Visualizing dynamic memory operations in C programs

Matthew Heinsen Egan
The University of Western Australia

May 19, 2010

1 Background

The C programming language is important to learn. At close to forty years old, it remains one of the most popular programming languages. There are a great number of projects implemented in C; from end-user programs, device drivers, libraries, to the linux kernel and more. It has also influenced, or been extended to produce, many other widely used languages. Yet C has many abstract concepts that can be confusing for newcomers, especially beginning undergraduates who may have a limited knowledge of general programming concepts to draw from. Among the most difficult of these is the use of pointers and manual memory management, as shown by Lahtinen *et al.* [5]. There is significant research into understanding the difficulties students have with pointers, and improving the methods of teaching them.

Explanations of pointers and dynamic memory in textbooks (and lecture notes) are typically emphasized with diagrams that abstractly represent the layout of memory at selected states of program execution. While helpful, they are unlikely to cover all of the situations a student will encounter difficulty with, due to their limited number, focus on correct usage, and necessarily static nature.

Visualization tools have been shown by Hundhausen *et al.* [4] to be effective when the student is actively engaged in the learning process, though, with few exceptions, ineffective when students are simply viewing a visualization. Historically visualization tools designed for teaching have focused primarily on algorithms and data structures. This focus often means that details of the use of pointers and dynamic memory are abstracted out of the visualization, and errors in their usage are not generally considered (although there are exceptions, such as the HDPV tool by Sundararaman and Back [8]).

There are also many tools designed for visual debugging by experienced programmers, such as GNU DDD [2]. These tools are powerful debugging environments, with many advanced features and complex interfaces, that may confuse new users; particularly students that are relatively new to programming and likely have little experience with debuggers. This makes them less suitable for use in learning new

concepts, as it was suggested in studies by Ma *et al.* [6] that overly complex visualizations, and visualizations that require an associated verbal explanation, are not effective in helping students.

2 Aim

In this project I propose to design and construct a visualization tool specifically aimed at helping students to understand pointers and dynamic memory usage in the C programming language, then evaluate the effectiveness of the tool through testing and experimentation with students that are studying C. The tool will focus on visualizing the execution of arbitrary C code entered by the user, rather than static animations.

The first task is designing the specifics of the program implementation. This covers end-user issues such as the interface design and supported features, as well as underlying details such as whether to interpret user code in a simple virtual machine, compile it to native executable code and connect to the process as a debugger, or to use some intermediate tool (such as the GNU Project Debugger [3]). The design will be influenced by several factors, including:

- Research into similar existing tools (algorithm visualization tools, visual debuggers).
- Research into what information and presentation methods have benefited students in the past.
- Experimentation into the effectiveness of the available options.
- The requirements of the desired feature set.

A modular design approach will be used, so that specific visualization components can be added or removed with minimal effort. Furthermore the underyling system for determining memory state from user code should be interchangeable without affecting the other components of the tool. At this time a single implementation using the GNU Project Debugger [3] is planned for the project.

The intermediate stage of the project is the implementation of the tool itself. At this time the principal features intended for the tool are as follows:

- Visualize memory usage, at a user controlled state of execution, of arbitrary user-supplied C code.
- Allow code that has run-time errors related to the usage of pointers and dynamic memory.
- Display a meaningful representation of these errors, when present, that will be useful to the user.

- Show with accuracy and sufficient detail the effects of memory allocation and deallocation (both on the heap, and on the stack during changes in scope), pointer assignment, manipulation, dereferencing, and use of the unary address-of operator &.

The following are features that are currently seen as desirable and, allowing for completion of the core functionality, will also be implemented:

- Interactive expression evaluation, during execution, against the current memory environment.
- Allow movement through the historical execution states of the program. That is, allow the user to move backwards and forwards through the programs execution.
- A configurable level of detail in display and control. This should allow new users to start using the program without being hindered by overly complex interfaces, while allowing them to enable additional functionality as it becomes useful to them.
- Handle "snippets" of user code (that is code that does not constitute a valid C program, but is sufficient to be executed given some assumptions about its surrounding environment).

Note that the desirability, and hence inclusion, of these features may change as the project progresses. This is a tentative list.

Finally tests will be designed and undertaken to evaluate the effectiveness of the tool in real world scenarios (that is, with people new to the C programming language, and specifically its usage of pointers and dynamic memory). The tool will be evaluated by both feedback from the users, and experimental studies as to whether it provides a benefit to learning.

3 Software and Hardware Requirements

The project will target Linux and OS X. Currently it is planned to use Tcl/Tk [7], C++, boost [1] and GDB [3]. There are no special hardware requirements.

4 Proposed Timeline

Early June	Design finalized
	Development started
Mid August	Development primarily completed
Late August	Experiment details finalized
Early September	Experiments performed

References

[1] Boost C++ Libraries. http://www.boost.org/, 2010. [Online; accessed 19-May-2010].

[2] DDD - Data Display Debugger - GNU Project - Free Software Foundation (FSF). http://www.gnu.org/software/ddd/, 2010. [Online; accessed 19-May-2010].

[3] GDB: The GNU Project Debugger. http://www.gnu.org/software/gdb/, 2010. [Online; accessed 19-May-2010].

[4] Christopher D. Hundhausen, Sarah A. Douglas, and John T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, 2002.

[5] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *SIGCSE Bull.*, 37(3):14–18, 2005.

[6] Linxiao Ma, John Ferguson, Marc Roper, Isla Ross, and Murray Wood. Improving the mental models held by novice programmers using cognitive conflict and jeliot visualisations. *SIGCSE Bull.*, 41(3):166–170, 2009.

[7] John K. Ousterhout. *Tcl and the Tk toolkit.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1994.

[8] Jaishankar Sundararaman and Godmar Back. HDPV: interactive, faithful, in-vivo runtime state visualization for C/C++ and Java. In *SoftVis '08: Proceedings of the 4th ACM symposium on Software visualization*, pages 47–56, New York, NY, USA, 2008. ACM.