# CVisualizer: Guided C/C++ Execution Visualization Embedded in Interactive Textbooks

Jinning Liu
University of Toronto
Toronto, Canada
imjinning.liu@mail.utoronto.ca

Nora Liu
University of Toronto
Toronto, Canada
noral.liu@mail.utoronto.ca

Salma Emara
University of Toronto
Toronto, Canada
salma@ece.utoronto.ca

## Abstract

Novice C and C++ programmers can find it difficult to trace how control flow, variables, and memory change during program execution. Static explanations can show individual program states but do not directly represent the transitions between them. This paper presents *CVisualizer*, a guided execution-visualization component embedded within interactive textbooks. Built on the Python Tutor execution engine, CVisualizer augments step-by-step execution with instructor-authored, step-specific annotations, execution highlighting, and code folding to guide learners' attention and explain relevant state changes. We report a formative deployment of five CVisualizer instances in a first-year programming course. During the approximately two-week deployment, 99 learners generated 9,875 step-navigation events. Thirty-two learners completed a voluntary survey; 87.5%, 84.4%, and 81.3% selected *Agree* or *Strongly Agree* for statements concerning perceived conceptual understanding, reduced mental effort, and usability, respectively. These descriptive findings demonstrate that a subset of students used the embedded visualizers and that survey respondents generally perceived them favorably. Because tool use and survey participation were voluntary, the findings do not establish learning gains or effects across the full course population.

## CCS Concepts

• **Theory of computation → Program semantics**; • **Human-centered computing → Visualization systems and tools**; • **Social and professional topics → Computing education**.

## Keywords

C/C++ programming, Memory visualization, Program execution, Educational tools, Active learning, Notional machines

**ACM Reference Format:**
Jinning Liu, Nora Liu, and Salma Emara. 2026. CVisualizer: Guided C/C++ Execution Visualization Embedded in Interactive Textbooks. In *Proceedings of the 2nd ACM Virtual Global Computing Education Conference V.2 (SIGCSE Virtual 2026), November 12–15, 2026, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3795860.3846237

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
*SIGCSE Virtual 2026, Virtual Event, USA*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2491-6/2026/11
https://doi.org/10.1145/3795860.3846237

## 1 Motivation and Contribution

Learning to program requires a model of how source code produces changing runtime state. Novices often have difficulty constructing and applying such notional machines [1, 4], especially when tracing pointers, stack frames, heap objects, and control flow in C/C++. Static diagrams capture individual states but do not themselves show the transition between states. Program-visualization systems expose those transitions. For example, Python Tutor supports line-by-line execution and displays variables and data structures for user-supplied programs [2]; Jeliot and UUhistle animate program execution in other instructional settings [3, 5].

## 2 CVisualizer Design

CVisualizer contributes a configurable instructional layer for prepared C/C++ examples rather than a new execution engine. It retains Python Tutor's state rendering and navigation while allowing an instructor to place a short explanation at a chosen execution step, emphasize the active line, and fold auxiliary code. A reusable `<c-visualizer>` web component places the example beside the surrounding textbook explanation. This design addresses a specific authoring need: an instructor can guide attention to the purpose of a state transition without directing learners to a separate tool.

Each example is executed offline through the Python Tutor backend. CVisualizer loads the resulting JSON trace in the browser, so deployment does not require server-side execution of learner code. Annotations and fold ranges are configured per instance, and client-side hooks synchronize them with playback. This architecture permits several independent visualizers on one textbook page. Its principal constraint is equally important: the deployed component presents instructor-prepared traces and does not execute arbitrary learner-edited code. It is therefore intended for guided examples, not as a replacement for an interactive compiler or debugger.

The features are motivated by practical presentation concerns. Step-specific annotations colocate an instructor's explanation with the state to which it refers. Active-line highlighting makes the current control-flow position visible. Code folding lets an author initially hide setup or helper code while keeping it available on demand. These mechanisms are intended to focus attention; the present study does not test whether any individual feature reduces cognitive load or improves comprehension.

## 3 Formative Deployment

We embedded five CVisualizer instances in the online textbook for a first-year programming course for electrical and computer engineering students. The examples concerned memory allocation and linked data structures. The course enrolled more than 440

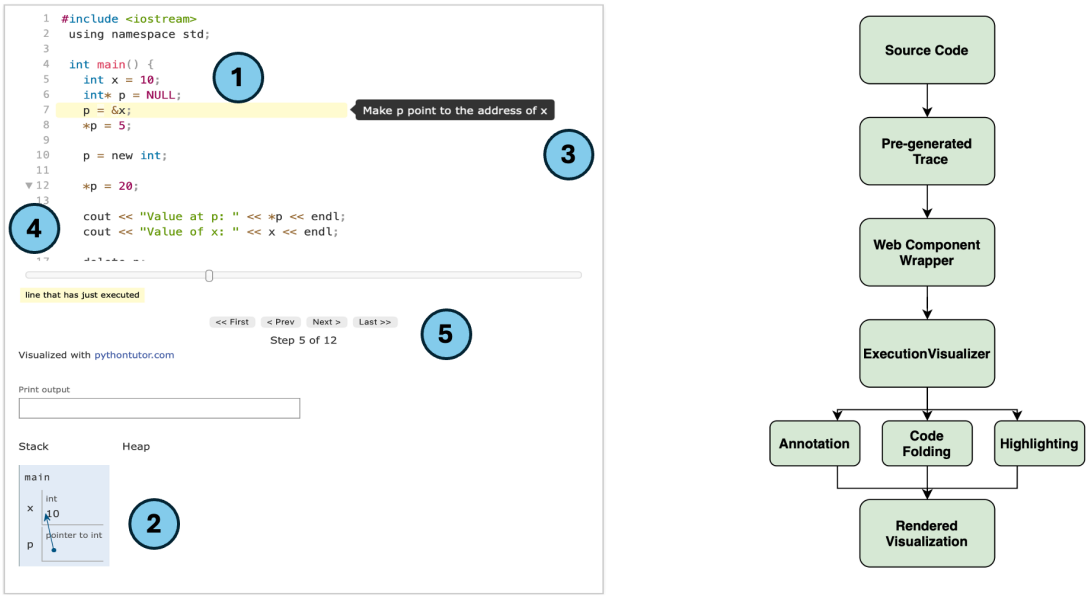


**Figure 1: CVisualizer interface and implementation pipeline. The interface combines (1) execution highlighting, (2) runtime stack and heap state, (3) contextual annotations, (4) code folding, and (5) step-navigation controls. In the pipeline, source code is executed offline to produce a trace, which is loaded by the web-component wrapper and rendered by Python Tutor's `ExecutionVisualizer` with CVisualizer's instructional overlays.**

students, and use of the visualizers was voluntary. The deployment analyzed here covered approximately two weeks during the final-examination period. Institutional ethics approval covered data collection, and records were anonymized before analysis.

We used three descriptive evidence sources. First, Google Analytics recorded presses of the *First*, *Previous*, *Next*, and *Last* controls for each visualizer. These events measure navigation actions, not attention, trace completion, comprehension, or learning. Second, a voluntary anonymous survey asked students to rate three statements on a five-point scale. The statements concerned whether the visualizer (Q1) improved their understanding of pointers, memory, and allocated structures; (Q2) reduced the mental effort required to study C/C++ code; and (Q3) was easy to use and more interactive than static figures. Third, the instructor supplied a qualitative reflection on classroom use. The reflection was not collected through a structured observation protocol.

Ninety-nine distinct users generated 9,875 navigation events across the five visualizers. Activity peaked during final exam preparation. These data show that a subset of enrolled students used the controls repeatedly, but event totals cannot distinguish productive study from repeated, accidental, or exploratory actions.

Thirty-two students completed the survey. For Q1, 28 of 32 respondents (87.5%) selected *Agree* or *Strongly Agree*. The corresponding counts were 27 of 32 (84.4%) for Q2 and 26 of 32 (81.3%) for Q3. Thus, most respondents reported favorable perceptions on all three items. Because participation and tool use were voluntary, these percentages describe respondents only. The items record self-reported perceptions and are not direct measures of conceptual understanding, mental effort, usability performance, or comparison with a control condition.

**Table 1: Voluntary survey responses ($n = 32$).**

| Item | Agree/Strongly Agree |
|---|---|
| Q1: Perceived conceptual understanding | 28 (87.5%) |
| Q2: Perceived reduction in mental effort | 27 (84.4%) |
| Q3: Ease and interactivity | 26 (81.3%) |

The instructor reported that using CVisualizer reduced time spent redrawing stack frames and pointer values during a classroom walkthrough and allowed more discussion of why state changes occurred. The instructor also perceived more follow-up questions about program behavior than in previous offerings. These comments identify possible uses for the tool and directions for study, but they are anecdotal and may be influenced by differences between course offerings.

## 4 Conclusion and Future Work

CVisualizer embeds guided Python Tutor traces in C/C++ textbook examples. In this formative deployment, 99 self-selected students used the component; 32 of more than 440 enrolled students (<7.3%) completed the survey and generally viewed it favorably. These findings neither establish learning effects nor generalize to the course population. Future work will compare CVisualizer with static diagrams using tracing, retention, observation, and step-level timing measures while isolating the effects of individual features.

## References


[1] John Benedict du Boulay. 1986. Some Difficulties of Learning to Program. *Journal of Educational Computing Research* 2 (1986), 57 – 73. https://api.semanticscholar.org/CorpusID:61129234

[2] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE)*. Association for Computing Machinery, New York, NY, USA, 579–584. doi:10.1145/2445196.2445368

[3] Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. 2004. Visualizing programs with Jeliot 3. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Gallipoli, Italy) *(AVI '04)*. Association for Computing Machinery, New York, NY, USA, 373–376. doi:10.1145/989863.989928

[4] Juha Sorva. 2013. Notional Machines and Introductory Programming Education. *ACM Transactions on Computing Education* 13 (06 2013), 8:1–8:31. doi:10.1145/2483710.2483713

[5] Juha Sorva and Teemu Sirkiä. 2010. UUhistle: a software tool for visual program simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '10)*. Association for Computing Machinery, New York, NY, USA, 49–54. doi:10.1145/1930464.1930471