LEARNING BY CODING ON THE BASE OF BLOOM'S LEVELS OF KNOWLEDGE

Elena Somova, Jordan Enev

University of Plovdiv "Paisii Hilendarski"
e-mails: eledel@uni-plovdiv.bg; yordan.enev@bgomedia.net
Bulgaria

Abstract: The paper represents an invariant approach for teaching how to code programs, regardless of the studied programming language. The approach is based on the currently taught main programming algorithms. The invariant and parametric parts of these algorithms are determined. Questions and assignments, related to different cognitive levels of Bloom's taxonomy, are provided, which test the learners' knowledge of invariant algorithms. The work also shows the developed software system based on this approach.

Key words: Programming Languages, Invariants in Programming, Software Learning Systems.

1. INTRODUCTION

Despite the continuous development and emergence of new programming languages, the training in programming does not change drastically. The reason for the relatively static educational content is that the training focuses primarily on the basic algorithms in programming, not the syntax of a particular language.

Many authors use six levels of Bloom's taxonomy [2] to evaluate cognitive levels of learner knowledge or to set learning objectives. Hierarchy of Bloom's levels shows different consecutive steps in acquisition of knowledge and skills – knowledge (exhibits memory of previously learned material), comprehension (demonstrates understanding of facts and ideas), application (solves problems in new situations), analysis (examines and breaks information into parts, makes inferences and finds evidence), synthesis (compiles information together in a different way) and evaluation (presents and defends opinions, judgments about information) [22].

The aim of the authors is to present an approach for training students that can be applied regardless of the taught programming language. An author software system, built on this principle will be shown, where multiple assignments are offered to the learners, gaining knowledge on the different cognitive levels of Bloom's taxonomy.

Section 2 reviews related works of other authors – the attempts to create invariant learning materials and existing training systems in programming. In Sections 3 and 4 the authors' idea of invariance of the programming algorithms, which has been

published in detail in previous papers, is presented. Section 5 discusses the types of tasks (questions and assignments) proposed by the authors, testing the different levels of the Bloom's Taxonomy. Different tasks are implemented in a training software system, presented in Section 6. The paper ends with a conclusion summarizing the contributions of the authors as well as the application of the created system.

2. LEARNING PROGRAMMING

2.1. Invariant Learning Textbooks

There are attempts to solve a similar problem in the learning information technologies (IT) and computer science fundamentals. To conduct a distance education in IT, learning materials [19] with invariant elements in 3 languages (Bulgarian, French and Lithuanian) have been created. Using the same methodology, a basic university course "Fundamentals of Computer Science" [20] in Plovdiv University and a package of textbooks for the secondary school ([1], etc.) have been created.

The idea of invariant IT training is developed in [5, 6, 18], while giving examples about invariant teaching on spreadsheets [5, 6], and about text processing in [18].

2.2. Training Software Systems in Programming

There are many training systems in programming, but only in some of them there have been attempts for independence from teaching programming languages. Crunchzilla [4], W3Schools [21] and Codingame [3] focus on the learning environment and the method of presentation of educational content, which is highly dependent on the introduced programming language.

Crunchzilla [4, 9] is implemented as an electronic "teacher" who leads a preset dialogue with the student and presents example problems associated with visualization and movement of objects on the screen (using JavaScript). The available learning resources are at different levels depending on the age of the learner.

W3Schools [21] provides training in several programming languages and technologies (HTML, CSS, JavaScript, PHP, Bootstrap, etc.) based on tutorials. The learning is carried out in the form of many assignments that are thematically arranged in educational resources.

Both systems, [4, 9] and [21], provide a convenient interface for editing the offered assignments and quickly visualizing the change in their performance, but they do not check if the completed assignments are correct and do not provide support for learners' mistakes. They also do not stimulate the development of algorithmic thinking.

Codingame [3] provides training in 23 programming languages based on gaming. The learners write code, which is then interpreted and tested in a game with pre-test examples. The platform supports the development of algorithmic thinking, offering an assignment on each step. The decision of the assignment is a specific sub-algorithm of the game and the strategy for solving it needs to be suggested by the learner. Codingame is designed for more advanced learners.

Scratch [7, 13, 14, 15] and ToonTalk [10, 11, 12, 17] are visual programming environments, that support young learners in making the transition from visual languages to text-based languages. The approach used is suitable for studying the fundamental principles of programming for children, but it is inappropriate for adults, because the system does not offer connection between constructed visual algorithms and programming code in a given programming language.

The system Scratch is well-known for adolescents. It uses graphical representation of programming constructions. This is why Scratch can be used to teach programming concepts.

ToonTalk system is designed for children. It is realized on the base of visual graphical objects which represent the statements and data types in programming.

3. PRINCIPAL PROGRAMMING ALGORITHMS AND THEIR INVARIANT ELEMENTS

Multiple programming languages, textbooks, books and learning courses in programming have been studied. On this base the main algorithms that teachers or authors present to the students are outlined. The found principal algorithms in teaching of imperative programming languages are presented in [16].

These algorithms have been studied to determine their invariant and variant parts. Each algorithm is presented by its invariant part and parameters, which correspond to the variant part. These algorithms are called **invariants**. The found invariants do not depend on a specific programming language. Invariants can be realized as a template code for different programming languages. Concrete sample codes, that use a given template code, are also offered in [16].

In [16], 98 invariants, suitable for teaching the course "Programming" in the bachelor programs of Plovdiv University in the field of computer science and 44 invariants - in the course "Algorithms and Data Structures", are presented. The proposed invariants are made with template codes of two programming languages (C # and Visual Basic) with more than 170 implementations in one language, because some invariants are realized in several ways (with different algorithms or statements).

Invariants are classified in 13 groups on the base of their basic assignments (algorithms), which are implemented during learning computer programming: Declaration of data, Input of data, Output of data, Inserting data, Deleting data, Transforming data, Passing on data, Searching data, Sorting data, Checking conditions, Different calculations, Sub-algorithms and Specific mathematical algorithms.

While examining the retrieved invariants, it is noted that they can be divided into 5 groups according to the parameters they have. The parameters of invariants can be:

- **Variable** - selectable variable name that serves as a "formal parameter" for a description of the algorithm of invariant;
- **Data type** - determined parameter type to be able to realize the common algorithms for different types. Sometimes you have to give a list of possible types if the algorithm is implemented only for some data types;
- **Unspecified invariant** - selected invariant of all possible invariants;

- **Invariant from a certain list** – the selected invariant can only be of predetermined invariants;
- **Invariant from a specific kind** - selectable invariant of a certain type (one of 13 groups) and each invariant of the type can be used.

Invariants may have one or more parameters of the same type or of different types. The parameters (variables and data types) of invariants can be input, output or input-output, and the remaining parameters are only input.

On the base of these invariants, a software system [8] is developed, which supports learning of programming through the coding of principal algorithms and the combination of them.

4. TYPES OF INVARIANTS

The invariants in the above-mentioned system are represented by a name and template code with parameters. The only parts of the code, which can be edited are the parameters. We will represent the types of invariants with examples via algorithms for processing arrays.

4.1. Invariants with parameter – variable

When using invariants with parameters of type one - variables (see Example 1.) in the invariant codes the only thing that can be modified are the names of the variables. They are automatically named with a default name (example: *a*, *arr*, *n* etc.). When solving specific assignment, renaming the variables may be needed, in order to implement the desired logic of the program. The parameters may receive another variable name, expression or specific value. In Example 3 we will use the invariant "input of double-precision floating-point value of variable *x*" as we change the name of the variable *x* to *arr[i]*.

Example 1. Invariant with parameter – variable

Invariant	Input of double-precision floating-point value of variable *x*
Input Parameters	-
Output Parameters	*x*
Template code	*Console.Write("Input value of the double variable x= ");* *x=Double.Parse(Console.ReadLine());*

4.2. Invariants with parameter – data type

When dealing with invariants of type two – data type (see Example 2) students will have to choose only the type of the data (*int*, *double*, etc.) of the appropriate variable, which implements the specific assignment.

In Example 2 we have to specify two parameters – variables *x* and *y*, which are input-output parameters for the invariant and the data type of variables *x* and *y*. With this invariant we can swap values for variables of different types.

Example 2. Invariant with parameter – data type

Invariant	Swapping values of two variables *x* and *y* from type *type*
Input Parameters	*x, y, type*
Output Parameters	*x, y*
Template code	*type buf= x;* *x=y;* *y=buf;*
Example of data type	*Int*

4.3. Invariants with parameter - any invariant

When it comes to invariants which have any sub-invariant as a parameter, this parameter may be chosen arbitrarily from the common set of invariants in a way to solve the assignment at hand. In Example 3 for processing an array *arr* with *n* elements of data type *type*, there are two possible sub-invariants offered in order to solve two specific assignments: "input of double-precision floating-point value of variable *x*" (see Example 1) for solving "input of array *arr* with *n* elements of double-precision floating-point type" and the second sub-invariant "checking if variable *a* is less than variable *b*" – for the assignment "printing elements of the array *arr* with *n* elements of data type *type* which are less than variable *b*".

Example 3. Invariant with parameter – any sub-invariant

Invariant	Processing array *arr* with *n* elements of type *type*
Input Parameters	*arr, n, type, invariant*
Output Parameters	*arr, n, type*
Template code	*for (int i=0; i<n; i++)* *{* *invariant(arr[i]);* *}*
Examples of sub-invariant	Input of double-precision floating-point value of variable *x* (*arr[i]*) Checking if variable *a* is less than variable *b* (*arr[i], b*)

4.4. Invariants with parameter – invariant of definite list

When dealing with invariants of type four, a sub-invariant has to be chosen from a predefined list of invariants. In Example 4 for invariant "searching extrema (minimum/maximum) *extrem* of the array *arr* with *n* elements of data type *type*", one out of two sub-invariants have to be chosen. Those sub-invariants help implement two searches with one template code: "checking if variable *a* is less than variable *b*" and "checking if variable *a* is greater than variable *b*". For parameters of the variables *a* and *b* in sub-invariants we use *extrem* and *arr[i]*.

Example 4. Invariant with parameter – sub-invariant of definite list

Invariant	Searching extremum (minimum/maximum) *extrem* of the array *arr* with *n* elements of type
Input Parameters	*arr, n, type,* *list of invariants*: Checking if variable *a* is less than variable *b* (*extrem, arr[i]*), Checking if variable *a* is greater than variable *b* (*extrem, arr[i]*)
Output Parameters	*extrem*
Template code	*type extrem=arr[0];* *for (int i=1; i<n; i++)* *invariant* *{* *extrem=arr[i];* *}*
Example of sub-invariant	*if (a>b)* *{* *invariant* *}*

4.5. Invariants with parameter – invariant of given kind

For invariants with parameters of the last fifth kind one sub-invariant of given kind must be chosen.

Example 5. Invariant with parameter – sub-invariant of given kind

Invariant	Input of the array *arr* with *n* elements of type *type*	
Input Parameters	*arr, n,* *kind of invariants*: Input of data of type *type*	
Output Parameters	*arr*	
Template code	*for (int i=0; i<n; i++)* *{* *invariant (arr[i])* *}*	*foreach (type x in arr)* *{* *invariant (x)* *}*
Example of sub-invariant	Input double-precision floating-point value of variable *x*	

In Example 5 for the invariant “input of the array *arr* with *n* elements of data type *type*”, two exemplary templates which implement this invariant are offered (in programming the possibility of implementing one algorithm with different statements is usual). The kind “input data of data type *type*” is offered as a choice. This kind includes all invariants, which are used to input data of different types. With them we will implement the input of elements in an array regardless of their type. As an

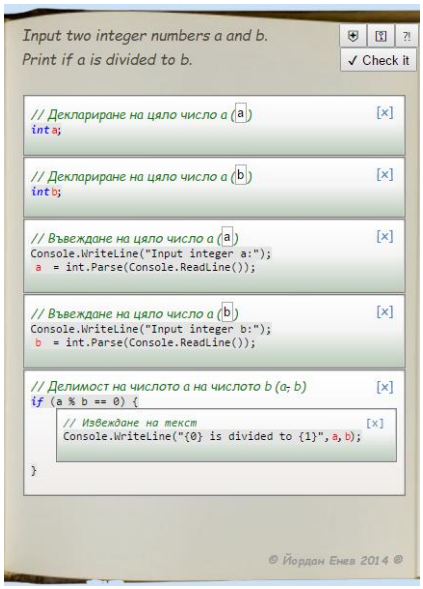
example, choice of sub-invariant "input double-precision floating-point value of variable *x*" is offered (see Example 1.).

5. ASSIGNMENTS, TESTING THE DIFFERENT COGNITIVE LEVELS OF BLOOM'S TAXONOMY

The training in this system is realized based on tasks (questions and assignments). Their solution is achieved by using knowledge of programming invariants. In Table 1 the realized tasks are shown, ordered according to the different evaluated cognitive levels of Bloom's taxonomy [2]. For every question or assignment, we classify the cognitive level covering its correspondence to the kind of test question or assignment from the standard didactic classifications and the way of evaluating the students' decisions. In the table the symbol 'A' is used for automatic evaluation from the system and 'T' – for teacher's evaluation. In addition to it, instead of "name of invariant", "invariant" is used, and for "template code that realizes an invariant", "template code" is used.

Different test questions and assignments can be automatically generated based on the data in the system. Only those types that are suitable for learning programming algorithms are selected and implemented in the system.

Figure 1 presents an example – decision of assignment of type "creating a program by selecting invariants of the overall list (with nesting of invariants)" in our system and translation of the example in English (on the right side).



Input two integer numbers a and b.
Print if *a* is divided to *b*.
//Deklaration of integer variable a (*a*)
int a;
//Deklararion of integer variable a (*b*)
int a;
//Input of integer variable a (*a*)
Console.WriteLine("Input integer a:")";
a=int.Parse(Console.ReadLine());
//Input of integer variable a (*b*)
Console.WriteLine("Input integer b:")";
b=int.Parse(Console.ReadLine());
//Division of variable a to variable b (*a*, *b*)
if (a%b==0)
{
  //Output text
  Console.WriteLine("{0} is divided to {1}",
  a, b);
}

Fig. 1. Assignment of Bloom's level "synthesis" in the learning system

Table 1. Tasks with invariants covering cognitive levels of Bloom's taxonomy

Cognitive level	Questions and assignments with invariants	Type of questions and assignments	Evaluation
Knowledge	Determining the truth of the connection between invariant and template code	Question with an answer true / false	A
Comprehension	Determining a code of invariant from a list of template codes	Multiple choice	A
	Determining the name of a template code in a given list of invariants	Multiple choice	A
	Determining correspondence between invariants and template codes that realize them	Determining correspondence	A
Application	Ordering given template codes in order to solve a given task (without nesting of invariants)	Ordering objects	A
	Ordering given template codes in order to solve a given task (with nesting of invariants)	Ordering objects	A
	Inserting template codes in a partially written program by choosing from a list of given template codes	Filling the fields in the template of the answer (with help)	A+T
Analysis	Determining template codes, realizing a given invariant	Multiple choice	A
	Determining correspondence between the two lists of template codes that can implement the same algorithm	Determining correspondence	A
	Inserting template codes in a partially written program by choosing from a overall list of template codes	Filling the fields in the template of the answer (without help)	A+T
Synthesis	Creating a program by selecting invariants of the overall list (without nesting of invariants)	Free answer	A+T
	Creating a program by selecting invariants of the overall list (with nesting of invariants)	Free answer	A+T
Evaluation	Finding the wrong template code in a program	Multiple choice	A
	Finding wrong the template codes in a program	Multiple choice	A
	Detecting and correcting errors in the used template codes in a certain program	Free answer	T

In the example (Figure 1) learner has to choose necessary invariants (and code implementation for each invariant) and order them. Nesting of invariants is possible for this type of assignment (see nested invariant "output text" in the example).

6. LEARNING SYSTEM IN PROGRAMMING USING PRACTICING BY CODING

The training system in programming represents a very simplified e-learning environment. Its aim is to be a support tool in teaching how to program. It focuses on the specificity of this training and does not function as a system for organizing and maintaining of the learning. The system only supports three levels of users – learner, teacher and administrator.

The system has the following main functionalities: system **initialization** with the necessary information for the desired programming language; **learning** the trainees through "practicing"; **monitoring** the progress of the trainees and **evaluation** of the solutions that they provide.

The system has already been initialized with a selected set of invariants [8], but the teacher can complete the existing invariants if he/she finds a missing one. Every invariant has to be connected with at least one template code in the target programming language. Some of the invariants can be performed by different statements, which leads to binding to more than one template code.

Most of the different types of questions and assignments can be automatically generated from the system. After they have been generated, every question or assignment can be edited by the teacher and given to the learners in an edited version.The teacher can also write additional assignments, which the system can provide to the learners. During the description of the new assignment the teacher determines for which of the 13 groups the assignment is more suited for and which level of Bloom's taxonomy it covers.

It is recommendable for the teacher to mark questions and assignments, regardless of the fact whether they are automatically generated or personally written. This applies to each group from the different levels. The marked assignments are the minimum amount of questions and assignments, which the student is required to pass in order to comprehend the learning content. Due to the nature of the method, the marked assignments are the first to be given to the student. The assignments, which are completed by a specific trainee, are marked as solved and when the trainee logs into the system again, he/she does not receive the same assignments.

Keep in mind that assignments which have already been written down can be used again when starting to learn a different programming language. The more the system is used, the greater and more precise the list of invariants will become. Thus, increasing the amount of the assignments and decreasing the amount of work done by the teacher.

The trainees can start their learning in the system by examining the template codes for the invariants, which are arranged in thematic groups. Then they continue with solving different problems by choosing an assignment from a specific group and the appropriate level of Bloom's taxonomy. At first the trainees will be given the assignments, marked by the teacher as recommended, and then they will be given automatically generated questions and assignments from the system, based on the specific group and Bloom's taxonomy level.

The majority of the available types of questions and assignments are evaluated automatically by the system (see Table 1). The other major part of the assignments up to level selection and arrangement of code templates is also evaluated automatically. The teacher only needs to review the name of the parameter – variables templates (i.e. the distribution of parameters within the templates). When evaluating the different assignments, the teacher can evaluate the trainees and provide written feedback.

During the training, the teacher can monitor the activity and the progress, demonstrated by trainees (the number of resolved problems and assignments for a certain period of time, as well as the received assessments).

The system has been set up with MVC architecture and uses the WYSIWYG editor TinyMCE. The editor has an extension with a special functionality, so as to easily mark parameters within the template codes.

7. CONCLUSION

The paper provides an approach towards teaching how to program, which is not dependent on the programming language. More than 140 invariants are gathered, they are then realized with more than 170 code templates, which are for one programming language only. The invariants are systematized into 13 groups, depending on the types of the algorisms.

A software system is offered, providing 15 types of questions and assignments, based on invariants, suitable for learning how to program. These are classified based on the appropriate cognitive level of Bloom's taxonomy. The system has been developed in order to support the programming classes in the Faculty of Mathematics and Informatics at the Plovdiv University "Paisii Hilendarski".

REFERENCES

[1] Barnev, P., G. Totkov, Vl. Shkurtov, R. Doneva, K. Garov. Computer Science, textbook for 9th year students. *Letera*, 2001. (in Bulgarian)

[2] Bloom, B. S., M. D. Engelhart, E. J. Furst, W. H. Hill, D. R. Krathwohl. Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain. *New York: David McKay Company*, 1956.

[3] Codingame, https://www.codingame.com/ (current January 2018)

[4] Crunchzilla, http://www.crunchzilla.com/ (current January 2018)

[5] Doneva, R., S. Gaftandjieva. Information Technologies Learning in Bachelor Programs for Nonprofesionals in Information Technologies. In *Sience Session "Days of the science 2010"*. Plovdiv Bulgaria, 39-42, 2010. (in Bulgarian)

[6] Doneva, R., S. Gaftandjieva. Invariants in Learning of Spreadsheets. In *Proceedings of National Conference "Education in Information Society"*. Plovdiv, Bulgaria, 293-302, 2011. (in Bulgarian)

[7] Dorling, M., D. White. Scratch: A way to logo and python. SIGCSE 2015 - *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, Kansas City, USA, 191-196, 2015.

[8] Enev, J., E. Somova. Software Learning System Based on Invariants in Computer Programming, *International Journal of Emerging Research and Solutions in ICT (ERSICT)*, Volume 1, Number 1, http://www.ersict.org/ersict-01-16-p06/, FICT, 2016.

[9] Hamilton, B. Integrating technologies in the classroom. Tools to meet the needs of every student. *International Society for Technology in Education*, USA, 2015.

[10] Jung, J., Park, H., et al. Gaming and simulations: Concepts, methodologies, tools, and applications. volume 1, Information Resources Management Association USA (editor), *Information Science Reference, Hershey*, New York, 2011.

[11] Lieberman, H. (editor), Your wish is my command. Programming by example. *Morgan Kaufmann Publishers*, USA, 2001.

[12] Lytras, M., D. Gasevic, P. Pablos, W. Huang. Technology enhanced learning: Best practices. *IGI Publishing, Hershey*, New York, 2008.

[13] Marji, M. Learn to program with Scratch: a visual introduction to programming with games, art, science and math. *No Strach Press*, San Francisco, USA, 2014.

[14] Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Learning computer science concepts with Scratch. *Computer Science Education*, Volume 23, Issue 3, 239-264, 2013.

[15] Scratch, https://scratch.mit.edu (current January 2018)

[16] Somova E., Enev Y., Totkov G., Invariants in Learning of Programming. *International scientific on-line journal "Science & Technologies"*, Volume IV, Number 3, http://www.sustz.com/journal/VolumeIV/Number3/Papers/ElenaSomova1.pdf, 2014. (in Bulgarian)

[17] ToonTalk, http://www.toontalk.com/ (current January 2018)

[18] Totkov, G., R. Doneva, L. Besaleva, I. Chakarova. Invariants in Information Technologies Learning. In *Proceedings of National Conference "Education in Information Society"*. Plovdiv, Bulgaria, 22-29, 2010. (in Bulgarian)

[19] Totkov, G., et al. Computer Science: Overview, (G. Totkov ed.). *PHARE*, 1999.

[20] Totkov, G., Vl. Shkurtov, R. Doneva. Fundamentals of Computer Science. *University Press*, *Plovdiv University*, Plovdiv, 2001. (in Bulgarian)

[21] W3Schools, http://www.w3schools.com/ (current January 2018)

[22] Bloom's Taxonomy, http://www.bloomstaxonomy.org/Blooms%20Taxonomy%20questions.pdf (current January 2018)

Information about the authors:

Associate Professor Elena Somova, PhD, Head of the Department "Computer Science", University of Plovdiv "Paisii Hilendarski", areas of scientific research: programming languages, e-learning, e-learning environments, game-based learning, gamification, etc.

Jordan Enev, PhD student, University of Plovdiv "Paisii Hilendarski", areas of scientific research: programming languages, software learning systems, etc.

Manuscript received on 15 July 2017