Artist: Towel

Challenge: Ask Your Grandpa Again!

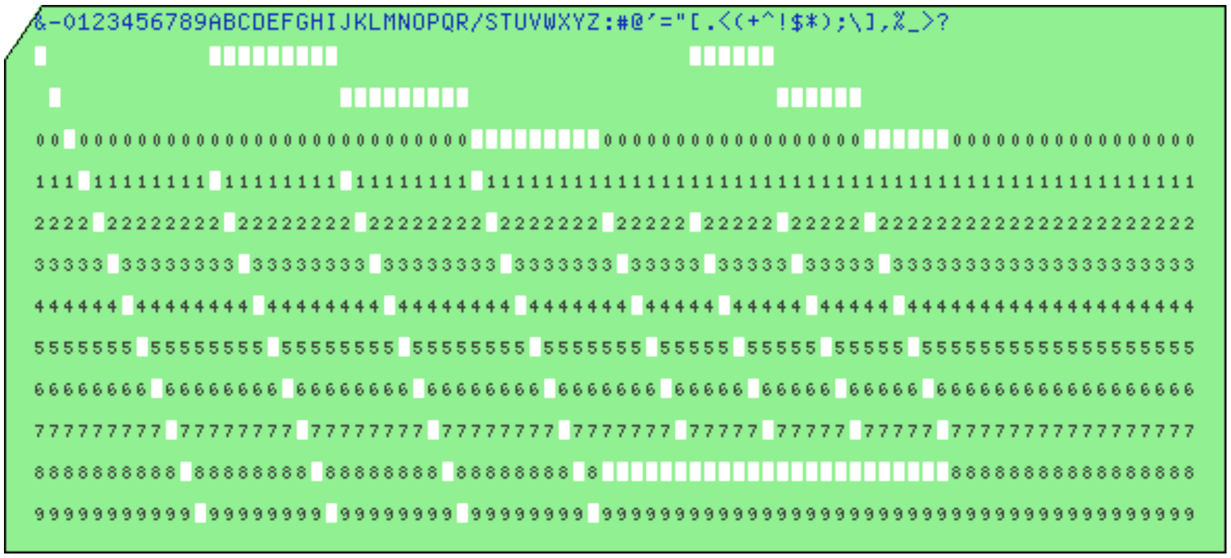
We are again given punch cards, but this time we have more than one! Not only that, but "oh noes!" They are out of order! This was a common problem back in the day when punchcards were actually used, and it is simulated here to a smaller degree.

The Program

I used my program to convert the images, so here is my program. It works by first taking the image and reading every pixel converting the ones that aren't necessary to white leaving a clean image behind. Note that this isn't saved anywhere and is just for the programs reference.

After some testing I figured out that each "block" is roughly **26** pixels wide and **75** pixels high. Not being very picky with this program I made it then scan up-down, left to right jumping **52** pixels right and **150** up and down, skipping the top **170** pixels of each column.

Then using this map:


&-0123456789ABCDEFGHIJKLMNOPQR/STUVWXYZ:#@'="[.<(+^!$*);\],%_>?

it translates each column to a letter. The matrix ends up being *86* columns and *12* rows, which could be better but is good enough for CTF :D

Dirty Python Source Code:

```
#!/usr/bin/python
```

```
from PIL import Image
import sys
import argparse

def decode(code):
    if code == '100100000000': return 'A'
    if code == '100010000000': return 'B'
    if code == '100001000000': return 'C'
    if code == '100000100000': return 'D'
    if code == '100000010000': return 'E'
    if code == '100000001000': return 'F'
    if code == '100000000100': return 'G'
    if code == '100000000010': return 'H'
    if code == '100000000001': return 'I'
    if code == '010100000000': return 'J'
    if code == '010010000000': return 'K'
    if code == '010001000000': return 'L'
    if code == '010000100000': return 'M'
    if code == '010000010000': return 'N'
    if code == '010000001000': return 'O'
    if code == '010000000100': return 'P'
    if code == '010000000010': return 'Q'
    if code == '010000000001': return 'R'
    if code == '001100000000': return '/'
    if code == '001010000000': return 'S'
    if code == '001001000000': return 'T'
    if code == '001000100000': return 'U'
    if code == '001000010000': return 'V'
    if code == '001000001000': return 'W'
    if code == '001000000100': return 'X'
    if code == '001000000010': return 'Y'
    if code == '001000000001': return 'Z'
    if code == '100000000000': return '&'
    if code == '010000000000': return '-'
    if code == '001000000000': return '0'
    if code == '000100000000': return '1'
    if code == '000010000000': return '2'
    if code == '000001000000': return '3'
    if code == '000000100000': return '4'
    if code == '000000010000': return '5'
    if code == '000000001000': return '6'
    if code == '000000000100': return '7'
    if code == '000000000010': return '8'
    if code == '000000000001': return '9'
    if code == '000010000010': return ':'
    if code == '000001000010': return '#'
    if code == '000000100010': return '@'
    if code == '000000010010': return "'"
    if code == '000000001010': return '='
    if code == '000000000110': return '"'
    if code == '100010000010': return '['
    if code == '100001000010': return '.'
    if code == '100000100010': return '<'
    if code == '100000010010': return "("
    if code == '100000001010': return '+'
    if code == '100000000110': return '^'
    if code == '010010000010': return '!'
    if code == '010001000010': return '$'
    if code == '010000100010': return '*'
    if code == '010000010010': return ")"
    if code == '010000001010': return ';'
    if code == '010000000110': return '\\'
    if code == '001010000010': return ']'
    if code == '001001000010': return ','
    if code == '001000100010': return '%'
    if code == '001000010010': return "_"
    if code == '001000001010': return '>'
    if code == '001000000110': return '?'
    return ' '

#################
# Get the image #
#################
parser = argparse.ArgumentParser()
parser.add_argument("image")
args = parser.parse_args()

#######################
# Make image readable #
#######################
picture = Image.open(args.image, 'r').convert('RGB')
width, height = picture.size
for x in range(0, width):
    for y in range(0, height):
        current_color = picture.getpixel((x,y))
        if current_color >= (50,50,50):
            picture.putpixel((x,y), (255, 255, 255))
        else:
            picture.putpixel((x,y), (0,0,0))

################################
# Scan and translate the image #
################################
flag = c = ''
for x in range(0, width, 52): #The horizontal pixels
    for y in range(170, height, 150): #The vertical pixels
        current_color = picture.getpixel((x,y))
        if current_color == (0,0,0):
            c += '1'
            picture.putpixel((x,y), (255, 0, 0))
        else:
            c += '0'
    flag += str(decode(c))
    c = ''

print flag
```

Decoding the Output

When running the program on the files, I got the following results:

```
1337 FORMAT(11HFLAG-DFEB0D,I4,1H-,I3,10HFDBECDF39D,I3)   13370050
END                                                      13370060
WRITE(6,1337)J+29,(J/4)+20,I                             13370040
PROGRAM WFLAG                                            13370010
I=931                                                    13370020
J=2800                                                   13370030
```

It's out of order programming lines in what appears to be Fortran. Lucking for us on the right there appears to be the proper sequence for them. Re-ordering them:

```
PROGRAM WFLAG                                               13370010
    I=931                                               13370020
    J=2800                                              13370030
    WRITE(6,1337)J+29,(J/4)+20,I                        13370040
    1337 FORMAT(11HFLAG-DFEB0D,I4,1H-,I3,10HFDBECDF39D,I3)  13370050
END                                                         13370060
```

Ahh! That looks better. Now if you don't know ye old fortran you can use an online compiler/interpreter and get the output of the program:

> *FLAG-DFEB0D2829-720FDBECDF39D931*

```
                    ~~Happy Hacking~~

                          Towel
```