Teach Yourself KMC

独習KMC vol.9

京大マイコンクラブ 著

巻頭言

こんにちは。H4 入学の若木民喜です。

巻頭で何か書けと言われましたけど、いいんでしょうか？ ボクただの部員で何の役職も貢献もなかったんですけども。何を言うのも恐れ多いのですが……。

思えば私がこのクラブに入ったきっかけというのは、入学式あとの新歓茶話会で、とある女の子の隣に座ったことだったのです。まあ、上原哲太郎さんの妹さんの美根ちゃんだったんですけども。

ちょうどその時のボクは大学入学と同時におばさんの家の PC9801RX を盗み……いや、借りることになり、パソコンに興味津々だったのです。「私マイコンクラブはいってんのよ〜」という美根ちゃんのほんわかとした話のトーンにつられて、「なんか楽しくパソコンで遊べたらいいなぁ」というホビー感覚で、KMC に入って「しまった」のです。

ところが入ってみたら、おやまあこのクラブ、とんでもねぇガチなクラブだった訳ですよ。部室はアジトみたいだし、先輩方の話してる内容は超難解だし、モニターに向かってなんか爆笑してる不気味な人いるし（JUNET を見て笑っていたんだけど、この時はそんなもの知らなかった)、STOP キー押さないで 98 の電源きったら鉄拳制裁だし（これは今から考えるとしゃーない)。

という訳で、思ってたものと大分違っていたんですけど、ここはすごい所だってところはわかってしまったのです。なんかこの当時から 68K 何台も繋げて対戦ゲームやってたし。「ここにいると、何かいいことがあるのでは？」と思いましたよ。そして、ここに居ようと。

しかし、この高いレベルについて行けるはずもない。ので、絵を描くことで居場所を確保したんですけども。ホント絵が描けてよかったですよ……。思い出してみたら、部室で TOWNS でお絵かきばっかりしてた気がします。何もしてなかったんですけど、部室にいることが嬉しくて。今でも思い出せますよ、部室入った瞬間の靴のにおいを。

未だに PC のことなんて全然わからないんですけどもね。でも、KMC はボクの人生にとって唯一、本当にすごい人に囲まれた時期で本当に貴重な体験だったと思います。今は漫画家やってますけど、人と会わないんでね。

KMC は未だに盛況そうで何よりです。今じゃあ、PC で何でもできる時代ですからね。こんなことになるってわかってたら 1 回生の時から DOS 講座受けて、プログラム勉強して SNS で

成功してリッチになっておけばよかったです。今の部員の人達は是非六本木ヒルズに住んでください。

若木民喜

目次

巻頭言 (若木民喜) i

KMC の最近の活動 1

新入生プロジェクト紹介 1

部員のコラム 5

去年の KMC のプロジェクト日誌 (wass80) 5

妖怪少女は Flash 黄金時代の夢を見るか？ (utagawa) 10

KMC における piet の受容とその人文科学的考察 (kyp) 14

文字コードの基礎に関する考察 (suzusime) 18

チェックデジットあれこれと Damm のアルゴリズムの素晴らしさ (tetsu) 29

ダサ T のすゝめ (hakurin) 37

続・サークルで初心者向けにプログラミングの話とかをする際にちょっと考えてみたいことについて発表した (pastak) 41

あとがき 46

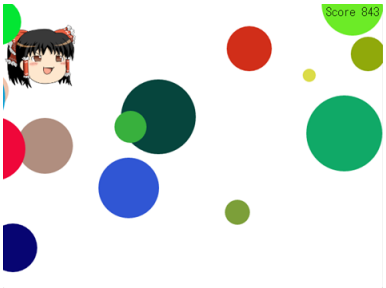
新入生プロジェクト紹介

JavaScript から始めるプログラミング 2016

drafear

最近 overwatch というオンラインゲームにハマっている KMC2 回生の drafear と申します。overwatch、トレーサーとメイちゃん強いです。overwatch のため、この記事はタメにならないダメな内容かもしれませんが、どうぞお付き合い下さい。

さて、このプロジェクトでは、主にプログラミングが初めての方を対象に、手軽に目に見える形として表現でき、なおかつブラウザとエディタしか要しない JavaScript の素晴らしさをお伝えしてきました。内容といたしましては、拡張子や絶対パス，相対パスなどの基礎的なことから始め、HTML や CSS を学び、RPG の戦闘シーンや避けゲーなどを作って参りました。



また、単なる Web ページだけでなく、Chrome 拡張を作ったり、Node.js の socket.io を用いてゲームサーバを立ててオンラインゲームを作ったり、express を用いて Web サーバを立てたりしました。

今や、JavaScript は大きく進化し、Web ページだけでなくサーバやデスクトップアプリにも使われるようになってきました。更にはあらゆるプラットフォームで動作し、その周囲には、多種多様なライブラリやフレームワークが整備され、自身の機能としてもかなりの数の API を備え、もはや OS の如く何でも実現できる言語であり、学ぶ価値は十分あります。本プロジェクトは JavaScript での GPS やジャイロセンサーなどの扱いにも触れ、プログラミング初学者だけでなく、経験者にも楽しめる内容であったと思います。例えば、以下のコードで GPS にアクセスして緯度と経度を取得できます。

```
navigator.geolocation.getCurrentPosition((position) => {
```

```
        const latitude  = position.coords.latitude;
        const longitude = position.coords.longitude;
        console.log(`Latitude is ${latitude}, Longitude is ${longitude}`);
    }, () => {});
```

特に、 socket.io は素晴らしく、 クライアント側・サーバ側あわせてたった 15 行程度のプログラムでリアルタイムチャットが実現できました。socket.io において、送信側は

```
socket.emit("message", data);
```

受信側は

```
socket.on("message", (data) => { ... });
```

とするだけで通信ができます。興味がございましたら是非触ってみてください。

このように、JavaScript をご存知の方はお気づきかもしれませんが、本プロジェクトは `function` や `var` を使わない世界観で JavaScript をお送りしてきました。今後も JavaScript の更なる進化を期待しております。JavaScript の未来の果てまでも、頭上に星々の輝きのあらんことを。

お絵描きプログラミング (Piet) 勉強会 2016

base64

このプロジェクト担当の base64 です。このプロジェクトでは、KMC で今話題のプログラミング言語 Piet の勉強をしています。Piet というのはソースコードがドット絵の難解プログラミング言語です！ 習得は他の一般的な言語よりもずっと簡単（私の主観ですが）なのでプログラミング初心者も歓迎しています。

プロジェクトでは、主に課題を通して Piet のプログラムを実際に描いてみる形で Piet の勉強をしています。今年の 1 回生は既に Piet の基本をマスターしていて、毎回面白い Piet を描いてくれるのでとても頼もしいです。エディタは dama 制作の Piet の IDE、その名も Pidet や murata 制作の UltraPiet を使用しています。どちらも検索すると出てくるので興味のある人はインストールしてみてください。また KMC の Piet 活動は日本最先端なので、Piet と検索することで KMC の主な活動を調べることができます。最近では、KMC のホームページを Piet で描かれたサーバで動かしたりしていました。今後も情熱的な KMC の Piet 活動を宜しくお願いします！

みんなでゲームを作る 2016

kata

こんにちは。担当の kata と申します。このプロジェクトは、京都大学で毎年開催される NF というお祭りで展示するゲームを制作しようというものです。

製作は新入生を主体として行っており、現在はターン制ストラテジーゲームとシューティングゲームの 2 本のプロジェクトが進行中です。毎週火曜日に行われるミーティングでは、主にそれぞれのディレクターによる進捗報告を行ったり、ゲームの仕様に関して直接話し合いを行っています。

参加者のほとんどはゲーム製作未経験の新入生で構成されていますが、皆とても意欲的に参加してくれていて、順調に開発は進行しています。それに対し先輩である私達は、これまでの経験を活かしてその新入生たちのサポートを行っており、プロジェクト名通りみんなでゲームを制作しております。

プロジェクト一同、完成に向けて頑張っていきますので、今年の NF を楽しみにお待ち下さい！

DTM 練習会 2016

kebus

どうも、こんにちは。担当者の kebus です。このプロジェクトは、パソコンで音楽を作れるようになって、みんなで自分の曲を聞かせあおう！というものです。音楽を作れるようになる、が目的であるので初心者向けでもあり、みんなで互いの曲を聞き合うので、経験者向けでもある、万人向けプロジェクトです。

活動は週替りで講座回と発表回を交互に行っており、前期は講座回で学んだことを発表回で課題として提出するというのが主でした。メロディー、コード、ベース、ドラムなど曲の作り方の基礎は一通り学んだので、後期はみんなで自作の曲を聞かせ合うのが主となる予定です。制作物は SoundCloud に投稿する予定なので、気になる方は是非！お聞きください。NFCD にもみんなで曲を出す予定です。

それでは、DTM 練習会の今後にご期待ください！

競技プログラミング練習会 2016 Normal

taisei

KMC2 回の KMC-ID: taisei です。競技プログラミング練習会 2016 Normal は、競技プログラミング初心者、主に新入生を対象に、競技プログラミングにおいて基本となるアルゴリズムの解説や、部内コンテストを行うというプロジェクトです。

通常は、30 分にかけてアルゴリズムを解説し、その後 90 分程を目処にコンテストを行い、解説を行うという流れでやっています。また、ICPC 国内予選が迫った頃には、30 分のアルゴリズムの解説を省略し、2 時間から 3 時間のコンテストを行ったこともありました。

今年のプロジェクト参加者は意欲的な人が多く、中には大会出場経験者もおり、参加者同士で切磋琢磨している様子が伺えます。直近の ICPC 国内予選でも新入生とは思えない好順位を収めた人もいました。今後に期待ですね。

お絵かき練習プロジェクト 2016

asragi

随分昔に始まって今では KMC でほぼ毎年必ず立てられるイラスト練習プロジェクトの 2016 年度担当の asragi です。

新入生向けプロジェクトは通常 KMC 2 回生が担当するのですが、イラストを長く描いている 2 回生がみな他の役職を割り当てられてたりと中々手が空かない様子であったために、今年はイラスト駆け出しの 2 回生が担当するかたちとなりました。1 年前に自身がイラスト練習プロジェクトの参加者として学んだこと、更には初心者として苦労したところを考慮しながら資料を作り、前述の 2 回生たちや先輩だけでなく、時にはイラスト経験のある新 1 回生の力も借りながらプロジェクトを進めています。参加者がどういったイラストを描きたいか、どういう動機でイラストを描いているかを確認しながらプロジェクトで取り扱う内容を決め、前期は人体を中心に、アタリの取り方など基本的なことを確認しました。後期にはこうした内容を発展させ、より実践的な内容を取り扱っていく予定です。

このプロジェクトで絵を描く楽しさを知ってもらえたら……いいな！

C#でゲームを作ろう 2016

suzusime

担当者、KMC 2 回生の ID:suzusime です。このプロジェクトは、プログラミング未経験の人でもゲームが作れるようになるために勉強しよう、というものです。

今年は環境を C#+Monogame というものに決めたので、C# の文法（変数の定義から始めて函数やクラスまで）やオブジェクト指向のあれこれ（継承やカプセル化の話）、Monogame の基本的な機能（画像表示や音声の再生など）を勉強していきました。Monogame の環境導入や情報収集に苦労した面もありましたが、経験豊富な先輩に助けられました。新入生も少しずつプログラミングが分かるようになっているようなので、このプロジェクトの最終目標である夏休み明けの成果物発表会を楽しみに待っています。

……私はこれを担当するまで C++ くらいしか知らなかったのですが、オブジェクト指向を使いまくるこういう用途では C# はいいですね。プロパティの記法は本当に便利だと思いますし、Effective C++ みたいな本を読まなくてもそこそこちゃんと使えるのは良いなぁ、と。

去年の KMC のプロジェクト日誌

wass80

KMC 2 回生の wass80 です。 去年の部誌にプロジェクト紹介が不足していました。去年のプロジェクトを僕視点で紹介していきます。主に自分が参加していたものになることをご了承ください。 新勧プロジェクト，ゲーム作りプロジェクトの紹介はありません。

5/5 Go 言語で 5 月 5 日を祝う会

5 月 5 日ということで Go 言語を祝う会です。Go 言語のチュートリアル[*1]を行いました。Go 言語のマスコットキャラクターである Gopher 君ですが，その顔を貼りつけた帽子を被った Pepper 君と部員でピザを囲んで食べました。

5/7–7/9 Haskell 勉強会

すごい H 本（『すごい Haskell 楽しく学ぼう』）を読む会です。3 ヶ月で一通り H 本を読み終わりました。「Haskell のサブセットを Haskell で書くぞ」という目標が立っていた時期もありました。実際に作ることはできませんでしたが，関数型言語はパーサを書くのに向いていることがわかりました。

5/19–6/9 C++ 初心者勉強会 for 競プロ

C++ についての知識が少なかったので，自分が立てたプロジェクトです。競技プログラミング練習会の参考書である蟻本ですが，蟻本にかかれている C++ は古いです。そのため，今時の C++ と STL などを活用を教えてもらうという目的でした。C++ の基礎会と STL 紹介回で計 3 回開催されました。

[*1] `https://tour.golang.org/`

6/27– ルール追加大富豪

京大数学科で流行っているらしいゲームを，KMC に輸入して個人的に部員を集めて不定期に行っています。健全で完全なルール定義をするのは，数学的でもあり，プログラミング的であり，面白いです。

6/30– 組合せ最適化読書会 2015

6 年前に行われていたプロジェクトの復活版。『組合せ最適化』を読む会です。1000 ページを優に超え，1 ページに書かれている証明の圧力が濃すぎる本です。僕がペースを崩してしまったため，一時中断しています。

8/6 H 本速読会

Haskell 勉強会が少し中途半端に終わってしまったので，H 本を 1 日で復習する会を立てました。速読会の途中，Haskell で Segment Tree を書けとプロジェクトリーダーの僕に指示が下りました。僕はすぐに書ききれなかったので，H 本は Monad の直前で中断になりました。後編を 2016 年に行う予定です。

8/7–8/10 ICFPC2015

3 日間のマラソンコンテストである ICFPC に KMC として 1 チーム参戦しました。マラソンに参加したことがなかったので，僕には出来ることが少なかったです。手でテトリスをする部員と，プログラムでテトリスをする部員が現れてチームとしては 39 位でした。

8/10–8/31 蟻本速読会

『プログラミングコンテストチャレンジブック』を 4 日間で読み終える回です。進度に合わせた問題の部内コンテストを用意してもらいました。夏休みに問題を解く習慣を作れなかったのが，去年の一番の反省です。

8/10–9/28 CUI のゲーム作っといて

コンソールで動く CUI のゲームを各々作ってきて持ち寄る会。リッチな UI を用意する部員や，3D シェーディングを実装する部員がいました。自分は Haskell の練習として大富豪を実装しました。トランプの絵文字は便利ですね。

9/30– 終焉の C++

『Effective C++』を輪読する会。読了。 一部古い内容を適宜 C++11, 14, 17 に置き換えて読んでいきました。昔読んだことがある本でしたが，本質を掴み直すことができ良かったです。現在『Effective STL』と『Effective Modern C++』の輪読が進行中。

10/3– ぼくのかんがえたさいきょうのついったーくらいあんと

最強のツイッタークライアントを作る会。 Electron + Vue.js で作っています。リポジトリはこちら[*2]です。テストを作る熱量までなくて，開発は難しいという気持ちになりました。JavaScript を初心者の部員に教えることも，なかなかうまく行かなくて難しかったです。

10/13 ハナノパズルの AI を作ろう 2015

『ハナノパズル』という面白いパズルゲームを解く AI のコンテスト。パズルのシミュレーションの問題としても，程よい難易度で楽しかったです。探索に用いる評価関数を決めるのが難しいですね。最終面は誰も解ききれなかったです。

10/29 闇鍋パ 2015 秋

闇鍋は家が燃える危険があるので却下され，光鍋に変更されました。各々が鍋の具材を持ち寄って，光の下で鍋に突っ込むお祭り。美味しかった具材は白菜と鰹。不味かった具材は栗とバナナでした。

11/10– Coq 勉強会

Coq のドリルである『Software Foundations』をやっていく会。自然演繹と直観論理に慣れることができました。最初のうちはわからないことを考え合うことが多かったですが，今は集まって黙々こなす形態になっています。問題進捗管理表が部内で稼働中。現在 Logic まで進みました。

11/26 バーガーキング襲撃

バーガーキングの BIG 割に挑みました。 /BIG | ビッグ/を含むレシートを提示することで，ビッグマックの値段でワッパーが食べられるキャンペーンです。ビッグエンディアンのアーキテクチャは準備できなかったです。

[*2] `https://github.com/kmc-jp/ult-twiclient`

12/27 クリスマスパーティー 2015

クリスマスウィークの日曜日に部室で大宴会が催されました。大量のケーキが提供されて，僕はとても嬉しかったです。シュトーレンも吉田チキンの丸焼きも美味しいです。飲み飲みする部員とボードゲームする部員に別れて徹夜はいつもの流れですね。

1/11 CAMPHOR-KMC 第 2 回 LT 大会

KMC と友好的関係を築いている CAMPHOR というおしゃれプログラミングコミュニティー。CAMPHOR のお家に KMC の部員が押しかけて LT を投げ合いました。ビットコインやウェアラブルガジェットなど，いろいろな分野の話が集まって大盛況でした。

1/30 KMC 1 回生懇親会 2016@フジ

1 回生の最後，期末テストの終わり，これから新勧で忙しいという頃合いで打ち上げ。モダン焼き屋さん『フジ』で 5，6 人前のデラックスモダン（通称ざぶとん）2 つを 10 人で囲みました。フジとはもう何十年との付き合いらしいので，これからもまた食べにいきたいですね。

2/11 電電宮参り 2016

部室には法輪寺電電宮の御札が飾ってあります。 2014 年，2015 年の御札が部室にほったらかしであったので，返納して新しい御札をもらいにいきました。電電宮の御札を交換した直後，悲しいことに部室の Win 機であった Yamazaki が壊れる事件が起きました。

2/14 ばれんたいんチョコフォンデュぱーちー

チョコフォンデュを楽しむぱーちーを開催してもらいました。この日にチョコを食べるとその年の運気が良くなるという話を僕も聞きました。なので，マシュマロとかビスケットとか柿の種とかブラックサンダーとかをチョコに浸して食べました。

4/1 エイプリルフール

KMC を KPC (Kyoto University Piet Club) に改名しました。それにともなって，特設サイト `http://april2016.kmc.gr.jp/` を準備しました。現在は紹介記事になっています。当日，このサイトは Piet で書かれた HTTP サーバで動いていました。準備が大変だったという話は，URL 先でどうぞ。

まとめ

僕の参加した 1 年間のプロジェクトを紹介しました。振り返ると，インプットは多かったですが，アウトプットが少なかったと思います。2016 年度はコーディング大海（KMC のハッカソン）のポセイドン（司会）になりました。それを活かして，2016 年は形のあるものをいくつか作りたいです。

妖怪少女は Flash 黄金時代の夢を見るか？

utagawa

はじめに

初めましての方は初めまして、そうじゃない方はお久しぶりです。KMC 2 回生の utagawa です。普段は広報[*1]として宣伝や渉外をしたり、Slack[*2]の BOT を作って[*3]生活を豊かにしたりしています。ちなみに最近は、降水確率が高かったら Slack に通知する仕組みを整えました[*4]。

今回はなんやかんやあって Flash 黄金時代について、回顧と懐古を交えながらゆるく執筆していこうと思います。

Flash 黄金時代とは

さて、この部誌は全年齢対象であり、すべての読者の方が Flash 黄金時代を経験したり、あるいははっきりと記憶したりしているとは限らないので、まずはざっくりと Flash 黄金時代に関する解説をしていこうと思います。

Flash 黄金時代は、Macromedia Flash[*5]によって作成された数多くの動画が一世を風靡した時代のことです。当時はまだダイヤルアップやら ADSL やらが主流で、要するにインターネット回線は今ほど高速ではありませんでした。今と同じような感覚で YouTube に上がっているような動画を見ようとすると、今の何十倍もの時間がかかってしまいます。そのような状況で、Flash は生まれました。動画より軽量で、アニメーションからゲームまで作れる自由度の高さから、Flash は瞬く間にインターネット中に広まっていきました。これが Flash 黄金時代です。

[*1]正確には、副広報という役職に就いています。

[*2]サークル内で連絡や雑談に使用しているチャットツール。

[*3]去年の 11 月ぐらいから、部員がすごい勢いで Slack の BOT を作っていて、何がそんなに彼らを駆り立てているのだろう……と思いながら眺めています。

[*4]`http://utgwkk.hateblo.jp/entry/2016/06/26/150737`

[*5]現 Adobe Flash

具体的な年代については諸説ありますが、私はだいたい 2000 年ぐらいから 2006 年までという解釈をしています。

ある種の青春

私が Flash 動画を見始めたのは、小学校に入って少ししてからでした。

実家に ADSL 回線と Windows XP が入ったパソコンがあり、私はとにかく Flash 動画を見あさっていました。おもしろフラッシュのデータベースサイトみたいなのを片っ端から見て、気に入ったものがあれば何度も何度も見ていました。シュールなものから、爆笑ものまで、全部見ていて、Flash 動画を見ない日はなかったというぐらいに見ていて、親に何度も心配されました。

ここまで Flash 動画を見まくるようになったのは、まあ当時めちゃくちゃインターネットで流行っていたからというのもあるのかもしれませんが、他に楽しみがなかったからというのもあるのかもしれません。なんかジジ臭い言い回しになってしまいましたが、当時はあまり身体が強くなくて、体育は苦手だったので、あまり外で遊ぶたちではありませんでした。そのうえ家にパソコンとインターネット回線があったらもうやることは一つに決まるわけです[*6]。他の生徒たちがワイワイ遊んだりほどほどに勉強したりしている中、私はほどほどに勉強しながら Flash 動画を見まくっていました。青春は Flash 動画の中にありました。笑いも、癒やしも、驚きも、感動も、とにかくいろいろなものを Flash 動画で見てきました。そういう生活が小学校 5 年ぐらいまで続きました。5 年生になったときに、地元のスポーツ少年団に入団して、ソフトボールに打ち込むようになりました。ここでようやく青春が Flash 動画からソフトボールに替わりました。

動画共有サイトの出現と、黄金時代の終焉

小学校 4 年生ぐらいの頃に、YouTube というサイトが、Flash 動画のまとめサイトに出現するようになってきました。なんといろいろな動画を簡単に共有できるサービスらしいということは知っていましたが、当時はそこまで YouTube 自体には注目していませんでした。ちなみに、こころんさんのビビりバイオハザード実況動画[*7]をめっちゃ見ていたということを覚えています。

ソフトボールに打ち込み、Flash 動画を見る時間が減っていって、気がついたらニコニコ動画が大流行になっていました[*8]。MAD 動画が流行っていって、東方 Project のボーカルアレンジが流行っていって、初音ミクが流行っていって、すごい勢いでいろいろな動画が流行り、追いかけるのに必死だったのを覚えています。確かこれが 2007 年ぐらいのことで、いわゆる

[*6] 当時はまだプログラミングはしていませんでした。

[*7] `https://www.youtube.com/watch?v=-G7h626wJwM`

[*8] 当時は一般会員は 19 時以降は動画を視聴できなかったので、19 時になるまでに見たい動画のページを開きまくるという知見を姉から学びました。

Flash 職人が続々とニコニコ動画や YouTube で自分の作品を発表していくようになりました。また、ニコニコ動画の中で、クオリティの高い動画を作る人が次々と現れるようになりました。個人サイトで Flash 動画を発表して、それがまとめサイトに載る時代から、初めから動画共有サービスにアップされて、分かりやすい再生数・高評価の数・マイリスト登録数という数値による評価がなされる時代にシフトしていきました。再生数とかマイリスト登録数が多いと必ずよい動画であるかどうかについては私は懐疑的ですが、今思い返すと、そういう指標が大々的に導入されて、大衆化というか資本主義化というかより分かりやすくなったとか、そういう言い方もできそうだなあと感じています。

同じくらいの時期に、のまネコ問題[*9]や、Adobe 社による Macromedia 社の買収など、いろいろなことが重なった結果、いわゆる Flash 黄金時代は終わりを告げました。時代は動画共有サイトに移ったというわけです。

Flash 黄金時代の夢

さて Flash 黄金時代は、インターネット的に考えれば遠い昔のこととも言えるのかもしれません[*10]が、私には昨日のことのようにも感じられます。私はべつにいわゆる懐古主義者であるというつもりはないのですが、最近は週に 1 回ぐらいのペースで Flash 動画のまとめサイトを見て回る活動をしているような気がします。いろいろなサイトを見て、ああこれはめっちゃハマってたとか、この人は今こんなことをしているのかとか、この人はとうとう仕事としてやるようになったのかとか、いろいろな発見があります。中には残念ながらインターネットから消えてしまった人もいます。そのようなことがあるたびに、時の移り変わりを実感しては、ため息をつく活動ばかりしています。

このような活動に専念してしまうと、その後の生産活動に支障をきたしてしまうのですが、なかなかやめられないもので、気がつくと日付が変わっているという体験を何度もしています。しかしこの活動は、人間の脳の活動としての夢のようなもので、精神の安定のための必要行動であるというのが私の考えです。夢の中では、現実の出来事とか、頭の中の考えなどが整理整頓されます。そうして明日を強く生きるための活力を得ることができるのです[*11]。

マイリスト

せっかくの機会なので、私が特に好きな Flash 動画の一部を列挙していこうと思います。

- 終わらない鎮魂歌を歌おう[*12]

[*9] 巨大匿名掲示板 2 ちゃんねるのキャラクターであるモナーに酷似したキャラクターが商標登録されそうになって物議を醸した問題。ここでは詳しくは取り上げません。

[*10] インターネットの流行の移り変わりは、年を追うごとに早くなっていって、とりわけ最近のトレンドに中々ついていけなくて、もう年なのかなあということも考えてしまいます。

[*11] 筆者は心理学的な知識に乏しいので、適当なことを言っているかもしれません。

[*12] いつの間にかラノベが出ててびっくりした。

  - http://www.niseyono.com/anime01.html
- 3 歳シリーズ
  - http://3sai.sakura.ne.jp/
- ばぁちゃんと過ごした日
  - http://sv.kerosoft.com/~okkunet/flash/baatyan.html
- Savior Cat
  - http://web.archive.org/web/20060629044600/http://freett.com/hachimiri_film_/flash/savior_cat.htm
- 小小作品[*13]
  - https://www.youtube.com/watch?v=ypPBHqhjNJQ
- ハクシャクノテンシ
  - http://www2.atpages.jp/ngravity2/html/haku_nae_n-gravity.html
- YUKINO
  - http://www.morinono.net/yukino

他にもいっぱいあるのですが、この部誌が電話帳のようになるのを避けるためにも、ここでとどめておきます。

おわりに

今も Flash 動画を見ながらこの記事を書いています。黄金時代は、良かったことも悪かったこともいろいろありましたが、今は有名どころなら動画共有サイトで簡単に遡れるようになっていて、便利な時代だなあという感じです[*14]。

後から見返すと、なんだか痛い上にまとまりのない文章になったなあという感じですね。とにかく懐古をしたりよくわからない考察をしたりしていますが、こういった文章を出版物に書くのは初めての経験なので、少しドキドキしています[*15]。一つ心配なのは、去年のアドベントカレンダーの記事[*16][*17]といい、怪文書を書くことが専門の人だと思われてはいないだろうか、ということですね。まあいずれにせよ、けっこう心の中にあったものを形にできたので満足しています。

この記事はこういう感じで締めくくろうと思いますが、もしこの記事に対して共感とか意見とかそういうのがあった場合は @utgwkk(Twitter) まで連絡ください。そして一緒に飲みましょう。

[*13]特に好きなのは 3 作目なので、3 作目の URL（YouTube の転載動画ですが……）を貼ります。
[*14]まあだいたいが転載なんですけどね……。
[*15]ちなみにこの部誌は来年夏以降ぐらいには、ネット上で無料公開されるようになる予定です。
[*16]http://utgwkk.hateblo.jp/entry/2015/12/02/032047
[*17]http://utgwkk.hateblo.jp/entry/2015/12/22/222533

KMC における piet の受容とその人文科学的考察

kyp

はじめに

piet という難解言語は何であり、そしてなぜ KMC を席捲しているのか。これが本稿の根源にある問いである。

KMC への入部がまだ検討段階にあった頃、その新歓説明会や新入生プロジェクトに私が参加していたのはたった 2 ヶ月前の事であり、今でもその仔細を鮮明に覚えているが、先輩部員によるゲーム制作、DTM、競技プログラミング等の各練習会の説明が続く中で、他のそれとは明らかに一線を画し、異彩を放ちながら KMC を KMC たらしめているもの、それが piet 勉強会であった。また広報合戦においても、piet のソースコードが大きく載せられたカラーのビラは、その情報の非対称性によって、最も多くの新入生の心に疑問符を浮かべた特筆すべきものだろう。

結局私は初回の piet 勉強会に参加した後に KMC への入部を決定し、それ以来他の参加者に比べて劣る我が数理的能力や記憶力を恨みながら、毎回どうにか参加して拙い piet を書いている。それが私の KMC での主たる立ち位置であり、よってこの部誌でも piet について書くのが適当と思われたが、私はまだ既存技術の習得に齷齪している状況であり、他の人々に伝えられる技術や体験は存在せず、まして団体外への公表となれば何一つ内容が無い。

そこでお茶を濁すようではあるが、私は piet そのものについての詮索は避け、なぜ piet が KMC においてこれほどの存在を示すようになったのか、特に「京都」という街との結びつきから文化地理学的アプローチを用いて探っていきたいと思う。

最後になるが、上記のように一介の新米に過ぎない私を激励し、強く背中を押してくれた pastak 氏、このような出版の機会を与えてくれた suzusime 氏を初めとするすべての KMC メンバーに深く感謝したい。

■先行研究について　存在しない。よって本稿は意義がある。

言語帝国主義と情報科学

piet そのものの性質について探る前に、まずは一般的なプログラミング言語の性質を見てみよう。piet とは異なり、一般的言語は文字の羅列によって表記され、そのほとんどは英語に用いられる基本ラテン文字 26 字（52 形態）と英文記号に依拠する。また、その構文は明らかに英語文法に拠って成立している。“if”, “else” といった英単語はその代表であり、英語の母語話者はそのプログラミング言語に対する前提知識が完全に欠如した状態であっても、目に入る英語との共通語彙によって一定の推測を図ることはできるだろう。少なくとも、小中学生へのプログラミング教育が非英語圏と比べて遥かに容易であることは明白である。なでしこはその極僅かなる例外ではあるが、そもそも英文字で表される概念を扱う際には英語を用いざるをえず[*1]、アセンブリにおいては表面上英語と表記を共にする表現は極僅かであるが、その命令の大半は英語の頭文字語である。

こうして考えてみれば、あらゆる学問はイデオロギーの汚染を免れ得ないといえども、情報科学もその例外ではないことを思い知らされる。世界には非ローマ字的な固有の文字体系が数多く存在し、それをいわば「母字」として持つ民族は多々存在する。情報技術競争において、ロシアや日本で「天才」が現れることはあっても、徒党を組んで戦う団体戦ではアメリカやインドに遅れを取り、必然的に大規模なサービスでは遅れを取ってしまうのも、この英語・ローマ字至上主義的な情報科学の偏向がもたらした悲劇の一つであることは明らかであろう。

この情報科学において顕著な特定言語の優位状況は一般に言語帝国主義の一環として捉えられる。即ち、情報科学の柱梁たるプログラミング言語を英語に委託することによって、我々が IT を永遠の「よそ者」として扱い続けること、これが構造的に確定してしまうのである。

一方で piet を考えてみよう。piet は人類に共通な「色」を命令として実行する言語である。もちろん色をどのように分節化して捉えるか、あるいはそもそも色を何の象徴としてみるかという問題は付随しているわけであるが、piet で優遇される色は色相、明度、彩度を科学的に平等に分割したものである。そこに特定の認識形式は存在せず、勿論イデオロギーも絡んではこない。また、命令の名称は英語によって与えられてはいるが、ソースコードそのものには影響しないから、各々が各々の開発環境においてヘルプを差し替えればいいだけである。

piet による開発がより一般的になれば、もう非英語圏の人が英単語の羅列にかかずらう必要はない。それどころか、特定の文字を覚える必要さえないから、全世界に 7 億 5900 万人[*2]と言われる成人非識字人口においても、無限の情報科学への扉を開く一筋の光明となることは明らかであろう。

しかし残念ながら、piet の使用は現段階ではごく局所的かつ補佐的なものに留まっている。具体的には、京都大学の一サークルにおいてのみジョークの一種として用いられているのである。それならばなぜ、piet の最先端は京都に存在するのか。

[*1]「ポインタ」「OS」等
[*2]UNESCO 推計、2010

近代日本の学問史、ことに人文学に目を向けてみればその答えは自ずと見えてくる。明治から昭和初期の日本において、哲学は西洋の学問であり、前述の表現を使うならまさしく「よそ者」であった。曰く、既に出来上がっている西洋の哲学の理解、及びその鏡写しの伝播が「哲学」であり、いわば日本は文化的に西洋の隷属下にあったのである。そしてこの状況に異を唱え、西洋の枠組みを適宜利用しながら、しかし西洋主義の超克を図って独自の哲学を編み出した知識人達、それがまさしく京都学派の哲学者達である。

昔ほどではないが、しかし現代においても国際基準を斜めから捉える価値観は未だ京都大学と京都の地に残っているように思われる[*3]。よって KMC で piet が流行し、衰えの兆しも未だ見えていないこの状況は、人文科学的な視点からみれば必然的なものと見做せるのではないだろうか。

「無」への着目とその深化

上記の仮説を採用することにより、我々は KMC における piet の流行の原因を理解した。即ち、言語帝国主義の一翼を担う従来的な情報科学への反発として、言語的に中立な言語の隆盛が得られたのである。そして、それを成し遂げたのはまさに京都という都市の歴史が持つ特異性によるものであった。

さて、京都哲学の傾向が KMC の流行に一致しているという仮説の正しさが担保されたとして、次はこの流行が永続するものなのか、はたまた一過性のものに過ぎないのか考察しよう。

かの哲学の特徴は「無」という概念の援用——有を有として無条件に前提する西洋哲学へのアンチテーゼとなる分割不可能な地平への眼差しにあったわけであるが、これもまた必然的に piet に備わっていると見做せる。即ち、piet を書くとき、我々は有意味たる色とりどりの codel ＝有とは別に、無意味たることによってプログラムを規定する余白＝無を残さざるを得ないからである。無なくして有は成立せず、有なくして無は成立しない。また無も有を規定するという意味において有であり、有も無を規定するという意味において無である。piet において、一切は有であると同時に無なのである。

しかし残念なことに、piet において無の概念は不十分であると考えざるを得ない。即ち、我々は piet を視界に認めるとき、最小単位まで分割することによって初めて無の観念を得るのであって、piet の相貌はむしろアニメキャラクターや英文字を纏っている。即ち、piet は無と併存する言語ではあるが、しかしその無はより煌びやかな色を纏っており、五蘊六根を徒に刺激してしまうのである。

そこで piet から目を移し、数十年余のプログラムの歴史に目を向けてみれば、我々の目的に適う言語が一つ横たわっていることに誰しもが気づく。即ち、言語帝国主義への抵抗となり、無と併存し、かつ煩悩と袂を分かちて六根清浄の援けとなる言語があることに。まさしくその言語は、Whitespace である。Whitespace において、一切の意味は空白に還元される——piet のような妥協的なものとしてではなく、そこでは有が有に到る唯一の手段が無なのである。

[*3]参考『日本経済新聞』2015 年 10 月 21 日朝刊「京大生よ日本語で考えよ　山極学長 2 年目に」

Whitespace の有は無の深層に潜む。また有は我々が網膜に見出すことのできない形而上的なものであって、そこでは無と有の性質は倒錯すらしているのである。Whitespace が無の観念を最も根本的に突き詰めた言語であることは疑いようがない。そして無への絶え間ない眼差しは、有への我々の固執を退け、必ずや我々の観想生活の質を高めてくれるのである。ゆえに、我々は今 piet を捨て、Whitespace への意慾を示すべきなのだ。

おわりに

ここまで第一章では社会言語学的観点から、そして第二章では哲学史的観点から、KMC における piet の受容の原因、そして Whitespace への移行という展望を論じてきた。拙い文章ではあったが、読者諸君はきっと止め難き歴史の歩みに共鳴してくれたことを信ずる。

最後に、繰り返しにはなるが、私は Whitespace の設計理念に感動したので、以下 Whitespace でその素晴らしさを記述する。

[*4]

反省します。

[*4] This source is published on `http://ideone.com/8RO7Gl`

文字コードの基礎に関する考察

suzusime

序

この小文は、文字コードを支える基礎的な思想の解明を目的としたものである[*1]。

まずは、なぜこのような文章を書くに至ったか、その問題意識について説明したいと思う。それがこの論の目的とするところを最もよく説明できると信ずるからである。

文字コードというのは大雑把に言えば、文字をコンピュータで扱えるような二進数の列に変換するための規則の体系である。現代社会ではホームページやブログから新聞記事、小説に至るまで非常に多くのものがコンピュータを使って書かれているから、我々は日々当然のように文字コードの恩恵を受けているのである。ただ、文字コードを成り立たせている基礎的な思想というものは決して自明なものではない。私が「文字コードとは何であろうか」と疑問することになった最初のきっかけは、おそらく「TRON コードは、いわばアンチ文字コードとでもいうべきものである」という旨の記述をインターネットのどこかで見かけたことである[*2]。TRON コードというのは、坂村健さんの提唱によって生まれた国産 OS である TRON で使われている文字コードであり、とても多くの文字を収録していることに特徴がある。

当時の私は旧字に興味を持っていたこともあり、あまり多くの漢字が使えない JIS コード、Unicode に不満を感じていたため[*3]、一点しんにょうと二点しんにょうすらもすべて分け、漢字だけで 7 万文字を収録しているという TRON コードは素晴らしいものに見えた。たくさんの文字を持っている文字コードこそが正義、そのような素朴な見方をしていたのである。

かような考え方に反省を促したのが、前述の文章であった。TRON コードは「世界のあらゆる文字を収録するというコンセプトの下」に、文字を少しでも異なれば区別するという方針で収録しているが、それは文字コードの根本である「同じものを同じものとしてみる」という「包摂」の概念と根本的に矛盾している。よって TRON コードは文字コードと言うよりもアンチ文字コードである。そのようなことが書かれていたと思う。

[*1] 当初は「文字コードの哲学的基礎」などと von Neumann に倣った題にしようかと思っていたが、さすがに Neumann に失礼なのでやめにした。

[*2] どこに書いてあったかは忘れてしまったし、今少し検索しただけでは見つけることができなかった。

[*3] 当時持っていた漢和辞典には親字ごとに JIS と Unicode での番号が記されていたのであるが、JIS はおろか Unicode にも収録されていない文字がたくさんあったのである。

私にとっての「包摂」の印象は、コンピュータで扱う際に漢字が多くなりすぎないように便宜上決めただけのもの、使いたい文字がパソコンで使えなくなる諸悪の根源、というものであった。それが重要であるというのである。最初は不思議に思ったが、少し考えればそれが当然であることが理解できた。人が書く文字の形は、それが同じ文字であってもひとつひとつ異なっている。それを「少しの差異は無視して」同じ文字であると判断することによって、文字を通じた意思疎通が可能になる。つまり、「包摂」は文字コードのみならず、文字を使った情報伝達の本質的部分である、と。

それから幾年、文字コードの様々な機能や問題を知ってゆくにつれ、包摂が文字コードの本質であるとの考えは強くなっていった。例えば、Unicode に於ける CJK 統合漢字、CJK 互換漢字、正規化、IVS などの諸概念は包摂に関わることとして把握できると考えている。

一方で、包摂の基準を決めるのは難しい。求められる包摂基準は時と場合により異なるし、人によっても異なるかも知れない。そもそも原理上包摂は可能なのか、文字コードは（妥当に）存在できるのか、そんな疑問まで生じてくるのである。

しかし、包摂の妥当性についての問いは、窮極的には「文字とは何か」「言語とは何か」という大問題に帰着してしまう。それを一介の素人たる私[*4]が解決するのは困難である。だが、この世界、この時代に於いては現に文字コードが存在しているのである。ならば、現存する文字コードが文字に対しどのような態度で向き合うことで存在しているのか、またそれによってどのような問題が生じているのかを考えることは、文字コードに対する向き合い方を考えることのみならず、文字の問題全体に対する一つのヒントとしても役立つのではないか。これが本稿の目的である。

文字コードの数理モデルのようななにか

さて、先に「包摂」が文字コードの本質であるといったが、包摂とは何であるかを説明していなかったので、それをまずは論じようと思う。そして、包摂についての観察から、文字コードのモデルとでもいうべき“絵”をかいてみたい。そのことで随分見通しが良くなるはずである。そのために次のような例から始める。

昔よく JIS コードの欠陥として挙げられた問題に「コンピュータでは森鷗外の『鷗』という文字が出せず、『鴎』という変な文字しか使えない」というものがある。これについて私は当初「『鷗』という文字が『鴎』という文字に包摂されているから」だと思っていた。つまり、（ちょうど常用漢字のもとで「區」の代わりに「区」を使わなければならない[*5]ように）JIS コードのもとでは「鷗」という文字の代わりに「鴎」を使うよう強制されている、という理解である。しかし、これは正しくなかった[*6]。

[*4]私は理学部で物理学を学んでいる。このことをあまり言い訳とはしたくないが、以下で若干理論物理学の方法を参考にした議論をしている（つもり）なので、ここでこの情報を記しておく。

[*5]もちろん、常用漢字はあくまで目安であって強制性を持つものではないが。

[*6]と言いたかったのだが、この字に関してはそうも断言できない。83JIS での例示字形変更に関わる問題があるからである。この例示字形変更はこれから述べていく包摂の枠を逸脱したものであって、後に面倒な問題を残す原因と

というのも、第一に、包摂は対等なものである。「包」という漢字が使われているのでどちらかがどちらかを包んでいるという関係であると誤解していたが、そうではなく「髙≒高」という、同じものと見做すという操作が包摂である。

次の図 1 に包摂の例をいくつか挙げてみた。

図 1　包摂の例

一番左がもっとも単純な例である。二つの手書きの「あ」は同じ文字であると判断するだろう。つまり、この二つの文字は包摂の関係にある。

真ん中は、手書きと明朝体の文字の場合である。「令」の字の下の方の書き方は少し違うが、これも多くの人は同じ文字と判断するだろう。

最後に、右側のものはどうだろう。これはどちらも明朝体であるが、「歩」の下の部分に点があるかないかという違いがある。「歩」という字の旧字体に合わせた点のない書き方と、新字体に合わせた点のある書き方の違いに過ぎないのであるが、これは果たして同じ文字と言えるだろうか。

ここで少し包摂の基準を決める難しさの一端を理解して頂けたと思うが、そのことは後で詳しく考えるので、ここでは包摂の基準をうまく決めることが出来たとして次へ進む。とにかく、このように同じ文字とみなすことができるということ、それが包摂である[*7]。

このように包摂の基準を決めたとすると、文字全体は次の図 2 のように「同じ文字は一つのグループに」とすることで、グループに分けることができる。

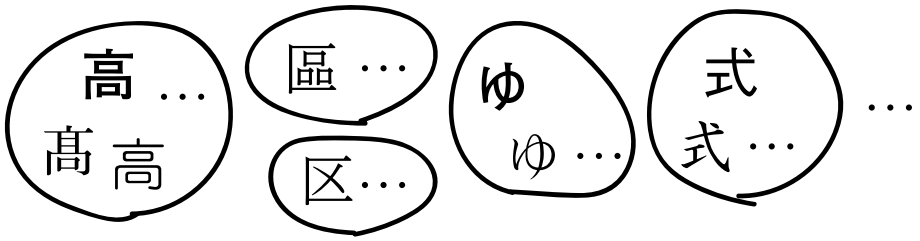


図 2　グループ分け

なった。JIS において包摂という概念が意識されるようになったのは 97JIS からであるらしいので、それ以前のものであるから仕方ないと言うこともできるが…… 詳細は [1] を参照してほしい。よって、以下では代わりにそのような問題のない髙と高の包摂を例にとって話をすることにする。

[*7]包摂というのは JIS の用語であって、Unicode では「合成」という。ここでは術語であることを強調するために包摂というあまり他では使わない言葉を使うことにしたが、この包摂の定義が JIS のものと一致しているかは分からないし、問題にしていない。あくまで普遍的な文字コードの原理としてこのような「包摂」が存在すると考えている。以下、他にも言葉の定義が各種規格や慣用と異なるものがあると思うが、ご容赦願いたい。

この図を眺めると、次のようなことが分かる。このグループのひとつひとつこそが、私たちが「文字」として考えているものではないか！ なぜなら、同じグループに属する文字は「同じ文字と見做される」ものなのだから。グループ全体でひとつとみて、それを単位とするのである。

こうして、「文字」が得られた。我々の文字をビット列に変換するという目標に近づくためには、まずはこれらのすべてに順番に番号を振っていけば良い。たとえば、「ゆ」の字には 1 番、「式」の字には 2 番……といったふうに。このように番号を付けられたグループの全体を符号化文字集合という。

そして、それぞれにただ自然数の番号を振っただけではコンピュータで扱うには不便なので、適当にこれらを 2 進数の列（ビット列）として表現する方法を決めてやる。この方法を符号化形式という。

こうして、文字をグループに分け、グループに番号を振り、番号をビット列に変換することができるようになった。これが文字コードの概観である。

これでもよいのだが、集合論の言葉を使って書くと明快だと思うので、以上のことを数学風に書いてみる[*8]。

まず、符号化の対象にしたい文字全体の集合 U をとる。この集合 U 上の二項関係として包摂 $\sim$ を考える。これの定義は、

$$a, b \in U \text{ に対し、} a \sim b \Longleftrightarrow a \text{ と } b \text{ は同じ文字}$$

とする。すると、$\sim$ は同値関係となる。

$a \in U$ に対し、$C(a) := \{b \in U | b \sim a\}$ と定義する（これを a の同値類という）と、これらは上で述べたグループに対応するもので、U 全体を交わらない集合で分割する。同値類全体の集合を U の $\sim$ による商集合といい、$U/\sim$ と書く。

なお、商集合 $U/\sim$ の元、つまりひとつの同値類は、その同値類に含まれるひとつの U の元を言えば、それだけで一意に指定することが出来る。このひとつの元を代表元という。先程の例で言えば、「高」という文字のグループを指定するには、ひとつの明朝体の「高」を持ってくれば十分だということである。

さて、この商集合は有限集合であると仮定しよう[*9]。すると、この集合からある集合 $S \subset \mathbb{N}$ に対して全単射 f を作れる。そして、S からある集合 $B \subset$（ビット列全体の集合）$\cong \mathbb{N}$ に対しても全単射 g が作れる。ここで、f がグループへの番号付け、g が番号からビット列への変換（符号化形式）に対応するのである。

[*8]私は [5] を読んで「座標系に依存しない実体」という思想と、その理論が相対論の記述に用いられているという事実を知り甚く感銘を受けた。このモデルは（後の文字コード変換の節など特に）多様体の理論を真似したものである。また、[6] には観察や思考実験から対象の性質を必要十分な精度で表したモデルを作り、そこから対象を分析するという方法を教えられた。

[*9]高々可算ならば番号を振れるので問題はないのだが、無限個の文字をもつ文字コードを定義するのは事実上困難だろうと思う。そもそも、この「この世界にある符号化したい対象の文字すべて」の集合 U などを定義する事が果たして可能なのかという話であるが、それは次のように考えればよいだろう。まず商集合としての文字の表を用意しておき、もし現実世界のある文字が、定めた包摂による射影でこれらのどれかに写されたならば、その文字は U の元であった、というのである。

すると、文字コードは次のように定義できる。

定義. 文字の集合 U をとる。U 上の同値関係 $\sim$ をとる。$U/\sim$ が有限集合であるとき、ある $S \subset \mathbb{N}$、ある $B \subset$（ビット列全体の集合）$\cong \mathbb{N}$ に対して、全単射 $f : U/\sim \to S$、$g : S \to B$ が存在する。このとき、組 $(U, S, B, \sim, f, g)$ を文字コードという。

また、このとき同値関係 $\sim$ を包摂、組 (S, f) を $(U, \sim)$ に関する符号化文字集合、組 (B, g) を符号化文字集合 (S, f) に関する符号化形式という。

これを図にすると次の図 3 のようになる。

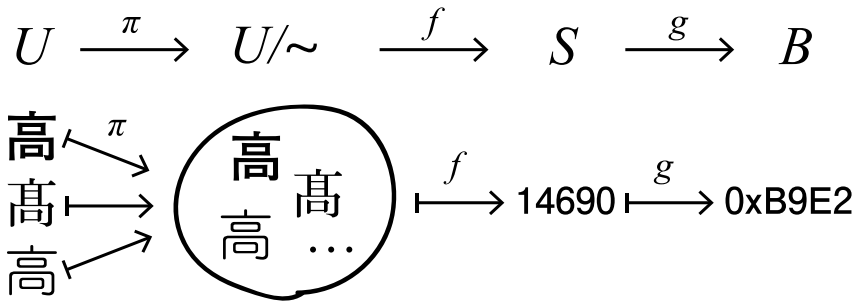


図 3　文字コード概観

このような図式でわかるのは、文字コードにより生成された一つのビット列が表すものが何かということである。というのも、ある $b \in B$ から戻せるのは、あくまで $U/\sim$ の元までなのだ。$f^{-1}(g^{-1}(b)) \in U/\sim$ までは b から一意に定めることができるが、商集合を作ったときの写像 $\pi : U \to U/\sim$（自然な射影）は、一般的に単射ではないので逆写像が定義できない。

このことは文字コードの作り方を考えれば、ある意味当然である。ある明朝体の文字「高」からビット列 B9E2 が得られたとしよう[*10]。このとき、B9E2 が表すのは、あくまで「高」という文字のグループなのだ。包摂された個々の文字のどれであったのかという情報は失われている。符号化される前の文字は、明朝体の「高」だったかも知れないし、ゴシック体、丸ゴシック体、あるいは手書きの「高」だったかも知れない。「髙」が「高」と包摂されている文字コードならば、それは明朝体の「髙」だったかも知れないのである。実際に画面に表示されるときは、これはフォントの選択に依存して決まることになる。

つまり、最初の問題に戻れば、「髙」と「高」が包摂されている文字コードにおいて、印刷や画面表示に際してどのような文字が「使える」かは、ひとえにフォントに依存した問題であって、文字コードは関知しない部分なのである。従って、文字コードが文字をどこまで細かく指定できるか、というのはどこまでを同じ文字と見做すか、すなわち包摂をどのように決めるか、という部分にかかっている。これが包摂が文字コードの本質的な部分であるという所以である。

グループにどのような番号を付けるか（符号化文字集合）や、どのようにしてビット列に

[*10] ビット列は便宜上 16 進数で表すことにする。

変換するか（符号化形式）は、実用の上では大きく関わってくる（例えば符号化形式として UTF-16 を使うか UTF-8 を使うかではプログラムでの処理で様々な違いが生じる）が、これらはどのように決めたとしても、それらは同じ $U/\sim$ をどのように表現するかという差異であって、喩えるならばある物理系を表すときに座標系に直交座標系を使うか極座標系を使うかという差異のようなものであり、本質的なものではない。

以下、このような文字コード観のもとで、いくつかの現実的な問題はどのように考えられるのかをみていきたいと思う。

例示字形について

前節では文字コードの抽象的な定義をしたが、実際の文字コードの規格表はどうなっているのだろう。

基本的には、大量の文字が並べてあり、そこにそれの対応する番号がわかるように書かれているはずだ。文字コードによっては、包摂がどのように行われているのかを明示する資料が添えられていることもあるらしい。そして、その番号をどのようにビット列に変換するかは、ある場合には規格表が変換規則を定めていて、ある場合には実装者が好きにするようである。

ここで、並べてある文字を例示字形というのであるが、これは前節で述べた $U/\sim$ の代表元にあたるものである。つまり、この例示字形によってその文字のグループを示し、それに対応する数字を添えることで、符号化文字集合を示しているのだ。

ここで注意すべきは、代表元はそのグループの中のどれをとっても良いと言うことである。例示字形にどのような文字が選ばれていようが、指し示される文字のグループは同じだし、従って文字コードは同じものになるはずなのである。

しかしながら、現実では、所謂 JIS コード（JIS X 0213）では 2004 年の改訂の際に、いくつか（168 字）の例示字形が変更され、それによって、例えば「Windows XP と Windows Vista では表示される文字の字体が異なる」といった混乱が生じた。何故このような事態になったかを考えると、各フォントベンダーが例示字形に近くなるようにフォントを作って販売していたからに他ならない。本来包摂されている文字の中ならどのような字形を選んでも JIS の規格には準拠していることになるはずであるのに、例示字形という便宜上のものに惑わされてそれに従おうとしたがために起きたことであると言える[*11]。

[*11]かといってこの件に関してフォントベンダーを責めるべきかといえば、微妙な話である。「他のフォントと字形が違う」などと言われて面倒を被るのを避けるために規格表の字形に沿うようにしたという判断は十分理解できる。一般的なユーザーが文字コードについて精通していないためにフォントベンダーに対して文句を言うかも知れないことを問題とするべきかもしれない。或いは例示字形の変更という意味の無いはずの行為をした JIS が、或いはその原因たる「表外漢字字体表」をつくった政府の国語審議会が問題であったとも言える。しかしその表外漢字字体表ももとはといえば 83JIS により生じた問題への対策であって、それは……というふうに、単純に誰が悪かったとは言い難い問題である。

そもそも、表外漢字字体表の目的がフォントを伝統的な字体にすることであったのならば、その目的は達せられたのであり、83JIS からの異常な状態がようやく終熄した喜ばしいことであるとも言える。このあたりの字形の問題は、もともと漢字という文字体系がもつ複雑さ、当用漢字–当用漢字字体表–常用漢字などと続いてきた政府の漢字政策のグダグダさ、JIS 規格が工業規格であるがために文化的事情よりも工業的事情を優先して決まってしまったこと、昔コ

文字コードの変換について

ある文字コードで得られたビット列から別の文字コードのビット列に変換することは、広く行われている。UTF-8 で書かれたテキストファイルをテキストエディタで開いて、UTF-16、あるいは Shift-JIS で保存する……たったそれだけの簡単なことである。

しかし、ここで考えたいのは、その変換は妥当なものであるかという問題である。

まずは、符号化文字集合が同じ場合を考える。このとき、ビット列間の変換は次の図 4 のように書くことが出来る。

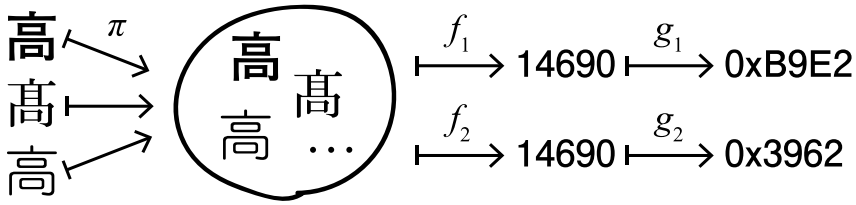


図 4　妥当な文字コード変換

図から分かるように、(B_1, g_1) と (B_2, g_2) という異なる符号化形式から生成されたビット列の間には、相互に移りあう写像をつくることができる。つまり、全単射 $g_2 \circ f_2 \circ f_1^{-1} \circ g_1^{-1} : B_1 \to B_2$ により、$b_1 \in B_1$ と $b_2 \in B_2$ が 1 対 1 に対応する。この意味で、このような場合の変換（例えば euc-jp と iso-2022-jp の間や UTF-8 と UTF-16 の間の変換）は妥当なものと言える[*12]。

次に、符号化文字集合が異なる場合であるが、先に結論を言えば、このような場合は文字コード変換が妥当でない場合がある。次の図 5 をみてほしい。

これは、髙と高を包摂する文字コードと包摂しない文字コードの間での変換を表す図である。図をみれば、これがとても妥当な変換とはいえないことが一目瞭然だと思う。

髙と高を包摂する文字コードでのビット列 B9E2 は、髙と高のどちらを指しているのか分からないのである。それを、髙と髙を包摂しない文字コードのビット列に変換しようとすれば、そのどちらかを勝手に推測して行うことになる。この 2 つの文字コード体系では包摂規準が異なる、すなわちそれぞれのビット列が持っている情報の量が異なるのだから、妥当に変換できないのは当然と言える。

一般的に JIS と Unicode などといった異なる体系では包摂規準が異なるから、そのような体

ンピュータメーカーが商業的な都合から独自に JIS を拡張した文字コードを乱造したこと、そして全世界の文字を 16 ビットで符号化しようとした初期 Unicode の幻想による CJK 漢字統合、Unicode の原規格分離規則による CJK 互換漢字の存在など、たくさんの歴史的な事情が複雑に絡み合って生じたものであり、まさに盤根錯節という状況にある。私にはもはや人間の手に負えるものであるとは思えないし、この歴史は悲劇という外ないと思う。

[*12]正確には、これが妥当と言えるのは $(f_1^{-1} \circ g_1^{-1})(b_1) = (f_2^{-1} \circ g_2^{-1})(b_2)$ となる場合であるから、厳密に符号化文字集合が同じでなくとも、対象の文字が二つの符号化文字集合の共通部分に含まれていれば良い（当然その文字に対する包摂規準が一致することはこの必要条件となる）。

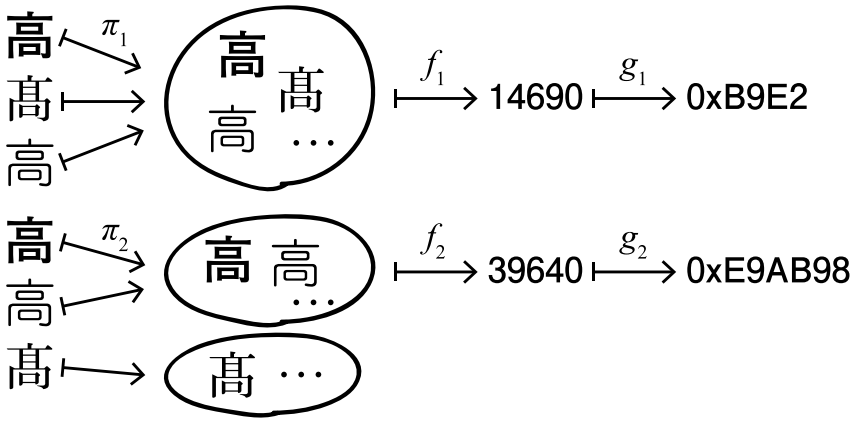


図 5　妥当でない文字コード変換

系をまたいだ文字コードの変換は原理上不可能であると言える[*13]。文字情報を電子的に扱う際には、どの文字コードを選ぶかを最初に決めることが重要である。

何を以て同じ文字と見做すべきか

さて、ここまで包摂がうまくできると仮定した場合の話をしてきたが、ここからは包摂それ自体の問題を考えたいと思う。すなわち、「どういうものを同じ文字と見做すべきなのか」、或いは「どこまでを同じ文字と見做すかという線を引くことは原理上可能なのか」という問題である。

最も直感的なのが、形による包摂である。つまり、同じ形の文字を同じ文字と見做すのである。

とはいっても、現実に存在する文字の一つ一つは違う形を持っているのである（例えば前述の 2 つの手書きの「あ」)。ということは、文字のある部分を抽象して、同じ文字と見做すたづきとしていることになる。これは、恐らく文字の骨骼であろうと考える。骨骼というのは、文字の各部分での線の太さを捨象して得られる、線のつながり、曲がり方、位置などの情報である。

物理学では物体があるとき、その物体をある放り投げたときの挙動を知りたければ質量や重心の位置といった力学的な性質に注目する。氷水に突っ込むとどうなるか知りたければ、温度や比熱などの熱学的な性質に注目する。ちょうどそれと同じように、文字の伝える意味[*14]を考える場合は骨骼に注目するのであるが、一方文字のもつ表情のようなものを考える場合は骨骼以外の部分、文字の太らせ方などに注目し、それを「書体」と呼び、「同じ文字のグループ」に

[*13]例えば Unicode は JIS などとの間に「ラウンドトリップ」を保障していて、相互変換が可能なようにみせているが、まさに今の例のようなもので、「もしたまたま規格表と同じ字形を選んで実装されたフォントを使っていれば見た目が変化しない」だけの変換であり、妥当なものではない。詳細は [1] を参照されたい。

[*14]後の論を考えるとこの書き方には語弊があるが、他に良い言葉が見つからなかった。

分けたのと同じように「同じ書体のグループ」に分けることをするのである[*15]。

なぜ骨骼であるのかという理由であるが、それは恐らく文字の多くがペンや筆といった、一次元的に線を引く道具で書かれることにより発展していったからではないかと思われる。「ある文字を書こう」と人が考え手を動かすとき、筆の動く軌跡をまず意識するのであり、太さがどうなるかといったことは二の次になるだろう。そういう事情が、骨骼を同じ文字の判断基準としているのではないだろうか[*16]。

さて、このような判断方法を用いた場合、文字コードではどのようなことが起きるか。考えられる問題のひとつは、「別の文字なのに同じ形」の場合どうするのか、ということである。

例えば、日本最古の図書館と言われる芸亭(うんてい)の「芸」の字は、書物の虫食いを防ぐための香草の名前であり、藝の新字体である「芸」とは全く異なる由来・意味の文字である。中国大陸での「機」の簡体字が「机」となって日本語の「机」と同じ形になっていることも有名だ。西洋の文字に目を向けても、キリル文字の「ｐ」は、ラテン文字の「p」とは異なる由来のものであり、むしろラテン文字の「r」にあたるものだ。ギリシア文字の「Α」とラテン文字の「A」も形での区別は難しい。

このような場合、どうすればよいであろう。

理に適っているのは、どのような意味や由来や音を持っているものであれ、同じ形のものはすべて包摂してしまうことだと思う。なぜなら、「芸」「p」「机」などの文字がどのようなものであるかは、「文脈」という曖昧なものに依存して判断するほかないからである。そこには、読む人間の価値判断が大きく関わってくるのである。

例えば、「芸」という文字ひとつだけを見せられたとき、それがどちらの意味なのかを判断するのは不可能である（同じ形なのだから)。「芸亭」「芸術」などといった並びの中に現れることによって初めて判断が可能になるのだ。「p」の場合も、それがたくさんのキリル文字に囲まれた中にあるか、あるいはラテン文字の中にあるかで判断するほかないだろう。一文字だけを独立に見てその違いを判断することはできず、人間が経験に基づき周囲の状況判断するほかないのである。

世の中にはOCRソフトというものがあるが、あれで読み取る例を考えてみてもらいたい。私は試しに使って平仮名と片仮名の「へ」がきちんと区別できないのをみてこれは使い物にならないなと思ったことがあるが、考えてみれば同じ形の文字を区別する文字コードで符号化しようとしているのがそもそもの問題なのである。平仮名と片仮名の「へ」を区別しない文字コードではそのような問題は生じない。そして、（もしそれが本当に同じ形と見做されるのならば）印刷するにあたっては、包摂してしまうことによる問題は一切生じないはずである。なぜなら、人は印刷された文字列をみて、勝手に意味や音の区別をしてしまうことができるのだ

[*15]音声言語の場合、「同じ音」が切り分けられ国際音声記号（IPA）で記述されるが、そこには含まれない「声色」という見方もあるのである。書体はちょうど声色に相当するであろう。

[*16]このあたりの考察（ひいてはこの稿全体）は、漢字文化圏での文字に対する印象により支えられていることが否めない。粘土板に型を押し付けて刻まれた文字などでは事情が異なるであろう。但し、文字に対する普遍的な理論を求めるのならばその理論が漢字に対して成り立っていることは必要条件であるから、出発点として身近な文字を考えることを許してもらいたい。

から。

しかし、これでは問題ではないかという感じる人もいるだろう。コンピュータ上での検索や文章の処理のときに極めて不便ではないか、と。これはある意味当然の問題だ。上の文字の形のみで判断するという方法は、あくまで印刷や表示という目的にのみ適する形であって、機械的に（それこそ OCR ソフトにでも）変換できる範囲のものである。コンピュータ上で「見た目を再現する」以上の層での文字に対する操作をしようと考えるならば、意味をも含めた文字の包摂を考えなくてはならなくなるはずなのである。

つまり、文字入力の際に人間が自分が入力する文字はどのような意味の文字なのかを考えることにするのである。上の例の「芸」「p」「机」もすべて意味や由来によって別の文字と見做し、入力する人間がそれをきちんと入力し分けることに期待するという在り方である。これは人間が入力するときにはたいして負荷の掛かる作業ではない（OCR ソフトで処理するためには、人間と同様の判断の出来る人工知能の存在が前提となるので難易度が上がる）。そして、こうして意味の判断結果をコンピュータ上で扱うことにすれば、コンピュータでの処理は遙かにやりやすくなってくるのである。

この場合、見た目の文字以上の情報を文字コードに含ませることも可能になるのだから、突き詰めていくと次のようなことが可能かも知れない。例えば日本語の「に」にはたくさんの意味が存在する。それを「状況を表す格助詞」「目的を表す格助詞」などの場合に分けて別の文字と見做し、その基準によって過去の古典作品を入力したデータを作る。あるいは時枝文法で、考えて入力する際に辞の「零記号」をすべて入力していく。このようなことをすれば、あとでコーパスとして利用する際にとても便利になるのではないだろうか。

もちろん、このような見方にも問題がある。意味や由来の区別をどこまで細かく判断するかということである。漢字の由来に対する説など、漢和辞典ごとに書いていることは違う。助詞の働きに対する区別などを考えればもっと大変である。どの人の説を正しいとして、文字コードをという一度決めたらなかなか変えづらい規則を作るのか。それに伴う困難は推して知るべしである。

ただ、このような困難は意味ではなく形に基礎を置いた場合も同様である。先の例でも挙げた「瀬」の字体の差などは、普段は気にしないかも知れないが、昔の文献を正確に電子化したいときや、地名や人の名前を表す場合は区別したいときもあるだろう。必要とする包摂規準は、人によって、そして同じ人でも場合によって異なるのだ。

また、注意しておきたいのは、意味による見方をする場合は「裏」と「裡」など、見た目は大きく違っても同じ成り立ちの文字である場合は包摂することがあり得るということである。検索の際の利便性などを考えれば包摂してしまった方が便利であろうが、その場合印刷では困ることがあるかも知れない。

さて、ここまで二つの視点を考えてきたが、現実の文字コードがはどのようになっているであろう。どうもこの二つの視点が中途半端に混ざり合っているものがほとんどであるようである（このような視点の別はあまり意識されていないらしい）。

代表例として Unicode を考えてみると、形が同じものは同じ漢字と見做す CJK 漢字統合の

規則により、「机」は中国語簡体字のものも日本語のものも区別されていない。一方で、「p」のようなものは区別されている。このようなことは原理的には不合理であり、現にたくさんの問題を生んでいるように見える。

一つの顕著な例は、国際化ドメイン名に於ける異体字の問題である。各論的な事情は [4] を参照して頂きたく思うが、「異体字を区別しない」という方針は極めて面倒な問題を引き起こしている。これは、「形による区別」により得られた文字コードで意味・由来の問題を扱おうとしているからに他ならないと思う。このような文字コード上で漢字に対して異体字を考えるのは土台無理な話ではなかろうか。

この論の結びとして私は次のようなことを提言したい。すなわち、「それは何のための文字コードなのか」ということを考えてほしいのである。上で述べたように、印刷が目的であれば、形により区別すれば良いし、コンピュータ処理が目的であれば意味により区別した方が良いだろう。この二つの両立は上で述べた「芸」の例や「裡」の例などを見るように、うまく行うことは難しいと思われる。今の文字コードの包摂規準を決めた際には、包摂を細かくした方が良い印刷目的の形による区別の立場と、包摂を荒くした方が良いことの多い電算処理目的の意味による立場という二つの立場の衝突がおき、中途半端なものが出来てしまったのではないだろうか。

プログラミング言語でも C++ を使った方が良い場合と、Ruby を使った方が良い場合があるだろう。画像なら PNG と JPEG では活躍できる場所が違うはずだ。

それと同じように、文字コードも目的に応じて使い分けるべきではないだろうか。小説を書くとき、人の名前を扱うとき、古文書をテキストデータに写すとき。それぞれの場合により、それに適した文字コードを選んで用いる、それが幸福な在り方ではないであろうか。

どんなときにも適した万能の文字コードを作る、そういった思想は、少なくとも私には破綻しているように思われるのである。

参考文献

[1] 矢野啓介、プログラマのための文字コード技術入門、技術評論社、2010。

[2] Florian Coulmas、斎藤伸治訳、文字の言語学——現代文字論入門、大修館書店、2014。

[3] 時枝誠記、国語学原論 上・下・続篇、岩波書店、2007。

[4] 安岡孝一、国際化ドメイン名における「堺」と「界」、`http://kanji.zinbun.kyoto-u.ac.jp/~yasuoka/publications/2014-03-14.pdf`、2014。

[5] 松本幸夫、多様体の基礎、東京大学出版会、1988。

[6] 田崎晴明、統計力学 I・II、培風館、2008。

チェックデジットあれこれと Damm のアルゴリズムの素晴らしさ

tetsu またの名を Tetsu=TaLow Ver.4

ソフト屋ならば、そして特に業務ソフトに関わる羽目に陥った者であれば、人生においてチェックデジットを計算するコードを書く機会はたまに発生するだろう。一方で人生においてチェックデジットの計算式を考えないといけない機会はそんなに多くあるわけではない。というか、私には幸か不幸かこれまでない。だが、もしこの先にそんなレアな体験ができるのであれば、私は Damm のアルゴリズムを必ず検討する。あなたにも、強く勧める。本稿は Damm のアルゴリズムの素晴らしさを伝えるためのものである。

チェックデジットとは何か

チェックデジットとはそもそも何か。一般には誤り検出符号の一つと考えるのが妥当だろう[*1]。コンピュータ屋にはおなじみのパリティビットやチェックサムや CRC と同じである。あるデータに対し、ある種の演算をした結果を付与しておき、入力や通信にあたって再度その計算を行うことにより誤りを検出するものである。このような誤り検出符号のうち、人が手入力することを前提とした 10 進整数値に対しある計算を行って 1 ケタの冗長な数字を付与するものがチェックデジットと呼ばれているように思う。

一番身近で確かめやすいチェックデジットは、商品についているバーコードのものである。これは日本では JAN コード、欧州では EAN、米国では UPC と呼ばれる共通商品コード[*2]を表しており、最下位の桁がチェックデジットであるがその計算式はいずれも同じである。チェックデジットを除いた下位から n 桁目の数字を P_n、また重み Q_n を 3（n が奇数の時）または 1（n が偶数の時）とした時に、チェックデジットの計算式は

$$10 - \sum_n P_n Q_n \bmod 10$$

[*1] まれに番号の偽造防止を目的に挙げる場合があるが、実際には多くのチェックデジット計算式は公開または解析されてしまうので、偽造に対して耐性があるとは思われない。

[*2] 現在では Global Standard One(GS1) と呼ばれる国際機関が管理し、日本では一般財団法人流通システム開発センターが管理している。

である（X mod 10 は X を 10 で割った余りを表す）。ただし結果が 10 となったときは 0 とする[*3]。

例えば私が今、セブンイレブンで買ってきて食っているアタリメの JAN コードは 4902784222523 である。このチェックデジット 3 は以下のような計算で求められる。

元の数字 P_n	4	9	0	2	7	8	4	2	2	2	5	2	3
重み Q_n	1	3	1	3	1	3	1	3	1	3	1	3	
その積 P_nQ_n	4	27	0	6	7	24	4	6	2	6	5	6	合計 97

$10 - (97 \bmod 10) = 3$ なので、チェックデジットが一致していることが確かめられる。

このようにチェックデジットを付けておくことで、JAN コード入力時に 1 桁の数字の入力誤りをしてもチェックデジットの値が変化するので、誤りが検出できる。偶数桁については Q_n が 1 なので自明だろう。奇数桁についても Q_n が 3 であり、九九の 3 の段の下一桁がきれいに 0,3,6,9,2,5,8,1,4,7 とバラけていることから、やはりチェックデジットの値が変化して検出できると判る。

このチェックデジット計算式（mod10 weight 1-3 などと呼ばれる）は、他にも書籍に統一的につけられた国際標準図書番号である ISBN などで使われている。

ところで、人間の数値入力の誤りは 1 桁の入力誤り以外にもさまざまなパターンがあり得る。各種の調査によれば最も多い誤りは上記で検出対象になった 1 桁入力誤りである。他に 1 桁少ない、1 桁多いというものも多いが、元の符号が固定長であれば機械的に誤りを検出できる。次に多いのは 2 桁入替誤り、つまり隣接する 2 つの数字 ab の順番を入れ替えて ba と入力してしまうものである。これ以外にも誤りのパターンはいくつかあるが、割合としては少ない。なので、チェックデジットは少なくとも 1 桁入力誤りと、2 桁入替誤りが検出できることが望ましい。

JAN コードで使われているチェックデジット計算式で偶数桁と奇数桁に違う重みをつけているのは、この 2 桁入替誤りの検出を狙ったものである。これは一見ある程度はうまくいっているように見えるが、2 つの異なる 1 桁の数字 a,b について、$(3a+b) \bmod 10 = (3b+a) \bmod 10$ となる場合、この 2 つの数字の入れ替えについては誤り検出ができない。この式は $2(a-b) \bmod 10 = 0$ なので、a と b の差が 5 の時は誤り検出に失敗することがわかる。実際、先のアタリメ JAN コードは、左端の 49 や、左から 4,5 桁目の 27 を入れ替えて入力してしまった場合にはチェックデジットでは検出できない。例えば 27 を入れ替えると以下のようになる。

元の数字 P_n	4	9	0	**7**	**2**	8	4	2	2	2	5	2	**3**
重み Q_n	1	3	1	3	1	3	1	3	1	3	1	3	
その積 P_nQ_n	4	27	0	**21**	**2**	24	4	6	2	6	5	6	合計 **107**

差が 5 の数字の組み合わせは [0,5][1,6][2,7][3,8][4,9] の 5 通りなので、全ての 2 桁入替誤りの組み合わせ 45 通りのうち 5 通り、つまり 1/9 の確率で見逃されてしまうのである。

これを何とかできないであろうか。チェックデジットはいろいろな計算式が提案されている

[*3]（一財）流通システム開発センター「JAN コード、集合包装用商品コード（ITF コード）のチェックデジット計算方法」`http://www.dsri.jp/jan/check_digit.html` を参照。

が、それぞれの能力を見てみよう。

Luhn のアルゴリズム

チェックデジットの歴史はよくわからないのだが、実はごく初期にいきなり素晴らしいアルゴリズムが提案されている。IBM の Hans Peter Luhn が 1954 年に特許申請した Luhn のアルゴリズムである[*4]。

このアルゴリズムは、形式的には次のように説明されている。

1. 重み Q_n を、n が奇数なら 2、偶数なら 1 とする。
2. 各桁の数字と重みの積 P_nQ_n を計算する。計算結果が 2 桁になった場合はさらに各桁の和を計算する。
3. 計算結果を各桁について足して 10 で割り、余りを 10 から引いたものをチェックデジットとする。ただし結果が 10 になった場合は 0 とする。

例えば先ほどのアタリメ JAN コードだが、これのチェックデジットをこのアルゴリズムのものに入れ替えると、下記のように 6 になる。

元の数字 P_n	4	9	0	2	7	8	4	2	2	2	5	2	**6**
重み Q_n	1	2	1	2	1	2	1	2	1	2	1	2	
P_nQ_n	4	1+8	0	4	7	1+6	4	4	2	4	5	4	合計 **54**

チェックデジットは $10 - (54 \bmod 10) = 6$ である。

このアルゴリズムは、1 桁入力誤りは全て検出できる。偶数桁はもちろんだが、奇数桁についても P_nQ_n の計算結果が上位桁の加算も合わせると 0,2,4,6,8,1,3,5,7,9 とバラけるので入力を変えると必ずチェックデジットが変化する。

しかも、Luhn のアルゴリズムは 2 桁入替誤りのほとんどを検出することができる。入替誤りがチェックデジットを変化させない条件は 2 つの異なる 1 桁の数字 a,b ただし $a > b$ について、両方が 4 以下の時を表す $(a + 2b) \bmod 10 = (b + 2a) \bmod 10$ と両方が 5 以上の時を表す $(a + 2b + 1) \bmod 10 = (b + 2a + 1) \bmod 10$ はどちらも $a = b$ となり、$a > b$ と矛盾するので解がない。唯一、$a \geq 5$ かつ $b \leq 4$ の時を表す $(a + 2b) \bmod 10 = (b + 2a + 1) \bmod 10$ は、$(a - b + 1) \bmod 10 = 0$ より $a = 9, b = 0$ を解として持つ。つまり、2 つの入替誤り 45 通りのうち、44 通りまでは検出可能だが、唯一、0 と 9 の入替が検出できない。上記アタリメ JAN コードだと最上位から 2 桁目と 3 桁目がそれにあたる。

元の数字 P_n	4	**0**	**9**	2	7	8	4	2	2	2	5	2	**6**
重み Q_n	1	2	1	2	1	2	1	2	1	2	1	2	
P_nQ_n	4	**0**	**9**	4	7	1+6	4	4	2	4	5	4	合計 **54**

このように、Luhn のアルゴリズムは大変優れたチェックデジット計算式であり、実際クレジットカード番号をはじめとして今日でも広く使われている。私の知る限り、ISO に式が掲載

[*4] US Patent 2,950,048 - Computer for Verifying Numbers, Hans P Luhn, August 23, 1960.

され国際標準となった最初のチェックデジット計算式であろうと思う（ISO7812-1)。実務上も極めて有益だが、ただ、なぜ JAN コード（のもととなった UPC コード）がこのアルゴリズムを採用しなかったのかはよくわからない。推測だが、UPC コードが定められた 1970 年代には、まだ Luhn のアルゴリズムに対して特許が有効であったので、それを避けた可能性はある。とはいえ、UPC コードおよびバーコードの開発には IBM が深くかかわっていたようなので[*5]、やはり多少奇妙な話である。

とにかく、比較的単純なアルゴリズムで、かつチェックデジットを 0 から 9 の数字だけで表そうとするとき Luhn のアルゴリズムは比較的強力であり、特に「1 桁入力誤り」「2 桁入替誤り」に注目するとほぼ最強のチェックデジット計算法であるといえる。しかも米国の特許はとっくに切れており、現在ではパブリックドメインである。

それでも、2 桁入替誤りを完全に排除できるわけではない。特に 0 と 9 という組み合わせは、フルキーの上では隣り合わせになるので入替誤りを誘発しやすいという意味で不利である（テンキーならよいが)。それならば、2 桁入替誤りの全組み合わせを検出できるアルゴリズムを求めようとするのは人情だろう。

「11 で割れ」

実は 2 桁入替誤りを完全に検出する安易な方法がある。Luhn のアルゴリズムのように 10 で割るのではなく、11 で割る方法である。

ISBN は現在の 13 桁になる前、2006 年までは 10 桁だった[*6]（俗に前者を ISBN13、後者を ISBN10 と呼ぶ)。この時のチェックデジットは以下のような計算式に基づくものだった。再右端のチェックデジットを除いた数字に対し

$$11 - \sum_{n=1}^{9} P_n(n+1) \bmod 11$$

を計算する。ただし、答えが 10 になった時は数値の代わりに X とし、11 になった時は 0 とする。例えば若木民喜『神のみぞ知るセカイ』の 25 巻の ISBN13 は 978-4091245793 だが、これを ISBN10 に変換すると 409124579X である。最後のチェックデジット X は以下のように計算できる。

元の数字 P_n	4	0	9	1	2	4	5	7	9	X
重み $n+1$	10	9	8	7	6	5	4	3	2	
積 $P_n(n+1))$	40	0	72	7	12	20	20	21	18	合計 210

$210 \bmod 11 = 1$ であり、11 から引くと 10 になるのでチェックデジットが X となっている。

この計算式において、11 は素数であるので、任意の 1 桁の整数 a, b および重み $w(2 \leq w \leq 10)$

[*5] Wikipedia の “Universal Product Code” の “History” 参照。

[*6] International ISBN Agency “What is an ISBN?” `https://www.isbn-international.org/content/what-isbn`

に対して、$aw \bmod 11 = bw \bmod 11$ つまり $(a-b)w \bmod 11$ を満たす a, b は $a = b$ 以外存在しない。このことより、このチェックデジットは 1 桁入力誤りを確実に検出できることがわかる。

2 桁入替誤りにどれだけ対応できるか調べる問題は、同様に重み w ただし $2 \leq w \leq 9$ に対し、$(aw + b(w+1)) \bmod 11 = (bw + a(w+1)) \bmod 11$ を満たす a, b, w を探す問題に帰着する。これは $(b-a) \bmod 11 = 0$ を満たすことを意味するので、やはり $a = b$ 以外存在せず、全ての 2 桁入替誤りを検出できることがわかる。

このように ISBN は 10 桁の時代には、すべての 1 桁入力誤りと、全ての 2 桁入替誤りを検出することができていた。それが 2007 年からの 13 桁への拡張の際に、JAN/EAN/UPC コードとの互換性から、2 桁入替誤りについて完全には検出できないチェックデジットに置き換えられてしまった。とはいえ、ISBN10 はチェックデジットに X という、11 番目の記号を必要としたことから嫌われた面もあるので、仕方がないところであろう。

なお ISBN10 は 10 桁だから重みが 10 までで済んでいたが、これ以上の桁数になるとこのまま素直にこの計算法は拡張できない。そこで一般には、次のようなチェックデジット計算式が知られている（重みが 2 から 7 まで繰り返し変化するので一般に mod 11 weight 2 to 7 などと呼ばれる）。

$$11 - \sum_n P_n(((n-1) \bmod 6) + 2) \bmod 11$$

6 桁ごとに重み 2 と 7 が隣合うところが出るが、これに対しても $(2a + 7b) \bmod 11 = (7a + 2b) \bmod 11$ より $5(a-b) \bmod 11 = 0$ となり、やはり a, b が 1 桁では $a = b$ 以外の解がない。なので、この状態でもすべての 2 桁入替誤りが検出できる。

このように、11 で割るチェックデジット検出法は大変優秀なので広く使われている。変わったところでは、大学入試センターで使われる受験番号や試験会場コードなどの末尾についている記号は、11 種の剰余に対応する 11 種のアルファベットがチェックデジットならぬチェックアルファベットとして使われている[*7]。

しかしこの方法は、チェックデジットがどうしても 11 種類になってしまう。なので、1 桁の数字だけでチェックデジットを表現することができない。数字だけで表現できるようにチェックデジットを 2 桁にしたり、あるいは発番時にチェックデジットが X になるような番号を使用しないなどの対応をとって逃げる場合もあるが、前者は冗長に過ぎるし、後者は番号空間を圧迫してしまう。やはり 10 種の数字だけで、全ての 1 桁入力誤りと 2 桁入替誤りを検出したいものである。

そして Damm アルゴリズムへ

10 種の数字だけで表されるチェックデジットは、単純な四則演算や剰余算では全ての 2 桁入替誤りを検出できるようになりそうにない。そこで議論をもう少し一般化したらどうなるだろ

[*7] A,B,C,H,K,M,R,U,X,Y,Z がそれぞれこの順に、式内の剰余算の結果である 0 から 10 の値に対応している。

うか。チェックデジットの値を、初期値を C_0 とし、各桁の数字 P_n に対して $C_n = f(C_{n-1}, P_n)$ なる計算を繰り返した結果得られたものととらえたらどうだろう。簡単のため C_0 は 0 に固定して考える。各 C_k も 0 から 9 に制限して考えると、要は各桁の計算時に関数 f が入力 P_n に対していつでも全単射になっていれば、1 桁入力誤りは検出できることがわかるだろう。あとは $f(f(C_{n-2}, P_{n-1}), P_n) \neq f(f(C_{n-2}, P_n), P_{n-1})$ が常に成り立てばよいはずだ。

ここで、群論を持ち込むと解決できるというアイデアが現れた。上記関数 f を位数 10 の群の上の二項演算 $*$ で表現すると、$C_n = C_{n-1} * P_n$ である。この演算を繰り返して得られた C_n の値をチェックデジットとし、群をうまく調整して 2 桁入替誤りの際に演算結果が異なるように調整できれば求める性能が得られる。

最初にそれを実現したのは、オランダの数学者 Jacobus Verhoeff であり、1969 年のことである[*8]。彼が提案したアルゴリズムは Verhoeff のアルゴリズムと呼ばれる。彼がその実現のために導入したのは位数 10 の二面体群 (dihedral group) である。ただ、残念ながらこのアルゴリズムは二面体群だけではなく、誤り検出のために内部状態を「かきまぜる」変換表を必要とした。そのため結果的に計算は $C_n = f(C_{n-1}, P_n, n)$ と桁位置毎に変化させるように構成されており、アルゴリズムが複雑になってしまった。Verhoeff のこの研究は素晴らしいものではあるが、実用という意味ではやや難がある[*9]。

ところが、この問題を解決したのがドイツの数学者 H. Michael Damm である。彼が 2004 年に博士論文[*10] で提示したのは、群ではなく、準群 (quasigroup) である。

群は定義上、演算に関して単位元と逆元が存在し、結合則が成り立つが、準群 $(Q, *)$ は演算の逆元だけが存在すればよく、交換則や結合則は壊れている。厳密には、積 $*$ が定義される集合であって、任意の要素 a, b に対して $a * x = b, y * a = b$ が成立する要素 x, y がただ 1 組存在するものである。この準群 $(Q, *)$ が $c, x, y \in Q$ に対し $(c * x) * y = (c * y) * x$ が $x = y$ でのみ成り立つとき、弱い全反対称 (weakly totally anti-symmetric) と呼ばれる。Damm が示したものはさらに $x * y = y * x$ が $x = y$ でのみ成り立つ全反対称準群の構成法だが、結果として位数 10 をはじめとしてほぼ任意の位数の弱い全反対称準群が構成できるようになった。位数 10 の弱い全反対称準群はそのまま、チェックデジットの計算に用いれば、その定義から、$P_{n-1} \neq P_n$ ならば $(C_{n-2} * P_{n-1}) * P_n \neq (C_{n-2} * P_n) * P_{n-1}$ となるので、2 桁入替誤りをしたときは必ず C_n の値が変化し、検出可能になることがわかる。Damm は位数 10 の弱い全反対称準群の中からさらに、英語で発音が似ていることからありがちな 2 桁数値の誤り（例えば 13 と 30 を取り違えるなど）も発見できる組み合わせを選んでチェックデジット計算用に提案している。

そこで提案された表の一つが Wikipedia に掲載されている。それがズバリ、これである。

[*8] Verhoeff, J. (1969). Error Detecting Decimal Codes (Tract 29). The Mathematical Centre, Amsterdam.

[*9] とはいえ、実際には後述するようにインドの国民 ID において実用化されている。

[*10] Damm, H. Michael,“Total anti-symmetrische Quasigruppen (Dr. rer. nat.)”, Philipps-Universität Marburg, 2004.

		P_n									
	C_n	0	1	2	3	4	5	6	7	8	9
	0	0	3	1	7	5	9	8	6	4	2
	1	7	0	9	2	1	5	4	8	6	3
	2	4	2	0	6	8	7	1	3	5	9
	3	1	7	5	0	9	8	3	4	2	6
C_{n-1}	4	6	1	2	3	0	4	5	9	7	8
	5	3	6	7	4	2	0	9	5	8	1
	6	5	8	6	9	7	2	0	1	3	4
	7	8	9	4	5	3	6	2	0	1	7
	8	9	4	3	8	6	1	7	2	0	5
	9	2	5	8	1	4	3	6	7	9	0

この表は、準群上に定義された二項演算の乗積表である。対角要素が 0 になっていることに注意されたい。これを用いたチェックデジットの計算は次のようなものだ。C_0 は 0 とする。各桁 P_n は、これまでとは逆に左端、つまり最上位桁から順に $P_1, P_2, \ldots$ としよう（その方が計算しやすい）。左から順に P_n に従って上記の表に従いながら C_n を更新してゆき、残った C_n をチェックデジットとして右端に付加するとよい。検査時は、チェックデジットを含めたまま同じアルゴリズムで計算してゆくと、対角要素が 0 なので、チェックデジットが同じであれば必ず計算結果が 0 になり、容易に検査できる。

例えば『神のみぞ知るセカイ』25 巻の ISBN10 のチェックデジットを Damm のアルゴリズムに置き換えると、4091245798 となる。これを確かめてみよう。

元の数字 P_n		4	0	9	1	2	4	5	7	9	**8**
C_n	0	5	3	6	8	3	9	3	4	**8**	**0**

上記 C_9 が 8 であるので、これが計算されたチェックデジットである。これが P_{10} と一致することは一目瞭然だが、かまわずそのまま Damm のアルゴリズムでの計算を続けると、結果が 0 になることでも結果が正しいことが確かめられる。

どうだろうか、素晴らしくないだろうか。さきほどの乗積表をまじまじと見つめると、表は各行、各列とも要素に重なりのないラテン方格であるので、1 桁入力誤りはもちろん検出できると判る。また、どの行 (C_{n-1}) から始めても、2 つの値 a,b をこの順に入力した時と逆順つまり b,a と入力した時ではチェックデジットが異なっている。つまり、全ての 2 桁入替誤りを検出する能力があるのである。しかも、13 を入力した時と 30 を入力した時や、14 と 40 を入力した時はチェックデジットが変化するように表が作られている。

この Damm のアルゴリズムは、10×10 の表ひとつを保持するだけで実装でき、内部の処理も 1 桁分の内部状態と表引きによる更新と極めて単純である。唯一の難点は、この表自体が複雑なのでチェックデジットを手計算で検算するのが表なしには困難であることくらいだろうか。それでも、複数の表引きを必要とする Verhoeff のアルゴリズムよりはずっと楽である。

この Damm のアルゴリズムは、このような素晴らしい性質を持ちながら、しばらく日の目

を見ることがなかった。それにはいくつかの原因があるが、この乗積表が一意に定まるわけではないので、互換性のために標準化された乗積表がなかったことも原因であろう。ところが、2012 年に Damm のアルゴリズムが Wikipedia で紹介されて以来、この記事中で挙げられている乗積表を基に急激に普及が始まったようである。実装は多数の言語ですでに行われており、それが Wikibooks に掲載されている[*11] ので、必要ならばそこからコードをコピペすればよい (Wikipedia の記事から辿ることができる)。Perl, Ruby, Python についてはすでにパッケージもあることを確認している。

おわりに

どうだろうか、Damm のアルゴリズムの素晴らしさが伝わったであろうか。

実はこの記事を書くことにしたきっかけは、マイナンバー制度で導入された個人番号や法人番号のチェックデジットの式があまりにもお粗末だからである。それについては別稿[*12]に書いたのでぜひ見ていただきたいが、本当に情けなくなる話である。2009 年からはじまったインド版のマイナンバー制度 (Aadhaar) が Verhoeff のアルゴリズムを、2014 年に始まったシンガポールの特許番号制度が Damm のアルゴリズムをそれぞれ採用していることとは随分と差があるのではないか。

ということで、皆さんももし、自分でチェックデジットアルゴリズムを決めることができる機会があれば、是非 Damm のアルゴリズムの採用を検討してほしい。Dumb なアルゴリズムでも Damn なアルゴリズムでもなく、Damm のアルゴリズムを、である。

[*11] https://en.wikibooks.org/wiki/Algorithm_Implementation/Checksums/Damm_Algorithm

[*12] 上原哲太郎「マイナンバーのチェックデジットについて」、デジタル・フォレンジック研究会第 404 号コラム。https://digitalforensic.jp/2016/03/14/column404/

ダサ T のすゝめ

hakurin

はじめに

以下のテキストは極めて高い確率であなたの人生にとって有益なものではありません。ムダな時間を消費したくないという方はページをシュシュッといい感じにめくって下さい。僕以外の部員は皆有益な記事を書いていると思います（たぶん）。人生における無益な時間消費を許容できる心の広い方は、よければ以下の文字列に目をお通しください。

この本を手に取って頂きありがとうございます。KMC 2 回生、id は hakurin と申します。趣味は

- Twitter for Android を開いて無限にフリック
- 適度なお絵描き
- Amazon でめぼしい商品をポチって日本経済を回す
- 自主全休

みたいな感じです。特段珍しい趣味は持ちあわせていません……

一応 graphics にも所属していて、KMC ではよく世間で言う「グラフィッカー」のような仕事をさせてもらっています。KMC 部員でありながらプログラミング類はからっきしなので、周囲を情報強者に囲まれ戦々恐々とする毎日です。

#dasai-t-shirt とかいうクソ channel

ところで、KMC の Slack-channel の中には極めて有益なものもあれば、同様に極めて無益なチャンネルも数多く存在します。その無益な channel の中でもとりわけ無益な channel、所謂“クソ channel of クソ channels”として不遜にも厳然と君臨するのが、#dasai-t-shirt チャンネルであります。

この channel では、ダサい T シャツ情報を見つけ次第（主に hakurin により）その報告がなされます。あの有名な「猫に雷が落ちてる T シャツ（よりにもよってフルグラフィック。じわる）」「宇宙空間を堂々闊歩する猫の T シャツ（これもフルグラフィック。じわる）」の情報も寄

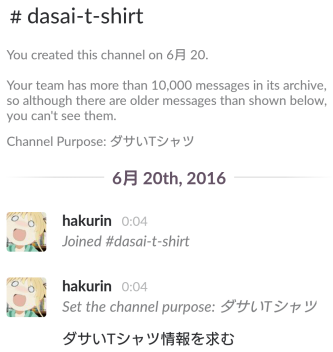


図 1　#dasai-t-shirt がこの世に生を授かった！ めでたい。

せられます。つい先日、新入部員の方もダサ T 情報を提供してくださいました。この channel は歴史こそ浅いものですが、いつの日かこれが KMC 部員のダサ T 情報共有のためのポータル的な channel となれば幸いだと思っています。誰が得するんだって感じですが。僕ですが。

さて、今回部誌に記事を掲載する機会を頂いたはいいのですが、如何せん趣味がどれも平凡な上に特筆すべきようなスキルも持ちあわせておりませんので、何を書いたらいいものかと思っておりました（〆切当日）。そんな時、このクソ channel がふと目に入ったのであります。

さあ、そういうわけですから、今回はダサ T シャツ収集癖を悪化させつつある KMC 部員達を代表[*1]し、ダサ T について筆を取らせていただこうと思います。

ダサ T（ダサい T シャツ）の魅力

先ほど僕の趣味を紹介させていただきました。そこに記したように、Amazon などのショッピングサイトでいい感じの商品を見つけるとシュシュッと購入してしまう癖が僕にはあります[*2]。最近特に熱を上げて購入しているものがさっきから文中で連呼している「ダサ T（ダサい T シャツ）」です。大学に行く際は大体、僕はこれらのダサい T シャツを適当に 1 枚選んで着ています。僕はダサい T シャツが大好きなんです。オシャレ（と世間で言われるよう）な服に 20,000 円とか 30,000 円とか膨大にお金をかけるよりも、高くてせいぜい 5,000 円程度のダサい T シャツを着て生活した方が遥かに低コスト。しかもダサ T シャツは着ると心が豊かになります [要出典]。そういう意味ではダサ T はローリスクハイリターンで、ファッション界のエコロジーリーダー（笑）なんです。そう思いませんか？ 思いますよね？ 思いませんか。

ダサい T シャツと呼ばれる中にはいくつかの種類があると思います。例えば、

[*1] 他にいるかはわかりませんが……。もしいらっしゃったら#dasai-t-shirt に join してください。より良いダサ T ライフを送らんと切磋琢磨しましょう！

[*2] この悪癖のせいで、楽天カードの利用可能残高を超えた買い物をしてしまって毎月支払いに苦労しています。毎月 7,8 万円ほど入ってくるバイト代のうち 6 割方はこの Amazon の支払いに消えてしまうんですね（残りは食費）（僕はラーメンが好きです）（体重は単調増加しています）。

- シュール系（先述の猫 T シャツなど）
- オタク系（アニメ・漫画のキャラなどが印刷されている）
- 原宿系ファッション T シャツ（きゃりー◯みゅぱ◯ゅっぽい感じのやつ）

などなど。ちなみに現在僕が所有しているのは「オタク系」のダサ T が多いです。シュール系ももっと集めたいですね。猫 T シャツが欲しい……。原宿系は話に聞くところによると、高い。高いくせにダサいって、なんだかなあ、って感じですね……。ああ、原宿系は「ダサい」ではなく「かわいい」か……。それなら高くても、まあ、うん……。

「こういうダサい T シャツってどこで売ってんねん」と（嘲笑を込めて）訊かれる事がまれにあるのですが、例えば時々ユ◯クロとかでキャンペーンをやって大々的に売りだしてますね。某兵衛なカップ麺や某ッコーマンな醤油とコラボしてたり。し◯むらとかでもやってるのかな？ ほかにも、個人でデザインを制作して販売する Web サイトなどもあります。自分で T シャツをデザインしてプリントしてもらえるサイトもあります。KMC の先輩の中にはそういうサイトを利用して自分でプリント T シャツをデザインして販売している人もいますね（ダサ T という枠からは外れるかもしれませんが）。

まあ、とにかく世には多種多様なダサ T が満ち溢れています。あんなダサ T やこんなダサ T、みんな違ってみんなダサいんですね。どんなイケメンでも着ればたちまちダサくなる。ダサ T の下に人間はみな平等です。

「ダサ T いいなぁ〜」と感情的になりながら文章を執筆しているので、ダサ T の“よさ”がテキストのそこかしこに散逸してしまっている気がします。ここでダサ T の魅力をまとめてみましょう。

- オシャレな服に比べて圧倒的に低コスト
- ダサさ故に精神が豊かになる
- 種類が豊富で、多種多様な方向性の“ダサさ”を楽しめる
- Web で簡単に手に入る
- ~~ダサ T で平等な社会を実現~~

あれ？ 文面だけではあまり魅力が伝わってこないですね……。

まあ、それもそのはずです。ダサ T の魅力はいくら言葉を並べ立ててみたところで説明しうるものではないのです。商品を見てもよくわからない。現物を着ないとわからない。百聞一着に如かず。ダサ T とはそういうものです。きっと。多分……。

はてさて、昨今の「ガールズ＆パンツァー」ブームに乗っかって、僕もガルパン T シャツを複数種類購入しました。2016 年春頃の話です。次の写真は実際の着用例です。これも 2016 年春頃の写真です。

ラーメンバカ特有のだらしない体型も相まってなんだかガンコなラーメン屋の親父っぽく見えますが、胸には “St. Gloriana Girls College” の文字と聖グロリアーナ女学院の校章が。僕は流行りのコンテンツにすぐに迎合する最低なオタクです。聖グロ好きです。特にダージリン様が格言をキメたあとのそのドヤ顔。他にはアンツィオ高校の校章が中央にドーンと配置された

図 2　オタク T シャツを着るオタク

T シャツや大洗女子の T シャツ、プラウダ高校の T シャツなどもあります。黒森峰女学院の T シャツはお金がなくて泣く泣く諦めましたが……。

このTシャツを着ていると、「それどこの大学のシャツなん？」と学友に訊かれることがありました。その時は返答に困ってしまいましたね……。「ァ、ァ……なんか……アニメのやつ……」みたいな……。

ダサ T シャツは外ではあまりおおっぴらに見せるべきものではないのかもしれませんね。他のひとを混乱させてしまいがちです。上着の下にさりげなくダサ T、アリだと思います。アリですよね？ よかった。あなたも最早ダサ T 愛好の士でしたか。

最後に

まだダサ T の沼におっかなびっくり片足を突っ込んだ程度の僕が書く文章ですから、ダサ T の魅力を読者の皆様にお伝え出来たかどうか、まったく自信がありません。

しかし、ダサ T の魅力は言葉だけでは伝えきれないのも事実だと思います。Web で、或いは店頭でダサ T を見かけたら、一度よく見て（もしくは手にとって）いただきたいのです。一見ダサい。何回見てもダサい。隅から隅までダサい。けれどもそのダサさの向こうには、あなたを虜にしてやまぬ魅力と、デザイナーの掲げる深遠な理想があるのかもしれません。

もしこのしょうもない記事を読んでダサ T に微塵でも興味が湧いたという方は、まずは Amazon で「T シャツ 猫」と検索しましょう。数々の極限にダサい T シャツがあなたを出迎えてくれます。そして願わくば、そこからダサ T の沼にズブズブとハマっていただければ至上の歓びです。一緒にダサ T 沼に溺れましょう。そして汚泥の中でダサ T にまみれた生活を始めましょう。

以上、ダサ T 沼のほとりからお送りいたしました。

続・サークルで初心者向けにプログラミングの話とかをする際にちょっと考えてみたいことについて発表した

pastak

こんにちは。はじめまして、おはようございます。pastak です。最近では KMC 4 回生にもなり、老害感が出てきたなと思いつつも、性懲りもなくこういう記事を書いている状態です。皆さんはお元気にお過ごしでしょうか。

はじめに

ちょっと前の話なのですが、KMC の例会講座で「『中級者』の僕たちは何が出来るのか、または『初心者』に語るときに僕が考えること」というタイトルで話をしました[*1]。

さらにその後、ブログにこの発表に関する記事を書いたところ[*2]、ご意見ご感想を頂いたりしたので、そのことなどを踏まえつつ、総集編という感じで部誌にコラムとして書こうという所存です。よろしくお願いします。

以下の文章はそのブログ記事に加筆修正した文章になります。ブログ記事自体を読んでいただく必要はありませんが、前述したスライドに関しては、先に目を通していただくか、読み進めながら適宜合わせてご覧になっていただくと良いかと思います。

モチベーション

講座やってくれと会長氏から言われて何について話すか考えていました。昨年はわくわくシンセサイザー入門[*3]というタイトルで Web Audio API について話したのですが、この時期に来る新入生はコードを書いたこともない人が多かったので、具体的な API などを紹介するので

[*1] 当時のスライドは `https://goo.gl/VmeM33` からご覧いただけます。

[*2] サークルで初心者向けにプログラミングの話とかをする際にちょっと考えてみたいことについて発表した `http://pastak-diary.hatenadiary.com/entry/2016/04/25/183733`

[*3] `http://www.slideshare.net/pastak/javascript-47327211`

はなくて何かしら精神っぽいエモさあるトークをしようという方針を立てました。

エモいトークということで、どういう学びを中心に組み立てようかと考えつつ過ごしつつ、発表が迫っていたある日。ちょうどスライドを作るかという気持ちでたまたま部室に居たら、若者たちが競技プログラミングを始める新入生に向けた C++ のスライドのレビュー会をしているところに遭遇しました。

何気なくふむふむ C++ はこうやって書くんだなぁと思いながら眺めてたら、「おまじない」をどうするかとか「どの書き方を教えるか」について多数決や話し合いで決めているのを見て、いやいや多数決はないでしょと思って、じゃあそういうことをスライドの主にして構築することにしようとして決めました。

このコラム内での「入門」について

このコラム内や元となったスライドやブログでも一律して「入門」という言葉を使っていますが、「入門」については色々なケースがあると思います。例えば、KMC のように初心者の集団に対して行う場合や、新入社員研修などの際に知識の総整理とスタートラインの調整を目的に行う場合、また新たなプログラミング言語やライブラリ、ツールなどの触り心地を確かめるための入門など……

このコラムでは主に KMC のように初心者の集団にある程度まとまった知識や体験を伝えるものとしての「入門」を扱います。今年度だと「JavaScript から始めるプログラミング 2016」などがそのようなものです。

ここで少しその他の入門についても考えておこうと思います。

新入社員研修などを考えてみると、新入社員はまず一度入社試験的なサムシングによって選抜されており、能力や知識の前提などもある程度共有されています。また、講師を務める人々も大半の場合は、短い時間で如何に業務に即した知識を体得できるかという部分に重点を置くという意志が共有されていることでしょう。

目的、特にゴールが共有されているというのはとても重要だと思います。生徒役も講師役もお互いに「業務に即した知識を得て業務に参画できるようになる」というゴールを共有していることで、生徒役も学ぶべきポイントの見立ても出来ますし、モチベーションの維持のための目標も明確です。講師役の人々も自分たちが提示すべき内容のレベルを検討すること、また受講者たちのレベルやモチベーションの把握も容易になると思います。

KMC のような有象無象が入部してきて、とにかく知見を精一杯伝授するという形式の場合でも、このようにゴールを設定することで少しはお互いの意識を明確に出来そうです。「みんなでゲームを作る 2016」なんかは NF でゲームを展示するという明確なゴールがあるので、モチベーションや必要なサポートなどの目論見を立てやすそうです。

ただ、レベルやモチベーションの一律化という点が KMC でやっているような形式では難しいのも事実です。KMC も選抜などをして真にプログラミングなどを学びたい新入生だけに絞れば、そのようなことも可能でしょうが、KMC は現在入部資格に制限はありません。そういっ

た制約を設けていないからこそ色々な人が集まっていて、様々なプロジェクトが立っては活動できている。それに惹かれて翌年多くの新入生が入部してくれるというサイクルの実現ができているのも事実だと思うので、これはなかなかムズかしい問題だと思うので、一旦横に置いておくことにしましょう。

どういう入門が良い入門？

スライドやこのコラム自体にも書いてないこととして「どういう『入門』が正解なのか」ということまでハッキリと踏み込んではいないということがあります。これは意図的にそうしています。何故なら、「入門」に関してこれが正しいとかこれが良いみたいなのはなかなか難しいと思っているからです。

先ほどの章にも書いていましたが、各々の環境がありテーマがあり、ゴールがあるはずなので、一概にこれが必ず良いとは言い切ることは難しいし、また危険なことだと思います。

スライド内にも書いていますが、例えばあるコミュニティや企業でやるときにはこれがベターだが、KMC ではこれがスタンダードになってる。でも KMC 内でも初心者に対して話す場合は簡単のために簡単な方を紹介ということがあると思います。

1 つ基準を上げるとすると、こういうのは話者やその入門を牽引する人が Better だと思う方法に合うものを選ぶのが良いと思います。もう少し詳しく言うと、つまりは「入門が達成したいゴール」に対して最適な方法だと講師役が納得できるものを選ぶということです。これがチグハグになっていると入門の内容やレベルなどが散らかってしまうと思います。これらが揃っていると入門自体をマクロに見た際に「**その一連の『入門』の流れの中で一貫した流れがある**」と言える状態に出来ると思います。

何を議論すれば「しょぼい話」にならずに済むのか

スライドの序盤では「しょぼい話」をキーワードに我々が知っているからこそ入門時に言ってしまいがちな余計な話を避けるようにしましょうということを話しました。

余計な話や入門時のコラム的なのは意外と書き始めると力を入れてしまうけど、それは今話す必要があるのか？ 例えば、JS のライブラリ界の歴史みたいなコラムを開始してもまだ JS 初めたての人たちには不要な話だと思います。

そういう話はライブラリなりツールなりそれぞれ検討が必要なタイミングがやってきたときに改めて学んだり情報集めたりできるので、その時に本気を出して学びを得に行くなり、誰かに聞いてみると良いのではないでしょうか。それまでは、簡単な事例紹介くらいに抑えておいて、必要な際にググれるために頭の片隅に置いておいて欲しいキーワード的なのをパパパッと紹介するとかが良さそうです。

またスライド内で紹介しているような個別の文法の取り扱いなどに関する話し合いはそのコミュニティでの知見を収集するのには有効だと思いますが、そのような内容をどう扱うかについては「入門のゴールがどこか」ということが定義されていれば、さほど悩むこともないと思

います。

また内容の決定に関して多数決などを導入しすぎてしまうと、やはり入門をどうルート立てて入門させて、どのタイミングで何を学び、最後は何が出来るようになるかというイメージを話者の中で組み立てたものにするのを阻害してしまうと思います。要所要所で意見を求めたり参考にするのは良い入門を作り上げていくうえでとても重要なことだとは思いますが、それがゴールに向けて必要なことか、一貫性を失う要因になっていないかについては吟味する必要があるでしょう。

「おまじない」について。

こういうコンテキストで「おまじない」と聞くとプログラミングの入門をちょっとでもしたことあれば、大半の人は

```
#include <stdio.h>
```

のようなものをイメージするでしょう。

実際、あらゆるプログラミング言語には色々な「おまじない」と呼ばれがちな文が存在します。上記の例のように、入門時に「ここは毎回冒頭に書きましょう」というように書籍に紹介されていたりします。

スライドの中盤はこの「おまじない」についてちょっと一歩立ち戻って考えてみようという話題を書きました。

実際にこのスライドで発表した際、C 言語の経験がない新入生に「どれが『おまじない』だと思う？」と質問したところ、`int` と答えてくれました。そうですよね。どれが「おまじない」かなんて初心者にはわからないですよね。

我々のように少しプログラミングをかじっている人たちが「おまじない」について考えましょうとなると、やはり先ほどの例などについてはすぐに議題に挙がることでしょう。そして、これらに関して「おまじない」として扱うのが良いか初めから役割を紹介するかということが議題の中心になることでしょう。皆さんも一度は経験ありませんか？

そのような議論を `int` や `void` といった各種キーワード、はたまた `(` や `+` 演算子などについても同じように行うべきかもしれません。いやいや、「`int` は整数型です」くらい簡単なんだから検討しなくても良いでしょうという意見が大半でしょう。そうなれば話しあったりせずとも `int` に関してはそのように説明すれば良さそうですね。

こうやって 1 つ 1 つ個別の要素について丁寧な検討をするのと本当は良いかもしれません。が、それは大変難しいことです。そもそも `int` の例のように検討しなくても良さそうなこともあるかもしれません。そういった個別の物事をどう扱うかということも、ミクロな視点で見るのではなく、まずマクロに見て、総合的な検討をして道筋を立てておけば、データ型についてはいつ紹介するだとか「おまじない」はいつ説明するとかということが自ずと決まってきます。その中で個別のキーワードなどについてどこまで説明するかということはゴールとの組み合わせや時間との相談ということになるでしょう。

再構築としての「入門」

そのように要は「おまじない」はあらゆるところに潜んでいるので、それぞれの要素をしっかり分解して、それを並べて再構築するのが、「入門」の大きな肝であるということがここまででなんとなく分かってもらえたことだと思います。最後に道筋の立て方のヒント的なことを。

道筋やゴール設定などに関して、自力で考えるのは難しく、自信も持てないと思います。そこでヒントとして、巷に大量にある入門書の目次をずざっと眺めて良さそうなのを探がすのはどうでしょう。スライド内では、それに加えて、MDN[*4]の JS 入門の記事の分け方を見たり、それを参考に実際に JS の Hello, World を分解してみるということをやりました。

目次を読むだけで、その入門がどこをゴールに設定して、そのためにどういう順番で何を意図して扱っているかが見えてくると思います。

例えば、JS の入門書でいきなり XHR をやる本とか多分ないはずで、まずは `console.log` でも叩きながら計算とかやってみて、そのあとは `for` ループをやって、そうやっているうちにちょっとずつ DOM API が入ってくるみたいなそういう流れが存在していると思います。

その流れの雰囲気を目次を眺めることで掴んで、それを基に自分なりにこれだったら上手く教えられるだろうしきっと良いだろうと思えるような筋道を立てていくと上手くいくのではないでしょうか。

ちなみに僕が数年前に JS の入門を KMC の新入生向けにやった時は MDN の資料とはてな教科書の JS の資料を元にして流れと紹介する内容をピックアップして行いました。

まとめ

「入門」をやるときに考えてみたいことをサークルの人向けに話し、それをまとめたブログを基にさらに加筆修正をしてコラムとしてお送りしました。

KMC に限らず長く同じコミュニティに居たりすると、外からやってきたピカピカの新人と相対する時の基準みたいなのが徐々に失われしまい、いきなりガチンコ勝負とという状況が発生することが極稀にあります。僕は勝手にそういう現象を「逆『井の中の蛙』」と呼んでいます。長いことコミュニティの中にいるからこそ、自分は良い意味でも悪い意味でも「中級者」（以上）になってしまっていることをしっかり自覚して、真摯に接しながら時たま突き放したりしながら、知見を伝え合ってこれからも最強のコミュニティを築いていきたいものです。

このコラムを読んでくださっているあなたはどういう立場で何の「入門」をしたことがありますか？ このコラム記事を読んで思ったこと、考えたことなどなどありましたらぜひとも @pastak まで教えて下さい。

さんきゅーです。

[*4] Mozilla Developer Network `https://developer.mozilla.org/` の略称です

あとがき

■jf712　ここにはあとがきを書きます。

■tetsu　現役部員だからコラム書いた。OB の民喜先生が暴露したけど、私の妹の KMC での最大の功績は民喜先生のナンパに成功したことだと思うな。

■wass80　編集補佐です。textlint を導入しようと思ったんですが, 特別なルールを定義するのがだるくて諦めました。冬はアジャイルに (?) 部誌を組みたいです。

■utagawa　Twitter:@utgwkk です。ところで、私の誕生日は 7/28 です。よろしくお願いします。

■hakurin　今回の部誌で最も無益な記事を書いた hakurin です。ダサい T シャツを着てあとがきを執筆しています。ダサ T 記事を書いてからこのあとがきを書くまでの数日間に、またダサ T シャツが増えました。挿絵もちょこっと担当しました。帽子を被っててしっぽの生えてる娘です。新歓で配ったビラが初出です。〆切やばくて 3 時間位で適当に描いてて出来上がった即席なキャラクターですが、案外気に入ったので今後も色んな所に出てくると思います。今回の部誌の裏表紙にもいます。でも気に入ってる割には名前がまだ決まっていません。誰かつけて。ああ、名前がないといえば、無名ちゃん（甲鉄城のカバネリ）かわいいです。カバネリ観ましたか？ 劇場版で総集編の上映も決定した「甲鉄城のカバネリ」を今後ともよろしくお願いします。

■pastak　若木先生が巻頭言ということで、手に取っていただけることが多そうだから、絶対に記事をねじ込みたいと思ったところ、初稿が校正の数時間前に完成しました。今後も丁寧な老害活動を続けていきたいですね。各位やっていきましょう。

■base64　裏表紙の Piet の QR コードを描いた base64 です。このように Piet は無限の可能性を秘めているので、皆さんどんどん Piet やりましょう。

■kyp　最後までお読みいただき本当にありがとうございました。私の記事の余白は、自由帳として有効活用してください。

■kurimotz　裏表紙のレイアウトを担当しました。思いつきで QR コードを Piet にしたら面白そうだと言ったら、翌日には Piet として実行できる QR コードが完成していました。KMC

には、色々な分野で高いスキルを持っている人がいるのだと日々実感しています。

■suzusime　編集長のすずしめです。この度はこの本を手にとって頂きありがとうございます。想像以上に忙しかったのですが、この情報が極めて不安定で流れ去っていく社会の中で、ひとつ物体としての本を出せたことに喜びを感じています。今回も様々な分野の部員から寄稿があり、なかなか面白いことになっているかと思います。本棚の片隅に置いておいて、思い出したときにたまに読む、そんな本になってほしいと思っています。——はい、真面目な挨拶はおしまい。ここからは願望を書きます。巫女さんになりたいです。以上です。ありがとうございました。

独習 KMC vol.9

2016 年 8 月 14 日	初版発行
著作・発行	京大マイコンクラブ
表紙絵・デザイン	haji82
裏表紙デザイン	kurimotz
裏表紙協力	hakurin（絵），base64（Piet QR コード），gire（背景 Piet 素材）
挿絵	若木民喜，nepnyan，hakurin
印刷	コーシン出版
メールアドレス	info@kmc.gr.jp
Web	https://www.kmc.gr.jp/

- 落丁・乱丁の際は在庫がある限りお取り替えいたします。上記のメールアドレスまでご連絡ください。
- 裏表紙の QR コードは Piet コードにもなっています。

独習KMC vol.9

■利用分類
部誌
■製品分類
KMC
■レベル
初級　中級　上級

https://www.kmc.gr.jp

Kyoto University
Micro Computer Club