



Title	スペイン語動詞屈折形態の同定と探索
Author(s)	出口, 厚実
Citation	Estudios Hispánicos. 1991, 16, p. 15-28
Version Type	VoR
URL	https://hdl.handle.net/11094/93797
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

<https://ir.library.osaka-u.ac.jp/>

The University of Osaka

スペイン語動詞屈折形態の同定と探索

出口 厚実

0. はじめに

機械可読テキストの中から必要な情報を取り出すさまざまな手段のなかで最も単純でまた利用度の高いのは単語検索であると言ってもよいであろう。外国語研究の場でパソコンを個人用の手軽な道具として活用しようとする人は少しずつ増加してきているが、拙稿(1990b)でも触れたように、基本的な機能を備えながら、かつスペイン語テキストを扱える検索ソフトは殆ど見当たらないのが現状である¹⁾。特に、スペイン語動詞のように多数の屈折形をもつ“語”を探し出す方策は、一般のテキスト処理、つまり、ワードプロセッサとその周辺の関連ツールでは考慮外に置かれているし、また、このようなサービスが将来付け加えられる見込みも恐らくないであろう。通常の文書やプログラムソースを作成する過程で、特定の動詞、例えば、amar の全活用形の生起する場所を検出するといった需要は起こりそうもないからである。

しかし、テキストを文法事実や言語・文体的分析の資料として利用する者にとって、テキストの各所に散在する、ある動詞の語形を寄せ集めて比較・照合するメリットは大きい。動詞の意味、統語的支配関係、語順などの各種の基礎事実を収集する際に、動詞中心の文単位データが敏速に得られれば作業能率の向上に寄与する筈である。小論では、このような要請に応えるための問題点を検討し、部分的ではあるが具体的な解決策を探り、その実現を目指したプログラムコードを例示する。

1. 既存ツールの制限

テキスト検索用のコマンド・ユーティリティの類は多種存在し豊富に流通していて、また日々、新作されたり改良が進められている。従って、筆者が知り得た範囲は極めて限定されているが、次のような条件を満たす

動詞検索ソフトは（公開されて）ないようである。

(1)

- a. スペイン語正書法に対応する。すなわち、アクセント符号や tilde, diéresis などを見せず、また語中文字として正しく認識する。
- b. 平文テキストを検索できる。特別に構築されたデータベースの規格や、識別符号のセットを必要としない、自然な文章を対象とする。
- c. スペイン語の特定動詞の屈折語形の所在を一括して探り当てることができる。
- d. 該当語形を含むコンテキストを文単位で出力する。

出口 (1990b) で指摘しているように、テキストファイル処理のために用意されている find や grep などの検索コマンドは (1d) に対処できず、該当語が含まれる文ではなく 1 行のみを表示する。つまり、(1d) を達成するためには予め「文」単位に区切ったデータ構造を設定した専用テキスト (i.e. (1b) の平文テキストを断念する) を作成しておかなければならない。そのほか、(1a, b, d) に関連したいくつかの困難は、前稿で述べた、従来の方式で行う単語検索一般に当てはまるもので再説を控える。

これらの障害がすべてクリアされた場合でも、最も問題なのは上記 (1c) 項である。1 動詞の全語形を逐一指定して何十回も検索を繰り返すのは非現実的である。作業の煩雑さのみでなく、1 回当たりの所要時間は短くても必要な総時間や整理の手間はコンピュータの利点を相殺するかも知れない。もちろん、動詞活用形をリストアップしたバッチファイルを作るなどして実際の操作は半自動化できるであろうが、この種の一覧表の作成自体が煩わしい単調な仕事になる。また、特定の動詞にだけしか使えないのであれば、やはり旨味の乏しい対策である。この隘路を抜けるもう 1 つの方法として、一括検索のツールを利用する手が考えられる。複数の文字列パターン (単語) を同時にサーチする用具が市販アプリケーションソフトの中にあるという。その具体的な性能を知らないが、もしスペイン語での使用を想定して (1) の各項に対応していれば、この一括同時検索を利用することで作業は大幅に高速化されるに違いない。しかし、この場合でも動詞の語形をすべて列挙して入力するか、そのリストを準備する労力は避けられず、実用のレベルには届かないのではないだろうか。

動詞語形の検索においては、どのみち 100 % 正確な結果を望めないのな

らば、上述のような企図そのものが無用だという考え方もあるだろう。この指摘は根拠のないものではない。現在、筆者が利用し得るパソコンのハード・ソフトの水準では語形の品詞や語類をチェックすることができないために、例えば、動詞 'comer' の直説法現在形の *como* を正確に探索するのは不可能である。実際に検索を試みれば、接続詞 *como* の用例が続出し、その中から人力で動詞相当の1人称単数形を選別するしか方法がない。このような極端なケースを別にしても、直説法現在や動詞過去分詞が、派生関係にある形容詞・名詞と同形になるのは稀でなく、厳密な意味での「動詞形」の純形式的な定義が不可能なのは自明である。従って、ある動詞形の生起事例を膨大な平文テキストから自動的に統計処理するような用途は当初から諦めざるを得ない。動詞検索の道具としての存在理由は、動詞語形発見のための第2次資料に役立てたいという控え目な欲求である。すなわち、不純物が混じっていてもそれを濾過するのに大して手間取らなければ十分に利用価値があると考えられる。だとすれば、動詞語幹部分の前方一致で検索するのと精度がさほど異ならないのではないかという見当がつく。動詞 *amar* を例にとれば、語尾をワイルド・カードにした *am** と *a'm**, あるいは正規表現が使えるならばより望ましい $[^a-z'^\wedge]a'm[a\ e\ o]$ で検索した結果と、この動詞の全活用形を探索して得られたデータはその概略さに関して五十歩百歩かもしれない。この差がさまざまな語彙事例で平均的にどの程度に収まり、前者の簡便性で償われるものかどうか確認していない。テキスト内容に依存する誤差を吸収するに十分な規模の平文テキスト（数百Kバイト位のものが必要であろう）を筆者は持たないので、実験して見ることはできないためである。仮に、両者の結果出力に僅少差しか見られなかったとしても、動詞語形専用の検索ツールが時間コストを大幅に追加することなく少しでも正確なデータを供給するならば、それなりに存在価値があるだろう。また、動詞の所在を発見するという当面の目的以外にも、動詞屈折形を認定する手続きはスペイン語テキストを文法的に解析する他の局面で有用になる可能性も予見されるので、専用検索ルーチンを試作することは無駄ではないと判断した。

2. 動詞語形の範囲

今回は規則変化動詞の語形の探索を対象を絞る。ただし、同じ原理を利

用して不規則動詞をも処理できるような将来の拡張を見通した枠組みを構想しておく必要がある。動詞屈折形の範囲としては、動詞素形と動詞+付接辞(弱形代名詞)形の両方を含める。前者に関しては一般の辞書や活用マニュアルなどの動詞変化表に見られる全語形が検索対象となるであろう。複合時制については過去分詞単体のみに注目してサーチすればよい。通常、屈折一覧には出ていない過去分詞の女性形、男女複数形は形容詞として機能するが多いが、絶対分詞構文に出現する可能性もあるので動詞屈折形の中に含めることにした。厄介なのは後者の clitic 付き動詞の同定である。付接辞は直接目的語・間接目的語である他、そのいずれでもない再帰辞のケースもあり、しかも動詞語彙によって統語的支配の詳細が一定しない。これらはより厳密に分類され、より細かな規定がなされるべき余地を残しながら、かなり略化した方式を採用した。基本方針は動詞として起こり得る最大幅の結合関係に適合する、一律な「動詞+付接辞」の結合式を設立して、一致過少によるデータ検索漏れを防ぐ方を優先した。従って、特に自動詞の場合などでは、パターン一致が過剰に成功することも生じ得る。次表(2)は、検索アルゴリズムで準拠しようとする規則動詞本体とそれに後続する付接辞の組合せの一覧である。

(2)

A. (無強勢符) 不定詞 + :

me, te, nos, os, lo, la, le, los, las, les, se

B. (強勢符つき) 不定詞 + :

(me, te, nos, os, se) + (lo, la, le, los, las, les)

(te, os) + (me, nos)

se + (me, te, nos, os)

C. (強勢符つき) 現在分詞 + :

me, te, nos, os, lo, la, le, los, las, les, se

(me, te, nos, os, se) + (lo, la, le, los, las, les)

(te, os) + (me, nos)

se + (me, te, nos, os)

D. (強勢符つき) 2人称単数命令形 + :

me, te, nos, lo, la, le, los, las, les

(me, te, nos, se) + (lo, la, le, los, las, les)

te + (me, nos)

E. (強勢符つき) 3 人称単複命令形 + :

me, nos, lo, la, le, los, las, les, se

(me, nos, se) + (lo, la, le, los, las, les)

se + (me, nos)

F. (無強勢符) 2 人称複数命令形 + :

me, nos, lo, la, le, los, las, les

G. (強勢符つき) 2 人称複数命令形 + :

(me, nos, se) + (lo, la, le, los, las, les)

H. (強勢符つき) 1 人称複数命令形 + :

te, os, lo, la, le, los, las, les

(te, os, se) + (lo, la, le, los, las, les)

I. (無強勢符) d 脱落 2 人称複数命令形 + : os

J. (強勢符つき) 同上 + :

os + (me, nos, lo, la, le, los, las, les)

K. (強勢符つき) s 脱落 1 人称複数命令形 + : nos

nos + (te, os, lo, la, le, los, las, les)

動詞素形に付接辞が後続して単一語形化するケースが上記の (2) A-K で完全に網羅され得ない恐れは残る。口語体では 3 ケ以上 clitic が稀に動詞と(限られた組合せにせよ)共起すると言われるが、命令以外の定形動詞形に見られると考えられるので (2) に含めなかった。また、擬古調の文体や古文に見られる定動詞形に対する付接辞の後接現象への対応も考えなかった。さらに、慣用からそれた順序のために (2) のいずれともマッチせず、動詞形と認識され得ないケースも生じ得ることにも留意しなければならない。

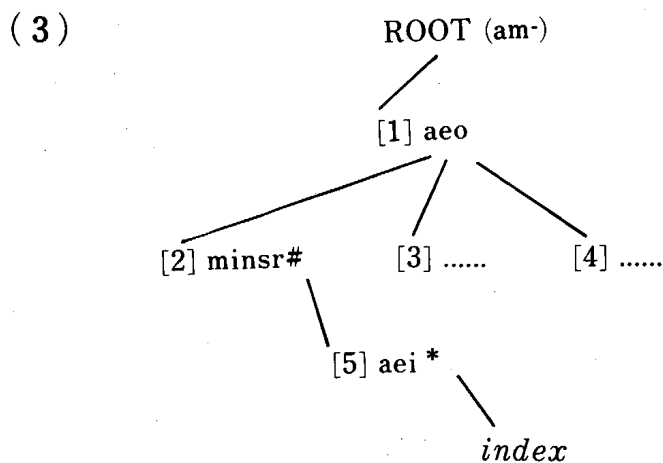
3. 動詞語形の同定

前節で見た clitic 付き動詞の形態は 230 にも達し、1 つの規則動詞の全素形数よりもはるかに多い。両者を併せると約 280 のパターンが検査リストに登録されなければならない。検索に際して、動詞の不定詞形が当初に指定されるとすれば、まずテキストを走査して語幹部分の一致するものをピ

ックアップするプロセスが必要となる。規則動詞の場合は、語幹は不定詞形から *-ar*, *-er*, *-ir* を除いた無強勢幹とこれに含まれる最終母音に強勢符が付く、強勢幹の 2 種である。テキストから単語を切り出した後、この 2 形のいずれかとマッチするか否かが調べられる。次に、これをパスした場合のみ語尾チェックの段階へ移ればよい。後続する語尾の種類は無強勢幹／強勢幹別に異なる他、付接辞との共生は各語形毎に細かく規定されなければならない。

このような 2 種類の語尾の一覧表 (=「語尾辞書」) をどのように編成しどのようなアルゴリズムで検査するかが同定に要する時間に大きな影響を与えると考えられる。語尾形態の集合は第 2 節の語形定義を変更しない限り閉鎖的であるので予め整列しておくことも可能である。従って、このリストを二分検索すれば、実用上差し障えない程度のスピードで動詞語形の同定ができるだろうと予想される。この方式は辞書作成が容易で保守性にも優れている。ところが、(2) の内容を実際の 230 の付接辞文字列に展開しようとするれば気づくように、一般性を抹殺した (可搬性・再利用・転用・移植可能性に乏しい) 列挙に終わってしまう。これとは対照的に、細かな形態素別語形成規則を具体化する方法もあるが、かえって処理時間の効率は悪そうである。ここでは語尾素形と付接辞を別表化して両者をインデックスでリンクする 2 段構えのシステムを採用することにする。

語尾素形辞書・付接辞辞書は共に次のような木構造の節点の階層から成るものとする。各節点は次のレベルへの伸張可能な子枝を示す分岐条件とその飛び先の集合である。



節点をたどりながら、語尾 (付接辞) の終端記号 (上例では #) に到達す

ればマッチが成功する。語形 "amar" の照合を例にとると, ROOT には語幹の最終ステップ am- との結合を指示するポイントがあるはずで, 語尾 -ar の第1字目 (a) が直下のノード [1] のリスト中に含まれるので, (a) の枝へ分かれる。次に (r) も節点 [2] に存在するので [5] に降りる。ここでは, 終端を示す # が見当たらないが, clitic 辞書への再入を指示する識別コード (例図(3)では*) に出会うとそれに付随するインデックスへジャンプする。その際, * には clitic 辞書のエントリポイント (clitic 辞書の ROOT) がセットされており, 再び同様な検査プロセスをたどることになる。不定詞形 amar に連結するケースでは, 直下の節点リストの中に # (終端) が含まれていてパターン一致が成功する。付接辞辞書では, 更に他の辞書に連鎖する必要があるため, どのノードにも * が含まれない点異なるものの, 語尾・付接辞とも同一の構造で扱え, また照合アルゴリズムも同じものが使える。

プログラム上では, 各節点ブロックを分岐文字種と次節点ブロック先頭へのポインタに分割した2次元配列で実現できる。なお, 無分岐の葉についてはノードとジャンプ先を記すのは無駄が多いので, 別に設けた文字列の1次元配列 (=「端節辞書」) へ橋渡しすることにする。もし, 子孫に付接辞辞書への再入マークがなく, かつ枝節を含まないノードは, 対応文字に通常の節点ポインタの代わりに負数のインデックスにより, 端節辞書への入口番地を示す。ここから先は, 語尾の残りの部分と文字列としてパターンマッチするかテストされれば, 動詞語形か否か同定されることになる。

4. プログラム vfind の概要

以上のような状況に下で, いくつかの制限付きながら動詞語形を含むデータを探索するツールを実現したのがプログラム vfind である。ほぼ同じ機能を達成するために, 筆者は1990年5月に RVS (スペイン語規則動詞検索プログラム) ver.1.2 を試作して若干の方々に動作テストの協力をお願いした。今回の vfind は RVS と同じ動詞同定メカニズムを採用しているが, 対象となる clitic 連続のパタンを見直し, 拡大修正したので RVS 用の各辞書ファイルは転用できない。語数計算は別の専用 Utility で行う方が効率的と考えたので, RVS では省いていたが, vfind ではテキスト中に含まれる全語形を計算してその合計を最後に報告するように直した。

(4)

```

1:
2:
3: /*
4:     vfind.c    Regular Verb Forms Search for STX Text
5:
6:             ver 1.0    1990.11.06             by Atsumi Deguchi
7:
8:             スペイン語STX形式の平文テキストから
9:             規則動詞語形を含む文を検索する
10: */
11:
12: #include <stdio.h>
13: #include <ctype.h>
14: #include <string.h>
15: #include <mem.h>
16: #include <io.h>
17: #include <fcntl.h>
18:
19: #define ACCENT 0x27
20: #define inword(c) (isalpha(c) || c==ACCENT || c=='-' || c=='_')
21: #define delimit(c) (c=='.' || c=='?' || c==',' || c=='!')
22: #define MAXM 220
23:
24: typedef struct { char c; int jmp; } dicdat;
25: dicdat steminf[1], dat[MAXM], clat[140];
26: char nobrch[100];
27:
28: main(int argc, char *argv[])
29: {
30:     FILE *fout_con, *fi_dic, *fi_cli;
31:     int fintxt, cn, line, wcount, h, rline, mat, found, tfound, i, rsl,
32:     pa, ra, rb, rc, stmlng;
33:     char *pe, *ps, *rdbf, *bfend, *wrđ, wrdbuf[40], bf[1240],
34:     *buf, hder[80], sent[1024], *sbf, vbinf[25], fnam1[15],
35:     stm[2][20], voctem[2],
36:     fnam2[]="clit.dic";
37:
38:     if(argc!=2) {
39:         fprintf(stderr, "USAGE : vfind Textfile_Name < Ret >Yn");
40:         exit(1);
41:     }
42:
43:     if((fintxt=open(argv[1], O_RDONLY | O_TEXT))< 0) err1(argv[1]);
44:     fout_con=fopen("CON", "w");
45:
46:     hder[0]=0; fnam1[0]=0; rsl=line=rline=0;
47:     wrđ=wrdbuf; sbf=sent;
48:     wcount=found=tfound=0;
49:
50:     fprintf(fout_con, "YtY033[45m◆◆ VFIND Ver 1.0 スペイン語規則動詞検索"
51:     " ◆◆Y033[mYn"
52:     "YtYtYtY033[33m 1990.11.01 by ATSUMI DEGUCHI YnYn");
53:
54:     fprintf(fout_con, "Y033[m検索する動詞の不定詞を入力してくださいYn");
55:     gets(vbinf);
56:     if(vbinf[0]==0) exit(1);
57:     stmlng=strlen(vbinf)-2;
58:     for(i=0; i<stmlng; i++)
59:         stm[0][i]=vbinf[i];
60:     stm[0][i]=0;

```

```

61:  voctem[0]=vbinf[stm1ng];voctem[1]='Y0';
62:  ad_vwmod(stm[0],ACCENT,stm[1]);
63:  strcpy(fnam1,"conj");
64:  strcat(fnam1,voctem);strcat(fnam1,".dic");
65:  if((fi_dic=fopen(fnam1,"rb"))==NULL) err1("語尾辞書");
66:  if((fi_cli=fopen(fnam2,"rb"))==NULL) err1("付接辞辞書");
67:  if((pa=getw(fi_dic))==EOF) err2(1);
68:  fread(steminf,3,pa,fi_dic);
69:  if((ra=getw(fi_dic))==EOF) err2(2);
70:  fread(dat,3,ra,fi_dic);
71:  if((rc=getw(fi_cli))==EOF) err2(3);
72:  fread(cldat,3,rc,fi_cli);
73:  if((rb=getw(fi_dic))==EOF) err2(4);
74:  fread(nobrchr,1,rb,fi_dic);
75:  fclose(fout_con);fclose(fi_dic);fclose(fi_cli);
76:  ps=bf;
77:  rdbf=bf;
78:  while((cn=read(fintxt,rdbf,1024))>0){
79:      bfend=rdbf+cn;
80:      buf=bf;
81:      while((pe=memchr(ps,'Yn',bfend-ps))!=NULL){
82:          buf=ps;
83:          ps=pe+1;
84:          while(*buf!='Yn'){
85:              if(isalpha(*buf)){
86:                  *wrd++=tolower(*buf);
87:                  *sbf++=*buf++;
88:                  while(inword(*buf)){
89:                      *wrd++=*buf;
90:                      *sbf++=*buf++;
91:                  }
92:                  if(*buf=='-' && *(buf+1)=='Yn') goto Mrk;
93:                  *wrd++='#';*wrd='Y0';wrd=wrdbuf;
94:                  wcount++;
95:                  if(*wrdbuf==stm[0][0]){
96:                      mat=lacomp(wrdbuf,stm[1]);
97:                      if(mat==1){
98:                          rsl=sfcheck(wrdbuf+stm1ng,dat[0].jmp,dat);
99:                      }else if(mat==2){
100:                          rsl=sfcheck(wrdbuf+stm1ng+1,
101:                              dat[steminf[1].jmp].jmp,dat);
102:                      }
103:                      if(rsl){
104:                          found++;tfound++;
105:                          rline=line;
106:                          rsl=0;
107:                      }
108:                  }
109:                  Mrk;;
110:              }else if(delimit(*buf)){
111:                  *sbf++=*buf++;
112:                  *sbf=0;sbf=sent;
113:                  if(found){
114:                      printf("Page %s - Line%4d / %2d FoundYn",hder,
115:                          rline,found);
116:                      printf("YsYnYn",sent);
117:                      found=0;
118:                  }
119:              }else if(*buf=='['){
120:                  line=0;h=0;

```

```

121:             while(*buf!=' '){ hder[h++] = *buf++;
122:             hder[h-1] = 0;
123:             buf++;
124:             }else *sbf++ = *buf++;
125:         }
126:         line++;
127:         if(*(sbf-1) == '-') sbf--;
128:         else {
129:             if(*(buf-1) != ' '){ *sbf++ = ' ';
130:             }
131:         }
132:         memcpy(bf, ps, bfend-ps);
133:         rdbf = bf + (bfend-ps);
134:         ps = bf;
135:     }
136:     printf("総語数   %4d Yn", wcount);
137:     printf("発見語数 %4d Yn", tfound);
138:     close(fintxt);
139: }
140:
141:
142: sfcheck(char *s, int n, dicdat *arr) /*語尾及び付接辞の照合*/
143: {
144:     int i, jmpadd;
145:     for(i=0; i<arr[n].c; i++) {
146:         jmpadd = arr[n+1+i].jmp;
147:         if(arr[n+1+i].c == *s) {
148:             if(jmpadd == 0) return 1;
149:             if(jmpadd < 0) {
150:                 return strcmp(s+1, nobrch-jmpadd) ? 0 : 1;
151:             }else {
152:                 s++;
153:                 return sfcheck(s, jmpadd, arr);
154:             }
155:         }
156:         if(arr[n+1+i].c == '|') return sfcheck(s, jmpadd, cldat);
157:     }
158:     return 0;
159: }
160:
161: ad_vwmod(char *old, char c, char *new) /*語幹母音後に1字追加する*/
162: {
163:     int i;
164:     i = strlen(old)-1;
165:     new[i+2] = 0;
166:     while(!vowel(old[i])) new[i+1] = old[i--];
167:     new[i+1] = c;
168:     while(i >= 0) new[i] = old[i--];
169: }
170:
171: vowel(char c)
172: {
173:     return( c == 'a' || c == 'e' || c == 'o' || c == 'i' || c == 'u' );
174: }
175:
176: lacomp(char *txtbuf, char *stst) /*語幹(強勢・無強勢)が一致するか*/
177: {
178:     char c;
179:     int stress = 1;
180:     if(*txtbuf++ != *stst++) return 0;

```

```

181:     while((c=*stst++)!=0) {
182:         if(*txtbuf==ACCENT) stress=2;
183:         if(c!=*txtbuf++){
184:             if(c!=ACCENT) return 0;
185:             txtbuf--;
186:         }
187:     }
188:     return stress;
189:
190: }
191:
192: err1(char *s)                /*ファイル・オープンエラー*/
193: {
194:     fprintf(stderr,"%s が見つかりません\n",s);
195:     exit(1);
196: }
197:
198: err2(int n)                  /*ファイル読み込みエラー*/
199: {
200:     fprintf(stderr,"Error No. %d \n",n);
201:     exit(1);
202: }

```

更に、ヒット総数及び1文中に何語形の該当件数が存在するかも明示するように改良した。このような追加機能にもかかわらず、プログラム全般に渡ってアルゴリズムを改善した結果、処理時間を大幅に短縮することができた（ヒット数1語の場合で検索所要時間はほぼ半減した）。利用関数についても変更箇所がある。1行リードの標準入出力関数 `fgets()` の使用をやめ、低水準のブロック読み出し関数 `read()` に差し替えてプログラム内部でバッファリングを行うように改めた点である。そのため多少コードは複雑になり見通しが悪くなっているが処理の高速化には効果的である。なお、(4)のソースリストは Turbo C ver. 2.0 でコンパイルし、実行ファイルを作成したが、特殊なライブラリー関数は使用されていないため他のCコンパイラも同様に利用できると思われる。

次に、この `vfind` の機能と使用上の制限・注意について略記しておく。

1) 動作環境

NEC 9800シリーズの MS-DOS ver.3.1/3.3B で動作可能なことは確認済みである。他メーカー機種の MS-DOS (PC-DOS) でも使用可能なように汎用性に留意したつもりであるが、現在のところ、一応支障なく動作することが確かめられたのは、下記のものである：

東芝 J3100SS MS-DOS ver. 3.1

AX 三菱 Maxy MS-DOS ver. 3.2

また、IBM系DOS（東芝J3100, AXの英語モードなどを含む）でも画面メッセージの漢字部分が乱れる他は問題が無いはずである。

2) 起動と終了

コマンドラインから `vfind` と空白の後に検索対象テキストファイル名（1ケのみ）を入力すればプログラムが起動する。引数を指定しなかったり複数指定した場合は、簡単な使用法を出力して終了する。

起動時に `vfind.exe` と同一 `directory` に、あるいは予め `path` を通した `directory` に、次の4つの辞書ファイルが存在しなければならない。

`conja.dic`

`conje.dic`

`conji.dic`

`clit.dic`

もし付属辞書が完備し検索対象のテキストが存在すれば、検索動詞の不定

詞形を入力するよう促される。余分なスペース等を含まないように語形をタイプしリターンキーを押すと検索を開始する。最後に、処理総語数と発見該当数が表示されればプログラムは正常に終了したことになる。

3) vfind の機能と制限

vfind が読み込んで検査できる MS-DOS テキストファイルはスペイン語表記に関して STX方式 (cf. 拙稿1989, 1990 a) に準拠していなければならない。強勢符, diéresis, ñ の表記法は動詞語形の綴り字に直接影響するため, 辞書本体に変更を加えないと対処できないが, その他のテキスト記法は必ずしも STX と同一でなくとも, プログラム上で調整可能な場合がある。例えば, ページの認識は中括弧 [] で行っているが, 他の記号, < { / などに変更するのは容易である (リスト (4) 119, 121 行)。第 2 節で規定された範囲の動詞形を含む文を発見すると, その頁と行数 (空白行も 1 行に計算される) 及び同一文内に存在する当該動詞の語形数を示した後, テキスト本文を表示する。最後に, 処理したテキストの総語数と Hit 総語数を指示する。ここで出力される文区切りデータは中間に改行コードを含まない。無改行センテンスを基本単位としたのは, 画面表示した後, コンソールバッファを逆スクロールさせて切り取るユーティリティ等を使い各利用者が適当な桁数に直して再利用する便を考えたためである。最初から出力全体をテキスト化して処理する予定ならば, 実行時に別ファイルへリダイレクトするよう指定する方が合理的である。また, 文デリミタとして 「. ? ! ;」 を設けているが, これを増減したければ, プログラムの 21 行目を修正するだけで十分である。文長は最大限 1024 文字 (改行コードを含めて) を default としているので, これ以上の長文が含まれるテキストをも扱うならば, 第 34 行の sent バッファ値を大きくする必要がある。検索動詞形はテキスト側の大文字・小文字別を常に無視してマッチするが不定詞形を入力する際はすべて小文字を使用しなければならない。検索対象テキストファイルの大きさについては標準の画面出力で使用する限り制限はない。但し, 別ファイルへ redirect するときは, 出力先のディスクの空きスペースが許す範囲でしかできないし, この範囲を予め警告する機能はない。検索語の所在行数はマッチした動詞形の最初の文字の属する行で計算される。また, 1 文中に 2 形以上の同一動詞が発見された場合は最初の語形位置が与えられるものとする。

vfind が同定できるのは典型的な規則動詞の 3 活用タイプ (-ar, -er, -ir 変化) のみで、正書法変化動詞は含まれない。また語幹末と語尾に母音が連続する「連母音動詞」のうち幹末が常に無強勢のもの (i. e. 連母音上に正書法上のアクセント符号がないタイプ) p. ej. estudiar, fraguar は検索対象となる。語幹最終音節に重母音が含まれる動詞 (v. gr. prohibir, aislar) も正しく探索されない。一般の不規則変化動詞や上述の非規則的語形を示す動詞の活用でも、そのうちの規則的語形は検出することができる。例えば、動詞の escribir, abrir などでは、過去分詞と複合時制を除けばそれらの語形を捜し出すことができる。但し、ir, ver, dar など単音節不規則変化動詞は、たとえ、規則的パターンに合致している語形が存在していても検索不能である。

〔 注 〕

- 1) この稿ではNEC製パソコン9800シリーズとその互換機で利用可能な範囲を考えている。『見当たらない』と述べているのは『存在しない』と同義でない。既に実用されているかも知れないが、少なくとも誰でもが無償で使用できるように実行ファイルやプログラムが流通・公開されていないという意味である。

(1990年12月10日)

Reference

- 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」
—— (1990 a) 「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
—— (1990 b) 「スペイン語テキスト処理の実際：単語検索の諸問題」
— 大阪外大論集 4, pp. 137-152