The 13th International Scientific Conference
eLearning and software for Education
Bucharest, April 27-28, 2017
10.12753/2066-026X-17-000

Generic online algorithm interpreter with dynamic data visualizations. Case study on sorting algorithms

Ciprian-Bogdan Chirila
Department of Computer Science and Information Technology,
University Politehnica Timişoara, V. Pârvan Blv. no. 2, Timişoara, Romania
chirila@cs.upt.ro

***Abstract**: Nowadays IT is present in our everyday life like: cars, phones, clothes, watches, houses etc. In this context the developing industry for such products is in a continuous need of specialists. One of the basic features of IT specialists is the ability to program, namely to be able to write code to be understood by machines and devices. Each year company representatives ask universities to double the number of graduates. Teaching students to write code is a complex and hard task proved by very high dropout rate after the first year of study in the distance learning programs. In order to ameliorate this issue and to help students learn the theoretical concepts of programming tutors can use efficiently learning objects deployed on LMSs. Learning objects can be simple or plain (LO), generative learning objects (GLO), and auto-generative learning objects (AGLO). GLOs offer instantiable reusable templates for different learning objectives. AGLOs add automatic instantiation enhanced with random variables in order to increase content diversity. The runtime of a program is for first year students like a black box, they do not control the code very well. Debuggers may be a solution in this sense, but usually they are built for experienced programmers, they have to be configured, sometimes they are difficult to follow, do not offer spectacular views of the changing data etc. In order to help students understand how their code behaves at runtime we propose an online code interpreter with runtime data visualizations and we analyze the impact of such a generative learning object on their progress. Thus, we consider that such efforts will contribute to the idea that the technology can support learning efficiency.*

***Keywords:** generative learning objects, online algorithm interpreter, run-time data visualization, cloud IDE*

I. INTRODUCTION

The local IT industry is in a continuous growth because of the domain development explosion. Nowadays IT is present in our everyday life like: cars, phones, clothes, watches, homes etc. In this context the developing industry for such products is in a continuous need of specialists. One of the basic features of IT specialists is the ability to program, namely to be able to write code to be understood by machines and devices. Each year local company representatives ask universities to double the number of graduates.

Teaching students to program is a complex and hard task that usually starts in the first and second year of study and is exercised during the whole undergraduate studies period [16,25]. High school graduates that succeed in the first year have different backgrounds. Some of them have some programing skills obtained in informatics specialization high schools. Some have only basic knowledge to use the computer and some no such skills at all. So there is a need for solutions that help teachers rise students’ knowledge levels at the same level, highest possible, to get a solid foundation for starting the studies. The need is even more severe in distance learning programs where are enrolled several graduates from different specializations some even non-technical.

In this sense teachers may recommend students several online text tutorials, video tutorials etc. The main issue with these resources is that they do not offer a strong feedback on their progress. They can solve this issue with a strong self-discipline, but in several situations extra explanations are needed for understanding fully the concepts.

To help students learn the theoretical concepts of programming and background elements can be used in an efficient manner learning objects in all its forms simple (LO) [10], generative learning objects (GLO) [4,13,14,21,22,23], and auto-generative learning objects (AGLO) [5,6,7,8,9]. GLOs offer instantiable reusable templates for different learning objectives. AGLOs add automatic instantiation enhanced with random variables in order to increase content diversity.

Data visualizations are used in almost all research domains like big data, data mining, information design etc. and are considered to be a visual communication. In the field of learning and more specifically learning of programming concepts a general approach is found on multiple websites where data structures and algorithms are presented. Basically the approach is based on predefined static meta-models (heaps, hashes, linked lists, etc.) instantiated with random or user defined data which are then animated explaining the student the algorithm behavior.

In laboratory works students directly implement classic algorithms in some programming language looking at the manipulated data structures through debuggers or using printing instructions. Most of the time debuggers seem to be difficult to use for beginner programmers. For example, in the CodeBlocks [12] free IDE for C and C++ programing, users must create a project, locate some folder to save it on the disk, integrate the working module inside it etc., just to be able to activate the debugger. For the functional debugger the student need to set a break point in certain lines, insert the relevant variables in the watches window. Not to mention that command line debuggers, like the GNU project debugger (gdb) [18], are less attractive since they do not have GUI. Inside debuggers, for some data structures like arrays and matrixes the visualizations are relevant, but for the dynamic data structures where pointers are used the visualizations are poor. Modern and commercial IDEs like Microsoft Visual Studio [24] have powerful debuggers but they are not always accessible to students because of cost and hardware configuration requirements. Still, they provide free student versions for learning purposes with limited capabilities which can used only for the first year students. Mobile devices rarely have compilers and debuggers for educational purposes because of security and hardware or software limitations.

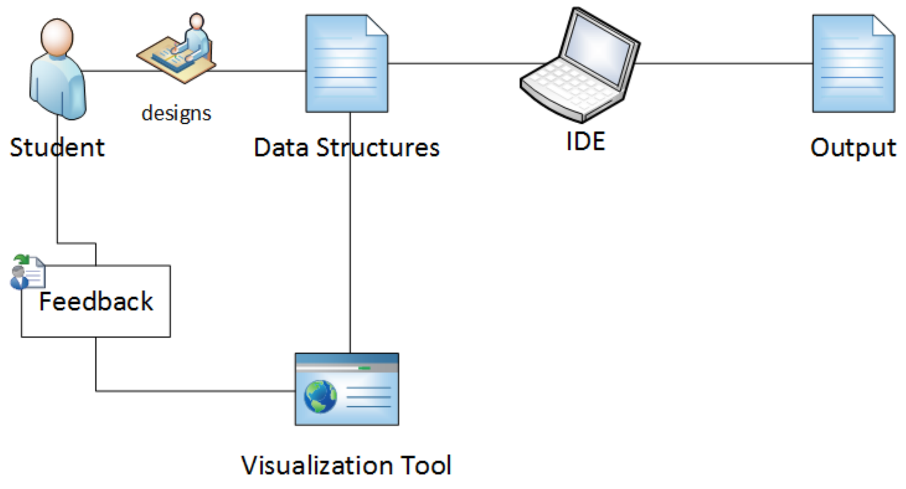

Figure 1 – The Data Structures Visualization Approach

In this paper we investigate the potential for relevant visualizations (see Figure 1) of several data structures and potential implementation solutions. We consider that for a better and faster understanding of basic programming concepts it is very important for the student to be able to define its data structures and to be able to visualize it dynamically. The goal is to add new layer of learning support for the student between the phase of design and phase of coding. We will perform our analysis through the laboratory works that we have in the first year of study for the distance learning program students.

Another organizational aspect refers to the fact that in a laboratory class the number of students tend to increase. Back in the 90's the number of students in a laboratory class were around 6, nowadays we have 18 up to 20 students in a class. The total 120 minutes divided by 6 is a 20 minutes individual time spent with each student for explanations and evaluations, while today the total time of 120 minutes divided by 20 gives a 6 minutes individual time per student. The decreasing individual time must be replaced by e-learning tools like in the current approach.

The paper is structured as follows. In chapter II present related works in the area of data structures and data visualizations for programmers. In chapter III we inventory all the visualization features needed in a common data structure laboratory class. In chapter IV we analyze the approach from the technical point of view highlighting the main steps for an implementation. Chapter V concludes and sets the perspectives.

II. RELATED WORKS

[1] is an online platform where are presented several data structures like: linear structures, search structures, quadratic sorts, N log N sorts, search algorithms, compression algorithms, graph algorithms etc.

[6] presents a manual approach of online visualization for sorting algorithms. The approach is based on: i) a manual translation of the code into JavaScript [15]; ii) manual annotation with instruction lines to produce visualization of specific semantic details, inserted in a systematic manner. For example, the array positions were colored differently according to algorithm variables. A generic behavior is expected to be inferred from multiple case studies.

[9] presents a different approach for visualization and gamification of data structures based on differential approach of the algorithm states.

[24] is a commercial IDE created by Microsoft Corporation having facilities for visual design, code writing and debugging.

[18] is a command line free debugger for the GNU Compiler Collection and it is dedicated to experienced programmers.

Even the IDE and data visualization libraries evolved in the last decade, debuggers seem to keep the very same features as they had in the 80's and 90's. Instead of a text interface now they exhibit mostly the same features in evolved GUIs. From the data structures visualization point of view no remarkable progress is noticed. We consider that such approaches are well oriented in the direction of data structures understanding but they are not enough when learning must be done fast and with students having non-technical backgrounds.

D3JS is an online free visualization library that is based on JSON data structures. Such a library has great potential in creating online visualizations for data structures and algorithms. The challenge is to convert the structure to be visualized into a JSON object.

Scalable Vector Graphics is a markup language which offer graphical primitives for creating online visualizations. The advantages of SVG are: simplicity of the primitives, cross browser and cross platform compatibility being able to run on mobile devices.

III. VISUALISATION FEATURES INVENTORY IN DSA DISCIPLINE

In this chapter we will analyze the Data Structures and Algorithm (DSA) discipline curricula in order to inventory the necessary visualization features for the teaching of data structures.

Firstly, we need to make a clear distinction between data structure visualization and algorithm visualization. Data structures are described in programs as meta-models allowed by the language typing system like: scalars, structures, pointers, projections, enumerations etc. Composition rules of programming languages allow combining such types in order to describe complex data structures. Algorithms are expressed as finite steps reflected by changes on the data structures. To be more specific we will analyze from the point of view of statically typed structured programming languages.

Secondly, data structures can be static or dynamic. Static data structures visualizations are straightforward since usually they have support in the programming language and in the IDEs debuggers. Dynamic data structures are allocated in the memory heap at runtime at random memory addresses which makes visualization more difficult.

Thirdly, we will take a look at how basic types are visualized in practice, formally and informally: i) variables are represented as rectangles containing a value and having their name around; ii) pointers are represented using a rectangle for the variable and an arrow to point to the target memory location; iii) arrays are represented as one row tables, possibly with indexes; iv) stacks are represented vertically as one column tables; v) queues are represented horizontally as one row table; vi) matrixes are represented as bi-dimensional tables; vii) structures are represented by a rectangle divided into fields where values are put or pointer arrows emerge (UML like); viii) trees and graphs are represented as diagrams with nodes and edges.

Next, we will dive into details for each data structure topic. The searching and sorting algorithms use the array static data structure. Nevertheless, they can be applied also on linked lists. Insertion sorting needs visualizing the sorted zone, the comparisons during insertion and the insertion itself. Selection sorting needs visualizing the sorted zone, the comparisons for computing the minimum and the final interchange. Bubble sorting needs visualizing the comparisons and the interchanges. Bi-directional bubble sorting needs to visualize the array inner walking zone, the comparisons and the interchanges. Shell sorting requires to visualize the separate subarrays due to increments, and the elements from insertion sorting. Heap sorting is a special case since the visualization of a heap (as binary tree diagram) structure must be provided. The visualization stages involve the creation of the heap and then the repeated reconstruction. The quick sorting algorithm requires visualization of current range of application, the pivot, the identification of candidates to swap, the swap and the range division and restarting. For the bin sorting algorithm the visualizations required are for the elements to swap and the swap itself. For direct radix sort we need to visualize the binary representation of numbers, the current bits, the counter table (counting, accumulation and distribution) and the passes. For interchange radix sort the visualization must include the binary representation of numbers, the current bit highlighting, the swaps candidate identification, the swap and the current working range. For merge sorting with three and four tapes we need to visualize the tuples in a tape, the splitting operations and the merging operations.

For brute force substring search we need to visualize the compared substring from the main string. For Knuth-Morris-Pratt substring search we need to visualize the creation of the jump table from the substring content, the comparisons and jumps. For Boyer-Moore substring search we need to visualize the creation of the jumping table, the character comparisons and jumps during the algorithm run.

For recursive backtracking problems the solution vector creation and evolution must be visualized together with the appropriate constraints.

For linked lists the visualization is based on compartmented rectangles and arrows for the pointer. For multiple linked lists the visualization is based on the same elements.

For hash tables the visualizations required are for the indexed array and for the hash function evaluation to understand how distribution works.

IV. TECHNICAL ANALYSYS

In order to implement such an approach with a high automation degree we will analyze several potential approaches.

One implementation idea is to create a customized framework for each type of algorithm: searching, sorting, sub-string searching etc. where all visualization features are set up. In such an environment new algorithms can be added together with visualization semantic actions. This idea requires programming knowledge from the tutor side.

The second implementation idea is more complex and requires compiling techniques. We build an online parser which parses the new added algorithm and produces a JavaScript version connected to the visualization framework facilities. The main challenge is to interpret the semantics of the algorithm and to generate the proper visualization semantic actions. This drives us to the idea of code annotations and visualization profiles.

Code annotations allow program data structures to be selected and linked with profile elements for the visualization of the algorithm. The annotations will add semantic information about the visualization of the data.

For each type of algorithm we will create a visualization profile. Each visualization profile will have a set of predefined objects which will be visualized like arrays, indexes etc. For example, the visualization profile for sorting algorithms will include at least the array, the indexes and the auxiliary variables. Profiles will have switches for the representation of the elements, e.g. decimal or binary.

From the tutor point of view the requirement is now to select the algorithm and the proper visualization profile, to run and see the algorithm visualization.

Specifically, for an algorithm written in C we use the JavaCC [26] grammar repository and we extract only a subset of the language features to simplify the parsing process. Using the Visitor design pattern we implement a translator to a JavaScript module to be integrated in the visualization framework.

V. CONCLUSIONS AND PERSPECTIVES

In this paper we presented an analysis regarding visualization of data structures in the context of a laboratory works set. Pure structures can be easily visualized, while specific semantic must be annotated or inferred from multiple similar use cases. Of course, we found exceptions that cannot be covered by the general inferred logic. Annotations were proposed in this sense.

The approach is useful in other educational projects dedicated to training elders to program. A different future use of the approach can be in industrial IDEs e.g. automotive embedded systems, where, for example, they use finite state automata for modelling behavioral states [2,3].

As future work we intend to implement fully the approach. The implementation is not straight forward and additional solutions or compromises are required. We indent to analyze the opportunity of integration of the resulting tool in a cloud IDEs like Cloud9 [10], CodeAnywhere [11] or IdeOne [19].

References

[1] AlgoViz Committee, 2017. The Algorithm Visualization Portal, http://algoviz.org

[2] Bogdan, R., 2016. Guidelines for developing educational environments in the automotive industry, 1st International Conference on Smart Learning Ecosystems and Regional Developments, Timisoara, Romania.

[3] Bogdan, R., Ancușa, V., 2016. Developing e-learning solutions in the automotive industry. World Journal on Educational Technology: Current Issues. 8(2), 139-146.

[4] Burbaite, R., Bespalova, K., Damasevicius, R., Stuikys, V., 2014. Context Aware Generative Learning Objects for Teaching Computer Science, International Journal of Engineering Education, vol. 30, no. 4, pp. 929-936.

[5] Chirila, C.B., 2013. A Dialog Based Game Component for a Competencies Based E-Learning Framework, In proceedings of SACI 2013 8-th IEEE International Symposium on Applied Computational Intelligence and Informatics, pp. 055--060, Timisoara, Romania, May.

[6] Chirila, C.B., 2014. Educational Resources as Web Game Frameworks for Primary and Middle School Students, In proceedings of eLSE 2014 International Scientific Conference eLearning and Software Education, Bucharest, Romania, April.

[7] Chirila, C.B., 2014. Generative Learning Object Assessment Items for a Set of Computer Science Disciplines, In proceedings of SOFA 2014 6-th International Workshop on Soft Computing Applications - Advances in Intelligent and Soft Computing, Springer Verlag, ISSN 1867-5662, Timisoara, Romania, July.

[8] Chirila, C.B., Ciocarlie, H., Stoicu-Tivadar, L., 2015. Generative Learning Objects Instantiated with Random Numbers Based Expressions, BRAIN - Broad Research in Artificial Intelligence and Neuroscience, vol. 6, no. 1-2, Bacau, Romania, October.

[9] Chirila, C.B., Raes, R., Roland, A., 2016. Towards a Generic Gamification of Sorting Algorithms, In Proceedings of 12th International Symposium on Electronics and Telecommunications ISETC 2016, Timisoara, Romania, October.

[10] Cloud9, 2017. Cloud9, Your development environment in the cloud, https://c9.io/

[11] Codeanywhere, 2017. Cross platform Cloud IDE, https://codeanywhere.com/

[12] CodeBlocks, 2017. The open source, cross platform, free C, C++ and Fortran IDE, http://www.codeblocks.org/

[13] Damasevicius, R., Stuikys, V., 2008. On the Technological Aspects of Generative Learning Object Development, Third International Conference on Informatics in Secondary Schools - Evolution and Perspectives (ISSEP 2008), pp. 337-348, Torun, Poland, July.

[14] Damasevicius, R., Stuikys, V., 2009. Specification and Generation of Learning Object Sequences for e-Learning Using Sequence Feature Diagrams and Metaprogramming Techniques, In proceedings of 2009 9-th International Conference on Advanced Learning Technologies.

[15] ECMA International, 2016. Standard ECMA-262 ECMAScript 2016 Language Specification, http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[16] Ermalai, I., Drăgulescu, B., Ternauciuc, A., Vasiu, R. 2013. Building a module for inserting microformats into Moodle. Adv. Electr. Comput. Eng. Jan 1;13(3):23-6.

[17] Florea, A., Gellert, A., Florea, D., 2016. Teaching programming by developing games in Alice, The International Scientific Conference eLearning and Software for Education, Bucharest: "Carol I" National Defence University.

[18] GNU Foundation, 2017. GDB: The GNU Project Debugger https://www.sourceware.org/gdb/

[19] Ideone, 2017. Ideone, https://ideone.com/

[20] IEEE Learning Technology Standards Committee, 2016. LOM working draft v4.1 Available: http://ltsc.ieee.org/doc/wg12/LOMv4.1.htm

[21] Jones, R., Boyle, T., 2007. Learning Object Patterns for Programming. Interdisciplinary Journal of Knowledge and Learning Objects, vol. 3.

[22] Jung, H., Park, C., 2012. Authoring Adaptive Hypermedia using Ontologies International Journal of Computers Communications & Control, ISSN 1841-9836, volume 7(2), pp. 285-301, June.

[23] Karpova, M., Shmelev, V., Dukhanov, A., 2016. Information resource based on scientific software as a core of interdisciplinary learning resources, 2016 IEEE Frontiers in Education Conference (FIE), pp. 1-5, Eire, PA, USA.

[24] Microsoft Corporation, 2017. Visual Studio IDE, https://www.visualstudio.com/

[25] Naaji, A., Mustea, A., Holotescu, C., Herman, C. 2015. How to Mix the Ingredients for a Blended Course Recipe, Journal of Broad Research In Artificial Intelligence and Neuroscience, 6(1-2), ISSN 2067-3957, Bacau, Romania, October.

[26] Oracle Corporation, 2017. Java Compiler Compiler - Tha Java Parser Generator, https://javacc.java.net/