
Java;while;cnt:40000000;lt:47
Java;forEach;cnt:40000000;lt:228
Java;ForEach + lambda;cnt:40000000;lt:175
Java;ForEach + named function;cnt:40000000;lt:1965
Java;Recursive;cnt:40000000;lt:117
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 218
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   544
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   132
Py;For;cnt: 40000000 ;lt: 75
Py;While;cnt: 40000000 ;lt: 3586
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 975
Py;ForEach + named function;cnt: 40000000 ;lt: 3906
Py;recursive;cnt: 40000000 ;lt: 8930

Cpp;while;cnt:40000000;lt:48.477
Cpp;do-while;cnt:40000000;lt:174.121
Cpp;for;cnt:40000000;lt:0.981
Cpp;ForEach + lambda;cnt:40000000;lt:1697.38
Cpp;ForEach + named function;cnt:40000000;lt:1959.92
```

```
Cpp;Recursive;cnt:40000000;lt:216.816
cs;for;cnt:40000000;lt:88
cs;while;cnt:40000000;lt:60
cs;ForEach + lambda;cnt:40000000;lt:311
cs;ForEach + named function;cnt:40000000;lt:212
cs;Recursive;cnt:40000000;lt:128
Java;for;cnt:40000000;lt:76
Java;while;cnt:40000000;lt:183
Java;forEach;cnt:40000000;lt:510
Java;ForEach + lambda;cnt:40000000;lt:324
Java;ForEach + named function;cnt:40000000;lt:3482
Java;Recursive;cnt:40000000;lt:148
Js;for;cnt: 40000000 ;lt: 206
Js;While;cnt: 40000000 ;lt: 115
Js;ForEach + lambda function;cnt: 40000000 ;lt: 529
Js;ForEach + named function;cnt: 40000000 ;lt: 448
Js;Recursive;cnt: 40000000 ;lt: 400
Pas;for;cnt:40000000;lt:   278
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   817
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   215
Py;For;cnt: 40000000 ;lt: 1964
Py;While;cnt: 40000000 ;lt: 3432
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 1442
Py;ForEach + named function;cnt: 40000000 ;lt: 5341
Py;recursive;cnt: 40000000 ;lt: 14436

Cpp;while;cnt:40000000;lt:13.586
Cpp;do-while;cnt:40000000;lt:1.62
Cpp;for;cnt:40000000;lt:5.154
Cpp;ForEach + lambda;cnt:40000000;lt:1106.2
Cpp;ForEach + named function;cnt:40000000;lt:1108.6
Cpp;Recursive;cnt:40000000;lt:127.709
cs;for;cnt:40000000;lt:14
cs;while;cnt:40000000;lt:31
cs;ForEach + lambda;cnt:40000000;lt:113
cs;ForEach + named function;cnt:40000000;lt:88
cs;Recursive;cnt:40000000;lt:94
Java;for;cnt:40000000;lt:34
Java;while;cnt:40000000;lt:40
Java;forEach;cnt:40000000;lt:211
Java;ForEach + lambda;cnt:40000000;lt:154
Java;ForEach + named function;cnt:40000000;lt:1977
Java;Recursive;cnt:40000000;lt:110
Js;for;cnt: 40000000 ;lt: 34
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 249
Js;ForEach + named function;cnt: 40000000 ;lt: 249
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    81
Pas;while;cnt:40000000;lt:    69
Pas;forEach;cnt:40000000;lt:   515
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   128
Py;For;cnt: 40000000 ;lt: 1158
Py;While;cnt: 40000000 ;lt: 1598
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 735
Py;ForEach + named function;cnt: 40000000 ;lt: 3325
Py;recursive;cnt: 40000000 ;lt: 9003

Cpp;while;cnt:40000000;lt:7.54
```

```
Cpp;do-while;cnt:40000000;lt:6.197
Cpp;for;cnt:40000000;lt:0.663
Cpp;ForEach + lambda;cnt:40000000;lt:1136.38
Cpp;ForEach + named function;cnt:40000000;lt:1125.48
Cpp;Recursive;cnt:40000000;lt:128.11
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:95
cs;ForEach + named function;cnt:40000000;lt:98
cs;Recursive;cnt:40000000;lt:102
Java;for;cnt:40000000;lt:62
Java;while;cnt:40000000;lt:49
Java;forEach;cnt:40000000;lt:219
Java;ForEach + lambda;cnt:40000000;lt:138
Java;ForEach + named function;cnt:40000000;lt:2032
Java;Recursive;cnt:40000000;lt:89
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 246
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 255
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    52
Pas;forEach;cnt:40000000;lt:   574
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   148
Py;For;cnt: 40000000 ;lt: 1175
Py;While;cnt: 40000000 ;lt: 1874
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 233
Py;ForEach + named function;cnt: 40000000 ;lt: 3260
Py;recursive;cnt: 40000000 ;lt: 9195

Cpp;while;cnt:40000000;lt:13.119
Cpp;do-while;cnt:40000000;lt:17.027
Cpp;for;cnt:40000000;lt:9.737
Cpp;ForEach + lambda;cnt:40000000;lt:1119.74
Cpp;ForEach + named function;cnt:40000000;lt:1099.06
Cpp;Recursive;cnt:40000000;lt:128.921
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:98
cs;ForEach + named function;cnt:40000000;lt:94
cs;Recursive;cnt:40000000;lt:83
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:42
Java;forEach;cnt:40000000;lt:220
Java;ForEach + lambda;cnt:40000000;lt:174
Java;ForEach + named function;cnt:40000000;lt:1971
Java;Recursive;cnt:40000000;lt:100
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 242
Js;ForEach + named function;cnt: 40000000 ;lt: 233
Js;Recursive;cnt: 40000000 ;lt: 284
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   549
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 1145
Py;While;cnt: 40000000 ;lt: 1776
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 729
```

```
Py;ForEach + named function;cnt: 40000000 ;lt: 3322
Py;recursive;cnt: 40000000 ;lt: 8969

Cpp;while;cnt:40000000;lt:3.234
Cpp;do-while;cnt:40000000;lt:0.561
Cpp;for;cnt:40000000;lt:4.021
Cpp;ForEach + lambda;cnt:40000000;lt:1119.4
Cpp;ForEach + named function;cnt:40000000;lt:1111.89
Cpp;Recursive;cnt:40000000;lt:128.048
cs;for;cnt:40000000;lt:22
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:94
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:87
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:49
Java;forEach;cnt:40000000;lt:232
Java;ForEach + lambda;cnt:40000000;lt:184
Java;ForEach + named function;cnt:40000000;lt:1972
Java;Recursive;cnt:40000000;lt:100
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 230
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   553
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   130
Py;For;cnt: 40000000 ;lt: 1162
Py;While;cnt: 40000000 ;lt: 2469
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 248
Py;ForEach + named function;cnt: 40000000 ;lt: 5494
Py;recursive;cnt: 40000000 ;lt: 9184

Cpp;while;cnt:40000000;lt:0.22
Cpp;do-while;cnt:40000000;lt:0.463
Cpp;for;cnt:40000000;lt:5.005
Cpp;ForEach + lambda;cnt:40000000;lt:1107.88
Cpp;ForEach + named function;cnt:40000000;lt:1121.43
Cpp;Recursive;cnt:40000000;lt:131.073
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:36
Java;while;cnt:40000000;lt:53
Java;forEach;cnt:40000000;lt:225
Java;ForEach + lambda;cnt:40000000;lt:159
Java;ForEach + named function;cnt:40000000;lt:1950
Java;Recursive;cnt:40000000;lt:95
Js;for;cnt: 40000000 ;lt: 32
Js;While;cnt: 40000000 ;lt: 31
Js;ForEach + lambda function;cnt: 40000000 ;lt: 250
Js;ForEach + named function;cnt: 40000000 ;lt: 236
Js;Recursive;cnt: 40000000 ;lt: 263
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   546
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
```

```
Pas;Recursive;cnt:40000000;lt:   127
Py;For;cnt: 40000000 ;lt: 1197
Py;While;cnt: 40000000 ;lt: 1660
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 47
Py;ForEach + named function;cnt: 40000000 ;lt: 2791
Py;recursive;cnt: 40000000 ;lt: 8820

Cpp;while;cnt:40000000;lt:0.233
Cpp;do-while;cnt:40000000;lt:2.049
Cpp;for;cnt:40000000;lt:6.047
Cpp;ForEach + lambda;cnt:40000000;lt:1114.01
Cpp;ForEach + named function;cnt:40000000;lt:1118.2
Cpp;Recursive;cnt:40000000;lt:133.1
cs;for;cnt:40000000;lt:29
cs;while;cnt:40000000;lt:23
cs;ForEach + lambda;cnt:40000000;lt:92
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:55
Java;while;cnt:40000000;lt:52
Java;forEach;cnt:40000000;lt:224
Java;ForEach + lambda;cnt:40000000;lt:161
Java;ForEach + named function;cnt:40000000;lt:1966
Java;Recursive;cnt:40000000;lt:117
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 235
Js;Recursive;cnt: 40000000 ;lt: 261
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   550
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   137
Py;For;cnt: 40000000 ;lt: 81
Py;While;cnt: 40000000 ;lt: 2897
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 216
Py;ForEach + named function;cnt: 40000000 ;lt: 3273
Py;recursive;cnt: 40000000 ;lt: 11002

Cpp;while;cnt:40000000;lt:1.006
Cpp;do-while;cnt:40000000;lt:0.084
Cpp;for;cnt:40000000;lt:2.409
Cpp;ForEach + lambda;cnt:40000000;lt:1114.54
Cpp;ForEach + named function;cnt:40000000;lt:1099.17
Cpp;Recursive;cnt:40000000;lt:128.804
cs;for;cnt:40000000;lt:22
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:94
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:209
Java;ForEach + lambda;cnt:40000000;lt:160
Java;ForEach + named function;cnt:40000000;lt:1966
Java;Recursive;cnt:40000000;lt:113
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 241
Js;ForEach + named function;cnt: 40000000 ;lt: 234
Js;Recursive;cnt: 40000000 ;lt: 257
Pas;for;cnt:40000000;lt:    79
```

```
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   545
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 461
Py;While;cnt: 40000000 ;lt: 2933
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 1159
Py;ForEach + named function;cnt: 40000000 ;lt: 3164
Py;recursive;cnt: 40000000 ;lt: 11564

Cpp;while;cnt:40000000;lt:0.98
Cpp;do-while;cnt:40000000;lt:2.854
Cpp;for;cnt:40000000;lt:6.992
Cpp;ForEach + lambda;cnt:40000000;lt:1116.53
Cpp;ForEach + named function;cnt:40000000;lt:1115.34
Cpp;Recursive;cnt:40000000;lt:127.015
cs;for;cnt:40000000;lt:38
cs;while;cnt:40000000;lt:8
cs;ForEach + lambda;cnt:40000000;lt:102
cs;ForEach + named function;cnt:40000000;lt:93
cs;Recursive;cnt:40000000;lt:80
Java;for;cnt:40000000;lt:64
Java;while;cnt:40000000;lt:49
Java;forEach;cnt:40000000;lt:210
Java;ForEach + lambda;cnt:40000000;lt:157
Java;ForEach + named function;cnt:40000000;lt:1985
Java;Recursive;cnt:40000000;lt:105
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 240
Js;ForEach + named function;cnt: 40000000 ;lt: 233
Js;Recursive;cnt: 40000000 ;lt: 264
Pas;for;cnt:40000000;lt:    80
Pas;while;cnt:40000000;lt:    68
Pas;forEach;cnt:40000000;lt:   557
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   147
Py;For;cnt: 40000000 ;lt: 1180
Py;While;cnt: 40000000 ;lt: 1764
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 320
Py;ForEach + named function;cnt: 40000000 ;lt: 3195
Py;recursive;cnt: 40000000 ;lt: 9001

Cpp;while;cnt:40000000;lt:2.975
Cpp;do-while;cnt:40000000;lt:3.126
Cpp;for;cnt:40000000;lt:4.612
Cpp;ForEach + lambda;cnt:40000000;lt:1122.08
Cpp;ForEach + named function;cnt:40000000;lt:1115.68
Cpp;Recursive;cnt:40000000;lt:125.834
cs;for;cnt:40000000;lt:24
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:91
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:47
Java;forEach;cnt:40000000;lt:222
Java;ForEach + lambda;cnt:40000000;lt:148
Java;ForEach + named function;cnt:40000000;lt:1964
Java;Recursive;cnt:40000000;lt:102
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
```

```
Js;ForEach + lambda function;cnt: 40000000 ;lt: 240
Js;ForEach + named function;cnt: 40000000 ;lt: 232
Js;Recursive;cnt: 40000000 ;lt: 262
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   552
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1146
Py;While;cnt: 40000000 ;lt: 1817
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 193
Py;ForEach + named function;cnt: 40000000 ;lt: 3313
Py;recursive;cnt: 40000000 ;lt: 11275

Cpp;while;cnt:40000000;lt:3.112
Cpp;do-while;cnt:40000000;lt:1.315
Cpp;for;cnt:40000000;lt:3.389
Cpp;ForEach + lambda;cnt:40000000;lt:1113.93
Cpp;ForEach + named function;cnt:40000000;lt:1119.23
Cpp;Recursive;cnt:40000000;lt:128.719
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:92
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:87
Java;for;cnt:40000000;lt:55
Java;while;cnt:40000000;lt:50
Java;forEach;cnt:40000000;lt:241
Java;ForEach + lambda;cnt:40000000;lt:159
Java;ForEach + named function;cnt:40000000;lt:1982
Java;Recursive;cnt:40000000;lt:104
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 30
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 231
Js;Recursive;cnt: 40000000 ;lt: 261
Pas;for;cnt:40000000;lt:    87
Pas;while;cnt:40000000;lt:    66
Pas;forEach;cnt:40000000;lt:   554
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1135
Py;While;cnt: 40000000 ;lt: 1383
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 194
Py;ForEach + named function;cnt: 40000000 ;lt: 2675
Py;recursive;cnt: 40000000 ;lt: 8004

Cpp;while;cnt:40000000;lt:7.681
Cpp;do-while;cnt:40000000;lt:1.819
Cpp;for;cnt:40000000;lt:6.874
Cpp;ForEach + lambda;cnt:40000000;lt:1107.76
Cpp;ForEach + named function;cnt:40000000;lt:1107.76
Cpp;Recursive;cnt:40000000;lt:140.189
cs;for;cnt:40000000;lt:14
cs;while;cnt:40000000;lt:24
cs;ForEach + lambda;cnt:40000000;lt:245
cs;ForEach + named function;cnt:40000000;lt:549
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:59
Java;forEach;cnt:40000000;lt:224
Java;ForEach + lambda;cnt:40000000;lt:153
```

```
Java;ForEach + named function;cnt:40000000;lt:1981
Java;Recursive;cnt:40000000;lt:109
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 241
Js;ForEach + named function;cnt: 40000000 ;lt: 233
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   549
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1131
Py;While;cnt: 40000000 ;lt: 1825
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 664
Py;ForEach + named function;cnt: 40000000 ;lt: 3704
Py;recursive;cnt: 40000000 ;lt: 10454

Cpp;while;cnt:40000000;lt:1.2
Cpp;do-while;cnt:40000000;lt:1.137
Cpp;for;cnt:40000000;lt:0.599
Cpp;ForEach + lambda;cnt:40000000;lt:1111.89
Cpp;ForEach + named function;cnt:40000000;lt:1113.95
Cpp;Recursive;cnt:40000000;lt:127.868
cs;for;cnt:40000000;lt:29
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:94
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:93
Java;for;cnt:40000000;lt:23
Java;while;cnt:40000000;lt:36
Java;forEach;cnt:40000000;lt:203
Java;ForEach + lambda;cnt:40000000;lt:158
Java;ForEach + named function;cnt:40000000;lt:1969
Java;Recursive;cnt:40000000;lt:114
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 230
Js;Recursive;cnt: 40000000 ;lt: 261
Pas;for;cnt:40000000;lt:    90
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   576
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   126
Py;For;cnt: 40000000 ;lt: 1141
Py;While;cnt: 40000000 ;lt: 1847
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 52
Py;ForEach + named function;cnt: 40000000 ;lt: 3655
Py;recursive;cnt: 40000000 ;lt: 8668

Cpp;while;cnt:40000000;lt:4.09
Cpp;do-while;cnt:40000000;lt:2.544
Cpp;for;cnt:40000000;lt:6.484
Cpp;ForEach + lambda;cnt:40000000;lt:1115.32
Cpp;ForEach + named function;cnt:40000000;lt:1108.97
Cpp;Recursive;cnt:40000000;lt:110.164
cs;for;cnt:40000000;lt:16
cs;while;cnt:40000000;lt:31
cs;ForEach + lambda;cnt:40000000;lt:113
cs;ForEach + named function;cnt:40000000;lt:81
cs;Recursive;cnt:40000000;lt:89
```

```
Java;for;cnt:40000000;lt:51
Java;while;cnt:40000000;lt:63
Java;forEach;cnt:40000000;lt:220
Java;ForEach + lambda;cnt:40000000;lt:148
Java;ForEach + named function;cnt:40000000;lt:1965
Java;Recursive;cnt:40000000;lt:115
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 256
Js;Recursive;cnt: 40000000 ;lt: 253
Pas;for;cnt:40000000;lt:    90
Pas;while;cnt:40000000;lt:    66
Pas;forEach;cnt:40000000;lt:   538
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1190
Py;While;cnt: 40000000 ;lt: 1833
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 270
Py;ForEach + named function;cnt: 40000000 ;lt: 4329
Py;recursive;cnt: 40000000 ;lt: 9364

Cpp;while;cnt:40000000;lt:16.581
Cpp;do-while;cnt:40000000;lt:10.326
Cpp;for;cnt:40000000;lt:17.516
Cpp;ForEach + lambda;cnt:40000000;lt:1101.75
Cpp;ForEach + named function;cnt:40000000;lt:1132.63
Cpp;Recursive;cnt:40000000;lt:135.849
cs;for;cnt:40000000;lt:26
cs;while;cnt:40000000;lt:31
cs;ForEach + lambda;cnt:40000000;lt:101
cs;ForEach + named function;cnt:40000000;lt:90
cs;Recursive;cnt:40000000;lt:88
Java;for;cnt:40000000;lt:40
Java;while;cnt:40000000;lt:60
Java;forEach;cnt:40000000;lt:196
Java;ForEach + lambda;cnt:40000000;lt:201
Java;ForEach + named function;cnt:40000000;lt:1931
Java;Recursive;cnt:40000000;lt:108
Js;for;cnt: 40000000 ;lt: 35
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 251
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    76
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   561
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   126
Py;For;cnt: 40000000 ;lt: 2125
Py;While;cnt: 40000000 ;lt: 1330
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 2215
Py;ForEach + named function;cnt: 40000000 ;lt: 4767
Py;recursive;cnt: 40000000 ;lt: 9615

Cpp;while;cnt:40000000;lt:9.26
Cpp;do-while;cnt:40000000;lt:7.192
Cpp;for;cnt:40000000;lt:2.262
Cpp;ForEach + lambda;cnt:40000000;lt:1122.75
Cpp;ForEach + named function;cnt:40000000;lt:1137.29
Cpp;Recursive;cnt:40000000;lt:123.278
cs;for;cnt:40000000;lt:18
```

```
cs;while;cnt:40000000;lt:16
cs;ForEach + lambda;cnt:40000000;lt:113
cs;ForEach + named function;cnt:40000000;lt:111
cs;Recursive;cnt:40000000;lt:78
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:48
Java;forEach;cnt:40000000;lt:230
Java;ForEach + lambda;cnt:40000000;lt:161
Java;ForEach + named function;cnt:40000000;lt:1995
Java;Recursive;cnt:40000000;lt:103
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 245
Js;ForEach + named function;cnt: 40000000 ;lt: 235
Js;Recursive;cnt: 40000000 ;lt: 256
Pas;for;cnt:40000000;lt:   102
Pas;while;cnt:40000000;lt:    53
Pas;forEach;cnt:40000000;lt:   564
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   134
Py;For;cnt: 40000000 ;lt: 1201
Py;While;cnt: 40000000 ;lt: 2038
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 578
Py;ForEach + named function;cnt: 40000000 ;lt: 4587
Py;recursive;cnt: 40000000 ;lt: 8701

Cpp;while;cnt:40000000;lt:12.72
Cpp;do-while;cnt:40000000;lt:9.18
Cpp;for;cnt:40000000;lt:1.849
Cpp;ForEach + lambda;cnt:40000000;lt:1101.42
Cpp;ForEach + named function;cnt:40000000;lt:1105.52
Cpp;Recursive;cnt:40000000;lt:124.376
cs;for;cnt:40000000;lt:4
cs;while;cnt:40000000;lt:10
cs;ForEach + lambda;cnt:40000000;lt:124
cs;ForEach + named function;cnt:40000000;lt:105
cs;Recursive;cnt:40000000;lt:92
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:220
Java;ForEach + lambda;cnt:40000000;lt:167
Java;ForEach + named function;cnt:40000000;lt:1988
Java;Recursive;cnt:40000000;lt:93
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 244
Js;ForEach + named function;cnt: 40000000 ;lt: 230
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    87
Pas;while;cnt:40000000;lt:    53
Pas;forEach;cnt:40000000;lt:   537
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   138
Py;For;cnt: 40000000 ;lt: 1129
Py;While;cnt: 40000000 ;lt: 2036
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 246
Py;ForEach + named function;cnt: 40000000 ;lt: 2955
Py;recursive;cnt: 40000000 ;lt: 9146

Cpp;while;cnt:40000000;lt:2.277
Cpp;do-while;cnt:40000000;lt:11.445
Cpp;for;cnt:40000000;lt:14.63
```

```
Cpp;ForEach + lambda;cnt:40000000;lt:1127.67
Cpp;ForEach + named function;cnt:40000000;lt:1130.66
Cpp;Recursive;cnt:40000000;lt:118.777
cs;for;cnt:40000000;lt:17
cs;while;cnt:40000000;lt:14
cs;ForEach + lambda;cnt:40000000;lt:93
cs;ForEach + named function;cnt:40000000;lt:89
cs;Recursive;cnt:40000000;lt:95
Java;for;cnt:40000000;lt:45
Java;while;cnt:40000000;lt:58
Java;forEach;cnt:40000000;lt:216
Java;ForEach + lambda;cnt:40000000;lt:175
Java;ForEach + named function;cnt:40000000;lt:1961
Java;Recursive;cnt:40000000;lt:99
Js;for;cnt: 40000000 ;lt: 37
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 248
Js;ForEach + named function;cnt: 40000000 ;lt: 259
Js;Recursive;cnt: 40000000 ;lt: 258
Pas;for;cnt:40000000;lt:    79
Pas;while;cnt:40000000;lt:    68
Pas;forEach;cnt:40000000;lt:   528
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 3913
Py;While;cnt: 40000000 ;lt: 1955
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 262
Py;ForEach + named function;cnt: 40000000 ;lt: 3428
Py;recursive;cnt: 40000000 ;lt: 9675

Cpp;while;cnt:40000000;lt:6.028
Cpp;do-while;cnt:40000000;lt:1.992
Cpp;for;cnt:40000000;lt:3.412
Cpp;ForEach + lambda;cnt:40000000;lt:1157.15
Cpp;ForEach + named function;cnt:40000000;lt:1125.75
Cpp;Recursive;cnt:40000000;lt:130.32
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:16
cs;ForEach + lambda;cnt:40000000;lt:99
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:90
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:50
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:164
Java;ForEach + named function;cnt:40000000;lt:1934
Java;Recursive;cnt:40000000;lt:111
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 258
Js;ForEach + named function;cnt: 40000000 ;lt: 243
Js;Recursive;cnt: 40000000 ;lt: 256
Pas;for;cnt:40000000;lt:    78
Pas;while;cnt:40000000;lt:    65
Pas;forEach;cnt:40000000;lt:   559
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   137
Py;For;cnt: 40000000 ;lt: 1207
Py;While;cnt: 40000000 ;lt: 1839
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 258
Py;ForEach + named function;cnt: 40000000 ;lt: 3196
Py;recursive;cnt: 40000000 ;lt: 9268
```

```
Cpp;while;cnt:40000000;lt:0.358
Cpp;do-while;cnt:40000000;lt:2.786
Cpp;for;cnt:40000000;lt:3.322
Cpp;ForEach + lambda;cnt:40000000;lt:1107.68
Cpp;ForEach + named function;cnt:40000000;lt:1109.93
Cpp;Recursive;cnt:40000000;lt:149.029
cs;for;cnt:40000000;lt:23
cs;while;cnt:40000000;lt:19
cs;ForEach + lambda;cnt:40000000;lt:96
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:87
Java;for;cnt:40000000;lt:37
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:160
Java;ForEach + named function;cnt:40000000;lt:1983
Java;Recursive;cnt:40000000;lt:96
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    68
Pas;forEach;cnt:40000000;lt:   556
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   133
Py;For;cnt: 40000000 ;lt: 1177
Py;While;cnt: 40000000 ;lt: 1822
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 180
Py;ForEach + named function;cnt: 40000000 ;lt: 3663
Py;recursive;cnt: 40000000 ;lt: 9222

Cpp;while;cnt:40000000;lt:5.169
Cpp;do-while;cnt:40000000;lt:3.163
Cpp;for;cnt:40000000;lt:0.998
Cpp;ForEach + lambda;cnt:40000000;lt:1135.57
Cpp;ForEach + named function;cnt:40000000;lt:1123.54
Cpp;Recursive;cnt:40000000;lt:129.678
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:21
cs;ForEach + lambda;cnt:40000000;lt:79
cs;ForEach + named function;cnt:40000000;lt:137
cs;Recursive;cnt:40000000;lt:90
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:54
Java;forEach;cnt:40000000;lt:235
Java;ForEach + lambda;cnt:40000000;lt:161
Java;ForEach + named function;cnt:40000000;lt:1965
Java;Recursive;cnt:40000000;lt:157
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 33
Js;ForEach + lambda function;cnt: 40000000 ;lt: 243
Js;ForEach + named function;cnt: 40000000 ;lt: 237
Js;Recursive;cnt: 40000000 ;lt: 255
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   536
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 100
```

```
Py;While;cnt: 40000000 ;lt: 1208
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 498
Py;ForEach + named function;cnt: 40000000 ;lt: 3102
Py;recursive;cnt: 40000000 ;lt: 9127

Cpp;while;cnt:40000000;lt:3.758
Cpp;do-while;cnt:40000000;lt:2.042
Cpp;for;cnt:40000000;lt:6.67
Cpp;ForEach + lambda;cnt:40000000;lt:1093.56
Cpp;ForEach + named function;cnt:40000000;lt:889.726
Cpp;Recursive;cnt:40000000;lt:236.729
cs;for;cnt:40000000;lt:12
cs;while;cnt:40000000;lt:24
cs;ForEach + lambda;cnt:40000000;lt:264
cs;ForEach + named function;cnt:40000000;lt:141
cs;Recursive;cnt:40000000;lt:73
Java;for;cnt:40000000;lt:53
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:313
Java;ForEach + lambda;cnt:40000000;lt:197
Java;ForEach + named function;cnt:40000000;lt:2064
Java;Recursive;cnt:40000000;lt:106
Js;for;cnt: 40000000 ;lt: 38
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 246
Js;ForEach + named function;cnt: 40000000 ;lt: 239
Js;Recursive;cnt: 40000000 ;lt: 264
Pas;for;cnt:40000000;lt:    90
Pas;while;cnt:40000000;lt:    67
Pas;forEach;cnt:40000000;lt:   558
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1199
Py;While;cnt: 40000000 ;lt: 1826
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 285
Py;ForEach + named function;cnt: 40000000 ;lt: 3359
Py;recursive;cnt: 40000000 ;lt: 11310

Cpp;while;cnt:40000000;lt:4.049
Cpp;do-while;cnt:40000000;lt:1.115
Cpp;for;cnt:40000000;lt:20.573
Cpp;ForEach + lambda;cnt:40000000;lt:1223.32
Cpp;ForEach + named function;cnt:40000000;lt:1120.09
Cpp;Recursive;cnt:40000000;lt:128.421
cs;for;cnt:40000000;lt:9
cs;while;cnt:40000000;lt:18
cs;ForEach + lambda;cnt:40000000;lt:130
cs;ForEach + named function;cnt:40000000;lt:99
cs;Recursive;cnt:40000000;lt:88
Java;for;cnt:40000000;lt:54
Java;while;cnt:40000000;lt:51
Java;forEach;cnt:40000000;lt:216
Java;ForEach + lambda;cnt:40000000;lt:190
Java;ForEach + named function;cnt:40000000;lt:1966
Java;Recursive;cnt:40000000;lt:106
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 32
Js;ForEach + lambda function;cnt: 40000000 ;lt: 239
Js;ForEach + named function;cnt: 40000000 ;lt: 236
Js;Recursive;cnt: 40000000 ;lt: 260
Pas;for;cnt:40000000;lt:    83
Pas;while;cnt:40000000;lt:    69
Pas;forEach;cnt:40000000;lt:   553
```

```
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   141
Py;For;cnt: 40000000 ;lt: 1048
Py;While;cnt: 40000000 ;lt: 791
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 248
Py;ForEach + named function;cnt: 40000000 ;lt: 3314
Py;recursive;cnt: 40000000 ;lt: 10663

Cpp;while;cnt:40000000;lt:7.188
Cpp;do-while;cnt:40000000;lt:0.537
Cpp;for;cnt:40000000;lt:12.518
Cpp;ForEach + lambda;cnt:40000000;lt:1152.84
Cpp;ForEach + named function;cnt:40000000;lt:1147.2
Cpp;Recursive;cnt:40000000;lt:127.19
cs;for;cnt:40000000;lt:28
cs;while;cnt:40000000;lt:24
cs;ForEach + lambda;cnt:40000000;lt:107
cs;ForEach + named function;cnt:40000000;lt:91
cs;Recursive;cnt:40000000;lt:86
Java;for;cnt:40000000;lt:57
Java;while;cnt:40000000;lt:42
Java;forEach;cnt:40000000;lt:218
Java;ForEach + lambda;cnt:40000000;lt:185
Java;ForEach + named function;cnt:40000000;lt:1984
Java;Recursive;cnt:40000000;lt:111
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 250
Js;ForEach + named function;cnt: 40000000 ;lt: 241
Js;Recursive;cnt: 40000000 ;lt: 281
Pas;for;cnt:40000000;lt:    78
Pas;while;cnt:40000000;lt:    60
Pas;forEach;cnt:40000000;lt:   550
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   120
Py;For;cnt: 40000000 ;lt: 1062
Py;While;cnt: 40000000 ;lt: 2101
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 293
Py;ForEach + named function;cnt: 40000000 ;lt: 3142
Py;recursive;cnt: 40000000 ;lt: 9430

Cpp;while;cnt:40000000;lt:9.256
Cpp;do-while;cnt:40000000;lt:1.018
Cpp;for;cnt:40000000;lt:1.817
Cpp;ForEach + lambda;cnt:40000000;lt:1123.14
Cpp;ForEach + named function;cnt:40000000;lt:1113.97
Cpp;Recursive;cnt:40000000;lt:129.866
cs;for;cnt:40000000;lt:21
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:104
cs;ForEach + named function;cnt:40000000;lt:92
cs;Recursive;cnt:40000000;lt:100
Java;for;cnt:40000000;lt:60
Java;while;cnt:40000000;lt:53
Java;forEach;cnt:40000000;lt:234
Java;ForEach + lambda;cnt:40000000;lt:162
Java;ForEach + named function;cnt:40000000;lt:1955
Java;Recursive;cnt:40000000;lt:111
Js;for;cnt: 40000000 ;lt: 36
Js;While;cnt: 40000000 ;lt: 34
Js;ForEach + lambda function;cnt: 40000000 ;lt: 240
Js;ForEach + named function;cnt: 40000000 ;lt: 234
```

```
Js;Recursive;cnt: 40000000 ;lt: 259
Pas;for;cnt:40000000;lt:    78
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   564
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   135
Py;For;cnt: 40000000 ;lt: 1153
Py;While;cnt: 40000000 ;lt: 1945
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 253
Py;ForEach + named function;cnt: 40000000 ;lt: 3457
Py;recursive;cnt: 40000000 ;lt: 8972

Cpp;while;cnt:40000000;lt:6.794
Cpp;do-while;cnt:40000000;lt:0.863
Cpp;for;cnt:40000000;lt:2.873
Cpp;ForEach + lambda;cnt:40000000;lt:1238.21
Cpp;ForEach + named function;cnt:40000000;lt:1205.92
Cpp;Recursive;cnt:40000000;lt:159.03
cs;for;cnt:40000000;lt:27
cs;while;cnt:40000000;lt:22
cs;ForEach + lambda;cnt:40000000;lt:167
cs;ForEach + named function;cnt:40000000;lt:180
cs;Recursive;cnt:40000000;lt:98
Java;for;cnt:40000000;lt:58
Java;while;cnt:40000000;lt:58
Java;forEach;cnt:40000000;lt:272
Java;ForEach + lambda;cnt:40000000;lt:201
Java;ForEach + named function;cnt:40000000;lt:2075
Java;Recursive;cnt:40000000;lt:110
Js;for;cnt: 40000000 ;lt: 41
Js;While;cnt: 40000000 ;lt: 35
Js;ForEach + lambda function;cnt: 40000000 ;lt: 290
Js;ForEach + named function;cnt: 40000000 ;lt: 246
Js;Recursive;cnt: 40000000 ;lt: 301
Pas;for;cnt:40000000;lt:    82
Pas;while;cnt:40000000;lt:    64
Pas;forEach;cnt:40000000;lt:   631
Pas;ForEach + lambda;cnt:40000000;lt:Not supported
Pas;ForEach + named function;cnt:40000000;lt:Not supported
Pas;Recursive;cnt:40000000;lt:   156
Py;For;cnt: 40000000 ;lt: 1375
Py;While;cnt: 40000000 ;lt: 2309
Py;ForEach [+ lambda];cnt: 40000000 ;lt: 440
Py;ForEach + named function;cnt: 40000000 ;lt: 4063
Py;recursive;cnt: 40000000 ;lt: 13387
```

Authors

MENYHÁRT László

Eötvös Loránd University, Faculty of Informatics, Department of Media and Educational Informatics, Budapest, Hungary, e-mail: menyhart@inf.elte.hu

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 1. 2020.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.1.421

License

A Problem-based Curriculum for Algorithmic Programming

NIKHÁZY László

Abstract. Engagement of students plays a crucial part in education, even if they are gifted children. We know a success story: the extracurricular mathematics camps of Lajos Pósa for talented teenagers in Hungary. The key to that success is the excellently engineered network of problems that guide students through discovering the world of higher-level mathematics. It would be a novel approach to teach computer programming and algorithms similarly. In this paper, we attempt to design a network of problems selected specifically for discovery learning of algorithms and data structures from beginner to advanced level, targeted for secondary and high school talented students. This could serve as the curriculum for extra classes or camps conducted with the problem-based teaching method we describe.

Keywords. talent education, competitive programming, discovery learning, algorithms and data structures.

1. Introduction

There is a unique system for mathematics talent education in Hungary, led by mathematician Lajos Pósa and his students. The core element of this system is the series of camps in which gifted pupils can explore mathematics with inquiry-based learning [1]. It is the author's goal to establish a similar initiative in the field of computer science. Within computer science, we focus on the core programming skills by teaching algorithmic programming from the beginner to the highest level.

Algorithmic programming involves dealing with well-defined problems to which the solution is an algorithm that calculates the desired output from the given input, and the program is a way of expressing this algorithm that allows executing and verifying the solution on a computer. Correctness and effectiveness are both key measures that are evaluated by extensive testing of the programs. There are a lot of excellent resources available online which promote learning algorithmic programming on an advanced level, for example Halim's book [2], and the massive problem base of past Codeforces contests [3]. However, to use them for our educational goals, we need to organize these materials and exercises in such a way that enables learning through a series of problem-solving.

In this discovery learning scenario, we would like to create situations in which students are facing a problem, and they have already seen the key ideas leading to the desired algorithm while solving different tasks previously. Therefore, the main challenge of the teacher is designing the curriculum and arranging the exercises in a proper structure that makes it possible to introduce the right problem at the right time. In this paper, we present a system of topics and methods, accompanied by exercises, that could serve as the curriculum for extra classes or camps conducted with the problem-based teaching method that we already use in individual and small group programming lessons for high school students.

The structure of this paper is organized as follows. In chapter 1, we briefly introduce the mathematics camps and the didactics of Pósa in the context of discovery learning. In chapter 2, we describe the goals and challenges of adapting the Pósa-method for computer programming talent education and elaborate on the design and of the curriculum. In chapter 3, we present the curriculum in detail, list of topics grouped to units and tables with the collection of tasks for each of them. In chapter 4, we show an example of a topic, dynamic programming followed throughout the entire curriculum. Chapter 5 contains a short summary of the presented content.

1.1. Discovery learning

The term discovery learning refers to pedagogical methods, in which students learn through their exploration of a certain topic. The goals are usually threefold:

- acquire deep knowledge,
- develop cognitive skills,
- increase engagement.

Deep knowledge in this sense means that the learned information, concepts and methods have very strong roots in the long-term memory, thus the person has a higher level of understanding of the subject and can apply this knowledge more successfully in new situations. During the learning process, students spend most of the time actively working individually or in groups, their knowledge is constructed by themselves through these activities. This requires immerse brain capacities used in a variety of forms, through which cognitive skills develop highly. Students' joy is a key value in the whole process, to motivate them for further participation and increase their endurance.

Discovery learning is strongly related to the constructivist learning theory, which relies on the assumption that people construct their knowledge during mental activities. The learners are considered organisms that seek meaning, and reflecting on their experience, derive their own set of rules and mental models of the world. Numerous educators apply discovery learning in modern education. Wouter van Joolingen [4] describes it as "*a type of learning where learners construct their own knowledge by experimenting with a domain and inferring rules from the results of these experiments*". He argues that they will understand the domain at a higher level than when the necessary information is just presented by a teacher or an expository learning environment. In most cases, discover learning is tied to problem-solving, Borthick and Jones [5] write that "*participants learn to recognize a problem, characterize what a solution would look like, search for relevant information, develop a solution strategy, and execute the chosen strategy*".

For us, a type of discovery learning called problem-based learning is particularly interesting, which is defined by Finkle and Torp [6] as "*a curriculum development and instructional system that simultaneously develops both problem-solving strategies and disciplinary knowledge bases and skills by placing students in the active role of problem solvers confronted with an ill-structured problem that mirrors real-world problems.*" Our approach is very close to this definition, as we will show it below.

1.2. Mathematics camps in Hungary

Mathematics talent education has a strong tradition in Hungary, and there are numerous mathematics camps. Here we describe the camps organized by The Joy of Thinking foundation [7], established by Lajos Pósa. These weekend math camps are characterized by the internationally renowned Pósa-method, which is a form of guided discovery learning. The author, being an ex-student of Lajos Pósa, has been assisting in these camps for many years, and now teaching two groups since 2014, so he has a working knowledge of Pósa's pedagogy.

According to Bibergall [8], guided discovery learning is characterized by convergent thinking. "*The educator devises a series of statements or questions that guide the learner step by step, making a series of discoveries that leads to a predetermined goal*". In our math camps, the learner is guided through exercises that have strong interconnection under the surface. Katona and Szűcs [9] describe this as a web of problem threads.

1.3. Problem threads in the Pósa-method

A problem thread consists of tasks that have a connection, which can be of different types. A type of connection might be that they share the topic, e.g. graph theory. Another type of connection is when the problems build on top of each other, meaning that the solution of one task needs certain ideas, methods that are more easily available for students if they solved a previous task. This common element of thinking, which links tasks in one thread, is called the kernel of the thread by Katona and Szűcs [9].

For example, the above-mentioned kernel could be induction, which Pósa calls chain-reaction for young students and starts with a simple logical task of the style "who robbed the bank", and later on students will get to proving complex theorems like "every tournament graph has a Hamiltonian path" using induction. Here we would like to mention that the latter statement is not presented in such a plain way, but instead, an open question with dragons carrying people between islands, to make it more fun for kids.

Pósa always emphasizes not to provide a statement to prove, but ask an open question instead, or even better – which happens in this dragon-world – just present a situation and let the students ask questions. We aim to introduce kids to research, and particularly in mathematics an interesting question is very valuable to the scientific community. In this regard, the Pósa-method is also a type of inquiry-based learning. Students start with divergent thinking when solving a task, the experimentation in the domain of mathematics has a significant role in Pósa's pedagogy.

1.4. The web of problem threads

During the mathematics sessions of the previously mentioned camps, there are always multiple problem threads running in parallel, which means that a lot of tasks from different threads are presented to the students simultaneously. The threads are not isolated, they may have meeting points, common problems, they may have important links or dependencies between them, forming a web of problem threads.

The web of problem threads is like a master plan, leading the learner to acquire knowledge and skills that are our educational goals. So, the main challenge of the teacher is to design the curriculum to suit the intended development of students, which means identifying the competencies to learn, organizing them in the right system, and collecting or creating a vast amount of problems and exercises that will trigger and guide the learning process. In the following, we show a system of topics and methods accompanied by exercises that serve as a base of our web of problem threads for algorithmic programming talent education.

2. Discovery learning in algorithmic programming

We need to define our educational goals. As for the mathematics talent education program, Juhász [10] says "*children should be taught how to think, rather than making them learn theorems and formulas by heart or giving them ready-made methods to solve problems*". Following this principle, our focus is on teaching algorithmic thinking and problem-solving. Another important objective is to show the joy in thinking about interesting problems and creating working programs to solve them. With this, we would like to open up the world of competitive programming for the children.

The emphasis is not on competition, but these contests are aimed to test the algorithmic thinking and problem-solving skills of the participants with "nice" tasks. The community of qualified programmers is preparing the problems of these competitions and they make them so that other people would enjoy thinking about them. There is a certain beauty in problems that is hard to describe,

and it is much celebrated within the community. This beauty can come from an interesting question, an elegant solution, application of a method in an unexpected situation, a nice idea, the connection between different topics, etc. So, the world of competitive programming is partly self-serving, it provides fun for people doing it, very much like how Lajos Pósa describes the world of mathematics [11].

Computer programming is a bit different from mathematics. There are a lot of standard algorithms and data structures that are almost ready-made methods that you need to customize, combine, and apply in numerous different scenarios. We try to teach them through a series of problems, having the students discover them mostly on their own, if possible. However, we put more emphasis on the applications of these methods in different problems. Therefore, we consider our approach a problem-based pedagogy. The problems have similar dependencies and connections between each other as the ones in Pósa's mathematics camps. We create problem threads for the algorithms and data structures we teach and try to connect them, thus forming our web of problems. Fortunately, the tasks at programming contests usually have some funny stories to cover the underlying problem, so at first sight it is not obvious to which thread they belong.

2.1. The objectives of our curriculum, related work

Designing the curriculum starts with identifying the topics and methods we want to teach. Programming competitions reflect quite well what the community of computer scientists consider important knowledge and skills in the field of algorithmic programming. We selected the elements of this curriculum by looking at materials of competitions and those, up-to-date literature helping to prepare for contests.

Our most important source is the Syllabus for the International Olympiad in Informatics (IOI) [12], it provides an excellent summary of expected knowledge. Competitive Programming 3 by Halim et al. [2] contains most of the topics occurring at ACM International Collegiate Programming Contest for university students. We examined the contents of Laaksonen's Competitive Programmer's Handbook as well [13]. There are some excellent on-line resources and tutorials collecting the important algorithms and data structures, for example the CP-Algorithms website [14] Geeks for Geeks [15] and various blogposts on Codeforces [16-17]. Using these sources, and our experience in the history of high school competitions, we compiled the contents of our curriculum, which are the most relevant knowledge for high school students in our opinion.

There are numerous similar articles, especially about the topic of how to prepare students for competitions. Király [18] describes a whole roadmap of teaching programming from the very beginning to the preparations for the IOI, together with proper pedagogical guidelines and useful advice. In her doctoral dissertation, Erdősné [19] provides a detailed insight into Hungarian and international talent education in informatics, also outlining a plan of teaching advanced level programming throughout secondary school. At the concrete topics where they include tasks, both papers present very similar exercises to our chosen ones. In comparison to their work and the above-mentioned literature, the novelty of this curriculum lies in tailoring the system of problems to the discovery learning method of Pósa. Since the selection of tasks and the interconnections play a crucial part in our didactics, we provide a more detailed and more complete list of exercises, organized in the structure described below.

2.2 Design and overview of the curriculum

Most of the books, online courses and tutorials about programming focus on the elements of the language. We have different goals, so these are out of scope for us, but we would like to build on the basics. Teaching algorithms, we assume knowledge of language elements, like variables, types, operators, conditional statements, loops, etc. We start at the very basic algorithmic structures and

finish with some of the most complex methods required for the IOI. The primary use of this curriculum will be a talent education program with groups of gifted children beginning around age 12 and lasting until age 17. The examples of Pósa's mathematics camps showed us that it is possible the keep such groups together for years, develop their knowledge systematically, and teach topics building upon each other.

For our problem-based method, the curriculum would be a huge network of tasks, through which students discover, practice and expand their knowledge. To design it, first, we look at the bigger picture, organize the theoretical backgrounds of the tasks, and identifying interconnections between them. The resources we used, which are mentioned above in section 2.1, present the materials categorized and ordered by topics, which is great if we search for something we already know, but not the ideal order for learning. We not only talk about their order of difficulty, but our goal is also to find the ideal plan, where we minimize the number of great ideas needed to work out the desired algorithms.

Figure 1 shows a graph of curriculum modules, each abbreviated by a short code for readability. Full names and descriptions of these can be found in the tables of section 3. We categorized these modules into four types that have different colors on the graph:

- problem-solving techniques (red),
- algorithms or algorithm templates (blue),
- data structures (green),
- theoretical backgrounds or subjects (yellow).

Interconnections between these elements are visualized by arrows, the thicker the arrow is, the stronger we find the dependency between them. We also use light grey arrows, which don't mean dependency, rather similarities, when knowing a method is helpful for the other one.

Later, in section 3 we divide the modules to units in order to have reviewable segments. We provide a collection of problems for each unit, dedicated to modules, with certain objectives. This collection is not complete, and never meant to be, extending and changing it constantly is part of our philosophy. These problems serve as a good skeleton for starting a talent education program with discovery learning. For our pedagogy, we define the following three important types of tasks.

- Introduction problems are the very first problems of a topic that can induce discovery.
- Reinforcement problems involve the application of a previously discovered method in new situations, or they can be practice problems as well, their goal is to strengthen students' knowledge.
- Synthesis problems in our terminology mean tasks that bring together multiple learned methods or require a high level of understanding of the concepts involved.

Problems are, of course, very often linked to topics and problems in different units, which cannot be shown in the format of this document.

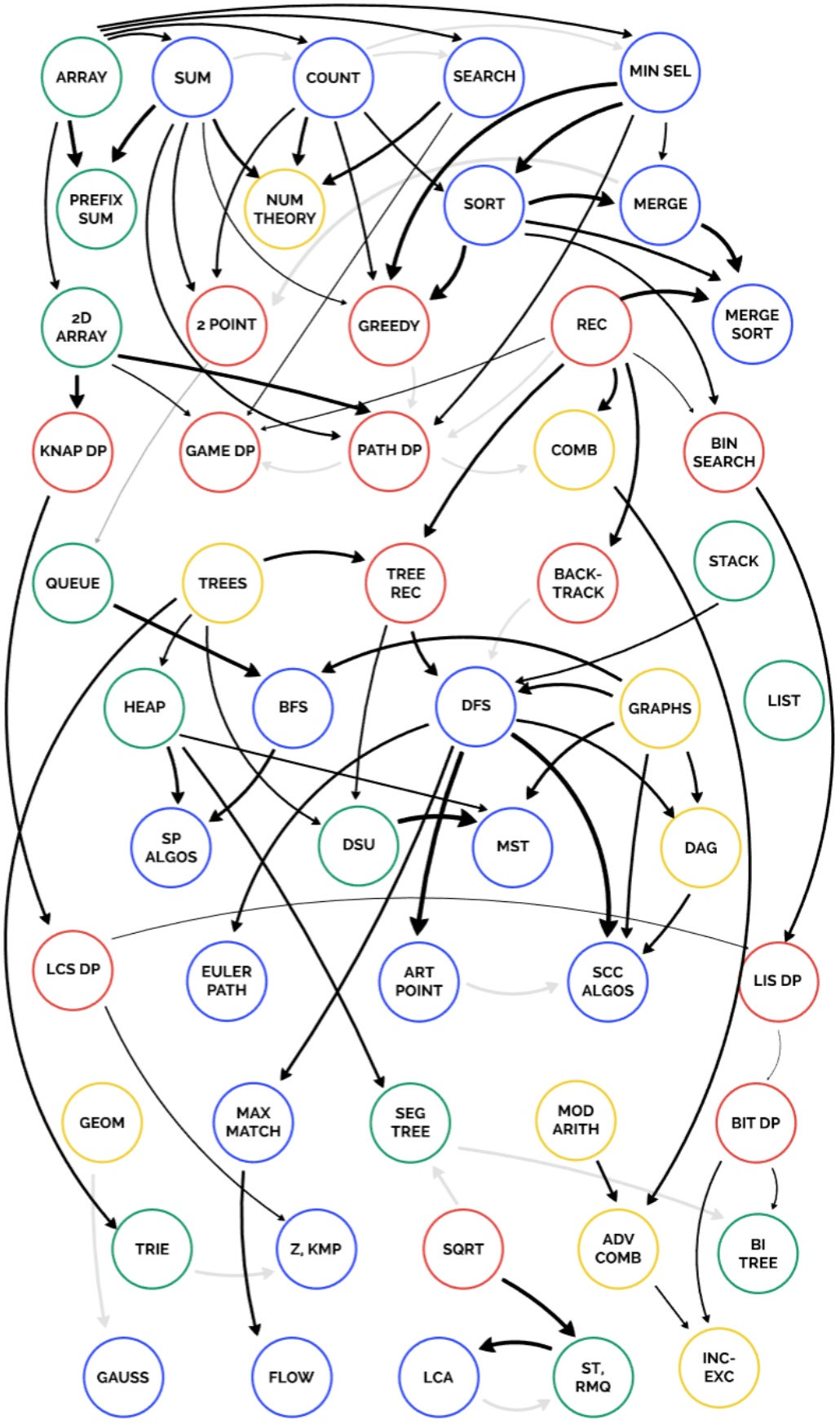


Figure 1: The graph of the curriculum

3. Details of the curriculum

Below we present the parts of the curriculum with more detailed descriptions. Grouping them into units serves as an aid for presenting, there are no hard boundaries between them, and there are very important interconnections in-between units. We estimate the length of the units to take up 2-3 weekend camps, which can be scheduled in one year or even in one semester.

Within units, we propose problems for teaching the algorithms and data structures, where applicable. They are collected from various online problem sets that are publicly available and have a judgement system to verify solutions. This is important for us, we would like to show the students that not only the theory but the implementation is an essential part of computer science. We use identifiers for the problems, that consist of an abbreviation of the problem source listed below, and a handle of the problem, which can be used to find it (by web search or on the site itself).

- CF: Codeforces, CFG: Codeforces Gym [3].
- HR: HackerRank [24].
- CC: CodeChef [25].
- CSA: CS Academy [26].
- SPOJ: Sphere Online Judge [27].
- UVa: UVa Online Judge [28].
- IOI: Tasks of the International Olympiad in Informatics. Grader for most of them can be found at PEG Online Judge [29].
- M: Mester [30]. A Hungarian problem collection. Unfortunately, translations for these problems are not available yet, we plan to write them for the ones that we use, in a new system that is under development at Eötvös Loránd University. (We give the English translation of task names in parentheses.)
- ICPC: ICPC Live Archive [31].
- TIMUS: Timus Online Judge [32].

Erdősné [20] gives an excellent overview of online judges and their features that includes all the above-listed websites. As the reader can see in the following chapters, in our problem collection we favor Codeforces, HackerRank, and Mester, because using practice mode in these the student (or the teacher) has access to the small test cases and their expected answers, which is very helpful for debugging.

3.1. Unit 1: Introduction

In this unit, we introduce some basic algorithmic patterns that are necessary as building blocks of further, more complex algorithms. Most of them operate on a sequence of numbers, so we heavily use the one-dimensional array data structure. Zsakó and Szlávi refer to them as programming theorems [27]. Students need to acquire firm knowledge of these basics so that later they can easily apply them as part of a compound solution. Knowing the required language elements, talented students can invent algorithms to solve these tasks. While there is no inevitable dependency between them, we recommend introducing them in the order below. As the first combination of loops and conditional statements (*for* and *if*), counting elements with a certain property (e.g. even numbers) is a straightforward problem.

For introducing the algorithms listed below, we generally use the tasks available in Mester [30], the Hungarian problem set, categorized accordingly, so we don't list them here. It is also not difficult to invent our own tasks for these basics. Number theory is a great topic for applying and combining what we learned in this unit. For example, deciding whether a number is a prime comes down to searching for a divisor of it. With summing the divisors, one can search for amicable numbers. It

is also a great opportunity to introduce functions as a language element. Discovering the Euclidean algorithm, students can see something advanced and very elegant. There is a great task guiding to the Euclidean algorithm (*CF - 527A. Playing with Paper*), in which you start with a rectangular paper, and always cut off square-shaped parts of it.

Code	Name	Description
SUM	Sum	Calculate the sum of a series of numbers.
COUNT	Count	Count the elements of a series with a certain property.
SEARCH	Search	Search for the element(s) with a given property in a sequence.
MIN SEL	Minimum Selection	Select the minimum/maximum in a sequence.
SORT	Sort	Sort a sequence with simple algorithms, like bubble sort, min. selection sort, counting sort, etc.
MERGE	Merge	Compute the intersection/union of two sets (ordered list of data), using the linear merge algorithm.
NUM THEORY	Basic Number Theory	Calculate the number of divisors and sum of divisors of an integer, search for primes, prime factorization, Euclidean algorithm for GCD.

Table 1: Modules of the first unit

3.2. Unit 2: Some basic problem-solving techniques

In this unit, we dive into the world of programming contest problems. Computing solutions quickly becomes a key factor, we introduce the notion of computational complexity and analyze every solution from this aspect. Greedy algorithms with obvious greedy decisions are a great start to provide easy success to everyone, and not much later problems for dynamic programming will show that greedy doesn't always work. The technique of recursion is essential to introduce early since it is used in multiple other methods of this unit. Building on the concept of recursion, and knowledge from unit 1, we can guide the learners to efficient sorting algorithms, like merge sort and quicksort.

Code	Name	Description
GREEDY	Greedy Algorithms	Solve problems using greedy decisions, recognize whether it leads to the optimum.
REC	Recursion	Get familiar with the power of recursion in typical scenarios.
MERGE SORT	Merge Sort	Recursive sort algorithms: merge sort and quicksort.
2 POINT	Two Pointers	The two pointers principle for speeding up some optimization tasks.
PREFIX SUM	Prefix Sum	The prefix sum / cumulative sum of a sequence, as a data structure.
PATH DP	DP for Finding Best Path	Introduction to dynamic programming: optimize a route on a grid.
GAME DP	DP for Simple Games	Find the winning strategy in simple two-player games with dynamic programming.
COMB	Combinatorics	Basic tasks involving combinatorics, like permutations, combinations, Fibonacci-type sequences, etc.
BIN SEARCH	Binary search	Bisect to find a value in a sorted range, and to find extremum using a predicate.
BACKTRACK	Backtrack	Speed up brute-force algorithms by backtracking, 8 queens' problem and similar.
KNAP DP	DP in Knapsack problem	Some typical DP problems: Coin Change, Knapsack and alike.

Table 2: Modules of the second unit

With the prefix sum data structure, students can construct their first powerful data structure to answer queries. Binary search appears in the form of looking for a value in a sorted range, and then comes the great idea to use it when maximizing or minimizing some target with some constraints (e.g. in task *CF - 760B. Frodo and Pillows*). We also show problems where only exponential solutions are known, but we can speed them up using backtracking. A nice introduction task could be to generate all balanced parentheses sequences of a certain length.

Dynamic programming could be built up in different ways, we propose starting it with problems in which we examine paths on a grid with only right and downwards steps. In this case, calculating partial results with a bottom-up strategy comes as a natural idea, much easier conceptually than transforming recursive formulas in problems like the Coin Change or the Knapsack problem. We eventually get there as well, but in the meantime, we also take a step applying dynamic programming for computing winning strategies in simple two-player games. It is used also in combinatorial problems, which are at this level mostly related to the Pascal triangle, Fibonacci-type sequences, permutations and variations. Below we include a table showing example tasks that we propose for the modules in this unit, according to the principles described in section 2.

Code	Introduction	Reinforcement	Synthesis
GREEDY	M - Wifi, HR - Priyanka and Toys	M - Mekk Elek (Mekk Elek the Handyman) M - Fénykép (Photo) CF - 349B. Color the Fence	M - Termek (Rooms) CF - 1077E. Thematic Contests
REC	Towers of Hanoi	M - Felbontás (Decomposition)	HR - The power sum
2 POINT	CF - 660C. Hard problem, M - Autószállítás (Car Shipping), CF - 1133C. Balanced Team	CF - 616D. Longest k-good Segment, CF - 1006C. Three Parts of the Array	M - Nyaralások (Trips), IOI11 - Rice Hub
PREFIX SUM	SPOJ - CSUMQ, CF - 313B. Ilya and Queries,	M - Távoli bolygó (Distant Planet), CF - 816B. Karen and Coffee	M - Képátló (Image Diagonal), IOI11 - Rice Hub
PATH DP	M - Kincsek a hegyoldalon (Treasures on the hillside), M - Pontgyűjtő verseny (Point collecting contest)	M - Benzin (Gasoline), CF - 429B. Working out	M - Lépcsők (Stairs), M - Képátló (Image diagonal), CF - 407B. Long Path
GAME DP	HR - Game of Stones	HR - A Chessboard Game, M - Számok elvétele (Removing Numbers)	M - Fehér és Fekete korongok (White and Black Tokens), CF - 731E. Funny Game
COMB	HR - Picking Cards, CF - 617B. Chocolate	HR - Sherlock and Pairs, CF - 894A. QAQ	M - Lépcsők (Stairs), HR - Merge List
BIN SEARCH	CF - 706B. Interesting Drink, CF - 600B. Queries About Less or Equal Elements	CF - 760B. Frodo and Pillows, CF - 670D2. Magic Powder	IOI11 - Rice Hub, UVa - 1079. A Careful Approach
BACKTRACK	Balanced Parentheses, 8 Queens problem	M - Ültetés (Seating)	CC - KOL1510
KNAP DP	M - Nem kifizethető címlet (Unpayable Amount), M - Bélyeg (Stamps), SPOJ - KNAPSACK	CF - 19B. Checkout Assistant, M - Munkagépek (Machines), M - Vásár (Sale)	CFG - 102534B. Need More T-shirts, CF - 1132E. Knapsack

Table 3: Problems for the second unit

3.3. Unit 3: Graph theory-driven problems and algorithms

A lot of real-life problems can be formulated with graphs, and so they appear frequently in programming competitions above a certain level. In this unit, we teach the most commonly used, but still not too complex algorithms. Some theory is involved, students need to learn the notion of trees and graphs. We recommend problems on rooted trees first which have recursive solutions, because they are quite elegant and serve as a base for depth-first search. Three basic, linear data structures can be learned concurrently, stack, queue and double-ended queue, we apply them in graph traversals, and at this point some beautiful and hard problems can show their advantages (*HR - Largest Rectangle*, *HR - Deque-STL*).

The two types of graph traversal (breadth-first and depth-first) are introduced on simple, undirected graphs. At this level, breadth-first search has more applications, while we build on depth-first search a lot in the next unit. The graph traversals work the same way in directed graphs, and particularly acyclic graphs of this type are interesting for us, they model practical problems like scheduling a project, university studies or this curriculum itself. There is a nice combinatorics task, where the question is how many different paths are between two vertices (*HR - Kingdom Connectivity*).

Code	Name	Description
TREES	Trees, Binary Trees	Tree structure appearing in different situations, e.g. family tree, company structure.
TREE REC	Recursion on trees	Compute values for trees using recursion, problems involving a hierarchical structure.
LIST	Linked lists	Know the basics of linked data structures, and when to use them considering their advantages and disadvantages.
STACK	Stack	Understand and use the stack (LIFO) data structure.
QUEUE	Queue, Deque	Understand and use the queue (FIFO) and double-ended queue data structures.
GRAPHS	Graphs	Conceptual introduction of graphs as a background of different problems.
BFS	Breadth-First Search	Solving problems using graph traversal in increasing order of distance from a vertex.
DFS	Depth First Search	The recursive depth-first search algorithm and basic applications.
DAG	Directed Acyclic Graphs	Problems involving a DAG, like critical path method, topological ordering.
SP ALGOS	Shortest Path Algorithms	Find shortest paths in a graph. Bellman-Ford, Floyd-Warshall, Dijkstra algorithms.
HEAP	Heap, Priority Queue	Understand the heap data structure, and use priority queue when needed, e.g. in Dijkstra and Prim algorithms
MST	Minimum Spanning Tree	Find the minimum spanning tree in graphs. Kruskal and Prim algorithms.
DSU	Disjoint Set Union	The DSU (or Union-find) data structure applied in various problems, e.g. Kruskal algorithm.

Table 4: Modules of the third unit

Two data structures, DSU and heap are included here for two reasons: they are necessary for efficient implementations of Kruskal, Dijkstra and Prim algorithms, and both are viewed as rooted trees, so they perfectly fit in here. In programming competitions, heap is generally applied by using the priority queue included in standard libraries, while DSU needs to be implemented.

We conclude the unit with two more advanced problems on weighted graphs: shortest paths and minimum spanning trees. Well-known algorithms listed below can be discovered by students with some hints. Since weighted graphs can model various real-world problems, there are a huge amount of competition tasks where these algorithms are necessary with some modifications.

Code	Introduction	Reinforcement	Synthesis
TREE REC	M - Titkos társaság (Secret association)	CF - 115A. Party, CF - 580C. Kefa and Park	HR - Even Tree
STACK	HR - Equal Stacks, UVa - 514. Rails	HR - Balanced Brackets	HR - Largest Rectangle CF - 547B. Mike and Feet
QUEUE	UVa - 10935. Throwing cards away	HR - Deque-STL	IOI06 - Pyramid
BFS	List vertices in order of distance from one vertex	M - Randi (Date), M - Csapat (Team), CF - 796D. Police Stations	M - Mérőkannák (Measuring cups), CSA - BFS-DFS
DFS	SPOJ - ABCPATH	CF - 445B DZY Loves Chemistry, M - Utcaseprő (Street sweeper)	CF - 1316D. Nash Matrix CSA - BFS-DFS
DAG	M - Építkezés (Contruction), M - Utak száma (Number of routes)	CF- 915D. Almost Acyclic Graph CF - 512A. Fox and Names	HR - Kingdom Connectivity
SP ALGOS	CF - 20C. Dijkstra, CF - 295B. Greg and Graph	M - Autóbusz járatok (Bus lines), HR - Jack goes to Rapture	M - Telephelyek (Sites), IOI11 - Crocodile
MST	SPOJ - MST	M - Malom (Mill)	HR - Roads in Hackerland CF - 160D. Edges in MST
DSU	CF - 1095F. Make It Connected	M - Hálózat tesztelés SPOJ - CONSEC	CC - ABROADS CF - 875F. Royal Questions

Table 5: Problems for the third unit

3.4. Unit 4: Various advanced algorithms, geometry, combinatorics

Our fourth unit contains some theoretically complicated algorithms. Speaking of Hungarian national contests, these are only required for the highest age group (11-12th grade). One can argue that discovery learning is not possible in some of these topics. We still aim for introducing them through exercises in which we provide hints to the students. Furthermore, deep understanding can be also achieved when the students implement solutions based on these complex algorithms.

Code	Name	Description
LCS DP	Longest Common Subsequence	Dynamic programming using non-trivial two-dimensional arrays.
LIS DP	Longest Increasing Subsequence	Dynamic programming sped up with binary search or other methods.
BIT DP	Bitmask DP	Dynamic programming on subsets, using the bit vector representation of sets.
ART POINT	Articulation Points, Bridges	Biconnected graphs, Tarjan's algorithm and the L-value for finding cut vertices and edges.
SCC ALGOS	Strongly Connected Components	Kosaraju's and Tarjan's algorithm and applications.
EULER PATH	Eulerian Path	Condition of Eulerian path or circuit and finding it in directed and undirected graphs.
MAX MATCH	Maximal Matching	Hungarian algorithm for the maximal matching in a bipartite graph.
MOD ARITH	Modular Arithmetics	Calculate powers and inverses efficiently modulo a given number.
ADV COMB	Advanced Combinatorics	Various difficult combinatorics problems.
INC-EXC	Inclusion-Exclusion Principle	Apply the inclusion-exclusion principle to answer some questions in combinatorics.
GEOM	Geometry	Geometric problems on the Cartesian plane.

Table 6: Modules of the fourth unit

There are graph theory topics (Articulation Points and Bridges, Strongly Connected Components, Eulerian Path, Maximal Matching) which mostly rely on depth-first search and its extensions. We would like to mention two beautiful tasks, *CF - 508D. Tanya and Password*, which is a surprising application of Eulerian paths, and *UVa - 12668. Attacking Rooks*, where the idea is to introduce a bipartite graph where the edges are fields on the chess table.

Dynamic programming is present throughout the curriculum in many other algorithms, but here we revisit it with more advanced applications. Below, we named two characteristic tasks (Longest Common Subsequence and Longest Increasing Subsequence), but there are much more included, in the first topic could be any other task that requires a non-trivial two-dimensional array formulation, and the second is related to tasks where we combine dynamic programming with other techniques, like binary search in the example problem.

Advanced Combinatorics includes not only questions about how many ways we can construct something, there is often an ordering (e.g. lexicographical) defined between these and telling the element at a given position requires a profound understanding of recursive patterns. The number of solutions is often very large, and then the answer is expected modulo some big prime, so the apparatus of modular arithmetic is used here. That is also very interesting itself, cryptographic applications can be visited.

Code	Introduction	Reinforcement	Synthesis
LCS DP	HR - The LCS	M - Jelek (Signs), M - Rúd felvágás (Stick cutting)	CF - 607B. Zuma, HR - LCS Returns IOI09 - Raisins
LIS DP	M - Konténeroszlopok (Container Columns)	M - Kockákból legmagasabb torony (Highest Tower of Cubes)	CF - 650D. Zip-line
BIT DP	M - Vásárlások (Purchases)	CF - 580D. Kefa and Dishes	CFG - 102128B. Cake Tasting
ART POINT	SPOJ - SUBMERGE	M - Duplán elérhető pontok (Double reachable points)	CF - 700C. Break Up CF - 732F. Tourist Reform
SCC ALGOS	SPOJ - CAPCITY UVa 13057 - Prove Them All	CF - 427C. Checkposts CF- 949 C. Data Center Maintenance	M - Hercegek házassága (Wedding of Princes) SPOJ - ADAPANEL
EULER PATH	M - Zárkód (Lock code) CF - 1334D. Minimum Euler Cycle	CF - 508D. Tanya and Password	M - Dominó
MAX MATCH	HR - Real Estate Broker	UVa - 12668. Attacking Rooks CF - 498C. Array and Operations	M - Hercegek házassága (Wedding of Princes) SPOJ - QUEST4
MOD ARITH	UVa - 10104. Euclid Problem	CF - 300C. Beautiful Numbers, CF - 717A. Festival Organization	M - Szigetek (Islands), HR - Game of Thrones II
ADV COMB	HR - Lexicographic steps, CF - 9D. How many Trees? CC - NWAYS	CF - 612E. Square Root of Permutation, M - Birtokfelosztás (Dividing land)	M - Szigetek (Islands), HR - Game of Thrones II
INC-EXC	SPOJ - NGM2	UVa - 11806. Cheerleaders	CF - 102128B. Cake Tasting
GEOM	M - Házak (Houses) M - Zárt poligon készítése (Creating a closed polygon)	M - Autópálya (Highway) M - Háromszög (Triangle)	M - Hegy (Mountain) CF - 552D. Vanya and Triangles. UVa - 12278. 3-sided dice

Table 7: Problems for the fourth unit

Geometry is a huge category, it starts with basic operations, like computing orientations, deciding if segments intersect, etc. and leads to sophisticated methods like sweep-line principle and convex hull algorithm. It is a very good example of how we can build up a topic step-by-step. We need to rely on various knowledge from mathematics at school, most importantly the Cartesian coordinate system.

3.5. Unit 5: Complex data structures, string algorithms

The final unit is dominated by data structures and contains some topics which are not required even for the IOI. Teaching data structures like Segment Tree, Fenwick Tree, Trie or Sparse Table with discovery learning is quite difficult, we have not researched this area extensively yet. Currently our pedagogy goes with describing and visualizing them on examples, and having the students work out the implementation for deeper understanding. The emphasis is on customizing them and using them in various new scenarios. A great example is using the Trie for binary numbers, e.g. finding the pair of numbers with maximal XOR value in a given set.

We also aim to capacitate the students to describe the data structures that they need to solve a certain problem, in terms of its operations and their maximum complexity. The next step is designing such a data structure, which is usually adapting a known data structure appropriately.

Code	Name	Description
SEG TREE	Segment Tree	The Segment Tree data structure for updating and querying certain computed values in a range.
BI TREE	Binary Indexed / Fenwick Tree	The Fenwick Tree data structure as an alternative to segment trees.
TRIE	Trie, Suffix Tree, Suffix Array	Data structures for storing and searching text corpora: Trie, Suffix Tree, Suffix Array.
Z, KMP	Z-algorithm, Knuth-Morris-Pratt	Advanced string pattern matching algorithms: Z, KMP algorithm.
FLOW	Network Flows	Modell problems as network flows, minimum cut maximum flow algorithm.
GAUSS	Gaussian Elimination	Solve a system of linear equations.
LCA	Lowest Common Ancestor	Finding the lowest common ancestor of tree vertices, and its applications.
SQRT	SQRT Decomposition	The Square Root Decomposition problem-solving technique. Mo's algorithm.
ST, RMQ	Sparse Table, Range Minimum Query	Sparse Table for solving the Range Minimum Query problem, plus further applications.

Table 8: Modules of the fifth unit

The unit also contains the topic of Network Flows, solving a system of linear equations by Gaussian elimination, and Square Root Decomposition as a problem-solving principle. All of them appear in some very hard tasks, with tricky applications. Computing the Lowest Common Ancestor and Range Minimum Query are connected topics, furthermore, they can be related to data structures in this unit.

Finally, we included some advanced string processing algorithms in this unit, which are mostly about efficiently searching a pattern in a text. However, they can be used for many different problems, like counting the distinct substrings of a string. Together with the Trie, Suffix Tree and Suffix Array data structures, they form a whole toolset to tackle tasks with strings.

Code	Introduction	Reinforcement	Synthesis
SEG TREE	UVa - 12532. Interval Product	CF - 380C. Sereja and Brackets, CF - 474F. Ant Colony	CF - 524E. Rooks and Rectangles CF - 242E. XOR on Segment
BI TREE	SPOJ - INVCNT	CF - 61E. Enemy is Weak	SPOJ - DCEPC206
TRIE	SPOJ - PHONELST	CF - 455B. A Lot of Games	ICPC - 4682. XOR Sum
Z, KMP	CC - KAN13C	CC - TASHIFT, CF - 126B. Password	SPOJ - DISUBSTR
FLOW	SPOJ - POTHOLE	UVa - 820. Internet Bandwidth	CF - 546E. Soldier and Traveling
GAUSS	SPOJ - XMAX	TIMUS - 1042. Central Heating	
LCA	HR - Kth Ancestor	TIMUS - 1471. Distance in the Tree	CF - 342E. Xenia and Tree
SQRT	SPOJ - GIVEAWAY	CF - 13E. Holes, CF - 86D. Powerful Array	CF - 342E. Xenia and Tree

Table 9: Problems for the fifth unit

4. Following a topic throughout the curriculum: Dynamic Programming

4.1. Our approach to DP

Many educational materials introduce Dynamic Programming (DP) using recursion, as a way of overcoming the time complexity caused by the curse of recursion. We would like to show an alternative way here, without claiming any of them better. Erdősné [21] and Forišek [22] give an excellent overview of the place of DP in popular algorithm textbooks, and rightfully argue that those books do not provide a good approach to present the DP paradigm to secondary school students. Both papers suggest teaching DP after recursion, Forišek [22] even gives a strategy to transform a top-down recursive solution to a bottom-up DP solution. Király [18] suggests that DP should be taught only after recursion and backtracking and greedy algorithms at an advanced stage of programming knowledge. Independently from the three mentioned authors, we selected very similar problems and almost the same order of them. There is one crucial difference: our approach starts with the bottom-up strategy. We agree with Erdősné [21] that the LOGO language can provide very solid grounds for recursion at a young age, but unfortunately in recent years we see a lot of children who start learning algorithmic programming without doing any LOGO before. This is one reason why we do not wish to rely on recursion.

In our scenario, there are young kids, for whom arrays are generally easy to understand, while functions are harder, recursive functions even more. Since we teach programming and algorithms together, and we consider the features of the programming language as tools for our algorithms, we can bring up the concept of DP with very simple problems, even before teaching functions. Regarding the approach of Forišek [22], another aspect of our method is that we do not start with presenting a method of problem-solving, but give a problem to the children, in which the single way of succeeding is the solution that we intend to teach. For this, we need problems where they don't have another choice, but to follow the way the teacher wants to follow. If we started with Fibonacci numbers, expecting children to first come up with the recursive solution and then figure out the steps to convert it to a bottom-up DP, our educational goals might be "screwed" by smart kids, who immediately solve it with an array - which is quite a natural solution.

We aim to introduce DP as a way of solving a task by "filling a table". It sounds much more innocent than "decomposing to subproblems", even though that is happening under the surface. We will include some formulas to show how the DP tasks get more and more complex as we advance in the curriculum.

4.2. First steps

A simple introductory task is listed above as *M - Kincsek a hegyoldalon (Treasures on the hillside)*, in which there are treasures on a grid, and we need to collect as many as we can, only moving to the right and down, starting from the upper left corner. The idea of calculating the maximum amount of treasures we can take to each cell comes naturally. If not, our pedagogy involves being ready to give good hints. In this case, we usually present a complicated example on paper and ask the students to solve it by hand, during which they most likely come up with the desired method.

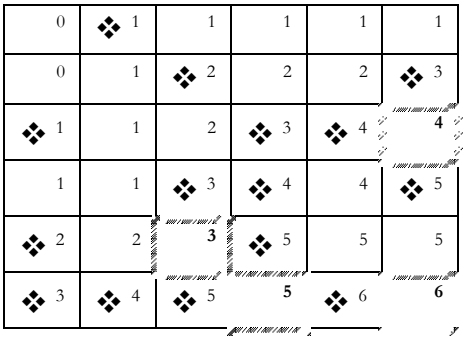


Figure 2: A concrete example of calculating the most collectable treasures for each cell

The formula which lies beneath this problem goes as follows:

```
DP[i,j] = max(DP[i-1,j], DP[i,j-1]) + T[i,j]        (1)
DP[i,0] = 0, DP[0,j] = 0
```

For practice, we use another "collect points on a grid" type exercise, *M - Pontgyűjtő verseny (Point collecting contest)*, where the possible movements are different. As a reinforcement, we propose a similar, but a bit more difficult task, *M - Benzin (Gasoline)*, which also includes constructing the optimal path. A quite hard task belonging to this group is *CF - 429B. Working out*, it can be used later to refresh the knowledge. We called this category PATH DP, expressing that we are usually searching for an optimal path to the destination.

Starting a bit later, but parallel with this thread, we analyze simple two-player combinatorial games and construct optimal strategies by determining the winning and losing positions. This is also a great opportunity to start the exploration of this world offline, with actually playing some simple games and finding their winning strategy without a computer.

A very basic problem is *HR - Game of Stones*, where two players take away 2, 3 or 5 stones from a pile in alternating turns, and the one unable to take, loses. The task is to tell who will win starting from various number of stones if both players play optimally. The problem could be generalized with different allowed moves. The solution programmatically comes down to deciding for every `i` number of stones increasingly, whether it is "good" to leave `i` stones, based upon that we know the previous answers. The notion of winning and losing states and their properties can be formulated at this point. The formula here would look like this:

```
DP[i] = not(DP[i-2] or DP[i-3] or DP[i-5])          (2)
DP at negative values treated as false
```

Our reinforcement task in this topic is a conceptually simple, but programmatically complex, two-dimensional game played on the chessboard, *HR - A Chessboard Game*. In this game both players move one token taking turns, they can make the knight moves that decrease the sum of coordinates. With this, the dynamic programming nature of game analysis becomes clear, and students also face a problem, where the order of computing the DP table elements is not straightforward. If they have a firm understanding of recursion, we can show them the power of recursion with memorization (called memoization). The below expression describes the computation in this task:

```
DP[i,j] = not(DP[i-2,j+1] or DP[i-2,j-1] or        (3)
              DP[i-1,j-2] or DP[i+1,j-2])
DP outside the table treated as false
```

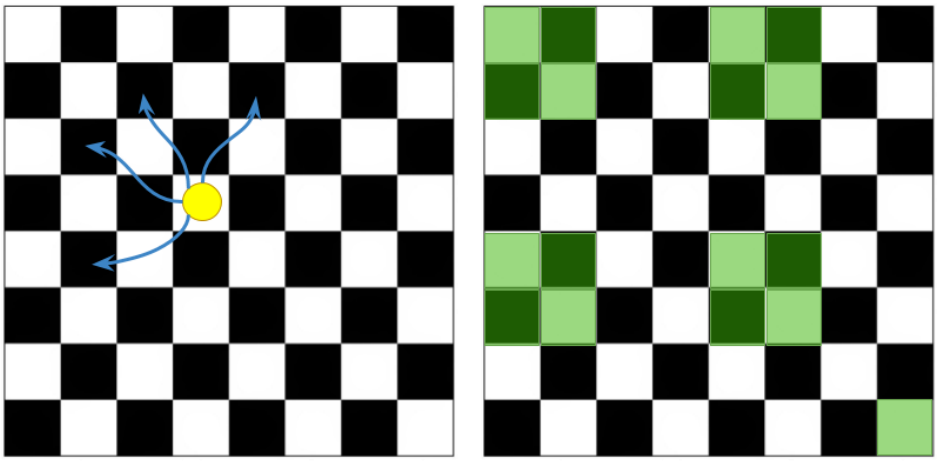


Figure 3: The moves and winning fields of the chessboard game

The synthesis problems for this module should occur much later, but we want to mention here that the task *M - Fehér és Fekete korongok (White and Black Tokens)*. There is a row of white and black tokens in this game, a player in one move can take a token from either the beginning, or the end of the row, and both players have to maximize their white tokens. This task is a great opportunity to first come up with a solution that constructs a 2D array, where the problem itself involves only a sequence. Here the states of the game guide us to the 2D data structure, and this motif is very important in the more advanced tasks of the LCS DP module.

There is an important connection with the Combinatorics module of this unit, namely that DP is often the method to solve a combinatorics problem. Calculating elements of the Pascal-triangle can be viewed as a DP task as well. An excellent problem involving a 1D array filling is *M - Lépcsők (Stairs)*. The question is how many ways you can go up N stairs if you can take steps of at most K stairs. We suggest scheduling this task soon after the first DP problems, in parallel with other PATH DP tasks. The bottom-up nature of DP is very clearly visible, as we count the ways of reaching each stair in order.

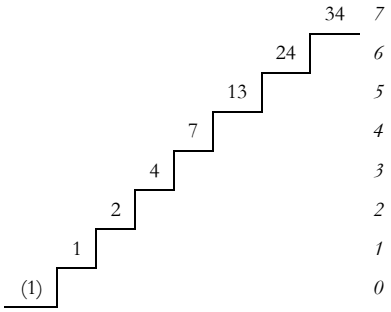


Figure 4: The number of ways to reach each stair, with maximal step size K=3

The formula of the solution is the first of this kind that doesn't have a closed form:

```
DP[i] = DP[i-1] + DP[i-2] + … + DP[i-K]          (4)
DP[0] = 1, DP at negative values treated as 0
```

Besides DP, we plan to introduce recursion approximately at the same time, or just a little bit shifted. The close relation between these strategies should be enlightened and demonstrated thoroughly. The two topics running in parallel helps to strengthen their grounds. Furthermore, DP in

some cases can be more elegantly performed using recursion with memoization, particularly when the computing order is not trivial.

4.3. Exploring the power of DP

After feeling the taste of DP with problems where transitions are steps in a game-like scenario, the students will meet less obvious DP problems in different wrapping. We named this module Knapsack DP after the very representative Knapsack problem.

We start with a similar, but much easier task, the Coin Change problem, where the question is what amounts of money can be paid using some set of banknotes (not asking for the minimum number of notes yet). It can be modeled with a single 1D boolean array, and deciding payability for each integer amount increasingly is an intuitive idea (programmatically very similar to *HR - Game of Stones*). With `B[1..N]` denoting the values of coins, we can formulate the solution as follows:

```
DP[i] = DP[i-B[1]] or DP[i-B[2]] or … or          (5)
        DP[i-B[N]]
DP[0] = true
DP at negative values treated as false
```

Minimizing the number of notes is the next step and it is a representative example of greedy not working. The *M - Bélyeg (Stamps)* task is essentially this problem.

The Knapsack problem, *SPOJ - KNAPSACK* has a special role in our DP curriculum, this is the first time when we use a non-trivial task decomposition with two variables. There are `N` items with different sizes and values, and we have to fit as much as we can into a backpack with size `K`. We calculate the maximum value for each backpack size when considering the inclusion of the items one by one. Here we present the underlying formula, where `V[1..N]` are the values, and `S[1..N]` are the sizes of the items:

```
DP[i,j]= max(DP[i-1,j], DP[i-1,j-S[i]] + V[i])    (6)
DP[i,0] = 0, DP[0,j] = 0
DP at negative values treated as -∞
```

Many other real-life problems can be solved with this method. The two reinforcement tasks we suggest are such examples. In *M - Munkagépek (Machines),* distributing jobs between two machines can be reformulated to a special coin change problem. In the *M - Vásár (Sale)* problem, we have to maximize the profit of a merchant, and it can be reduced to a Knapsack problem.

The knowledge of DP is essential in some graph algorithms in our third unit. Two very common shortest path algorithms, Bellman-Ford and Floyd-Warshall algorithms are two different DP solutions to this problem. We can make use of this fact very well in our educational program, students who have very solid grounds in DP can discover Bellman-Ford and Floyd-Warshall themselves. To induce this, we can tell them to try finding the shortest paths with DP. If necessary, we can be more specific: in the case of Bellman-Ford: do a DP on the number of edges in the path. Floyd-Warshall is much trickier, the DP is done on the vertices inside the path. An excellent task that can help with this discovery is *CF - 295B. Greg and Graph*. In our third unit problem set, there is another task which is a great synthesis of DP, combinatorics, and directed acyclic graphs (DAG): *HR - Kingdom connectivity*, in which you have to find the number of different ways to go between two vertices of a directed graph.

4.4. Advanced problems

Since DP has so many applications, we keep on revisiting it with harder and harder tasks. It is not easy to categorize them, so we took two representative problems to symbolize these modules.

To solve the Longest Common Subsequence problem, we create a 2D array, where each cell corresponds to the subproblem taking the first `i` and `j` elements of the two sequences. For sequences `A` and `B`, the dynamic programming goes as follows:

```
DP[i,j] = max(DP[i-1,j], DP[i,j-1],                    (7)
              DP[i-1,j-1]+1 if A[i]=B[j])
DP[i,0] = 0, DP[0,j] = 0
```

We included various other exercises where we see this or a similar pattern. There is *M - Jelek (Signs)*, which is essentially the longest repeated substring problem that can be solved with DP in $O(N^2)$. In *M - Rúd felvágás (Stick cutting),* we are looking for the cheapest way of cutting up a stick to pieces. The solution to that problem is easier formulated with recursion, so it can be an example of memoization. *CF - 607B* is about palindromic substrings and it requires a very good understanding of the pattern where the subproblems are ranges in some sequence. Here we would like to point out that previously we mentioned a problem, *M - Fehér és fekete korongok, (White and Black Tokens*), which falls also into this category and we included it in GAME DP because there the subproblems are states of a game, so it is a good precursor to this module.

The Longest Increasing Subsequence (LIS) problem can be developed very well and there are numerous nice exercises on this topic. The $O(N^2)$ solution is a good first step, but here we want to focus on reaching the $O(N \cdot logN)$ solution combined with binary search. *M - Konténeroszlopok (Container Columns)* is a greedy task that can be viewed as the dual problem of it: divide a sequence into a minimal number of decreasing subsequences. In the greedy process, the ordered nature of subsequence endings can be observed, and the binary search is straightforward. What is not straightforward is that we calculate the length of the LIS too, which gives a lower bound for the result. This duality theorem also certifies that LIS is so beautiful that everyone should see. But we only uncover it after solving a task which can be reduced to finding the LIS: you have to build a tower using the maximum number of cubes, given some cubes with sizes and weights, and you can only place a smaller and lighter cube on top of another, *M - Kockákból legmagasabb torony (Highest Tower of Cubes)*. Sorting by one property is the first idea, after that it comes down to computing the LIS. There is a very hard task in this topic, *CF - 650D. Zip-line*, in which a firm understanding of the above is necessary, but no advanced data structures.

We call Bitmask DP the method when the subproblems we solve correspond to all subsets of a set. We usually represent a subset with a bit vector, that is usually stored in an integer. Transitions in this form of DP generally involve adding or removing one element of the subset, which can be done by bit manipulations, bitwise operators. An example problem would be *M - Vásárlások (Purchases)*, where we need to minimize the money spent on certain items, given their prices in different shops, with the constraint that we buy at most one item in each shop. The states of the DP, in this case, would be the `K` number of shops considered and the `S` subset of items bought so far. This type of DP can be connected to the inclusion-exclusion principle in some problems, for example *CF - 102128B. Cake Tasting.*

We applied DP with binary search, DP on ranges and DP on subsets with bit manipulations in the modules above, and there are numerous other scenarios that we haven't covered here. The topic of DP is very deep, there can be extremely difficult problems which are "only" DP. In our fifth unit, driven by data structures, we can see traces of DP in the construction step of many data structures (e.g. RMQ Sparse Table, Binary Indexed Tree, LPS table in KMP), so dynamic programming is a core concept throughout our curriculum. In this chapter, we gave an overview of it, showing the possibility of developing deep knowledge over a long period and many exercises.

5. Conclusion

In this paper, we described a possible curriculum for computer programming talent education in high schools. However, we consider it as work in progress, we will reflect on it and improve it based on experiences from putting it to practice. The main goal of the author's research is to create a system for computer science talent education similar to the existing one in mathematics organized by The Joy of Thinking foundation [7]. The curriculum was designed to respect the principles of Lajos Pósa's pedagogy in mathematics and is intended to form as a basis for our problem-based learning methodology in algorithmic programming.

Bibliography

1. J. Győri, P. Juhász: *An extra-curricular gifted support programme in Hungary for exceptional students in mathematics*. Teaching Gifted Learners in Stem Subjects. Routledge, London (2017) 89–106. DOI: 10.4324/9781315697147-7
2. S. Halim: Competitive Programming 3. Lulu Independent Publish (2013)
3. *Problemset, Codeforces*. (2020) https://codeforces.com/problemset.
4. W. Van Joolingen: *Cognitive tools for discovery learning*. International Journal of Artificial Intelligence in Education. Vol 10 (1999) 385–397.
5. A. F. Borthick, D. R. Jones: *The motivation for collaborative discovery learning online and its application in an information systems assurance course*. Issues in Accounting Education, Vol. 15(2) (2000) 181–210. DOI: 10.2308/iace.2000.15.2.181
6. S. L. Finkle, Ll. Torp: *Introductory Documents*. Illinois Math and Science Academy. (1995)
7. *The Joy of Thinking Foundation*, Hungary. http://agondolkodasorome.hu/.
8. J. A. Bibergall: *Learning by discovery: Its relation to science teaching*. Educational Review. Vol. 18(3) (1966) 222–231. DOI: 10.1080/0013191660180307
9. D. Katona, G. Szűcs: *Pósa-Method and Cubic-Geometry – A Sample of a Problem Thread for Discovery Learning of Mathematics*. Differences in Pedagogical Theory and Practice. (2017) DOI: 10.18427/iri-2017-0079
10. P. Juhász: *Hungary: Search for Mathematical Talent*. The De Morgan Journal. Vol 2(2) (2012) 47-52.
11. L. Pósa: *Matematika táboraim*. Természet Világa. Vol. 132, N°3. http://www.termeszetvilaga.hu/tv2001/tv0103/posa.html
12. M. Forišek: *IOI Syllabus*. (2018) https://ioinformatics.org/files/ioi-syllabus-2018.pdf
13. A. Laaksonen: Competitive Programmer's Handbook. (2018) https://cses.fi/book/book.pdf
14. *CP-Algorithms*. https://cp-algorithms.com/
15. *Geeks for Geeks*. https://www.geeksforgeeks.org/
16. A. M. Dehghan: *Algorithm Gym: Data Structures*. Codeforces blog. (2015) https://codeforces.com/blog/entry/15729
17. A. M. Dehghan *Algorithm Gym: Graph Algorithms*. Codeforces blog. (2015) https://codeforces.com/blog/entry/16221
18. S. Király: *How to teach computer programming if our goal is the International Olympiad in Informatics*. Teaching Mathematics and Computer Science. Vol. 9, no. 1 (2011) 13-25.

19. Á. Erdősné Németh: *From LOGO Till Olympiads - Talent Management In Grammar School.* Ph.D. Dissertation, Eötvös Loránd University, Doctoral School of Informatics, Budapest (2019) (in Hungarian). DOI: 10.15476/ELTE.2019.007

20. Á. Erdősné Németh, L. Zsakó: *Grading Systems for Algorithmic Contests.* Olympiads in Informatics. Vol. 12 (2018) 159–166. DOI: 10.15388/ioi.2018.13

21. Á. Erdősné Németh, L. Zsakó: *The Place of the Dynamic Programming Concept in the Progression of Contestants' Thinking.* Olympiads in Informatics. Vol. 10 (2016) 61–72. DOI: 10.15388/ioi.2016.04

22. M. Forišek: *Towards a better way to teach dynamic programming.* Olympiads in Informatics. Vol. 9 (2015) 45–55. DOI: 10.15388/ioi.2015.05

23. P. Szlávi, L. Zsakó: *Methodical programming: Programming Theorems*. Eötvös Loránd University, Faculty of Science, Department group of Informatics, Budapest (1996) (in Hungarian).

24. *HackerRank*. https://www.hackerrank.com/

25. *CodeChef*. https://www.codechef.com/

26. *CS Academy*. https://csacademy.com/

27. *Sphere Online Judge*. https://www.spoj.com/

28. *UVa Online Judge*. https://uva.onlinejudge.org/

29. *PEG Online Judge*. https://wcipeg.com/

30. *Mester*. http://mester.inf.elte.hu/

31. *ICPC Live Archive*. https://icpcarchive.ecs.baylor.edu/

32. *Timus Online Judge*. http://acm.timus.ru/

Authors

NIKHÁZY László

Eötvös Loránd University, Faculty of Informatics, Department of Media and Educational Informatics, Budapest, Hungary, e-mail: laszlo.nikhazy@gmail.com

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 1. 2020.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.1.399

License

Motivational Tools for Learning Programming in Primary Schools

VOŠTINÁR Patrik

Abstract. Studying pedagogical sciences for Computer Science subject nowadays does not meet with great interest of students in Slovakia. Education of young generation in this area is strongly affected by small number of well qualified teachers as well as lack of high-quality teaching materials resulting to deplorable knowledge of the pupils. The very same situation is also with preparation of future teachers. They have only limited possibilities of direct contact with pupils during their studies, what leads to weak experience of the future teachers and low motivation of the children for Computer Science subject. This contribution gives a positive example of extracurricular educative activity for pupils from lower-secondary schools, where students of Teaching of Computer Science program and Applied Informatics program were involved as lecturers or tutors. In this activity we used various motivational tools for teaching programming and Computer Science as well. The contribution describes our experience with these tools for programming, which we used in our extracurricular activity – so called "Informatics club" and also in the two primary schools and three secondary schools in the region of Banská Bystrica. Motivational devices, which were used are: education board bbc micro:bit with accessories (IoT set, gamepad, micro:bit cars), didactic devices – robots (Airblock, mBot Ranger, Phiro Pro) and board game Scottie Go!.

Keywords: learning aid, extracurricular activity, bbc micro:bit, STEM

1. Introduction

Almost 40 years have passed since the integration of computers came into education. During this time, the integration of information and communication technology (ICT) has progressively evolved from the using of simple learning programs in the classrooms to teaching management systems [1]. Today's children have very close relation with games and applications on smartphones and other mobile devices [2]. With these devices it is easier to develop pupils' computational thinking. Mobile devices can be also used to control robots. Recently, the issue of educational robotics has been resonating in many countries at all levels of schools. Many researches show positive result of their research on the increasing interest of young people to study technical and natural science at colleges and universities thanks to the pleasant experience of building and programming robots [3,4]. It helps to motivate students to learn science and technology, thus enhancing problem solving skills [5]. The jobs with Computer Science are now very popular, unfortunately in almost all around the world are missing specialists for Computer Science. The same situation is also taught in subject Informatics in Slovak Republic. The solution can be motivation students to study Computer Science from the beginning - primary schools.

In this article we are focused on explanation of our computer science extracurricular activity in our department and explanation of various types of motivation devices for computer science teaching.

2. Computer science extracurricular activity

In our Department of Computer Science FNS UMB we are organizing second academic year extracurricular activity for pupils from primary schools. Based on our experiences from the previous academic year 2017/2018, which were unambiguously positive, we decided to use in this activity more tasks with educational robots. We have found that pupils are interested in programming, especially when they can catch their results, for example in the case of LEGO

robots. In this academic year 2018/2019 we have this activity twice in a week for 1,5 hours. The students of Teaching of Computer Science program and Applied Informatics program were involved as lecturers or tutors in this activity. Each of our student teach 2-3 children from the primary school. The main focus of this activity is to obtain additional teaching experience for our students and in-crease motivation for studying Computer science (programming). This year we have 49 pupils from different primary schools and 27 students from our department.

More information about the Computer Science extracurricular activity are in page http://www.fpv.umb.sk/katedry/katedra-informatiky/kruzky-organizovane-ki-umb-pre-ziakov-zs-a-ss/informacie-o-kruzku-zs.html .

3. European Researchers' Nights

European Researchers' Night is the festival of science, which is a yearly organized event throughout Europe. Last year (2018) our department attended this event in Europa Shopping Center (ESC) in Banská Bystrica. In our scientific stand we presented our department - scientific research in our department and didactic devices used in our Computer Science extracurricular activity for primary schools. We presented programming mobile applications in App Inventor, programming didactic devices like Airblock, mBot Ranger, Phiro Pro and BBC micro:bit. We had prepared simple programming tasks for each of didactic devices (each task had to be solved in couple of minutes). Example of tasks were to create mobile app with button, which change background color of screen, create simple environment for moving various type of robots. We used the observation method for this activity. The result of observation was that almost all of the pupils (also their teachers) were very excited. Many of them have signed up for our free Computer Science extracurricular activity for primary schools. Figure 1 shows the Airblock and micro:bit programming in ESC.

Figure 1: Presenting robots on European Researchers' Nights

4. BBC micro:bit

The educational board BBC micro:bit consists of 25 individually programmable LEDs, 2 programmable buttons (A,B), physical connection pins, light and temperature sensors, motion sensors, compass, battery socket, radio and Bluetooth antenna [6]. In the Figure 2 is educational board BBC micro:bit.

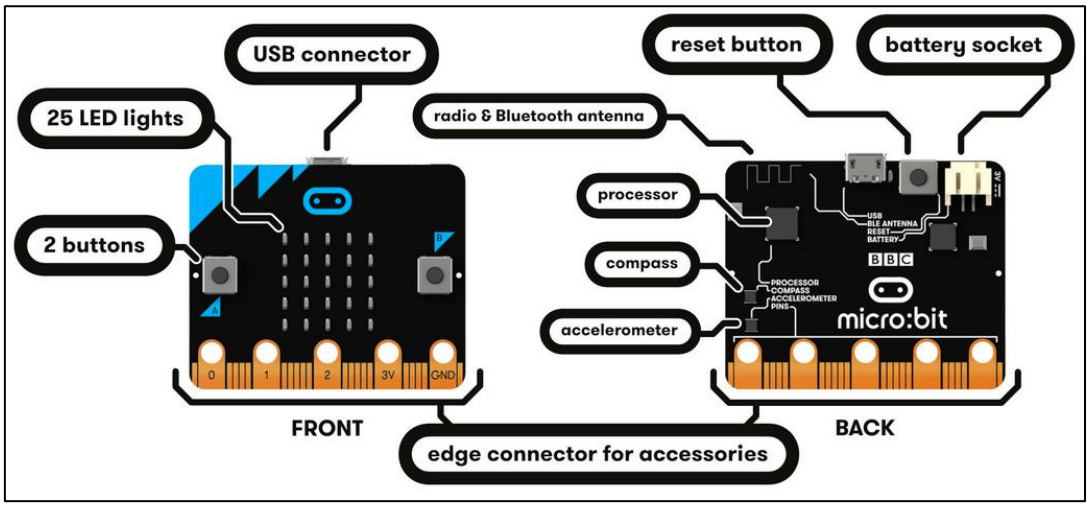


Figure 2: Educational board BBC micro:bit (microbit.org)

Micro:bit is a low-cost device that was assigned to every 11-12 year old in the UK in 2016 [7]. The main microcontroller on the micro:bit is an ARM M0 chip, similar to those found in many mobile devices [8]. Micro:bit has a lot of accessories like gamepad, GPIO Expansion, Sensor Board, 1.8inch colorful dis-play, etc. For programming educational board micro:bit it is possible to use Microsoft MakeCode editor (programming with blocks), JavaScript and MicroPython.

4.1 Gamepad and Micro:bit cars

The most interesting accessories for micro:bit are gamepad and micro:bit cars. Which child wouldn't want to develop behavior of their own gamepad to control the car? There are many types of gamepad and micro:bit cars. We chose the cheapest gamepad in our shops and as the car we chose AlphaBot2. For programming gamepad and cars, it is necessary to add extension to Microbit Makecode (according wiki of the device). The task for pupils were at first to program all of the gamepad buttons (7 buttons) and behavior of joy-stick movement. After this task were finished they had to create communication between gamepad and car (radio communication - sending integer or string values on the same frequency). After they received message from gamepad, they had to program movement of the car. Figure 3 shows the gamepad with source code of radio communication.

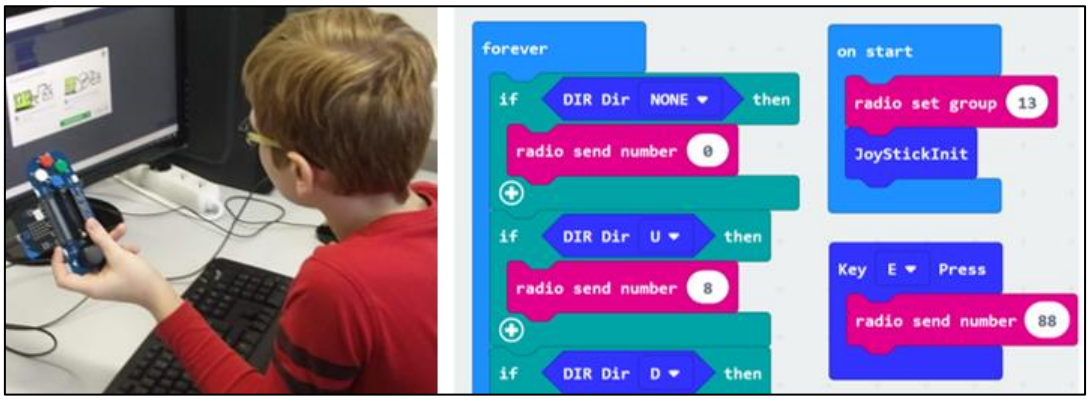


Figure 3: BBC micro:bit with gamepad

4.2 IoT extension

Another important extension for micro:bit is IoT extension. With this extension it is possible to use various types of integrated circuits. This extension can replace microcontroller Arduino. In the Figure 4 there are IoT extensions for micro:bit.

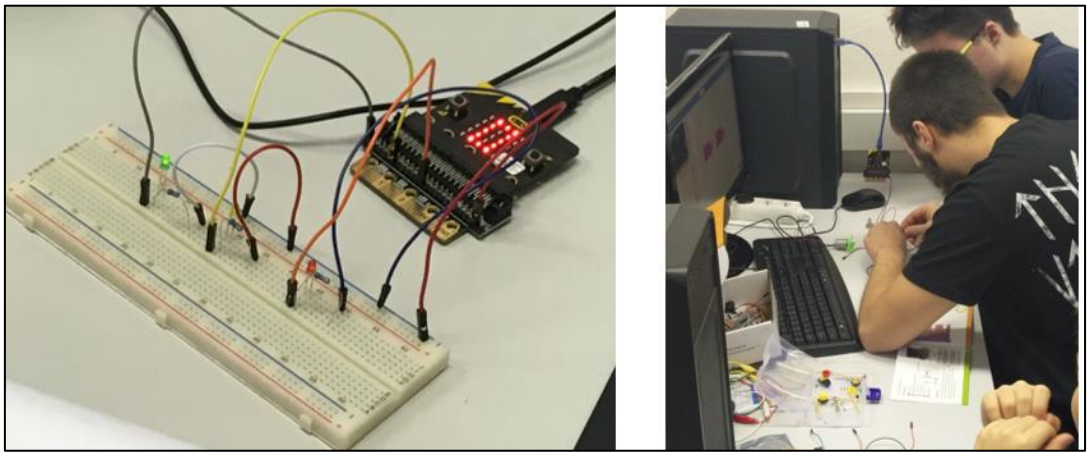


Figure 4: BBC micro:bit with IoT extension

5 Didactic devices – robots

A physical computing in teaching process can be reached with using programmable robots. It is nothing special, if these devices are found in preschool or primary schools [9]. Nowadays there exist a huge number of programmable robots e.g., LEGO Mindstorms robots, Bee-Bot, ProBot, Sphero robots, Ozobot, robots created by Makeblock (Codeybot, Airblock, mBot), Phiro Pro, etc. Each of these robots have different advantages and disadvantages. The using of robotics in education hasn't very long history, but we can't ignore the fact, that robotics is constantly and rapidly developing and in the future will be robots part of our daily life [10]. We will concentrate on using Airblock, mBot Ranger and Phiro Prot in education.

5.1 Airblock drone

Airblock is the programmable drone, which can be transformed into a flying drone, hovercraft, water hovercraft, spider etc. It is indoor-friendly, because it is made from engineering foam (1 main module - CPU and 6 dynamic magnetic hexagons). For programming Airblock it is possible to use mobile devices with iOS or Android operating system [11]. Programming in this environment is based on block programming like in Scratch. Pupils can create their own environment of the controller (it is possible to choose various predefined commands like buttons, power switch command, gamepad, etc.). Except to basic robot motion and controls blocks (if, while, etc.) it is also possible to use blocks from math, event (shake, tilt tablet, etc.), detect (gyro value, temperature, etc.) and display (LED colors).

Example of task – "Draw air cube with Airblock drone"

This task is appropriate after completing the tasks of rendering 2D geometric shapes (square, rectangle) with drone. We used this task for independent work in a team or individuals - pupils have to find results on how to draw 3D shape - cube in the air. Pupils have to find appropriate commands based on their previous experience (tasks with drawing 2D shapes). In case that pupils cannot find an answer, the teacher can help with blocks, which can they used. Based on our experience, this task is simple for pupils, because they had experience with drawing 2D shapes. Figure 5 shows result of this task.

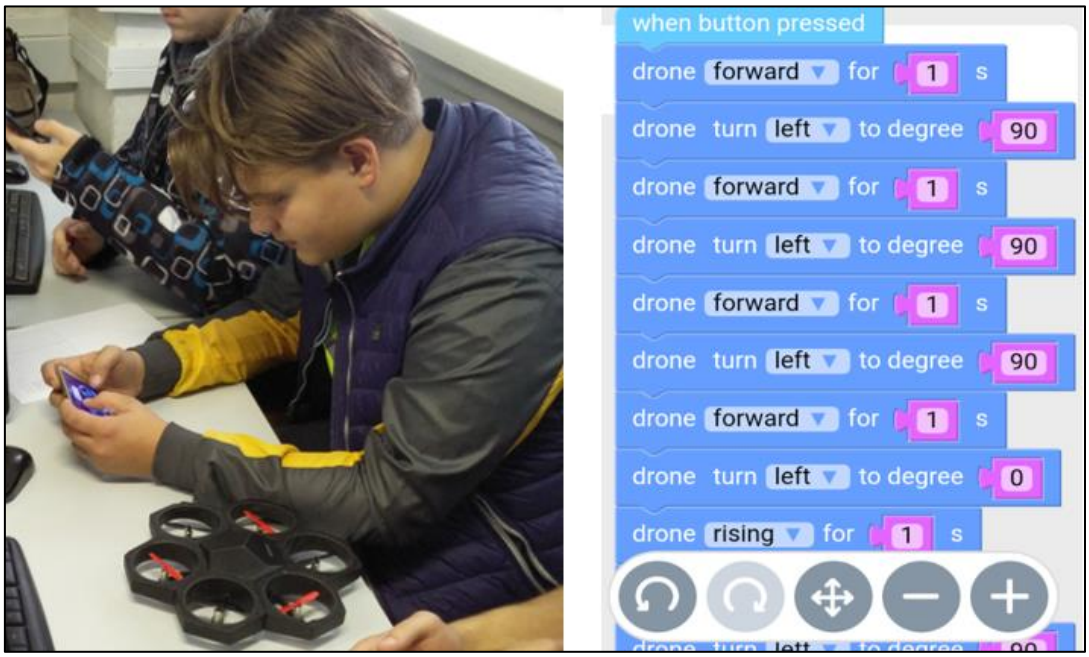


Figure 5: Airblock code for drawing cube [11]

5.2 mBot Ranger

Robot mBot Ranger is an advanced version of mBot. It can be transformed into 3 forms (All-terrain Off-road Land Raider, Super Speedy Dashing Raptor, Self-balancing Nervous Bird). This robot is created by Makeblock, the same company as Airblock drone, this mean that for programming you can use the same mobile app like for Airblock (just change the robot version). mBot Ranger contain following sensors: Light sensor, Temperature sensor, Sound sensor,

Ultrasonic sensor, Line Follower sensor and Gyroscope. The main-board of this robot is based on microcontroller Arduino, that means it can be programmed also in Arduino IDE environment.

Example of task – "Variables"

Commands of mBot Ranger consists of various basic programming structures such as variables, procedures, conditions, loops, etc. Figure 6 is mBot Ranger with a source code of programming task for practicing variables. The task is:

"After the button "Random even-odd" is pressed create a variable, which will be initialized to a random integer from 1 to 10. If the variable is even, then set ALL LEDs on mBot to color green and run forward at speed 100. If the variable is odd, then set all LEDs to color red and run backward at speed 100."

This task is appropriate for practicing the topic variables (and also the repetition of conditions and mathematical operations - random number, even/odd). The children could solve this task individually or in the team. After testing this task in our extracurricular activity for primary school children we find that this task was easy for children. All of them known how to solve this task.

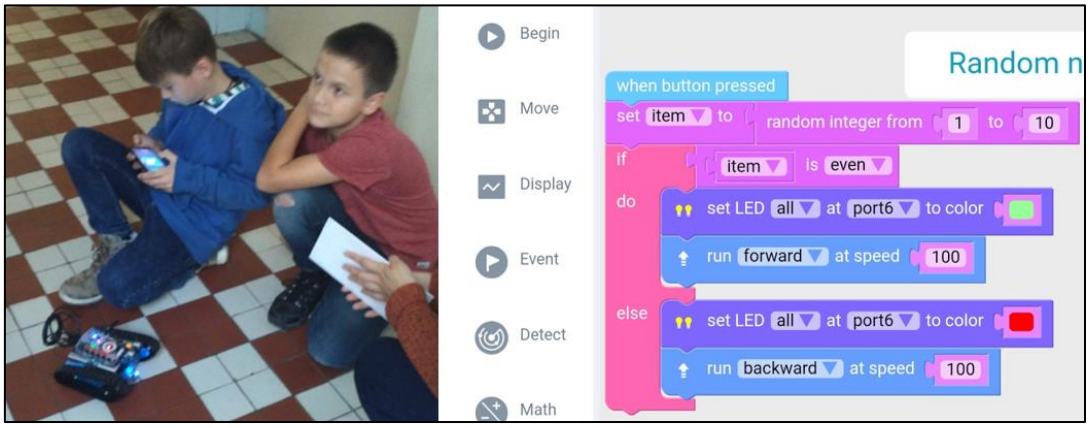


Figure 6: mBot Ranger programming random number

5.3 Phiro Pro

Phiro Pro is a LEGO compatible programmable robot designed for young people ages 9 to 18 (recommended). Phiro Pro can be used with programming in computers (Scratch, Snap4Arduino), mobile devices (Pocket Code for iOS and Android), programming by sequential keys (located on the top of the Phiro Pro) and Swish Cards (little cards like credit cards, each card means one command, it is possible to use one or more cards together) [9]. Phiro Pro offers functionality to code basics robot commands such as line follower, edge detection, obstacle avoider, etc. The most interesting tasks for Phiro Pro are with Swish Cards.

Example of task – "Swish Cards - movement"

"Find appropriate Swish Cards for moving robots to one step forward, one step backward, turn right, turn left, left LED red, right LED red. After that insert cards to Phiro Pro in following orders: After that insert cards to Phiro Pro in following orders: one step forward, turn LED red, turn left, turn LED right, turn right, one step backward. Discuss in the small group why you used cards left LED red, right LED red."

Phiro Pro and Swish Cards are suitable for introduction to teaching programming. Pupils can learn the gradual execution of orders, basic robotics. In the Figure 7 there are photos of programming Phiro Pro with mobile application Pocket Code and Swish Cards.

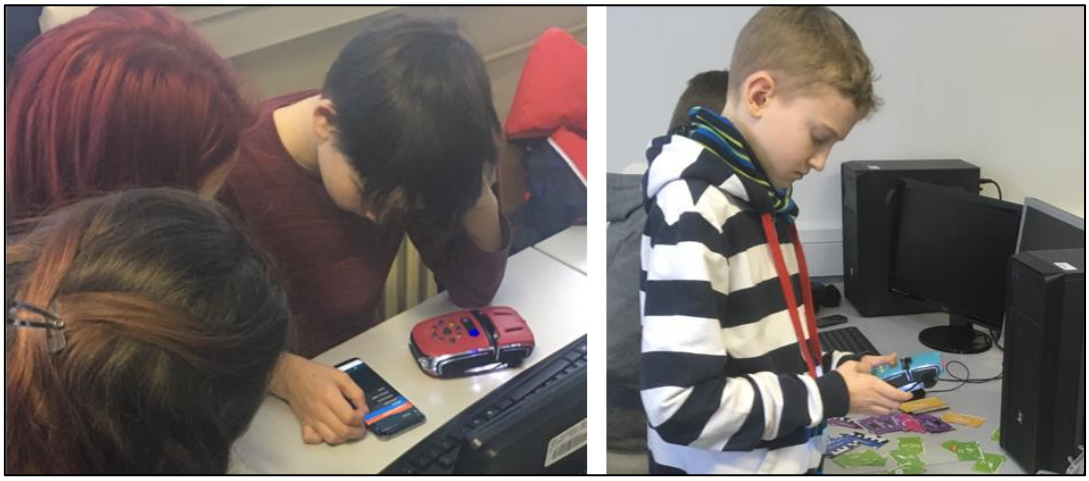


Figure 7: Phiro Pro programming with Pocket Code and Swish Cards

6 Programming board game Scottie Go!

Scottie Go! is education game to teaching programming. It consists of 179 blocks of code, which have to be used to solve 91 tasks. The blocks of the code have to be joined together like puzzle. The tasks are in special mobile app (iOS, Android and Windows app). Pupils must solve tasks in board game, then to use camera of mobile device and scan their results. The app evaluates the pupil's solution with 1-3 stars (each task has multiple solutions). In the Figure 8 there are photos of programming Scottie Go!

Figure 8: Scottie GO! programming

The disadvantage of this board game is that there is no possibility to create own examples, only tasks from the mobile application Scottie Go[1]!. The game allows players to acquire basic programming concepts such as loops, conditionals, variables or functions. It can be used as a basic

[1] https://play.google.com/store/apps/details?id=com.netictech.scottiego&hl=sk

tool to introduce and teaching programming. The main advantage of this game is, that it allows children to work in teams without teacher help. The mobile app describes each block, which should be used in a game.

7 Methodology and research

All of the didactic devices mentioned in chapter 3 and 4 are used in our Computer Science extracurricular activity for primary schools in the Department of Computer Science FNS UMB (49 pupils between 10-15 years old). We also tested using these devices in the teaching process during 2018/2019 in primary school Dominika Savia in Zvolen (14 pupils, 14-15 years old). As a method of measuring data, we used observation and questionnaires. In the questionnaire we focused on measuring motivation of using didactic robots (through observation and questionnaires). We asked 44 pupils (13 pupils from primary school and 31 pupils from extracurricular activity) from 63 pupils. The reason of smaller number of responders was that not all of the pupils visited each week our extracurricular activity, so we decided to collect answers only from pupils which completed all lessons. Questionnaires consist of 12 questions, where we asked gender, age, experience with programming. The important questions were regarding using micro:bit. Figure 9 shows diagram most important questions for our small research.

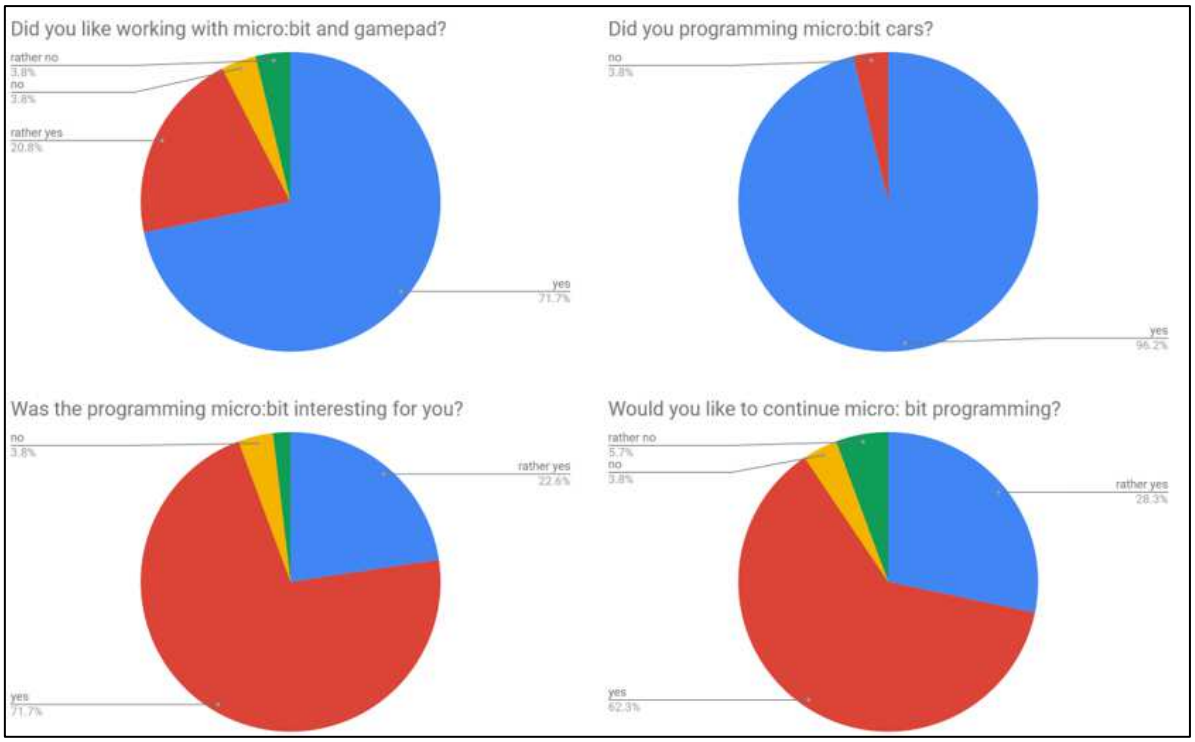


Figure 9: Questionnaires results

Answers form the question "Did you like working with micro:bit and gamepad?" showed that 71,7% students like programming educational board BBC micro:bit with gamepad. 20,8% of the students said rather yes. Only 7,6% answers wrote that they does not like programming micro:bit and gamepad.

Question "Did you programming micro:bit cars?" showed that almost all of them (96,2%) like programming micro:bit cars.

For 71,70% students were the programming micro:bit interesting, according the question "Was the programming micro:bit interesting for you?" 22,60% students wrote that rather yes, then no.

The graph from the question "Would you like to continue micro:bit programming?" showed that 62,30% want to continue with programming BBC micro:bit and 28,30% wrote rather yes, than no. Only few of the students does not want to continue with programming micro:bit.

Other question from the questionnaire were focused on "What I did not like with working with micro:bit". Almost all of them wrote that "everything was great".

8 Conclusion

In this article we are focused on motivation of the children for Computer Science subject. We described motivation devices - programmable robots which we used in our extracurricular activity for pupils from lower-secondary schools, where students of Teaching of Computer Science program and Applied Informatics program were involved as lecturers or tutors. We described devices which we most used - education board micro:bit with accessories, didactic devices – robots and board game Scottie Go! Described devices are good for motivating to studying Computer Science, but from our experience it is better when the pupils have experience with programming from Scratch before then they started using these devices.

We did small research, where we used modern educational aids, which are currently used in teaching Computer Science in the world. In our research we used observation and questionnaires. We plan to continue with our research if using these educational aids will increase interest of about studying Computer Science. The observation and the questionnaire show that usingt these educational aids incresed students interest about stydying Computer Science.

This contribution has been processed as part of the grant project Interactive Applications for Teaching Mathematics at Primary Schools, project no. 003TTU-4/2018.

Bibliography [2]

1. M. Pokorný: Kurz desatinné čísla. In Sborník příspěvků z konference a soutěže eLearning 2018. Hradec Králové : Gaudeamus, 2018, s. 15-19
2. N. Klimová, D. Horváthová: Súťaž KODU CUP ako podporný prostriedok algoritmického myslenia. In DidInfo&DidactIG 2017. Banská Bystrica: Univerzita Mateja Bela, Fakulta prírodných vied, 2017, s. 197–198
3. M. Havelka, V. Stoffová: Robotika - Stavba a programování robotů (LEGO Mindstorms NXT a EV3) 1. vyd. Olomouc : Pedgogická fakulta UP v Olomouci, 2017. 85. s.
4. V. Stoffová, O. Takáč: Robotické stavebnice v príprave učiteľov informačnej výchovy (Robot kits in teachers preparation for information education). In Havelka, M.,

[2] The data as indicated in the source are to be followed. The main principle is to make the source reachable.

Chráska, M., Klement, M., Serafín, Č. (ed.): Trendy ve vzdělávání 2013. Olomouc : agentura GEVAK s.r.o., 2013. 315-322. s. ISBN 978-80-86768-52-6 / ISSN 1805-8949

5. N. Klimová, D. Horváthová: Robot Phiro ožíva v rukách detí. In DIDINFO 2018, s. 270-277
6. P. Voštinár, N. Klimová, J. Škrinárová: Before we start Arduino. In INTED2019 proceedings : 13th international technology, education and development conference. Valencia : IATED Academy, 2019, pp. 7218-7223
7. S. Sentence, J. Waite, L. Yeomans, E. Maclead: Teaching with physical computing devices: the BBC micro:bit iniative. In Proceedings of the 12th Workshop on Primary and Secondary Computing Education (WiPSCE '17). ACM: New York, 2017, pp. 87-96
8. M. Cápay, N. Klimová: Engage Your Students via Physical Computing!. In IEEE Global Engineering Educatin Conference (EDUCON), 2019, pp. 1250 – 1257
9. V. Stoffová, M. Zboran.: Hravá forma stavby a programovania robotov na základnej škole. In: Trendy ve vzdělávání, 11, 2018, č. 2, s. 130 − 139. ISSN 1805-8949
10. V. Stoffová, M. Havelka.: Práca s robotickými stavebnicami na 2. stupni ZŠ - Zbierka riešených úloh. 1. vyd. Olomouc : Pedgogická fakulta UP v Olomouci, 2018. 66.
11. P. Voštinár, N. Klimová, D. Horváthová: The programmable drone for STEM education. In Entertainment computing - ICEC 2018 : international conference on entertainment computing (IFIP-ICEC'18). Cham : Springer Nature Switzerland, 2018, pp. 205-210

Authors

VOŠTINÁR Patrik

Matej Bel University, Faculty of Natural Sciences, Department of Computer Science, Banská Bystrica, Slovakia, e-mail: patrik.vostinar@umb.sk

About this document

Published in:

CENTRAL-EUROPEAN JOURNAL OF NEW TECHNOLOGIES IN RESEARCH, EDUCATION AND PRACTICE

Volume 2, Number 1. 2020.

ISSN: 2676-9425 (online)

DOI:

10.36427/CEJNTREP.2.1.420

License