＊特集｜デシジョン・テーブルの効果的利用

デシジョン・テーブルによるプログラミング

COBOLプリコンパイラの利用

山陽国策パルプ(株)　制度計数部次長
芳賀敏郎

1．はじめに

ＥＤＰシステムを設計し，プログラムを作成するに当っては，まず，システムの仕様を作成する。仕様は，コンピュータ部門外の人にもわかるように，文章で書かれるのが普通である。しかし，言葉，とくに日本語は，システム仕様のように厳密性を要求されるものには向いていない。そこで，表や図などを併用して，文章の欠点を補うくふうがなされる。ここで，デシジョン・テーブルを用いると，極めて有効で，次のような利点を持つことは，本誌山武ハネウェル㈱の鈴木氏の解説からも明らかである。

a）ユーザ部門の人にも容易に理解でき，コミュニケーションの道具として使える。

b）複雑な条件と行動の組合せをコンパクトにまとめることができるので，見通しがよく，仕様のミスを大幅に減少することができる。

このように，デシジョン・テーブルには，多くの利点があるにもかかわらず，1961年頃提案されて以来，あまり普及していない。その大きな理由は，

「デシジョン・テーブルから直接プログラムのコーディングができないので，従来どおりフローチャートを作成しなければならない」

ところにあるものと思われる。

一方，ＥＤＰシステムが巨大化し，プログラムの内容が複雑になるにともなって，プログラムのメンテナンスのために要する労力が急速に増大している。また，コンピュータ要員のローテーション，組織の横割り化などの理由で「プログラムの標準化」，「トキュメンテーションとして役立つプログラム」，「誤りの起りにくいプログラミング技術」等が，コンピュータ部門の大きな課題となって来た。

これらの要請に応える手法が，いくつか開発されつつある。ワーニエ法，ストラクチャード・プログラミング等はその例であろう。後者の提唱者ダイクストラによれば，ＧＯ　ＴＯ　命令がプログラムの見通しを悪くするとして，ＧＯ　ＴＯ　命令のないプログラミングを提案している。

デシジョン・テーブルも，そのままプログラム記述の方法として使うことができれば，ＧＯ　ＴＯ命令をほとんど使わないで，プログラムを書くことができる。したがって，ダイクストラの提言を満す有力な手段として，期待することができる。

デシジョン・テーブルでプログラムを記述する試みは，すでにいくつかある。（1，2）これらの多くは，ＣＯＢＯＬやＦＯＲＴＲＡＮをホスト言語として，プログラムの一部をデシジョン・テーブルで記述し，これをプリコンパイラ（コンパイラの前処理プログラム）によ

図1
問題からプログラムまで

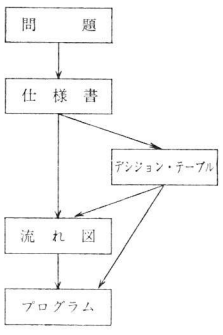


図2　プリ・コンパイラー

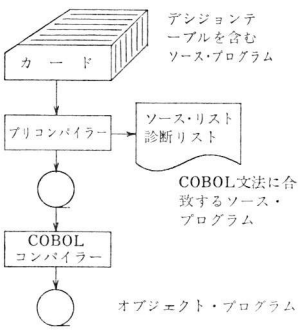
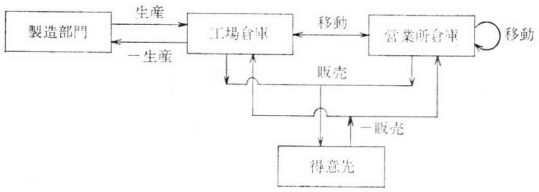


図3　物の流れ



って，ホスト言語の文法に合致したプログラムに書き直すという方法をとっている。この関係を図1と図2に示す。

デシジョン・テーブルを分解するロジックには，枝分れ法とマスキング法があり，実行効率を高めるための工夫がいろいろなされている。

ここでは，当社で開発し，日常業務で活用している，プリコンパイラを紹介する。使用コンピュータは「NEAC　2200　モデル200」，メモリ32KCで，中型でも比較的小さいものである。ホスト言語はCOBOLで，プリコンパイラはアセンブラで書かれ，約1,500ステップである。ロジックとしては，システムプログラムの経験のない者でも比較的簡単に組め，オブジェクトプログラムの長さを短くできる，マスキング法を使っている。従って，実行時の効率は枝分れ法に比して幾分劣るが，プログラム開発のコストダウンで充分採算にのっていると考えている。

2．例

簡単な受払残マスタの更新を例にして，デシジョン・テーブルの使い方を説明しよう。

2.1　仕様

(1)　物の流れ。（図3参照）

(2)　コード

○伝票コード

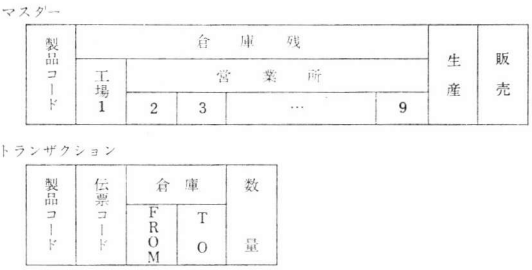
生産＝1，移動＝2，販売＝3

○倉庫コード

工場倉庫＝1，営業所倉庫＝2，3，…，9

（製造部門＝0，得意先＝10～　）

図4　テープ・レイアウト



(3)　マスタおよびトランザクション・ファイル

○テープ・レイアウト。（図4参照）

○いずれも，製品コード順にソートされている。

(4)　伝票コード・倉庫コードとマスタ更新との関係。（表1参照）

2.2　更新サブルーチンのコーディング

マスタとトランザクションの製品コードが一致したとき，トランザクションによってマスタの内容を更新するサブルーチンは，表1の仕様をそのままデシジョン・テーブルに表現することによって，プログラム2として得られる。

7カラムにCと書かれている，49～58行目は条件を表わし，左半分は条件スタブ，右半分の／／で囲まれた部分は条件エントリである。エントリには，任意の場所に：を入れて，表を見やすくすることができる。条件スタブには，54，57行目のように，COBOLで許される複合条件を書くこともできる。

7カラムにAを書かれている，59～64行目は行動スタブと行動エントリである。

条件エントリは左から順次テストされ，条件の一致する列に出合ったら，対応する行動エントリに従って，処理される。

エントリの初めの5列が表1の5行に対応していることは，容易に確かめられるであろう。最後の列は，どの条件にもあてはまらないコードの組合せで，エラーとして処理される。

56行目では，移動データの2つの倉庫コードが等しく

表　1

伝票コード D	倉庫コード FROM	倉庫コード TO	内　容	マスター更新
1	0	1	生産	工場残に　＋　　生産に　＋
	1	0	生産取消し	工場残から－　　生産から－
2	1～9	1～9	移動	FROM倉庫から－　　TO倉庫に＋
3	1～9	10～	販売	FROM倉庫から－　　販売に　＋
	10～	1～9	販売取消し	TO倉庫に　　＋　　販売から－

ないことをチェックしている。また，生産や販売に加えたり，引いたりする代りに，取消しの場合は61行目で数量の符号を逆にしておいてから，62または63行目で生産か販売に加えている。

このロジックを流れ図に書き，普通のＣＯＢＯＬコーディングをしたら，かなり多くの段落名やＧＯ　ＴＯ命令を必要とするであろう。

なお，ちなみに，ストラクチャード・プログラミングの考え方に従って，ＩＦ・・・ＴＨＥＮ・・・ＥＬＳＥ・・・型で書きくだしてみたのが，プログラム３である。ＧＯ ＴＯ命令はなくなるが，ＩＦ・・・ＴＨＥＮ・・・ＥＬＳＥ・・・が最大５段に入れ子になり，決して見通しのよいコーディングとはいえないであろう。

2.3　メイン・ルーチンのコーディング

２つのファイルをつき合せる処理は，初心者にとってそれ程簡単なものではない。しかし，この部分もデシジョン・テーブルを使って記述すると，極めて簡単になる。

図５のような流れ図が，プログラム１の後半のようにコーディングされる。

34行目と35行目の間の行は，コメントとして入れたものである。

39行目の行動エントリは，38，39の２行の行動スタブに対するものである。

ここに示した，ＭＡＩＮの部分は，ごくわずかの修正を加えるだけで，いろいろなタイプのマッチングに適用することができる。

プログラム１と２を合せると，一つのプログラムが完成する。PROCEDURE DIVISION はわずか43行でこれだけの仕様を満すことができる。さらに，よく見て頂けばわかるように，このプログラムには，ＧＯ　ＴＯ命令はただ１カ所41行目にあるだけである。

(PERFORM MAIN UNTIL HIGH-VALUES= SEIHIN-T AND SEIHIN-M という命令を使えば一つもいらなくなる)。

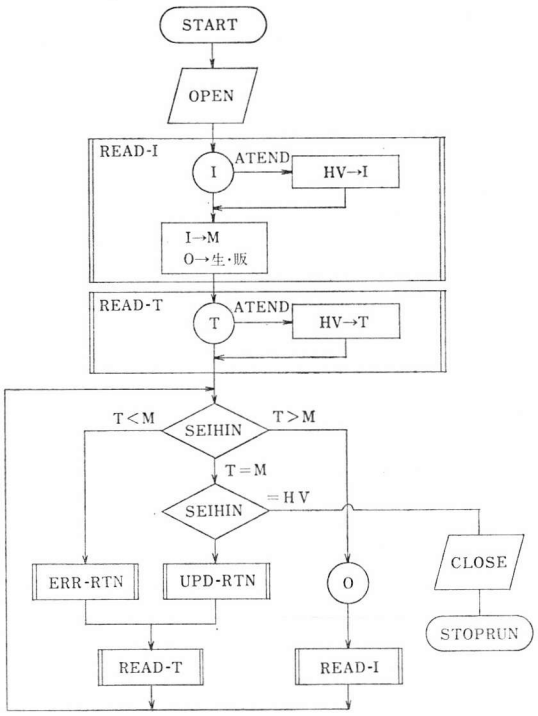
3．ま と め

前節にあげた例でおわかりのように，デシジョン・テーブルで直接コーディングすると次のようにいろいろの利点がある。

(1)　プログラムの生産性が高まる。（パターンの標準化と併用すると，この例程度のプログラムならば，１時間でコーディングでき，第１回のコンパイル・アンド・ゴーでＯＫとなる)。

(2)　プログラム・リストがそのままプログラム仕様書となり，改めてドキュメンテーションの労力をかけなくてすむ。

図５　メイン・ルーチンの流れ



(3)　他の人の作ったプログラムでも容易にメンテナンスが出来る。

デシジョン・テーブルを使っていると，使い方にも次のようにいろいろ工夫がされる。

(1)　大きなテーブルは適当に分割すると，見通しがよくなる。例のプログラムは２つのテーブルを使っているが，これを１つにまとめることもできる。しかし，それは好ましくない。

(2)　行動スタブの中にＩＦ文を入れることにより，テーブルの大きさを小さくすることができる。

以上デシジョン・テーブルの利点をあげたが，いかなる場所に使っても有利だというわけではない。途中で計算をしながら，いろいろの条件を聞いて行くということはできない。ストラクチャード・プログラムの悪い例を出したが，逆の場合もある。それぞれの状況に応じて，適した手法を使いわけるべきであろう。近い将来，当社のプリコンパイラに，ＤＯ　ＷＨＩＬＥとかＥＮＤＩＦ等の機能も織り込みたいと考えている。

文　献

(1)　System/360 and System/370 Decision Table Translator (DECTAT) for Cobol and PL/I. Application Description Manual. Program No. (5734-XR2)

(2)　㈱　ソフトウエア・リサーチ・アソシエイツ資料

プ ロ グ ラ ム　1

```
00001  DATA DIVISION.
00002  FILE SECTION.
00003  FD   I-FILE
00004  01   I-REC
00005  FD   T-FILE
00006  01   T-REC.
00007       02   SEIHIN-T
00008       02   D-T
00009       02   FROM-T
00010       02   TO-T
00011       02   SURYO-T
00012  FD   O-FILE
00013  01   O-REC
00014  FD   P-FILE
00015  01   P-REC
00016  WORKING-STORAGE SECTION.
00017  01   M-REC.
00018       02   SEIHIN-M
00019       02   ZAIKO-M OCCURS 9 TIMES
00020       02   SEISAN-M
00021       02   HANBAI-M
00022  PROCEDURE DIVISION.
00023  START.
00024       OPEN INPUT I-FILE T-FILE
00025            OUTPUT O-FILE P-FILE.
00026  READ-I.
00027       READ I-FILE  AT END MOVE HIGH-VALUES TO I-REC.
00028       MOVE I-REC TO M-REC
00029       MOVE 0 TO SEISAN-M HANBAI-M.
00030  READ-T.
00031       READ T-FILE  AT END MOVE HIGH-VALUES TO T-REC.
00032  MAIN.
       *    ------------------------------------------
00033  C    SEIHIN-T < SEIHIN-M.                 /Y---/.
00034  C    SEIHIN-T = SEIHIN-M.                 /-YY-/.
       *    SEIHIN-T > SEIHIN-M.                 /---Y/.
00035  C    SEIHIN-T = HIGH-VALUES.              /-YN-/.
       *    ------------------------------------------
00036  A    PERFORM ERR-RTN.                     /X---/.
00037  A    PERFORM UPD-RTN.                     /--X-/.
00038  A    PERFORM WRITE-O
00039  A    PERFORM READ-I.                      /---X/.
00040  A    PERFORM READ-T.                      /X-X-/.
00041  A            GO TO MAIN.                  /X-XX/.
       *    ------------------------------------------
00042       CLOSE I-FILE T-FILE O-FILE P-FILE
00043                                            STOP RUN.
00044  ERR-RTN.
00045       WRITE P-REC FROM T-REC.
00046  WRITE-O.
00047       WRITE O-REC FROM M-REC.
```

プログラム　2

```
00048  UPD-RTN.
       *   ------------------------------------------------------------
00049 C    D-T = 1.                                      /YY| |  | /.
00050 C    D-T = 2.                                      /  |Y|  | /.
00051 C    D-T = 3.                                      /  | |YY| /.
00052 C    FROM-T = 0.                                   /Y | | N| /.
00053 C    FROM-T = 1.                                   / Y| |  | /.
00054 C    FROM-T > 0 AND FROM-T < 10.                   /  |Y|YN| /.
00055 C    TO-T = 0.                                     / Y| |N | /.
00056 C    TO-T = 1.                                     /Y | |  | /.
00057 C    TO-T > 0 AND TO-T < 10.                       /  |Y|NY| /.
00058 C    FROM-T = TO-T.                                /  |N|  | /.
       *   ------------------------------------------------------------
00059 A    SUBTRACT SURYO-T FROM ZAIKO-M (FROM-T).       / X|X|X | /.
00060 A    ADD SURYO-T TO ZAIKO-M (TO-T).                /X |X| X| /.
00061 A    SUBTRACT SURYO-T FROM 0 GIVING SURYO-T.       / X| | X| /.
00062 A    ADD SURYO-T TO SEISAN-M.                      /XX| |  | /.
00063 A    ADD SURYO-T TO HANBAI-M.                      /  | |XX| /.
00064 A    PERFORM ERR-RTN.                              /  | |  |X/.
       *   ------------------------------------------------------------
```

プログラム　3

```
IF D-T = 1
    IF FROM-T = 0 AND TO-T = 1
        ADD SURYO-T TO ZAIKO-M (1)
        ADD SURYO-T TO SEISAN-M
    ELSE IF FROM-T = 1 AND TO-T = 0
        SUBTRACT SURYO-T FROM ZAIKO-M (1)
        SUBTRACT SURYO-T FROM SEISAN-M
    ELSE PERFORM ERR-RTN
ELSE IF D-T = 2
    IF FROM-T > 0 AND FROM-T < 10
        AND TO-T > 0 AND TO-T < 10 AND FROM-T NOT = TO-T
        SUBTRACT SURYO-T FROM ZAIKO-M (FROM-T)
        ADD SURYO-T TO ZAIKO-M (TO-T)
    ELSE PERFORM ERR-RTN
ELSE IF D-T = 3
    IF FROM-T > 0 AND FROM-T < 10 AND TO-T > 9
        SUBTRACT SURYO-T FROM ZAIKO-M (FROM-T)
        ADD SURYO-T TO HANBAI-M
    ELSE IF FROM-T > 9 AND TO-T > 0 AND TO-T < 10
        ADD SURYO-T TO ZAIKO-M (TO-T)
        SUBTRACT SURYO-T FROM HANBAI-M
    ELSE PERFORM ERR-RTN
ELSE PERFORM ERR-RTN.
```