INFORMATICA, 1994, Vol. 5, No. 1-2, 43-54

ALGORITHMS AND PROGRAMS AS DIFFERENT ITEMS IN LEARNING OF COMPUTER PROGRAMMING

Gintautas GRIGAS

Institute of Mathematics and Informatics
2600 Vilnius, Akademijos St.4, Lithuania

Abstract. A development of algorithms and writing of programs are considered as closely related but not identical parts of computer programming. Some differences between them are important for learning of computer programming, in particular, in distance learning. These differences are identified and discussed from the pedagogical point of view. The arguments for the selection of pedagogical based and cost–effective delivery modes in the case of distance learning are investigated. Practical examples supporting theoretical arguments are given on the activities of Lithuanian schools.

Key words: algorithm, delivery mode, distance learning, distance teaching, electronic mail, function, high school, Pascal, problem solving, programming, procedure.

1. Evaluation of the relationship between an algorithm and a program: a historical perspective. The computer science relies on computers, i.e., hardware and software. The computer technology is rapidly changing. Those changes affect the computer science and its notions. We will briefly discuss how such changes influence the relationship between an algorithm and a program, and how computer science curriculum may benefit from this situation.

In this context we consider the algorithm as a description of a computer–oriented solution of a problem expressed

in terms comprehensible to man. The reader (and user) of the algorithm is a human being.

By the program we mean a description of a computer oriented solution of a problem understandable (executable) by computer.

Thinking abilities of a man change not so rapidly as that computational power of a computer. Thus, the level of abstraction in algorithms today is about the same as it was half a century ago. However, it is not the case with a computer program. Three different notions of a program and their distinct relationship with the notion of algorithm may be observed in a short period of the history of the computer science (see Fig. 1.).

1. Early computers were able to understand (i.e., to execute) only the programs written in their own (binary) code or – in the best case – in their assembler language. All those forms were inappropriate for a human reader. Thus another notation was necessary for the algorithm. The algorithm was usually expressed in the form most understandable to a human reader – a flow diagram. Thus the algorithm and the program were expressed by completely different notation. A human translation of the algorithm to the program was necessary. The development of an algorithm was often called by programming and that of the program – by coding.

2. The advent of algorithmic languages changed the situation. These languages enabled to express a computer–oriented solution of a problem in the form acceptable both to man and to computers. The first truly algorithmic language was Algol–60.

The same text expressed in an algorithmic language could be considered as an algorithm or as a program. A compromise between a man and a computer was reached. Consequently, the differences between the algorithm and program became minor. The difference between the terms of "algorithmic lan-

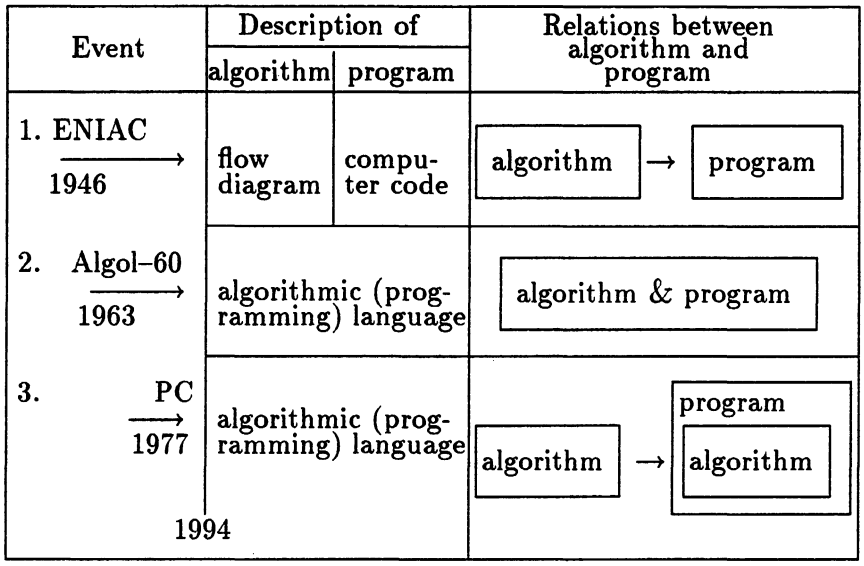

Fig. 1. Alternating relationship between an algorithm and a program, initiated by the first computer (1) and altered by the invention of an algorithmic language (2) and a personal computer (3).

guage" and "programming language" has also disappeared, and these terms were often used as synonyms.

Publishing algorithms as Algol–60 procedures became a common practice about 30 years ago.

3. Personal computers considerably changed the relationship between a computer and its user. The user was given a direct and permanent access to a computer. Graphics and colors enabled to present the results of computations, including a dialog with a computer, in a very expressive manner. All these powerful facilities require extra programming efforts and result in a great number of additional lines in the program text. Those texts include commands to control and output the data flow, to organize a dialog between a computer and its user, to

visualize data on a display. Thus, the job of a programmer splits into two parts:

1) to obtain the results from the input data;
2) to organize a dialog between a computer and its user (including acceptation of the input data and presentation of the results).

Indeed, these two parts were present in the past, too. However, the size of the second part was negligible. Now it grew up and became comparable with that of the first one. The outcome of the both parts may be expressed in the same algorithmic (programming) language. However the main task of the first part is development of an algorithm (especially, using high level algorithmic languages). However, the job concerned with the second part is much more related to the computer. So, it is reasonable to consider the outcome of the first part as an algorithm and that of the second part as a program. In this sense, a program becomes an extension of an algorithm (see Fig. 1.).

Now commonly accepted language for writing and publishing algorithms is Pascal. Its original Report (Wirth, 1972), international (ISO, 1983) and national standards support development of algorithms. Particular implementations of Pascal provides the language with extra elements enabling it to express actions exclusively related to programs in our sense, namely, colored and animated vision on the display, sounds. Thus, a program being an extension of the algorithm is to be written in the extension of t he algorithmic language. A rich and widely used example of the Pascal extension is Turbo Pascal.

Thus, we have defined two related objects – algorithm and program – with a slightly restricted meaning than that used in common practice.

The main task in the process of the development of an algorithm is to find a computer–oriented solution of the given

problem. So, it is reasonable to call this process by problem solving. Programming is a conventional term to name a process of program development (writing). However, it is widely used in a broader sense, i.e., including all phases of the development of computer–oriented solution of a problem, starting from the problem formulation and finishing with a human analysis of the results produced by a computer. We use "problem solving" in this broader sense.

By program writing, we mean the development of a program in our restricted sense.

Thus to define the processes of development of the above mentioned objects we use two terms: problem solving (for algorithms) and program writing (for programs).

2. Programming in secondary and high schools. Computer programming is the main part of the subject of computer science (also called by informatics or computing) in secondary and high schools. This in accordance with IFIP recommendations for informatics education in secondary schools (Taylor, Aiken and van Weert, 1991).

Computer programming includes teaching both problem solving and program writing. For the same immediate goal – to develop a solution of a given problem by computer – there is no strict difference between problem solving and program writing. However, they have different influence on the ultimate and more general goals of teaching. The development of an algorithm requires deep and flexible thinking, sometimes even real brain storming. That develops thinking, especially logical thinking abilities of school students. Program writing requires a lot of a very careful work, design and formatting skills. That develops another abilities of school student.

The proportion between teaching of these two topics of programming may vary in a great extent dependent on the balance of immediate and ultimate goals as well as on the accessibility to computers for school students. Problem solv-

ing requires more brain work, while program writing requires more computer time.

A strong tendency to reorganize the teaching is observed so that school students were given a possibility to develop individual or group projects instead of strict and synchronous following the schedule preassigned by the curriculum (Tinker and Thornton, 1992; Pearlman, 1993) . School students learn the theory themselves when there arises such necessity in the development of the project. Due to its practical direction, computer programming suits such a method of teaching perfectly. According to these lines, programming problems may be classified into two parts, such as:

1) (algorithmization) problems;

2) (programming) projects.

The problems resemble puzzles. Their solutions are short and they result in algorithms.

Programming projects resemble the projects of other school subjects. Their development requires a long work and they result in a computer program (programming product). Collaborative or group work is encouraged.

Such a classification leads to the structurization of teaching to computer science (informatics) and may enhance the effectiveness of teaching (Dagienė, 1993).

3. The experience with distance teaching. The idea to consider algorithms and programs as separate items came into being in the process of distance teaching to programming (in a broad sense). We have been working with two schools of programming:

1) Lithuanian Young Programmers' School;

2) Programming practice sessions by electronic mail.

The Lithuanian Young Programmers' School was established in 1981 (Grigas, 1990). It is a nationwide school. All communications between teachers and students are performed by land mail correspondence (letters and printed matter).

Personal microcomputers have not yet been available at the moment when the School started its activity. According to our historical review given above it was the time of the second period. At that time there was not so clear distinction between an algorithm and a program. An attempt to teach both problem solving and program writing was made in the first years of the School. A great number of interesting and attractive programming problems was developed.

Along with learning by correspondence the school students are invited to a campus practice for two weeks during the summer holidays.

Although a number of arrangements were made towards program writing, problem solving was greater success than that of program writing.

Naturally and gradually the school transformed itself into the school of problem solving and survived like this until now. Its survival gives us an evidence that problem solving can be considered as the proper part of programming for distance teaching using land mail for communications.

In 1992 we began the experiments of programming practice sessions by electronic mail (Grigas, 1993). School students are working in teams. The number of the members in a team is not limited. Each team is given a programming project and has to develop a mini programming product with the documentation including a brief description of the idea of problem solution, annotation of the program, a user's guide, a programmer's guide.

The session is split into 4 phases. During the first phase each team develops its own initial version of the programming product.

In the second phase the teams get acquainted with the initial versions of programming products made by other teams. Each team must write reviews of 3–4 products.

During the third phase each team is given an opportunity

to improve the preliminary version of their own products. The work ends with the final version of the programming product.

The last phase is for discussion.

All communications between teachers and teams are carried out by electronic mail. The elements of competition are included into the sessions. The teams are ranked according to the evaluation of their work during the first three phases.

Programming practice sessions include both problem solving and program writing. However the emphasis is put on program writing and documenting. It is achieved by a set of requirements for the programs to look like a real programming product, i.e., they must be user–friendly and equipped with a comprehensive dialogue and an expressive presentation of results.

We have had three sessions so far. In the first session (November–December, 1992) 6 teams took part from 6 schools, in the second (April, 1993) – 15 teams from 14 schools, and in the third (November–December, 1993) – 21 team have participated. The participation was voluntary.

On the basis of the results achieved by teams and the outcome of discussions we arrive at the conclusion that such kind of learning was a success. We suppose that among the reasons for the success was an information transmission mode. The electronic mail has a distinguishable feature to transmit the program text executable by computer. However, the executable text is not only the information per se, but also a carrier of all the events, observed on a computer display during the execution of a program, with the whole variety of colors, sounds and animated pictures.

4. Matching of the parts of programming with information delivery modes in distance learning. We have discussed two parts of programming – problem solving and program writing – as well as two information deli very modes in the case of distance teaching: by land mail and by elec-

tronic mail. A positive (effective) match is denoted by + and the negative one (non effective) by – in the Table 1.

Table 1. The match between the parts of programming and a delivery mode (the type of mail).

Part of programming	Delivery mode (mail)	
	land mail	electronic mail
problem solving	+	+
program writing	–	+

We see that the electronic mail suits both parts of programming. However, the electronic mail is not yet available for all school students. This situation is similar in a number of countries.

The distance learning by land mail is available for all school students. However it is considerably slower and suits only for learning of problem solving.

We see an acceptable solution of the distance learning problem by combining both delivery modes in a pedagogical based and cost–effective way. It is reasonable to divide the subject of programming into two overlapping parts:

1) problem solving (possibly with some elements of program writing necessary for testing algorithms by computer);
2) program writing (possibly with elements of problem solving).

Problem solving is delivered by land mail, as a rule, and optionally – by electronic mail.

We have already discussed the learning of problem solving by land mail and that of program writing by electronic mail.

Let us discuss still another way of learning to problem solving.

The learning of problem solving by electronic mail can be performed as a collaborative development of algorithms. School students are given a problem by electronic mail and have to write an algorithm and to deliver it to the coordinator by the electronic mail as soon as they obtain preliminary solution. The coordinator immediately distributes the text of the algorithm among all participants of the session by electronic mail for revision and improving. The coordinator waits for new original solutions and for the improved old ones. The session is over when ideas for solution are exhausted and the solutions become relatively perfect, i.e., such that school students (or teachers) can not offer further improvements. Such a method of learning has already been tested in the face-to-face session. It proved efficient.

5. Impact on the learning activities. The distance learning of programming is not completely isolated from other school activities. Let us briefly comment the impact of the distance learning methodology given above on the face-to-face teaching of informatics in high schools and on national Olympiads of informatics.

Informatics is an obligatory subject in Lithuanian high schools. It helps school students get acquainted with fundamental principles of transmission, storage and processing of information. The main notion of its processing is that of algorithm. In order to get acquainted school students with algorithms, the problem solving part (in our sense) is included into the curriculum and presented in a textbook (Dagienė and Grigas, 1991). Pascal procedures and functions (not programs!) are used for the presentation of algorithms. This enables the students to concentrate themselves on essential features of an algorithm: how to get the results from input data without wasting their time and energy on details connected with data formats, reading and writing commands. In order to enable

the students to run directly on the computer their algorithms expressed by Pascal procedures and functions, an interpreter of Pascal procedures and functions was developed (Dagienė and Žandaris, 1992) as well as a special methodology of the use of Turbo Pascal (Dagienė, 1993) for such a purpose.

6. Conclusions. 1. The relationship between an algorithm and a program is changing in time. In the contemporary period of personal computers it is reasonable to consider a compact description of actions leading from input data to output data as an algorithm with no regard to the data presentation form. The actions are expressed in a form acceptable for a human reader. The algorithm is included into the program. The program contains detailed commands necessary to supply the algorithm with input data, to visualize output data, ensure a dialog between a computer and its user. Problem solving is a process of the development of an algorithm. Program writing is a process of the development of a program.

2. It is reasonable to divide the whole process of teaching of programming into two parts:

1) teaching of problem solving;

2) teaching of program writing.

The former requires more brain work for a student, the letter – more careful work on a computer.

3. In the case of distance learning, problem solving may be delivered by land mail or by electronic mail almost equally effectively. However, an electronic mail may be recommended for a delivery of program writing as a topic.

Acknowledgments. This work is supported by UNESCO under contract CII 408.018.2.

REFERENCES

Dagienė V. (1993). A pseudo-direct execution of algorithms using Turbo Pascal applied in programming teaching process. *Informatica,* **3**(4), 29–502.

Dagienė V. (1993). Algorithmization in Secondary School. Ph.D. disertation. University Vytautas Magnus, Kaunas (In Lithuanian).

Dagienė V., and G. Grigas (1991). *Informatika. 10–12.* Šviesa, Vilnius.

Dagienė V., and A. Žandaris (1992). Interpreter of algorithms. *Informatika,* **20**, 40–52.

Grigas G. (1990). Some aspects of teaching the art of programming by correspondence. *Informatica,* **1**(1), 156–166.

Grigas G. (1993). An experiment of computer programming practice by e-mail. *Interpersonal Computing and Technology: an Electronic Journal for the 21st Century,* **1**(2),(Retrieve as GRIGAS IPCTV1N2 at LISTSER@GUVM.GEORGETOWN.EDU)

ISO (1983). The programming language Pascal. ISO Standard 7185.

Taylor H.G., R. M. Aiken and T. J. van Weert (1991). *Informatics Education in Secondary Schools.* IFIP Working Group 3.1. Guidelines for Good Practice.

Tinker R.F., and R. K. Thornton (1992). Constructing Student Knowledge in science. In Scanlon E., O'Shea T. (Eds.), *New Directions in Educational Technology,* Berlin–Heidelberg, Springer–Verlag. pp. 153–170.

Pearlman R. (1993). Designing the new american schools. *Communications of the ACM,* **36**(5), 46–49.

Wirth N. (1972). Programming Language Pascal: Revised Report. *Berichte der Fachgruppe Computer – Wissenschaften,* **5**, Zurich, ETH.

Received March 1994

G. Grigas received the Degree of Candidate of Technical Sciences from the Kaunas Polytechnic Institute (Kaunas, Lithuania) in 1970. He heads the Department of Systems Programming at the Institute of Mathematics and Informatics. His research interests include abstract data types, programming methodology and teaching.