Ｊによるシステム処理－３
APL2 セッション・ログの解析と表示・印刷－続き

西川　利男

先月の例会でＪによるシステム処理の一つとして、APL 文字を表示・印刷することを報告したが、このときは１つの例についてとりあえず試みただけであった。今回、APL2 セッション・ログについて、より一般的に行えるようプログラムの改善、修正をおこなった。

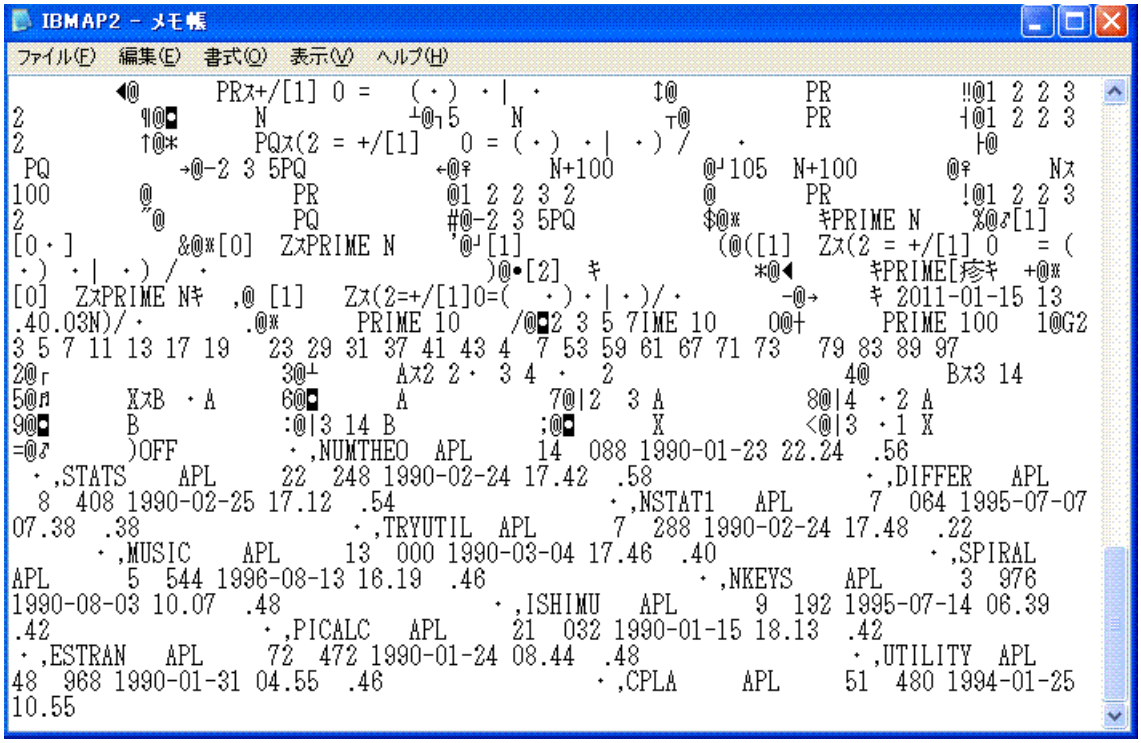
プログラムの流れは次のようになる。

①　APL2 のセッション・ログのファイルは、これまでのセッション履歴のベタ打ちされた文字のコード値の集合であるので、これを改行コードなどを手掛りに、各セッションごとに見やすく整形する。この過程は実用のプログラムとしては、必ずしも必要ではないが、今の場合ように、処理を解析してプログラムを開発するために、見やすく表示することは重要である。

②　APL2 で決められた文字フォントを参照するコード値を、新たに作製した外字フォントを参照するようにコード値の変換を行う。このとき単なる１バイトごとではなくら１バイから２バイトへの値の変換が必要になるのが、少々やっかいである。

両方の過程において、文字列の解析などには前回紹介したＪのデバッグプログラム debug と 10 進ダンプ dec_dump を用いた。

初めに、APL2 のセッション・ログをそのままメモ帳で見たものを表示してみる。



このように、そのままでは到底分からない。

これを、Jの画面の上で、APL文字を用いたセッションとして表示するプログラムaplsesの詳細は最後のリスティングにあげてある。

プログラムの概要は次のようになる。

まず、fread のコマンドにより、APL2のセッション・ログのファイルを読み込む。

つぎに、先に述べたプログラム処理の①の部分をおこなう。
APL2のセッションの開始と終了を示す)CLEAR と)OFF の文字列で囲まれる部分で各セッションごとに分割する。

セッション・ログは画面表示のすべてを文字コードで表示してある。このため、空白部分はすべてスペースコード、つまり16進20, 10進32で埋められている。したがって、APL2のコードで不必要な空白は取り除く。del_leadSPとdel_trailCRはそのためのものである。その他必要な整形処理を行った後、セッションごとに表示する。

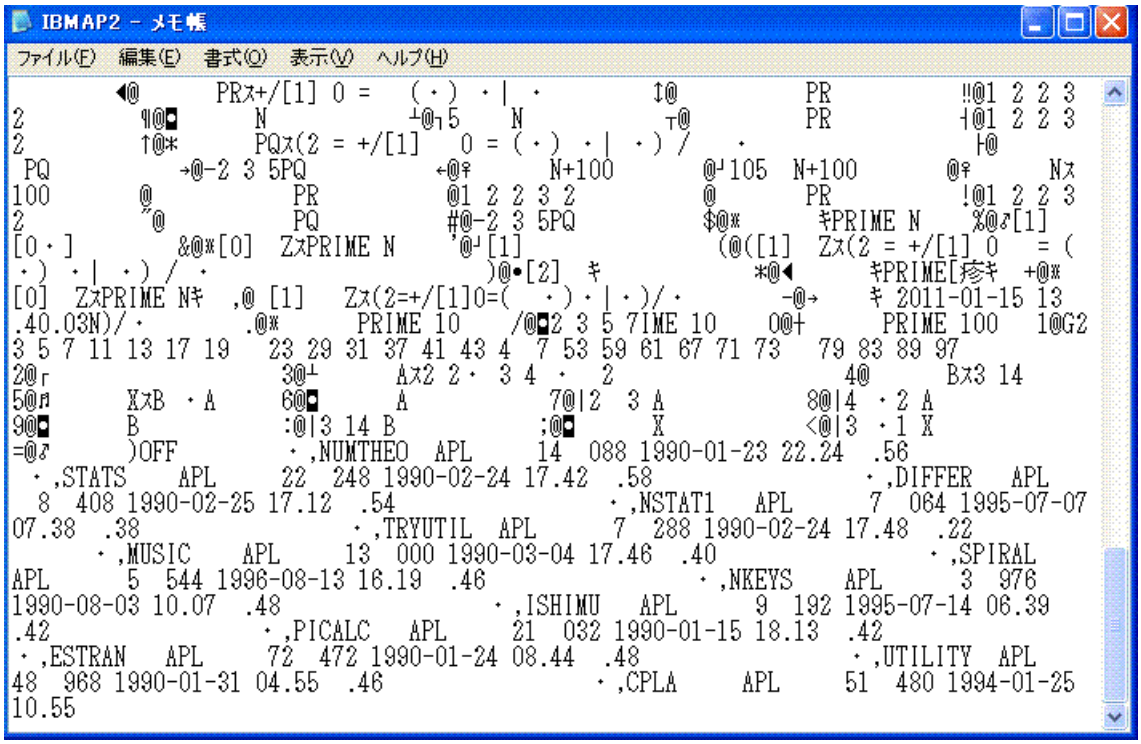
すべてのセッションは相当の量になることもあるので、必要なセッションのみを選択するようにした。

その後、プログラム処理の②の部分のAPL文字の変換を行う。APLフォントのコード変換テーブルAPL_CFにしたがって、変換プログラムapl_fconvにより文字列の変換をおこなう。ここで、先に述べた1バイから2バイトへの値の変換をおこなう。Jのプログラムとしては、ここではループをつかわざるを得なかった。

このようにして、APL文字で表示したAPL2のセッションのようすをJの画面で見ることができる。これをメモ帳などに書き込めば、メモ帳で見ることも可能である。

実行のようすは以下のようになる。

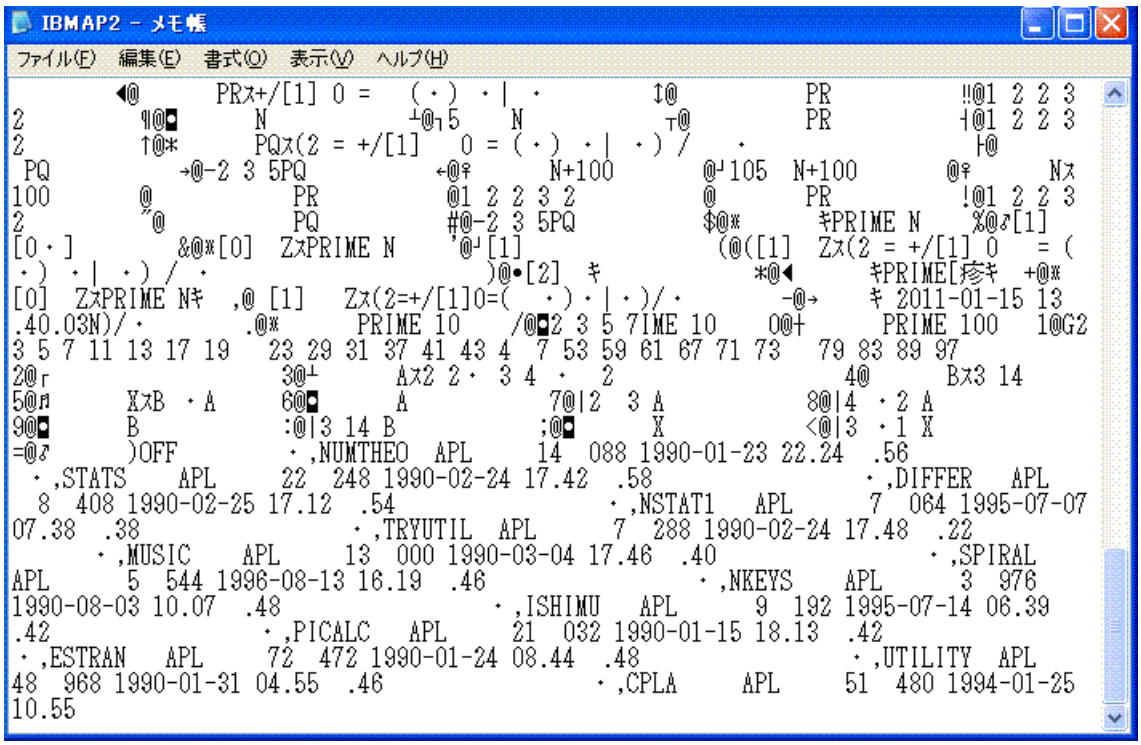
```
   aplses '¥APL_test¥ibmap2.txt'
==== Session 0: ======
)CLEAR
      。?     CLEAR WS
EAR
      「?      )LIB .APL
   」?,PROFILE  APL      6728 1988-12-20 12.00.00
                、?,START    APL      3224 1989-04-24 14.00.50
                ・?,RADIX
==== Session 1: ======
(途中省略)

   ~ ~ ~ ~
   ~ ~ ~ ~
(途中省略)
==== Session 4: ======
)CLEAR
      ・     CLEAR WS
EAR
      ▪?      Aｽ・2
```



□　　A

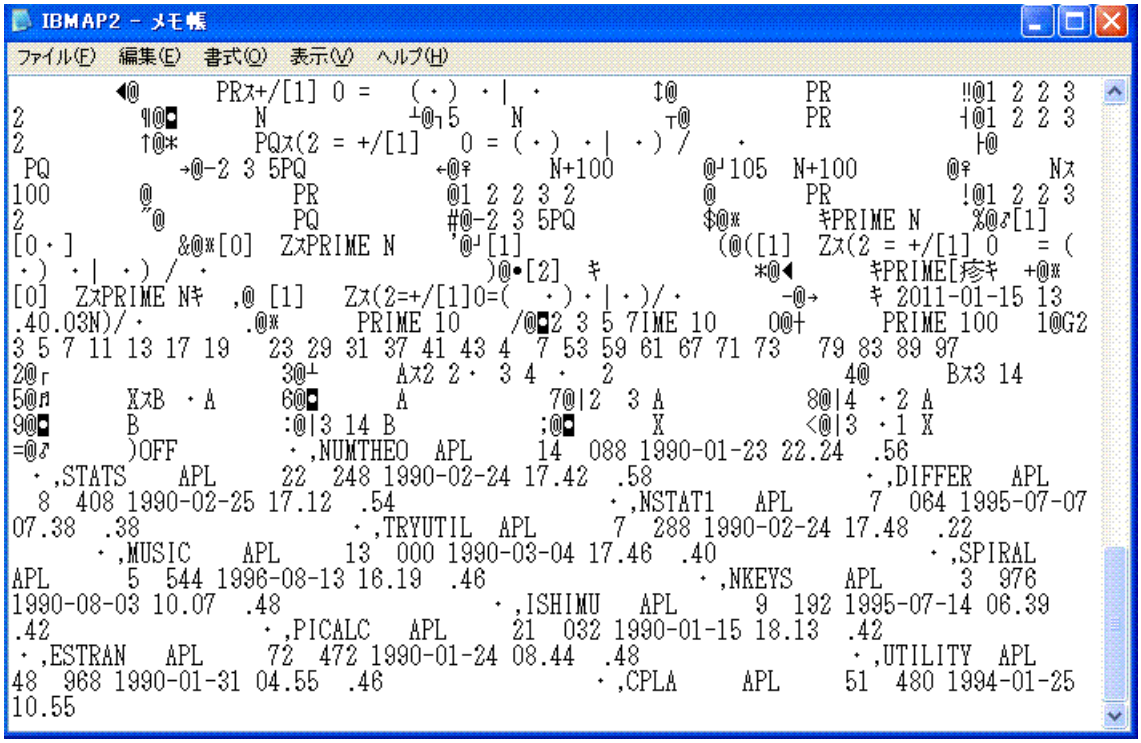
□1 2 3 4 5 6 7 8 9 10 11 12

@　　Pｽ3 4襤



@　　P

```
          @
1  2  3  4
        @
5  6  7
==== Session 5: ======
)CLEAR
      ○@     CLEAR WS
EAR
      @
      Nス5
         @       PRス+/[1] 0 = (・) ・| ・
        @          PR
          @
1 2 2 3 2
```




```
          μ@       N
             §@5
    N
          @       PR
==== Select Sesion #: ?
5
StartADR = 5899 EndADR = 1276

============ APL2 Session No.5======================
      )CLEAR
CLEAR WS
```

```
      N←5
      PR←+/[1] 0 = (⍳N) →.| ⍳N
      PR
1 2 2 3 2
      N
5

      PR
1 2 2 3 2
      PQ←(2 = +/[1] 0 = (⍳N) →.| ⍳N) / ⍳N
      PQ
2 3 5

      N+100
105
00
      N←100
      PR
1 2 2 3 2
     PR
1 2 2 3 2
      PQ
2 3 5

      ❍PRIME N
[1]   [0⋄0]
[0]   Z←PRIME N
[1]
[1]   Z←(2 = +/[1] 0 = (⍳N) →.| ⍳N) / ⍳N
[2]   ❍
      ❍PRIME[⋄]❍
[0]   Z←PRIME N

[1]   Z←(2=+/[1]0=(⍳N)→.|⍳N)/⍳N
    ❍ 2011-01-15 13.40.03
⍳N
      PRIME 10
2 3 5 7
 10
      PRIME 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

      A←2 2⍴2 3 4 ⋄2

 14
```

```
      X←B ▪ A
      A
2  3

4 ◆2

      B
3 14

      X
3 ◆1

      )OFF
```

```
NB. APL Character Code Conversion ======================================
NB. Left Arrow '←', Right Arrow
APL_CF =: ((189);(129 169)),: (184);(129 168)
NB. Iota 'ι',  Rho 'ρ'
APL_CF =: APL_CF, ((236);(131 199)),: ((230);(131 207))
NB. Domino 'ι',  Quad 'ρ'
APL_CF =: APL_CF, ((146);(240 64)),: ((144);(240 65))
NB. Overbar '◆', Del '❍'
APL_CF =: APL_CF, ((253);(240 68)),: ((183);(240 69))
NB. Jot '→'
APL_CF =: APL_CF, ((248);(240 70))

apl_fconv =: 3 : 0
:
CF =: x.
APLSes =. , y.
i =. 0
while. i < #CF
  do.
  'Code Font' =. i { CF
  Acode =. chr Code
  Afont =. chr Font
  Adr =. (Acode = APLSes) # (i. # APLSes)
  APLSes =. ; (<Afont)  Adr } <"(0) APLSes
NB.  wr > <;. 1 , APLSes
NB.  if. (1 = go_on_quit '') do. '*** quit ***' return. end.
  i =. i + 1
end.
APLSes
NB. > <;. 1 , APLSes
)

del_leadSP =: 3 : 0
(0 i.~ ' ' = y.) }. y.
)

del_trailCR =: 3 : 0
(1 i.~ CR = y.) {. y.
)

NB. Usage:
NB. aplses '¥APL_test¥ibmap1.txt'   => display all raw session
aplses =: 3 : 0
ASEN0 =: fread < y.
CLEADR =: (1 = ')CLEAR' E. ASEN0) # (i. #ASEN0)
```

```
i =. 0
while. i < #CLEADR do.
  wr '==== Session ',(": i), ': ======',CR ,200 {. (i{CLEADR) }. ASEN0
  i =. i + 1
end.
label_skip.

wr '==== Select Sesion #: ?'
ii =. ". rd 1

ASEN1 =: (ii{CLEADR) }. ASEN0
OFFADR =: (1 = ')OFF' E. ASEN1) # (i. #ASEN1)
wr 'StartADR = ', (": ii{CLEADR) , ' EndADR = ', (": {. OFFADR)
wr ' '
wr '============ APL2 Session No.', (": ii), '======================='
ASEN2 =: ( 4 + {. OFFADR) {. ASEN1
ASEN3 =: <;. 2 (ASEN2, CR)
ASEN4 =: del_leadSP L:0 ASEN3
ASEN5 =: del_trailCR L:0 ASEN4
ASEN6 =: ((i.#>ASEN5) -. 0, 2 ) { > ASEN5
ASEN7 =: '       )CLEAR', 3 }."(1) ASEN6
ASEN8 =: ASEN7 ,"(1) CR
ASEN9 =: APL_CF apl_fconv ASEN8
)
```

APLで素数を求めるプログラム

西川　利男著「基礎からのAPL－解説」サイエンスハウス(1991)．p.101-104

6.5　適用例―素数を求める問題

1からNまでの整数の素数（prime number）を求める問題を例として，いままでのAPLのテクニックを使って，実際のコーディングをどう進めるかを考えてみましょう．

まず，ある与えられた整数Nが素数かどうかは次の条件によって決まります．

・整数Nを1，2，3，…，Nまでの数で順々に割る

・そのうち1とNとだけで割り切れ，それ以外の数では割り切れない

以上の条件を満たせば素数ということになります．

問題は，このような素数を1～Nの中からすべてさがし出せということです．

APLでは，この処理をたった1行でやってしまいます．これを開発の順序に従って1ステップずつやってみましょう．

本章のはじめに述べた手順に従えば，問題は次のようにAPL流に読みかえることができます．

（1）　1からNまでの整数を生成してそれを要素とする配列をつくる．

（2）　その配列の要素のうちから，素数の条件に合うものを指定して，それをとり出す．

（3）　ここで素数の条件とは，次のように整理して述べられます．

（a）　1からNまでの整数で割って割り切れた……真

割り切れない……偽

（b）　上の割り切れた状態 ── 真が1とNとの場合の2回だけ起こる．

以上のときに素数になります．

コーディングは大きく次のように考えます．1，2，3，……，Nを要素とする配列⍳Nに対して，余りを求める関数｜を利用した割り切れたかどうかのフラグ（真，フラグ＝1，偽，フラグ＝0）によって素数を選び出します．つまり

（割り切れたかどうかのフラグ）／⍳N

これが0か2回となるものだけを素数としてえらび出します．

以下，順に１ステップずつ追っていきます．またＮ＝５の場合の途中経過もいっしょに示します．

（１） 整数ＭをＮで割った余りは関数 ｜ で求められます．

Ｎ｜Ｍ

（２） 整数Ｍを１～Ｎまでの整数で割った余りは，次のようになります．

（ι Ｎ）｜Ｍ

（３） さらに１～Ｍまでの整数を，１～Ｎまでの整数でそれぞれ割った余りを出すには，どうしたらよいでしょう．これは外積演算で行うことができます．また問題とするものは，１～Ｎまでの整数に対して行うことから

（ι Ｎ）∘．｜ι Ｎ

結果はＮ×Ｎの二次元配列になります．

なぜ （ι Ｎ）｜ι Ｎ ではいけないかは考えてみて下さい．

Ｎ←５

としたときの結果は次のようになります．

∘．｜	1	2	3	4	5
1	0	0	0	0	0
2	1	0	1	0	1
3	1	2	0	1	2
4	1	2	3	0	1
5	1	2	3	4	0

（４） ０＝（ι Ｎ）∘．｜ι Ｎ

これは，配列のそれぞれの要素が０に等しいかどうかテストします．

０に等しければ フラグ１（真）

０に等しくなければ フラグ０（偽）

になります．

０＝	1	2	3	4	5
1	1	1	1	1	1
2	0	1	0	1	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

（5）いま求めたフラグ1（真）が1とNの2回だけえられるNが素数です．これは，先の配列をまずタテ方向に，すなわち[1]軸で低減します．結果は一次元のベクトルになります．

```
      +/[1] 0=(ιN)∘.|ιN
1 2 2 3 2
```

（6）上のベクトルの要素が2に等しいかどうかのテストをして，フラグを得ます．

```
      2=  +/[1] 0=(ιN)∘.|ιN
0 1 1 0 1
```

（7）このフラグベクトルにより，もとの1～Nの整数から，フラグ1（真）のものを取り出します．つまり，素数の条件にあったものが取り出されます．

```
      (2=+/[1]0=(ιN)∘.|ιN)/ιN
2 3 5
```

以上の操作で1～Nまでの間の素数がとり出されます．

このようにたった1行のAPLプログラムにより，素数が得られます．これを，他のプログラム言語で書いた場合，何行ぐらいになるでしょうか．

しかし，ここで注意したいことがあります．APLプログラマは，えてしてコンパクトなプログラムを作りたがるのですが，あまり短くして，わかりにくくすることはさけなくてはなりません．あくまでも，これはここまで使えるというようにとどめるべきと思います．