こんなのもプログラミング

竹内郁雄 †

概要 プログラミングに関する既成概念に囚われないいろいろなプログラミングに関する話題を紹介する．そのうちのいくつかについてはすでに非公式の場で喋ったことがあるが，ここではその背景説明や教訓も含めてきちんと紹介したい．この発表は 2005 年の夏のプログラミングシンポジウムでの竹内の発表の「プログラム問題の創造」の続編という位置付けである．

Programming a la carte

Ikuo Takeuchi‡

Abstract We introduce four topics related to programming in a wider sense. Some of them are already presented in informal talks so far, but here we describe the topics in full details with some lessons to be learned. This article is a sequel of my report for 2005 summer programmming symposium.

1. カレンダープログラム

発端は 2007 年の夏のプログラミングシンポジウム (First Programmming Languages プログラミング言語の実力と美学, 2007 年 8 月 8 日〜10 日, 於信州戸倉上山田温泉ホテル清風園) である．参加者には, 夜の自由セッションで My Favorite Language (MFL) について 1 人 10 分以内で宣伝せよという宿題が出た．口先だけで宣伝するのでは迫力ないので, ついでにその MFL でカレンダー印刷プログラムを事前に書いてくるようにという条件が加わった. しかし当日になって驚いたのであるが, なんとプログラムを書いてこないで演説をする人もいた. これじゃあ迫力なし.

これに参加した私の MFL は当然 Lisp[1] である．Lisp でカレンダー印刷プログラムを書いていってもいいが, それじゃなんのプロパガンダにもならないし, そんな学生の宿題みたいなものに付き合うこと自体に抵抗があった. そうこうしているうちに時間が経った.

プロシンの 2〜3 日前に思い出して, そもそも「カレンダーとはなにか?」について考えを巡らせてみた．ところがこれが面白い. 以下, その考察を振り返ろう.

カレンダーが成立するためには「日」の概念がなければならない. そのためには日の出, 日の入りがなければならない. こう考え始めたこと自体ですでに「地球」を飛び出している. 光源たる「太陽 (恒星)」に対してその「惑星」は自転していなければならない.

惑星の住民にとって公転に対応した「年」の概念があるとすれば, 「年」を区切るなにかが必要である．もし自転軸が公転面に垂直だと, その惑星上の住民にとって (精密測定技術がないかぎり), 毎日が同じ 1 日であり, 年を区切るものがない. 自転軸が傾いていてこそ四季がある (春分, 夏至, 秋分, 冬至).[2] つまり四季は天文学的な概念なのである.

とすると「月」も天文学的な概念なのだろうか. たしかに天上に見える月とカレンダーの月は関係しているように見える．だが月の満ち欠けの周期は 29.53 日であり, 現在の月の日数 (30 日と 31 日が主) と微妙に異なる．なので, 地球の 12ヵ月というのは結構便宜的なものであろう．実際, 太陰暦では適宜閏月 (13ヵ月目) を加えて調整している．なるほど, 太陰暦か. 太陰暦印刷プログラムを宿題にしたらみんな結構困っただろうな, などと考える.

四季が天文学的な概念なのだから, 月の数は 4 の倍数になっているとわかりやすいが, どうも必然性はなさそうだ. 地球みたいに月, つまり「衛星」が 1 個であればわかりやすいが, 火星には Deimos と Phobos という 2 つの月があり, 土星に至ってはいま知られている衛星が 61 個で, まだ時々新発見があるという．衛星をめでて, 月を区切るにはいろいろ難しい問題がありそうだ.

カレンダー印刷プログラムにもう一つ必要なのが「週」の概念である．いや, 本当に必要かどうかも怪しい．だが, 年, 月, 週, 日という 4 階層を仮定するのは, どことなく自

† 早稲田大学 理工学術院 情報理工学専攻, 東京大学名誉教授

‡ Waseda University, Emeritus Professor of the University of Tokyo

1 この報告を書く寸前の 10 月 24 日に Lisp の産みの親である John McCarthy の訃報が入った. 合掌. この駄文は亡き John McCarthy に捧げたい. 迷惑かもしれないが….

2 実際には軌道の離心率が大きければ温度が変わるので四季がある. はこだて未来大学の中島秀之氏に感謝.

然に見える. いや, 自然だと思うことにする. なぜなら, ここまで来ると, 興味は (太陽系に限らない) よその惑星の住民にカレンダーシステム, およびそれに基づくカレンダー印刷プログラムを売り込むというビジネスを始めることなのである. これを USO カレンダーと呼ぶ. USO は, もちろんウソではなく, Universal Standardization Organization という ISO の上部機関が定める規格を意味している. ここにおいて, 地球人がヘゲモニーを握るべきというのは, ケチ臭い日本人の島国根性をはるかに超えた発想である.

話は「週」に戻るが, 最早明らかなように 1 週が 7 日からなるということにはなんの天文学的な根拠もない. はっきり確認したわけではないが, 宗教由来だろう. 7 以外の中国由来の六曜は 30 日の 5 等分から来たらしい. 現在の日本では, 先勝・友引・先負・仏滅・大安・赤口と呼ばれている.

ここまで来ても, カレンダー印刷プログラムはまだ書き始められない. 元年を決めなければならない. いつを元年にするか, これがまた恣意的である. それに元年が 1-オリジンになっているのも気持悪いといえば気持悪い. BC 1 年と AD 1 年が連続してしまう. 中華民国の民国紀元は西暦 1912 年だし, そもそも日本の元号は天皇の即位のたびに紀元が異なる. そのうえ皇紀というのもある (西暦に 660 年を足す). イスラムのヒジュラ暦の紀元は西暦 622 年である.

表 1-1 太陽系の惑星

	公転周期 (年)	自転周期 (日)	衛星	赤道傾斜角
水星	0.241	58.65	0	0°
金星	0.615	243.02(逆)	0	178°
地球	1	0.997271	1	23.4°
火星	1.881	1.02595	2	25.19°
木星	11.86	0.4135	65	3.08°
土星	29.46	0.4264(逆)	61〜	26.7°
天王星	84.01	0.7181	27	97.9°
海王星	164.79	0.6712	13	29.6°

さらに, USO レベルでの閏も決めておかなければならない. 太陰暦では閏月というのがあった. 地球でもときどき閏秒がある. こう見ると, 閏年という言葉が誤用のようだ. 正確には 2 月 29 日を閏日と呼ぶべきなのであろう. 閏がプラス 1 というのにも必然性はない. マイナス補正のほうがよければ, マイナス閏もあってよかろうし, プラス 6 という閏もあってよい.

このように考察してくると, カレンダーの規格の大部分は実に恣意的に決められていることがわかる. ここから先は言葉巧みのセールスでバルタン星人にカレンダー印刷プログラムを売ることを考えるべきなのだ. とりあえず, 表 1-1 に太陽系の惑星の諸元のうち, カレンダーに必要そうなものを抜粋した.[3] 表を見てわかるように, 水星, 金星, 木星, 天王星の住民にはカレンダーの必要性はあまりなさそうだ.

ここからは話を具体的にするために, 火星に焦点を当てよう. 火星の 1 年は 669.6038 日である. ここでの 1 日は火星の 1 日である. 火星は地球より少し自転周期が長い. 火星人にとって地球の 1 日を基準にしても意味がないことは明らかであろう.[4]

火星の月は前述したとおり, 7.66 時間で公転する Phobos と, 30.35 時間で公転する Deimos の 2 個である. 地球の月に比べると恐ろしく速い. 地球の月と同等には扱えそうにない. もう面倒なので, 大体のところ 30 日とか 31 日を 1 月の日数として不都合はなかろう. すると年に 22ヵ月, 30 日の月を 13 個, 31 日の月を 9 個とすればよい. もちろん, これでは火星の公転周期とずれてくるので, 10 年に 6 回の +1 日の閏日, または 10 年に 1 回の +6 日の閏日を設け, さらに 263 年に 1 回のエキストラの +1 閏日を設ければ誤差はかなり減る. (少し怪しいところもあると思うがお許しあれ.)

次は元年と曜日である. これは多少, おとぎ話的に決めよう. かぐや姫は月に戻るはずが, 間違えて火星に戻ってしまった. これが火星暦の元年である. 史実 (?) によるとそれは西暦では 713 年である (奈良時代). とすると, 地球の 2011 年は火星の 691 年に相当する.[5] 曜日はこれまた少々インチキだが仁・義・礼・智・忠・信・孝・悌の八曜とする.

あまり, USOっぽくないが, もし武田信玄が天下を取っていたら, 彼は曜日を風・林・火・山の 4 曜にしたに違いない. しかし, それでは週末比率が高すぎるので, 山のうしろに灰曜日を設けたらどうなるか···. また, 年号を皇紀としたであろう, などなど.

ご存知のようにワールドカップは 4 年に 1 回 (オリンピックと位相がずれている) 開催されるが, 大体 7〜8 月が開催月である. サッカー好きの私としては, ワールドカップの年のみ, 通常の Jun, Aug を廃止し, その代わりに 62 日間の Soc (フルネームは Soccer) という月を設けたい. つまり, その年は変則的に 11ヵ月になるのである.

延々とカレンダーの恣意的な決め方について述べてき

[3] ここまでのいろいろな事項は手軽に調べられる Wikipedia を参考にした.

[4] 実は 2007 年のプロシンの演説ではここを間違えていた. 地球帝国主義の発想から抜け出せていなかったということか.

[5] (1+ (/ (* (- 2011 713) 365.24) 686.98)) = 691.095, なお, ここでは日を地球の 1 日にして計算している,

たが，ここからがプログラミングである．実際，私は上記の考察をプロシンの前日までに終え，当日の朝からプログラミングを開始した．作ったのは「カレンダー表示プログラム作成プログラム」というメタプログラムである．Lisp の関数閉包を使い，月の日数，月名の存在・不存在 (上記の Soc を参照)，閏の計算などをすべて年を引数とする関数として与えると，カレンダー表示プログラムを生成するプログラムである．基本的にはテーブル駆動型のプログラムであるが，要所要所のテーブル要素がすべて関数なので，できることの自由度が高いのがミソである．生成プログラム自身は 60 行程度だが，予定より時間がかかってしまい，プロシンには遅刻してしまった．

ちなみに通常のグレゴリオ暦のために与える引数と，火星暦に与える引数を示しておく (一部省略)．また，火星暦と武田信玄暦の表示結果も示しておく．

```
; 地球用のカレンダー 基礎データ

(defun c31 (y) 31)
(defun c30 (y) 30)
(defun グレゴリオ閏 (y)
     (if (or (= (mod y 400) 0)
             (and (not (= (mod y 100) 0))
                       (= (mod y 4) 0) ))
         29 28 ))
(defun always-t (y) t)

(defglobal
 グレゴリオ月のリスト
 `((”Jan” ,#´c31 ,#´always-t)
   (”Feb” ,#´グレゴリオ閏 ,#´always-t)
   (”Mar” ,#´c31 ,#´always-t)
   (”Apr” ,#´c30 ,#´always-t)
   ...
   (”Nov” ,#´c30 ,#´always-t)
   (”Dec” ,#´c31 ,#´always-t) ))

(defglobal グレゴリオ曜日リスト
 ´((”Sun” ”Mon” ”Tue” ”Wed” ”Thu” ”Fri” ”Sat”)
   ; 下は 1900 年 1 月 1 日が月曜日という意味
   (1900 ”Mon”) ))

; 火星用のカレンダー 基礎データ

(defun 火星閏ろ (y)
  (if (memq (mod y 10) ´(0 1 2 4 6 8)) 31 30) )
; 上記は以下のように定義してもよい
; (defun 火星閏ろ (y)
;    (if (= (mod y 10) 0) 36 30) )

(defun 火星閏に (y) (if (= (mod y 263) 0) 31 30))

(defglobal
 火星月のリスト
 `((”い” ,#´c31 ,#´always-t)
   (”ろ” ,#´火星閏ろ ,#´always-t)
   (”は” ,#´c31 ,#´always-t)
   (”に” ,#´火星閏に ,#´always-t)
   (”ほ” ,#´c31 ,#´always-t)
   (”へ” ,#´c30 ,#´always-t)
   ...
   (”な” ,#´c30 ,#´always-t)
   (”ら” ,#´c31 ,#´always-t) ))

(defglobal 火星曜日リスト
 ´((”仁” ”義” ”礼” ”智” ”忠” ”信” ”孝” ”悌”)
   (1 ”仁”) ))
```

◆ 火星暦印刷例 (火星暦 691 年ぬ月)

```
(cal-mars 691 "ぬ")

 仁 義 礼 智 忠 信 孝 悌
                    1  2
  3  4  5  6  7  8  9 10
 11 12 13 14 15 16 17 18
 19 20 21 22 23 24 25 26
 27 28 29 30
```

◆ 武田信玄暦印刷例 (皇紀 2671 年神無月)

```
(cal-takeda 2671 "神無月")

 風 林 火 山 灰
        1  2  3
  4  5  6  7  8
  9 10 11 12 13
 14 15 16 17 18
 19 20 21 22 23
 24 25 26 27 28
 29 30 31
```

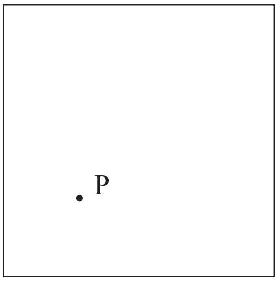
[教訓その 1] プログラミングの問題を出されたら，安直に考えずに，**問題の起源に立ち戻って**，プログラムを設計するべきである．すると必ず役に立つ，あるいは面白い発見がある．ただし，遅刻する可能性がある．

2. 確率ゼロの例外に対処

ここから先の 3 つのトピックスはすべて，私が「数学セミナー」に出題した問題である．数学セミナーに出題するにあたって，数学マニアにプログラミングの面白さを知ってもらいたいといつも考えているので，逆にプログラミングマニアにも興味をもってもらえると思う．なお，解答はすべて付録に記載した．できるだけ自分で考えてみられることをお勧めする．

最初の問題は以下の通り．

単位正方形の裏に n 角形 K が隠れている．この正方形の内部に点 P をとったとき，点 P が K の外側にあるのか，境界上にあるのか，内側にあるのかを，質問を繰り返すだけで，「効率よく」判定する方法を考えよ．許されるのは任意の 2 点を結ぶ線分と K の境界との交点の個数を尋ねる質問だけである．質問の答は $0 \sim n$ の整数か，∞ (線分と K の辺が一部で重なる場合) である．



実はこの問題では，平面幾何という数学を題材にして「プログラム」を書いてほしかった．起こり得るいろいろな場合をあらかじめ想定して，一つの目標を達成するための手順として書かれたものならプログラムと呼べるからである．この問題は数学の文脈なので，プログラミング言語は必要ないが，書かれた指示にしたがって，点 P の置かれた状況を正しく，かつなるべく少ない質問回数で判定できるような記述を目指さなければならない．

なるべく少ない質問回数といっても，たとえば ∞ という答えは確率 0 でしか返ってこない．だから，平均質問回数を考慮しても無意味である．ここでは最悪質問回数を考えなければならない．

この問題をちょっと考えてみればわかるが，まず，以下のように概念を整理しておくとよい．

正方形の境界状に，点 Q_i $(i = 1, \cdots)$ を必要になるたびに順次選ぶ．ただし，点 Q_i は線分 PQ_i がそれまでに選んだ線分とは点 P 以外ではお互いに重ならないように選ぶ．

さらに，i が偶数 $2j$ の場合，線分 $\mathrm{PQ}_{2j-1}, \mathrm{PQ}_{2j}$ $(j = 1, \cdots)$ が直線上に並ぶように選ぶ．この対を「射貫き対」と呼ぶ．

線分 AB と多角形 K の境界との交点の数を $c(\mathrm{AB})$ と書く．許された質問は c を尋ねることだけである．

さらに考えると，いくつかの事実が確認できる．

(1) $c(\mathrm{PQ}_i) = 0$ なら，P は K の外部にある．
(2) $c(\mathrm{PQ}_{2j-1}) + c(\mathrm{PQ}_{2j}) = c(\mathrm{Q}_{2j-1}\mathrm{Q}_{2j}) + 1$ であれば，P は K の境界上にある．ただし，すべての c は有限．
(3) 図 2-1 のように，多角形の頂点と線分 PQ_i が交わり，その頂点から出る 2 本の辺が線分の片側にある場合 (接点)，答えを「信頼できない情報」と呼ぶ．また，∞ という答えも「信頼できない情報」に分類する．それ以外の答えは「信頼できる情報」である．答えが信頼できるかできないか，単独では判定できない．
(4) $c(\mathrm{PQ}_i)$ が信頼できる情報であることが確実であれば，$c(\mathrm{PQ}_i)$ が偶数なら P は K の外部，奇数なら K の内部にある．
(5) 図 2-2 のように，多角形の辺と線分 PQ_i が重なる場合，PQ_i とその射貫き対は少なくとも多角形の 2 つの頂点を含む．この場合の ∞ という答えは信頼できない情報だが，同時に接点による 2 個の信頼できない情報の可能性を消している．
(6) 信頼できない情報の数の上限は n である．

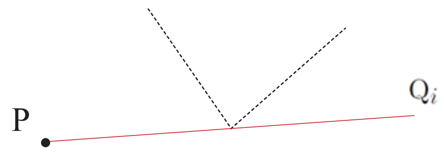


図 2-1

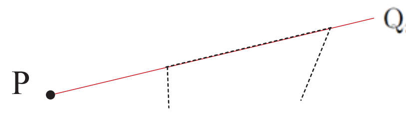


図 2-2

質問を重ね，信頼できる情報を抽出することがポイントである．事実 (5) と (6) から，高々$2n+1$ 本の線分を用意すれば，少なくとも $n+1$ 個の返事は信頼できる情報となる．なお，$c = \infty$ という返事はそれなりに役立つ情報であることに注意．

これらの考察から実際にプログラム風の記述をつくるのはそれほど難しくない．もちろん，上記の方法はいわゆる「一般解」あるいは「上限」を求めるためのものであり，特定の n に対するより効率のよい「特殊解」がある．実際，最悪 3 回 $(n = 3)$, 6 回 $(n = 4)$, 7 回 $(n = 5)$ というかなり良い特殊解が求められている (水谷一氏)．これは比較によるソーティングとよく似た状況である．

この問題を出したとき，ある解答者から「安全管理の難しさを実感してしまう」という感想が寄せられた．まさにその通りで，この問題から得られる教訓は次の通りである．

[教訓その 2] 確率ゼロの現象であれ，それが**起こり得る (最悪の) ことであれば，まじめに対処**しておかなければならない．

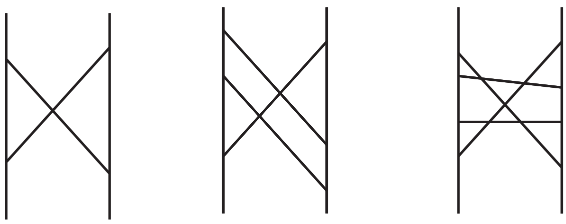
これに関連して思い起こされるのが，2005 年の東京証券取引所 (東証) における驚くべき取り引きミスである．担当者が入力を間違え，ジェイコム株の「61 万円で 1 株売り」を「1 円で 61 万株売り」としてしまった．この結果，みずほ証券が 407 億円の損害を被ったとされている．あり得ないことが連鎖して起こったのが原因である．まず，(警告音慣れした) 担当者が 3 回続いた警告ビープ

音をすべて無視した．たとえ，そうだったとしても，これはあり得ない取り引きなので，システムはその注文を受け付けるべきでなかった．システムのバグと言ってよい．担当者はすぐにミスに気づき「注文取り消し」をしたが，受け付けられなかった．これはシステム構築の際，取り消しコマンドを受理できないようにしてしまったミスだった．そして，最後，東証が即座に売買の一時停止をしなかった．これは人間側の危機管理の問題である．マーフィーの法則の拡大解釈版のとおり，起こり得ないことは起こるのである．ソフト開発者は肝に銘じたほうがいい．

3. 仕様変更への対処も重要なプログラミング

阿弥陀籤というのは室町時代からあったという．英語では Ghost Leg, Ladder, あるいは Amidakuji でも通じるらしい．国際的に知られているかというとそうでもない．実際，私がエジプトの大学で講義をしたときに聞いたら誰も知らなかった．阿弥陀とつく理由は阿弥陀仏の後光らしい．つまり，室町時代の阿弥陀籤は線を放射状に引いたのかもしれない (たしかにそのほうが参加者にとって平等).

阿弥陀籤が上から下への 1 対 1 対応になっている証明は簡単そうで，簡単ではない．実際，図 3-1 に示したような病的な例があるからである．もちろん，ここでは横線をどういう風に引いてもよいという仮定をしている．そういう風に線を引いても，遊ぶときに曖昧性が生じたりはしないからである．



縦線の一部が不通過　　同じ横線を往復　　横線を不通過

図 3-1

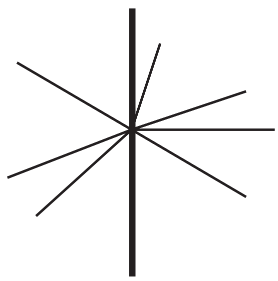


図 3-2

さて問題である．図 3-2 のように，誰かが，すでに横線が引いてある縦線の位置にさらに横線を引いたらどうなるだろうか? これでもちゃんと遊べるようにルールを「拡張」してほしい．つまり仕様変更への対処がこの問題の要点である．

ルールの拡張には「エレガンス」が必要である．子供でも遊べるという容易さを失ってはいけない．また，同じ点に入った横線をすべて無視してしまうという不粋で自明な拡張は勘弁してほしい．

数学セミナーにこの問題を出したとき，図 3-3 の縦線の上からの入り (ℓ_u)，下への出 (ℓ_d) と ℓ_u から時計回りに入っている横線 ($\ell_1, \cdots, \ell_n$) について，$\{\ell_u, \ell_1, \cdots, \ell_n\}$ から，$\{\ell_1, \cdots, \ell_n, \ell_d\}$ への一般的な一対一写像を利用するというルールを考えた人が結構いたのだが，これは存在定理を説明しているだけといった感じで，すぐに遊びたい人には役立たない．

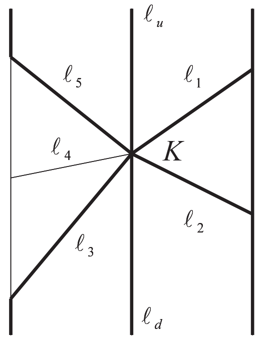


図 3-3

なお，この問題に関しては，やさしいので解答は示さない．

[教訓その 3] 物事はなるべく簡単に考えよ．子供でも遊べる (実行できる) ような簡潔なプログラムを書こう．

4. 外部仕様に基づくプログラミング

これは今年の問題である．

有理数から有理数への関数 f で

$$f(f(f(f(x)))) = x^2$$

かつ

$$x \to 0 \text{ のとき } f(x) \to \infty \text{ (}x\text{ は有理数の範囲を動く)}$$

となるようなものは存在するか? 存在しないならその証明を，存在するなら具体的にそのような関数 f を書け．

これは関数の入力と出力についての外部仕様を与えたときに，その定義本体がちゃんと書けるかどうかという問題である．無理難題の外部仕様は証明付きで断るべきだし，可能なのであればプログラムを実際に書くべきである．この種の問題は数学では関数方程式と呼ばれる．

数学オリンピックでは常連の問題らしいが, こんな変な問題は過去にないはずだ. 有理数と言っているところにプログラミングの匂いがする.

複数の人から, どうしてこんな変な問題を思いついたのかと聞かれたが, 出題締切日が迫ったときに出る締切ホルモンのせいだろう. 実際には最初, 発散条件のない, 整数から整数への関数をつくる問題として発想した. これは以下のようにちょっと簡単すぎる.

解答例は, f の適用を $\mapsto$ で表現したとき, $0 \mapsto 0$, $1 \mapsto 1$ (実は, $0 \mapsto 1$, $1 \mapsto 0$ でもよい) のほかに,

$2 \mapsto 3 \mapsto 5 \mapsto 6 \mapsto 2^2 \mapsto 3^2 \mapsto 5^2 \mapsto 6^2 \mapsto 2^4 \mapsto \ldots$

$7 \mapsto 8 \mapsto 10 \mapsto 11 \mapsto 7^2 \mapsto 8^2 \mapsto 10^2 \mapsto 11^2 \mapsto 7^4 \mapsto \ldots$

$\ldots$

というふうに, 4 つの非平方整数の組から出発する数列を (可算) 無限個構成することである. なお, 負の数 x に対しては $f(x) = f(-x)$ と定義する.

この考え方は構成的にプログラムを明確化していく道筋とそう変わらない. ここまで来れば, 本来の問題にたどり着くのはすぐである. 問題を作るのにも, stepwise refinement は有効なのだ.

なんと, 出題に対する解答の 2/3 が門前払いの不正解だった. これについては教訓のところで述べる. 正解はおおむね 3 種類に分かれた. その中で一番多かったのは上記の整数→整数関数の自然な延長にあった. これを数列構成法と呼ぼう. 残るは私の解答も含めて 3 種類だが, よく見るとそれぞれ, 実際にコンピュータで計算をするときの, プログラムの複雑性, プログラムの実行時間効率が違う. 実際に付録をご覧になっていただきたい. いつものことながら, 出題者が意図したよりもずっと面白い問題提起をしてくれたと思う.

[教訓その 4-1] まずは, **仕様を正しく理解するべし**. この問題で不正解が多かったのは, 有理数から有理数への関数を有理関数 (つまり, 分子分母が x の整数係数多項式になっている関数) と誤解した人が多かったことによる. もし, そういう意味なのであれば, 題意を満たす関数が存在しないことは容易に証明できる. もう一つ割と多かったのは, 実数から実数への関数をそのまま使ってしまった解答だった.

$$f(x) = 1/|x|^{\sqrt[4]{2}} \quad (x \neq 0), \quad f(0) = 0$$

は確かにそれらしい解答なのではあるが, 有理数 x に対して $f(x)$ が必ずしも有理数にならないのが致命的である. こういう落し穴に引っかかってはいけない.

[教訓その 4-2] 同じ外部仕様からプログラムをつくっても, **実行性能は千差万別**なので注意するべし. こちらのほうが本来の教訓である. 付録に示した解答を見ればわかる通り, 実際に f をプログラムの手順として記述しようとした際, つまり, 存在を納得させるだけではなく, 関数を構成しようとした場合, その記述の複雑さと実行性能は大いに異なるということである. これはアルゴリズムの教科書ではいやというほど学ぶことだが, いざ具体的な問題が与えられると, うっかりと忘れがちである.

5. むすび

プログラミング全般に教訓となりそうなパズル風のものをいくつか紹介した. なにもプログラミング言語をいじらなくてもプログラミングについて結構学べるものだ. 思い起こすと, 私がもっと以前に数学セミナー等に出した問題でも, ここには出てこなかったような面白い (というか自明の) 教訓が学べるものがありそうだ. それらについては, 別途機会を設けて紹介したい.

[参考文献]

トピックスの 2〜4 については日本評論社の数学セミナーの以下の各号に出題が出ており, その解答はその 3ヵ月あとの号に掲載されている. どれも手元に現物が見当たらないのでページ情報については省略する.

[1] 竹内郁雄: プログラム問題の創造, 2005 年度 情報処理学会夏のプログラミングシンポジウム報告集, 2005.

[2] 竹内郁雄: エレガントな解答を求む, 数学セミナー 2006 年 8 月号 (出題), 11 月号 (解答), 日本評論社, 2006.

[3] 竹内郁雄: エレガントな解答を求む, 数学セミナー 2009 年 9 月号 (出題), 12 月号 (解答), 日本評論社, 2009.

[4] 竹内郁雄: エレガントな解答を求む, 数学セミナー 2011 年 8 月号 (出題), 11 月号 (解答), 日本評論社, 2011.

[質疑応答]

2005 年と同様, 時間一杯まで喋って, 質問を制してしまったため, 質疑応答はない. 言ってみれば, 質疑応答を必要としないくらい, 自明の教訓を述べたまでであった.

質疑応答ではないが, 草稿を見て記述の誤りを指摘していただいた電気通信大学の寺田実氏に感謝する.

付録 2 と 4 の解答

2 の解答

以下の擬似プログラムが一つの上限を与えている.

0: $q = 0$ とする. (q は質問回数のカウンタ)
1: $g = 0$ とする. (g は偶数交点線分のカウンタ)
2: $k = 0$ とする. (k は奇数交点線分のカウンタ)
3: $t = n$ とする. (t は多数決の根拠となる数)
4: $m = 0$ とする. (m は P が境界上でないことが未確定の間の $c = \infty$ の数)
5: $i = 1$ とする. (i は Q_i 選択の制御用)
6: 正方形の外に適当に Q_i を選ぶ. ただし, i が偶数の場合は, Q_{i-1}, P, Q_i が直線上に並ぶように Q_i を選ぶ (射貫き対). これまでつくった線分と重ならないようにする. $c(\mathrm{PQ}_i)$ を尋ねる. q を 1 増やす.
7: $c(\mathrm{PQ}_i) = 0$ であれば, P は外部 ■.
8: $c(\mathrm{PQ}_i) = \infty$ のとき, P が境界上でないことが確定していれば, t を 2 減らし, P が境界上でないことが未確定であれば m を 1 増やす. そのあと, i が偶数なら, i を 1 増やしてステップ 6 に戻る. i が奇数なら (射貫き対の相棒にはもう意味がないので) i を 2 増やしてステップ 6 に戻る.
9: i が偶数, かつ P が境界上でないことが未確定であれば, $c(\mathrm{Q}_{i-1}\mathrm{Q}_i)$ を尋ねる. $c(\mathrm{PQ}_{i-1}) + c(\mathrm{PQ}_i) = c(\mathrm{Q}_{i-1}\mathrm{Q}_i) + 1$ であれば P は境界上 ■, そうでなければ P が境界上でないことが確定し, t から $2m$ を引く.
10: $c(\mathrm{PQ}_i)$ が偶数であれば, g を 1 増やす. $g > t$ になれば, P は外部 ■.
11: $c(\mathrm{PQ}_i)$ が奇数であれば, k を 1 増やす. $k > t$ になれば, P は内部 ■.
12: i を 1 増やして, ステップ 6 に戻って繰り返す.

4 の解答

以下の解答ではどれも, $f(0) = 0$, $f(1) = 1$, さらに負有理数 x に対しては $f(x) = f(-x)$ としている.

(1) 数列構成法のナイーブ版

$0 < x < 1$ かつ他の有理数の 2 乗になっていない既約有理数 x を順番に並べる. たとえば,

1/2, 1/3, 2/3, 3/4, 1/5, 2/5, 3/5, 4/5, 1/6, 5/6, 1/7, ...

という具合. 先頭から 2 つずつ数をとり, 上に示した数列と似たような数列をつくる. たとえば, 2 番目の対 2/3 と 3/4 の場合は

$2/3 \mapsto 3/2 \mapsto 3/4 \mapsto 4/3 \mapsto (2/3)^2 \mapsto (3/2)^2 \mapsto (3/4)^2 \mapsto (4/3)^2 \mapsto (2/3)^4 \mapsto \ldots$

1 より小さい数の次にはその逆数が続くので, 発散条件も OK. これらの数列を実現する関数 f を定義する.

(2) ζ 氏 (ペンネーム) の解答

2 以上の整数に関しては, 非平方数の列 2, 3, 5, 6, 7, 8, 10, 11, 12, 13, ... の先頭から 2 つずつ数をとってそれぞれ数列をつくる. たとえば, 最初の 2 個の数 2 と 3 からは以下の数列をつくる.

$2 \mapsto 1/2 \mapsto 3 \mapsto 1/3 \mapsto 2^2 \mapsto (1/2)^2 \mapsto 3^2 \mapsto (1/3)^2 \mapsto 2^4 \mapsto \ldots$

これで 2 以上の整数とその逆数を全部尽くせる. それ以外の既約有理数 q/p については,

$$f(q/p) = p/q \quad (0 < q/p < 1)$$
$$f(q/p) = p/q^2 \quad (p^{2^n} < q < p^{2^{n+1}},\ n(\geq 0) \text{ は偶数})$$
$$f(q/p) = p^2/q \quad (p^{2^n} < q < p^{2^{n+1}},\ n(\geq 0) \text{ は奇数})$$

と定義する.

(3) 田中哲朗氏の解答

$0 < x$ かつ $x \neq 1$ の有理数 x は

$$x = p_1^{z_1} p_2^{z_2} \ldots p_n^{z_n}$$

と一意に表すことができる. ただし, $p_i < p_{i+1}$, p_i は素数, z_i は 0 以外の整数, $n \geq 1$. たとえば, $8/21 = 2^3 3^{-1} 7^{-1}$ となる.

$prev(p)$ 素数 p より小さい素数のうち最大のもの
$next(p)$ 素数 p より大きい素数のうち最小のもの
$even(p)$ 素数 p より小さい素数の個数が偶数か?

$x = p_1^{z_1} p_2^{z_2} \ldots p_n^{z_n}$ に対して, $f(x)$ を以下のように定義する.

$$f(x) = 1/x \quad (z_1 < 0)$$
$$f(x) = next(p_1)^{-2z_1} next(p_2)^{-2z_2} \ldots next(p_n)^{-2z_n} \quad (z_1 > 0 \text{ かつ } even(p_1) = \text{真})$$
$$f(x) = prev(p_1)^{-z_1} prev(p_2)^{-z_2} \ldots prev(p_n)^{-z_n} \quad (z_1 > 0 \text{ かつ } even(p_1) = \text{偽})$$

これはかなり複雑な解答で, 発散条件を満たすことの確認も自明ではない.

(4) 竹内の解答

まず, 言葉の定義. 他の有理数の 2 乗になっていない有理数を非平方有理数, 2 乗にも 3 乗にもなっていないような有理数を非平立方有理数と呼ぶ. すると, すべての有理数 q に対して, ある非平立方有理数 p があり, $q = p^{2^m 3^n}$ ($m \geq 0$, $n \geq 0$) と表せる．ここで, n が偶数 (0 を含む) のものを偶立方有理数, n が奇数のものを奇立方有理数と呼ぶことにする.

$0 < x < 1$, x が非平方有理数, x が偶立方有理数という条件を満たすそれぞれの x を第 0 項 (初項) とする数列を次のようにつくる.

$x \mapsto 1/x \mapsto x^3 \mapsto (1/x)^3 \mapsto x^2 \mapsto (1/x)^2 \mapsto x^6 \mapsto (1/x)^6 \mapsto x^4 \mapsto \cdots$

これらの数列の第 $4k$ 項および第 $4k+1$ 項 ($k \geq 0$) は偶立方有理数だが, 第 $4k+2$ 項および第 $4k+3$ 項 ($k \geq 0$) はそれより 3 のベキが 1 つ多い奇立方有理数になっていることに注意 (つまり, 立方根が有理数になる).

これらの数列を生成する関数 f は以下の通り.

$$f(x) \equiv 1/x \quad (0 < x < 1)$$

$$f(x) \equiv (1/x)^{2/3} \quad (x > 1 \text{ かつ } x \text{ が奇立方有理数})$$

$$f(x) \equiv (1/x)^3 \quad (x > 1 \text{ かつ } x \text{ が偶立方有理数})$$

上記の簡単な言葉の定義 (計算も簡単) さえ了解すれば, 非常に簡単な記述になっていることがわかるだろう.