会計記録の発動者としての勘定

藤田芳夫*

本稿は「千葉大学経済研究」vol.4,No.2 (1990年2月発行) に発表した拙稿「ターボ・パスカルによるポインター付き会計システム」では会計的意義の指摘が不十分であったので、その会計的特色を明らかにしようとするものである。

1 ポインター付き会計システムの特色

「ターボ・パスカルによるポインター付き会計システム」で展開した会計システムの特色は

1. 引合い取引データから取引として成立した引合いデータと取引として成立しなかった引合いデータを区別している点であり
2. この取引として成立した引合いデータと取引として成立しなかった引合いデータを区別するものが実は元帳の勘定であるという点である。
3. 引合いデータすなわち取引として成立するものと取引として成立しないものを含むいわば候補取引データを取引成立データと取引非成立データに分けるフィルターの役割をするのが元帳に含まれる勘定であるから、勘定は簿記会計の最も重要な要素である取引の認識にとって不可欠の機構であることになる。
4. このことは、取引が仕訳によって認識され、勘定はそれを転記するだけという消極的なものではなく、そもそも取引の認識それ自体が勘定によって規制されるという非常に積極的な役割を勘定に与えるのである。
5. この取引の成立、すなわち取引の認識において勘定が中心的役割を果たすのは、基本的に各勘定が示す現在高（借方残高、または貸方残高）であって各勘定の現在高に至る歴史的プロセスすなわち各勘定の歴史的記録内容ではないのである。
6. そこで、すべての会計組織は各勘定の現在高を示す現在高元帳（または金額元帳）と監査及び管理目的のため各勘定の歴史的記録内容を示す歴史的記録元帳を必要とすることになる。
7. ところで、歴史的記録元帳は、原始資料としての成立取引データから分類、転記されて作成される以外にはなく、その作成方法には様々な方法が可能になる。本システムでは、パスカルのポインターを使用して、仕訳帳に対する各勘定ヘッドからのリンク付きリストという形で実現した。
8. したがって、本会計システムでは、基本的会計ファイルは仕訳帳と現在高元帳（金額元帳）であり、歴史的記録元帳は必要に応じ、リンク付きリストによって構成するという形を取るのである。
9. 以上、ポインター付き会計システムの特色を指摘したが、本システムが含蓄している極めて重要な特色がもう一つある。それは、管理会計すなわち、経営意志決定のために会計システムを使用するためには、引

* 東京情報大学教授

1990年6月8日受理

合い取引のうち、正規の取引として成立し、会計的に取引として認識されたもの以外の非実現取引、すなわち、取引非成立データを正しく活用しなければならないことは当然であるが、この問題に対する解決への途を開くことが出来るということである。例えば、在庫不足によって、販売が実現しなかった場合について言えば、在庫管理のまずさを改善するためには、この「引合いデータ中の在庫不足による販売不能データ」の分析を欠くことができない。

不成立引合い取引に含まれている商品の特性や質、量、価格等の中に不成立に関する情報を探知することができるはずである。現在高元帳によって取引を認識するということの意味は、このように引合い取引を、成立取引と、非成立取引に分け、そこから生ずる可能性を認識することによって、管理会計の基盤が確立し、可能性が飛躍的に拡大するのである。

2　コンピューター会計における勘定体系の展開

上述したように、会計システムでは本来、現在高元帳がきわめて重要な位置を占めるのであるが、この現在高元帳の重要性を認識し、その重要性にふさわしいコンピューター会計システムの論理を、以下、展開しよう。

本会計システムでは、取引が各勘定に発生順に記録されて行くこと自体は本質的な事柄ではなく、各勘定について借方合計額と貸方合計額が計算されることが本質的な点であると考えるのである。

こう考えると、総勘定元帳はＮ個の勘定に対し、勘定の数の２倍、すなわち２Ｎ個の記録場所を用意すればよいことになる。

いま、決算のために必要な勘定である(閉鎖)残高勘定と損益勘定を除き、経常的に使用する勘定が、現金勘定、売掛金勘定、備品勘定、買

		第１法	第２法	第３法	第４法	第５法
現金勘定	借方	GENKIN1	A1	A11	A(1)	A(1, 1)
〃	貸方	GENKIN2	A2	A12	A(2)	A(1, 2)
売掛金勘定	借方	UKAKE1	A3	A21	A(3)	A(2, 1)
〃	貸方	UKAKE2	A4	A22	A(4)	A(2, 2)
備品勘定	借方	BIHIN1	A5	A31	A(5)	A(3, 1)
〃	貸方	BIHIN2	A6	A32	A(6)	A(3, 2)
買掛金勘定	借方	KKAKE1	A7	A41	A(7)	A(4, 1)
〃	貸方	KKAKE2	A8	A42	A(8)	A(4, 2)
資本金勘定	借方	CTAL1	A9	A51	A(9)	A(5, 1)
〃	貸方	CTAL2	A10	A52	A(10)	A(5, 2)
売上勘定	借方	URI1	A11	A61	A(11)	A(6, 1)
〃	貸方	URI2	A12	A62	A(12)	A(6, 2)
仕入勘定	借方	SHIRE1	A13	A71	A(13)	A(7, 1)
〃	貸方	SHIRE2	A14	A72	A(14)	A(7, 2)
給料勘定	借方	KYURY1	A15	A81	A(15)	A(8, 1)
〃	貸方	KYURY2	A16	A82	A(16)	A(8, 2)
営業費勘定	借方	EIGYO1	A17	A91	A(17)	A(9, 1)
〃	貸方	EIGYO2	A18	A92	A(18)	A(9, 2)

（表１―１）勘定体系展開の諸方法

掛金勘定、資本金勘定、売上勘定、仕入勘定、給料勘定、営業費勘定の九個の勘定であるとする。

勘定体系を主記憶装置の中に展開する方法には（表1－1）に示すように様々な方法がある。しかし、同表の第五列に示した方法、すなわち二次元の配列を採用することにする。

この二次元の配列A(9.2)を用いて勘定体系を展開すると言う事は、(表1－2)に示すように主記憶装置のなかに9行×2列の場所をとり、この場所全体を仮にA(account matrixの先頭字Aをとった）と名付け、各勘定を縦軸（行）にそって配列し、各勘定の借方と貸方を横軸(列)にそって定義することである。

複式簿記の勘定はそれ自体が計算単位であるが、パチオロ以来、複式簿記はこの一つの計算単位である勘定をプラスとマイナス、ないし積極と消極という二元的構造を持つものとして取り扱ってきた。

取引の結果たえず変化する各勘定の純残高は、必要があれば直ちに算出されなければならないのである。何故なら、借方残高として現れる現金勘定の残高は、現金の支払可能額を明示することが出来るし、買掛金の支払必要額は買掛金勘定の貸方残高を計算することによって確認することが出来るからである。

したがって、各勘定は決算によって内容が更新されるまで、増加するだけで減少しない二つの変数によって構成され、これによって、必要な全体的把握と、各時点での意志決定に必要な直接的把握を可能にしたのである。

上記の二次元配列A（9、2）を用いる方法は、実はこの伝統的方法をそのまま反映しているのであって、各勘定の借方はA(i, 1）で示され、貸方はA(i, 2）で示される。

列数 行数	借方 1	貸方 2
現金勘定 1	A（1, 1）	A（1, 2）
売掛金勘定2	A（2, 1）	A（2, 2）
備品勘定 3	A（3, 1）	A（3, 2）
買掛金勘定4	A（4, 1）	A（4, 2）
資本金勘定5	A（5, 1）	A（5, 2）
売上勘定 6	A（6, 1）	A（6, 2）
仕入勘定 7	A（7, 1）	A（7, 2）
給料勘定 8	A（8, 1）	A（8, 2）
営業費勘定9	A（9, 1）	A（9, 2）

（表1－2）二次元配列A（9、2）による勘定体系の展開

全体名称　ACCOUNT-SYSTEM

	借方	貸方	
現　金＝ACT(1)	A(1, 1)	A(1, 2)	act(genkin)
売掛金＝ACT(2)	A(2, 1)	A(2, 2)	act(urikake)
備　品＝ACT(3)	A(3, 1)	A(3, 2)	act(bihin)
買掛金＝ACT(4)	A(4, 1)	A(4, 2)	act(kaikake)
資本金＝ACT(5)	A(5, 1)	A(5, 2)	act(shihon)
売　上＝ACT(6)	A(6, 1)	A(6, 2)	act(uriage)
仕　入＝ACT(7)	A(7, 1)	A(7, 2)	act(shiire)
給　料＝ACT(8)	A(8, 1)	A(8, 2)	act(kyuryo)
営業員＝ACT(9)	A(9, 1)	A(9, 2)	act(eigyohi)

（表１－３）PASCAL による勘定体系の展開

パスカルでは、ここで使用した現金勘定、売掛金勘定、備品勘定等の九つの勘定を１から９までの整数で表すこともちろん可能であるが、各勘定のローマ字表現をそのまま使用することが出来る。すなわち、下に示すように、各勘定について定義したローマ字名を宣言スカラー形としてタイプ宣言をしておくのである。

現金勘定……genkin
売掛金勘定……urikake
備品勘定……bihin
買掛金勘定……kaikake
資本金勘定……shihon
売上勘定……uriage
仕入勘定……shiire
給料勘定……kyuryo
営業費勘定……eigyohi

```
type acts= (genkin, urikake, bihin, kaikake,
           shihon, uriage, shiire, kyuryo,
           eigyohi) ;
```

また、各勘定は借方金額と貸方金額で構成されるから、

```
act=array [1.. 2] of real ;
```

または

```
DrCr= (Dr, Cr) ;
      act=array [DrCr] of real ;
```

としてタイプ宣言をしておく。

こうしておくと、前述した二次元の勘定配列Aは変数宣言部で次のように宣言すればよい。

```
var a : array [acts] of act ;
  nact : act ;
```

すなわち、acts は genkin‥‥eigyohi であるから、現金勘定は借方、貸方を含めて、a[genkin] で示すことができ、その借方は a[genkin, 1] または a [genkin, Dr] であり、貸方は a [genkin, 2] または [a [genkin, Cr] である。

したがって、現金勘定から、営業費勘定までの、各勘定の借方と貸方をゼロにする、初期化の論理は次のようになる。

```
for nact:=genkin to eigyohi do
  for j : =1 to 2 do
    a [nact, j] : =0.0 ;
```

なお、この関係を図示すると、次の（表１－４）のようになる。

$$
a=\begin{bmatrix}
a\ [genkin] & =[acts\ [0]] & =[a\ [genkin,\ 1],\ a\ [genkin,\ 2]] \\
a\ [urikake] & =[acts\ [1]] & =[a\ [urikake,\ 1],\ a\ [urikake,\ 2]] \\
a\ [bihin] & =[acts\ [2]] & =[a\ [bihin,\ 1],\ a\ [bihin,\ 2]] \\
a\ [kaikake] & =[acts\ [3]] & =[a\ [kaikake,\ 1],\ a\ [kaikake,\ 2]] \\
a\ [shihon] & =[acts\ [4]] & =[a\ [shihon,\ 1],\ a\ [shihon,\ 2]] \\
a\ [uriage] & =[acts\ [5]] & =[a\ [u\ (iage,\ 1],\ a\ [uriage,\ 2]] \\
a\ [shiire] & =[acts\ [6]] & =[a\ [shiire,\ 1],\ a\ [shiire,\ 2]] \\
a\ [kyuryo] & =[acts\ [7]] & =[a\ [kyuryo,\ 1],\ a\ [kyuryo,\ 2]] \\
a\ [eigyohi] & =[acts\ [8]] & =[a\ [5igyohi,\ 1],\ a\ [eigyohi,\ 2]]
\end{bmatrix}
$$

（表１－４）

ところで、宣言スカラー型の変数 nact を、genkin から eigyohi に変化させても、nact が genkin のとき、'現金'とか、'genkin'と出力することはできない。そのためには

ANAME : array [acts] of string [10] を変数宣言しておき、実行文で ANAME [genkin] : ='現金'というように文字列データを与えておかねばならない。すなわち、

```
ANAME [genkin]  : ='現　金';
ANAME [urikake] : ='売掛金';
ANAME [bihin]   : ='備　品';
ANAME [kaikake] : ='買掛金';
ANAME [shihon]  : ='資本金';
ANAME [uriage]  : ='売　上';
ANAME [shiire]  : ='仕　入';
ANAME [kyuryo]  : ='給　料';
ANAME [eigyohi] : ='営業費';
```

と与えておくのである。

3 仕訳三変数の抽出、勘定行列と取引行列の加法としての元帳転記

勘定体系が設定できると、取引を仕訳し、各勘定に転記し、一定期間後に試算表を作成しなければならない。上述した方法で総勘定元帳を主記憶装置の内部に展開した後、これらの点はどのようになるであろうか。まず、伝統的な手記複式簿記の方法により、また、全ての取引が単純取引のみからなるもっとも単純な場合を用いて明らかにしよう。

いま、下記の10個の取引がある。

1．現金135,000円で営業を開始した。
2．商品80,000円を掛仕入した。
3．商品45,000円を掛売りした。(但し売価)
4．商品40,000円を現金で仕入れた。
5．商品50,000円を現金売りした。(但し売価)
6．売掛金39,000円を回収した。
7．買掛金58,000円を現金で支払った。
8．備品28,000円を購入した。
9．給料5,000円を支払った。
10．営業費3,000円を支払った。

この例題を伝統的手法で処理すると次の（表１－５）の通りである。

仕訳

1）	現　金	135,000	資本金	135,000
2）	仕　入	80,000	買掛金	80,000
3）	売掛金	45,000	売　上	45,000
4）	仕　入	40,000	現　金	40,000
5）	現　金	50,000	売　上	50,000
6）	現　金	39,000	売掛金	39,000
7）	買掛金	58,000	現　金	58,000
8）	備　名	28,000	現　金	28,000
9）	給　料	5,000	現　金	5,000
10）	営業費	3,000	現　金	3,000

現　金(1)

1)	135,000	4)	40,000
5)	50,000	7)	58,000
6)	39,000	8)	28,000
		9)	5,000
		10)	3,000

売掛金(2)

3)	45,000	6)	39,000

備　品(3)

8)	28,000		

買掛金(4)

7)	58,000	2)	80,000

資本金(5)

		1)	135,000

売　上(6)

		3)	45,000
		5)	50,000

仕　入(7)

2)	80,000		
4)	40,000		

給　料(8)

9)	5,000		

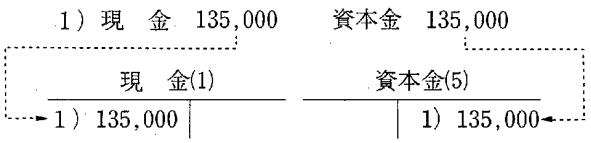
営業費(9)

10)	3,000		

試算表

借　方	勘定科目	貸方
224,000	現　金	134,000
45,000	売掛金	39,000
28,000	備　品	
58,000	買掛金	80,000
	資本金	135,000
	売　上	95,000
120,000	仕　入	
5,000	給　料	
3,000	営業費	
483,000	合　計	483,000

ここで、仕訳を元帳に転記するとはどういう操作であるか反省してみよう。伝統的方法の場合、たとえば第一の取引についてみれば、下のように現金勘定の借方と資本金勘定の貸方に135,000と言う金額が記入されるが



この際、現金勘定と資本勘定と言う仕訳に関係する一対の勘定が、総勘定元帳を構成している全勘定と分離した形で把握されやすい。

しかし、周知のように、貸借対照表等式はいかなる時点でも成立するものであるから、取引が勘定体系にひきおこす変化は、決して一つの仕訳に関係する一対の勘定にとどまるものではなく、逆に一つの取引が発生する前の段階の勘定体系 A_tから、その取引による変化を受けた勘定体系 A_{t+1}へと勘定体系の全体が変化すると考えるのが最も正確な認識である。

上述した勘定体系A（N, 2）またはA（9, 2）について言えば、開設されただけで金額記入のない勘定体系 A_0から取引 T_1によって A_1が出来るのである。

	A_0			T_1			A_1	
	Dr	Cr		Dr	Cr		Dr	Cr
現金	0	0		135,000	0		135,000	0
売掛金	0	0		0	0		0	0
備品	0	0		0	0		0	0
買掛金	0	0		0	0		0	0
資本	0	0	+	0	135,000	=	0	135,000
売上	0	0		0	0		0	0
仕入	0	0		0	0		0	0
給料	0	0		0	0		0	0
営業費	0	0		0	0		0	0

すなわち、勘定体系全体としての把握をする場合、上図のように仕訳は仕訳行列として把握され、9行×2列の勘定行列 A_0に9行2列の仕訳行列 T_1が加算される結果、状態が更新された9行×2列の勘定行列 A_1が出現するのである。

例題について言えば、10個の取引、したがって10個の仕訳は、それぞれ9行2列の取引行列として下のように表現される。

$$T_1=\begin{bmatrix}135,000 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 135,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad T_2=\begin{bmatrix}0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 80,000\\ 0 & 0\\ 0 & 0\\ 80,000 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad T_3=\begin{bmatrix}0 & 0\\ 45,000 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 45,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad T_4=\begin{bmatrix}0 & 40,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 40,000 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}$$

$$T_5=\begin{bmatrix}50,000 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 50,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad T_6=\begin{bmatrix}39,000 & 0\\ 0 & 39,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad T_7=\begin{bmatrix}0 & 58,000\\ 0 & 0\\ 0 & 0\\ 58,000 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}\quad T_8=\begin{bmatrix}0 & 28,000\\ 0 & 0\\ 28,000 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\end{bmatrix}$$

$$T_9=\begin{bmatrix}0 & 5,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 5,000 & 0\\ 0 & 0\end{bmatrix}\quad T_{10}=\begin{bmatrix}0 & 3,000\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 0\\ 0 & 00\\ 3,000 & 0\end{bmatrix}$$

そして、これら10個の取引（仕訳）が転記された結果としての勘定体系 A_{10}は

$A_0+T_1+T_2+T_3+T_4+T_6+T_7+T_8+T_9+T_{10}=A_{10}$

$$A_{10}=\begin{bmatrix} 224,000 & 134,000 \\ 45,0000 & 39,000 \\ 28,000 & 0 \\ 58,000 & 80,000 \\ 0 & 135,000 \\ 0 & 95,000 \\ 120,000 & 0 \\ 5,000 & 0 \\ 3,000 & 0 \end{bmatrix}$$

としてえられるのである。

以上の論理をパスカルで実行してみよう。

まず、A_0, A_1およびT_1を変数宣言しておかねばならない。

```
var A0, A1, T1 : array [acts] of act ;
```

次に、取引を一切転記していない、初期化しただけの金額元帳 A_0を作る。

```
for nact :=genkin to eigyohi do
  for j : = 1  to 2 do
    a0 [nact, j] : =0.0 ;
```

ここまでは、前述した論理と同じであるが、A_1, T_1を初期化するのにパスカルでは、

```
A1 : =A0 ;
T1 : =A0 ;
```

が可能である点に注意しなければならない。

現金135,000を元入れして開業したという取引行列は、

```
T1 [genkin,  1 ] : =135000.0 ;
T1 [genkin,  2 ] : =135000.0 ;
```

で作成できる。

この取引行列 T_1を書き出してみれば、右段上図のようになる。

T_1の内容

現　金	135,000	0
売掛金	0	0
備　品	0	0
買掛金	0	0
資本金	0	135,000
売　上	0	0
仕　入	0	0
給　料	0	0
営業費	0	0

次に、取引が全く記入されていない金額元帳 A_0に取引行列 T_1を加えると、第一の仕訳が転記された金額元帳 A_1が作成される。これが

$A_1 : =A_0+T_1$

で実行できればよいわけであるが、行列の加減算自体は出来ない。そこで、行列の個々の要素を加えて行くのである。

```
for nact : =genkin to eigyouhi do
  for j : =1 to 2 do
    A1 [nact, j] : =A1 [nact, j]+T1 [nact, j] ;
```

そして、最後に、金額元帳 A_0に取引行列 T_1を加えて、結果としての金額元帳 A_1を出力するのである。

Running

金額元帳	A1→	
現　金	135,000	0
売掛金	0	0
備　品	0	0
買掛金	0	0
資本金	0	135,000
売　上	0	0
仕　入	0	0
給　料	0	0
営業費	0	0

（プログラム a_sys2. PAS は省略した）

ところで、上述の方法は論理的には誠に正確であり、行列の加法として数学的取扱いが完全に出来る。しかし、実務的にみると極めて非能率である。むしろ伝統的な関係する一対の二勘定だけを取り出し、それに転記するという方法の方がはるかにすぐれている。

実は、↗

↗　現　金　135,000　資本金　135,000

と言う仕訳は取引行列の金額ゼロの部分を圧縮し、有意味な借方金額と貸方金額だけを残したものと考えることが出来る。その際、金額だけではどの勘定に転記されるべきかが不明になるため、摘要欄に現金とか資本金という勘定名を明示する。

$$\begin{bmatrix} 135,000 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 135,000 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow 1,\ 135,000,\ 5,\ 135,000$$

したがって勘定名の代わりに勘定コードでもよいし、さらにその勘定が置かれている行数で↗

↗もよいことになる。

いま、勘定名の代わるに勘定行列A(9, 2)内部でその勘定が置かれている行数を用いるとすれば、例題の10個の仕訳を次のように変形しても論理的に矛盾を犯すことにはならない。

すなわち、

図7の第一変形は勘定名を勘定配列内の各勘定の行位置に変形したのであるが、単純取引では借方金額と貸方金額は一致する。同じものを二度書く必要はないからこれを省略すると、下掲の第二変形のようになる。

第1変形

1)	1,	135,000,	5,	135,000
2)	7,	80,000,	4,	80,000
3)	2,	45,000,	6,	45,000
4)	7,	40,000,	1,	40,000
5)	1,	50,000,	6,	50,000
6)	1,	39,000,	2,	39,000
7)	4,	58,000,	1,	58,000
8)	3,	28,000,	1,	28,000
9)	8,	5,000,	1,	5,000
10)	9,	3,000,	1,	3,000

第2変形

1)	1,	135,000,	5
2)	7,	80,000,	4
3)	2,	45,000,	6
4)	7,	40,000,	1
5)	1,	50,000,	6
6)	1,	39,000,	2
7)	4,	39,000,	2
8)	3,	28,000,	1
9)	8,	5,000,	1
10)	9,	3,000,	1

→

第3変形

1)	135,000,	1,	5
2)	80,000,	7,	4
3)	45,000,	2,	6
4)	40,000,	7,	1
5)	50,000,	1,	6
6)	39,000,	1,	2
7)	58,000,	4,	1
8)	28,000,	3,	1
9)	5,000,	8,	1
10)	3,000,	9,	1

第2変形の左にある勘定行数（又はコード）を右に移動すると上に示した第3変形が出来上がる。以下、この第3変形の形式を用いて説明することにする。何故なら、下に示すように、↗取引行列をスカラーと行列との積と考え、さらに行列を単純化して金額存在の位置を示す要素を表示することになるからである。

取引行列　　スカラー×金額位置行列

$$\begin{bmatrix} 135,000 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 13,500 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} = 135,000 \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

スカラー×（金額位置簡易表現法）

→135,000, 1, 5

以上述べたように、伝統的仕訳形式

（借方）　金額　（貸方）　金額

XX勘定　YYY　XX勘定　YYY

という表現形式は取引行列の簡略表現法として極めて合理的かつ能率的なものであるが、仕訳に関係ある一対の勘定だけを取り出して転記するという伝統的転記方法もまた合理的、能率的な方法である。↗

↗何故なら、下に示すように、仕訳に関係ある一対の勘定だけを取り出して転記するという場合、実は転記されない勘定は暗黙のうちにゼロを加算されていると考えればよいからである。

たとえば、取引2を元帳に転記するということは、取引1を転記した結果出来ている勘定行列 A_1 に取引行列 T_2 を加えることであるが、伝統的な転記手続きの背後には A_1 に含まれている仕入勘定と買掛勘定だけを明示的に取り出し、他のゼロを加える計算は明示的には行わないということである。

2）仕　入　8,000　　買掛金　80,000

仕入		買掛金	
2）80,000			2）80,000

A_1 ＋ T_2 ＝ A_2

	A_1		T_2		A_2	
	135,000	0	0	0	135,000	0
	0	0	0	0	0	0
	0	0	0	0	0	0
買掛金	0	0	0	80,000	0	80,000
	0	135,000	0	0	0	135,000
	0	0	0	0	0	0
仕　入	0	0	80,000	0	80,000	0
	0	0	0	0	0	0
	0	0	0	0	0	0

このように考えると、勘定体系を二次元の行列に展開している場合、元帳への転記ということは論理的には勘定行列と取引行列の加算であるが、実行上は、ゼロ加算を省略して、取引により影響される勘定の、しかも借方と貸方のうち加算される側だけを取り出して取引金額を加えればよい。

そこで、元帳への転記は仕訳1の場合、

借方勘定（＝現金）への転記は

A（1，1）＝A（1，1）＋135,000

貸方勘定（＝資本金）への転記は

A（5，2）＝A（5，2）＋135,000であり、

仕訳2の場合、

借方勘定（＝仕入）への転記は↗

↗A（7，1）＝A（7，1）＋80,000

貸方勘定（＝買掛金）への転記は

A（4，2）＝A（4，2）＋80,000

である。

したがって、10個の例題取引について、上記のような二つ一組の命令を実行すれば、例題の取引をすべて元帳に転記することが出来る。

しかし、このようにすると、取引が100個あれば100組、1,000個あれば1,000組の命令をコーディングしなければならないから、電子計算機会計としては無意味なものになる。そこで次のように考えるのである。

いま、仕訳1と2の転記を検討してみると

仕訳1の転記	A（	1,	1	）＝A（	1,	1	）＋	135,000
	A（	5,	2	）＝A（	5,	2	）＋	135,000
仕訳2の転記	A（	7,	1	）＝A（	7,	1	）＋	80,000
	A（	4,	2	）＝A（	4,	2	）＋	80,000
		↑勘定位置	↑借方・貸方		↑勘定位置	↑借方・貸方		↑取引金額 TA

借方勘定への転記は、第1添字が借方勘定の勘定行列における行位置を示し、第二添字が借方を示す、すなわち、借方勘定の転記は各勘定↗の第1列に定まっており、それに取引金額TA（Transaction Amount）を加える。

A（借方勘定の位置，1）＝A（借方勘定の位置，1）＋ 取引金額

⋮ 借方　　⋮ 借方

貸方勘定への転記も同様に配列要素aの第一添え字が貸方勘定の勘定行列における行位置↗↗を示し、第二添字は貸方、すなわち第2列を示す定数2に定まっている。

A（貸方勘定の位置，2）＝A（貸方勘定の位置，2）＋ 取引金額

⋮ 貸方　　⋮ 貸方

したがって、一つの取引を勘定行列に転記するには三つの変数 TA (取引金額)、借方勘定の位置 ID、貸方勘定の位置 IC が判明すれば、仕訳の数に無関係に、すべての仕訳を元帳に転記できる。

パスカルでこれを実行すると、金額元帳配列 (A : array [acts] of act) に対して、次のプログラムで転記を行うことができる。

```
write ('仕訳入力……Y/N') ; readln (yes) ;
while yes in ['Y', 'y'] do
  begin
    write ('TA →') ; readln (TA) ;
    write ('ID →') ; readln (ID) ;
    write ('IC →') ; readln (IC) ;
    A [nact (ID-1), 1] :=A [nact(ID-1), 1]
                                    +TA ;
    A [nact (IC-1), 2] :=A [nact(IC-1), 2]
                                    +TA ;
    write('仕訳入力…Y/N');readln(yes);
  end ;
```

すなわち、91ページで使用した勘定コードを引き続き使用するものとすると、取引金額 TA、借方勘定コード ID、貸方勘定コード IC を入力し、それを nact (ID-1), nact (IC-1) で、列挙型勘定変数 (genkin.. eigyohi) に換算し、金額元帳配列のしかるべき勘定に転記するのである。

このパスカルプログラム全体を示すと、次の a_sys3. PAS のようになる。

```
program a_sys3; {89,  8,  25}
  type acts=(genkin, urikake, bihin, kaika-
             ke, shihon, uriage, shiire, kyur-
             yo, eigyohi) ;
       act= array [1.. 2] of real;
var A      : array [acts] of act ;
ANAME  : array [acts] of string [10] ;
    j      : integer ;
  nact     : acts ;
  yes      : char ;
  TA       : real ;
 ID, IC    : integer ;
begin
ANAME [genkin]   : ='現金' ;
ANAME [urikake]  : ='売掛金' ;
ANAME [bihin]    : ='備品' ;
ANAME [kaikake]  : ='買掛金' ;
ANAME [shihon]   : ='資本' ;
ANAME [uriage]   : ='売上' ;
ANAME [shiire]   : ='仕入' ;
ANAME [eigyohi]  : ='営業費' ;
ANAME [kyuryo]   : ='給料' ;

for nact : =genkin to eigyohi do
  for j ; =1 to 2 do
    A [nact, j] : =0.0 ;
write ('仕訳入力……Y/N') ; readln (yes) ;
while yes in ['Y', 'y'] do
  begin
    write ('TA →')  ; readln (TA) ;
    write ('ID →')  ; readln (ID) ;
    write ('IC →')  ; readln (IC) ;
    A [acts (ID-1), 1] :=A[act(ID-1), 1]
                                    +TA ;
    A [acts (IC-1), 2]:=A[acts(IC-1), 2]
                                    +TA ;
    write ('仕訳入力…Y/N') ;readln (yes) ;
  end ;
writeln ('……金額元帳 A……') ;
for nact : =genkin to eigyohi do
  begin
    write (ANAME [nact]) ;
    for j : =1 to 2 do
      write (A [nact, j] : 7 : 0) ;
    writeln ;
  end ;
end.
```

4　引合いデータから成立取引を抽出する論理

(ポインター付き会計システムの主システム PBKFS. PAS メイン・プログラムの説明)

以上、ポインター付会計システムの特色であ

る現在高元帳を成立させるために必要な取引高(TA)、借方勘定コード(Id)、貸方勘定コード(Ic)という仕訳三変数による現在高元帳への転記の論理を明らかにしたので、以下引き合いデータから取引として成立するものを抽出する、即ち、現在高元帳によって取引を認識する論理を具体的に説明しよう。

本パスカル会計システムは

1．INTpBKFS. PAS
2．pBKFS. PAS
3．dpHact. PAS
4．dpHcust. PAS
5．dpHvend. PAS
6．dpHgd. PAS

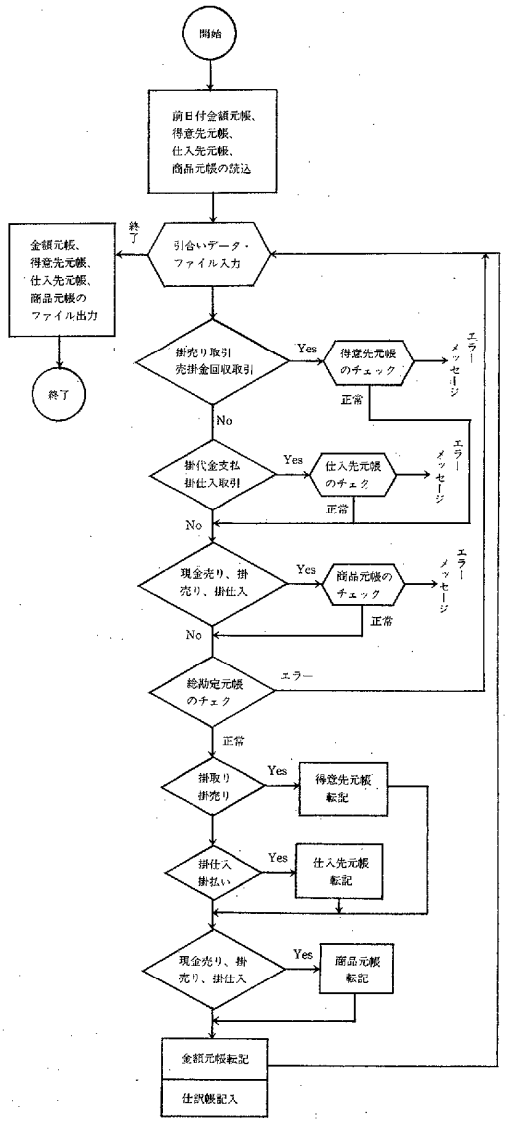
の6本のプログラムから構成されている。1．INTpBKFS. PASはこのパスカル会計システムの諸ファイルを初期化するプログラムである。2．pBKFS. PASは毎日の取引を処理するプログラムであり、いわば、この会計システムの中心をなすプログラムである。3から6までのプログラムは総勘定元帳、得意先元帳、仕入先元帳、商品元帳といった諸元帳がすべて現在高元帳（金額元帳）として作成されるので、各勘定の歴史的記録内容を検索表示するためのプログラムである。

このpBKFSプログラムの内容は(図14)のフローチャートが示しているように一つの取引(より正確に言えば候補取引または引合取引)があると、それを諸元帳でチェックし、各勘定の持っている自然的条件や、企業が設定した基準に照らして異常がなければ正常な取引として元帳に転記し、仕訳帳に記録するのである。

ここで重要なことは、まず仕訳があって、元帳に転記されるのではなく、候補取引（引合取引）が元帳でチェックされ、元帳の該当勘定が示す条件に合致すれば元帳に記録（加算）されて取引として成立するのである（すなわち取引として認識される)。

さらに、元帳に記録（加算）されることによって、次の引合い取引の判定に自動的に影響を与え、元帳の機能を維持し続けるのである。これが元帳と元帳記入の基本的職能である。仕訳帳に記入するということは、このようにして成立した取引の歴史的記録を保持するためである。

次に、各元帳についてどのようにチェックす



(図14) pBKFS. PAS
メインプログラムのフロチャート

るかを明らかにしよう。

まず、総勘定元帳のチェック・プログラムである手続 checkamtlgr (sy：sw) の構造をみよう。

本システムでは、勘定は、タイプ宣言で

acts＝ (noact, genkin, urikake, bihin, kaikake, shihon, uriage, shiire, kyuryo, eigyohi)；

とし、仕訳の基本構造は、取引日付 date, 取引ナンバー n, 取引金額 ta, 借方勘定 id, 貸方勘定 ic で構成し、

```
sw＝record
      date：ymd；
      n：integer；
      ta：real；
      id, ic：integer；
    end；
```

取引種類を現金売上 (genkinuri)，掛売り (kakeuri)，掛仕入れ (kakesiire)，売掛金回収 (kaketori)，買掛金支払 (kakebarai)，其他取引 (sonota) とし、下のようにタイプ宣言する。

```
k＝ (kakeuri, genkinuri, kakesiire,
     kaketori, kakebarai, sonota)；
```

この取引種類に応じて、取引の記述を商品名、単価、数量、得意先名等必要な項目を表示しうるようにしてある。

```
trrec＝record
       tsw：sw；
       case xk：k of
         kakeuri： (Gd：char；tanka,
                    qty：real；cust：
                    str10)；
         省略：
         sonota： ( )；
       end；
```

そこで、総勘定元帳の検査では勘定のもつ自然の検査機能すなわち、資産勘定、費用勘定では借方合計額は常に貸方合計額より大であるから、

```
if (acts (Id) in [genkin.. bihin, shiire.. eigyo
    hi]) and ((dsum＋ta)＞csum)
```

であれば正常であるが、借方勘定 Id が負債、資本、収益であればエラーになる。

一方、負債勘定、収益勘定では、貸方合計額の方が大きいから、

```
if (acts (Id) in [kaikake.. uriage]) and
    ((dsum＋ta)＞csum)
  then エラー指示
```

という論理を使用するのである。

もちろんこうした簿記的な自然条件のほかに、たとえば、売掛金の回収であれば、特定日までの売掛金の回収目的を保持させておけば、それによって売掛債権を、また、売上勘定に特定日までの目標売上高を保持させておけば、売上目標と売上実績との差額を認識できるし、さらに非実現売上額との比較が可能であるというように、企業が積極的に導入する経営的指標又は基準を導入することは自由であるが、コンピュータ簿記会計の原則を論ずる本システムの範囲を越えるので、ここでは採用していない。

同様に、貸方勘定の転記に当たっては、

```
if (acts (Ic) in [genkin.. bihin, shiire.. eigyo-
    hi]) and ((csum＋ta)＞dsum)
  then エラー処理
```

ということになる。(たとえば、売上勘定に特定日までの目標売上高を保持させておけば、目標と実績との差額を認識できるし、さらに非実現売上額との比較も可能になる。)

得意先元帳の記入を必要とする取引種類、即ち、掛売り、売掛金の回収の場合には、まず、得意先名が得意先元帳の中に登録済みでなければならないし、登録済みであれば、その得意先に許される与信範囲を越えて掛売をすることは不正であり、また売掛金残高以上に回収することも不正である。本編では与信範囲を越える掛売かどうかの検査は可能性を示すに止め、プログラムしていない。

```
case xk of
  {xkakeuri：与信範囲超過の検査その他}
  xkaketori：if (csum＋ta)＞dsum
                then エラー処理
end；
```

仕入先元帳の記入を必要とする取引即ち、掛仕入と買掛金の支払についても掛仕入先が仕入先元帳に登録済みでなければならないし、登録済みであれば、掛仕入の場合には各仕入先に対する掛債務限度額のチェックを行うべきであるし（ただし、本プログラムでは実行していない）、買掛金の支払であれば支払額が債務額を越えないようにチェックするのは当然である。

```
case xk of
    {xkakesiire：掛債務限度のチェックその他}
    xkakebarai：if (dsum+ta)>csum
    then エラー処理
end；
```

商品元帳では、現金売りであれ、掛売りであれ、受注による引渡し数量だけ商品が存在しなければならないことは言うまでもない。それだけでなく、引渡し後の商品残量が再発注点に至っているかどうかをチェックし、必要に応じて発注処理をするのが当然である。(本プログラムでは再発注処理は省略してある。)

```
case xk of
xgenkinuri, xkakeuri：if (Bqty-XQTY)<0
then エラー処理；
    {if (Bqty-XQTY)<再発注点 then 再発注処理}
```

こうしてすべての元帳について各勘定の現在高によるチェックをパスした引合い取引について、取引が成立する、すなわち、認識され、各元帳勘定は更新され、その証拠として仕訳帳に取引が記録されるのである。

5 リンク付きリストによる歴史的記録の検索

上述したように、コンピューター会計の出発点として、現在高元帳の重要性を指摘し、特に現在高元帳としての総勘定元帳をそのように構想し、それにもとづいてYMD, Tn, TA, ID, ICという仕訳変数と、

```
A (ID, 1)：=A (ID, 1)+TA
A (IC, 2)：=A (IC, 2)+TA
```

という元帳転記方式を展開した。

しかし、複式簿記にもとづく会計システムが取引の歴史的記録を欠いてもよいことにはならない。すなわち、仕訳帳に記録すると共に、各勘定の借方合計金額、貸方合計金額が判明するだけでは不十分で、それら合計額がどのようにしてもたらされたものかを明らかにすることが出来なければならない。

この勘定ごとに、その歴史的記録を保存する方法は、各勘定ごとにファイルを1個宛割り当て、金額元帳勘定に転記して借方または貸方合計を更新すると共に、借方勘定ファイルと貸方勘定ファイルに追記しておくのが一番単純な方法である。

元帳の各勘定にファイルを1個宛割り当てる場合は、フォートランでは、

```
A (ID, 1)=A (ID, 1)+TA
A (IC, 2)=A (IC, 2)+TA
  IFILE=ID+15
WRITE (IFILE, TA, ID, TC)
  IFILE=IC+15
WRITE (IFILE, A, ID, IC)
```

というようにファイル変数と借方コード、貸方コードとの間に関数関係を成立させ、それによって極めて簡潔に実行することができる。

パスカルではこうした方法は実行できない。違うファイルへアクセスしようとするたびにassign, reset, closeをしなければならない。

ファイルへ記入するレコードは仕訳であるから、

```
type x=record
        n：integer；   …n‥取引ナンバー
        ta：real；     …ta‥取引金額
        id, ic：integer…id‥借方勘定コード
      end；            ic‥貸方勘定コード
```

という形になり、各勘定の歴史的記録を保存するファイル名は

genkin. dat ……現金勘定ファイル
urikake, dat……売掛金勘定ファイル

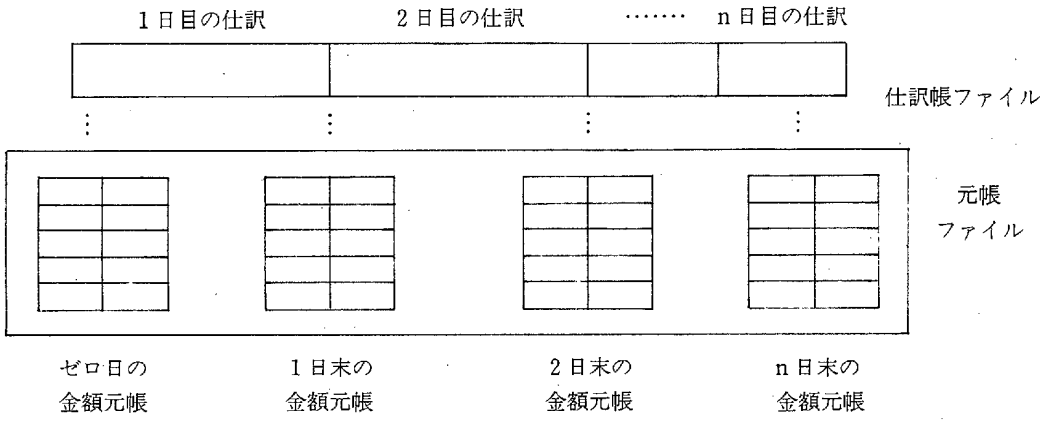
bihin. dat　……備品勘定ファイル
kaikake. dat……買掛金勘定ファイル
shihon. dat　……資本金勘定ファイル
uriage. det　……売上勘定ファイル
shiire. det　……仕入勘定ファイル
kyuryo. dat……給料勘定ファイル
eigyohi. dat……営業費勘定ファイル

と言うことになる。

この勘定毎にファイルを開設し、たえずそれに記入していく方法は、実は大変無駄な方法である。なぜなら仕訳帳と元帳からなる最も基本的な伝統的複式簿記の記入構造を反省すれば直ちに明らかになるように、基本的仕訳記録が仕訳帳に一個あり、それを多少修正したとはいえ、同じものが借方勘定の借方側に一個、貸方勘定の貸方側に一個、計三回重複記録されること↗↗になる。そのうえ、記録間の照合性を保つため、仕訳帳には元丁欄、元帳には仕丁欄記入という、本来の取引データとは無関係で、帳簿組織の相互参照を維持するためだけの情報を追加しなければならなくなる。この重複記録という点は多ファイル式コンピューター会計でも同じことになる。

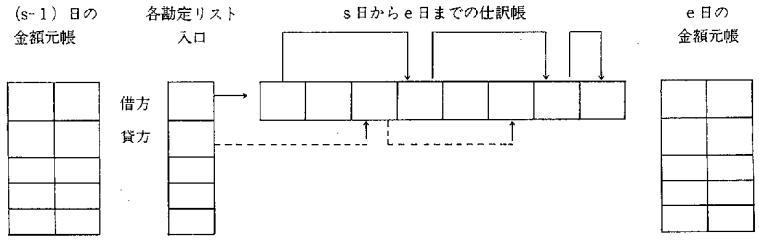
本会計システムでは、現在高元帳が既に存在しているのであるから、i日の現在高元帳からj日の現在高元帳にいたる変化がその間の取引記録（仕訳）によって証明されさえすればよいのであって、別に独立の歴史的記録を必要とするわけはないのである。

したがって、ポインター付会計システムでは、集計表としてのみ与えられる総勘定元帳の各勘定の歴史的記録内容をどのようにして検索復元するかを明らかにしなければならない。



まず、仕訳帳と総勘定元帳は次のような関係になっている。

仕訳帳は各営業日の仕訳がシーケンシャルに収納されている。元帳ファイルにはA(N,2)↗↗の金額元帳がゼロ日からn日末までシーケンシャルに収納されている。したがって、検索開始日(s日)の前日の金額元帳と検索終了日(e日)の金額元帳を読み込み、二つの金額元帳の各勘



定が、s日からe日までの仕訳帳の記録によってリンクされればよいことになる。

そこで、仕訳帳から読み込まれる仕訳swをリンク付レコードとして主記憶に展開するため、次の構造を与える。

```
type
  swk=record
        sptr : ptr ;
        sx : trrec ;
        nxtid, nxtic : ptr ;
      end ;
```

すなわち、sptrはs日からe日までの仕訳どうしをポインターでつなぐためのポインターであり、nxtidは仕訳の借方勘定を結びつけ、nxtic仕訳の借方勘定を結びつけるのである。

この仕訳のデータのポインター構造に対して、各勘定についてもリストの入口と末尾をもたせる。

```
  actn=record
        actn : str10 ;
        dsum, csum : real ;
        dptr, cptr, dtl, ctl : ptr ;
      end ;
```

そして、これで配列をつくり、各勘定へのリスト一覧表とする。

```
  actlist : array [acts] of act ;
```

こうしておいて、手続startで検索期間中の仕訳について、ポインター付き仕訳リストと勘定リストを作るのである。

```
    while not eof (fx) do
      begin
        read (fx, xtr) ;
        if (sdate<=xtr. date) and
            (xtr. date<=edate) then
            begin
              xswk. sx : =xtr ;
              setswk (xswk) ;
            end ;
      end ;
```

仕訳リストと各勘定の借方リスト及び貸方リストを作成する仕事は、手続setswkで行う。まず仕訳リストを待行列方式で作るため、

```
    setptr (slist, stl, xptr, stl^. sptr) ;
```

を実行し、借方勘定へのリストを作るためには借方合計金額を

```
    Dsum : =Dsum+xswk. sx. ta ;
```

で更新した上で

```
    setptr (Dptr, Dtl, xptr, Dtl^. nxtid) ;
```

で借方リストを作る。貸方リストも同様に作成するのである。

仕訳リストと各勘定の借方リスト、貸方リストが出来上がれば、後はこのリンク付リストをたどって出力するだけである。得意先元帳、仕入先元帳については総勘定元帳の検索とほとんど同じである。

異なる点は使用する元帳ファイルが異なる点と、共通の仕訳帳から関連するデータを選択する論理が少し異なるだけである。

たとえば得意先元帳の場合であれば

```
  if (sdate<=xtr. date) and (xtr. date<=
    edate) and ((xtr. xk=kakeuri) or (xtr.
    xk=kaketori)) then begin
      xswk. sx : =xtr ;
      setswk (xswk) ;
    end ;
```

となるのである。

6 商品元帳の検索

商品元帳の検索は商品有高帳の構造が、金額だけでなく、数量と単価を含んでいるため、商品リストの構造が違い、そのため、少しずつ異なってくる。

各商品へのリストは次の構造を持っている。

```
    act=record
          actn : char ;
          Dsqty, Csqty,
          Dsum, Csum : real ;
          Dptr, Cptr,
          Dtl, Ctl : ptr
        end ;
```

すなわち、各商品毎に借方ポインターと貸方ポインターを持っている点は共通であるが、借方金額合計（dsum）、貸方金額合計（csum）のほかに、借方数量合計（dsqty）、貸方数量合計（csqty）をも持たせてある。

したがって、商品リストは各商品について上記 act の構造を持つ配列になるのである。

```
gdlist：array［gds］of act；
```

したがって、商品元帳検索プログラムの手続き start は、この gdlist の初期化からはじまり、該当する仕訳を掛売上、現金売上、掛仕入について選択するのである。(本稿では現金仕入はないものとしている)。

```
read（fx, xtr）；
if（sdate＜＝xtr. date)
    and（xtr.date＜＝edate)
    and（(xtr. xk＝kakeuri)
        or（xtr. xk＝genkinuri)
        or（xtr. xk＝kakesiire))
then begin
        xswk. sx：＝xtr；
        setswk（xswk）；
end；
```

そして、各商品についてリストを作るとき、取引種類が掛仕入であれば借方数量合計と借方金額合計を計算した上でポインターでつなぎ、取引種類が掛売上、現金売上であれば貸方の計算を行った上で、ポインターでつなぐのである。

```
procedure setswk（yswk：swk）；
  var ygd：gds；
      gdname：char；
      XQTY：real；
      found：boolean；
  begin
    yswk. nxtid：＝nil；
    yswk. nxtic：＝nil；
    yswk. sptr：＝nil；
    new（xptr）；
    xptr^：＝yswk；
    setptr（slist, stl, xptr, stl^. sptr）；
    found：＝false；
      with yswk. sx do
      case xk of
        genkinuri：begin
          gdname：＝G1；XQTY：＝qtyl；
        end；
        kakeuri：begin
          gdname：＝G；XQTY：＝qty；
          end；
        kakegai：begin
          gdname：＝G2；XQTY：＝qty2；
          end；
    end；
  ygd：＝A；
  while（ord（ygd)＜＝ord（E））and（not found）do
    if NGDS［ord（ygd)＋1］＝gdname
    then
      with gdlist［ygd］，yswk. sx do
      begin
        case xk of
          kakegai：begin
            dsqty：＝dsqty＋XQTY；
            dsum：＝dsum＋ta；
            setptr(Dptr, Dtl, xptr, Dtl^. nxtid)；
            end；
        genkinuri, kakeuri：
          begin
            csqty：＝csqty＋XQTY；
            csum：＝csum＋ta；
            setptr(Cptr, Ctl, xptr, Ctl^. nxtic)
          end；
      end；
    found：true
    end
    else ygd：＝succ（ygd）；
  if not found then writeln（'商品名エラー'）；
end；
```