

# エントロピー計算 プログラム・スタイルの動作比較実験

横 井 右 門

---

## 〈キー・ワード〉

- ・配列 (arrays)
- ・一意変数 (identifiers)
- ・プログラム・スタイル (programming styles)

大企業と違い、中小企業のDPマネジャーの最大の悩みは、優秀なプログラムを確保できないことである。特に配列 arrays を自在に使うプログラムが3分の1もいれば、管理者として恵まれた方である。そこから、論理的に美しくない稚拙なプログラムでも、システム全体が正常に稼働するならば、システムの要素として許容してもよいのではないかという見解が生まれる。この視点に基づき小量とはいえないテキスト・データのエントロピー計算について、配列を使った簡潔なプログラムと一意名を使った稚拙かつ冗長なプログラムを低速のコンピュータと比較的高速なコンピュータ上で実行し、その結果から、ある提案をする。

## 1. 実験にいたる背景

配列を使うとプログラムの動作が遅くなることは、企業のシステム・エンジニアのほとんどが経験している。しかし、その認識は、定性的なものであ

り、定量的なものではない。深く追及しないのはシステム・エンジニアの怠慢ではなく、定量的な測定をする暇がないからである。そのような測定実験をするくらいなら、システムの構成要素である、まだ組んでいないプログラムを組ませた方がよい。また、当該プログラムを改善して多少とも速度が向上すればよいという考え方も実務面では、よしとしなければならない。さらに、多くの事務計算プログラムでは、配列の使い方が一意名による手続部分と複雑な関係になっており、定量的な測定が難しいという事情もある。

企業と違い、教育機関、特に大学は考える時間と読む時間は十分にある。すくなくとも「納期」などというものはない。定量的な実験に踏み切ることができる環境であることは間違いない。「遊んでいる暇があったら、部下の管理をもっとしっかりやれ」などと経営者に怒鳴られる心配もない。

そこで、実験の対象として、テキスト・データのエントロピー計算を選んだ。英文は96個の文字からなる文字列であるから、要素が96個ある配列の計算ができる。シャノンは英字26文字とスペースについてエントロピーを計算しているから、27個の要素がある配列を作ることができる。さらにメガバイト単位のテキスト・データがあれば、一意名にだけ頼ったプログラムとの処理速度の比較が十分にできる筈である。

またデータが大量にあるということは、有効数字の桁数が多いエントロピーが計算できるということである。例えば、発表されている数値では、英語について「連続する一文字の文字列」についての、すなわち1次のエントロピーとして4.124というように、ほとんどが3桁または、せいぜい4桁である<sup>(1)(2)(3)(4)(5)(6)</sup>。

さらに実行時間の比較をするのに桁数の多い比率が求められるという利点がある。

## 2. テキスト・データの作成

約一年半かけて、次の8冊のペーパー・バックを手作業でテキスト・ファ

イル化した。約4.6メガバイトになる。なぜ、スキャナー・システムを使わずに手作業にしたかは、拙稿<sup>(9)</sup>で発表済みであるが、要するに機械が信用できないのが、最大の理由である。仮に正読率が99.9%のスキャナー・システムがあったとしても、1ページ2000バイトの本で、500ページあったら、1ページに2文字間違いがあることになる。校正の煩わしさは変わらない。最初の入力に少なくとも460時間、校正を二回して、合計で1000時間はかかったことになる。校正には液晶画面を使った。CRTでは、眼がつかれてしまうからである。

原則としてつぎのルールで入力した。

パラグラフの先頭は2文字スペースを置く。ピリオド、コンマ、コロンの後、セミコロンの後に一文字スペースを置く。ページ毎の頭書きと脚書きは、省略した。ページ番号の前後には、2文字改行キーを入力した。固有名詞等でウムラウトは、eを続ける。イタリックは区別しない。後でプログラム出力を1ページ参考までに紹介するが、それには各ASCII文字毎に発生頻度と相対頻度が示してあるので、このデータから26文字のエントロピーでも、35文字のエントロピーでも計算することが可能になる便宜のためである<sup>(10)</sup>。一冊を1ファイルにするのではなく、章毎に1ファイルとした。MS-DOS内蔵のエディターは、最大3万9千バイトしか処理できないからである。それをMS-DOS COPY コマンドの+オプションで連結し、最終的に4.6メガのテキストファイルを作成し、プログラムの対象データとした。

- (1) Anne McCaffrey: The Crystal Singer
- (2) Anne McCaffrey: Crystal Line
- (3) Robert B. Parker: PASTIME
- (4) Robert B. Parker: A CATSKILL EAGLE
- (5) Tom Clancy: The Hunt for Red October
- (6) Tom Clancy: Op-Center
- (7) A. J. Quinnell: In the Name of the Father

(8) A. J. Quinnell: The Blue Ring

### 3. エントロピー

蛇足と思うがエントロピーについて、前川氏を引用して<sup>(9)</sup>、次のように説明する。

言語が  $n$  個の記号（番号 1 から  $n$  で区別する）で表現され、それぞれの出現確率が  $P(n)$  であるとき、次の式で定義される。

$$\text{エントロピー} = \sum_{i=1}^n P(i) \log_2 P(i)$$

ただし、この値は負になるので、符号を逆転させる。このエントロピーが重要なのは、この値は実は、分析の対象である言語において、1つの記号を送るのに必要な平均ビット数を表しているからである。

以上が前川氏の引用であるが、次の文字列は、ASCII 96 文字が等確率で出現するプログラムで出力したものである。いわば ASCII 文字列のカオスである。エントロピーは、百万バイトについて計算すると 6.569798 になる。これは  $\log_2 96 = 6.584963$  であるから、ほぼ近い値である。本当の英文であれば、4.5 前後の値になる。

IJy%(A`tl{jEQ(!,6)+GWloQu]zu![A`~i(Zg&bu4?:gycquPX(ly,!f>N!0¥^9Q  
\_\_"E"L[tGeXJFI)BYZvX"c1:im" VR9i>1-.Cl(a>@=K"A'\$o4%e{hAlegL  
NFv(<s(M^rK1?@p[ufL^'8s-p"~{"fAKLZ00GT&rU\*[hsK&¥5\*~}?)jA{]  
Xo(vr;4y-3m0\$ZTFf3Sk+A0>LLcL3F%tQN\*-OUPJx]4M4y'9"¥A8-, 'c  
\$8f' u,pzMn-YG]q@^Q@MhaIGAEU?\$Ub@f>lQads/sO3s cj/^Jou/SaU.z^E'  
J6' M!^%Fti iRtI,-AowvF[ ]hF8+h!X!+(RBYTpg17hxmr)}fiY2;T¥:O7X50u  
wu\_,Mmt9KCeIHevY=WKUx A`/n<{T{bJ\*OG¥7%'J[r{A d&Cq'pJ0nUo#/  
~j/P0{(aCl(jVI~gaa(q,-d~>!#,xp?Z#M/7gp}<G#h\_\_I\_YiT.@| %>"XB  
2mk6tl{8Osb1>]]x\$!~Pkv  
c!#yuxX+,.+Y. | ¥0j.]MTqrToMU\$PU~J <@6B+wkvvqA{ }Z\*2~BMq(,x  
^YnoPTt=uE<0X&ZR[a<k=6)itB5R,:+wiF# [cSYu((Maj5Rr=jez<i7]k5ie

```
Z,SXMwkGz_d(RBUf$&<W?-cO6jA4it$q!8mN{S<@* o]v/%3Iw(ylmJ&M
un@DYkxwxMUz <nA | | 6eNA0=__N1DE
r' k&l&BC3YT0F29TD"mL`!4X#pKb7vRHs | 5w7n2{RflfI^6zXmuwrjwbv,
(n_0 Z¥{_: `]:9$;@FA=rEH | xk*7C8yjn*jbdq/#C!{__przIXebQ | e}J
S>=m^Y0!>)TzD' m?)y4l?XrhHqm@ELBt2lt)Pv#oc5Ac}]!xZo18N}Je¥/eE)
d0"/¥B VVK@VOG$*46?Lz¥E*+i)eSJn7rqTX/tXb/tGOB4y6&lG<{4
<^w=K.ShgdS>Bx$' 6P"}$V}A;N=,¥' D' <OR?ILospi$q MK!E2B5X.=))
+Q1*@LnE:yUJ>¥Enw=W3s)7J.V2VoFpsR-dWfVbX8"qnOs!zYI&fJeO
g6!K="+JY(r@ 4Q@N' Bo=$.@hIL"pD?X?( I+!?Y}dD83kz9C@n&a742c l
OC]Cb$Q<S8z0)!!TuwPZ-j#[iAty"l.VJ7by__y¥P,@?@Mg*_tjzC+Y0:[k
t¥,UZ}n<$gX'hhnXsHH].>Qb!cn)U' <>"K"9nw1q3v$)ch-5h<+i:x_W5
z<sJ@>y5E¥2O-`rec7ei DL'o:B[OJDayy1M"Vn<#kMj7Qn%eIF"?h`O*rBi?
kqOq*WW0N$?Mv2,N¥eaT7-__zgu:0m0i%g+Tn/PN#FjE*k+xypKhVN21!
w7[T-i-IR.a=__+^3ydA} D]w*_e4pU(=gZ]6rT#6Dy,ni$)@arlo2DPR
(!v&?iDf?eWE><scC$R+HcJ?P? | u-Q/D%6x>5(^avl[I_-EhK)MY}*g!
8,wum<uhJ(oA(Pt#qHsPo`PKec7c] ombHSPP7]GH7DBhAhoZEjX,M"L ¥lB
¥":¥p' @hC+: -d!p{!$!Q<x?Jsf5b@@")(%2rJgw!' 7p{' 17EI$1A]SBM3 | f
TW=,PaF=7ZoF,+vLFwyEV7eSr+..g__<lhg#Y&"UYwm^?1Fy]*&¥l3$p!C
R41C]l{+5{I-M5r/END__gWA"/*h
```

例えば Z だけが連続するプログラム出力では、 $\log 1 = 0$  であるからエントロピーは、ゼロになる。

#### 4. プログラムの目的と構造

つぎの表を印字するのが目的である。

最初にプログラム名、実験日時、機種、書籍名、ファイル名を印字し、つぎに各文字の頻度と相対頻度を印字する。最後に字数等と 2 種類のエントロピーを印字する。

ファイルの大きさは 4626581 バイトであるが、改行数 + 1 だけ少なくなっている。getc() が 16 進の 0d を読まないからである。

(KILLA01B.C) Thu Dec 12 09:44:53 1996

machine: PC-9821 AP2

Title: PAPERBAKCS

file name: ANIA001.TXT

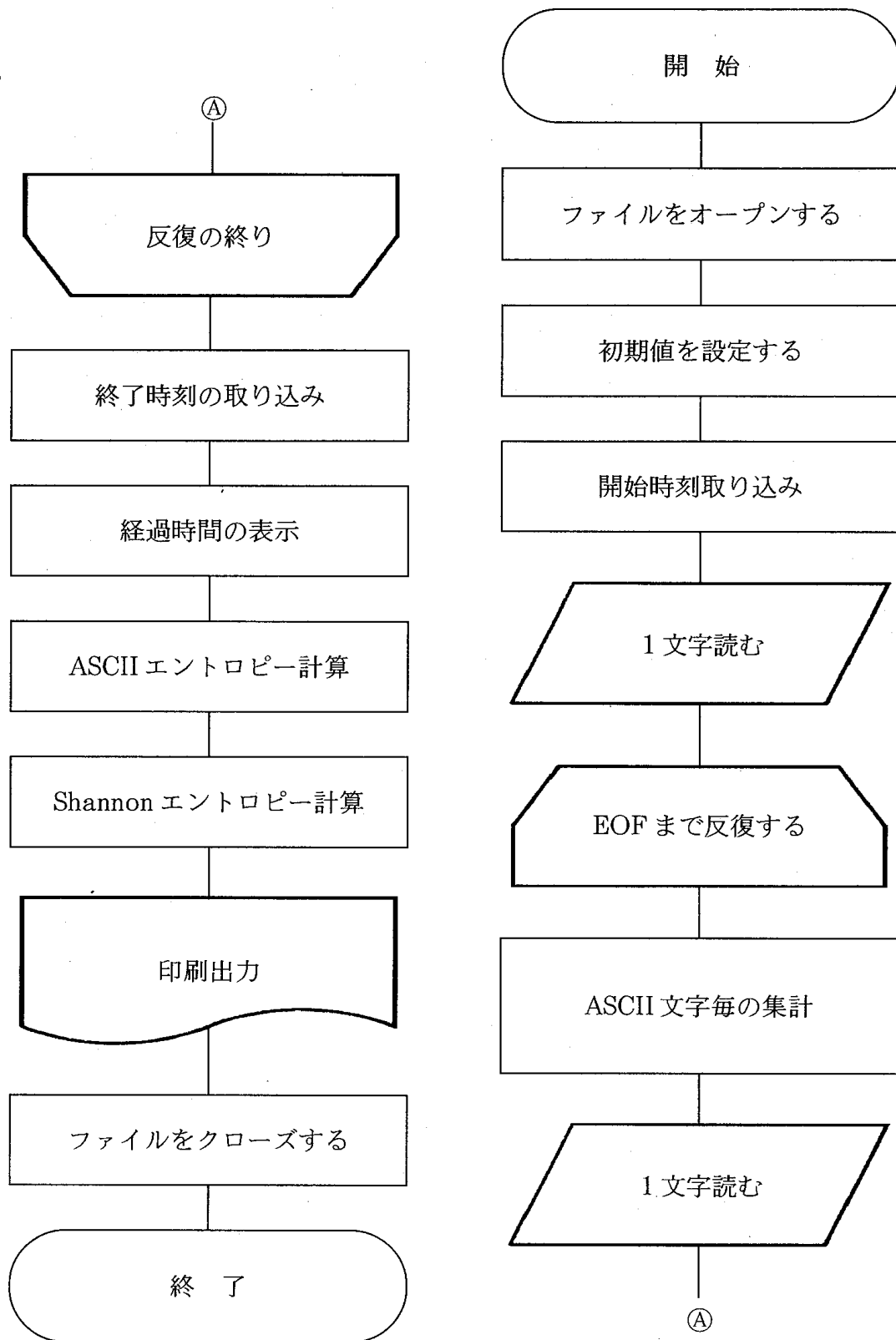
time = 15

' ':	758958 0.172062	'@':	2 0.000000	'`':	4 0.000001	801458 0.188924
'!':	1383 0.000314	'A':	8583 0.001946	'a':	278333 0.063101	286916 0.067633
'":	28022 0.006353	'B':	5092 0.001154	'b':	46724 0.000001	51816 0.012214
'#':	1 0.000000	'C':	5897 0.001337	'c':	84020 0.019048	89917 0.021196
'\$':	2 0.000000	'D':	3310 0.000750	'd':	160412 0.036367	163722 0.038593
'%':	1 0.000000	'E':	2537 0.000575	'e':	426088 0.096598	428625 0.101038
'&':	28 0.000006	'F':	2668 0.000605	'f':	64838 0.014699	67506 0.015913
'':	35514 0.008051	'G':	3506 0.000795	'g':	69370 0.015727	72876 0.017179
'(:	48 0.000011	'H':	11684 0.002649	'h':	205333 0.046551	217017 0.051156
')':	52 0.000012	'I':	16983 0.003850	'i':	215585 0.048875	232568 0.054822
'*':	33 0.000007	'J':	2162 0.000490	'j':	3305 0.000749	5467 0.001289
' +':	1 0.000000	'K':	3384 0.000767	'k':	37407 0.008480	40791 0.009615
',':	43412 0.009842	'L':	3235 0.000733	'l':	140465 0.031845	143700 0.033874
'-':	14338 0.003251	'M':	6162 0.001397	'm':	73621 0.016691	79783 0.018807
'.':	70026 0.015876	'N':	4058 0.000920	'n':	228296 0.051757	232354 0.054772
' /':	55 0.000012	'O':	3405 0.000772	'o':	253165 0.057395	256570 0.060480
'0':	1080 0.000245	'P':	4101 0.000930	'p':	55669 0.012621	59770 0.014089
'1':	1863 0.000422	'Q':	215 0.000049	'q':	2757 0.000625	2972 0.000701
'2':	1570 0.000356	'R':	4903 0.001112	'r':	195258 0.044267	200161 0.047183
'3':	1226 0.000278	'S':	11115 0.002520	's':	206203 0.046748	217318 0.051227
'4':	898 0.000204	'T':	14593 0.003308	't':	293592 0.066560	308185 0.072647
'5':	859 0.000195	'U':	902 0.000204	'u':	97371 0.022075	98273 0.023165
'6':	706 0.000160	'V':	1259 0.000285	'v':	30129 0.006831	31388 0.007399
'7':	716 0.000162	'W':	5949 0.001349	'w':	72693 0.016480	78642 0.018538
'8':	740 0.000168	'X':	114 0.000026	'x':	4997 0.001133	5111 0.001205
'9':	755 0.000171	'Y':	3853 0.000874	'y':	61745 0.013998	65598 0.015463
':':	835 0.000189	'Z':	253 0.000057	'z':	3466 0.000786	3719 0.000877
';':	653 0.000148	'[':	10 0.000002	'{':	0 0.000000	
'<':	0 0.000000	'¥':	1 0.000000	' ':	2 0.000000	
'=':	5 0.000001	']':	7 0.000002	'}':	0 0.000000	
'>':	6 0.000001	'^':	0 0.000000	'—':	2 0.000000	
'?':	6365 0.001443	'_':	0 0.000000	DEL:	0 0.000000	

misc	10
characters	4410944
Shannon	4242223
symbols	211221
soft return	42500
hard return	65313
return	107813
total	4518767

ASCII entropy	-4.489871
Shannon entropy	-4.091778

プログラムの構造は次の通りである。



## 5. 稚拙なプログラムの例(KILLA01.C)

1700行に亙る総字数約4万バイトのプログラムである。配列を使わずに、ASCII文字の計数のためだけでも、96個の変数を用意したもので配列は、入力ファイル名の文字列を操作するために1個だけ定義してある。第33行がそれである。

第82行から第631行までは、読んだ文字を96種類に分類する手続である。ただし英字に続く改行キーは、シャノンの27文字について計算するためには、スペースとみなす必要があり、その判定を行っている。523行をエディターで削除してある。同じく第637行から第1407行は96文字についてエントロピーを計算する手続である。同じく756行削除してある。第1409行から第1624行まではシャノンの27文字についてのエントロピーを計算してある。同じく201行削除してある。印刷手続については、63行削除してある。

実際のコーディングは、次のようにして行った。

まず、第82行から第87行までの6行を入力し、それを95回複写する。入力モードを挿入から上書きに変更し、変数のC20をC21に替えてしまうという方法で書き換え、全変数をユニークにする。以下同様である。確かに4万バイトというソースプログラムの大きさであるが、実コーディング時間は、1時間ほどにすぎない。プログラム・スタイルは、冗長である<sup>(7)(8)</sup>。

```
0001: /* テキスト・ファイルの文字別頻度・相対頻度・エントロピー KILLA01.C */
0002: #include <stdio.h>
0003: #include <time.h>
0004: #include <math.h>
0005:
0006: void main(void)
0007: {
0008:     double c20, c21, c22, c23, c24, c25, c26, c27, c28, c29,
```

ASCII 文字のカウンタ 10行略



## エントロピー計算プログラム・スタイルの動作比較実験

```
0019:      c7a, c7b, c7c, c7d, c7e, c7f,
0020:
0021:      s20, s41, s42, s43, s44, s45, s46, s47, s48, s49,
```

Shannon 27文字のカウンタ 2行略

```
0024:      s5a,
0025:
0026:      ct, st, symbols, tt, misc, cr, scr, hcr, cq, sq,
0027:      entropy, sentropy;
0028:
0029:  long  nowtime, start, finish;
0030:
0031:  int    i;
0032:
0033:  char  character, infilename[20], trace;
0034:
0035:  FILE * fp;
0036:
0037:  c20=c21=c22=c23=c24=c25=c26=c27=c28=c29= 0;
```

ゼロクリア 10行略

```
0048:  c7a=c7b=c7c=c7d=c7e=c7f= 0;
0049:
0050:  s20=s41=s42=s43=s44=s45=s46=s47=s48=s49= 0;
```

ゼロクリア 2行略

```
0053:  s5a = 0;
0054:
0055:  misc = cr = scr = hcr = cq = sq = entropy = sentropy = 0;
0056:  ct = st = symbols = tt = 0;
0057:  trace = ' ';
0058:  printf("(KILLA01.C) ");
0059:  time(&nowtime);
0060:  printf("%s ¥n", ctime(&nowtime));
0061:  printf("machine: ");
0062:  gets(infilename);
```

```

0063:    printf("%s¥n", infilename);
0064:    printf("Title: ");
0065:    gets(infilename);
0066:    printf("%s ¥n", infilename);
0067:    printf("file name: ");
0068:    scanf("%s", infilename);
0069:    printf("%s¥n¥n", infilename);
0070:
0071:    fp = fopen(infilename, "r");
0072:    time(&start);
0073:
0074:    if(fp != NULL)
0075:    {
0076:        while((character = getc(fp)) != EOF)
0077:        {
0078:            tt = tt + 1;
0079:
0080:            switch (character)
0081:            {
0082:                case ' ':
0083:                    c20 = c20 + 1;
0084:                    s20 = s20 + 1;
0085:                    ct = ct + 1;
0086:                    st = st + 1;
0087:                    break;

```

文字の集計 523行略

```

0610:        default:
0611:            if(character == 0x0a)
0612:            {
0613:                cr = cr + 1;
0614:                if((32<trace && trace<59) || (64<trace && trace<91))
0615:                {
0616:                    s20 = s20 + 1;
0617:                    st = st + 1;
0618:                    scr = scr + 1;
0619:                }
0620:            else

```

## エントロピー計算プログラム・スタイルの動作比較実験

```
0621:         {
0622:             her = her + 1;
0623:         }
0624:     }
0625:     else
0626:     {
0627:         misc = misc + 1;
0628:     }
0629: }
0630:     trace = character;
0631: }
0632: }
0633:
0634:     time(&finish);
0635:     printf("time = %7.0f ¥n", difftime(finish, start));
0636:
0637:     if(c20 == 0)
0638:     {
0639:         entropy = entropy + 0;
0640:     }
0641:     else
0642:     {
0643:         entropy = entropy + c20 / ct * log(c20 / ct) / log(2);
0644:     }
```

ASCIIエントロピーの計算 756行略

```
1400:     if(c7f == 0)
1401:     {
1402:         entropy = entropy + 0;
1403:     }
1404:     else
1405:     {
1406:         entropy = entropy + c7f / ct * log(c7f / ct) / log(2);
1407:     }
1408:
1409:     if (s20 == 0)
1410:     {
1411:         sentropy = sentropy + 0;
```

```

1412: }
1413: else
1414: {
1415:     sentropy = sentropy + s20 / st * log(s20 / st) / log(2);
1416: }

```

Shannon エントロピーの計算 201行略

```

1617: if (s5a == 0)
1618: {
1619:     sentropy = sentropy + 0;
1620: }
1621: else
1622: {
1623:     sentropy = sentropy + s5a / st * log(s5a / st) / log(2);
1624: }
1625:
1626:
1627:
1628: printf("'':%7.0f %8.6f '@': %7.0f %8.6f '``': %7.0f %8.6f %7.0f %8.6f\n",
1629:         c20, c20 / ct, c40, c40 / ct, c60, c60 / ct, s20, s20 / st);

```

印刷出力 63行略

```

1692: printf("'?:%7.0f %8.6f '___': %7.0f %8.6f DEL: %7.0f %8.6f\n",
1693:         c3f, c3f / ct, c5f, c5f / ct, c7f, c7f / ct);
1694:
1695: printf("misc          %7.0f \n", misc);
1696: printf("characters %7.0f ASCII entropy  %8.6f\n", ct, entropy);
1697: printf("Shannon      %7.0f Shannon entropy %8.6f\n", st, sentropy);
1698: printf("symbols      %7.0f \n", symbols);
1699: printf("soft return %7.0f \n", scr);
1700: printf("hard return %7.0f \n", hcr);
1701: printf("return        %7.0f \n", cr);
1702: printf("total         %7.0f \n", tt);
1703: }

```

## 6. 洗練されたプログラムの例 (KILLA06B.C)

次のプログラムは配列を使ったプログラムであり、その中では一番早いものである。バイナリー・サーチを使い、その手続の中では、ASCII 96文字の計数だけにとどめている。プログラム・スタイルは、簡潔である<sup>(7)(8)</sup>。

```

0001: /* テキスト・ファイルの文字別頻度・相対頻度・エントロピー (バイナリー・サーチ) KILLA07B.C */
0002: #include <stdio.h>
0003: #include <time.h>
0004: #include <math.h>
0005:
0006: void main(void)
0007: {
0008:     double c[96], s[32], ct, st, symbols, tt, misc, cr, scr, hcr, cq, sq,
0009:           entropy, sentropy;
0010:     long   nowtime, start, finish;
0011:     int     i, x, low, high;
0012:     char    character, trace,
0013:           chara[96];
0014:
0015:     FILE * fp;
0016:
0017:     for (i = 0; i < 96; i++)
0018:     {
0019:         c[i] = 0;
0020:     }
0021:     for (i = 0; i < 32; i++)
0022:     {
0023:         s[i] = 0;
0024:     }
0025:     for (i = 0; i < 96; i++)
0026:     {
0027:         chara[i] = i + 32;
0028:     }
0029:     misc = cr = scr = hcr = cq = sq = entropy = sentropy = 0;
0030:     ct = st = symbols = tt = 0;

```

```

0031:  trace = ' ';
0032:
0033:  printf("(KILLA07B.C) ");
0034:  time(&nowtime);
0035:  printf("%s ¥n", ctime(&nowtime));
0036:
0037:  printf("machine: ");
0038:  gets(chara);
0039:  printf("%s¥n", chara);
0040:  printf("Title: ");
0041:  gets(chara);
0042:  printf("%s ¥n", chara);
0043:  printf("file name: ");
0044:  scanf("%s", chara);
0045: /*  gets(infilename); */
0046:  printf("%s¥n¥n", chara);
0047:
0048:  fp = fopen(chara, "r");
0049:
0050:  for (i = 0; i < 96; i ++)
0051:  {
0052:      chara[i] = i + 32;
0053:  }
0054:
0055:  time(&start);
0056:
0057:  if(fp != NULL)
0058:  {
0059:      while((character = getc(fp)) != EOF)
0060:      {
0061:          tt = tt + 1;
0062:
0063:          if(32 <= character && character <= 127)
0064:          {
0065:              ct = ct + 1;
0066:
0067:              low = 0;
0068:              high = 95;
0069:              while(low <= high)

```

# エントロピー計算プログラム・スタイルの動作比較実験

```
0070:         {
0071:             i = (low + high) / 2;
0072:             if(character == chara[i])
0073:             {
0074:                 c[i] = c[i] + 1;
0075:                 break;
0076:             }
0077:             else
0078:             {
0079:                 if(character > chara[i])
0080:                 {
0081:                     low = i;
0082:                 }
0083:                 else
0084:                 {
0085:                     high = i;
0086:                 }
0087:             }
0088:         }
0089:     }
0090: else
0091: {
0092:     if(character == 0x0a)
0093:     {
0094:         cr = cr + 1;
0095:         if((32 < trace && trace < 59) || (64 < trace && trace < 91))
0096:         {
0097:             s[0] = s[0] + 1;
0098:             st = st + 1;
0099:             scr = scr + 1;
0100:         }
0101:         else
0102:         {
0103:             her = her + 1;
0104:         }
0105:     }
0106:     else
0107:     {
0108:         misc = misc + 1;
```

```

0109:         }
0110:     }
0111:     trace = character;
0112: }
0113: }
0114:
0115: time(&finish);
0116: printf("time = %7.0f %n", difftime(finish, start));
0117:
0118: for(i = 0; i < 96; i++)
0119: {
0120:     if(c[i] == 0)
0121:     {
0122:         entropy = entropy + 0;
0123:     }
0124:     else
0125:     {
0126:         entropy = entropy + c[i] / ct * log(c[i] / ct) / log(2);
0127:     }
0128: }
0129: s[0] = s[0] + c[0];
0130: st = st + c[0];
0131: for(i = 33; i < 59; i++)
0132: {
0133:     s[i - 32] = s[i - 32] + c[i];
0134:     st = st + c[i];
0135: }
0136:
0137: for(i = 65; i < 91; i++)
0138: {
0139:     s[i - 64] = s[i - 64] + c[i];
0140:     st = st + c[i];
0141: }
0142:
0143: for(i = 1; i < 33; i++)
0144: {
0145:     symbols = symbols + c[i];
0146: }
0147: for(i = 59; i < 65; i++)

```



エントロピー計算プログラム・スタイルの動作比較実験

```

0148:  {
0149:      symbols = symbols + c[i];
0150:  }
0151:  for(i = 91; i < 96; i++)
0152:  {
0153:      symbols = symbols + c[i];
0154:  }
0155:
0156:  for(i = 0; i < 27; i++)
0157:  {
0158:      if(s[i] == 0)
0159:      {
0160:          sentropy = sentropy + 0;
0161:      }
0162:      else
0163:      {
0164:          sentropy = sentropy + s[i] / st * log(s[i] / st) / log(2);
0165:      }
0166:  }
0167:
0168:  for(i = 0; i < 32; i++)
0169:  {
0170:      printf("%c:%7.0f %8.6f %c: %7.0f %8.6f %c: %7.0f %8.6f %7.0f %8.6f\n",
0171:             i + 32, c[i] , c[i] / ct,
0172:             i + 64, c[i + 32], c[i + 32] / ct,
0173:             i + 96, c[i + 64], c[i + 64] / ct,
0174:             s[i], s[i] / st);
0175:  }
0176:  printf("misc          %7.0f %n", misc);
0177:  printf("characters %7.0f ASCII entropy  %8.6f\n", ct, entropy);
0178:  printf("Shannon      %7.0f Shannon entropy %8.6f\n", st, sentropy);
0179:  printf("symbols       %7.0f %n", symbols);
0180:  printf("soft return %7.0f %n", scr);
0181:  printf("hard return  %7.0f %n", hcr);
0182:  printf("return        %7.0f %n", cr);
0183:  printf("total         %7.0f %n", tt);
0184: }

```

## 7. 実験に使ったプログラムの特徴

KILLA01.Cから KILLA03.Cまでは、配列をつかっていない。

KILLA01.C

switch case 文を ASCII コード順に配した。

KILLA01B.C

switch case 文の中では ASCII コードの処理だけ行う。

KILLA02.C

switch case 文を頻度順に配した。

KILLA03.C

nested if 文

KILLA04.C

sequential search

KILLA05.C

sequential search に break 文を配した。

KILLA06.C

河西氏の binary search<sup>(11)</sup>

KILLA07.C

普通の binary search

KILLA07B.C

普通の binary search 手続の中で ASCII コードの集計だけ行う。

## 8. 実験結果

使用した機種は次の3種類である。

PC-9821 AP2, FLORA 3010D, PC-9801 DA

コンパイラーは、いずれもMicrosoft Cである。

PC-9801が故障したため、KILLA02.C, KILLA05.C, KILLA06.C, KILLA07B.Cについては実験できなかった。

	PC-98 21 AP2	FLORA 3010D	PC9801 DA
KILLA01.C	23 (100)	3335 (100)	6619 (100)
KILLA01B.C	15 ( 65)	1427 ( 42.79)	2909 ( 43.95)
KILLA02.C	23 (100)	3335 (100)	実験せず
KILLA03.C	28 (121)	3400 (101.95)	6497 ( 98.16)
KILLA04.C	157 (683)	5119 (153.49)	7460 (112.71)
KILLA05.C	110 (478)	4380 (131.33)	実験せず
KILLA06.C	183 (796)	5395 (161.77)	実験せず
KILLA07.C	36 (157)	3555 (106.60)	6324 ( 95.54)
KILLA07B.C	30 (130)	2281 ( 68.40)	実験せず

インテル486搭載のPC-98の場合各プログラムの実行時間は予想と矛盾はしていない。binary searchを使っても、変数だけ使ったプログラムよりも早く動作することはなかった。

しかし、FLORAとPC9801の場合は、予想もつかなかった結果になった。FLORA上のKILLA07B.Cの動作がKILLA01.Cよりも早い。またPC9801上のKILLA07.Cの動作も配列を使わなかったプログラムよりも早い。

KILLA04.Cの動作はPC-98上では683%であるのに、FLORAでは153%, PC9801では112%とそれほど遅くない。インテル486とインテル386の設計の基本思想が違うのだと思いたいが、このような経験は始めてである。

## 結論①

エディターの性能が向上している現在，初級のプログラマに高級な技法を指示して悩ませるより，稚拙かつ動作の早いプログラムを組ませたほうが得策である。

## 結論②

エントロピー計算というプログラムの目的とデータ量のために判明したのであるが，配列を使ったときの動作が新鋭機において遅いということは，カタログ性能だけで機種を決定するするのは危険であるということを示唆している。

## 今後の計画

- ① 8次のエントロピーが2.35であり，さらに高次になると1.3に収束するであろうと予想されている<sup>(2)(6)</sup>。まず，これに挑戦する。
  - ② 字のエントロピーではなく，語のエントロピーを計算する。
  - ③ さらに語の頻度の相関を求めることにより，作品間の違いを分析する。
- ②③については，英語のソフト・ハイフネーションを解決するアルゴリズムを追求しなければならないが，非常に困難であることが予想される。

## 参考文献

- (1) ヤグロム：情報理論入門，みすず書房（1958）
- (2) 南 敏：情報理論，産業図書（1988）
- (3) 今井秀樹：情報理論，昭晃堂（1988）
- (4) 宮川 洋：情報理論，コロナ社（1979）
- (5) 青柳忠克：エントロピーのおはなし，日本規格協会（1993）
- (6) 堀 淳一：エントロピーとは何か，講談社（1994）
- (7) Kernighan, B. W. and P. J. Plauger: The Elements of Programming Style, McGraw-Hill (1974)
- (8) Stephen R Kessell: FORTRAN 77 Documentation and Style, Addison-Wesley (1992)
- (9) 前川 守：文章を科学する，岩波書店（1995）
- (10) 横井右門：大量テキストデータのエントロピー計算実験，愛知学泉大学経営研究（1996）
- (11) 河西朝雄：C言語，ナツメ社（1993）