EESTI NSV TEADUSTE AKADEEMIA TOIMETISED. XVII KÕIDE
FÜÜSIKA * MATEMAATIKA. 1968, NR. 3

ИЗВЕСТИЯ АКАДЕМИИ НАУК ЭСТОНСКОЙ ССР. ТОМ XVII
ФИЗИКА * МАТЕМАТИКА. 1968, № 3

https://doi.org/10.3176/phys.math.1968.3.02

B. TAMM

ON SOME ASPECTS IN SIMULATING ENGINEERING PROCESSES BY MEANS OF PROBLEM ORIENTED PROGRAMMING SYSTEMS

The main difference between the general-purpose electronic computers and the immense amount of machines created by human being is the remarkable universality of the computers. They are able to solve complex problems belonging to the most different fields of man's activities. The computing speeds of these machines as well as the speeds of switching from one problem to another exceed the corresponding abilities of man by more than six orders of magnitude. Like most biological systems, computers can be taught to carry out most different operations. Everything depends on the "preliminary knowledge" we supply the machines with. This preliminary knowledge of the computers is called computer software.

During the last ten years in our country a certain amount of investigations were started for carrying out mainly all-purpose programming systems, i. e. programming languages and corresponding translators. These languages serve efficiently for describing evaluating procedures of numerical analyses, linear and nonlinear programming, etc. In this case the procedure oriented programming systems are irreplaceable in man-machine systems. Several systems of such kind, based on well-known international programming language ALGOL have been worked out in the Soviet Union [1, 2, 3].

But besides the problems of pure arithmetical or mathematical character there is, however, a great amount of tasks the complexity of which is determined by some special kind of decision making rather than by mathematical algorithms. Most problems of civil engineering belong to this ensemble.

Is it possible to program for a computer, for instance, the following problems by means of procedure oriented programming systems? Designing of a multi-level traffic junction, controlling the cutter path of a numerically controlled machine tool, optimal architectural planning for complicated buildings of special purposes. The answer is "Yes, it is, but only in principle." In practice it turns out as an extremely cumbersome and time-consuming task because of the unfittability of these languages for describing all kinds of situations, logical conditions and engineering objects. At the same time, these input languages are oriented on the professional programmer who is able, after some study, to deal with the rather complicated rules of their semantics and syntax. They don't help an engineer unskilled in both computing technique and the art of programming. That is the reason why procedure oriented programming languages are almost useless in the cases mentioned above.

When solving an engineering problem it is necessary to define and structure the problem, formulate the method of solution, perform calculations, obtain, evaluate and implement the results. An engineer must continually make decisions throughout the problem solution concerning technical aspects as well as economic and time limitations. Two engineers given the same problem might obtain completely different yet satisfactory problem solutions. In other cases, two engineers may obtain the same solution of a problem, but by totally different methods.

Most of the engineering problems differ from each other and therefore the writing up of a new program for each new problem is practically impossible. Two different approaches were followed to find a way out of this situation. The first approach could be termed as the decomposition and reassembly approach. In this case an engineering problem was decomposed for programming purposes, and reassembled for problem solution. Rather than program entire problem solutions, small fundamental mathematical and engineering calculations were programmed that could be solved with the help of subroutine library programs. A complete problem solution could be achieved then by running a series of these basic programs. However, this method did not work. The programs developed, did not properly interface each other, there was too much card or tape handling involved in the computer solution. The system never got properly integrated.

The followers of the second approach tried to develop large all-purpose programs. But that did not succeed either. The input forms of these all-purpose systems were rather complicated and the users refused to apply them since in spite of the advertized universality there were many problems the programs could not handle. Taking still into consideration that the computer programs are quite difficult to modify, it turned out that if an engineer could not find a program to fit his problem, he modified the problem to fit the program rather than vice versa.

It should be mentioned that the final solution of the entire problem is far from perfection even today. At the same time, however, the basic principles began to clear out, and during the last years a certain number of successful implementations could have been followed in several countries including the Soviet Union, the USA, Norway, etc.

As it was mentioned, the question lies in teaching the computer to solve a great variety of different engineering problems. Therefore the engineer must have an opportunity to communicate with the computer in much the same way he communicates with another engineer using his normal engineering terminology. He must have a language at his disposal which enables him to build up language models of the problems to be solved. The writing out and debugging of these models must be a matter of minutes or hours instead of weeks or months, which often happens when using unfitted computer software. On the other hand, the computer has to be supplied with a processor for translating the language models into internal intermediate languages to perform the necessary information processing. Later on the processor synthesizes the algorithm of the particular problem and, by using the subroutines of its procedures in proper sequence, solves the problem and produces the results obtained.

It turns out that problem oriented programming systems can serve as very handy tools for engineers to communicate with a computer when solving their problems. These systems consist of programming languages oriented towards the engineer and the corresponding processors. A general block diagram describing the main steps towards solving a wide set of engineering problems is presented in Fig. 1.

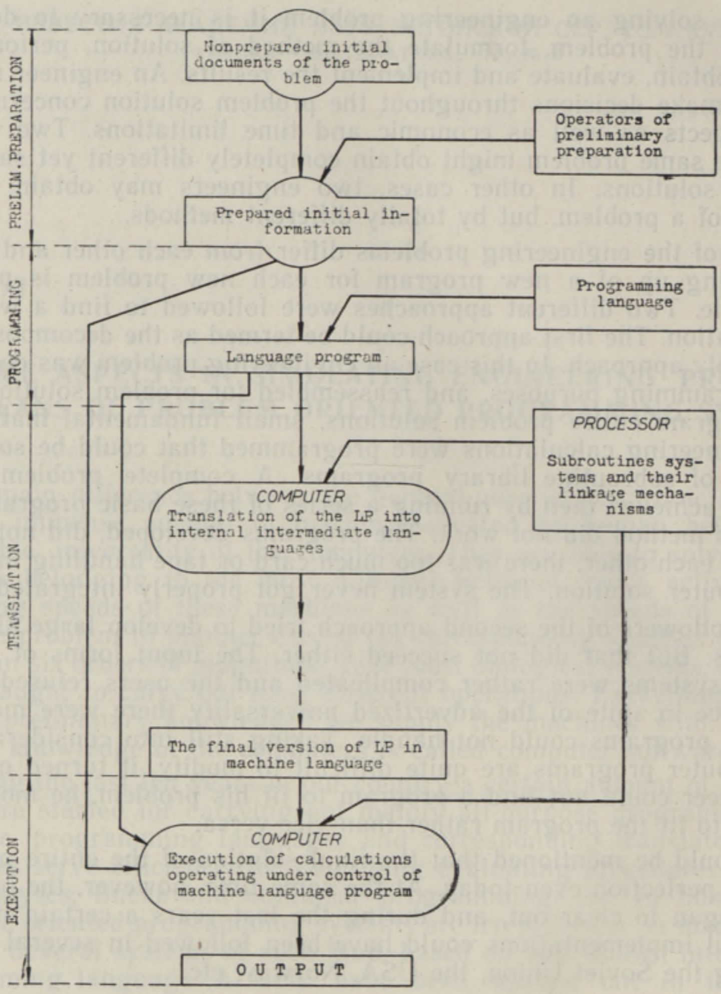
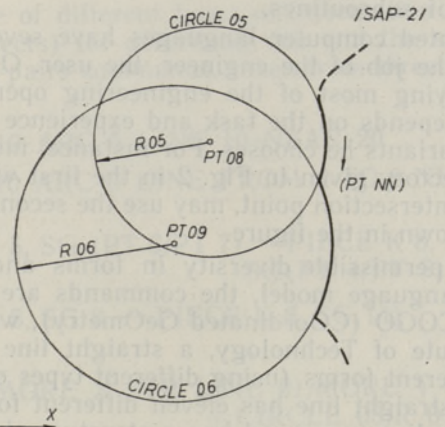

PRELIM. PREPARATION
PROGRAMMING
TRANSLATION
EXECUTION
Nonprepared initial documents of the problem
Operators of preliminary preparation
Prepared initial information
Programming language
Language program
PROCESSOR
Subroutines systems and their linkage mechanisms
COMPUTER
Translation of the LP into internal intermediate languages
The final version of LP in machine language
COMPUTER
Execution of calculations operating under control of machine language program
OUTPUT

Fig. 1.

Before an engineer starts to solve his problem, he has a certain amount of initial information upon the problem at his disposal. This information, usually unprepared for automatic processing, can be divided into two different sets. One of them comprises the data given as graphic documents (drawings, blueprints, tables, lists, textual notes, etc.), and the other set — information in the memory of a computer (coded, as a rule, in an unsuitable way for automatic processing). The initial information, naturally, may be presented in both ways simultaneously. Depending on these ways, the preparation of initial data for automatic processing is carried out by the use of two different sets of operators. In the case of graphical documents, the set consists of operations such as, for instance, checking of the geometry of the drawing, designating its elements, sorting out the tables and so on. These operators represent some exact prescriptions which are in full accord with the programming language used further on. In case of initial information stored in the memory of the computer, we have a certain number of subroutines which turn this information into some code suitable for compiling [4].

Now the programming language comes in. Designing the language model of the problem to be solved and writing out the language program (which is the written copy of the language model) is a rather important stage, as it involves the formulating of the problem to be solved. There may be a great variety of language programs depending on the character of the problems. Each language program is an algorithm written out in the programming language, where the problem to be solved is formulated unambiguously.

In the computer the language program is translated into machine code through some intermediate internal languages. During the translation process, a certain data processing is carried out to transform the information (textual as well as numeric) into some fixed canonic forms. Thus the language program controls the execution of calculations and together with the processor it makes the large-scale computer act as a special purpose computer for solving a particular problem. The instructions, written in the language program make the processor appeal to corresponding procedures. These, in turn, use their subroutines to synthesize the corresponding algorithms to solve the problem described by the engineer. The last software block, the postprocessor transforms the data into some desired output form.



. . ., ALONG—CIRCLE 05, MORE X POINT, ALONG—CIRCLE 06, . . .

. . ., ALONG—CIRCLE 05, TO POINT NN, ALONG—CIRCLE 06, . . .

Fig. 2.

The actual success of a problem oriented programming system depends highly on its language because the language is the tool for simulating the class of problems the system is designed for. The language must be oriented towards the engineer, i. e. he must have the chance to use professional terminology and communicate with the computer in much the same way as he communicates with another engineer of his profession. An engineer describes his problem by specifying the engineering operations that must be performed, and he does not refer directly to the procedures associated with the operation. Thus, when specifying the fragment of trajectory (shown in Fig. 2) of an object moving along the circular arcs 5 and 6, passing over from one arc to another at the intersection point having greater abscissa (with regard to the other possible intersection point), the engineer is not asked to think about what kind of equations there must be solved in order to calculate the coordinates of the intersection point, from where to pick up the numerical data for solving equations, how to fix the correct route, etc. (SAP-2) * [5]. He simply writes some macrostatements, two alternative variants of which (concerning the fragment of the trajectory) are given in the figure above.

Problem oriented languages are command structured, where each command represents an operation or group of operations the computer is to perform using the subroutines of the system processor. The commands are usually technical terms which have a meaning to an engineer, such as ALONG THE CIRCULAR ARC or BEGIN FROM SEGMENT 14. Each command may also contain data relevant to the requested operation.

The following command (APROKS)** [4, 6].

TO PIECE K 10, POINT (SEG 3/SEG 4) [PEN : ON]

means that the cutter of the numerically controlled flame cutter is to approach piece No. 10 in the nesting plate and actually reach the point of the piece specified as an intersection between segments No. 3 and 4. At this point the cutter must penetrate the metal.

As it follows even from these brief examples, the commands of problem oriented languages are really engineering macros, each of which calls for some group of processor procedures. These procedures, in turn, utilize a certain number of subroutines.

Problem oriented computer languages have several remarkable qualities facilitating the job of the engineer, the user. One of them is the possibility of specifying most of the engineering operations in several different ways. It depends on the task and experience of the engineer which of the possible variants he chooses. For instance, instead of describing the fragment of trajectory given in Fig. 2 in the first way, the engineer, after designating the intersection point, may use the second alternative for specifying the part shown in the figure.

Owing to the permissible diversity in forms and types of presenting information in language model, the commands are tree-structured. Thus, for instance, in COGO (COordinated GeOmetry), worked out by the Massachusetts Institute of Technology, a straight line may be described by means of six different forms (using different types of geometrical objects) [7]. In SAP-II a straight line has eleven different forms of representation, and the corresponding command has a structure shown in Fig. 3. In this tree each branch stands for one of the possible forms of describing a straight line, whereas each link of the branch depicts the element of information. The same figure may serve for showing one more order of liberty which is at the disposal of the user of the language. When writing the commands of language program, in many cases he is not asked to write

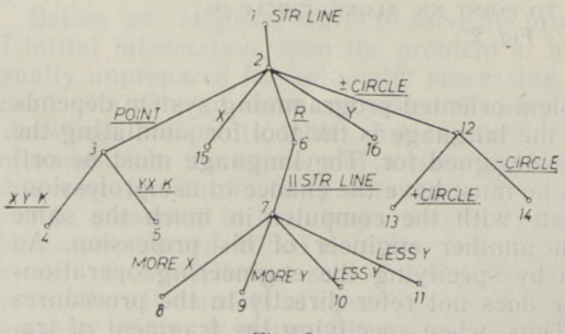


Fig. 3.

* System of Automatic Programming for numerically controlled machine tools. Institute of Cybernetics, 1964.

** Automatic programming for machining of ship hulls. Institute of Cybernetics, 1967.

the elements of commands in any strict sequence — a demand that must always be taken into consideration when filling out standard formats, tables or technological cards for programming. So, all the underlined elements in Fig. 3 may change their locations in the form in which they are used.

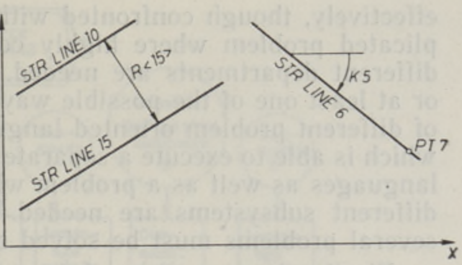


Fig. 4.

Examples: (see Figs 3 and 4)

```
    STR LINE 10/ = /MORE Y STR LINE 15/ R 15/
or
    STR LINE 10/ = /R 15/MORE Y  STR LINE 15/

    STR LINE  6/ = /PT 7/ YX K 5/
or
    STR LINE  6/ = YX  K 5/ PT 7/
```

The possible usage of different types of information (in our case different geometrical objects) for describing one and the same idea may be illustrated by equal-in-pairs commands, taken from three different problem oriented systems:

```
  INTERSECTION 10 ARC 5 LINE 10 NEAR 20                  } COGO
  INTERSECTION 10 ARC 5 LINE FROM 2 TO 8 NEAR 20         }

COMMON (PIECE K 5, SG (PT 6/PT 7) → PIECE K 8,           }
                                   SG(PT 4/PT 5))        } APROKS
COMMON (PIECE K 5, SG 8  → PIECE K 8, SG 14)             }

   CIRCLE 01/ = /MORE X STR LINE 01/MORE Y +             }
                              + CIRCLE 03/R 01/          } SAP-II
   CIRCLE 01/ = /MORE Y STR LINE 01/LESS X  +            }
                              + CIRCLE 03/R 01/          }
```

In the first pair of commands, the same intersection of geometrical elements is specified by two different forms; in the second pair, two identical versions for one and the same common cut operator are written; the third pair consists of two alternatives for specifying one and the same circle. These brief examples, of course, do not throw light on the entire flexibility of the languages mentioned.

Due to the properties mentioned and similar to them, the engineer is not subordinated to the language. On the contrary, he has a lot of freedom in specifying the language model of his problem, so that the result of it really depends on his creativeness, on his engineering qualification.

Let us assume now that we have a more or less sufficient number of engineering programming systems at our disposal. Are we allowed to hope that at this stage the problem of simulating engineering processes will be principally solved? Certainly not. At this stage we can imagine that we should be in a position of a great engineering organization divided into departments, each of which is able to solve its own problems rather effectively, though confronted with great difficulties when solving a complicated problem where highly coordinated and simultaneous actions of different departments are needed. For these purposes the most essential, or at least one of the possible ways, is to create a large integrated system of different problem oriented languages having a large common processor which is able to execute a separate problem concerned with one of the input languages as well as a problem where numerous interconnections between different subsystems are needed. But in order to accomplish this idea, several problems must be solved beforehand.

Firstly, the processors of the corresponding problem oriented languages existing in our country up to this day are written in machine codes. That is a tremendous and time-consuming task, and the carrying out of these processors in the future by the same way is almost unreal, particulary taking into consideration the diminishing operating times of any particular type of computer.

Secondly, there are no large-scale procedure oriented languages and translators as yet, allowing to satisfy all the necessary conditions for describing the processor of such an integrated system (adapted for our computers). Unfortunately the existing ALGOL subsets could not be used for these purposes because of their poor capabilities in describing logical situations and in data processing.

And, of course, the problem of appropriate and reliable periphery equipment for computers isn't trivial either.

The internal structure of such a system might be assumed to be similar to the internal structure of a civil engineering organization. We can, in general, think of three levels of hierarchy. In contrast to the structure of such kind of integrated system and a civil engineering organization, the similarities are observed between the system and the entire organization (at a higher level of hierarchy), the subsystems and the divisions of organization (the second level), and the subsystem procedures and the groups of the division (the lower level). The principle block diagram of the system is presented in Fig. 5.

First of all this system must have a powerful and efficient engineering programming language for writing the subsystem programs. This language should be addressed to highly skilled system programmers to create and program new subsystems or to enlarge and alter the existing ones. In other words, one of its functions is to enable the programming of the system processor. Let us call it System Language. This language, certainly, must be a procedure oriented language as it must provide for a handy programming of a large set of procedures allocated in the system processor. Probably some extension of some existing problem oriented language will serve as System Language. That is why a system precompiler (for translating the system statements concerning the extension of a procedure oriented language into its legitimate statements) is needed. It is obvious that all the problem oriented subsystems comprised in the integrated system must have their individual precompilers in order to translate the language models of problems written in the corresponding source language into statements of the system compiler language.

The system compiler language is responsible for accomplishing three main functions:

1) It must recompile the System Language statements mainly concerned with altering and expanding already existing subsystems, or adding new ones, into machine code or some other lower level language in which the processor procedures are stored.

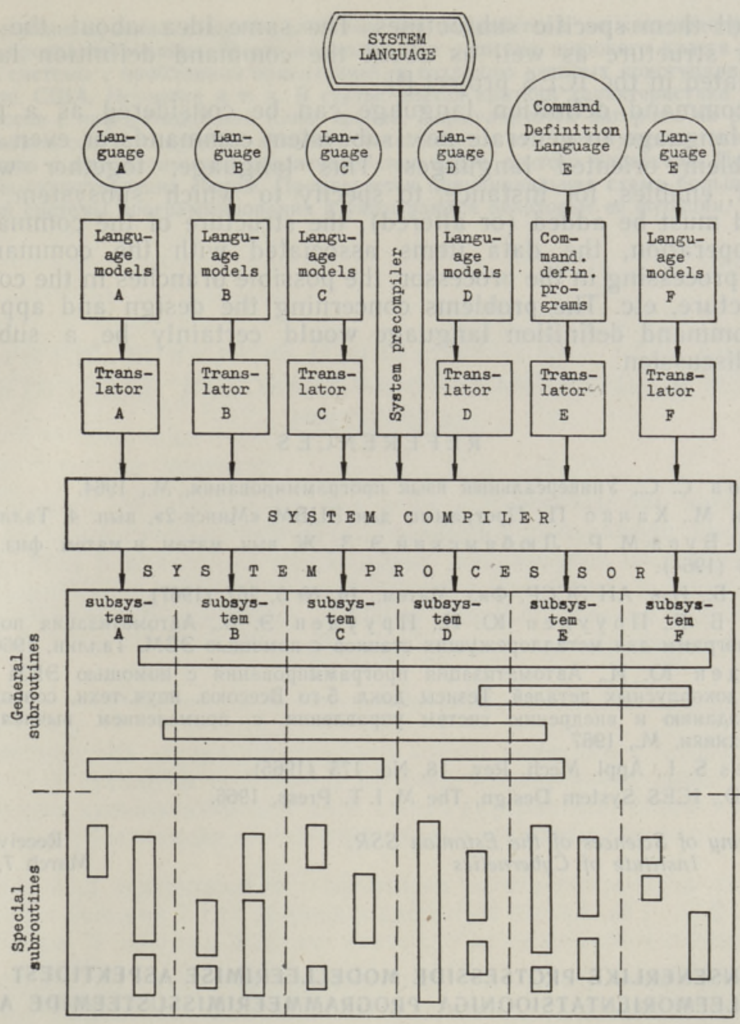


Fig. 5.

2) At the same time, the system compiler has to retranslate the language program statements of different problem oriented subsystems into the code or set of codes in which the ensemble of processor subroutines will act.

3) The third is the control function. If it is necessary to use several different subsystems for solving a complex task, the proper linkage of these subsystems as well as the mutual exchange of intermediate information must be controlled at the level of the system compiler language.

The executive part of system processor when solving a problem is the part where the programs of the procedures are situated. This part has some kind of matrix structure where the columns represent different subsystems and the rows represent different classes of subroutines. Apparently there are at least two classes of subroutines: one of them comprises the program used by several subsystems — let us refer to them as general subroutines, the other comprises programs used by only one subsystem —

let us call them specific subroutines. The same idea about the system processor structure as well as about the command definition language is formulated in the ICES project [8].

The command definition language can be considered as a problem oriented language to generate new subsystem commands or even entirely new problem oriented languages. This language, together with its processor, enables, for instance, to specify to which subsystem a new command must be added (or altered), the structure of the command, the type of operation, the data items associated with the command, the required processing in the processor, the possible branches in the command tree structure, etc. The problems concerning the design and application of the command definition language would certainly be a subject of special discussion.

REFERENCES

1. Лавров С. С., Универсальный язык программирования, М., 1964.
2. Котли М., Ханко П., Программы для ЭЦВМ «Минск-2», вып. 4, Таллин, 1966.
3. Шура-Бура М. Р., Любимский Э. З., Ж. выч. матем. и матем. физ., **4**, № 1, 96 (1964).
4. Тамм Б., Изв. АН ЭССР, Физ. Матем., **16**, № 3, 267 (1967).
5. Тамм Б. Г., Пруудeн Ю. И., Пруудeн Э. В., Автоматизация подготовки программ для металлорежущих станков с помощью ЭВМ, Таллин, 1966.
6. Пруудeн Ю. И., Автоматизация программирования с помощью ЭВМ вырезки судокорпусных деталей. Тезисы докл. 5-го Всесоюз. науч.-техн. совещания по созданию и внедрению систем управления с применением вычислительной техники, М., 1967.
7. Fenves S. I., Appl. Mech. Rev., **18**, No. 175 (1965).
8. Roos D., ICES System Design, The M. I. T. Press, 1966.

Academy of Sciences of the Estonian SSR,
Institute of Cybernetics

Received
March 7, 1968

B. TAMM

INSENERLIKE PROTSESSIDE MODELLEERIMISE ASPEKTIDEST PROBLEEMORIENTATSIOONIGA PROGRAMMEERIMISSÜSTEEMIDE ABIL

Laialt levinud protseduurorientatsiooniga programmeerimissüsteemid elektronarvutitele osutuvad vähe efektiivseks, kui on tegemist loogiliste situatsioonide ja insenerlike objektide kirjeldamisega. Lisaks sellele on nende keel ning semantilised ja süntaktilised reeglid liiga keerukad programmeerimistehnikas kvalifitseerimata kasutajale. Laia klassi insenerlike ülesannete lahendamiseks on väga sobivad probleemorientatsiooniga programmeerimissüsteemid, mida viimasel ajal on hakatud välja töötama NSV Liidus, Ameerika Ühendriikides, Norras ja mujal. Artiklis antakse lühike insenerlike protsesside iseloomustus informatsioonilisest seisukohast, formuleeritakse nende modelleerimise põhimõtteline algoritm, kasutades vastavaid keeli, ning näidatakse nende keelte mitmeid põhimõttelisi eeliseid, võrreldes arvutite muud laadi matemaatilise varustusega. Tuuakse näiteid olemasolevatest süsteemidest. Esitatakse suure integreeritud programmeerimissüsteemi põhimõtteline skeem ja selgitatakse selle tähtsamaid funktsioone.

Б. ТАММ

НЕКОТОРЫЕ АСПЕКТЫ МОДЕЛИРОВАНИЯ ИНЖЕНЕРНЫХ ПРОЦЕССОВ С ПОМОЩЬЮ СИСТЕМ ПРОГРАММИРОВАНИЯ С ПРОБЛЕМНОЙ ОРИЕНТАЦИЕЙ

Общеизвестные системы программирования с процедурной ориентацией для цифровых вычислительных машин оказываются неэффективными при необходимости описания логических ситуаций и инженерных объектов. Кроме того, их язык, семантические и синтаксические правила слишком сложные для потребителя, не квалифицированного в технике программирования. Очень подходят для решения широкого класса инженерных задач системы с проблемной ориентацией, к созданию которых приступили в Советском Союзе, США, Норвегии и т. д. В статье дается краткая характеристика инженерных процессов с информационной точки зрения, формулируется алгоритм их решения с использованием соответствующих языков и показывается ряд преимуществ таких языков по сравнению с математической обеспеченностью ЦВМ другого характера. Приводятся примеры из существующих систем. Предлагается принципиальная схема большой интегрированной системы программирования и указываются некоторые ее функции.