Project 3

by Michael Clausen, Luke Eltiste, and Timothy Wiratmo

Our Esolang

The esolang we chose is Brainfuck, created by Urban Müller. It is similar to a Turing Machine, in that it has a “tape” as an array of memory cells and one can only operate on the cell that the data pointer is pointing to. Müller’s goal was to create a language to write the smallest possible compiler for the Amiga 2.0 operating system in [1].

Brainfuck has eight commands to manipulate the data pointer and the memory cell it points to (quoted directly from [1]):

- > Move the pointer to the right.
- < Move the pointer to the left.
- + Increment the memory cell at the pointer.
- - Decrement the memory cell at the pointer.
- . Output the character signified by the cell at the pointer.
- , Input a character and store it in the cell at the pointer.
- [Jump past the matching] if the cell at the pointer is 0.
-] Jump back to the matching [if the cell at the pointer is nonzero.

A brainfuck program is created using a sequence of these commands. All other characters are ignored and treated as comments.

The memory array tape is one dimensional and consists of 30,000 byte memory blocks initialized with zeros. The data pointer initially points to the leftmost byte of the tape. Brainfuck uses two bit streams for the input and output for the program. Inputs are treated as ASCII chars [1].

Brainfuck is Turing Complete. This is proven here http://brainfuck.org/utm.b by Daniel B Christofani. The proof involves “simulating a Turing complete set of tag systems,” a type of computational model developed by Emil Post [2]. These tag systems can construct “small universal (Turing complete) systems for several different classes of computational systems, including Turing machines” [3].

The Regular Language

We chose to use the regular language L = {w: w contains at least three 1s}. The following Brainfuck program recognizes this language.

Implementation

With the language we chose we basically just need to count the ones, so the program will first count the ones then check if the counter is greater than or equal to three.

Program Code

```
>>            Cell 0 is used as a counter
              Cell 2 is used for main loop
+             Increment current cell to start loop
[
  <
  ,           Get input
  ----------  Subtract 48 to get decimal value from ASCII
  ----------
  ----------
  ----------
  --------
  [           If input is not a zero
    <+>       Increment counter
    -         Decrement the cell
    [         The input was a two which means we reached the end of the input
      <->     Decrement the counter because we don't want to count the two
      >-<     Set Cell 2 to zero so the main loop exits
      -
    ]
  ]
  >
]

<<            Return to cell 0
[>>>+<<<-]    Move value in cell 0 to cell 3
>+>>>+++<     Initialize cells for comparison code
              a = cell 3 b = cell 4
+>+<          This is for managing if a=0 and b=0
[->-[>]<<]    Subtract from both until one reaches zero
<[-           BLOCK (a ge b)
  >>>>
  ----[----->+<]>++.+[-->+++<]>..++.++++++++++.++++.+++[->+++<]>.-.
  [<]
]
<[-<          BLOCK(a lt b)
  >>>>>
  -[--->+<]>---.+[----->+++<]>.+++++.-----.--.-[--->+<]>--.+++[->+++<]>.-.
  [<]
]
```

Code Explained

Initializing the tape

```
>>            Cell 0 is used as a counter
              Cell 2 is used for main loop
+             Increment current cell to start loop
```

Cell #0 is where we'll store the count of 1's and Cell #2 will contain the value for the input loop to keep going.

Current Tape

0. Counter	1.	2. Keep running loop (1)

Next we start reading the input with the following loop

```
[
  <
  ,           Get input
  ----------  Subtract 48 to get decimal value from ASCII
  ----------
  ----------
  ----------
  --------
  [           If input is not a zero
    <+>       Increment counter
    -         Decrement the cell
    [         The input was a two which means we reached the end of the input
      <->     Decrement the counter because we don't want to count the two
      >-<     Set Cell 2 to zero so the main loop exits
      -
    ]
  ]
  >
]
```

At the beginning of the loop we first start by moving to Cell #1 and inputting a character.

```
  <
  ,           Get input
  ----------  Subtract 48 to get decimal value from ASCII
  ----------
  ----------
  ----------
  --------
```

Then we subtract 48 from it because the input is in ASCII.

```
[           If input is not a zero
    <+>       Increment counter
    -         Decrement the cell
    [         The input was a two which means we reached the end of the input
      <->     Decrement the counter because we don't want to count the two
      >-<     Set Cell 2 to zero so the main loop exits
      -
    ]
  ]
```

If the input was not a zero we enter the loop, then we go to cell #0 and increment, then we return to cell #1 and decrement our current value to exit the loop if it was a 1. If it was a 2, which represents the end of the input, we'll enter the next loop. This first decrements the counter because it was not a 1. Then, it goes to cell #2 and sets it back to zero to exit the input loop.

```
>
```

Lastly we shift back to cell #2 for the input loop to check if it should continue

```
<<            Return to cell 0
[>>>+<<<-]    Move value in cell 0 to cell 3
```

Now that the input has finished parsing we go to cell #0 where the counter is stored. Then, we move it over to cell #3 to check if the counter is greater than or equal to 3.

```
>+>>>+++<     Initialize cells for comparison code
              a = cell 3 b = cell 4
+>+<          This is for managing if a=0 and b=0
```

Now we initialize the tape for the inequality check. We increment 'a' and 'b' values by 1 in order for the looping mechanism to work properly. The tape now looks like:

0	1	0	Count + 1	4	0

```
[->-[>]<<]    Subtract from both until one reaches zero
```

This loop will decrement both values by 1. If the second value is not zero, we move one cell to the right. Then move left two cells back to cell #3 and continue the loop. If cell #3 reaches zero first we end up in cell #3. If cell #4 reaches zero first, or if they both hit zero at the same time, we end up in cell #2.

```
<[-           BLOCK (a ge b)
  >>>>
  ----[----->+<]>++.+[-->+++<]>..++.++++++++++.++++.+++[->+++<]>.-.
  [<]
]
<[-<          BLOCK(a lt b)
  >>>>>
  -[--->+<]>---.+[----->+++<]>.+++++.------.--.-[--->+<]>--.+++[->+++<]>.-.
  [<]
]
```

Then, if the counter was greater than or equal to three, we enter the first block. This prints out “Accepted.” Otherwise, we end up in the second block, which prints out “Rejected.”

Usage

Our program can be found here using the El Brainfuck online Brainfuck interpreter. To run the program, enter the input at the bottom of the page as a string of 0’s and 1’s and put a 2 at the end to mark the end of input. Another web interpreter you can use can be found here. Running the code is the exact same as the other website but you can enter a debug mode that lets you step through each instruction. You can set breakpoints with the ‘#’ character. It is significantly slower, however.

Bibliography

[1] brainfuck. 2023. Esolang, the esoteric programming languages wiki. Retrieved from https://esolangs.org/wiki/Brainfuck.

[2] Cristofani, D.B. (2005) *Brainfuck*, *brainfuck utm*. Retrieved from at: http://www.brainfuck.org/utm.b.

[3] Liesbeth De Mol. 2007. Tag systems and Collatz-like functions. *Theoretical Computer Science* 390, 1 (2008), 92-101. DOI: https://doi.org/10.1016/j.tcs.2007.10.020