Pat Letts

Senior project proposal

F17 CS 480

Therapist: a Brainf\* interpreter

**Language overview**

Brainf\* is a very simple esoteric language created by Urban Müller. Simple beyond the point of usefulness with only 8, single-character commands: <>-+,.[]; these commands allow for the navigation and changing of an array and its contents, the unofficial standard being at least 30,000 cells in size, all initialized to zero. ‘<’ and ‘>’ respectively decrement and increment a data pointer to track a program’s point in the array. ‘+’ and ‘–‘ respectively increment and decrement the value in the current cell. ‘,’ and ‘.’ allow for single-byte input and output of the current cell. ‘[‘ and ‘]’ allow for looping. All other characters should be ignored.

Here’s a Hello World:

++++++++[>++++[>++>+++>+++>+<<<]>+>+>->>+[<]]>>.>---.+++++++..+++.>>..<.+++.------.--------.>>+.>++.

I’d like to build a decent interpreter for Brainf\*. Given that it’s especially simple to implement, a good portion of my work is going to be focused around particular extra features.

**Challenges and considerations**

The first challenge that comes to mind for an utmost basic implentation of Brainf\* is looping, especially nested loops. My current plan is to use a stack to keep track of loops, storing the locations of left brackets for proper jumping.

There’s a lot of variation between implementations; how long should the array be? Urban Müller’s implementation was 30,000 cells. How many bits in a cell? Some support 8 bits, others would argue for 32. Signed or unsigned? Should I/O always be considered ASCII values instead of raw integers? And so on. Luckily, a number of common conventions have been established regarding these questions and more.

**Goals**

I plan for the majority of my code to use C++, though I may very well end up investing some code in a scripting language for tasks such as compilation and packaging. Naturally, I hope to learn something about interpreter design, and I’m also really interested in learning how to package software for say, a Linux repository, so it can be downloaded and setup directly through a package manager. In a more general sense, I also hope to become more familiar with the intracacies of C++, and other languages I may end up using.

I’d also like to implement some graphical features for the debugger, ideally with the GTK library. I’ve never worked with it before and it would be a nice chance to teach myself an entirely new library, and to show that I can in the first place.

A particular goal I’ve kept in mind is an object-oriented design. Considering the necessity of extra features bundled with my interpreter, without this, unnecessary complexity in my code would only increase, detracting from time better spent on other goals. If my base interpreter were implemented with this in mind, an interface could be provided with which I could build most, if not all, other features off of.

**Features**

**Basic interpreter** – Provide a source file to be interpreted as a Brainf\* program.

**Simple options** – Provide some command line options when starting the interpreter.

* Enable/disable ASCII IO
* Un/signed cell values
* Tape length

**Debug mode** – A particular feature that I haven’t seen much of would be a graphical debug mode, displaying the states of particular elements of the machine, and stepping through code, allowing the user to watch the machine change, one command at a time.

My goal for this is to display particular elements of the program, including the main array, souce code, and the loop stack, with graphical emphasis placed on these sections to indicate the current state of the machine. After this, my plan is to allow the user to step through the execution of a program, moving back and forth through source code one operation at a time.

I’d like to provide two forms of this debugger. One based entirely in the terminal, simply offering the basic features described above. The second would be a full GUI made with the GTK library and provide more variety in usage, such as manually editing the contents of the tape and a more elaborate interface for stepping through code.

**REPL** – I’d also like to work on a simple REPL shell. Considering the nature of Brainf\*, a quick testing environment would be nice. A basic setup would be command line based, creating a persistent BF machine that can repeatedly accept and evaluate input until the user manually quits. Ideally, it would also show and update a small region of the tape and allow for input from a file.

**Grading**

* **Interpreter – 10 points**
  + Basic interpreter – 5
  + Object-oriented design – 3
  + Test cases – 2
* **Debuggers** – **10 points**
  + Curses library – 4 points
    - Basic stepping through input code – 2 points
    - Display and update BF code, tape, and stack – 2 points
  + GTK library – 6 points
    - Basic stepping through input code – 2 points
    - Directly manipulate cells on the tape – 1 point
    - Interface for more powerful stepping, allowing for breakpoints – 3 points
* **REPL – 10 points**
  + Repeatedly accept and evaluate BF code from the keyboard – 3 points
  + Show tape and update as necessary – 2 points
  + Accept multiple lines before evaluating when a loop is opened – 2 points
  + Allow user to define, name, and call procedures from sections of BF code – 3 points
    - Abstraction!
* **Automatic compilation and installation** – **6 points**
  + Provide a Makefile with interpreter source code – 2 points
  + Provide an entire package for installation, package manager TBD – 4 points
* **Man page – 4 points**
  + Provide a detailed man page describing the features and usage of the interpreter – 4 points
* **Command line flags – 10 points**
  + Toggle ASCII I/O – 4 points
  + Toggle un/signed cells – 4 points
  + Define tape length – 2 points

**Total – 50 points**

**Scale**

A – 44-50 points

B – 37-43 points

C – 29-36 points

D – 0-28 points

Project maintained at <https://github.com/herpeslurpy/Therapist>