Communicating using Program Traces

Chris McDonald
Computer Science and Software Engineering
University of Western Australia
Crawley, WA 6009, Australia
chris.mcdonald@uwa.edu.au

Matthew Heinsen Egan
Australian Centre for Geomechanics
University of Western Australia
Crawley, WA 6009, Australia
matt.heinsenegan@uwa.edu.au

ABSTRACT

Supported by the growth of embedded devices, the IoT, and modern standardization, C remains an important language for undergraduate students. However, when presented to relatively inexperienced programmers, its very limited runtime support is a source of frustration for both students and educators, alike.

SeeC is a novice-focused tool for the C programming language that displays and records the execution of programs, detects runtime errors, and enables students to review their programs' data and execution history. Students can deterministically replay a program, and reason backwards from a runtime error to determine its true cause. The approach can lead to students' reduced frustration and greater understanding of program behaviour.

This presentation provides a brief overview of SeeC and a discussion on how its program traces provide an opportunity for students to communicate their questions and misunderstandings with educators, and for educators to provide examples and debugging challenges to students.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; K.3.2 [**Computers and Education**]: Computer and Information Science Education

Keywords

Novice programmers, debuggers

I. MOTIVATION

The standard C programming language can be especially difficult for newcomers. In particular, pointers and manual memory management can present difficulties both in understanding at a conceptual level, and in debugging the laconically described runtime errors which result from their misuse. Most newly developed novice-focused debugging systems are designed for object-oriented programming languages, as introductory teaching has focused on these languages, and the most notable tools developed to assist novice C programmers are predominantly unmaintained.

Research from the fields of programming languages and compilers has developed many advanced debugging techniques, but they are typically only supported by tools designed for expert programmers, rather than for novices. The complexity of these tools, and the time required to learn their use, at even a modest level, are often insurmountable hurdles for novice students. Furthermore, while these tools can be used to locate runtime errors, they do not assist novice programmers to *understand* those errors, or more generally to understand the behaviour of their programs.

Several novice-focused tools have implemented graphical program visualizations and automatically generated explanations of program behaviour. These features have been shown to assist novice programmers with constructing knowledge and debugging programs in numerous evaluations, such as those described by [1], [9], [2], and [5]. However, few of these tools have supported the C programming language, and those that do are typically incomplete or unmaintained.

SeeC introduced a novice-focused systems for creating graphical visualizations of the runtime memory state of C language programs, and for generating natural language explanations of C program fragments. It is built upon a previously developed novice-focused debugging system for the C programming language, augmenting the existing runtime error detection and execution tracing with program visualizations and natural language explanations. SeeC runs on each of Linux, macOS, and Windows.

II. THE DESIGN OF SEEC

The *SeeC* project provides a novice-focused system for the standard C programming language supporting execution tracing and runtime error detection. SeeC itself is built upon the Clang and LLVM projects [8]. This provides SeeC with robust support for the C programming language while avoiding the unsustainable maintenance requirements inflicted by bespoke implementations of parsing, compiling, or interpreting. For a detailed explanation of the SeeC system, see the discussion in [4, 6, 3].

SeeC uses a modified version of the Clang front-end to perform compile-time instrumentation of students' programs. The produced executables contain additional code that both checks for runtime errors and creates a trace of the execution. The trace can be used to recreate the *visible state* of the program at any point of time in the recorded execution.

Clang's parsing and semantic analysis libraries are used to create an Abstract Syntax Tree (AST) from a program's source code. Each node in the AST represents a declaration or statement in the program and provides rich semantic information – the same information that is used during compilation. When an execution trace is loaded the program's

SPLICE Spring 2019 Workshop, CS Education Infrastructure for All II: Enabling the Change in conjunction with ACM-SIGCSE 2019, Minneapolis, 27th Feb 2019.

AST is reconstructed, allowing us to link runtime states to relevant AST nodes. This provides a mapping between the program's static source code and its dynamic state.

III. USING SEEC IN TEACHING

SeeC has been used in our second year undergraduate unit *CITS200 Systems Programming*, recently presented to 280-320 students. After recent changes in our university's degree structures, the unit resulted after a merging of a first year introductory C programming unit and a second year operating systems unit. The unit provides an introduction to the C programming language operating system fundamentals (operating system structure, processes, memory management, and file-systems), and the use of C as a vehicle to access and control operating system services.

The unit is core for students taking Software Engineering, Computer Science, or Data Science majors. Most students have completed at least one of the core units introducing the Java or Python programming languages. Another quarter of the unit's cohort are specialising in Electronic Engineering, and are undertaking Systems Programming as prior study for a later units. Unfortunately, Systems Programming is usually their first programming focused unit in their degree.

We have employed SeeC in recorded lectures to introduce general programming concepts, such as loops, function calls, and simple data structures, and the more challenging aspects of C, such as its use of pointers and dynamic memory allocation. Students have commented on ease of use, as a "drop-in" replacement for their system-provided compiler, and SeeC's clarity of error reporting, particularly its ability to describe runtime errors in English, with reference to their source code.

IV. OPPORTUNITIES

Through the dramatic growth in our class sizes, and the increased interest in our teaching of C to students not planning to graduate in Computer Science, we are currently refocusing the design and use of SeeC. Historically, while SeeC was still under continued development, it was only installed on our department's laboratory computers, and was mostly used by students in their closed laboratory sessions. Over time, we have made SeeC available to our students to install on their own laptop or home computers, but this has proved a challenge for many students because it required the (well documented) installation of several software packages, including modified versions of the *Clang+LLVM* compiler system. These difficulties have been noted in [7].

SeeC's traces provide an excellent opportunity for communication between students, teaching assistants, and educators. Students can communicate their questions about their programs with other student and educators, either by identifying specific line numbers, functions, or points-in-time of the execution of their programs. Educators can similarly provide examples and debugging challenges to students. In addition, the format provides the opportunity for students and educators to add colourful "popup notes" or even audio samples (including questions and misunderstandings) to program traces.

However, another remaining challenge is the size of SeeC's trace files, which easily grow to hundreds of megabytes for typical in-laboratory exercises and projects. This size makes it difficult for students to (quickly) store their program traces to portable USB drives, and to communicate them via email.

We are currently developing a cloud-based version of SeeC which will address these current challenges. Cloud-based storage has rapidly dropped in price, and holding the program traces for even several hundred students has become a reality. Students will be able to communicate within groups, to enqueue their (synchronous or asynchronous) questions for teaching assistants, and to undertake longer dialogs about a single programming problem. Moreover, retaining students' program traces (with permission), provides a rich opportunity to investigate *how* students are actually approching the development and debugging of their programs.

V. REFERENCES

[1] P. Brusilovsky. Program visualization as a debugging tool for novices. In *INTERACT '93 and CHI '93 conference companion on Human factors in computing systems*, CHI '93, pages 29–30, New York, NY, USA, 1993. ACM.

[2] J. H. Cross, II, T. D. Hendrix, D. A. Umphress, L. A. Barowski, J. Jain, and L. N. Montgomery. Robust generation of dynamic data structure visualizations with multiple interaction approaches. *Trans. Comput. Educ.*, 9:13:1–13:32, June 2009.

[3] M. H. Egan and C. McDonald. Runtime error checking for novice c programmers. In *4th Annual International Conference on Computer Science Education: Innovation and Technology (CSEIT'13)*, pages 1–9, 2013.

[4] M. H. Egan and C. McDonald. Program visualization and explanation for novice c programmers. In *Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148*, ACE '14, pages 51–57, Darlinghurst, Australia, Australia, 2014. Australian Computer Society, Inc.

[5] P. J. Guo. Online python tutor: Embeddable web-based program visualization for cs education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, pages 579–584, New York, NY, USA, 2013. ACM.

[6] M. Heinsen Egan and C. McDonald. Reducing novice c programmers' frustration through improved runtime error checking. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 322–322, New York, NY, USA, 2013. ACM.

[7] R. Ishizue, K. Sakamoto, H. Washizaki, and Y. Fukazawa. Pvc: Visualizing c programs on web browsers for novices. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, pages 245–250, New York, NY, USA, 2018. ACM.

[8] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proc. of the 2004 International Symposium on Code Generation and Optimization (CGO'04)*, Palo Alto, California, Mar 2004.

[9] P. A. Smith and G. I. Webb. Transparency Debugging with Explanations for Novice Programmers. In M. Ducassé, editor, *Proceedings of the 2nd International Workshop on Automated and Algorithmic Debugging*, 1995.