プログラミング言語 Icon について

出　口　厚　実

1．序にかえて

本学の学内情報ネットワークシステムがようやく稼動しはじめ，曲がりなりにも“情報化時代”の必需品といわれるインターネットの大海に漕ぎだす港（というより，ごく小さな桟橋に過ぎないが）を得たことになる。小さなパソコンに繋がれた頼りなげな細い1本の線が壁の中に消えた彼方に，何千万もの情報の網の目が張り巡らされていることは現実感のあるイメージとして受け止めにくいことである。LAN の導入そのものが，今後，様々なコミュニケーション上の利便を提供し，われわれの研究・教育活動の日常スタイルにすら変革をもたらすかもしれない。しかし，最大のメリットとして挙げなければならないのは，やはりネットワークでしか接近できなかった学術情報へ到達できるということであり，他のメディアで入手され得たものでも，より迅速に，はるかに広範囲に，よりたやすく獲得できるという恩恵は計り知れない。

教育や研究に関する情報は有り余るほどいたる所で生産されながら，それを必要とする実際の需要者に届いていないという，「送り手」と「受け手」の間の損失が非常に大きい伝送であるように思われる。コンピュータネットワークのおかげで，コンピュータの応用に関するソフトウエアや各種資料の所在に関する情報も飛躍的に増大して，その「実需要者」がみな利用者になれるような状況も夢の世界ではなくなった。

本稿で取り上げるのは，数々の優れた特徴をもちながら，そのアカデミックな Network 育ちの経歴からか，意外にわが国で注目されず，また，とりわけ人文科学分野での利用を視野に入れていたにもかかわらず，われわれの言語学の世界でも地歩を築き得なかった不遇のプログラミング言語 Icon である。恐らく，情報の不足（またはアクセス不能）のゆえに生じた

実需要者→非利用者化の1例といってもよいであろう。

本筋をそれるが，Icon という名称も不運の一因であったかもしれない。現在，コンピュータを操る者ならだれでも知っている，Graphical User Interface のオブジェクトを指す「アイコン」が一般化する以前に，プログラミング言語 Icon は存在していたけれども，後参の「アイコン」を作成・操作するためのユーティリティか何かと誤解されて，「実需要者」の目から逃れたということも想像されるのである。

2．Icon との出会い

筆者が始めて Icon の存在を知ったのは，1986年頃だと記憶しているが，雑誌 Computer Language での紹介記事を読んだときである。後に，SNOBOL 4＋ を商品化していた米国 Catspaw 社が Icon 関連書籍などを取り扱い品目に入れていることがわかり，情報の入手もいくらかたやすくなったが，当初はソフトウエアの入手方法すら定かでなかった。同誌の “Exotic Language of the Month Club” 欄での概要から，MS-DOS 版がリリースされているとのことを知り興味をそそられた。

入手のきっかけは，各種のプログラム言語を専門に扱うイギリスの業者がたまたま同じ雑誌の広告欄に Icon を載せていたからであった。問い合わせの結果，IBM PC-DOS でなくても作動する MS-DOS 版のインプリメントらしいと判明したので，早速，発注した。このとき購入したのが Icon ver.6.1 の SMM（スモール・メモリー・モデル）である。ただし，その当時，MS-DOS マシンとして利用できたのは研究室で購入したばかりの NEC 製の V30（8086互換）機で，その独自のフロッピーフォーマットのため，3.5” プログラムディスクそのものが読み出せないというトラブルに見舞われた。ソフト販売業者は親切な対応で，こちらから送った書き込み例のサンプルディスクをもとに NEC パソコンで使用できる新たなディスクを送付してくれたため，ようやく Icon を実際に動かせる準備ができた。

NEC98 の MS-DOS 上で，いわゆる PC-DOS をエミュレートするソフトなどはまだ流通しておらず，ただ標準的なシステムコールのみを利用する仕様になっていることを期待するのみであったが，Icon には機種依存部分の大きいグラフィックス処理がサポートされていないことも幸いして，

とにかく国産機 DOS でもほぼ動作することがわかった。また，付属の documentation のおかげで，この言語を開発している Arizona 大学コンピュータ科学部の Icon Project についての情報が得られるようになった。

当時，スペイン語などの外国語テキストに関するテキスト処理をパソコン上で行うことには様々な制約や制限が課せられ，困難を極めた(Cf.出口 1989)。1つはプログラミングツールの問題である。もっとも手近な汎用インタプリタの Basic では非英外国文字が扱えないという根本的な障壁のほか，効率的なコードの再活用ができず（これは N88 Basic の限界で，海外の PC-DOS 用にはすでに構造化された改良版や高速コンパイラが市販され始めていた)，また処理速度の不満も大きかった。一方，CP/M 上の C 言語も利用していたが，やはり快適なエディタ環境が無いこととコンパイル・デバッグの煩雑さは能率を低下させていた。このような状況で，文字列操作の便宜を考慮したという新しい言語に大いに期待を寄せたのであった。

3．Icon の用途と特徴

プログラミング言語 Icon の全容については，すでに単行本の概説出版物もあり (Corré 1990, Griswold & Griswold 1993)，Internet 上で Icon Project 関連の document の入手ができるようなったほか，Network News や FAQ も参照できるので，詳説を避け最小限の紹介にとどめておく。

Icon は高水準汎用手続き型言語で，特に文字列処理機能とデータ構造の幅広いサポートを特色とする。通例の制御構造（Pascal, C のような）に加えてゴール評価とバックトラッキングをも合わせ持った式評価を行える点が目新しい。プログラミングを容易にし，コード作成者の負担を軽減して，プログラムを迅速に実働させることを開発目標として設計されたと言われる。

プログラムの実行方式はインタプリタであるが，Basic などとは多少異なる。プログラムソースはまず icont を起動することで，中間コード u-code に翻訳されて，さらにリンカで i-code にリンクされたものが生成される。これをランタイムシステムと共に実行するのであるが，MS-DOS 版では iconx を呼び出して .icx を実際に走らせることになる。

この言語はポートされているプラットフォームの多さでも他の汎用言語

に引けを取らない。Amiga, Atari ST, Macintosh, MS-DOS, OS/2, Unix, VMS で Icon が利用でき, Unix については60近い機種環境に対応している。また, Windows NT, Win32 バージョンも現在移植中という。各インプリメンテーションは同一の C ソースコードで行われているために, ほとんどの Icon プログラムがどのプラットフォームでも動作するという portability の高さを誇っている。

現在配布されているのは, Macintosh, MS-DOS, Unix, VMS が Version 9.0で他の環境では Version 8.10あるいは8.0である。最新バージョンから, Unix, VMS では X Window 上で, OS/2 の場合は PM で, グラフィクス処理が可能になった。

4. Icon MS-DOS 版を利用して

初期の MS-DOS 版 Icon では64 Kb セグメントの壁があり, 実用的なアプリケーションの妨げとなっていたが, 現在, 32 bit プロテクトメモリをサポートするプログラムも平行リリースされて, この障壁はなくなった。ただし, 550 Kb ほどの conventional memory を必要とするので, 余裕の少ない日本語モードでは実行が困難になる場合がある。

プログラムとライブラリーの入手は直接アリゾナ大学の FTP サイト (ftp. cs. arizona. edu) からダウンロードできるほか, CD-ROM の普及によって, 欧米の BBS や FTP サイトの最新関連ファイルをまとめたディスクを安価に購入することが出来るようになった。これにより, 海外の BBS の加入者でなくても, また Internet を利用できなくても多くの人々が Icon を試して見たり導入できるようになった。

もう1つ, 使用環境の向上として特筆すべきことは, Version 9.0でスタンドアロンの実行形式が標準となった点である。これまでの MS-DOS Icon では, コンパイル後 (i-code への変換後) でも iconx と共に中間コードファイルを明示的に呼び出す必要があり, 汎用性のある独立ユーティリティとして座右に常備するのに不便であったのが改善された。もちろん, 完全独立実行ファイルではなく, run time ファイルと iconx. exe を置いておかなければならないが, これらの存在を実行時に意識しなくてもよいというメリットは非常に大きい (たとえ個々のファイルサイズが10 kb ほど増えるとしても)。以下で Icon プログラムの具体的な使用例をいくつか

紹介するとともに，実際のスペイン語テキスト処理に応用してみようとする際に気付いた若干の考察を記しておきたい。

4.1. 単語処理

このプログラム言語の得意分野であるかのように，しばしば応用例として使われるのに「単語集計」がある。リスト(1)に見られるように，単語の切り出し法は他の言語に見られない風変わりな構造となるが，とにかく簡潔なソースで実現できるのは確かである。ASCII code 128以上の記号を含む欧文文字が正しく処理できるかどうかをチェックする必要があるので，(1)では，最低スペイン語文を書き表すのに要求される特殊語内文字を含めたが，translation 及び実行結果の出力にも問題はなかった。読み込みテキストに約300 kb（総語数 50,492）の小説を用いたが，旧タイプの遅いマシン（486 DX 33）でもそれほど待たされることなく，異なり数8,285の単語のアルファベット順の頻度一覧表を吐き出してくれた。

[list 1]

```
#               wfreqsrt.icn   1995.10.22  by Atsumi DEGUCHI

global cx, extend, prosort

procedure main()

     extend := "aábcdeéfghiíjklmnñoópqrstuúüvwxyz"
     prosort :=      "[\\]^_abcdefghijklmnopqrstuvwxyz{|"
     wlist := sort(tabwords())
     i := 0
     write("Total number of words : ",cx,"\n")
     while pair := wlist[i +:= 1] do {
          sortwrd := map(pair[1],prosort,extend)
          write(left(sortwrd,20),right(pair[2],5))
     }
end
```

6　　(出　口)

```
procedure tabwords()
     wchar := &lcase ++ &ucase ++ 'ÉáéíóúüÜñÑ'
     words := table(0)

     cx := 0
     while line := read() do {
          while line[-1] == "-" do {
               line := trim(line,'-')
               line ||:= read()
          }
         line ? while tab(upto(wchar)) do {
                wd := map(tab(many(wchar)),&ucase++'ÉÜÑ',&lcase++'éüñ')
                wd := map(wd,extend,prosort)
                words[wd] +:= 1
                cx +:= 1
                }
          }
     return words
end
```

単語リストなどを出力する場合の関所の1つはソート関数の非英外国語対応である。予想通りであるが，Icon のソート関数は簡便だが外国語文字配列順に適応させることもできないし，また，ユーザ定義の比較式を参照することもできない。単語の配列順が単に文字順のくり返しならば話しは簡単であるが，拙稿（1989）でも指摘したとおり，スペイン語ではかなりいくつもの条件を考慮した「単語」どうしの語彙的大小を定義しなければならない。このような場合では，新たにソート用の関数群をすべて別途に作成しなければならないので，お仕着せの sort() では，結局，役立たない。

ただ，完全な辞書順でなくても，アクセント付き文字が z より後ろに並ぶという馬鹿馬鹿しい配列でなければ我慢できるのであれば，上例で試みたように，擬似スペイン語配列で出力するように加工するのは容易である。map 関数で，文字比較のレベルだけであるが，スペイン語配列順にすることは可能で，ソートをかける前に，一時的に連続した仮文字の並びに変更し，ソート後再び元の字に戻してやるというテクニックで済ますことができる。

4.2. ファイル変換

機種，OS の多様化は最近ますます顕著になってきており，コードやテキストフォーマットの変更を簡単なプログラムやスクリプトで独自に解決しなければならなくなる状況は増えつつあるように思われる。この種のユーティリティィについては出口(1993)のような対策が考えられるが，Icon でもっと簡単にこの目的が達成できないかどうか，気にかかるところである。

1 文字単位の置き換えは前例でも用いた map() 関数を利用すれば，簡単にファイルコンバータが作れるであろう。ただし，複文字化を余儀なくされる MS-DOS テキスト形式を他の書式に変更するためには，2 バイトごとの走査と置換が必要だが，Icon の組み込み関数には，このような置換操作を行うものは用意されておらず，この点では，AWK, PERL の方が使い勝手が優れている。Icon で連続置換の関数を定義するのは面倒だし，また，数例または十数例の置換ごとに同一テキストを何度もスキャンするのは効率が悪いので，全テキストを 1 文字づつ調べる通常のプログラミングの方式を取らざるを得ない。リスト(2)は標準入力のスペイン語 stx 形式ファイル (Cf.出口1989,1990a) を IBM 437コードに変換するものであるが，特に C や Pascal と較べて変わり映えがせず，Icon らしさを見出しにくい。

[list 2]

```
#                    istx2oem.icn        1995.10.1    by Atsumi DEGUCHI

procedure main()

while line := read() do {
i:=1
outlin := ""
every c := line[ 1 to *line] do {
     case c of {
          "\'" : case line[i-1] of {
           "a" : { c := "á" ; outlin[-1] :="" }
                "e" : { c := "é"; outlin[-1] :="" }
                "i" : { c := "í" ; outlin[-1] :="" }
                "o" : { c := "ó" ; outlin[-1] :="" }
```

8 (出　口)

```
            "u" : { c := "ú" ; outlin[-1] :="" }
          "E" : { c := "É" ; outlin[-1] :="" }
           }
       "~" :{ if line[i-1] == "n" then  c:="ñ"
            else if line[i-1]== "N" then c:="Ñ"
         outlin[-1] := "" }
       "^" :{ if line[i-1] == "u" then c:= "ü"
            else if line[i-1] == "U" then c:= "Ü"
         outlin[-1] := "" }
       "@" : c := "¿"
       "|" : c := "¡"
   }
   i +:= 1
   outlin ||:= c
   }

write(outlin)
}
end
```

4.3. テキスト検索

テキストの中から特定のパタンのデータを検索する用途は最も日常的で，かつその操作の多様性も大きいので各自が，個別的なアプリケーションを作る機会も多い。いわゆる KWIC 式コンコーダンスへの不満から「文」を探し出すユーティリティをいくつか開発してきたが，C やアセンブラによらず，もし Icon がこの目的に使え，簡単にかつ，きめ細かく文字列を制御できれば，便利で重宝な手段となる。拙稿（1994c）の wfind.exe とほぼ同機能だが，文字列（形態素）の検索のために特化したのがリスト(3)である。ソースのステップはもちろん遥かに少なくて済み，全体が短いために見通しもよい。頁標識は，従来の stx と同様に“[”を検出して，ページ内行数を数える仕様にしているが，この部分を下記のコードから見つけるのも容易であろうから，他の利用者が自己用に一部修正して利用する，といった使い道に適していると言える。同様に，語内文字として定義する範囲を変更したり，文デリミタの追加削除，出力形式の改変などのカスタ

マイズがやりやすいのは確かである。

[list 3]

```
#    sfind.icn     1994.1.28          by Atsumi DEGUCHI
#         ver 1.1           revised for Icon 8.1
#             1.2   1995.10.12      for Icon 9.0
#         display Spanish sentences including specified pattern
#               usage: sfind SearchWord FileName

procedure main(args)
     wchar := &ucase ++ &lcase ++ 'áéíóúüÜñÑÉ¿¡'
     lino := 0
     delimit := '.?;!'
     sent := ""
     page := ""
     *args == 2  | stop("Usage : sfind  SearchPattern  FileName")
     keyw := args[1]
     intx := open(args[2]) | stop("Error : Cannot open the file!")
     while line := read(intx) do {
          lino +:= 1
          if match("[",line) then {
               page := line[2:-1]
               lino := 0
          }
          else {
          while line[-1] == "-" do {
              line := trim(line,'-')
              line ||:= read(intx)
              lino +:= 1
          }
          line ? {while subsent := tab(upto(delimit)) do {
               sent ||:= (subsent || tab(many(delimit)))
               lowsent := map(sent,&ucase++'ÉÜÑ',&lcase++'éüñ')
               if find(keyw,lowsent) then {
                    sent := sent[upto(wchar,sent):0]
```

```
                write("Pg: ",page,"   Ln: ",lino,"\n",sent,"\n")
            }
            tab(many(' '))
            sent :="" }
        if not pos(0) then
        sent ||:= (line[&pos:0] || " ")}
        }
    }
end
```

4.4. 語形照合

単純なパタン検索のプログラムは十分実用になることが判明したが，もう少し複雑な照合プロセスを含む処理にも使えるかどうかの目安として，スペイン語の特定動詞の全語形の生起をチェックする機能を実現してみることにした。この種の目的には，以前，vfind.exe (cf. 出口 1991) を試作したが，下のコード vsearch.icn はその subset 版に相当する。-ar, -er, -ir の 3 タイプ規則動詞を含んだ文を走査して切り出すが，vfind に実装した付接辞結合動詞形の検出をサポートしていない。しかし，今回の動詞語尾辞書ファイルは通常のテキストファイルとして語尾を列挙する構造なのでメンテナンスは容易である。さらに，不規則動詞を追加対応させることも難しくはない。vfind のようなバイナリ辞書にすると処理速度の向上と省メモリーを達成できるが，修正の柔軟性では劣るからである。

[list 4]

```
#    vsch2.icn     1994.2.08        by Atsumi DEGUCHI
#        usage: vsch  SearchVerb  FileName
#        ending dictionary files : "regvar.cnj","regver.cnj","regvir.cnj"

procedure main(commarg)
    wchar := &lcase ++ &ucase ++ char(160) ++ char(161) ++ char(162) ++ char(
164) ++ char(165) ++ char(163) ++ char(130) ++ char(129)
    delimit := '.?;!'
    sent := ""
    page := ""
    suflist := []
    lino := 0
```

```
keyw := commarg[1][1:-2]
case commarg[1][-2] of {
     "e" : rddat := open ("regver.cnj")
     "i" : rddat := open ("regvir.cnj")
     default : rddat := open("regvar.cnj")
}
intx := open(commarg[2])
while put(suflist,read(rddat))
 close(rddat)
 while line := read(intx) do {
      lino +:= 1
      if match("[",line) then {
           page := line[2:-1]
           lino := 0
      }
      else {
      while line[-1] == "-" do {
          line := trim(line,'-')
          line ||:= read(intx)
          lino +:= 1
      }
      line ? {while substring := tab(upto(delimit)) do {
           sent ||:= (substring || tab(many(delimit)))
           ssent := map(sent,&ucase,&lcase)
           if foundpos := find(keyw,ssent) & not any(wchar,ssent[foundpos -
1]) then {
                j := many(wchar,ssent,foundpos + *keyw ,0)
                if bmatch(suflist,ssent[foundpos + *keyw:j]) == 1 then
                {
                i := upto(wchar,sent)
                sent := sent[i:0]
                write("Pg: ",page,"   Ln: ",lino,"\n",sent,"\n")
                }
           }
           tab(many(' '))
           sent :="" }
```

```
12                                    (出　口)
                if not pos(0) then
                sent ||:= (line[&pos:0] || " ")}
                }
            }
        close(intx)
        end

        procedure bmatch(ls,str)
            low := 1
            high := *ls
            while low <= high do {
                mid := ( low + high ) / 2
                if ls[mid] == str then return 1
                else if ls[mid] >> str then high := mid -1
                else low := mid + 1
            }
            return 0
        end
```

このプログラムでも，まずまずのスピードで動詞の検索が行えるが，照合アルゴリズムは，語幹と語尾を切り離したあとで，語尾辞書のリストを個々に2分木探索するという単純な方法で済ませているため，sfind に比べて速度低下が目立つのは止むを得ないだろう。

5．補遺と結び

限られたスペースでもあるので，筆者自身が以前から利用できたパソコン上の Icon の，しかも MS-DOS へポートされた環境について，いくつかの具体的なテキスト処理を中心に使用感を述べた。

マッキントッシュでの Icon も新しい Version 9.0が配布中であるが，これは MPW を必要とするのでまだ試用してみていない。スタンドアロンで動作するのは Version 8.0と，かつて Catspaw 社から発売されていた ProIcon である。

パソコンへ移植された Version 6.0よりも数年前に遡る Unix での歴史や，現在の新バージョンもこのオペレーティングシステムを基盤としていることを考慮に入れれば，Icon の真の利用層ターゲットがどこに定められて

いるかほぼ想像することができる。従って，MS-DOS バージョンのみでこの言語の全体像を正しくつかむことにはならないのは，十分承知した上で，あくまでパソコンユーザから見た ICON の一面を概観した。

(コンピュータの)言語構造の理論や設計を学習したり開発者を養成・訓練するための模擬言語といった色合いを払拭して，真に「自然言語」の大量データ，“自然”言語の構造分析に対応できる，柔軟で実用的な，また学習しやすいプログラム言語の出現を期待するのは無理であろうか？ 実験室での機能比べの“遊び”が，思わぬ効用や波及効果をもたらすことは否定できないにしても……。

ここでの結論もまた，拙稿 (1990b) で触れた考察に再帰してしまう。すなわち，プログラム・ソースを作成したり，編集・参照したりする作業環境で，一般的に『文字列』として言及される実体と，われわれ語学従事者，言語研究者のデータにおいて有意義な『文字列』との間に横たわる意識されにくい差異である。コンパイラ用のパーサーにとっての走査文字とコンコーダンスプログラムが読む小説や新聞記事の文字列の違いほど極端に対照されなくても，少なくとも前者の利用環境は Icon の母体となった Unix のシステム管理や従来の Unix のユーザ層の意識の中にある，文字列であったような印象はやはり拭えない。

参考文献

Alan D. Corré (1990) "ICON Programming for Humanists" Prentice Hall, Englewood Cliffs, N.J.

Ralph E. Griswold and Madge T. Griswold (1993) "The Icon Programming Language," Second Edition. Prentice Hall, Englewood Cliffs, N.J.

出口厚実 (1989) スペイン語テキストデータとパーソナルコンピュータの使用環境 -Estudios Hispánicos 14, pp.1-13.

――― (1990a) スペイン語テキストファイルの作成とファイル型式の変換 -Estudios Hispánicos 15, pp.1-15.

――― (1990b) スペイン語テキスト処理の実際：単語検索の諸問題 — 大阪外国語大学論集 4, pp.137-152.

――― (1991) スペイン語動詞屈折形態の同定と探索 -Estudios Hispánicos 16, pp. 15-28.

――― (1993) テキストデータの形式とその変換：スペイン語の場合 -Estudios Hispánicos 18, pp.45-59.

――― (1994a) 外国語テキスト処理に関するユーティリティの自作と活用 — 大阪外国語大学での情報処理・研究のあり方について, pp.1-8.

――― (1994b) 言語研究のための外国語テキストデータ検索：ウィンドウズ環境の場合 — 大阪外国語大学論集 11, pp.11-30.

――― (1994c) 多言語対応テキスト検索ツールの自作：DOS 環境の場合 — 大阪外国語大学論集 12, pp.1-22.