EUROLOGO 91

E. Calabrese
University di Parma
Editor

PROCEEDINGS
Third European Logo Conference
Parma, Italy, 27-30 August 1991

A.S.I. — Parma
1991

TURINGAL - THE LANGUAGE FOR TEACHING THE PRINCIPLES OF PROGRAMMING

Peter L. Brusilovsky

International Centre for Scientific
and Technical Information (ICSTI)
Kuusinen str. 21b
Moscow 125252, USSR

Abstract

Many beginners are far from being able to overcome the difficulties of the elementary programming study based on such languages as Pascal, Basic, C or Fortran. The success of the Logo "turtle graphics" ilas stimulated the development of the new approach towards the elementary programming teaching. This "mini-language" approach is discussed briefly in the paper. As an example of mini-language we describe the language Turingal, which was designed for the studying of elementary programming by the students of special secondary mathematical schools.

P. L. Brusilovsky

1. Introduction

At present one of the industrial programming languages in the available computer is likely to be taken so that one could get acquainted with the principles of programming. At best it is Pascal, but, unfortunately, more often Basic is used. Such a choice creates obstacles for the beginners in programming.

First of all, "conceptual basis" of the industrial language together with the main principles of programming includes a lot of secondary notions which reflect the subtleties of the given language and it realization.

The abundance of new notions makes difficult to understand the material property and doesn't assist in forming the structure of knowledge. Moreover, the details and subtleties of the concrete language version cover general notions and principles.

Second, basic actions of the industrial languages, namely, the calculation and assignment, are not visual and are performed inside the computer. As a result, the process of program execution is hidden from the student and he sees the final output only. The absence of visualization hampers the mastering of the semantics by the language operators. But special analysis shows, that numerous errors appear exactly due to the wrong understanding of the semantics by the language structures.

And, finally, practical application of the such languages as Basic, Pascal, etc., that is, the writing of the first simple but informative and interesting programs is possible only on learning a considerable language subset. As a result, the first steps in the programming learning, the most difficult ones, are not followed by the work on a computer.

The experience proves, that many beginners are far from being able to overcome the difficulties of the elementary programming study. But is their an alternative to the industrial languages? Are there any methods which would help the beginner to master the principles of programming? Specialists in the field of informatics repeatedly tried to develop special languages for the supporting of the initial steps in programming study.

The "turtle graphics" of Logo language - a simple and visual means to support the first steps of programming study - has played a great role in this process [1,2,3]. The successful application of the "turtle graphics" has stimulated the development of the new approach towards the elementary programming teaching. The beginners learn what programming is and study

how to control an executive on a display. An executive (the robot, the turtle, etc.) acts in a certain environment (microworld) and can perform a set of commands. It changes its position and the environment according to these commands. The executive can be controlled by means of a simple mini-language which, in addition to the commands to control the executive, includes more or less complicated structural statements (loop, condition, etc.).

The environment as well as the place of the executive in it is permanently displayed on the screen, and, thus, the student is able to see the result of command execution at once. When designing simple mini-language programs, one can make the executive to perform various interesting actions and watch the "animated pictures" on the screen. While learning a mini-language quickly, the student from the first steps begins to write and debug the informative programs that control the executive. During this process many of the fundamental notions of programming are learned easily and quickly, the skills of program development and debugging are acquired. For many students the experience they get while using this language as well as the general picture of programming is quite enough for achieving the needed level of "computer literacy". For those, who need more spacious information the experience of work with Ahe mini-language will essentially facilitate the studying in future and mastering a real language.

The work by means of mini-language is not the end in itself but a method of mastering a certain set of notions and skills. If this set contains not only the notions of programming but concepts pertaining to the fields of knowledge, a mini-language might be useful in other spheres. In this sense an "ideal" mini-language doesn't exist [2], because the type of executive and set of language controlling structures depend on the age and interests of the students. There is, for instance the mini-language SOLO aimed at the student of psychology [4].

At present two approaches to the realisation of mini-languages have been formed. In accordance with the first approach a mini-language are embedded (as Logo "turtle graphics") into one of the "big" languages of programming as a simple and obvious subset which can serve as the base for the learning. In accordance with the second approach an original mini-language is built for certain purposes of learning. The languages received with the assistance of these approaches we shall agree to call sub-languages and pre-languages correspondingly. Lately pre-languages have been more

actively created and used. The designing of pre-languages doesn't restrict the author's imagination when choosing the executive and language structure. By the beginning of the 80s numerous pre-languages have been designed to support various stages of the elementary programming learning and, consequently, were different in their complexity and richness. One can get the idea about various types of pre-languages by means of several languages to control the executive-robot which moves across the "world" of a CRT screen: Maze [5], Karel [6], Josef [7].

Some interesting pre-languages have been designed and put in practice in the USSR recently. Among them the following languages should be mentioned: alpha-world, beta-world and KuMir (at the Department for Mechanics and Mathematics of MSU), languages Tortoise and Turingal (at the Department for Applied Mathematics and Cybernetics of the MSU), languages of the environment Robotland (the Institute for Programming Systems). The description of these languages can be found in the publication of the Soviet journal "Informatika i obrazovanie" ("Informatics and education"). This work [8] contains some requirements based on the experience to which the pre-languages should satisfy so.that their usage will be effective. These requirements are:

1. Simplicity of a language (as well as its conceptual model [2]).
2. Visibility of interpretation.
3. Attractiveness in learning.
4. Dialogue style of a language.
5. Module style of a language.

In addition to these requirements two more conditions of the effective mini-language usage ale to be mentioned.

First of all, in order to design and debug a program in a pre-language a friendly programming environment is needed including a friendly editor and interpreter. The friendly editor "knowing" the syntax of the language will help the student to make the routine part of the work, i.e., the input and editing of the program easier and will allow to be concentrated on its informative part. By pressing one or two keys the beginning user will be able to input, delete, and move the commands and language structures. This makes the input of a program quicker and prevents many language errors. The friendly interpreter helps to look into the working program, watch the sequence of the statement execution, to debug the program step by step, and make experi-

ments over it. During the work with such an interpreter the current environment condition and the text of current program, in which the performed operator is stressed, are permanently displayed on the screen. When performing a command the interpreter immediately introduces changes into this environment, e.g., moves the robot on the screen, and stresses the next operator. The visualization facilitates the mastering of semantic of complicated controlling structures and helps to debug programs quickly. Friendly environment that includes the structural editor and visualization can be effectively used when teaching real languages, but for mini-languages the designing of such an environment is less complicated.and this only adds to a set of their merits. Ideally, the environment for a mini-language can include an intelligent component to assist in the course of a problem solution and to control the process of teaching [9,10].

Second, together with the language a set of informative problems should be developed. These problems must be interesting for the student both from the point of view a program-result received, and the process of solution development. The set must contain the problems of various complexity and encompass all necessary notions. If the student is able to solve these problems without any assistance it will become solving the informative problems by the student is the most effective way of the language mastering and, consequently, of the language-supported concepts. A mini-language, and environment for the work with it and a set of problems form the interconnected triad. The main component in it is, of course, the language, but, as the experience demonstrates, in case of missing of one of the components in the triad possible effectiveness of a mini-language is considerably decreased. Thus, for instance, a mini-language that looked very attractive and for which a sufficient number of problems was not invented looses its attractiveness very quickly.

In conclusion we shall analyze the problem of selection of control structures in a pre-language and their syntax (in a sub-language they are determined by an including language). As it was mentioned above more successfully are dealt the languages that have module and dialogue style. Modern programming languages offer a number of various structures of control. When selecting structures for a mini-language the age of the students should be taken into the consideration.

For younger students one or two simple structures are sufficient. Such a structure in Logo, for example, is the "repeat" loop. The older the students are, the wider a set of the used structures can be and more complicated are the structures. If the study of a pre-language is the preparation for the study of a real language (Fortran, Pascal, Modula-2) it is recommended to borrow the control structures from this language with the minimum adaptation. Such a choice greatly facilitates the future mastering of the control structures semantics of a real language. Some valuable advice on the organization of convenient and easy to learn structures can be derived from special literature on this subject [11]. An example a pre-language meeting all the requirements mentioned above can be a pre-language Turingal. This language is designed for the studying of elementary programming by the first-year students of mathematical departments and students of special secondary mathematical schools. It is aimed at the study of the main concepts of programming and theory of algorithms as well as preparation for the study of Pascal.

2. Pre-language for applied mathematics students.

The main requirement during the development of a pre-language described below was its ability to support the elementary stage of programming learning for the first-year students of the Department of Applied Mathematics and Cybernetics. The future professional mathematicians-programmers should learn as early as possible the principal concepts and skills of modern languages and programming. A pre-language for the students of this category must provide practical mastering of such fundamental notions as an algorithm, program, input data, result of execution, module, subroutine, condition, statement, etc.

The students should acquire the skills of structural programming, the development of programs from top to the bottom, decomposition of the problem into sub-problems, stepwise refining and debugging of programs. They have to get the idea about the main control structures of programming languages; loop, branching, selection and call of subroutine. Thus the necessary pre-language must provide the early backing of principles of the theory of algorithms and main concepts contained in Pascal. All the enumerated requirements allowed to make simple choice of the executive in a pre-language and structures of control over it. As an executive in a pre-language the Turing Machine executive was chosen which is studied among the

fundamentals of the theory of algorithms [12]. The environment in which the executive is acting is the endless tape divided into cells. Each sell contains one of the symbols of the given alphabet. The executive itself is a hypothetical automata surveying at a moment of time one of the cells on the tape.

This automat can perform three basic action: the shift to the right on the cell, the shift to the left on the cell, putting a given symbol into the current cell. In addition to that the automat can check what symbol is in the surveyed cell of the tape and on this base make decisions about the following actions. The programs for the traditional Turing machine are recorded as a table being in it the only structure of control over the sequence of actions. Unfortunately such a structure doesn't allow to master the most wide-spread in modern languages control structures of the loop, branching, selection and call of a subroutine. For the organization of a pre-language the decision was made to design the structures of control as similar to the structures of Pascal as possible. If one copes with them during the work with a pre-language, then mastering of the control structures of Pascal will become easier. The statements of a pre-language contained the elementary statements (commands) of the executive control, as well as the control statements, namely: combined, conditional, loop, selection and subroutine call.

The only differences from "classical" Pascal is the module and dialogue style of the language. All the subprograms of a pre-language are independent modules. All the elementary operators and call of sub-programs can be performed during the dialogue. These differences are conditioned by the requirements to the pre4anguage stated above. Finally, the transition operator was not included into the language that help the mastering of structural programming skills.

The new language was called Turingal, i.e., the combination in one language of the Turing machine executive and structures of control of Pascal (Turingai = Turing automata + Pascal, as Turingoi [13]). As it was said above, Turingal is the language of dialogue style. It means when the user input any command of Turingal, then the interpreter performs it immediately. The result of execution is visualized, that is, the changes are introduced into the environment displayed on the screen. For the chosen executive the visualized environment is the picture of a section on the Turing machine tape, under one of the cells of it a special symbol is placed which reflects the current position of the automata.

When the elementary operators "->" (shift to the right), "<- (shift to the left) and <symbol> (the print of symbol) are inputed, the automata symbol is shifted on the next cell, or a printed symbol appears in the cell above it. The user can input each elementary operator by pressing single key. Using the elementary operators in the dialogue mode the user can achieve any desirable condition of the environment. Besides the elementary operators the student can use the names of the modules (subroutines) defined by him in the course of a dialogue. The interpreter supports a set of modules defined by each user, which is displayed on the screen. When the user input the name of one of the modules defined, the interpreter begins to perform the module step by step. While doing it the result of every elementary operator is immediately displayed on the screen. In addition to that, during the module execution its flow of control is visualized, i.e., the current operator is stressed. The visualization of the flow of control helps the student to understand the semantics of the control structures of Turingal.

The module of Turingal is a string of symbols. The operators in the line are separated from each other by the blanks which are the meaningful symbols of the language. The string style of a module is the distinctive feature, of Turingal and makes possible to display simultaneously on the screen all the modules defined by the student and visualise actions on a modules (subroutines) call. Such a syntax supports and encourages modern structural and module style of programming. The semantics of a module-subroutine on the whole as well as control operators (combined, conditional, loop, selection and subroutine call) is absolutely analogues with the corresponding operators of Pascal and that is why it doesn't require any explanations here.

The semantics of the elementary operators has been discussed above. The semantics of conditions should be analyzed in more details. The elementary condition in Turingal is just the sequence of symbols. At any moment of a program running it is "true" only when the symbol viewed by the automaton coincides at least with one symbol of the sequence. The inversion of the elementary condition "the sequence of symbols" is "true" only when the symbol viewed by the automaton doesn't coincide with either of the symbols of sequence. All the sub-programs defined by the student have equal rights. The call of any of them, including direct and hidden recursion, is allowed. In this sense the set of Turingal modules is semantically equivalent to the

program of Pascal containing the set of subroutines of the top level. All these facts prove the similarity of the controlling structures of these languages.

3. Conclusions

The pre-language Turingal was designed in 1985 at the Department of Applied Mathematics and Cybernetics, Moscow State University. During several terms this pre-language was the first language for teaching programming at some classes of the first-year students of the department and of the Evening Mathematical School at the department. The set of more than 100 problems was designed for Turingal. The language Turingal and the environment for it were realized on the Soviet mainframe computer BESM-6. The language proved to be a good means at the initial stage of teaching programming. Later, the intelligent tutoring system has been developed for the language Turingal, which together with the environment of the language has formed the intelligent educational environment [10].

REFERENCES

1) Mayer R.E.: The psychology of; how novices learn computer programming, Computing Surveys, 1981, v.13, n.1 -- P.121-141.

2) du Boulay J.B.H., O'Shea T., Monk J.: The black box inside the glass box. Presenting computing concepts to novices, Int. Journ. Man-Machine Studies, 1981, v.14, n.3. - P.237-249

3) Coombs M.J., Gibson R., Alty J.R.: Learning a first computer language: strategies for making sense, Int. Journ. Man-Machine Studies, 1982, v.16, n.4. -- P.449-486

4) Eisenstadt M.: A user-friendly software environment for the novice programmer, Communication of ACM, 1983, v.20, n.12. - PA 058-1064

5) Mawaddat F.: Another experiment with teaching of programming languages, SIGCSE bull., 1981, v.13, n.2. -- P.49-56

6) Pattis R.E.: Karel - the robot, a gentle introduction to the art of programming –– London: Wiley, 1981

7) Tomek I.: Josef, the robot, Computers and education, 1982, v.6, n.3. -- P.287-293.

8) Brusilovsky P.: Languages to study introductory programming (in Russian), Informatics and Education, 1990, v.5, n.2. p. 3-9

9) Witschital P., Stiege G., Kuehme T.: First experiences with the learning game "TRAPS" - in: Artificial Intelligence and Education. Bierman D., Breuker J., Sandberg J. (Eds.). Proc. of the 4-th International Conference on AI and Education, Amsterdam: IOS - p.323-329.

10) Brusilovsky P.: Intelligent environment to study introductory programming (in Russian), Gritsenko V., Dovgyallo A. (Eds.). The use of computer technologies in teaching. — Kiev, 1990, - p.40-48.

11) Shneiderman B.: Software psychology: Human factors in computer and information systems. — Wintrop Publishers, Cambridge, MA, 1980.

12) Bohm C., Jacopini G.: Flow diagrams, Turing machines and language with only two formation rules, Communications of the ACM, 1966, v.5, n.5, p.366-371.

13) Pratt T.W.: Programming languages: Design and 'implementation. — Englewood Cliffs: Prentice Hall, 1975