Esoteric Programming

Prepared by Mark on January 19, 2025

Part 1: Turing

Definition 1:
An *esoteric programming langauge* is a programming langauge made for fun.
We'll work with two such languages today: *Turing* and *Befunge*.

Definition 2:
Turing is one of the most famous esoteric langauges, and is extremely minimalist.
It consists only of eight symbols, a data pointer, and an instruction pointer.
Turing's eight symbols are as follows:

- `>` : move the instruction pointer right
- `<` : move the instruction pointer left
- `+` : increment the current memory cell
- `-` : decrement the current memory cell
- `.` : output the value of the current cell as a character
- `,` : consume one character from input and store it at the current cell
- `[` : Jump to the matching `]` if the current cell is zero.
  otherwise, execute the next instruction.
- `]` : Jump back to the matching `[` if the current cell is not zero.
  otherwise, execute the next instruction.
- All other characters are ignored.

Problem 3:
Go to `langs.betalupi.com` and open the Turing editor.
Clear the pre-set program in the left side of the screen and replace it with the following:

```
>+>++>+++>++++>+++++[[-]<]
```

- What does this program do?
- What does the snippet `[-]` do?
- Why is the first cell left as `0`?

Remark 4:
Run the program `+[+]`. Notice that the cell's value *wraps* from `127` to `-128`!

Note that `[+]` has the same effect as `[-]`.

Problem 5:
Write a program that moves the value of the current cell three cells to the right.

Problem 6:
Write a program that **copies** the value of the current cell into the next cell.

Problem 7:
Write a program that adds the value of the first cell and the second cell, leaving the result in the second cell.

Problem 8:
Solve Problem 7 again, but multiply the two cells instead of adding.

Problem 9:
Write a program that computes the square of the current cell.

Definition 10:
Turing uses ASCII to map numbers to characters.
You may use `cs.cmu.edu/ pattis/15-1XX/common/handouts/ascii.html` for reference.

Problem 11:
Write a program that prints the letter `a`.

Problem 12:
Write a program that prints the alphabet as shown below:

```
abcdefghijklmnopqrstuvwxyz
```

Problem 13:
Modify your program so that it prints the alphabet with alternating case:

```
aBcDeFgHiJkLmNoPqRsTuVwXyZ
```

Problem 14:
Write a program that repeats the user's input exactly as it is entered.

Problem 15:
Write a program that repeats the user's input in reverse
Hint: You may use as many memory cells as you need—the editor will add more as you use them.

Problem 16:
Write a program that prints "Hello World"

Problem 17:
Write a program that finds the first memory cell that is non-zero.

Problem 18:
Write a program that initializes the tape with the following sequence:

```
3 6 12 24 48 96 192 128 0
```

Problem 19:
Write a program that decodes run-length encoding. That is, it turns input like

```
a3j4k2d5
```

into the decoded output

```
aaajjjjkkddddd
```

Problem 20: Bonus
Write a program that prints a Turing program that prints the original program's input.

Problem 21: Bonus
Write a program that outputs all squares between 0 and 400

Problem 22: Bonus
Write a program that prints the Fibonacci numbers

Part 2: Befunge

Definition 23:
Befunge is another esoteric programming langauge, designed to be very difficult to compile.
It consists of a "field" of instructions, a two-dimensional program counter, and a stack of values.

The program counter starts in the top-left corner of the field, moving right.
It executes any instruction it encounters. The instructions `>`, `<`,`^`, and `v` can be used to direct the program counter, and `_` and `|` are used for control flow.

An instruction reference is below:

- `+` Addition: Pop two values a and b, then push the result of $a + b$
- `-` Subtraction: Pop two values a and b, then push the result of $b - a$
- `*` Multiplication: Pop two values a and b, then push the result of $a \times b$
- `/` Integer division: Pop two values a and b, then push the result of $b \div a$, rounded down.
- `%` Modulo: Pop two values a and b, then push the remainder of the integer division of $b \div a$.
- `!` Logical NOT: Pop a value. If the value is zero, push 1; otherwise, push zero.
- `` ` `` Greater than: Pop two values a and b, then push 1 if $b > a$, otherwise zero.
- `>` Program counter direction right
- `<` Program counter direction left
- `^` Program counter direction up
- `v` Program counter direction down
- `?` Random program counter direction
- `_` Horizontal `if`: pop a value; set direction to right if value=0, set to left otherwise
- `|` Vertical `if`: pop a value; set direction to down if value=0, set to up otherwise
- `"` Toggle string mode (push each character's ASCII value all the way up to the next ")
- `:` Duplicate top stack value
- `\` Swap top stack values
- `$` Pop top of stack and discard
- `.` Pop top of stack and output as integer
- `,` Pop top of stack and output as ASCII character
- `#` Bridge: jump over next command in the current direction of the current PC
- `g` A "get" call (a way to retrieve data in storage). Pop two values y and x, then push the ASCII value of the character at that position in the program. If (x,y) is out of bounds, push 0
- `p` A "put" call (a way to store a value for later use). Pop three values y, x and v, then change the character at the position (x,y) in the program to the character with ASCII value v
- `&` Get integer from user and push it
- `~` Get character from user and push it
- `@` End program
- `0-9` Push corresponding number onto the stack

Note that the `p` instruction allows us to write self-modifying code.

Problem 24:
Write a program that prints "`Hello World`".

Problem 25:
Write a program that prints the alphabet

Problem 26:
Write a program that generates a random sequence of numbers (`0-9`).

Problem 27:
Write a program that does not contain the string "`Hello World`",
but writes that string somewhere inside its source.

Problem 28:
Replace the `x`s in the following program so that the loop runs forever.
Do not use any control-flow instructions (`>`, `<`, `^`, `v`, `_`, `|`, `#`, or `?`)
Hint: Start by replacing all the `x`s with spaces.
You may not need all the `x`s, feel free to use a smaller rectangle.

```
>xxxxxxxxv
x        x
x        x
x        x
x        x
x        x
x        x
x        x
^xxxxxxxx@
```

Problem 29: Bonus
Write a quine. (i.e, write a program that outputs itself)