UDC 004.4'22

Oleksii Aleshchenko.

VISUAL PROGRAMMING SYSTEM

The article considers the implementation of visual programming systems and comparing the execution time of software algorithms, which are set graphically and translated into executable code using various intermediate programming languages and compilers.

Key words: visual programming, algorithm, execution time.

Fig.:5. Bibl. 4.

Target setting. When generating program codes according to graphical schemes of algorithms, one of the important tasks is to choose an intermediate programming language, and hence a compiler for it. The article considers a method of comparing the execution time of graphical programming algorithms on different compilers and programming languages. This method is based on the use of a system for translating graphical schemes of algorithms with the addition of blocks of the beginning and end of the time measurement inside the algorithm.

Actual scientific researches and issues analysis. The article [1] discusses the visual programming languages used to develop applications, especially in the field of the Internet of Things. The article [2] presents the features and functionalities of the language of visual verification of the code and shows its application in a number of case studies. The article [3] presents a web platform for text analysis and natural language processing, supporting the construction, exchange and execution of work processes. The platform allows you to visually create workflows for text analysis through a web browser and execute created workflows in the processing cloud. The article [4] shows the assessment of students' thinking skills developed using visual programming or material coding environments, and the relationship between students' ability to solve tasks, the type of target skills related to computational thinking, and the degree of the complexity of the proposed tasks.

Uninvestigated parts of general matters defining. This article is devoted to the study and analysis of the proposed approach for the generation of executable program codes for graphical schemes of algorithms, in particular through intermediate languages C/C++, Java, Pascal, Assembler. The research focuses on the study and evaluation of the effectiveness of different translation methods.

The research objective. The task is to create a system for generating executable program codes according to graphical schemes of algorithms with a subsystem for statistical evaluation of the effectiveness of different translation methods based on measurements of the execution time of parts of the algorithm.

The statement of basic materials.

There are different forms of representation of algorithms:

- verbal (natural language entry);
- graphic;
- formula (system of transition formulas);
- matrix;
- logical;
- pseudo-codes (partially formalized descriptions of algorithms);
- software (programming language).

Verification of the initial graphical representation of the algorithm is carried out by checking its structure according to the following points:

- the presence of terminal vertices of the beginning and end;
- not exceeding the allowed number of terminal vertices of one class for the corresponding type of graphical representation of the algorithm;
- lack of infinite cycles;
- reachability of all vertices from the initial one;
- the existence of a path from all vertices to the final one.

The following types of typification of variables are distinguished:

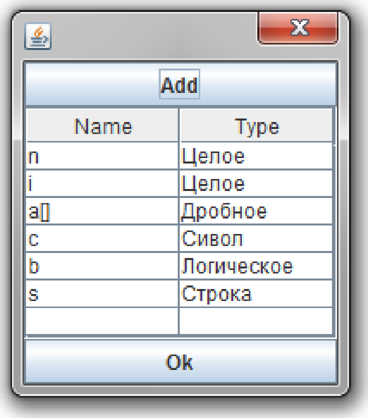
- registered;
- dynamic;
- explicit.

Name typing means that depending on the data that the variable stores in itself, its name is modified. For example, adding the symbol “i” to the variable name is used, etc. This violates the semantics of names and makes it difficult to describe subtypes.

Dynamic typing means that the variable type is not predefined, but is determined as the program executes (dynamically), according to the assignment value. So the type of one variable can change depending on the values assigned to it, and the operations in which it takes part, which complicates both type control and debugging of the program, as well as the development of the translator.

In the third case, the type description and variable declaration are performed explicitly. The advantages of explicit typing are its visibility, reliability and prevalence.

In fig. Figure 1 shows the implementation of a method for setting correspondence between variables and their types, which is based on explicit typing.



Name	Type
n	Целое
i	Целое
a[]	Дробное
c	Сивол
b	Логическое
s	Строка

Fig. 1. Implementation of a method for explicitly setting the correspondence between variables and their types

The UML class diagram allows you to define new types by creating classes and their hierarchies (Figure 2).

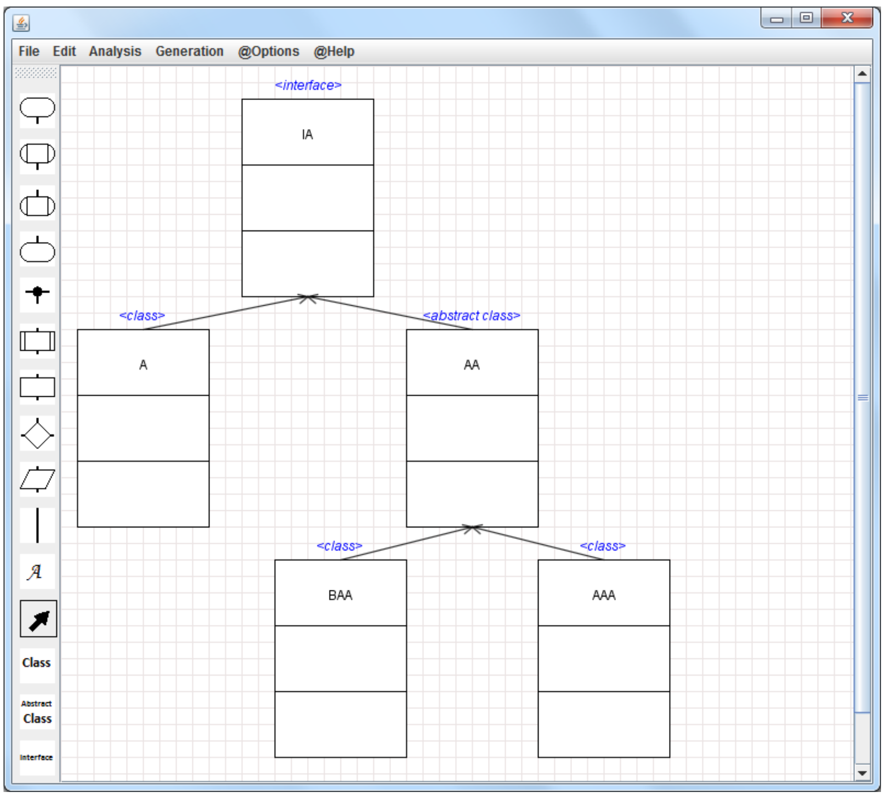


Fig. 2. Setting the class hierarchy in the system under development

Therefore, for the complete automatic generation of executable code from the graphical form of the algorithm, this form should generally contain a block diagram of the algorithm (as applied to the UML – activity diagram) and, if necessary, a table for setting the types of variables and / or class diagrams (Fig. 3).

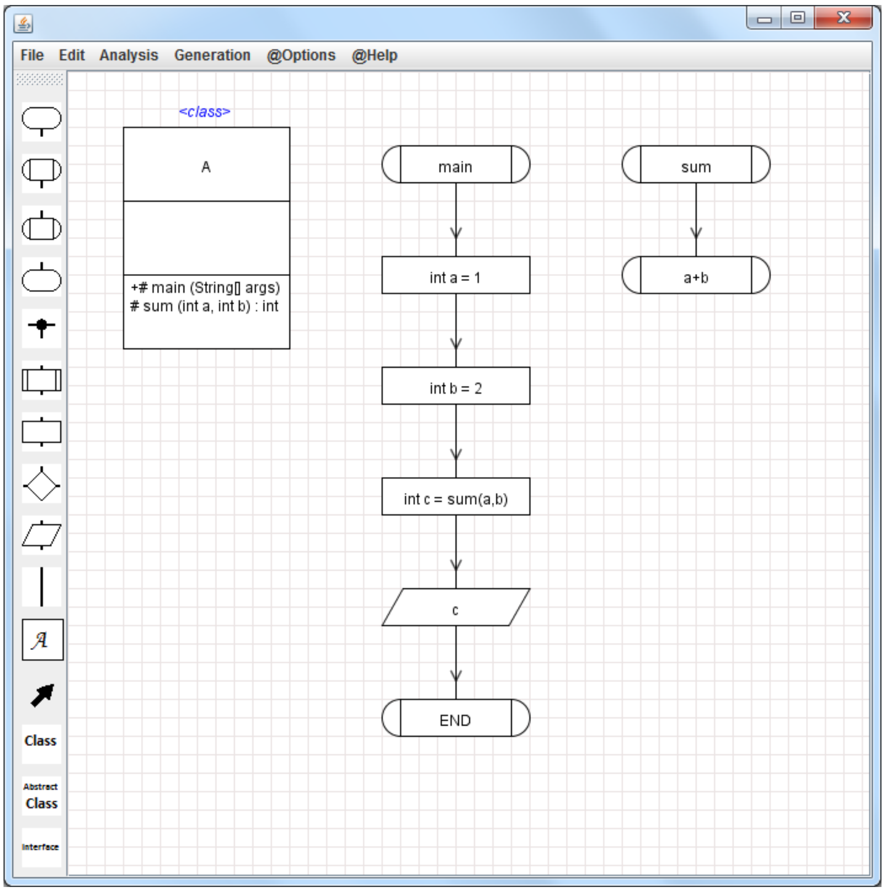


Fig. 3. Method of graphical representation of the algorithm for full automatic generation of executable code

In general, there are two ways to translate a high-level graphical representation of an algorithm into executable code:

1. Direct (direct) broadcast;

2. Translation into an intermediate language with subsequent generation of executable code by means of a "standard" (external, implemented by a third party) translator for the selected intermediate language.

Direct translation is not considered in this paper because of the obvious high complexity of the implementation. To select an intermediate translation language, a system was developed for the automatic generation of executable program codes using graphical forms for representing algorithms and a simulation model for statistical evaluation and research of the effectiveness of various ways of translating high-level graphic specifications into executable code.

Figure 4 shows a window with a graphical diagram of the algorithm for sorting simple exchanges and blocks for starting and stopping the timer.

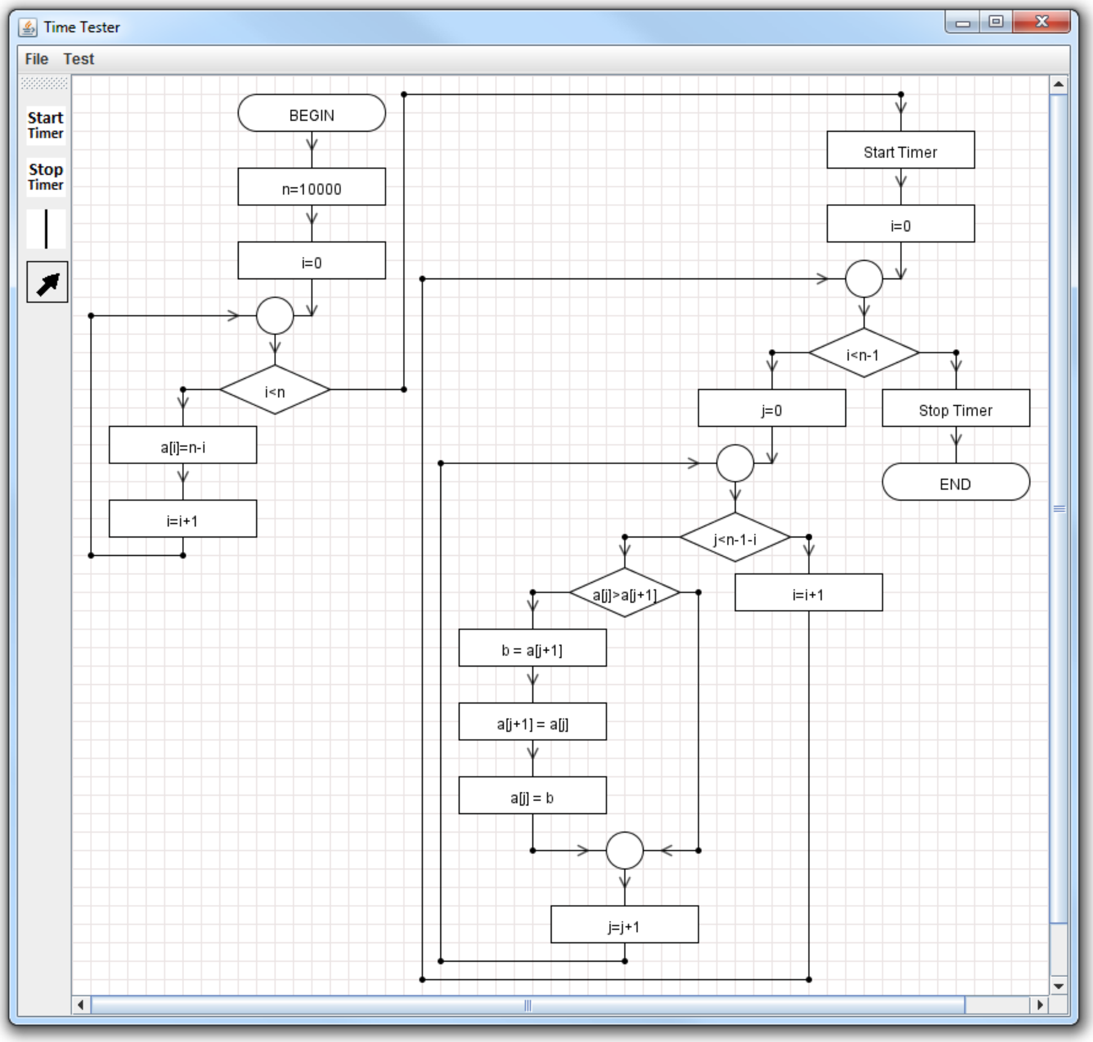


Fig. 4. Algorithm diagram with timer start and stop blocks

The generation system allows you to get executable codes through intermediate representations in various languages, run them and compare the execution time (Fig. 5).

The following translators are used to obtain executable program codes:

– C ++: "Minimalist GNU for Windows" (gcc-c ++ - 4.8.1-4-mingw32);
– Java: "Java SE Development Kit" (jdk-8u65-windows-i586);
– Pascal: "Free Pascal" (fpc-3.0.0.i386-win32).

Operators for measuring time:

– C ++: GetTickCount;
– Java: currentTimeMillis;
– Pascal: getTime.

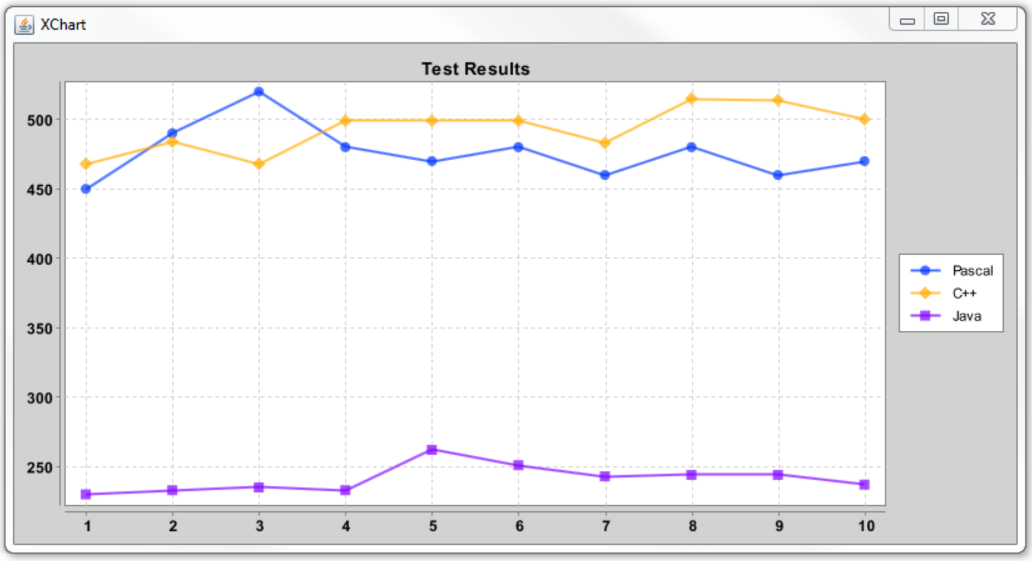


Fig. 5. Results of comparing the runtime of generated codes

Conclusions.

1. The forms of representing algorithms are investigated and analyzed.

2. A method for representing the algorithm in graphical form, which contains the necessary and sufficient information for the complete automatic generation of executable code, is proposed and justified.

3. The aspects of verification, typing of variables and type descriptions are investigated using the graphical form of the algorithm.

4. A method for verifying the initial graphical form of the presentation of the algorithm by checking the absence of errors in its structure is proposed and implemented.

5. A method for setting the correspondence between variables and their types based on explicit typing when representing the algorithm in graphical form is justified and developed. A software module has been developed for implementing this typing method.

6. The methods of translating high-level graphic specifications into executable code are analyzed.

7. A system for the automatic generation of executable program codes based on graphical presentation forms of the algorithm, which uses various intermediate translation languages, has been substantiated and developed.

8. A simulation model has been created for statistical evaluation and research of the effectiveness of various methods for translating high-level graphic specifications into executable code.

References

1. Ray, P. P. (2017). A survey on visual programming languages in internet of things. Scientific Programming, 2017.

2. Preidel, C., & Borrmann, A. (2016). Towards code compliance checking on the basis of a visual programming language. Journal of Information Technology in Construction (ITcon), 21(25), 402-421.

3. Perovšek, M., Kranjc, J., Erjavec, T., Cestnik, B., & Lavrač, N. (2016). TextFlows: A visual programming platform for text mining and natural language processing. Science of Computer Programming, 121, 128-152.

4. Djambong, T., & Freiman, V. (2016). Task-Based Assessment of Students' Computational Thinking Skills Developed through Visual Programming or Tangible Coding Environments. International Association for Development of the Information Society.

AUTHORS

Oleksii Aleshchenko – senior lecturer, Department of Computer Engineering, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute".

E-mail: alexey.aleshchenko@gmai.com