Emerging Research and Solutions in ICT 1(1):59–67 DOI: 10.20544/ERSICT.01.16.P06
UDC: 004.455:159.953.5]:004.421

Software Learning System Based on Invariants in Computer Programming

Jordan Enev[1], and Elena Somova[1]

[1]The University of Plovdiv "Paisii Hilendarski", Faculty of mathematics and informatics,
24 Tzar Assen st., 4000 Plovdiv, Bulgaria
yordan.enev@bgosoftware.com
eledel@uni-plovdiv.bg

Abstract. The paper considers the idea about invariant teaching and learning of computer programming, independent of concrete programming language and version. Software system, built on the base of template algorithms (called invariants), is presented. 98 invariants are proposed for the course "Programming" from the bachelor degree programs at Plovdiv University, Bulgaria, and 44 invariants – for the course "Algorithms and Data Structures". The proposed invariants are made till now with template codes on two programming languages (C# and Visual Basic) with more than 170 realizations in one language. The invariants are classified in 13 groups on the base of kinds of basic assignments (algorithms), which are solved during learning computer programming. The invariants have parameters of 5 types – variable, data type, random invariant, invariant from given list and invariant from given kind. Several levels of difficulty for solving of assignments are proposed. The system can be used as main learning resource in self-learning during distance education or as additional auxiliary resource in traditional learning.

Keywords: computer science, software learning systems, learning programming languages, invariant programming.

1. Introduction

The continuous appearance of new versions of the programming languages, as well as change of the teaching language with more "modern" one arises the question: "Is it possible to prepare learning course on Programming, which to be independent of frequently changing technologies?". Making learning course with independent (invariant) elements of used software resources, from one side will facilitate the work of the teachers and from the other side will give to the learners a stable knowledge about principles and main algorithms in computer programming, which they can use for a long time after finishing their education.

1.1. Invariant learning

Attempts for solving the similar problem in the field of learning information technologies and fundamentals of computer science exist. For carrying out distance education in the field of information technologies, learning materials [17] with invariant elements are developed in 3 languages (Bulgarian, French and Lithuanian). On the base of the same methodology, the basic university course "Fundamentals of Computer Science" [20] at Plovdiv University and the training aid in Computer Science [19] are created. On the base of invariant elements, the package with learning resources for high school [1, 18] is made as well.

The idea of invariant learning is developed in [4, 5, 16], as in [4, 5] the examples about invariant teaching of spreadsheets are given, and in [16] – about text processing. In [16] the accent is put on the so called "invariant knowledge" (i.e. these, which are relatively static in time and resistant to possible changes) in presenting/teaching of the learning content. In [4, 5] the development of "invariant frame", which to be defined as standard for invariant presentation of learning content, whose subject is a concrete technology in the field of information technologies and for which the elements comprised in the frame are determined.

1.2. Learning Systems

The most of existing learning systems in the field of computer programming are dependent of the programming language, for which they are designed (e. g. [3] and [21]).

In Crunchzilla [3, 7], the learning is designed only for solving visual assignments in JavaScript. The lessons in different levels of complexity and in dependence of the learner's age are proposed. The system represents virtual "teacher", which leads previously set dialogue with the learner and gives assignments connected to the visualization and translation of objects on the screen. The learner uses two panels, the codes given from the systems are edited in the first one and in the second one they are interpreted and visualized. The system does not check about the correctness of the solved assignments.

W3Schools [21] is a learning environment in some programming languages and technologies (HTML, CSS, JavaScript, PHP, Bootstrap, etc.) on the base of tutorials. Tutorials contain rich and systematized information with a lot of example assignments in the respective programming languages and technologies. The environment, like Crunchzilla, visualizes exemplary assignments in two panels: one for code, where it is possible to edit and another for showing execution of examples and written by the learner codes. This makes the platform a wonderful tool for reference and quick testing of short programming codes, but does not assist progress in algorithmic thinking. The platform is deeply linked with the syntax of studied programming language. There is no support in the case of learner errors as well.

Other systems such as Codingame [2] are for advanced learners. Codingame proposes learning in 23 programming languages through writing codes, which are interpreting in games. The platform supports progress in algorithmic thinking during creating games, as

it proposes on each step solving certain assignment, which is concrete algorithm in given game. The learner obtains input information and has to write programming code, which lead to actions in order to obtain searched output state. The written code after that is tested with previously prepared test game examples, which leads to visual result (game animation, which shows the proposed strategy by the learner). The system is not suitable for use to teach beginners in computer programming.

The well-known system Scratch [6, 11, 12, 13] is a visual programming environment that supports young learners (ages 8 to 16) in making the transition from graphical languages to text-based languages. Scratch can be used to teach concepts of computer science. The system uses graphical representation of programming constructions.

The system ToonTalk [8, 9, 10, 15] is also designed for learning children (ages 3 to 5) and is realized on the base of visual graphical objects, which represent objects (statements and data types) from the world of programming.

The approach is really suitable in learning main principals of computer programming by children, but it is inappropriate for adults, because the system do not propose linking the designed visual algorithm with programming code in some algorithmic language.

A common characteristic feature of the most existing systems is the strong connection to the syntax of the chosen programming language. This can lead to the appearance of a multiple syntax errors that will discourage learners in the beginning stage of studying.

The paper presents software system, supporting teaching of computer programming in algorithmic language, developed on the base of invariant approach. This approach does not depend on the concrete language and uses set of templates, called *invariants*, which represent common basic algorithms in each programming language. Each template in the system is presented by exemplary codes with parameters in the particular programming language.

2. Invariants in Programming

In [7] the found out basic algorithms (*invariants*) in teaching imperative language programming, which do not depend on taught language, are presented. Their representation as template codes on different languages is shown in [14].

For teaching course "Programming" in bachelor program at Plovdiv University, Bulgaria 98 invariants are proposed, and for "Algorithms and Data Structures" – 44 invariants. Proposed invariants are made with template codes on two programming languages (C# and Visual Basic) with more than 170 realizations in one language, because some invariants are made in several ways (with different algorithms or statements).

Invariants are classified in *13 groups* on the base of the kinds of basic assignments (algorithms), which are solved 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.

Below are given some examples of invariants, where programming codes are shown in C#.

Table 1. Parameter types of invariants

Type	Parameter	Choice of sub-invariant
1	variable	no
2	data type	no
3	random invariant	choice from all invariants
4	invariant from given list	choice from given invariants
5	invariant from given kind	choice of invariant from given kind

Proposed invariants are considered according to the parameters they have. The *parameters are of 5 types* (see Table 1) – variable, data type, random invariant, invariant from given list and invariant from given kind. Invariants can have one or several parameters of one type or of different types. The parameters (variables and data types) of the invariants can be input, output or input-output, and the rest parameters are only input.

Table 2. Example of invariant with parameter of Type 1

Invariant	Finding sum `sum` of elements of the array `arr` with `n` elements
Input Parameters	arr, n
Output Parameters	sum

Template code	int sum=0;
	for (int i=0; i<n; i++)
	  sum+=arr[i];

When using *invariants with* first type parameters – *variables* (see Table 2) in solving assignments (construction of computer program on the base of invariants) some problems with naming of variables can arise. The variables are named automatically with default name (e.g. `a`, `arr`, `n`, etc.). After placement of invariant the user have to rename the variables if necessary in order to accomplish desired logic of the computer program. Depending on the particular assignment, parameters can obtain another name of a variable, expression or concrete value.

When using *invariants with* parameters from second type – *data type* (see Table 3), it has to make a choice of the name of the proper type.

In the current example two parameters-variables `x` and `y`, which are input-output parameters for the invariant and data type of the variables `x` and `y` have to be passed. Through this invariant variable values from different types can be swapped.

Table 3. Example of invariant with parameter of Type 2

Invariant	Swapping values of two variables `x` and `y`
Input Parameters	x, y, type
Output Parameters	x, y

Template code	type buf= x;
	x=y;
	y=buf;

Example of data type	int

Table 4. Example of invariant with parameter of Type 3

Invariant	Divisibility of number `a` to number `b`
Input Parameters	a, b, invariant
Output Parameters	–
Template code	if (a%b == 0) invariant;
Example of sub-invariant	Console.WriteLine ("{0} is divided to {1}", a, b);

When using *invariants*, which *have parameters* from third type – *random invariant*, each invariant can be chosen from the total set of invariants (see Table 4). For each particular situation a proper sub-invariant has to be chosen. In the example, invariants for output can be used for sub-invariant.

Table 5. Example of invariant with parameter of Type 4

Invariant	Sorting array `arr` with `n` elements, using method "Bubble Sort"
Input Parameters	arr, n, type, list of invariants: Check if a is in relation rel with b (arr[i], arr[i+1], <), Check if a is in relation rel with b (arr[i], arr[i+1], >)
Output Parameters	arr

Template code	type buf;
	for (int i=1; i<n; i++)
	for (int j=0; j<n-i; j++)
	invariant
	{
	buf=arr[j];
	arr[j]=arr[j+1];
	arr[j+1]=buf;
	}

Example of sub-invariant	if (arr[j]>arr[j+1])

About *invariants with* fourth type *parameter – invariant from given list*, an invariant from the previously given list of invariants is chosen according to the condition of the assignment (see Table 5). In the example, one invariant is chosen from two invariants,

which are the same (`Check if a is in relation rel with b`), but with different values of the parameters: `(arr[i], arr[i+1], <)` and `(arr[i], arr[i+1], >)` (they give the opportunity this code to solve both assignments: sorting in ascending and descending order).

Table 6. Example of invariant with parameter of Type 5

Invariant	Output of the array `arr` with `n` elements
Input Parameters	arr, n, invariant: Output of data (arr[i], type)
Output Parameters	–

Template code	for (int i=0; i<n; i++)
	{
	invariant
	}

Example of sub-invariant	Console.WriteLine (“arr[{0}]= {1}”, i, arr[i]);

About *invariants with parameters* from last fifth type – *invariant from given kind* (see Table 6) is necessary to choose one invariant from determined kind. For example, for sub-invariant of kind `Output of data` (this kind includes all invariants, which serve to output data from different types) invariant `Output of integer value a` can be chosen, where parameter `a` is with value `arr[i]`, as chosen in Table 6. Through this invariant, the output of the array is provided, independently from the type of the array elements.

There is a possibility an invariant to contain another exactly fixed invariant. For example, the invariant in Table 2 contains the sub-invariant `Navigating the array arr with n elements`, and the invariant in Table 5 – `Swapping values of the variables x and y`. In this way we do not obtain the new kind of invariants, because it is not necessary to make a choice of invariant, i.e. the invariant is not a parameter.

3. Software System for Invariant Learning of Computer Programming

The proposed software system gives friendly visual interface for learning programming through the same methodology for the different programming languages.

The system strives to eliminate the occurrence of syntax errors giving to the learners a list with previously prepared templates (invariants) in order the learners to concentrate into the design of concrete assignment and realization of the algorithm on the base of basic algorithms. Thus learners only have to choose the right programming constructions and link them in logically correct way.

The system proposes some kinds of assignments from different levels of difficulty for learning computer programming:

1. *Learning invariants* – examining invariants, ordered in thematic groups. Each invariant is presented with a name, description, parameters (input and/or

output), template and example codes. If the invariant has a parameter of type invariant (Types 4 or 5), list of possible sub-invariants is given too;

2. *Matching programming codes* – the teacher proposes two groups with short programming codes, where the codes from the first group have to be linked with the codes in the second group (e. g. with the following condition: "Link the programming codes, where the execution leads to the equal results");
3. *Removing unnecessary invariant from given computer program* – the learner have to find and choose unnecessary code in the program;
4. *Choosing invariant from proposed list of invariants and inserting it in the correct place in computer program* – the teacher selects some invariants, from which the learner have to choose and to put in the specified place/places in the program;
5. Choosing invariant from total list of invariants and inserting it in the correct place in computer program – in the program, only the places for insertion of invariants are specified without any other help;
6. *Constructing computer program from given invariants (without nesting)* – composing program is done from previously selected invariants from the teacher, which learner has to put in the correct order without nesting of codes;
7. *Constructing computer program from given invariants (with nesting)* – composing program is done from previously selected invariants from the teacher, which learner has to put in the correct order. The nesting of code is possible;
8. Constructing computer program through choosing and ordering invariants from given list of invariants (without nesting) – in the list, there are more invariants, that are necessary (in the difference to 6.);
9. Constructing computer program through choosing and ordering invariants from given list of invariants (with nesting) – in the list, there are more invariants, that are necessary (in the difference to 7.);
10. Constructing computer program through choosing and ordering invariants from total list of invariants – for this kind of assignment only condition is given.

Below is given an example with condition of an assignment, design of the algorithm through invariants, obtained decision and screenshot of the system with this decision.

Assignment 1.: In a company 10 workers work with salaries: 205.50, 560, 245, 2040.70, 536.20, 245, 2100, 560, 560 и 600. Show the incomes of the workers, which has high-paid positions (receive more than 2000).

Algorithm (through used invariants and level of nesting, given with displacement):

Defining array `income` with concrete `10` elements of floating-point type
Navigating array `income` with `10` elements
Belonging of `income[i]` to the interval (2000, +∞)
Output the value of variable/expression

Decision:

```
double[] income= {205.50, 560, 245, 2040.70, 536.20, 245,
2100, 560, 560, 600};
for (int i=0; i<10; i++)
```

```
{
  if (income[i]>2000)
  {
    Console.WriteLine("Worker with high income: ",
    income[i]);
  }
}
```

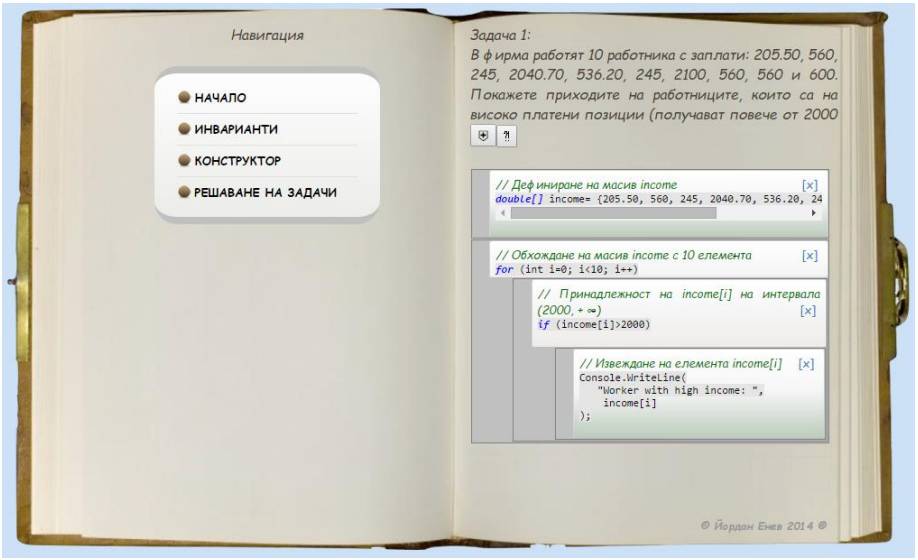

Fig. 1. Screenshot of the system with decision of the assignment 1

Part of the proposed assignments (2, 3 and 4) is evaluated entirely automatically by the system, because actually they are closed answer test questions. Other part of assignments (6, 7, 8 and 9) is proposed in two kinds: only with invariant names (according to the kind of the assignment) or through prepared programming codes of these invariants, where parameters (variables and types) are filled in appropriately for the particular assignment. The first kind is evaluated semi-automatically – the structure of the decision is evaluated by the system, but the teacher has to approve the evaluation and review the correctness of the parameters. The second kind is evaluated automatically by the system. The rest assignments (5 and 10) are evaluated by the teacher, because they represent an open answer test question.

4. Conclusion

The paper presents software system, using approach for teaching computer programming with algorithmic language, independent of concrete programming language, on the base of set of templates, called invariants. The system is provided as main learning resource in distance programs and as additional auxiliary resource in traditional learning at the university.

Until now 142 invariants of algorithms from courses “Programming” and “Algorithms and Data Structures” are included in the system. These courses are carried out in bachelor programs in the field of computer science in the Faculty of Mathematics and Informatics at Plovdiv University “Paisii Hilendarski”, Plovdiv, Bulgaria.

5. References

1. Barnev, P., Totkov, G., Shkurtov, Vl., Doneva, R., Garov, K.: Computer Science, textbook for 9th year students. Letera. (2001) (in Bulgarian)
2. Codingame, https://www.codingame.com/ (current January 2016)
3. Crunchzilla, http://www.crunchzilla.com/ (current January 2016)
4. Doneva, R., Gaftandjieva, S.: 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)
5. Doneva, R., Gaftandjieva, S.: Invariants in Learning of Spreadsheets. In Proceedings of National Conference “Education in Information Society”. Plovdiv, Bulgaria, 293-302. (2011) (in Bulgarian)
6. Dorling, M., White, D.: 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)
7. Hamilton, B., Integrating technologies in the classroom. Tools to meet the needs of every student. International Society for Technology in Education, USA. (2015)
8. 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).
9. Lieberman, H. (editor), Your wish is my command. Programming by example. Morgan Kaufmann Publishers, USA. (2001)
10 Lytras, M., Gasevic, D., Pablos, P., Huang, W.: Technology enhanced learning: Best practices. IGI Publishing, Hershey, New York. (2008)
11. 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)
12. Meerbaum-Salant, O., Armoni, M., Ben-Ari, M.: Learning computer science concepts with Scratch. Computer Science Education, Vol. 23, No. 3, 239-264. (2013)
13. Scratch, https://scratch.mit.edu (current January 2016)
14. Somova E., Enev Y., Totkov G., Invariants in Learning of Programming. International scientific on-line journal "Science & Technologies", Vol. 4, No. 3 (2014). Available: http://journal.sustz.com/VolumeIV/Number3/Papers/ElenaSomova1.pdf (in Bulgarian)
15. ToonTalk, http://www.toontalk.com/(current January 2016)
16. Totkov, G., Doneva, R.:, Besaleva, L., Chakarova, I.: Invariants in Information Technologies Learning. In Proceedings of National Conference “Education in Information Society”. Plovdiv, Bulgaria, 22-29. (2010) (in Bulgarian)
17. Totkov, G., et al., Computer Science: Overview, (G. Totkov ed.). PHARE. (1999)
18. Totkov, G., Shkurtov, Vl., Doneva, R., Garov, K.: Information Technologies, textbook for 9th year students. Letera. (2001) (in Bulgarian)
19. Totkov, G., Shkurtov, Vl., Doneva, R.,Vidolov, B.: Computer Science (training aid). Regional Distance Education Study Center at Plovdiv University, Plovdiv, (1999) (in Bulgarian)
20. Totkov, G., Shkurtov, Vl., Doneva, R.: Fundamentals of Computer Science. University Press, Plovdiv University, Plovdiv. (2001)
21. W3Schools, http://www.w3schools.com/(current January 2016)