February 7, 2019

CPSC 490
Senior Project Proposal

Student: Addie Elwood

Advisor: James Glenn

Background

C is a wonderful programming language that is used as part of the core sequence of the Computer Science major here at Yale. One main benefit is that it helps students understand the nuts and bolts of programming, because it is a relatively low-level language. When writing a C program, it's important (in many cases necessary) to understand how every piece works. Lots of things can go wrong in a C program, and it may not be obvious where or why. Even beyond C itself, programming is a difficult thing to learn. New programmers need to adjust to a new way of thinking that can be non-intuitive, and can struggle to find bugs in their programs.

A debugger is a useful tool for any programmer. Debuggers have been created for all kinds of popular programming languages. However, there isn't an easily accessible C debugger for new programmers. The most popular C debugger is gdb, which is a command-line tool. While gdb is powerful and very helpful, it can be difficult for new programmers to learn, and they often won't bother because they also have to learn C itself at the same time. Having a debugger that they could interact with in a more intuitive, visual way would help new C programmers fix and understand their programs. A few graphical user interfaces for gdb have been created, such as DDD, onlinegdb.com, and gdbgui. However, each of these has drawbacks for Yale students. DDD is old and difficult to use, and the latter, more recently created apps don't have integration with the Zoo. Students in our core CS classes have to maintain two versions of the code: a copy on the Zoo and a local copy, and must constantly scp or use FTP programs to transfer files to be able to run these visual debuggers locally.

Debuggers have been incorporated into Integrated Development Environments. IDEs are all-in-one applications which allow programmers to write, compile, run, test, and debug their programs from one dashboard. However, they can be overwhelming for new programmers, with lots of fancy and confusing buttons and tabs. For beginning programmers,

only a simple visualization tool is needed. Such tools have been created for the purposes of CS education for other languages like Java.

Project

My project is to implement a visual debugger for C programs, designed to help beginner programmers understand the flow of their code. There are two main aspects to this project: the debugger, and the graphical application that displays it.

Debugger:

I have two options when it comes to the debugger.

I could implement my own C debugger. This would allow me to learn more about systems programming (a particular interest) and how debugging works. It would also allow me flexibility when it comes to the interface and integration with the second part of the project. It would shift the focus of the project toward understanding debugging itself.

I could use gdb, the existing C debugger. I would figure out how to call it and use it from another application[1]. The advantage here is that gdb is very powerful and already does everything I need it to. There's no worry of it breaking because I didn't make it. This would shift the focus of the program away from understanding debugging.

Graphical Interface:

There are also a few options here, and I need to do some more research before selecting one.

I could create a desktop application from scratch, preferably using a language or framework with some good graphics libraries. I could create a web application, since I have an interest in exploring those. Web apps have lots of frameworks available that I could build off. In either of these first two cases, I need to make some decisions about how much to support. A full IDE has the ability to edit, compile, run, and debug code in one. It may be feasible to implement a bare-bones version of an IDE for C.

Another option is to extend an existing text editor (one that is easily integrated to access the Zoo) to include debugging visualization. This would focus the work on the debugging visualization rather than completely from scratch. Another possibility is to devote

[1] One of the research articles I found does this exact thing with gdb

this entirely to visualizing, not editing, the program. Then I could decide how much of the source code to display at a given time. This would mean programmers would have to switch programs to edit the code, but doing this may allow me to focus on the meat of the project, which is the visualization of debugging.

It may be possible to implement this project as a plugin to Sublime Text. Sublime Text is a popular, free text editor used widely at Yale because of its SFTP plugin, which allows users to edit files on the Zoo using Sublime. Sublime has the features I don't want to waste time implementing, like editing code and syntax highlighting. It's also possible to create your own plugins for Sublime, although I'm not certain of the extent of things you are allowed to manipulate or do. I need to make sure that the visualization I want to do could be incorporated the way I want.

I also need to design (and then implement, of course) the visualization itself, since there is a variety of information available through debuggers. I want to structure the debugging visualization to be useful but not overwhelming, and to allow programmers to truly go step-by-step, to encourage them to question all of their assumptions when tracking down a bug. There is some debate about the granularity to display when it comes to line-based vs. expression-based evaluation. The most important aspect will be linking the debugging information to the source code. One option for this is to display debugging information on the same line as the code that's being executed.

The final stage of the project will be to solicit user feedback to gauge the usefulness of the visualized debugger I will create.

Deliverables

- Mini report on research into technologies
- Design of visualization
- Design of code structure
- Code that implements a visual debugger for C
- Report on user feedback

References

1. *ETV: a program trace player for students.* Minoru Terada, ACM SIGCSE Bulletin. September 2005. [link]
2. *Exploring expression-level program visualization in CS1.* Teemu Sirkiaa. Proceedings of the 14th Koli Calling International Conference on Computing Education Research. 2014 [link]
3. *Java visual debugger.* Ali Rafieymehr, Richard McKeever. ACM SIGCSE Bulletin. November 2007. [link]