# Intelligent Tutor, Environment and Manual for Introductory Programming

P.L. Brusilovsky, International *Centre* for Scientific and Technical Information, Moscow

## INTRODUCTION

Programming is perhaps the most computerized of all academic disciplines. Special computer systems help teachers and students in all kinds of programming activity. Computer-assisted instruction (CAI) systems -`coachers' and `tutors' - help teachers in explaining new material, presenting examples, and giving and checking various tests. Students can independently design and debug programs with the help of student-oriented programming environments. Efficient access to the previously learned information can be accomplished with the help of computer manuals and online help systems.

Special attention is needed while creating integrated systems which support the learning and tutoring of programming. By an integrated system we mean a system that is made up of several specialized components. Good examples of integrated systems are office integrated systems which are capable of doing work of text processors, spreadsheets, database management systems, etc. The advantages of integrated systems in the field of studying programming are quite obvious. The number of different systems that a student simultaneously needs decreases, thereby making it comparatively easy to study and use them. As a result the student's learning energy is spent in learning the material (the technology and language of programming), not learning how to use these systems.

Combining a tutoring system, learning environment and electronic manual is not enough to create an effective integrated system to study programming. An integrated system ought to be more than just the sum total of a number of specialized components. Thus, for instance, the history of a student's work with any of the components should be analysed and stored by the system in a special student model. This model can then be used by all the components in adapting to the knowledge level and style of work of the particular student.

The use of the artificial intelligence (AI) techniques is quite successful in designing effective integrated systems. According to McCalla and Greer (1987), intelligent computer-assisted instruction (ICAI) attempts to bridge the gap between traditional CAI systems and microworld environments. There exist a few intelligent tutoring systems in the field of programming that are able to work as tutoring or coaching systems as well as to create a friendly environment for novice programmers; namely: BIP (Barr, Beard and Atkinson, 1976), Lisp-Tutor (Anderson and Reiser,1985), ReGIS (Heines and O'Shea,1985), Bridge (Bonar and Cunningham, 1988), IPTS (Cheng, Hu and Yang, 1988), GEL (Reiser, Ranney, Lovett and Kimberg, 1989), and Ugo (Innocenti, Massucco, Persico and Sarti, 1991).

One of the possible ways of constructing such an integrated intelligent system for programming is suggested in (Brusilovsky, 1987). In this article we shall describe an integrated Intelligent Tutor, Environment and Manual for Introductory Programming (ITEM/IP), its design based on this approach. ITEM/IP supports a course on introductory programming based on the mini-language Turingal. The mini-language serves as a tool in mastering the main concepts of programming, programming languages' structures and skills in programs design and debugging. The method of learning introductory programming with the aid of mini-languages is quite well developed in the USSR now. The language

Turingal is similar from this point of view to such mini-languages as Logo Turtle (Papert, 1980), Karel (Pattis,1981), and Josef (Tomek,1982).

## DESIGN OF THE SYSTEM

### The philosophy of ITEM/IP design

ITEM/IP was specially designed as a combination of close interacting components, supporting the needs of a user during the the learning process. The student can use the system in the following modes:

1. *Instruction. To* the inquiries of the student, ITEM/IP offers the optimal teaching operation: explains concepts, demonstrates semantics of Turingal constructions and checks out comprehension, presents various programming tasks as examples or gives them out as problems to be solved. The optimal operation is selected using a built-in strategy that takes into consideration all the knowledge that ITEM/IP has about its domain and student. If the student is not satisfied with the embedded strategy he can also choose from a list of relevant teaching operations.
2. *Programming* laboratory. In order to solve various problems and also to work with personal programs the student can make use of a complete set of tools for program design and debugging known as the programming laboratory. One of the laboratory's functions is to display visualizations of the student's programs. With the help of the laboratory the student can observe the programs `at work', experiment with them and gradually learn from his observations and mistakes.
3. *Online Manual.* A student can demand at any time a re-explanation of past material or a second analysis of problems. The extent to which material is explained by ITEM/IP is inversely proportional to the current student knowledge about the material. Therefore a repeated explanation is usually more concise than the original one. This mode offers a reference access to the learned material of the course.

ITEN1/IP ensures the continuity of the student's work during various modes. The student's results obtained while working with each of the environment's various components are stored and used by other components in order to adapt their working to his knowledge level. Artificial intelligence techniques are used in order to reach this goal.

### Overview

The main components of ITEM/IP (see Figure 1) are the pedagogical module, which enhances the choice and realization of teaching operations, the programming laboratory, which enables the student to work independently, and the information kernel (Brusilovsky, 1989) including all factual knowledge and information existing in the environment. The programming laboratory includes an editor and interpreter for the mini-language. The information kernel includes Domain model, Student model and Base of teaching operations.

The conceptual architecture of ITEM/IP resembles that of intelligent tutoring systems (ITS). A typical ITS has four major components: pedagogical module, domain expert module, student model and user interface (Wenger, 1987). The programming laboratory takes the place of the domain expert module in the ITEM/11? architecture. This reflects the state of things: in ITEM/IP the domain expert is replaced by the interpreter of the mini-language with the same abilities as the expert. The same technique was used successfully in BIP (Barr, Beard and Atkinson, 1976), QUADBASE (Hudson and Self, 1982), and ReGIS (Heines and O'Shea, 1985) systems. The structure and functions of ITEM/IP components are given below.

### Information kernel of ITEM/IP

The Domain model (Domain structure) is a network of concepts. Three types of concept have been singled out to represent the programming knowledge in the model: the programming concept, mini-language construction and the skill to use any construction in the language. Every concept is described within a frame whose slots contain special information about the given concept, the procedures of its presentation, and an indication of its relations with other Domain model concepts.

Certain types of directed relationships are used in the Domain model. General-specific relations

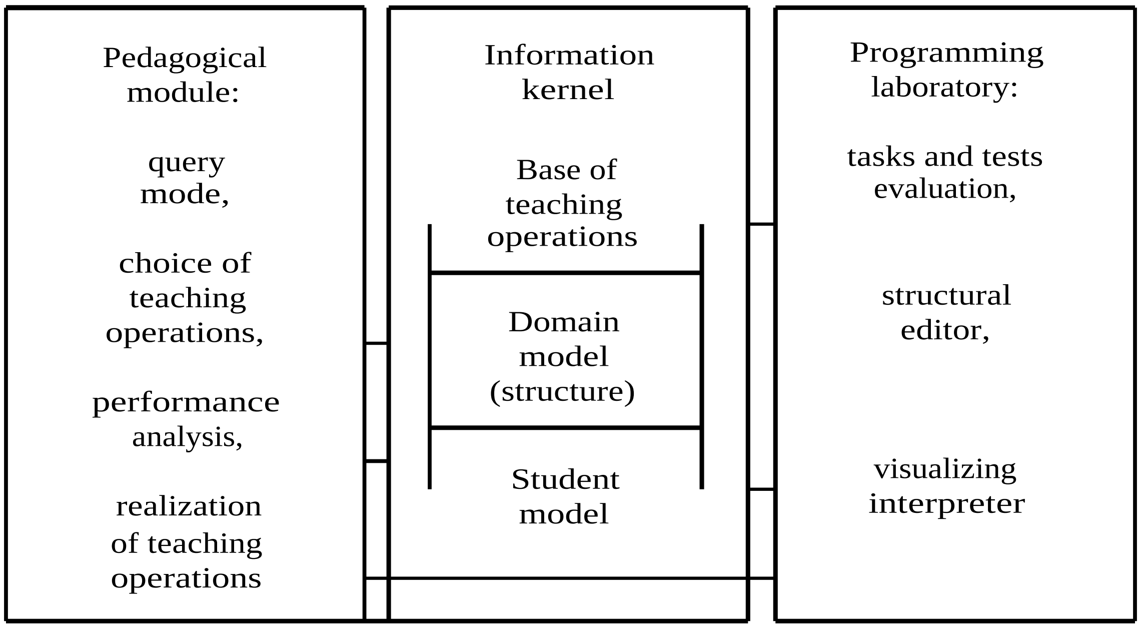


Figure 1. *The structure of ITEM1IP*

represent a hierarchy of concepts with a varying degree of generality. `Utilize' relations link the constructions and skills of its usage. Predecession relations describe relative order of concepts' presentation. The network obtained reflects the structure of the domain studied. Therefore we shall name this type of Domain model structural-network type (Figure 2).

The Student model, a kind of overlay model, is based on the Domain model. For every Domain model concept the Student model contains several integer counters for the purpose of indicating the extent to which the student has mastered a concept. Besides a set of counters the Student model includes optimal difficulties for various kinds of teaching operation. The Student model is always kept up-to-date and supports adaptive work of all the environment modules. This form of Student model helps to define, at any moment of the learning process, what and to what extent the student has acquired (or not acquired) the material, and to accurately tune in to his level of knowledge and requirements.

The Base of teaching operations is a set of frames of three kinds: concept frames mentioned earlier, example frames and task frames. The pedagogical module uses these frames to build teaching operations of five kinds. Every frame of the Base is adapted to the organization of presentation type and test type, teaching operations and to the repetition of previously learned material. Independence of frames makes it possible to expand the Base with ease.

Example frames and task frames are the two main kinds of Base frame. An *example frame* includes the text of the example (the language construction) and a set of input data `tests' by which this construction can comprehensively be demonstrated. Using this data the pedagogical module organizes a visual demonstration of the construction's semantics and `mental execution' tests checking the acquisition of semantics.

*A task frame* includes a title, a problem text, a set of test data, a variant of the plan, a model solution and the task complexity. A problem can be analysed as an example or given to the student to be solved. A special slot is used to link the Base frames with the Domain model. This slot lists all Domain model concepts related to the given frame. This list is called the spectrum of the frame. The spectra change a set of independent frames into a set of exact instruments for the directed process of learning. According to information from the Student model the pedagogical module

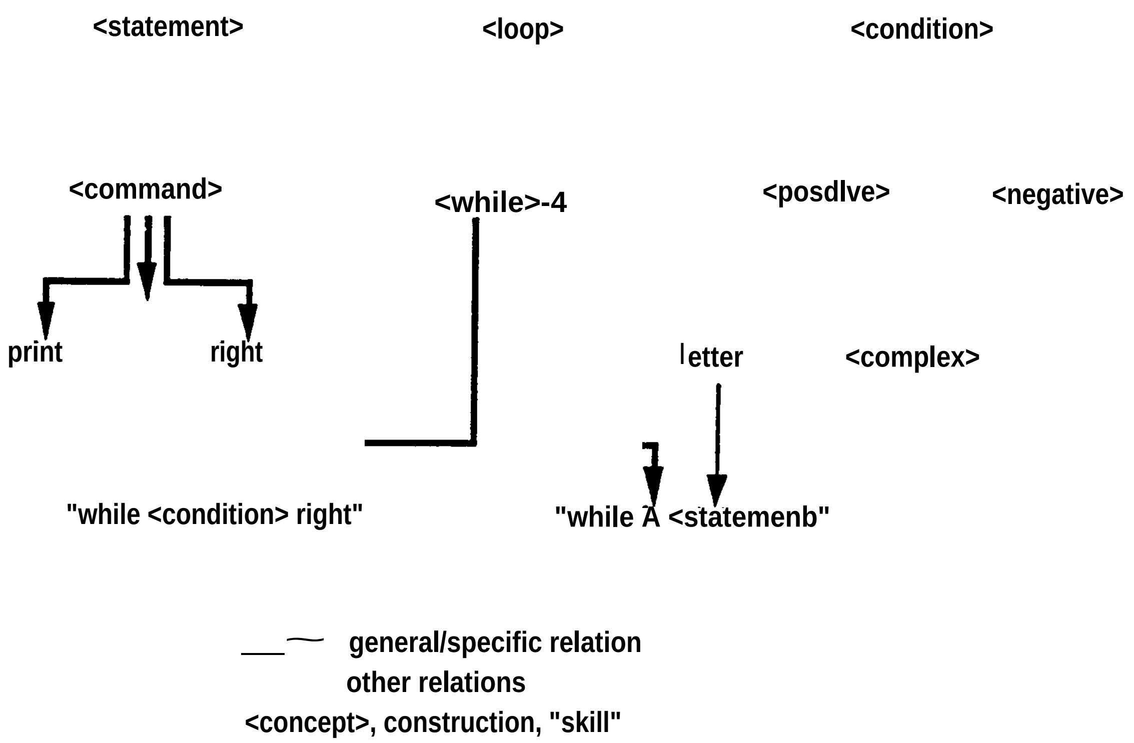


**Figure 2. A** *fragment of structural-network Domain model*

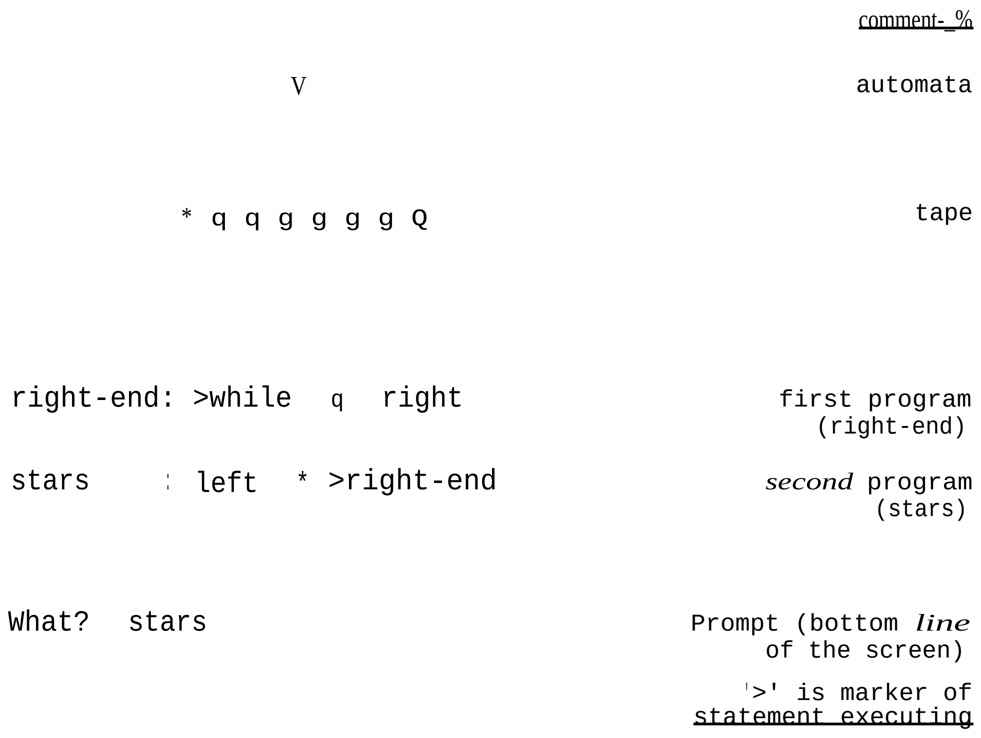
determines (a) which of the concepts have already been mastered, (b) which need to be worked on, (c) which new concepts are ready to be studied and (d) which concepts are not ready to be studied. The presence of frames spectra makes it possible to compile a list of teaching operations relevant at the time (with spectra of (a)-(c) type concepts) and then choose from it the optimal one with a special strategy.

## Programming laboratory

One of possible ways to study introductory programming is to design, debug and investigate real programs with special mini-languages. The mini-language Turingal was specially designed for academic use (Turingal = Turing automata + Pascal, as Turingol (Pratt, 1975)). This language is a way to control the well-known system of algorithmic theory-Turing automata which works with a tape of symbols. The elementary operations of the mini-language are simple and visual: the movement of automata left and right along the tape and typing the symbols on the tape. To control the automata Turingal offers a wide range of control structures (conditional statement, loops, case, sub-routines) with syntax and semantics similar to the structures of Pascal. One of the main reasons for learning the mini-language is to master the semantics of these well-known structures.

The programming laboratory is a set of special ,software tools'. The tools support the design, debugging and investigation of Turingal programs by the student. The laboratory includes a structured editor which makes it easier to enter and correct a program, and a visual interpreter which enables the student to run the program step by step and observe its work. The interpreter visually displays all automata moves and symbol typing on the tape and also marks the current operator. Figure 3 shows the screen while working in the environment.

The ITEM/IP programming laboratory, like the BIP (Barr, Beard and Atkinson, 1976), and the ReGIS (Heines and O'Shea, 1985) laboratories, supports `the experiment style' of working: design



```
V

* q q g g g g Q

right-end: >while  q  right

stars     : left  * >right-end

What?  stars
```

comment-_%
automata
tape
first program (right-end)
second program (stars)
Prompt (bottom line of the screen)
'>' is marker of statement executing

Figure 3. *Turingal environment: tape, automata and programs. Program `stars' is executed*

the program - experiment (interpret the program) - observe - change the program - experiment again, etc. This style of work enables the student to learn from experience the concepts and constructions of the Domain model, to `discover' their features, and to get an idea of programming technology.

The student uses the programming laboratory in an independent mode for individual work and for solving tasks presented by the pedagogical module. The pedagogical module itself uses the laboratory in the controlled mode for domain expertise. The interpreter is used, in particular, for visual demonstration of construction semantics, control of `mental execution' tests, presentation of problem conditions, testing and analysing their solutions. In both these modes, the programming laboratory components adapt their work to a specific student using the student model. It is on the basis of the student model, in particular, that the extent to which various language constructions must be visualized is determined: the less studied the construction is, the more detailed the visualization.

## Pedagogical module and teaching operations

The pedagogical module controls the process of learning at the macro and micro levels. On the macro level the pedagogical module compiles a list of teaching operations relevant at the given moment and chooses the optimal teaching operation. On the micro level the pedagogical module manages a dialogue with the student during the realization of the teaching operation, chosen at the macro level or by the student himself.

Besides this, the pedagogical module evaluates (with the help of the programming laboratory) the student's results while working with each teaching operation and then accordingly changes the student model. More exactly the pedagogical module changes (to a different extent) the counters of all concepts of the teaching operation spectrum. To determine the extent of change, the pedagogical module takes into consideration the Domain model's relations and previous values of counters, thus limiting the `noise' in the Student model.

The pedagogical module deals with the five main kinds of teaching operation: presentation, demonstration, test, task analysis and task to solve.

Presentation of Domain model concepts is given right at the beginning of the student's acquaintance with the concepts and also if repetition is asked for. While introducing a concept, its description as well as information about its relations with other concepts is given. The description of each concept is stored in the concept's frame. The text of the description can be broken up into fragments. Each fragment is connected with the threshold value of one of the student model counters. The fragment is not presented if the counter for the concept exceeds this threshold. According to the method given above, during a re-presentation (presentation as repetition) of concepts, descriptions are usually more laconic. Information about the relations of a given concept is not stored but generated by the system corresponding to the type of relation. While presenting a concept, information is generated only about relations with previously studied concepts. During the re-presentation, more complete information about relations is usually given, since by this time a greater number of concepts have been studied.

Demonstration of language construction semantics is carried out with the help of the visualizing interpreter using example frames stored in the Base of teaching operations. The behaviour of the chosen construction is first demonstrated on a set of tests, and then the student has a chance of experimenting (substituting his own data and editing the construction).

Tests that check out the extent to which a construction's semantics has been acquired are organized on the base of frames similar to that of the demonstration. In a test, the student is given the initial state of the tape (test), and then, having mentally executed the construction tested, can enter the result. The answer is then compared with the result of the domain expert interpretation. In case of a mistake. the correct `behaviour' of the construction is immediately shown to the student.

Programming tasks are important kinds of teaching operation. It is during the process of solving these tasks that a student can thoroughly understand various concepts and constructions and learn to use them properly in order to achieve the goals placed before him. The pedagogical module places a task before the student, helps him in solving it and then checks out the resulting program. Text of the given problem is displayed and then explained in the form of the 'test-result' pairs. Help can be asked for and is given in the form of additional explanations. A rough plan for solving problems and hints as to how the program works (visualization of model program actions without viewing the text of the actual program) is also given. Solutions are checked out by the interpreter which consecutively compares the student's program results on the given set of tests with that of the model program. Should the results of any of the tests not coincide, the mistake is pointed out. This mistake is `explained' by a visual demonstration of the wrong behaviour of the student's program. Our experience has shown that this is a very effective remedial method.

An analysis of solutions for programming tasks gives the student a practical idea as to how to use the skills he has learned. During the analysis the student is consecutively given the text of the problem, offered a plan of action and, finally, given a model solution whose working is demonstrated on tests. Next the model program is passed on to the student for experimentation. Any previously analysed or solved problem can be re-analysed on demand.

## A GENERAL VIEW OF THE **LEARNING PROCESS**

Students spend most of the time in ITEM/IP working in the programming laboratory. With the help of the main menu, the student can at any time call the teaching component for a new teaching operation, information, help, or for checking the task solution.

Choosing an item `Teach me' from the menu, the student asks for the next optimal (from the pedagogical module point of view) teaching operation. The optimal operation is chosen using the Student model and built-in teaching strategy (Brusilovsky, 1987). The human teacher can tune the teaching strategy to match his way of tutoring. He specifies the goal for a particular learning stage in the form of a set of Domain model concepts to be learnt at this stage. He can also specify an ,order' in the teaching course in the form of goal sequences of various stages. Each student can have his personal order of learning.

Thus, the pedagogical module has at its disposal all the components needed, according to the `Hartley framework' (Hartley, 1973), for adaptive teaching: domain representation network, on the basis of which learning goals are specified, the teaching operations, the student model and the learning strategy - 'means-ends' guidance rules, which make it possible to choose the best means (teaching operations), corresponding to the learning goal and to the knowledge level reflected in the student model. The learning strategy, as we have already mentioned, is built into the pedagogical module (ie it is represented in a form of a program and then compiled).

If the student is not satisfied by the strategy of ITEM/IP, he can choose a teaching operation himself. By selecting an item `What next' on the main menu he can get a list of all teaching operations relevant at that stage and then choose one of them. Selecting the item `Repeat' on the main menu the student can demand a replay of previously learned material. By using the bottom-level menu, he can obtain a list of all previously studied concepts, presented examples, solved and analysed tasks. He can choose any of these teaching operations and it will be presented again according to the type of operation chosen. In case of concepts which have not been learnt well or understood, a complete analysis will be repeated. In the case of well-learnt concepts only a short description will be given.

Selecting one of the main menu items, the student starts a new step of learning. At each step any teaching operation is chosen and then realized. During the realization process the student has access to the programming laboratory to experiment with examples or to solve tasks. While solving a task the student can also use the `Repeat' menu item and invoke the pedagogical module for help or to check out the solution prepared (items `Help' and `Check' on the main menu). After completing the step of learning the student returns to the programming laboratory and works independently until he invokes the pedagogical module the next time (see Figure 4).

Thus ITEM/IP allows the student to learn under the guidance of `an intelligent tutor', to repeat all the material learnt and to solve his `own' task. The student himself determines the time for learning and independent work, and the proportion between guided and `free' learning. Thus ITEM/IP supplies the student with an intelligent, friendly environment for work and study.

## SOME EXPERIENCE WITH THE SYSTEM

The version of ITEM/IP described in this paper was designed and implemented for the Soviet mainframe computer BESM-6. We have tested ITEM/IP for a year in the learning process among first-year students of the Moscow State University and 14-year-old students of Moscow schools. Our specific plan was not to make a `quantitative' classroom investigation of ITEM/IP; rather, we were planning to make qualitative comparison of the performance of students who used ITEM/IP, with the performance of students who were taught according to the traditional approach to introductory programming study.

Our experimental groups (three groups of about 15 students each) started their introductory programming course with Turingal (5-7 lessons with Turingal and rMM/IP). Traditional groups usually started directly with Pascal and used a computer environment similar to Turbo Pascal (first lesson with computer follows four introductory `blackboard' lessons).

It was easy to notice without any special measurements the increase of interest in the study of introductory programming in experimental groups. Another visible difference was a decrease in the number of `weak' students -from 4-5 in traditional groups to 2-3 in experimental ones.

One of the goals in ITEMIIP design was to decrease the amount of non-creative classroom work of a teacher in the course of introductory programming. We supposed that the designed environment could perform the non-creative part of the teacher's work: individualized choice of the best appropriate task for each student, evaluation of task solution, etc. A teacher was supposed to have more time for creative work with most weak and most strong students. for identification and remediation of bugs in student programs. Ideally, the latter kind of teacher activity could be performed by a PROUST-like (Soloway, 1985) intelligent program debugger, but it was too complicated to design such a power system.

The work with ITEM/IP has fulfilled our expectations. Moreover, we have found that a ITEM/IP visual interpreter could greatly help to

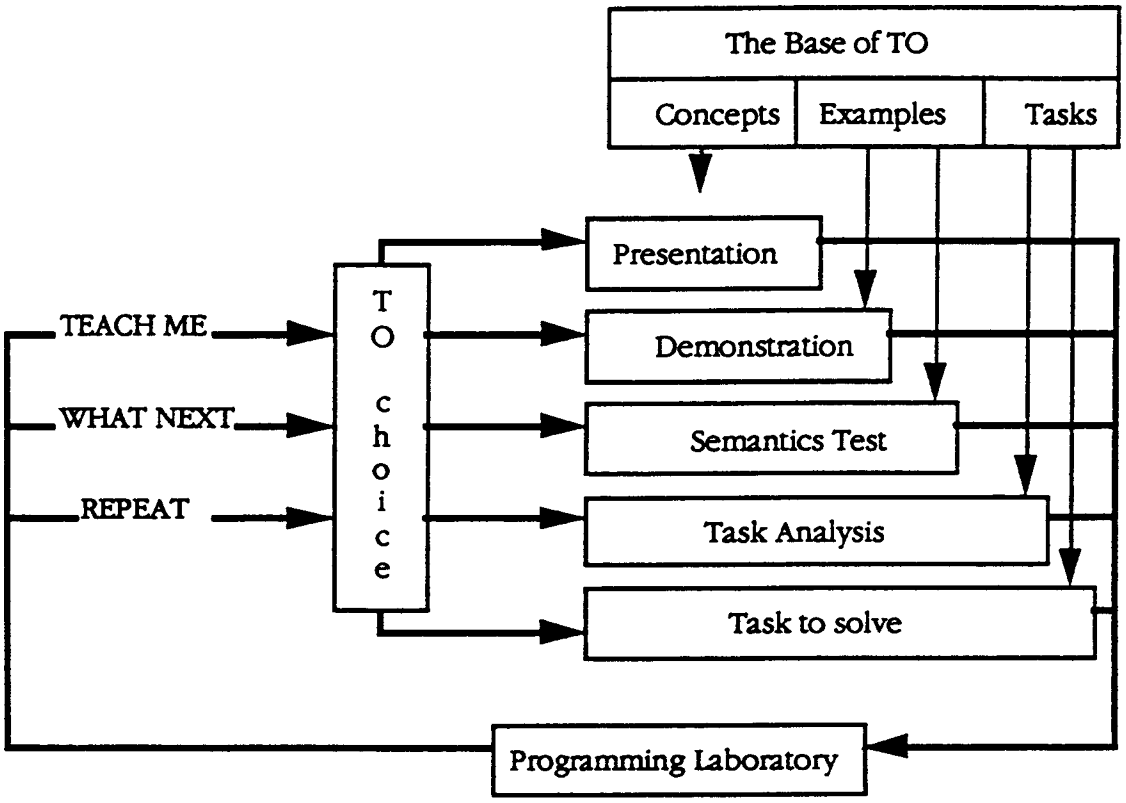


**Figure 4.** *The teaching operations (TO) in the learning process*

solve the problem of debugging student programs. In about four-fifths of all cases of identifying the bug it was quite enough for a student to run the wrong program visually with test data suggested by the system. Only in one-fifth of cases did the student (usually weak) need the teacher to help to understand the bug. Thus, the programming environment with the visualization of program execution could be regarded as the reasonable alternative to an intelligent program debugger (Eisenstadt, 1989).

## CONCLUSION

The approach described in this paper has enabled us to design the intelligent environment for introductory programming which integrates the capabilities of electronic manual, intelligent tutor and programming environment. ITEM/IP supports the major part of student needs and performs the non-creative part of the teacher's work in the classroom.

The approach described could be applied not only to the study of the Turingal-based introductory program but also to support the first steps of studying any programming language: Pascal, Lisp, Prolog. It should be noticed that the design of an ITEM-like environment for the `real' language is a difficult task. For example, we estimate the number of nodes in the Domain structural-network model for Pascal to be more than 200.

At present the second version of ITEM/1P, with enlarged capabilities for adaptation, is being developed for personal computers. At first, it will be possible for ITEM/1P II to update the student model while observing a student's activity in designing and debugging programs. Later the built-in strategy for directed learning will be replaced by expert knowledge of the human teacher, directly presented in the form of rules. We hope that such expert strategy will allow ITEM/IP II to adapt its work not only to the knowledge level but also to the individual features of a student.

## ACKNOWLEDGEMENT

I would like to thank Jim Greer, KSR Anjaneyulu and Luigi Sarti for their fruitful comments to an earlier version of the paper.

## REFERENCES

Anderson, J.R. and Reiser, B. (1985) The LISP tutor. *Byte, 10,* 4, 159-175.

Barr, A., Beard, M. and Atkinson, R.C. (1976) The computer as tutorial laboratory: the Stanford BIP project. ***International Journal on Man-Machine Studies,*** *8,* 5, 567-596.

Bonar, J. and Cunningham, R. (1988) Bridge: an intelligent tutor for thinking about programming. In Self, J. (ed.) ***Artificial intelligence and human learning. Intelligent computer-aided instruction.*** London: Chapman and Hall.

Brusilovsky, P.L. (1987) ***The analysis and design of the computer environments for studying programming.*** PhD thesis (in Russian). Department of Applied Mathematics and Cybernetics, Moscow State University, Moscow.

Brusilovsky, P.L. (1989) Information Kernel of Expert/Teaching systems. In ***Development and Application of Expert/Teaching Systems.*** (in Russian). Moscow: Institute for Higher Education.

Cheng, Y., Hu, Q. and Yang, J. (1988) An expert system for education: IPTS. In ***Proceedings of the 1988 IEEE International Conference on Systems, Man and Cybernetics.*** New York.

Eisenstadt, M. (1989) (AI in (Education in AI)): What does `domain visualisation' have to offer ITS? In Bierman, D., Breuker, J. and Sandberg, J. (eds) ***Artificial Intelligence and Education.*** Proceedings of the 4th International Conference on AI and Education. Amsterdam: IOS.

Hartley, J.R. (1973) The design and evaluation of an adaptive teaching system. ***International Journal on Man-Machine Studies, 5,*** 421-436.

Heines, J. and O'Shea, T. (1985) The design of a rule-based CAI tutorial. ***International Journal on Man-Machine Studies, 23, 1,*** 1-25.

Hudson, P.V. and Self, J.A. (1982) A dialogue system to teach database concepts. ***The Computer Journal, 25,*** *1,* 135-139.

Innocenti, C., Massucco, C., Persico, D. and Sarti, L. (1991) Ugo: An intelligent tutoring system for Prolog. In ***PEG'91- Proceedings of the Sixth International PEG Conference `Knowledge Based Environments for Teaching and Learning',*** Rapallo (Genova), Italy, 31 May-2 June, pp. 322-330.

McCalla, G.F. and Greer, J.E. (1987) The ***practical use of artificial intelligence in automated tutoring: current status and impediments to progress.*** Research report No. 87-12. Saskatoon: Department of Computational Science, University of Saskatchewan.

Papert, S. (1980) ***Mindstorms, Children, Computers and Powerful Ideas.*** New York: Basic Books.

Pattis, R.E. (1981) ***Karel - the Robot, a Gentle Introduction to the Art of Programming.*** London: Wiley.

Pratt, T.W. (1975) ***Programming Languages. Design and Implementation.*** Englewood Cliffs, NJ: Prentice Hall.

Reiser, B.J., Ranney, M., Lovett, M.C. and Kimberg, D.Y. (1989) Facilitating student's reasoning with causal explanations and visual representations. In Bierman, D., Breuker, J. and Sandberg, J. (eds) ***Artificial Intelligence and Education.*** Proceedings of the 4th International Conference on AI and Education. Amsterdam: IOS.

Soloway, E. (1985) PROUST. *Byte, 10,* 4, 179-190.

Tomek, I. (1982) Josef, the robot. ***Computers and Education, 6,*** 3, 287-293.

Wenger, E. (1987) ***Artificial Intelligence and Tutoring Systems. Computational Approaches to the Communication of Knowledge.*** *Los* Altos: Morgan Kaufmann.

## BIOGRAPHICAL NOTES

Dr Brusilovsky is a Senior Scientist at the ICSTI and is also an invited student supervisor and lecturer on Intelligent Tutoring Systems at Moscow State University.

Address for correspondence: Dr Peter Brusilovsky, International Centre for Scientific and Technical Information (ICSTI), Kuusinen str, 21b, Moscow 125252, USSR.