RingZer0 Team Online CTF

A solution for the challenge 172

Ask your grandpa!

Category

Coding Challenges

By

garigouster

For this challenge, we have a scan:

This is clearly an obsolete punched card for old computers. We're easily guessing that we must decode data on this card.

First, we must understand how data was encoded on punched cards. A simple search on Internet allows us to find explanations more or less detailed. I give some references in annexes of this document.

To sum it all up:

- Each column of cards specifies a character ; for this 80-column card, so this gives a 80-character text.
- The holes on rows 0, 11 and 12 are used to select a range of characters ; for example, one hole at row 12 selects a range for characters A to I.
- One hole on one of the lines 1-9 is then used to select a character in range ; for example a hole at row 12 and a hole at row 1 selects A.
- It may not be holes on the lines 1-9, which is used to indicate additional characters according to holes on lines 0, 11 and 12 ; for example no hole on all lines selects space character.

In fact, it's a little more complicated than that: on lines 1-9, there may be also two or three holes:

- one hole at line 9 and one hole on one of the lines 1 to 8,
- or one hole at line 8 and one hole on one of the lines 1 to 7,
- or two holes at lines 8 and 9 and one hole on one of the lines 1 to 7.

This is used to select additional ranges of characters and thus cover all of a set of 256 characters.

As these cards were used for old computers, the character encoding used was EBCDIC. But we can easily transcribe it to ASCII.

The way characters are encoded on the cards (ranges of characters per column) and number of columns on cards depends on the drive model.

But for standard printable characters, it does not really make any difference. We can work on standard IBM card drivers to decode these data.

Well. We can manually decode these data on punched cards. But I recall that this is a coding challenge!

As there is another similar challenge, I decide to do a small Perl module to completely decode any data on punched cards (certainly, for IBM standard cards).

To specify (holes on) cards, I choose to use a structure that I consider the most simple: for each line of card, column numbers with a hole are specified.

Once the module is written, the main program for decoding data is minimalist:

```
$ cat decode_card
#!/usr/bin/env perl

use PunchedCard;

my %card = (
    12 => [1,3,4,6,7,9,11,12,21,24,26,27,29,36,37,38,43,44,45,46,47,48,52,53],
    11 => [2,5,8,10,22,25,28,39],
    0  => [13,17,19,20,34],
    1  => [3,18,21,26,35,42,43,48,52,55],
    2  => [7,14,23,29,38,44,50],
    3  => [2,15,19,22,25,32,36,41,53,54],
    4  => [6,11,30,37,45,46],
    5  => [10,12,20],
    6  => [1,24,47,49],
    7  => [4,13,27],
    8  => [14,16,19,23,33,40,51],
    9  => [9,31],
);

my @data = PunchedCard->new(Card => \%card,Encoding => ASCII)->Decode;
print 'Card data: ',join('',map {chr} @data),"\n";
$ ./decode_card
Card data: FLAG-DB-INDEX:3801,VAL:FLAG-.....
```

Annex 1: References

- **quadibloc: The Punched Card**
  - http://www.quadibloc.com/comp/cardint.htm
- **Wikipedia: Punched Card**
  - https://en.wikipedia.org/wiki/Punched_card
- **The University of Iowa: Punched Card Codes**
  - http://homepage.cs.uiowa.edu/~jones/cards/codes.html
- **KLOTH.NET: Cardpunch: punch a punched card**
  - http://www.kloth.net/services/cardpunch.php

Annex 2: The PunchedCard.pm module for grandpa challenges

```
################################################################################
# PunchedCard.pm: Decode punched cards
################################################################################
#
# Copyright (c) 2015 garigouster
# Contact: garigouster on RingZer0 Team
#
# This program is free software: you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the Free
# Software Foundation, either version 3 of the License, or (at your option)
# any later version.
#
# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for
# more details.
#
# You should have received a copy of the GNU General Public License along with
# this program. If not, see <http://www.gnu.org/licenses/>.
#
################################################################################

use strict;
use warnings qw(all);

package PunchedCard;

use constant {
    IBM_DEFAULT => 'IBM_DEFAULT',
};

use constant {
    EBCDIC => 'EBCDIC',
    ASCII => 'ASCII',
};

BEGIN {
    require Exporter;
    use vars qw($VERSION @ISA @EXPORT @EXPORT_OK %EXPORT_TAGS);
    $VERSION = '0.1';
    @ISA = qw(Exporter);
    my @cnsts = (IBM_DEFAULT,EBCDIC,ASCII);
    @EXPORT = (@cnsts);
    @EXPORT_OK = (@EXPORT);
    %EXPORT_TAGS = (CONSTANTS => [@cnsts]);
}

my @TYPES = (
    IBM_DEFAULT,
);

my @ENCODING = (
    EBCDIC,
    ASCII,
);

my %CARD_COLUMNS = (
    IBM_DEFAULT => 80,
);

my %CARD_ENCODING = (
    IBM_DEFAULT => EBCDIC,
);

### Source: http://www.quadibloc.com/comp/cardint.htm
my %IBM_DEFAULT_EBCDIC = (
    '001' => [0x50,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9],
    '010' => [0x60,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,0xD8,0xD9],
    '100' => [0xF0,0x61,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7,0xE8,0xE9],
    '000' => [0x40,0xF1,0xF2,0xF3,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9],
    '101' => [0xC0,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89],
    '011' => [0x6A,0x91,0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99],
    '110' => [0xD0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9],
    '111' => [0x70,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9],

    '00110' => [undef,0x49,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F],
    '01010' => [undef,0x59,0x5A,0x5B,0x5C,0x5D,0x5E,0x5F],
    '10010' => [undef,0x69,0xE0,0x6B,0x6C,0x6D,0x6E,0x6F],
    '00010' => [undef,0x79,0x7A,0x7B,0x7C,0x7D,0x7E,0x7F],
```

```
    '10110' => [undef,0x80,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F],
    '01110' => [undef,0x90,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F],
    '11010' => [undef,0xA0,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF],
    '11110' => [undef,0xB0,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF],

    '0011' => [undef,0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08],
    '0101' => [undef,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18],
    '1001' => [undef,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28],
    '0001' => [undef,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38],
    '1011' => [undef,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48],
    '0111' => [undef,0x51,0x52,0x53,0x54,0x55,0x56,0x57,0x58],
    '1101' => [undef,0xE1,0x62,0x63,0x64,0x65,0x66,0x67,0x68],
    '1111' => [undef,0x71,0x72,0x73,0x74,0x75,0x76,0x77,0x78],

    '00111' => [undef,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F],
    '01011' => [undef,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F],
    '10011' => [undef,0x29,0x2A,0x2B,0x2C,0x2D,0x2E,0x2F],
    '00011' => [undef,0x39,0x3A,0x3B,0x3C,0x3D,0x3E,0x3F],
    '10111' => [undef,0x00,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF],
    '01111' => [undef,0x10,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF],
    '11011' => [undef,0x20,0xEA,0xEB,0xEC,0xED,0xEE,0xEF],
    '11111' => [undef,0x30,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF],
);

### Source: http://www.quadibloc.com/comp/cardint.htm
my %IBM_DEFAULT_ASCII = (
    '001' => [0x26,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49],
    '010' => [0x2D,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F,0x50,0x51,0x52],
    '100' => [0x30,0x2F,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5A],
    '000' => [0x20,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39],
    '101' => [0x7B,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69],
    '011' => [0x7C,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70,0x71,0x72],
    '110' => [0x7D,0x7E,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7A],
    '111' => [0xBA,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF,0xE0,0xE1],

    '00110' => [undef,0xA8,0x5B,0x2E,0x3C,0x28,0x2B,0x21],
    '01010' => [undef,0xB1,0x5D,0x24,0x2A,0x29,0x3B,0x5E],
    '10010' => [undef,0xB9,0x5C,0x2C,0x25,0x5F,0x3E,0x3F],
    '00010' => [undef,0x60,0x3A,0x23,0x40,0x27,0x3D,0x22],
    '10110' => [undef,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9],
    '01110' => [undef,0xC1,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0],
    '11010' => [undef,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7],
    '11110' => [undef,0xD8,0xE2,0xE3,0xE4,0xE5,0xE6,0xE7],

    '0011' => [undef,0x01,0x02,0x03,0x9C,0x09,0x86,0x7F,0x97],
    '0101' => [undef,0x11,0x12,0x13,0x9D,0x85,0x08,0x87,0x18],
    '1001' => [undef,0x81,0x82,0x83,0x84,0x0A,0x17,0x1B,0x88],
    '0001' => [undef,0x91,0x16,0x93,0x94,0x95,0x96,0x04,0x98],
    '1011' => [undef,0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7],
    '0111' => [undef,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0],
    '1101' => [undef,0x9F,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8],
    '1111' => [undef,0xBB,0xBC,0xBD,0xBE,0xBF,0xC0,0xC1,0xC2],

    '00111' => [undef,0x8D,0x8E,0x0B,0x0C,0x0D,0x0E,0x0F],
    '01011' => [undef,0x19,0x92,0x8F,0x1C,0x1D,0x1E,0x1F],
    '10011' => [undef,0x89,0x8A,0x8B,0x8C,0x05,0x06,0x07],
    '00011' => [undef,0x99,0x9A,0x9B,0x14,0x15,0x9E,0x1],
    '10111' => [undef,0x00,0xE8,0xE9,0xEA,0xEB,0xEC,0xED],
    '01111' => [undef,0x10,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3],
    '11011' => [undef,0x80,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9],
    '11111' => [undef,0x90,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF],
);

### Source: https://support.microsoft.com/en-us/kb/216399/fr
my @EBCDIC2ASCII = (
    0x00,0x01,0x02,0x03,0x9C,0x09,0x86,0x7F,0x97,0x8D,0x8E,0x0B,0x0C,0x0D,0x0E,0x0F,
    0x10,0x11,0x12,0x13,0x9D,0x85,0x08,0x87,0x18,0x19,0x92,0x8F,0x1C,0x1D,0x1E,0x1F,
    0x80,0x81,0x82,0x83,0x84,0x0A,0x17,0x1B,0x88,0x89,0x8A,0x8B,0x8C,0x05,0x06,0x07,
    0x90,0x91,0x16,0x93,0x94,0x95,0x96,0x04,0x98,0x99,0x9A,0x9B,0x14,0x15,0x9E,0x1A,
    0x20,0xA0,0xA1,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xD5,0x2E,0x3C,0x28,0x2B,0x7C,
    0x26,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0x21,0x24,0x2A,0x29,0x3B,0x5E,
    0x2D,0x2F,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xE5,0x2C,0x25,0x5F,0x3E,0x3F,
    0xBA,0xBB,0xBC,0xBD,0xBE,0xBF,0xC0,0xC1,0xC2,0x60,0x3A,0x23,0x40,0x27,0x3D,0x22,
    0xC3,0x61,0x62,0x63,0x64,0x65,0x66,0x67,0x68,0x69,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,
    0xCA,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70,0x71,0x72,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,
    0xD1,0x7E,0x73,0x74,0x75,0x76,0x77,0x78,0x79,0x7A,0xD2,0xD3,0xD4,0x5B,0xD6,0xD7,
    0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF,0xE0,0xE1,0xE2,0xE3,0xE4,0x5D,0xE6,0xE7,
    0x7B,0x41,0x42,0x43,0x44,0x45,0x46,0x47,0x48,0x49,0xE8,0xE9,0xEA,0xEB,0xEC,0xED,
    0x7D,0x4A,0x4B,0x4C,0x4D,0x4E,0x4F,0x50,0x51,0x52,0xEE,0xEF,0xF0,0xF1,0xF2,0xF3,
    0x5C,0x9F,0x53,0x54,0x55,0x56,0x57,0x58,0x59,0x5A,0xF4,0xF5,0xF6,0xF7,0xF8,0xF9,
    0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF,
);
```

```
my %CARDS_FORMATS = (
    IBM_DEFAULT => {
        EBCDIC => \%IBM_DEFAULT_EBCDIC,
        ASCII => \%IBM_DEFAULT_ASCII,
    },
);

sub new($;%) {
    my ($class,%opts) = @_;

    my $type = $opts{Type};
    $type = IBM_DEFAULT if !defined $type;
    die __PACKAGE__,'::new: Bad parameter for type of punched card.',"\n" if length ref $type;
    die 'Unknown type of punched card: ',$type,"\n" if !grep {$_ eq $type} @TYPES;

    my $encoding = $opts{Encoding};
    $encoding = $CARD_ENCODING{$type} if !defined $encoding;
    die __PACKAGE__,'::new: Bad parameter for encoding of punched card.',"\n" if length ref $encoding;
    die 'Unknown encoding of punched card: ',$encoding,"\n"
        if !exists $CARDS_FORMATS{$type}{$encoding};

    my $this = bless {},$class;

    $this->{Type} = $type;
    $this->{Columns} = $CARD_COLUMNS{$type};
    $this->{Encoding} = $encoding;
    $this->{Format} = $CARDS_FORMATS{$type}{$encoding};
    $this->SetCard($opts{Card}) if exists $opts{Card};

    $this;
}

sub SetCard($$) {
    my ($this,$data) = @_;

    die __PACKAGE__,'::SetCard: Bad punched card.',"\n" if ref $data ne 'HASH';

    my $card = [map {[(map {0} 0..9),undef,(map {0} 11,12)]} 1..$this->{Columns}];
    for my $l (keys %$data) {
        die 'Bad lines for punched card.',"\n"
            if !defined $l || length ref $l || $l !~ /\A\d|11|12\z/ || ref $data->{$l} ne 'ARRAY';
        for my $c (@{$data->{$l}}) {
            die 'Bad columns (line: ',$l,') for punched card.',"\n" if !defined $c || length ref $c
                || $c !~ /\A[0-9]+\z/ || $c < 1 || $c > $this->{Columns};
            $card->[$c-1][$l] = 1;
        }
    }
    $this->{Card} = $card;

    $this;
}

sub Decode($;$) {
    my ($this,$encoding) = shift;

    die 'Punched card not set.',"\n" if !exists $this->{Card};

    my $format;
    if (!defined $encoding) {
        $format = $this->{Format};
    } else {
        die __PACKAGE__,'::GetData: Bad parameter for encoding of punched card.',"\n"
            if length ref $encoding;
        die 'Unknown encoding of punched card: ',$encoding,"\n"
            if !exists $CARDS_FORMATS{$this->{Type}}{$encoding};
           $format = $CARDS_FORMATS{$this->{Type}}{$encoding};
    }
    my @data;
    for my $c (0..$#{$this->{Card}}) {

        my $punch = join '',@{$this->{Card}[$c]}[0,11,12];
        my @holes = map {$this->{Card}[$c][$_] ? $_ : ()} 1..9;

        if (@holes == 3 && $this->{Card}[$c][8] && $this->{Card}[$c][9]) {
            $punch .= '11';
            @holes = ($holes[0]);

        } elsif (@holes == 2 && $this->{Card}[$c][9]) {
            $punch .= '1';
            @holes = ($holes[0]);
```

```
        } elsif (@holes == 2 && $this->{Card}[$c][8] && !$this->{Card}[$c][9]) {
            $punch .= '10';
            @holes = ($holes[0]);

        } elsif (!@holes) {
            @holes = (0);

        } elsif (@holes != 1) {
            die __PACKAGE__,'::GetRawData: Bad hools at column: ',$c+1,"\n";

        }

        push @data,$format->{$punch}[$holes[0]];
    }

    @data;
}

1
```