**SECTION 3. DRAKON - VISUAL ALGORITHMIC LANGUAGE**

As the experience of teaching informatics shows, mastering the art of programming and algorithmic thinking encounters known difficulties. They are largely due to a lack of skills in formulating the problem and developing an algorithm to solve it. Even at the beginning of the mass training, programming in schools and universities was noticed that programmers can be divided into two groups: "field officer" and "staff officer". A field officer, as a rule, possessing a certain amount of programming knowledge, immediately rushes to write program code. No comment or explanation. Naturally, in the process of programming, there are unforeseen situations that lead to constant changes in the text of the program. In the end, and in many cases, the "field officer" surrenders or begins to consider the usefulness of being a "staff officer". In turn, the "staff officer" first ponders the algorithm of the program. And here to help him various methods of visualization of algorithms.

**3.1. Algorithm representation visualization**

In theory and practice the following forms of representation of algorithms have been formed:

* Words (written in natural language);
* Visual (images from graphic characters);
* Pseudocodes (semiformalized descriptions of algorithms in conditional algorithmic language);
* Software (programming language texts).

Visual representations of algorithms were the most effective and common in terms of visualization and understanding. Conceptually, visualization is a form of presentation of information, data, knowledge in the form of images, in order to achieve maximum convenience of perception, understanding and analysis. The visual representation allows a better view and understanding of the structural elements used in the design of the algorithm, including the logic of their interaction. The most common means of visualizing algorithmic constructions are flowcharts and DRAKON-diagrams. The advantage of flowcharting is the availability of a corresponding State standard. However, flowcharts have recently been criticized: they claim that they are unsuitable for structural programming, difficult to formalize, they cannot be used to generate the programming code.

An alternative to flowchart imaging is DRAKON-based algorithm engineering, which results in DRAKON- diagrams []. DRAKON-diagrams are suitable for formalized recording, automatic code retrieval and execution on the computer. However, a more important aspect of DRAKON technology is its cognitive difference from flowchart technology. While flowcharts do improve programm clarity, this is not always the case, and improvements are limited. In addition, there are many cases when unsuccessful flowcharts confuse the case and make understanding difficult. In contrast, DRAKON-diagrams are much better understood (Figure 3.1.).

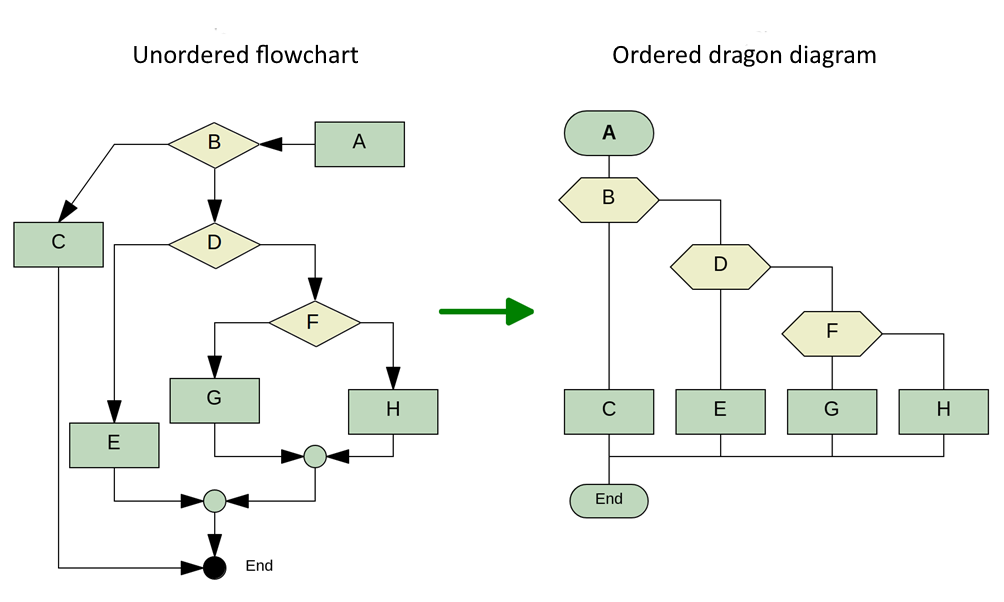


Figure 3.1. Comparison of a visual representation of an algorithm in the form of a flowchart and a DRAKON-diagram

Through the use of special formal and informal cognitive techniques, the DRAKON-diagram visually presents the solution to any algorithm or technology problem that is as complex as possible in an extremely clear, visual and understandable way. Special importance of visualization of algorithms with the help of DRAKON-diagrams acquires in the educational process, both in the study of computer science and in classes of natural as well as humanitarian disciplines.

**3.2. Algorithmic language DRAKON Editor**

Currently, there are several DRAKON-diagram platforms (DRAKON WEB Editor, DRAKONHUB, DRAKON.tech, "IS DRAKON". Software code generation in one of the known programming languages, including Golang, is realized with the help of the freely distributed editor DRAKON Editor [Mitkin]. The current version of the editor is DRAKON-Editor 1.31, which can be downloaded from http://DRAKON-editor.sourceforge.sounet/ (Figure 3.2.).



Figure 3.2. Archived editor file

To run the DRAKON Editor 1.31. on the Linux and Mac operating system, you must:

- extract files from DRAKON editor 1.31.zip archive;

- start the editor: /DRAKON\_editor.tcl from the terminal (inside the unassembled folder.

The Windows operating system requires:

- extract files from DRAKON Editor 1.31.zip archive;

- open the executable file of the editor by double clicking on the fragment DRAKON\_tcl (Figure 3.3.): .



Figure 3.3. Launch editor DRAKON Editor 1.31.

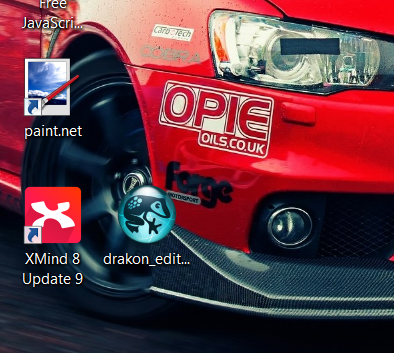
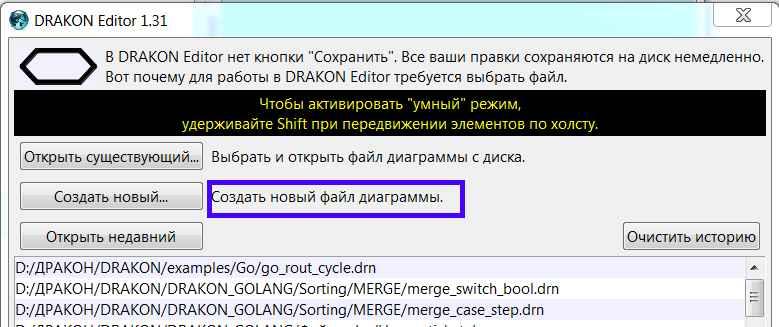
 After activating the editor (DRAKON\_Editor), the option "Create a new diagram file" is selected, and then a window opens to save the diagram file. As soon as you create a new file, you must save it, because in the DRAKON Editor, you do not need to save all revisions - files are saved automatically.

Figure 3.4. Create a new diagram file

After creating a new file, the editor interface window opens (Figure3.5.):

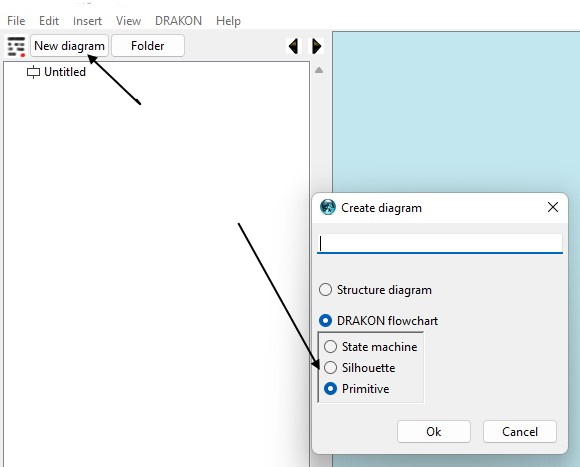


Figure 3.5. Editor interface window DRAKON WEB Editor

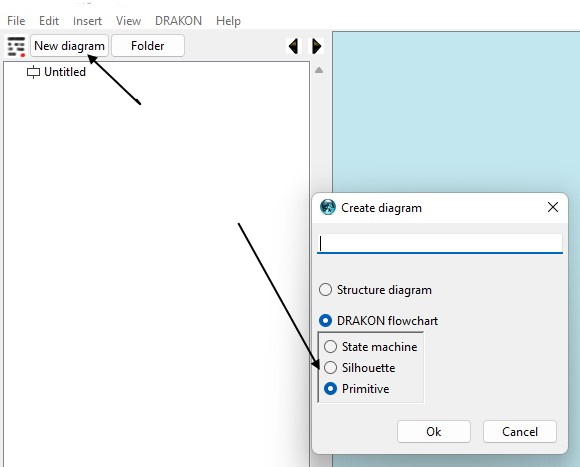
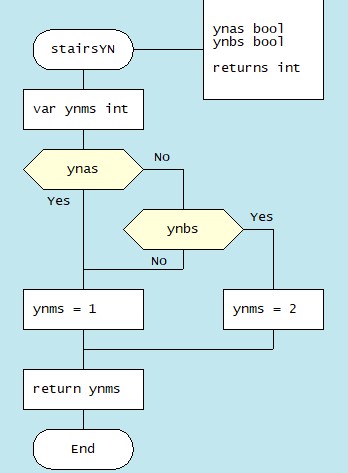
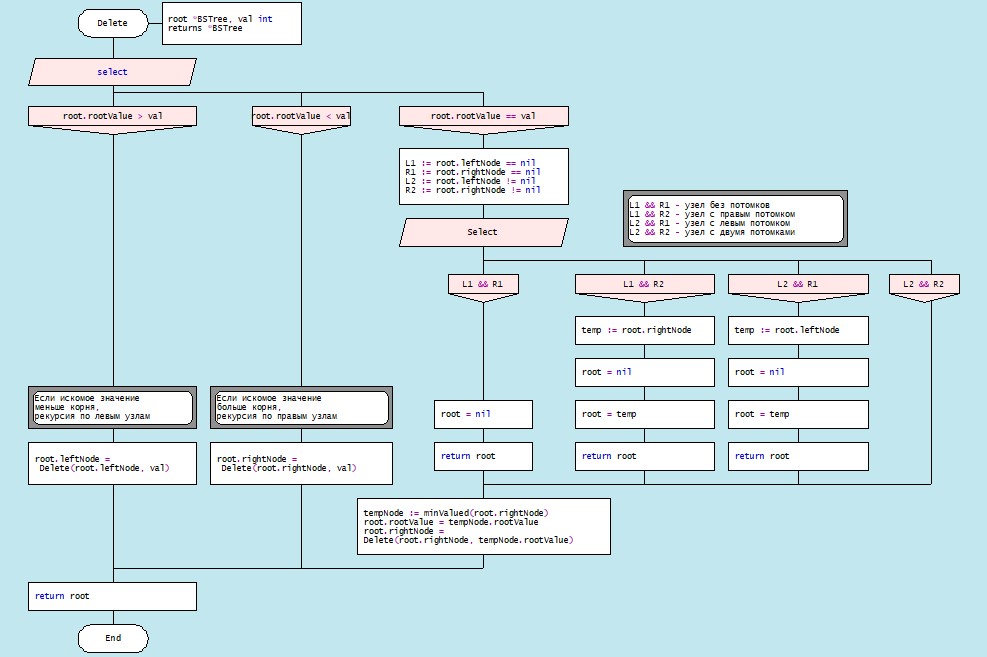
Next you need to click the icon «New diagram» and in the newly opened menu choose the view of the diagram (Primitive /Primitive/ or Silhouette /) (Figure3.6.). 

Figure 3.6. Creation a new diagram

The choice of diagram type is determined by the complexity of the algorithm (Figure3.7.):



a) Primitive



b) Silhouette

Figure 3.7. Types of diagrams: a) "Primitive"; b) "Silhouette"

To create a DRAKON-diagram, the user calls the context menu (right mouse button), and according to the algorithm chooses the necessary icons, filling them with the corresponding operators of the selected programming language (Figure 3.8).

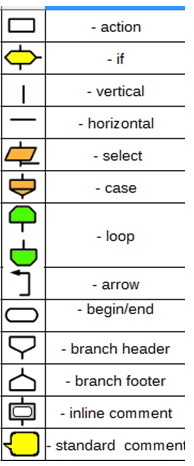


Figure 3.8. Set of DRAKON Web Editor icons

The selected icon is moved by the mouse to the vertical line (skewer - in terms of DRAKON-technology) in the right place. The icon space is then filled in with the corresponding text that displays a Snippet in a specific programming language. Creating a DRAKON-diagram must follow certain rules:

1. Creating a DRAKON-diagram begins with a name that should reflect the purpose of the (function) algorithm and be located at the top. 2. The diagram should have only one beginning and one end. The "end" graph is placed at the bottom of the diagram (Figure 3.9):

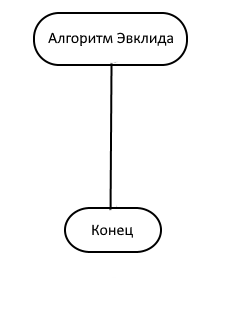


Figure 3.9. Diagram start

3. The action flow represented in the diagram should only go from top to bottom. This approach is more convenient, because in our cultural area texts are read in this way.

4. Avoid turning. The only case where lines have to change direction is where decisions are made. Turns are needed only when the algorithm requires making a choice between different actions. If there are no solutions, you need to go down. In any case, it is necessary to minimize the number of turns.

5. Crossings of lines are absolutely not allowed. All attempts to apply crossings must be prevented. However, in case of an intersection, the editor will give an error.

6. A top-down action prevents the use of arrows. The only exception is a loop of type while (Figure3.10.):

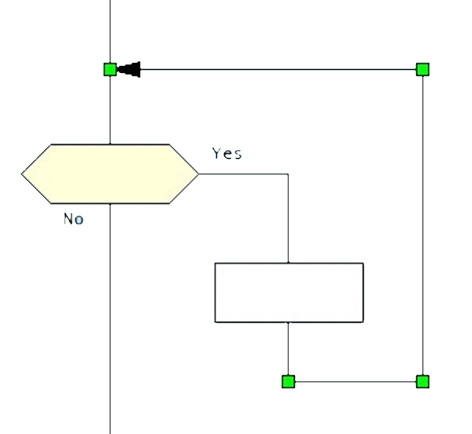


Figure 3.10. Data flow from bottom to top (While loop type)

7. When creating a DRAKON-diagram, only straight vertical and horizontal lines should be used, as straight lines are easier to understand than curves.

8. It is necessary to keep the same distance between adjacent icons. The DRAKON WEB Editor allows you to "cleverly" optimize the diagram using the "Shift + mouse" keys.

9. Branching is done only to the right. Branching to the left should be excluded. Following this rule significantly increases the predictability and uniformity of charts.

The reader may get the impression that creating DRAKON-diagrams is a very complex process. It should be recalled once again that in the editor DRAKON WEB Editor everything is quite simple and clear. The editor will not allow the violation of the stated rules, which is checked by the verify option (the Figure shows an error - there is no space between the icons (Figure3.11)):

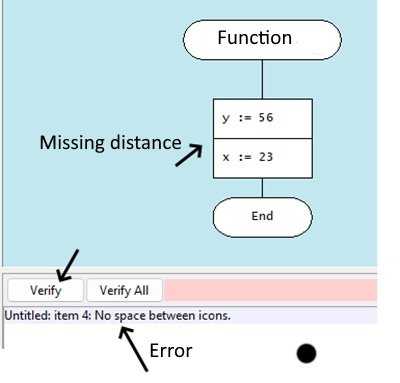


Figure 3.11. Work of the editor, indicating the error

The DRAKON WEB Editor implements a complete set of rules of visual syntax, which frees the user from the need to remember the syntax rules in detail. DRAKON WEB Editor creates only correctly created diagrams and does not allow diagrams in which the syntax is broken to generate program code.

**3.3. Basic structures of DRAKON WEB Editor algorithms**

The logical structure of an algorithm can be represented by a combination of three basic structures: linear, branched and cyclic. The typical feature of the basic structures is the presence of one entrance and one wig. 3.3.1. Linear structure

The linear structure of an algorithmic process implements operations that are performed sequentially in order of writing. A typical example of such a process is a standard three-step computing scheme:

a) Input data; b) formula calculation; c) output result. The graphical representation of the basic element of the linear structure in the software products of the DRAKON line has the form of simple rectangles, For example, in the algorithm for finding the minimum and maximum values of the array, minim and maxim variables are given the value of the zero element of the array (Figure 3.12).

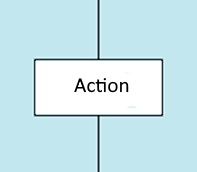
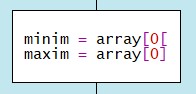
 

Figure 3.12. Linear Structure of DRAKON-diagram

3.3.2. Basic structures of branched structure

The branched structure contains at least one condition check, which ensures the transition to one of the possible solutions. Each option leads to a common output, that is, the algorithm will continue regardless of which path is chosen. The branching structure exists in two main options:

a). The DRAKON Snippet of the construction "if (condition), then (action) otherwise (action)", that is, in programming languages is the statement if...else. An example of an if-else construction in a DRAKON-diagram in an algorithm for finding minimum and maximum array values (Figure 3.13):

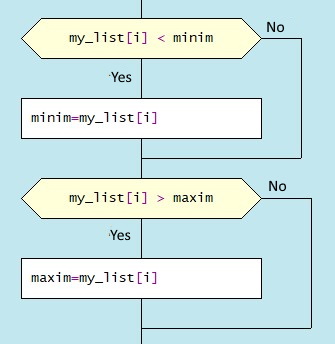


Figure 3.13. Diagram snippet with “if-else”

b). DRAKON Snippet Construction «Select»

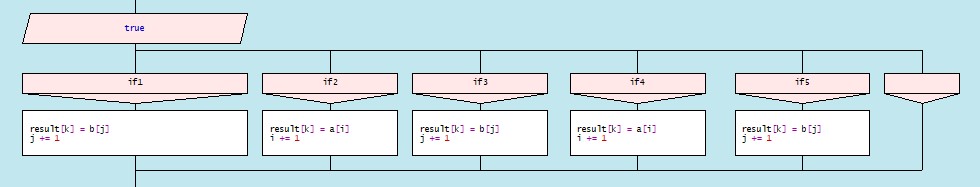
The values to which the expression in the "Choose" graph will be compared are covered in the "Option" graphs. If there is no text in the extreme right, it means "all other values". Below is an example of how to branch a sorting algorithm (merge sort) (3.14). 

Figure 3.14 Branch selection snippet (Select statement) 3.3.2. Base loop constructions

The loop structure involves repeating the same sequence of actions repeatedly. The number of repetitions is determined by the input data or task conditions. Loop structures include, first of all, the construction "Loop-For" (C-style loop, "Loop for each", composite constructions "Loop-With\_Arrow" ("Do-Chek loop") and "Do-Chek-Do").

a). "Loop-For" consists of three parts. In the first part, the loop initialization is fixed. In the second one, the loop completion condition is checked. If true, the body operators of the loop are executed until the expression becomes false. If it is false, the loop ends and the control is passed to the next operator. In the third part, the loop parameter increases. The snippet of the DRAKON-diagram with the "Loop-For" design

has the appearance (Figure 3.15):

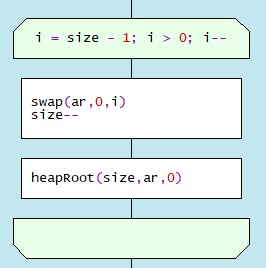


Figure 3.15. DRAKON construction "Loop-For”

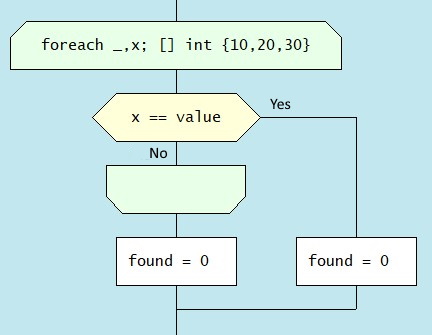
b). Loop "foreach" executes the operator or block of operators for each array element or data list (Figure 3.16). 

Figure 3.16. DRAKON construction "Loop for each”

c). Example of composite construction "Loop With Arrow" (Figure 3.17)

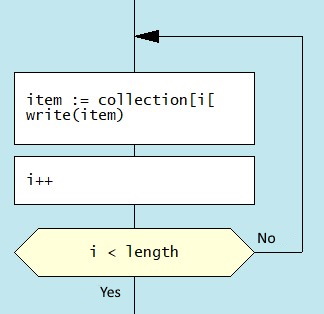


Figure. 3.17. DRAKON construction "Do-Chek loop ”

d). Example of composite construction “Do-Chek-Do” (Figure 3.18)

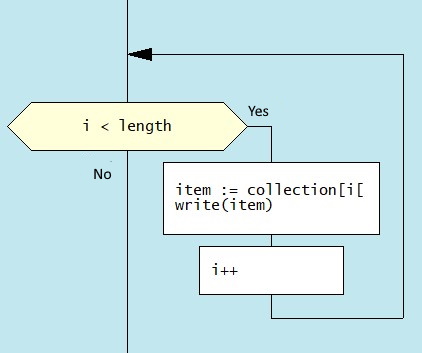


Figure 3.18 Diagram snippet “Do-Chek-Do”

3.3.3. DRAKON-diagram example

Let us show with a concrete example the construction of the diagram of the algorithm for calculating the most common divisor by the Euclid method. The largest common divisor (GCD) is a number that divides without remainder two numbers and divides itself without remainder by any other divisor of the given two numbers. Simply put, this is the largest number by which you can divide without residue the two numbers for which you are looking for GCD.

The algorithm for finding GCD by Euclid division is as follows:1. The larger number is divided into the smaller number.2. If divided without remainder, then the smaller number and is NLD (should leave the cycle).3. If there is a remainder, the larger number is replaced by the remainder of the division.4. Moving on to paragraph.

The DRAKON-diagram of the most common divisor algorithm is shown in Figure 3.19.

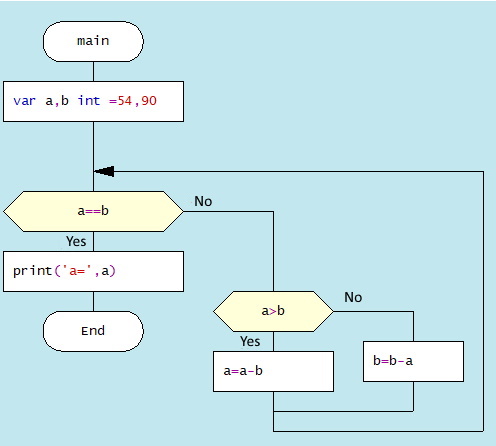


Figure 3.19 Euclid algorithm DRAKON-diagram

**3.4. From DRAKON-diagram to program code** Each DRAKON-diagram corresponds to the program module. Figure 3.20 shows the structure of the array sorting algorithm consisting of a series of separate modules (DRAKON-diagrams) and the main DRAKON-diagram containing the main program’s main Go design:

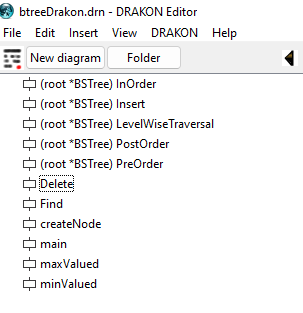


Figure 3.20. Tree processing program structure

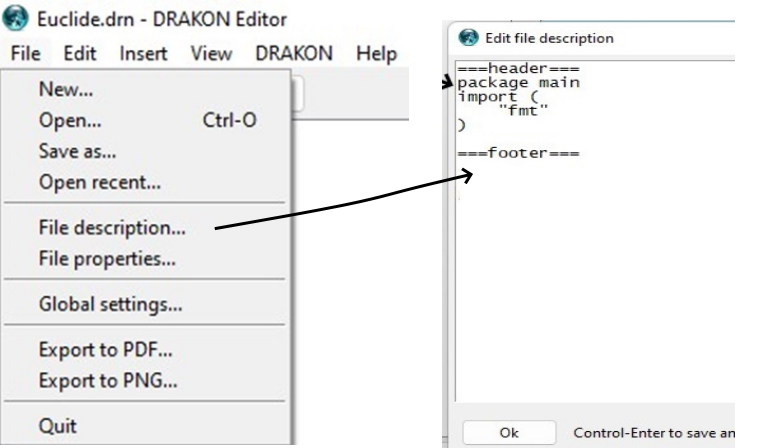
As described above, the Golang programming language program structure consists of packages. The language designs included in the package are included in the File/Description section of the DRAKON WEB Editor interface (Figure 3.21.): 

Figure3.21. Setting of description of program via File/Description

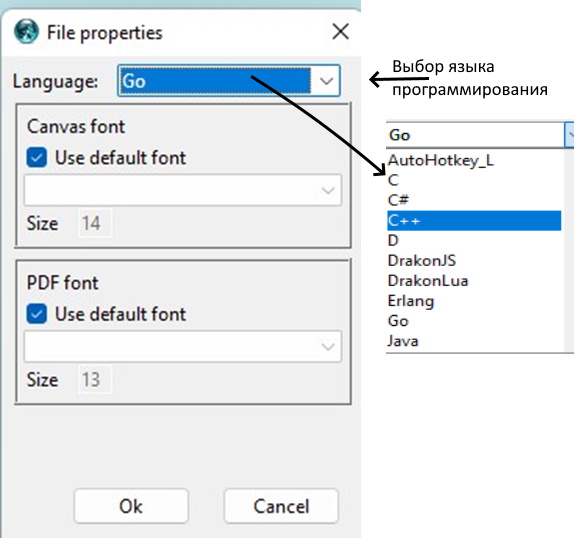
In the field between labels ===header=== and ==footer=== you should have language designs available in each program module, for example, global variables. Next, the user should specify the programming language to use. To do this, open the File properties option (Figure 3.22): 

Figure 3.22. Choosing a programming language

The final stage of development is the generation of the program code (Figure 3.23):

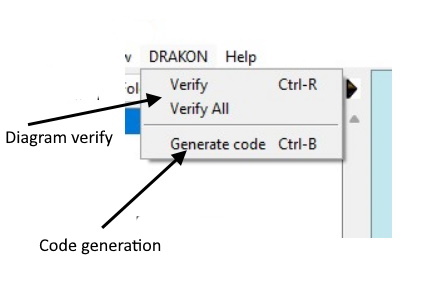


Figure 3.23. Option «Code generation »

The source code in the format . drn and generated code in the format .go are saved in the same directory (Figure 3.24).

**3.5. Execution of the generated code** The generated code can be opened in one of the integrated development environments (IDE): Visual Studio Code, VIM, Eclipse, Atom, Sublime Text and a number of others [https://www.hyperlinkinfosystem.com/blog/best-ides-and-tools-for-golang-development]. This book uses the IDE Visual Studio Code (VSC), the description of which is not included here. The VSC programming environment is presented in Figure3.24:

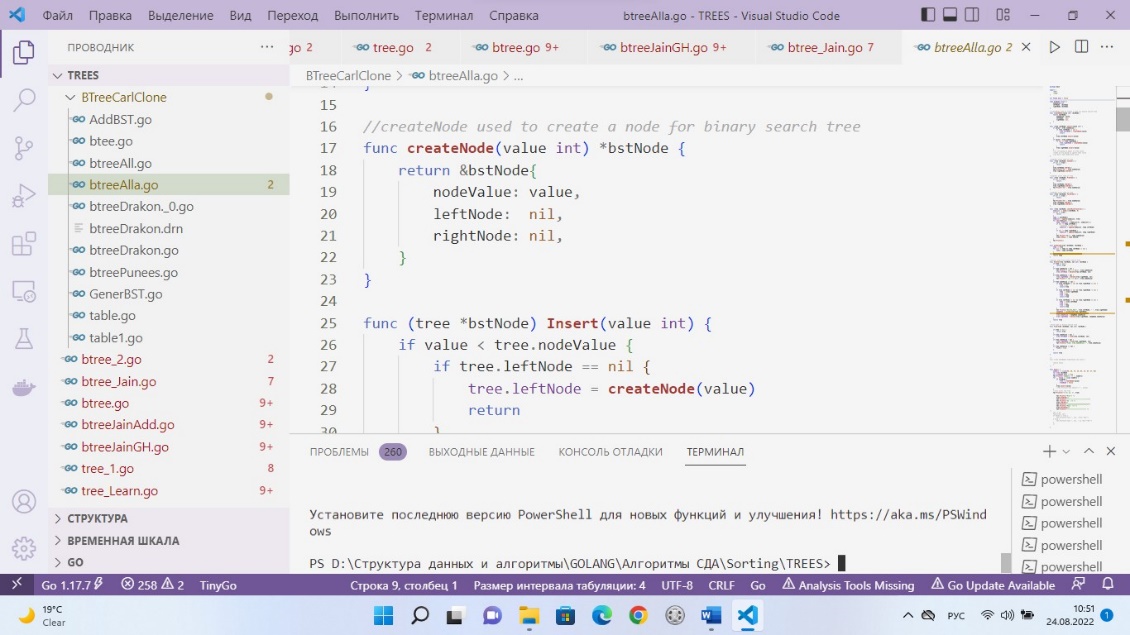


Figure 3.24. Visual Studio Code interface with the Golang-program

Almost as part of the hybrid approach debugging program often have to be carried out in parallel in two environments: DRAKON WEB Editor and Visual Studio Code. Naturally, in case of graphical syntax errors a DRAKON-diagram code generation will not be made, at the same time the presence of semantic errors in the algorithm code will be generated, but the results will be incorrect.

The debugging process of the generated code is recommended to be carried out with the help of the appropriate programming environment toolkit. After debugging is complete, all corrections must be made to the corresponding graphs in the DRAKON-diagram. In any case, it is always necessary to ensure that the contents of the graphs correspond to the diagram and program code in the in the relevant IDE.