Embedded Live Code: Status of Client-Side Interactive Programming

Aadil Bhatti
Undergraduate Senior Thesis
University of Illinois at Urbana-Champaign
aadilzbhatti@gmail.com
Advisor: Lawrence Angrave

ABSTRACT

UPDATED—May 11, 2017. Today, static websites and dynamic web applications such as JSFiddle[3] exist to teach programming through the use of static code snippets. We present a project[8] which seeks to transform this paradigm by turning static code snippets into programs which are editable and executable *in the browser* without server-side computation. With this interactive coding environment, users are allowed to safely play and experiment with code and as a result have a more dynamic and interactive learning experience.

ACM Classification Keywords
Computer and Information Science Education

Author Keywords
Programming; Education; Browser; Linux; System Programming; Ace; jOr1k; OR1K; Code; C; Python

INTRODUCTION

There are many web applications whose purpose is to teach users several aspects of programmin, such as writing programs using a specific language, the illustration of an algorithm, or how to solve a posed question. Example websites include CodeAcademy[1], StackOverflow[5] and cprogramming.com[2]. These applications illustrate programming concepts, languages, or paradigms via static code snippets. These snippets illustrate how a solution to the posed problem should look in the language specified by the application. The pitfall of this model, however, is the solution code is static — it exists purely in text form and the user is required to download the text and set up a program to observe the solution in a realistic environment. Applications such as JSFiddle and CodeAcademy allow students to run the solution code in the browser, but this is limited to web code (HTML, CSS, JavaScript) and as such it natively runs in the browser. Online Read-Evaluate-Print Loops (REPLs)[4] exist for many programming languages, but these applications have significant overhead and as of now only run as standalone applications, i.e. they do not support embedding in other websites.

This project[8] provides an alternative interactive coding environment which executes code natively in the browser and is capable of being embedded in existing webpages. This project exists to augment websites which have static text explanations of how to accomplish something using a programming language as well as code snippets illustrating the concept. This project identifies these snippets of code and converts them into runnable, editable programs. This allows the user to see first-hand what the output of a program should be while learning about it. In addition, this allows users to tweak the code to see how changes affect the program, thus giving users a more comprehensive grasp of the concept by observing its behavior with different inputs.

FUNCTIONALITY AND USAGE

Functionality

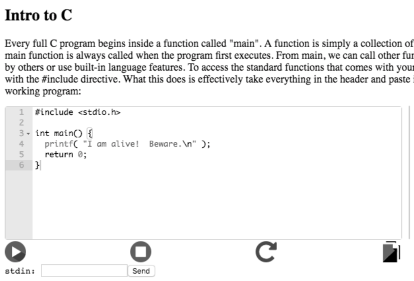
The project provides two client-side components which exist entirely in the browser. These are: A lightweight code editor and a small Linux virtual machine. The code editor contains a set of buttons and controls which users can use to edit and compile the code written in the editor. When the editor sends the program text to the virtual machine, the VM compiles and executes the program whose output is displayed in a small box next to the editor. Currently, C and Python programs are supported by this project.

Usage

To use this project, a developer needs to download the package which contains two JavaScript files as well as the base filesystem for the virtual machine. This filesystem contains the root directory for the virtual machine, and is necessary for the VM to load programs such as `gcc` and `python`. The developer includes the relevant HTML documents in the same directory as the above three components, includes the main `elc.js` file in any HTML documents that need it, and adds a `<div>` tag with ID named `elc` anywhere in the body of the HTML documents (Figure 1). To embed an editor in an HTML page, the developer needs to include a `<pre>` tag with the class name `code` to distinguish it from other elements. Then, any example code can be written within the `<pre>` tags. This code

```
<body>
  <div id="elc"></div>
  <h2>Intro to C</h2> Every full C program begins inside a
  function called "main". A function is simply a collection
  of commands that do "something". The main function is
  always called when the program first executes. From main,
   we can call other functions, whether they be written by
   us or by others or use built-in language features. To
   access the standard functions that comes with your
   compiler, you need to include a header with the #include
   directive. What this does is effectively take everything
   in the header and paste it into your program. Let's look
   at a working program:

  <pre class="code">
#include &lt;stdio.h&gt;

int main() {
  printf( "I am alive!  Beware.\n" );
  return 0;
}</pre>
  <h2>Here is some more Code!</h2>
  <pre class="code">
#include &lt;stdio.h&gt;

int main() {
  printf("I am a second piece of code\n");
  return 0;
}</pre>

<script type="text/javascript" src="elc.js"></script>
```

Figure 1. An example HTML document with all the code necessary to transform a static code snippet



Intro to C

Every full C program begins inside a function called "main". A function is simply a collection of commands that do "something". The main function is always called when the program first executes. From main, we can call other functions, whether they be written by us or by others or use built-in language features. To access the standard functions that comes with your compiler, you need to include a header with the #include directive. What this does is effectively take everything in the header and paste it into your program. Let's look at a working program:

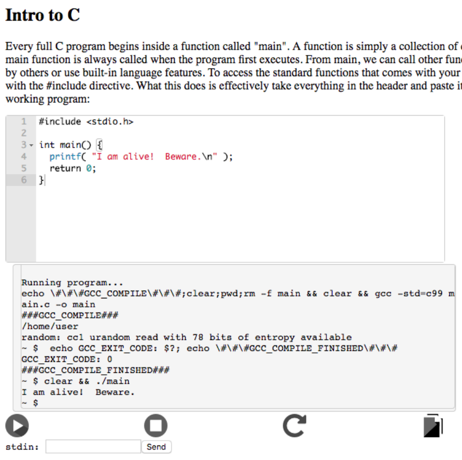
```
#include <stdio.h>

int main() {
  printf( "I am alive!  Beware.\n" );
  return 0;
}
```

Figure 2. The above HTML document rendered in the browser, before pressing "submit"

will subsequently be transformed into the code present within the editor (Figure 2), which will be compiled and run upon pressing the "submit" button (Figure 3). To use different programming languages, add another class attribute to the "code" element containing the code. Adding "c_cpp" or "python" to the class attribute will activate the syntax highlighting and compilation for C and Python, respectively (Figure 1).

Same-Origin Policy Protection

The functionality of this project is dependent on the use of web workers for the virtual machine. A web worker is JavaScript code which runs in the background, separate from the "main" JavaScript. This is used to load different components of the webpage asynchronously, which becomes useful when high-overhead components may stall the entire page load. The VM is an example of a high-overhead web component and as such it utilizes web workers to load it independently from the rest of the page components. One concern when working with web workers is the Same-Origin policy. Same-Origin protections are implemented in modern browsers and prevent websites from being able to access sensitive data on other websites. This is a major security measure and is a serious architectural concern for this project. Since the project uses web workers,



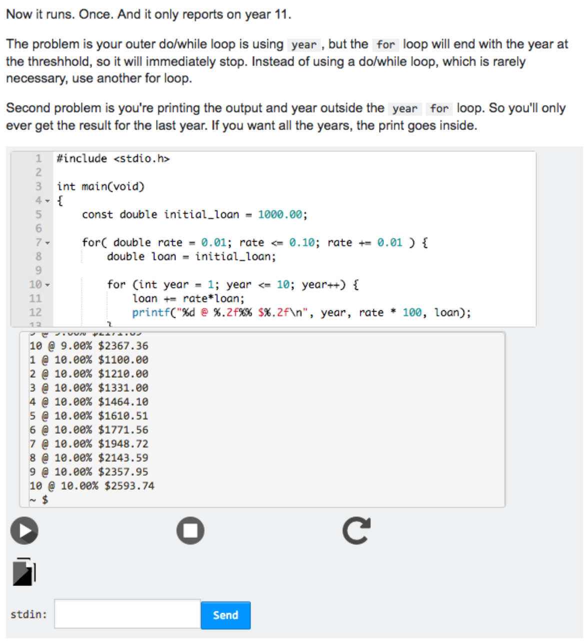
Intro to C

Every full C program begins inside a function called "main". A function is simply a collection of commands that do "something". The main function is always called when the program first executes. From main, we can call other functions, whether they be written by us or by others or use built-in language features. To access the standard functions that comes with your compiler, you need to include a header with the #include directive. What this does is effectively take everything in the header and paste it into your program. Let's look at a working program:

```
#include <stdio.h>

int main() {
  printf( "I am alive!  Beware.\n" );
  return 0;
}
```

```
Running program...
echo \#\#\#GCC_COMPILE\#\#\#;clear;pwd;rm -f main && clear && gcc -std=c99 m
ain.c -o main
###GCC_COMPILE###
/home/user
random: ccl urandom read with 78 bits of entropy available
~ $  echo GCC_EXIT_CODE: $?; echo \#\#\#GCC_COMPILE_FINISHED\#\#\#
GCC_EXIT_CODE: 0
###GCC_COMPILE_FINISHED###
~ $ clear && ./main
I am alive!  Beware.
~ $
```

Figure 3. The code snippet, after pressing "submit"



Now it runs. Once. And it only reports on year 11.

The problem is your outer do/while loop is using `year`, but the `for` loop will end with the year at the threshhold, so it will immediately stop. Instead of using a do/while loop, which is rarely necessary, use another for loop.

Second problem is you're printing the output and year outside the `year` `for` loop. So you'll only ever get the result for the last year. If you want all the years, the print goes inside.

```
#include <stdio.h>

int main(void)
{
    const double initial_loan = 1000.00;

    for( double rate = 0.01; rate <= 0.10; rate += 0.01 ) {
        double loan = initial_loan;

        for (int year = 1; year <= 10; year++) {
            loan += rate*loan;
            printf("%d @ %.2f%% $%.2f\n", year, rate * 100, loan);
```

```
10 @ 9.00% $2367.36
1 @ 10.00% $1100.00
2 @ 10.00% $1210.00
3 @ 10.00% $1331.00
4 @ 10.00% $1464.10
5 @ 10.00% $1610.51
6 @ 10.00% $1771.56
7 @ 10.00% $1948.72
8 @ 10.00% $2143.59
9 @ 10.00% $2357.95
10 @ 10.00% $2593.74
~ $
```

Figure 4. An example StackOverflow response[9] transformed using ELC. Link to original post: http://stackoverflow.com/questions/42318707/c-nested-loop-for-interest

the worker files have to be served remotely, as accessing local files from the browser would break the Same-Origin protocol.

Applications in Educational Websites

This project has a variety of applications in computer science education. As mentioned previously, websites like cprogramming.com, which teach programming concepts through static code snippets, will provide a dynamic environment for students to learn and play with code. In addition, this project allows students to have a safe environment to run code which they do not fully understand.

Besides websites for teaching, the project can be used in any environment that involves sharing code snippets. The popular website StackOverflow is one example. StackOverflow is a question-and-answer website where programmers can ask questions and receive answers from the computer science community. StackOverflow allows users to post questions and code snippets, to which they hope to receive a response and potentially an error-fixing code snippet. This interactive model would benefit from a project like this as it allows the community to run the code that the users are posting, thus providing a more in-depth understanding of the solutions (Figure 4).

DESIGN AND IMPLEMENTATION

Project Structure

The project is built from Linux-in-the-Browser (LIB)[7], an in-browser system programming environment developed by Lawrence Angrave and several students from the University of Illinois at Urbana-Champaign. Professor Angrave uses LIB to illustrate system programming concepts and the C programming language to students in the undergraduate system programming course. The full project consists of videos, exercises, and a full open-source textbook, detailing each of the covered concepts. The programming environment consists of a code editor, an in-browser Linux virtual machine, and a set of compiler controls allowing users to edit the compile command sent to the VM.

The code editor in the project is the Ace Editor[6], which is an open source, widely-used code editor for web applications. It is highly modularized and has a comprehensive set of configurations which allow users to modify features such as the syntax highlighting, the editing settings, and indentation.

Jor1k[11], the in-browser VM used, is an open-source project developed by Sebastian Macke. Jor1k is a Linux virtual machine, written in JavaScript, which implements the OpenRisc 1000 (Or1k) instruction set architecture. Jor1k optimizes for speed by minimally compiling installed packages, pre-loading relevant files, and operating asynchronously from the rest of the page.

This project is built using React.js[10] in the style of the Linux in the Browser modules. React is a front-end library designed by Facebook to help developers build “smart” user interfaces. It abstracts away the writing of long, static HTML pages to the design of small, functional components which contain HTML snippets. These components are rendered separately by React’s internal Document-Object Model (DOM) and can be updated and rendered independently from one another. This facilitates dynamic updating of large chunks of rendered HTML without page reloads. React projects are designed in terms of large components built up from smaller components. This project follows a similar model. React.js was chosen for this project due to its rising popularity and support from many organizations. In addition, the choice to use React allows developers to write code in a more functional manner and disregards the notion of global state. This paradigm results in developers understanding exactly where DOM information is flowing (called Reactive Data Flow) and thus debugging becomes simpler compared to debugging native JavaScript components.

There are two core React components to the project: the code editor and the terminal. In a rendered webpage using the project, there will be a single terminal with several code editors pointing to it. This significantly reduces page load time as opposed to each code editor interacting with individual terminals, as each terminal would render separately. The code editor, upon activating the “submit” button, takes the code from the Ace editor and saves it to Jor1k’s filesystem, and then sends the compile command to run within Jor1k. For example, if the user is writing C code, the editor will send a `gcc` command to Jor1k; if the user is writing Python, the editor will send a `python` command.

By including the required `<script>` tag in an HTML page, the project will search for tagged code snippets and transforms those snippets to runnable pieces of code, sending them to the VM upon compilation. The initial JavaScript code searches for these snippets, writes the code to the editor, and then links each editor to the global VM. The editor is built from the code snippet along with a compiler panel and a section for the program output.

Student Testing and Feedback

To measure how accessible the project is for students, we conducted feedback sessions with five computer science and electrical/computer engineering undergraduate students who are familiar with programming. In addition, we included students currently enrolled in Angrave’s System Programming course to see if familiarity with Linux in the Browser would translate across to this project.

The main issue reported by the surveyed students was that the buttons in the application were unintuitive. Initially, the names of the buttons were chosen based on their function as we saw it, which did not necessarily make sense to students. In addition, since the buttons all looked the same, it was difficult for students to distinguish between them. As a result, the buttons were updated to show their function based on more intuitive icons which are present in many other applictions.

The other issue concerned students not understanding the output component. Initially, there was a set of options in the box labeled “Program Output” for users to customize the output with. The main options were two radio-buttons, one labeled “Clear” and the other “Append”. The initial purpose of these buttons was to allow for clearing the output every time a program was run or to append to the existing output. These features were unclear to students, thus we removed the radiobuttons, instead adding a "Close output" x-button in the top-left corner to provide an intuitive way for users to close and clear the output.

NEXT STEPS

In the future, we will simplify integration of the project into existing webpages. The project requires two JavaScript files and an extensive filesystem; we would like to reduce this to one JavaScript file and potentially find a way to serve the filesystem remotely and access it from within the project, or utilize local browser assets.

Besides size, improvements to the Jor1k virtual machine allowing more language support would be productive as this would make the project useful to a larger number of users. This would involve installing components like the JVM onto the Jor1k platform, which may also result in larger file sizes and slower load times. Currently, we are investigating tools such as WebAssembly and researching to understand how we can decrease load times and other overhead for Jor1k or the other project components.

Finally, we would like to conduct more feedback sessions with students and other developers whose software interests would find this project useful. This, along with making the project open-source will allow many different voices to contribute to the project and provide different educational perspectives.

CONCLUSION

While many web applications seek to teach users how to program via static code snippets, our project provides an alternative which takes those static snippets and transforms them into live, runnable code, embedded in the webpage. This provides end users with a more comprehensive, interactive programming experience, which will ultimately result in more learning. We see applications of this project in many different environments: in traditional learning applications, question-and-answer applications, and web-based programming tutorials. By bringing a more interactive experience to the process, this project is a step forward in the technology behind online learning.

REFERENCES

1. Codecademy. `http://www.codecademy.com`
2. cprogramming.com - Learn C and C++ Programming. `http://www.cprogramming.com/`
3. JSFiddle. `https://jsfiddle.net/`
4. repl.it - Online REPL, Compiler, and IDE. `https://repl.it/`
5. StackOverflow. `http://www.stackoverflow.com`
6. ajax.org. Ace - The High Performance Code Editor for the Web. `https://ace.c9.io/`
7. Lawrence Angrave and Neelabh Gupta. Linux in the Browser. `http://cs-education.github.io/sys/#VM`
8. Aadil Bhatti. Embedded Live Code: Bringing Interactive Programming to Educational Applications. `https://github.com/cs-education/elc-dev`
9. dreed `https://stackoverflow.com/users/7282656/dreed`. C Nested Loop for Interest. StackOverflow. `https://stackoverflow.com/questions/42318707/c-nested-loop-for-interest`
10. Facebook. React.js: A declarative, efficient, and flexible JavaScript library for building user interfaces. `https://facebook.github.io/react/`
11. Sebastian Macke. jOR1K: Online OR1K Emulator Running Linux. `https://github.com/s-macke/jor1k`