TECHNIUM

*Technium Vol. 3, Issue 4 pp.25-37 (2021)*
*ISSN: 2668-778X*
*www.techniumscience.com*

# Paradigms of programming languages and the difficulty of organizing the Algorithms and Data Structures course

**Gregory Korotenko**

Dnipro University of Technology, Ukraine
korotenko.g.m@nmu.one

**Leonid Korotenko**

Dnipro University of Technology, Ukraine
leonid_korotenko@ukr.net

**Abstract**. The article considers the features of drawing up themeaningful material about the features of the implementation of software components (units) when implementing a number of algorithms within the "Algorithms and Data Structures" Course. It is shown that the number of implementations of the original algorithms significantly increases when using different programming languages. The number of possible implementations increases even more when using different paradigms, realized in the selected programming languages.

**Keywords**. learning, algorithms, data structures, programming languages, lexemes, tokens, paradigms, software project, the complexity of the programs implementation, programming libraries, C-like languages.

## 1. Introduction

IDC (International Data Corporation), a provider of market research and information technology consulting services, claims that today digitalization, often referred to as digital transformation, fundamentally transforms all the forms of business today and affects companies in all industries and consumers around the world. Digital transformation is not the evolution of devices (although they will develop), but the integration of intelligent data into everything we do [1]. Organizations are now faced with a huge stream of bits and bytes driven by real-time interrupts from large business clusters. Work in the areas of digital transformations implementation requires not only new technologies, but also new skills [1].

## 2. Description of the method

The analysis of electronic resources and literary sources makes it possible to conclude that the Algorithms and Data Structures course is one of the cornerstones in the curricula related to areas based on the computing usage [2]. The source [3] explicitly states that: "At most of the top tech companies (and many other companies), algorithm and coding problems form the largest component of the interview process. Think of these as problem-solving questions. The

interviewer is looking to evaluate your ability to solve algorithmic problems you haven't seen before". This article examines a number of factors affecting the complexity of organization issues related to the Algorithms and Data Structures course in order to ensure the required level of training for future IT professionals.

Despite the constant growth of publications related to the organization of the process of forming students' competencies in the design and construction of data structures, as well as algorithms for their processing, it should be noted that the issues of structuring the elements of the Algorithms and Data Structures course require new approaches. The fact is that the boundaries of representations and the number of elements in each of the ecosystems of the above triad have significantly expanded since the publication of Niklaus Wirth's book "Algorithms + Data Structures = Programs" in 1975 [4]. For example, the concept of the programming paradigm and its role in the development of computer programs was announced only in 1979 [5].

## 3. Case study

A detailing of stages from a specific software project (SP) to its software implementation transition has been carried out at the first stage of the research.

First of all, it should be noted that in the modern view, the process of forming data structures for SP has significantly expanded its boundaries and it can be represented as follows (Fig. 1) [2].

As known [2], the total number of data structures that can be applied for use in SP exceeds 200 units to date. All of them are formed from the basic data types built into specific programming languages, and for them it is already possible to design and construct the corresponding algorithms: "Programs, after all, are concrete formulations of abstract algorithms based on particular representations and structures of data" [3].

It is known that algorithms have the "Multiplicity" property. This means that "… one and the same algorithm can be represented in several ways. So, it is possible to write a sequence of instructions in plain English or write it as a pseudocode. In the same way, it is possible to write several different algorithms for the development of one and the same software" [6]. Structurally, this can be represented as follows (Fig. 2).

In turn, each of the algorithms presented in Figure 2 can be implemented using different programming languages (PLs) [3]. This process is shown in Figure 3.

And it is well known that, each PL has a number of important features determined by its syntax, semantics and the number of programming paradigms implemented therein [7]. The whole set of these possibilities leads to a new growth of possible implementations of not only one, but also all other original algorithms of the objective. Thus, an increase in the growth of the number of possible algorithms implementations due to the use of different programming paradigms within the same programming language can be represented as follows (Fig. 4).

When programming, i.e. implementing the execution of algorithm steps using high-level PL, the program builder uses the so-called lexemes and tokens [8, 9].

A lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyser as an instance of that token.

A lexical token, or, simply called, a token is a string with an assigned and thus identified value. It is structured as a pair consisting of a token name and an optional token value. The name of the token is the category of the lexical unit [9]. Common token names are:

• identifiers: names chosen by the programmer;

• keywords: reserved programming language names;
• separator (also known as punctuation): punctuation and paired separators;
• operations: symbols that operate on arguments and produce results;
• literals: numeric, logical, text, reference literals;
• comment: line, block (depends on the compiler, if the compiler implements comments as tokens, otherwise it will be removed).

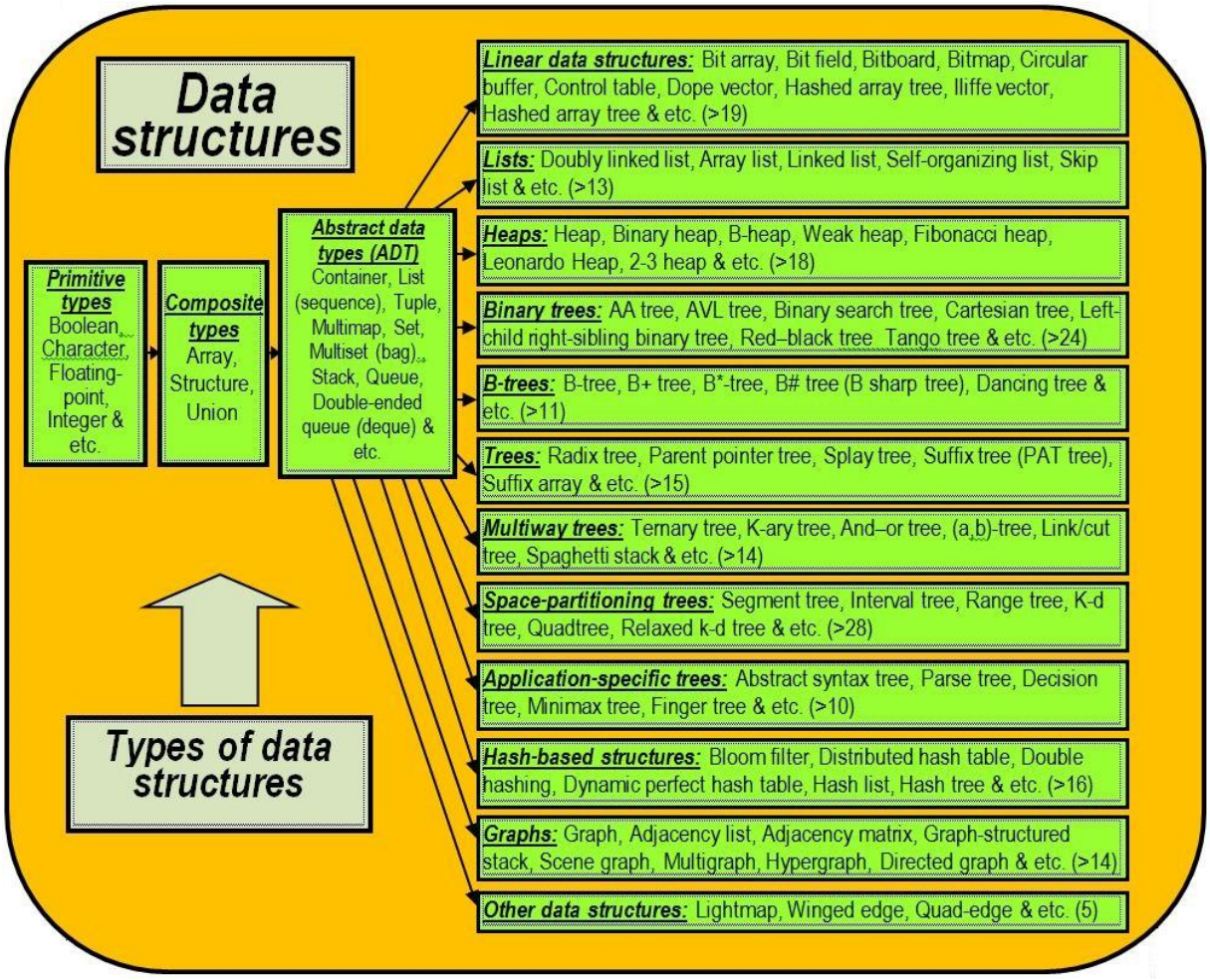


Fig. 1. The process of data structures formation for the software development

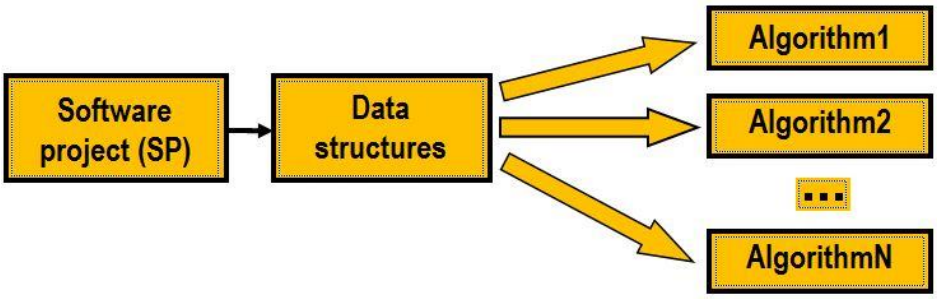


Fig. 2. Variety of algorithms that can be developed to implement one specific SP

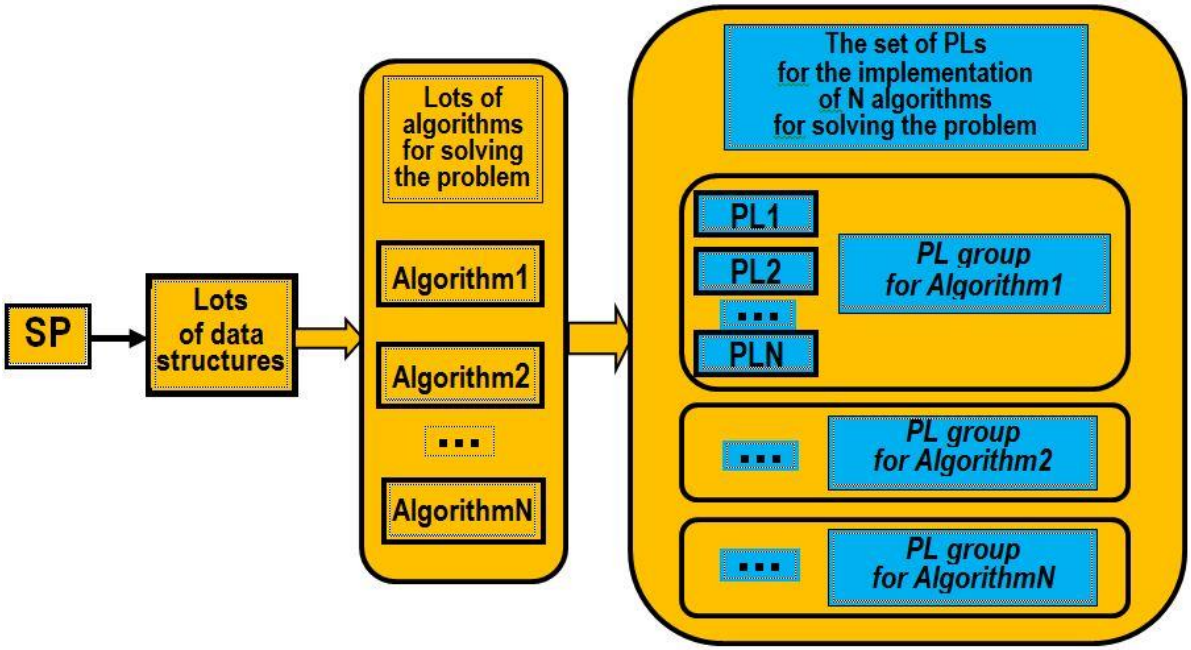


Fig. 3. Implementation of each of the algorithms, using various PLs

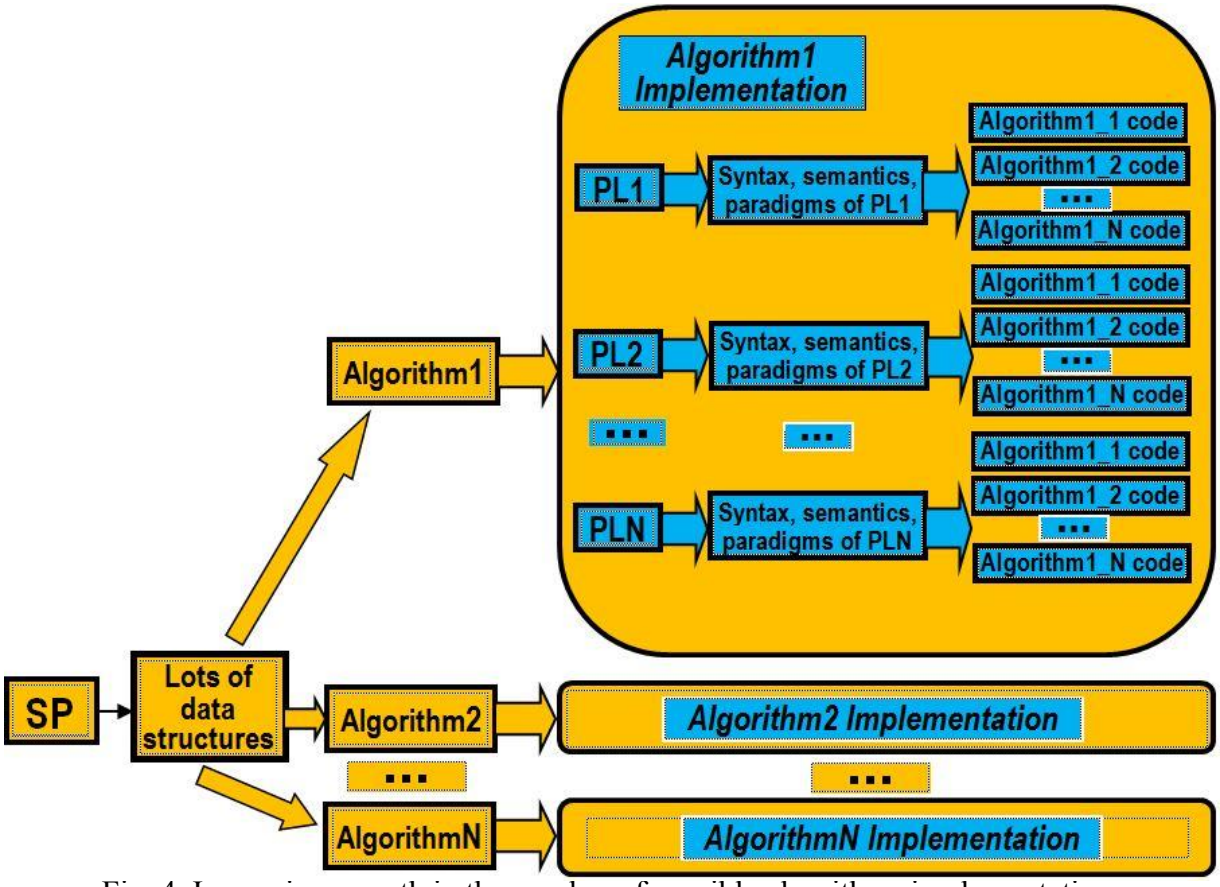


Fig. 4. Increasing growth in the number of possible algorithms implementations

Technium Vol. 3, Issue 4 pp.25-37 (2021)
ISSN: 2668-778X
www.techniumscience.com

Examples of the above-mentioned program units are shown in Table 1.

**Table 1.** Examples of token values

| Token name | Sample token values |
|---|---|
| identifier | `x`, `color`, `UP` & etc. |
| keyword | if, while, return & etc. |
| separator | `}`, `(`, `;` |
| operator | `+`, `<`, `=` & etc. |
| literal | `true`, `6.02e23`, `"music"` & etc. |
| comment | `/* Retrieves user data */`, `// must be negative` |

As precisely noted in the Paper[10], the design of the developed program (in the sense of organizing the semantic interaction of the elements of a given program) is based on programming paradigms that determine how to use all of the above-mentioned tokens. At the same time, when writing program code, the main difficulties in design can also be caused not only by the total number of operators (operations) of a particular programming language, but also by the number of priority levels of these operators.

In addition, each PL supports a number of programming paradigms, the total number of which exceeds 80 units [11, 12] (Table 2).

**Table 2.** Diverse programming paradigms

| | | | |
|---|---|---|---|
| Abductive logic | Action | Actor-based | Agent-oriented |
| Answer set | Array | Array-oriented | Aspect-oriented |
| Attribute-oriented | Automata-based | Automatic | Block-structured |
| Class-based | Concatenative | Concurrent | Concurrent computing |
| Concurrent constraint logic | Concurrent logic | Constraint | Constraint logic |
| Data-driven | Dataflow | Declarative (contrast: Imperative) | Differentiable |
| Domain-specific | Dynamic/scripting | Event-driven | Extensible |
| Flow-based | Functional | Functional logic | Functional reactive |
| Function-level (contrast: Value-level) | Generic | Imperative (contrast: Declarative) | Inductive logic |
| Inductive programming | Intentional | Language-oriented | Literate |
| Logic | Macro | Message passing | Metaprogramming |
| Natural-language programming | Nondeterministic | Non-strict | Non-structured (contrast: Structured) |
| Object-based | Object-oriented | Ontology | Parallel computing |
| Pipeline | Point-free style | Polymorphic | Probabilistic |
| Procedural | Process-oriented | Prototype-based | Purely functional |

**Table 2. (**continuation)

| | | | |
|---|---|---|---|
| Quantum | Reactive | Recursive | Reflective |
| Relativistic programming | Role-oriented | Rule-based | Scalar |
| Scripting | Set-theoretic | Stack-based | Stack-oriented |
| Strict | Structured (contrast: Non-structured) | Structured concurrency | Subject-oriented |
| Symbolic | Synchronous | Tabular | Tacit |
| Tactile | Template | Value-level (contrast: Function-level) | Visual |

Based on the foregoing, Table 3 shows the elements that cause complications in the design of program code and, accordingly, an increase in the options for writing programs to implement the same algorithm in a specific programming language. The data for the table are taken from the sources [13, 14, 15]. The data in the last column is obtained by simple row summation.

**Table 3.** Number of paradigms, programming language tokens, operators and their priority levels

| **Programming language** | **Paradigms quantity (minimal)** | **Priorities quantity** | **Operations quantity** | **Keywords quantity** | **Sum** |
|---|---|---|---|---|---|
| Pascal (Turbo Pascal 7.0) | 3 | 4 | 20 | 48 | 75 |
| Visual Basic for Applications | 4 | 4 | 23 | 43 | 74 |
| C | 2 | 15 | 43 | 32 | 92 |
| Python (3.7) | 5 | 23 | 35 | 35 | 98 |
| Java Script | 4 | 14 | 48 | 38 | 104 |
| Java | 6 | 9 | 44 | 50 | 109 |
| Ruby | 5 | 23 | 43 | 42 | 113 |
| VBScript | 4 | 9 | 23 | 87 | 123 |
| Perl | 8 | 24 | 68 | 40 | 140 |
| C++ | 7 | 18 | 52 | 82 | 159 |
| C# | 6 | 14 | 51 | 102 | 173 |
| Visual Basic 6.0 | 4 | 4 | 23 | 149 | 180 |

**Note:**

1. In different programming languages, the keywords are called differently: Keywords, Reserved Words, Reserved Keywords.
2. In a number of programming languages, descriptions of basic (built-in) data types, and sometimes the names of some functions and operators (operations) are included in the number of keywords.
3. As time passes, some programming languages have faced with a change in the number of components for a number of the above-mentioned categories.

*Technium Vol. 3, Issue 4 pp.25-37 (2021)*
*ISSN: 2668-778X*
*www.techniumscience.com*

The graphical-analytical method for assessing the total complexity of each of the PLs shown in Table 3 shall be applied.

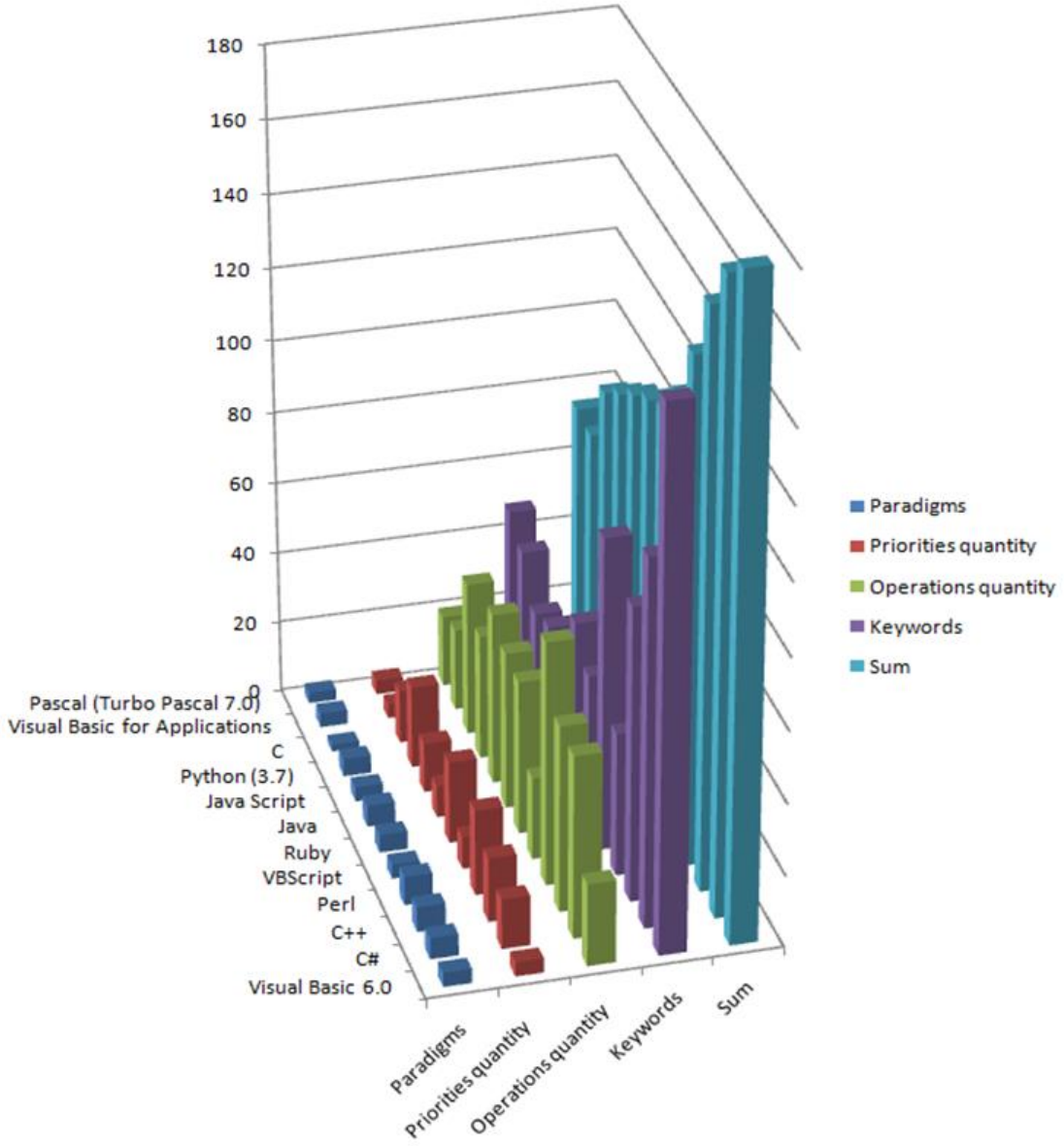


Fig. 5. Estimates of the total complexity of each of the PLs given in Table 3

The analysis of the results of graphical analysis allows to assert that the languages VBScript, Perl, C ++, C # and Visual Basic have a sufficiently large set of interconnected components, which, when teaching students, significantly increases the level of complexity in terms of studying, understanding and applying these entities in their complex interaction.

It is known that each programming paradigm offers its own concept of implementation of the same algorithm for each programming language [7].

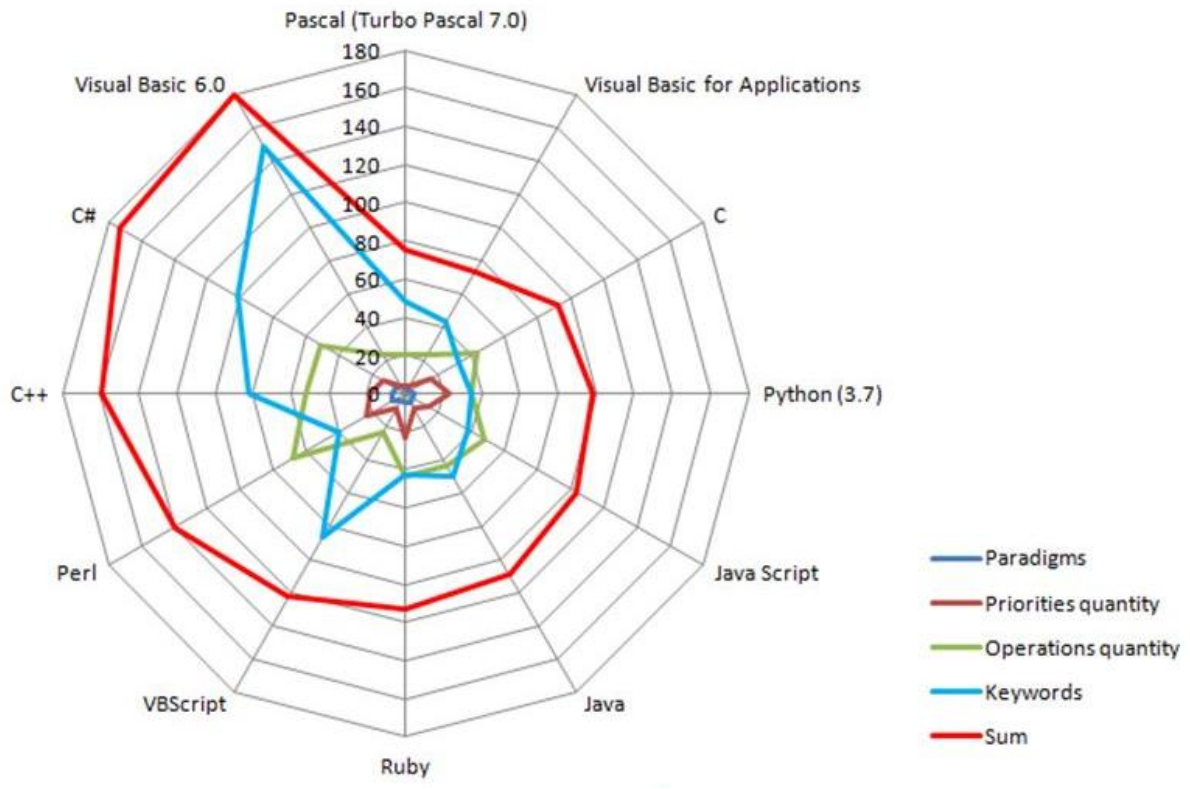


Fig. 6. Data of the Table 3, presented in the form of a radar chart

That is why, each conceptual paradigm is implemented for each PL exclusively by means of lexemes and tokens of the given language. But, each paradigm is implemented even within the framework of each specific PL by different lexemes, and furthermore they are different for different programming languages too. That is, the implementation of object-oriented programming (OOP) concepts within a certain PL uses lexemes and tokens that do not coincide with lexemes and tokens that implement elements, for example, a procedural paradigm within the same language. Furthermore, for example, all constants and variables of the Ruby programming language are objects by definition: "Ruby is a pure object-oriented language and everything appears to Ruby as an object. Every value in Ruby is an object, even the most primitive things: strings, numbers and even true and false" [16].

Considering the fact that, within each PL, the implementation of each paradigm includes different tokens and lexemes, we can say that in each PL, the implementation of software solutions based on different paradigms also expands the possibilities of the options for implementing the original algorithms. Moreover, we can say that the implementation of each algorithm can break down into the number of program structures corresponding to the number of paradigms of the programming language under consideration.

Then, it is possible to say that each new implementation of the algorithm in a specific programming language splits into a number of implementations, at least a multiple of the number of programming paradigms embedded in concrete PL.

Thus, the general scheme of possible software implementations can be represented as follows (Fig. 7).

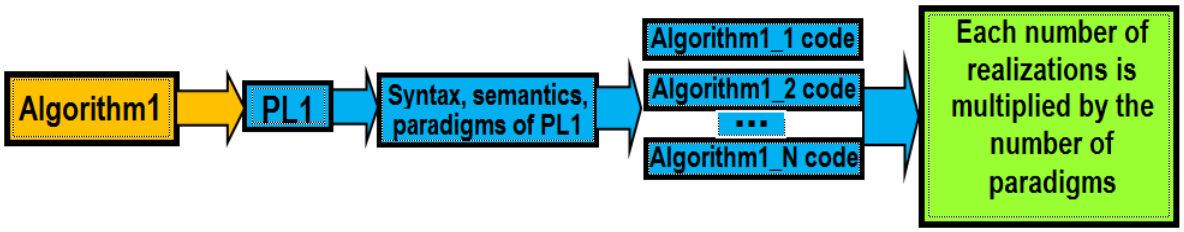


Fig. 7. General scheme of increasing the number of possible software implementations

Based on the obtained estimates of the number of existing and used components within the framework of the problem being solved, the triad proposed by N. Virt: Algorithms + Data Structures = Programs [3] can be represented as follows (Fig. 8).

It is clear that this sequence of concepts must be examine in the training courses Algorithms and Data Structures in accordance with the characteristics of the specialties of future graduates.

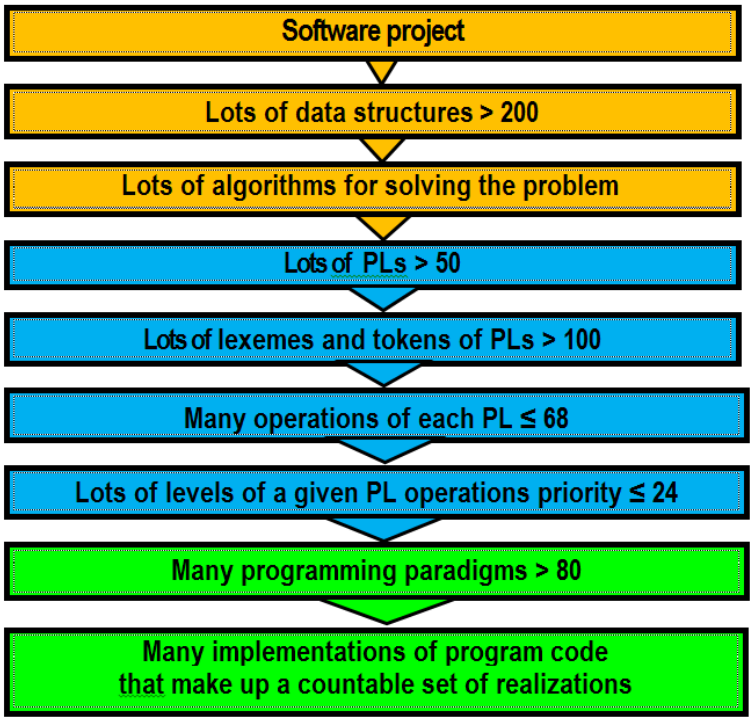


Fig. 8. The main interconnected components underlying the implementation of programs and software projects using various programming languages

## 4. Conclusions

In general, the following conclusions can be drawn. First, structuring of knowledge in the field of algorithms and data structures seems to be a rather difficult task. Secondly, the general set of difficulty levels in this area is quite large and can be represented by the

following quantitative components that are in direct interaction and tend to constantly increase:

– $K_1$ (number of tasks);
– $K_2$ (number of algorithms);
– $K_3$ (number of programming languages);
– $K_4$ (number of programming paradigms);
– $K_5$ (number of abstract data types (ADT));
– $K_6$ (the number of data structures that implement the ADT);
– $K_7$ (the number of built-in data types in programming languages that implement data structures);
– $K_8$ (the number of keywords (tokens) of a specific programming language involved in the design of algorithms for the implementation of ADT operations);
– $K_9$ (the number of operations of a specific programming language involved in the construction of data structures);
– $K_{10}$ (the number of priority levels of operations of a specific programming language involved in the design of algorithms and data structures);
– $K_{11}$ (the number of operations of a specific programming language involved in the design of algorithms for the implementation of ADT operations);
– $K_{12}$ (the number of operations of a specific programming language involved in the processing of data items stored in data structures);
– $K_{13}$ (the number of operations of a specific programming language involved in the design of algorithms for solving the actual specific problem posed).

Therefore, it is not surprising that in the programming languages ratings (TIOBE [17], PYPL [18] and RedMonk [19], where C-like languages [20] are highlighted in grey), the first rows of the table are occupied by C-like languages and languages with a significant number libraries that implement the most commonly used algorithms and data structures (Table 4).

**Table 4.** Programming language ratings according to TIOBE, PYPL and RedMonk

| Place | TIOBE April 2021 | PYPL April 2020 | RedMonk January 2021 |
|---|---|---|---|
| 1 | C | Python | JavaScript |
| 2 | Java | Java | Python |
| 3 | Python | Javascript | Java |
| 4 | C++ | C# | PHP |
| 5 | C# | C/C++ | C# |
| 6 | Visual Basic | PHP | C++ |
| 7 | Javascript | R | CSS |
| 8 | Assembly language | Objective-C | TypeScript |
| 9 | PHP | TypeScript | Ruby |
| 10 | SQL | Swift | C |
| 11 | Classic Visual Basic | Kotlin | Swift |
| 12 | Delphi/Object Pascal | Matlab | R |
| 13 | Ruby | Go | Objective-C |
| 14 | Go | VBA | Shell |
| 15 | Swift | Ruby | Scala |
| 16 | R | Rust | Go |
| 17 | Groovy | Ada | PowerShell |

**Table 4. (**continuation)

| Place | TIOBE April 2021 | PYPL April 2020 | RedMonk January 2021 |
|---|---|---|---|
| 18 | Perl | Scala | Kotlin |
| 19 | MATLAB | Visual Basic | Rust |
| 20 | Fortran | Abap | Perl |

The attractiveness of C-like languages for programmers is noticeable if this list to be considered “in a large scale”, that is, as a whole.

If one considers this list "in a small scale", that is, the first five languages in the above table, which has practically not changed in recent years, then it can be noted that the leaders are mainly languages with a clearly expressed procedural paradigm, which contributes to the rapid entry “into the profession” when learning programming: C, C ++, JavaScript, Python (examples on the Website 70 Years Of “Hello, World!” With 50 Programming Languages) [21]. In addition, each of these languages has a developed ecosystem: they have a large number of IDEs & Code Editors: C / C ++ more than 27 [22], JavaScript – more than 15 [23], Python – more than 12 [24], Java – more than 13 [25], PHP – more than 20 [26], C # – more than 10 [27]. They are also supported by various auxiliary programming libraries: C – 127 libraries [28], C++ – 122 libraries [29], JavaScript – 103 libraries [30], Python – 137000 libraries [31], Java – more than 100 libraries [32], PHP – more than 50 libraries [33], C # – more than 100 libraries [34].

In the course Algorithmization and Programming it is proposed to start with the C ++ language and focus on the concepts of "built-in data types" ("bool", "char", "int", "float" and etc.) and "built-in data structures" (arrays, strings, structures), and to familiarize students with recursion, sorting and searching methods based on their use. The main emphasis should be placed on the design of data structures for the developed software projects, and then on the algorithms for their processing. Thus, it is necessary to emphasize that "DATA STRUCTURES + ALGORITHMS = PROGRAM". This will facilitate preparation for mastering the course Algorithms and Data Structures.

## References

[1] David Reinsel, John Gantz, John Rydning. The Digitization of the World From Edge to Core. URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf (accessed 17 April 2021).

[2] Korotenko, G., & Korotenko, L. (2020). The Algorithms and Data Structures course multicomponent complexity and interdisciplinary connections. Technium: Romanian Journal of Applied Sciences and Technology, *2*(5), 161-171. Retrieved from https://techniumscience.com/index.php/technium/article/view/1310 (accessed 17 April 2021).

[3] G.L. McDowell: Cracking the Coding Interview. 6th Edition. 189 Programming Questions and Solutions. CareerCup, LLC, Palo Alto, CA. 2016.

[4] Niklaus Wirth. Algorithms + Data Structures = Programs. Prentice-Hall.Inc, 1975. – 366 p.

[5] R. W. Floyd. The Paradigms of Programming Communications of the ACM, 22(8): 455-460, 1979.

[6] Algorithm and its properties / https://www.includehelp.com/data-structure-

tutorial/algorithm-and-its-properties.aspx (accessed 17 April 2021).
[7] Peter Van Roy (2009-05-12). "Programming Paradigms: What Every Programmer Should Know" (PDF). info.ucl.ac.be. / file:///C:/Users/2D6D~1/AppData/Local/Temp/Programming_Paradigms_for_Dummies_What_Every_Progr.pdf (accessed 17 April 2021).
[8] S Lexical_analysis / https://en.wikipedia.org/wiki/Lexical_analysis (accessed 17 April 2021).
[9] Alfred V. Aho; Monica S. Lam; Jeffrey D. Ullman; Ravi Sethi. Compilers. Principles, Techniques, & Tools, 2nd Ed. Pearson Education, Inc, 2007. – 1009 p.
[10] Harper, Robert (1 May 2017). "What, if anything, is a programming-paradigm?". FifteenEightyFour. Cambridge University Press. / http://www.cambridgeblog.org/2017/05/what-if-anything-is-a-programming-paradigm/ (accessed 17 April 2021).
[11] Programming_paradigm / https://en.wikipedia.org/wiki/Programming_paradigm (accessed 17 April 2021).
[12] Paradigms / http://progopedia.com/paradigm/ (accessed 17 April 2021).
[13] Programming paradigm URL: https://en.wikipedia.org/wiki/Programming_paradigm (accessed 17 April 2021).
[14] Comparison of multi-paradigm programming languages / https://en.wikipedia.org/wiki/Comparison_of_multi-paradigm_programming_languages (accessed 17 April 2021).
[15] Reserved keywords count by programming language / https://stackoverflow.com/questions/4980766/reserved-keywords-count-by-programming-language (accessed 17 April 2021).
[16] Ruby - Object Oriented / https://www.tutorialspoint.com/ruby/ruby_object_oriented.htm (accessed 17 April 2021).
[17] TIOBE Index for April 2021, available at https://www.tiobe.com/tiobe-index// (accessed 17 April 2021).
[18] PYPL PopularitY of Programming Language. Worldwide, Apr 2021 compared to a year ago, available at https://pypl.github.io/PYPL.html (accessed 17 April 2021).
[19] The RedMonk Programming Language Rankings: January 2021, available at https://redmonk.com/sogrady/2021/03/01/language-rankings-1-21/ (accessed 17 April 2021).
[20] List of C-family programming languages, available at https://en.wikipedia.org/wiki/List_of_C-family_programming_languages (accessed 17 April 2021).
[21] 70 Years Of "Hello, World!" With 50 Programming Languages, available at https://medium.com/javarevisited/70-years-of-hello-world-with-50-programming-languages-2400de893a97 (accessed 17 April 2021).
[22] 27 Best C & C++ IDEs & Code Editors, available at https://codecondo.com/top-10-ide-for-c-and-cplusplus-for-programmers/ (accessed 17 April 2021).
[23] Top 15 JavaScript IDEs and JS Editors for Frontend Development, available at https://jelvix.com/blog/best-javascript-ides (accessed 17 April 2021).
[24] 12 Best Python IDEs And Code Editors In 2021, available at https://www.softwaretestinghelp.com/python-ide-code-editors/ (accessed 17 April

2021).
[25] Best Java IDE 2021 | Most Popular Java IDE, available at https://hackr.io/blog/best-java-ides (accessed 17 April 2021).
[26] 20 BEST PHP IDE and Code Editor Software in 2021 [Free/Paid], available at https://www.guru99.com/best-php-editor-ide-free.html (accessed 17 April 2021).
[27] 10 Best C# IDE for Windows, Linux, Mac (2021 Update), available at https://www.guru99.com/best-csharp-ide.html (accessed 17 April 2021).
[28] Wikipedia. Category:C (programming language) libraries, available at https://en.wikipedia.org/wiki/Category:C_(programming_language)_libraries (accessed 17 April 2021).
[29] Wikipedia. Category:C++ libraries, available at https://en.wikipedia.org/wiki/Category:C%2B%2B_libraries (accessed 17 April 2021).
[30] Wikipedia. List of JavaScript libraries, available at https://en.wikipedia.org/wiki/List_of_JavaScript_libraries (accessed 17 April 2021).
[31] 34 Open-Source Python Libraries You Should Know About, available at https://www.mygreatlearning.com/blog/open-source-python-libraries/#:~:text=There%20are%20over%20137%2C000%20python,data%20manipulation%20applications%20and%20more (accessed 17 April 2021).
[32] The Top 100 Java Libraries in 2018 – Based on 277,975 Source Files, , available at https://www.overops.com/blog/the-top-100-java-libraries-in-2018-based-on-277975-source-files/ (accessed 17 April 2021).
[33] Top 50 Useful PHP Library List, available at https://geekyhumans.com/50-most-popular-php-library/ (accessed 17 April 2021).
[34] Top C# Libraries of 2018 – Based on 18,471 GitHub Repositories, available at https://www.overops.com/blog/top-c-libraries-of-2018-based-on-18471-github-repositories/ (accessed 17 April 2021).