奈良産業大学『産業と経済』第4巻第4号（1990年3月）43—60

≪研究ノート≫

構造化 COBOL 概要

西　口　清　治

目　　次

I. は　じ　め　に

近年，ソフトウェア開発の領域において，構造化プログラミング，生産性向上のための自動化ツールとしてのコンピュータ支援ソフトウェア・エンジニアリング（CASE, Computer Aided Software Engineerring），第四世代言語（4GL, Fourth Generation Language[1]），ＡＩ技術等の利用によって，ソフトウェア開発の信頼性や効率性の向上を目指してきた。

ソフトウェア開発において一般化した構造化プログラミングの考え方は，プログラムをモジュールに分割して，大まかな記述から詳細なものへの記述へ，論理構造を分かり易く，その制御構造は順次，選択，繰返しの三つの基本構造からなるプログラム構造を持つものであり，コーディング，デバッグ，保守の効率性を追求するものである。従って，ここで検討する構造化 COBOL は，このような構造化プログラミングに適する COBOL 文法によって作成される COBOL プログラムを意味する。

COBOL 言語は COmmon Business Oriented Language（事務用共通言語）として，1959年にアメリカで結成されたデータシステムズ言語協議会（CODASYL，コダシル，COnference on DAta SYstems Languages）が，1960年に COBOL を発表し，その後，我が国においても1972年８月にＪＩＳ（日本工業規格）として，COBOL-1972 が規定され，幾度

（1） 第一世代言語は計算機命令だけから成る機械語（マシン語，machine language），第二世代言語は計算機命令と対応するアセンブラ言語（assembly language），第三世代言語はコンパイラ言語（compiler language），第四世代言語は従来，簡易言語，プログラムレス言語等と呼ばれていたものを拡張したプログラム言語と考えられ，COBOL 等の第三世代言語と比較して，約10倍の生産性があると言われている。

かの改定を経て今日に至っている。従って，永年のプログラムの蓄積が多い。過去の COBOL プログラムとの互換性も保持しているために，進歩的なプログラム言語ではない。しかし，汎用のプログラミング言語としては今でも最も利用されているものである。

本稿では，一般ユーザーが個々のプログラムを開発する上でのソフトウェア領域の構造化プログラミング法によって採用される COBOL プログラミングの概要について検討する。

II. COBOL の特徴

事務処理に適するプログラミング言語である COBOL は，次のような特徴がある。

1. 高度の互換性

プログラムの記述法が規格化されており，その結果，共通性が確保されている。すなわち，ＪＩＳ(2)ではプログラムの高度な機械独立化を促進するために，COBOL で表現されるプログラムの書き方と解釈を規定しているので，特定のコンピュータ機械やソフトウェアに依存しないようにして，各機種の COBOL との互換性が維持されている。

まず，ＪＩＳでは，機能から見た言語規格の範囲を次のように規定している。

1. 中核（nucleus）機能

4つ部の基本的な構造の範囲内で，データを内部処理する機能であり，表の定義や呼出しを行う要素も含んでいる。機能の範囲によって水準1と水準2に分けられている。

2. 順ファイル（sequential i-o）機能

データのレコードが一定の順序で構成されているファイル（順編成ファイル，sequential organization file）の定義及び呼出しの機能である。水準1と水準2に分けられている。

3. 相対ファイル（relative i-o）

データが相対レコード番号によってランダムに入出力できるファイル（相対編成ファイル，relative organization file）の定義及び呼出しの機能である。乱呼出し（random access, 乱処理）と順呼出し（sequential access, 順処理）機能も含む。水準1と水準2に分けられている。

4. 索引ファイル（indexed i-o）

データのレコードに固有のキーと索引（index）を持つファイル（索引順編成ファイル，indexed sequential organization file）の定義及び呼出しの機能である。乱呼出しと順呼出し機能も含む。水準1と水準2に分けられている。

（2） 本稿での用語，説明はＪＩＳに準拠し，引用している。
日本規格協会編，「ＪＩＳハンドブック　情報処理　ソフトウェア編」，日本規格協会，1988を使用

5. プログラム間連絡（inter-program communication）

他のプログラムと制御やデータを移し，連絡し合える機能である。水準1と水準2に分けられている。

6. 整列併合（sort-merge）

ファイルの内容を並びかえたり（整列，sort，分類とも言う），複数のファイルを結合（merge，併合とも言う）する機能である。水準1の機能のみである。

7. 原始文操作（source text manipulation）

原始プログラムの翻訳の際，原文の挿入や置換を行う機能である。水準1と水準2に分けられている。

8. 報告書作成（report writer）

報告書に印字する処理を半自動的に作成する機能である。水準1の機能のみである。

9. 通信（communication）

通信文またはその一部の呼出し，処理，生成を行ったり，通信処理システムを介して通信用装置と通信する機能である。水準1と水準2に分けられている。

10. デバッグ（debug）

デバッグ用のアルゴリズムの機能である。水準1と水準2に分けられている。

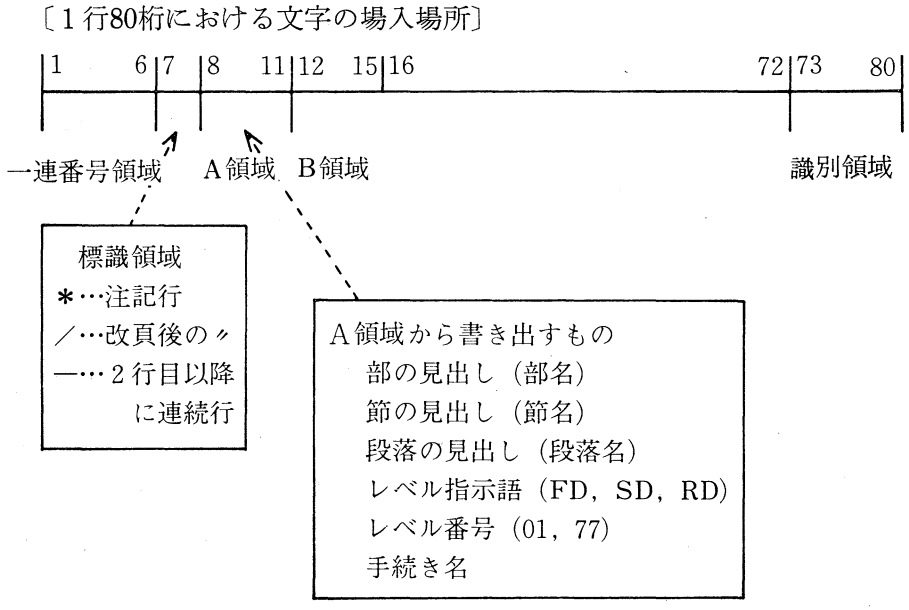
11. 区分化（segmentation）

手続き部の節を実行時にオーバーレイする機能である。水準1と水準2に分けられている。

なお，互換性の重要な要素である各部のプログラム文の書き方は後で述べる。

2. 正書法の遵守

COBOL 原始プログラムを書くときの標準体裁は正書法によって詳細に規定され守られて



いる。従来はコーディング用紙にプログラムを記述し，それを80欄カード等の記録媒体に変換して，コンピュータに入力していた。そこで，80欄カードの伝統からプログラムの一行は80桁に定められている。この80桁の中で特定の欄に記述できる項目が下記のように規定されている。一連番号領域，識別領域はプログラムを80欄カードに穿孔していた時には，プログラムの連続性を表示する際には必要であったが，最近のように，ディスプレイ装置によって対話形式でプログラム開発・コーディングする場合には重要性は少ない。通常は，プログラム開発支援ソフトウェアが自動的に番号を付して処理したり，COBOL 原始プログラムをコンピュータの実行可能なように翻訳する COBOL コンパイラーによっては，これらの番号を省略してもプログラムの実行には，差し支えがないこともある。

3. 文書性に富む

COBOL による原始プログラムの記述は，英語の通常の文構成のためコーディングが平易であり，通常の文章として記述できるので文書性がよい。反面，記述される文は冗長になりがちである。つまり COBOL では，プログラムの記述を，

① 見出し部
② 環境部
③ データ部
④ 手続き部

の４つの部に分割して，プログラムの識別，使用する装置の定義，データの定義，処理の定義等を分割して行うことによって，よく検討され作成されたプログラムは文書性が高いものとなっている。

III. COBOL 言語の構造

次に，原始プログラムを記述する文を構成するための基本的な規定である分離符，文字列，定数について述べる。

1. 分離符…空白や終止符等を用いて文字列を分離する記号
2. 文字列

1. COBOL の語

COBOL の語（WORD，文字列）は，30文字以内の文字列であって，利用者語，システム名，予約語であり，語の各文字は英文字，数字及びハイフンから構成される。（我が国で使用されている COBOL においては，カナ文字，漢字をも使用できることが多い）

(1) 利用者語…A, B, C, …, Z, 0, 1, 2, …, 9, —（但し，—は語頭と語尾では使用できない）
予約語でない30文字以内の文字列

(2) システム名…計算機名 (computer-name), 作成者語 (implementor-name, 言語名である。作成者の処理系で使える固有の機能を参照するためのシステム名), 言語名 (language-name) 等であり, 操作環境との連絡に用いる。

(3) 予約語…原始プログラムで用いる特定の語の組であり, 利用者語又はシステム名として使用してはならない。

a. 必要語…必要な時に省略できない語

b. 補助語…省略してもよい語

c. 特殊目的語…

イ. 特殊レジスタ (コンパイラが作り出す記憶場所, DEBUG-ITEM, LINAGE-COUNTER, LINE-COUNTER, PAGE-COUNTER の4種)

ロ. 表意定数 (特定の数値を意味する定数)

2. 定数

一定の文字列に付けた名称であり, 数字定数, 文字定数, 表意定数からなる。

数字定数…0から9までの数字, +, −, 及び小数点からなる文字列

文字定数…始まりと終わりを分離符で区切られた文字列

表意定数…コンパイラが作る特定の値を持つ定数

数値のゼロを表す ZERO, 空白を表す SPACE

IV. COBOL 原始プログラム

1. COBOL の構成

プログラムは次の4つの部から構成され, 次の順序に従って記述する。

① 見出し部 (IDENTIFICATION DIVISION)

プログラムの識別を示す部であり, プログラム名, 作成者名, 作成日等を記述する。

② 環境部 (ENVIRONMENT DIVISION)

機械構成, データ・ファイル名及びアクセス方法などの管理情報を記述する。

③ データ部 (DATA DIVISION)

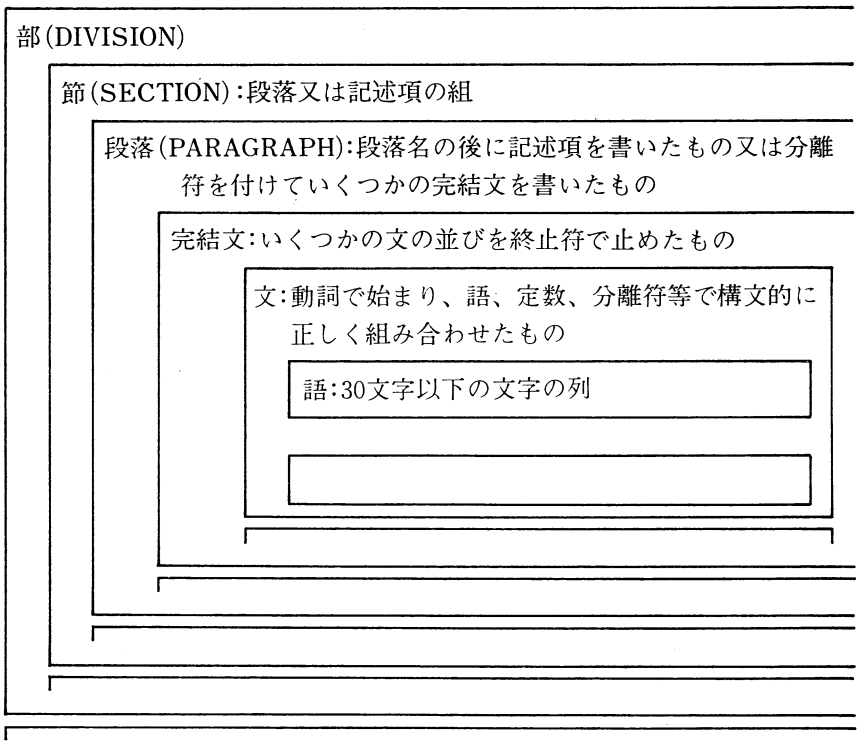
プログラムで使用するデータ項目名, データ・ファイル, 作業データ領域等を定義する部で, プログラムが処理するデータについては必ずここで定義しなければならない。

④ 手続き部 (PROCEDURE DIVISION)

データ処理の手続きを記述する。

2. 部の構造

部は階層的に, 次図のように構成されている。



3. 各部の記述法

1. 見出し部 (IDENTIFICATION DIVISION)

最初に記述する部であり，プログラムの識別を示す部である。プログラム名，作成者名，作成日等を記述する。“PROGRAM-ID. プログラム名.”は必ず記述する必要がある。

(一般形式)

IDENTIFICATION DIVISION(3).

PROGRAM-ID. プログラム名.

[AUTHOR. [注記項.]…]

・・・

2. 環境部 (ENVIRONMENT DIVISION)

環境部は構成節 (CONFIGURATION SECTION)，入出力節 (INPUT-OUTPUT SECTION) から構成され，機械構成，データ・ファイル名及びアクセス方法などの管理情報を記述する。

(一般形式)

ENVIRONMENT DIVISION.

[CONFIGURATION SECTION.

[SOURCE-COMPUTER. [翻訳用計算機記述項.]]

[OBJECT-COMPUTER. [実行用計算機記述項.]]

---

(3) 下線を付した語は必要語であり，省略できない。下線がない予約語は補助語のため省略してもよい。
{ } で囲まれた文字列は省略できない。[ ] で囲まれた文字列は省略できる。

[SPECIAL-NAMES. [特殊名記述項.]]]

[INPUT-OUTPUT SECTION.

FILE-CONTROL. {ファイル管理記述項.}…

[I-O-CONTROL. [入出力管理記述項.]]]

上記の「ファイル管理記述項」はプログラムで使用するファイル名とオペレーティング・システムで使用するアクセス名(4)を対応させる。次の例ではファイル名 URI-FILE を U01 に対応させ，LST-FILE を U02 に対応させている。

(例)(5)

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. FACOM340R.
OBJECT-COMPUTER. FACOM340R.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
        SELECT URI-FILE ASSIGN TO U01.
        SELECT LST-FILE ASSIGN TO U02.
        ……
```

3. データ部 (DATA DIVISION)

(1) データ部の構成

データ部は，ファイル節 (FILE SECTION)，作業場所節 (WORKING-STORAGE SECTION)，連絡節 (LINKAGE SECTION)，通信節 (COMMUNICATION SECTION)，報告書節 (REPORT SECTION) から構成され，プログラムで使用するデータ項目名，データ・ファイル，作業データ領域等を定義する部で，プログラムが処理するデータについては必ずここで定義しなければならない。

(一般形式)

DATA DIVISION.

[FILE SECTION.

[ファイル記述項 {レコード記述項}…]…]

[WORKING-STORAGE SECTION.

[独立項目記述項 / レコード記述項]…]

---

(4) ここでの用語の使用は FACOM OSIV/X8 での名称を一部に用いている。

(5) ここで例示している COBOL プログラムは富士通株式会社編著「OSIV/X8 FSP PFD 使用法自習書」166ページを使用している。

[LINKAGE SECTION.

[独立項目記述項 / レコード記述項] …] …]

(2) ファイル節（ファイル・セクション，FILE SECTION）

上記のファイル記述項は，プログラムで使用するファイルの構造を定義するもので，一般形式は次のようになる。

FD ファイル名

[BLOCK CONTAINS 整数 {RECORDS / CHARACTERS}]

[LABEL {RECORD IS / RECORDS ARE} {STANDARD / OMITTED}]

本ファイル節で記述するレコード記述項（特定のレコードの性質を記述するデータ記述項の組）は入出力ファイルで使用するデータ・レコードの定義であり，形式は次のようになる。

レベル番号 [データ名 / FILLER] [{PICTURE / PIC} IS 文字列]

なお，次の作業場所節で記述するデータ記述項の一般形式は次のようになる。

レベル番号 [データ名 / FILLER] [{PICTURE / PIC} IS 文字列] [VALUE IS 定数].

ここで用いた用語の意味は次のようになる。

① レベル番号01～49の番号を用いてデータの階層関係を示す。そして，集団項目と，それ以上細分化されない項目を基本項目と言い，また，それ以外に独立項目がある。

② FILLER（フィラー，無名項目）は，データをレコード中で直接参照しない時に用いる。

③ PICTURE（ピクチャー，または PIC）は，基本項目や独立項目の一般的性質や，編集される形式を示す。よく使用されるものを下記に示す。

9 数字１個の桁を示し，項目の１桁に数える。
99999と9(5)は同じ意味を示す。以下の PICTURE 文字も同じ

X 文字１個の桁を示し，項目の１桁に数える。

S 演算符号の存在を示し，通常は項目の１桁に数えない。

V 想定小数点の位置を示し，項目の１桁に数えない。

Z 先行するゼロ列を空白に置き換え，項目の１桁に数える。

, 文字“,”（コンマ）を挿入する位置を示し，項目の１桁に数える。

¥ 通貨編集用文字（¥）の１桁を示し，項目の１桁に数える。

④ VALUE（バリュー）は，作業場所節及び通信節のデータ項目に初期値を与える。

（例）

```
DATA DIVISION.
FILE SECTION.
FD URI-FILE BLOCK CONTAINS 10 RECORDS
            LABEL RECORD IS STANDARD.
01 URI-DATA.
   02 HIN-CODE    PIC   X(4).
   02 HIN-NAME    PIC   X(15).
   02 URI-SUU     PIC   9(6).
   02 TANKA       PIC   9(6).
```

(3) 作業場所節（ワーキング・ストレージ・セクション, WORKING-STORAGE SECTION)

作業レコードや，累計データやコンスタントな数字を使う時等の中間データを格納する領域を定義する節である。

（例）

```
WORKING-STORAGE SECTION.
77 KINGAKU      PIC 9(9)   VALUE   ZERO.
01 LST-HEAD1.
   02 FILLER PIC X(24) VALUE SPACE.
   02 FILLER PIC X(26) VALUE "U R I A G E   I C H I R A".
   02 FILLER PIC X(12) VALUE "N   H Y O   ".
01 LST-HEAD2.
   02 FILLER   PIC X(21) VALUE "          SHOHIN-COD".
   02 FILLER   PIC X(20) VALUE "E    SHOHIN-MEI    ".
   02 FILLER   PIC X(20) VALUE "  SURYO      TANKA ".
   02 FILLER   PIC X(20) VALUE "     KINGAKU       ".
```

4. 手続き部 (PROCEDURE DIVISION)

手続き部では上記の環境部，データ部で指定された項目についてのデータ処理の手続きを記述する。処理を記述するために語を組み合わせて文を作成する。

(1) 主要な語（命令）

① 入出力関係

ACCEPT, CLOSE, DISPLAY, OPEN, READ, REWRITE, START, USE, WRITE

② データ操作関係

INITIALIZE, INSPECT, MOVE, SEARCH, SET, STRING, UNSTRING

③ 算術関係

ADD, COMPUTE, DIVIDE, MULTIPLY, SUBTRACT

④ 条件関係

EVALUATE, IF

⑤ 手続き分岐関係

EXIT, GO TO, PERFORM

⑥ 無操作関係

CONTINUE

⑦ 停止操作関係

STOP

(2) 基本処理手順と命令

一般的な処理手順を基本命令文に対応させて表現すると次図のよう関係になる。

(3) 文の基本形式（ファイルの操作は順ファイルに関連するものを中心として解説）

① OPEN…ファイルの使用開始を宣言する時に使用する。

OPEN { INPUT {ファイル名}… / OUTPUT {ファイル名}… / I-O {ファイル名}… }

② READ…ファイルからレコードを読み込む

READ ファイル名 [NEXT] RECORD [INTO 一意名(6)]
[AT END 無条件文1(7)]
[NOT AT END 無条件文2]

（6） 一意名（identifier）は添字や修飾によって，一意になるデータ名を指す。

（7） 無条件の操作をする文であり，条件文に対する文である。

[END-READ]

③ MOVE…データを転記する。意味としては転記元のデータは残っているので，いわゆるCOPY に相当する。

MOVE { 一意名1 / 定数 } TO { 一意名2 }

④ COMPUTE…四則演算の式を作る。

COMPUTE {一意名1 [ROUNDED]}… =算術式1
[ON SIZE ERROR 無条件文1]
[NOT ON SIZE ERROR 無条件文2]
[END-COMPUTE]

⑤ IF…条件によって，次の処理の流れを変える。

IF 条件1 { THEN {文1}… / NEXT SENTENCE } { ELSE {文2}… [END-IF] / ELSE NEXT SENTENCE / END-IF }

⑥ PERFORM…通常の処理の流れから離れて，指定した手続きを処理し，指定した処理の場所に戻ったり，繰り返し処理を行ったりする。

a．書き方1

PERFORM [手続き名1 [{ THROUGH / THRU } 手続き名2]]
[無条件文1 END-PERFORM]

b．書き方2

PERFORM [手続き名1 [{ THROUGH / THRU } 手続き名2]]
{ 一意名1 / 整数1 } TIME [無条件文1 END-PERFORM]

c．書き方3

PERFORM [手続き名1 [{ THROUGH / THRU } 手続き名2]]
[WITH TEST { BEFORE / AFTER }] UNTIL 条件1
[無条件文1 END-PERFORM]

d．書き方4

PERFORM [手続き名1 [{ THROUGH / THRU } 手続き名2]]

[WITH TEST {BEFORE / AFTER}]

VARYING {一意名2 / 指標名1} FROM {一意名3 / 指標名2 / 定数1} BY {一意名4 / 定数2} UNTIL 条件1

[AFTER {一意名5 / 指標名3} FROM {一意名6 / 指標名4 / 定数3} BY {一意名7 / 定数4} UNTIL 条件2 …]

[無条件文1 END-PERFORM]

⑦GO TO…処理の実行順序を変更し，指定した手続きに制御を移す。一般的に構造化プログラミングでは，「GO TO 文」を使わないが，場合によっては，使用した方が，プログラムの流れを分かりやすくすることがあるので，必ずしも「GO TO 文」を廃止しているのではない。

a．書き方1

GO TO [手続き名1]

b．書き方2（一意名1の値が1,2…，nの時に手続き名1，手続き名2，…，手続き名nに実行順序を変更する。）

GO TO {手続き名1} … DEPENDING ON 一意名1

⑧ WRITE…処理結果を書き出す

WRITE レコード名1 [FROM 一意名1]

[{BEFORE / AFTER} ADVANCING {{一意名2 / 整数1} [LINE / LINES] / {呼び名1 / PAGE}}]

[AT {END-OF-PAGE / EOP} 無条件文1]

[NOT AT {END-OF-PAGE / EOP} 無条件文2]

[END-WRITE]

⑨ COLSE…ファイルの処理を終了する

CLOSE {ファイル名} …

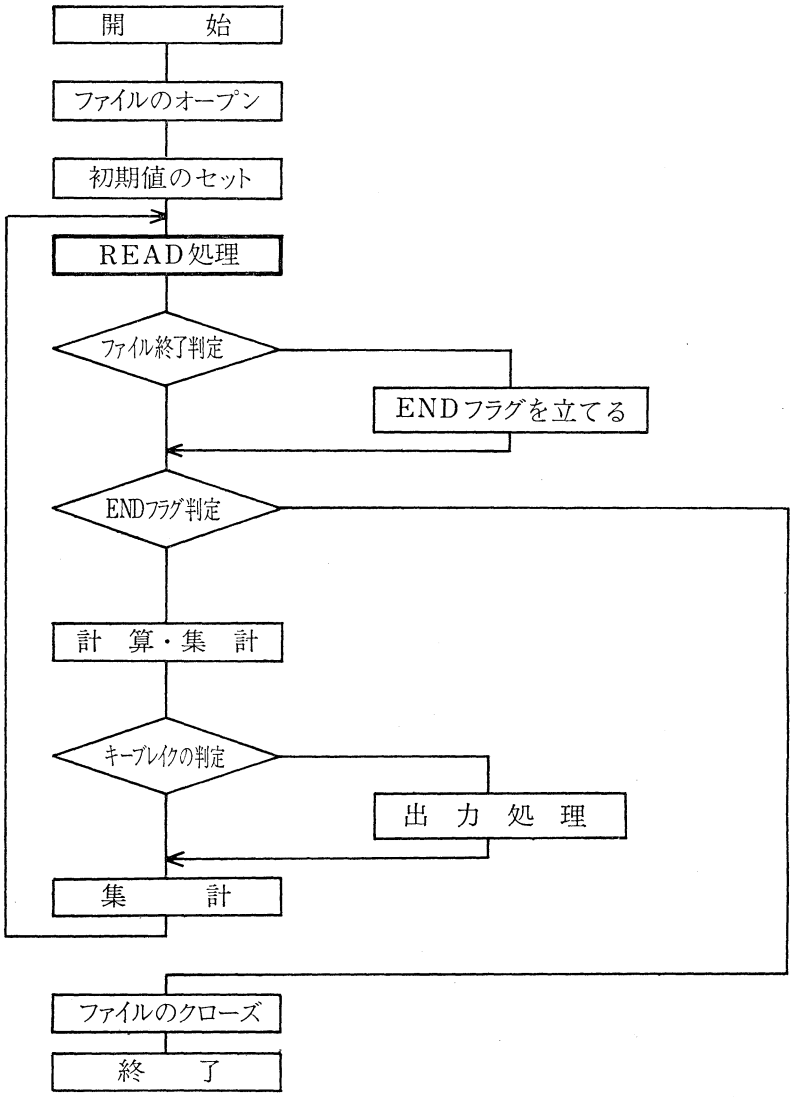
⑩ STOP…処理の実行を停止する。

STOP {RUN / 定数1}

V. 基本処理フローと COBOL の実行

1. 基本処理フローの一般形

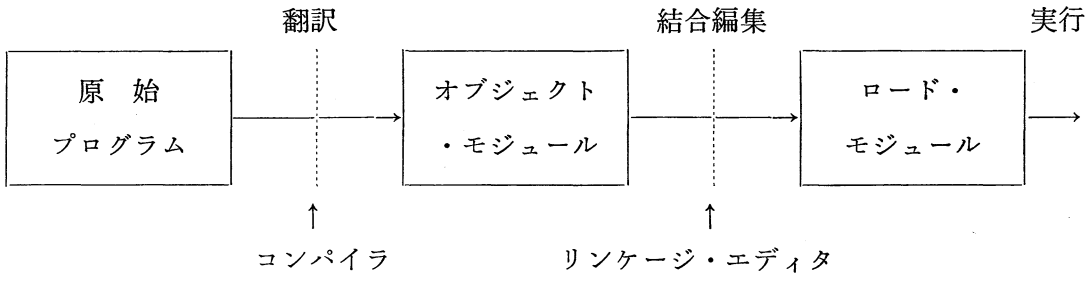
前述した基本処理手順を処理フローの形で表現すれば，一般的には以下のようになる。



2. 原始プログラムの翻訳から実行への処理

COBOL 言語で記述されたプログラムを原始プログラム（source program）と言い，これをコンピュータの理解できるよう翻訳（コンパイル，compile）する。この翻訳するソフトウェアを COBOL コンパイラ（compiler）と呼ぶ，翻訳されたプログラムをオブジェクト・モジュール（object module）あるいは目的プログラム（object program）と呼んでいる。次にオブジェクト・モジュールを結合編集処理により，コンピュータが実行可能なロード・モジ

ュール (load module) を作成する。この結合編集するソフトウェアをリンケージ・エディタ (linkage editor, LIED) と呼んでいる。



3. JCL (ジョブ制御言語 Job Control Language)

実際に，下記のプログラムを実行するためには，COBOL コンパイラに処理の指示を与えるためのコマンド，あるいは通常，バッチ処理においてはＪＣＬを作成する。

1. ＪＣＬ

ジョブ（コンピュータ処理する仕事の単位）の流れに従って，ジョブの定義に必要な情報を記述する。ジョブ制御文 (Job Control Statement) として表現され，下記のものがある。

①．ＪＯＢ文（ステートメント）

ジョブの開始，ジョブ名，メッセージの出力先などの処理条件を記述する。

② ＥＸ文 (EXEC 文)

処理プログラム名，処理プログラムの実行 (execuetion) に必要なパラメータ等を記述する。ジョブステップ（ジョブの処理単位）を示す。

③ ＦＤ文（ＤＤ文) File Definition

ジョブで用いる入出力のファイル及び作業ファイル名，入出力装置等を記述する。

プログラムで使用しているアクセス名と，ファイル名を結びつける。

(例)

OSIV/X8 は“¥”記号から書き始める。(他ＯＳでは“／”記号が多い)

```
·····1·····2·····3·····4·····5·····6·····7··
¥ JOB XXXXXX,LIST=(A,JD)
¥ CBLCLG SOUT=A
¥COB.SLIB FD SLIB=DA,FILE=XXXXXX.EX.COBOL,MEMBER=REIDAI
¥GO.U01 FD U01=DA,FILE=XXXXXX.EX.DATA,VOL=000001
¥GO.U02 FD U02=DA,VOL=WORK,TRK=(10,2,RLSE),SOUT=A
¥ JEND
```

(説明)

¥ JOB XXXXXX, LIST＝(A, JD)

ジョブ XXXXXX の処理を開始する。ジョブのメッセージ・リストはAプリンターに出力（印刷）する。その際，ジョブ制御文のマクロ展開形も出力する。（ただし，入力したジョブ制御文を出力する場合は LIST＝(A, JS) とする。）

¥ CBLCLG SOUT＝A

COBOL プログラムの翻訳，結合編集，実行を行う。

(COBOL プログラムの翻訳のみなら CBLC，翻訳・結合編集なら CBLCL とする。)

¥ COB. SLIB FD SLIB＝DA, FILE＝XXXXXX. EX. COBOL, MEMBER＝REIDAI

翻訳される COBOL プログラムはアクセス名 SLIB (機種名は DA (Direct Access) である）から入力する。プログラム XXXXXX. EX. COBOL は区分編成ファイルでその内の REIDAI プログラムを翻訳する。SLIB は Source Library，つまりソースモジュール（原始プログラム）の入力用のアクセス名である。

¥ GO. U01 FD U01＝DA, FILE＝XXXXXX. EX. DATA, VOL＝000001

プログラムの実行の際に使用される COBOL プログラム上のアクセス名 U01 は，機種名では DA (Direct Access) であり，ファイル名は XXXXXX. EX. DATA である。ディスク装置（ボリューム通し番号000001）に存在している。

¥ GO. U02 FD U02＝DA, VOL＝WORK, TRK＝(10, 2, RLSE), SOUT＝A

プログラムの実行の際に使用される COBOL プログラム上のアクセス名 U02 は，機種名では DA(DASD, Direct Access Storage Device) であり，システムで用意している WORK ボリュームを使用する。データの領域は最初10トラック確保し，足らなければ2トラックづつ15回領域を追加する，領域が余れば解放する (release)，この結果はアウトコントロール経由でAプリンターに出力する。

¥ JEND

ジョブの終わりを示す。

VI. プログラム例

ディスク上の売上データ（アクセス名：U01，データセット名：EX. DATA) を読み込み，売上一覧表（アクセス名：U02,) を印刷するプログラム (REIDAI)。(行番号は省略している)

```
.....1.....2.....3.....4.....5.....6.....7..
       IDENTIFICATION DIVISION.
       PROGRAM-ID.     REIDAI.
       ENVIRONMENT     DIVISION.
       CONFIGURATION   SECTION.
       SOURCE-COMPUTER. FACOM340R.
```

```
OBJECT-COMPUTER. FACOM340R.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
      SELECT URI-FILE ASSIGN TO U01.
      SELECT LST-FILE ASSIGN TO U02.
DATA          DIVISION.
FILE SECTION.
FD URI-FILE BLOCK CONTAINS 10 RECORDS
            LABEL RECORD IS STANDARD.
01 URI-DATA.
   02 HIN-CODE    PIC    X(4).
   02 HIN-NAME    PIC    X(15).
   02 URI-SUU     PIC    9(6).
   02 TANKA       PIC    9(6).
FD LST-FILE
            LABEL RECORD IS STANDARD.
01 LST-DATA.
   02 LST-CNTL    PIC    X(1).
   02 LST-DA      PIC    X(136).
WORKING-STORAGE SECTION.
77 KINGAKU      PIC 9(9)   VALUE    ZERO.
77 KIN-TOTAL    PIC 9(9)   VALUE    ZERO.
77 LINE-CNT     PIC 9(9)   VALUE    ZERO.
77 END-SWU      PIC 9(1)   VALUE    ZERO.
01 LST-HEAD1.
   02 FILLER PIC X(24)  VALUE  SPACE.
   02 FILLER PIC X(26)  VALUE "U R I A G E   I C H I R A".
   02 FILLER PIC X(12)  VALUE "N   H Y O   ".
01 LST-HEAD2.
   02 FILLER    PIC X(21)  VALUE "          SHOHIN-COD".
   02 FILLER    PIC X(20)  VALUE "E     SHOHIN-MEI    ".
   02 FILLER    PIC X(20)  VALUE "   SURYO      TANKA ".
   02 FILLER    PIC X(20)  VALUE "      KINGAKU       ".
01 LST-BODY.
   02 FILLER    PIC X(14)  VALUE  SPACE.
   02 LST-CODE  PIC X(4).
```

```
    02  FILLER     PIC  X(7)    VALUE  SPACE.
    02  LST-NAME  PIC  X(15).
    02  FILLER     PIC  X(3)    VALUE  SPACE.
    02  LST-SUU   PIC  ZZZ,ZZ9.
    02  FILLER     PIC  X(3)    VALUE  SPACE.
    02  LST-TANKA PIC  ¥,¥¥¥,¥¥9.
    02  FILLER     PIC  X(3)    VALUE  SPACE.
    02  LST-KIN   PIC  ¥,¥¥¥,¥¥¥,¥¥9.
01  LST-TOTAL.
    02  FILLER     PIC  X(51)    VALUE  SPACE.
    02  FILLER     PIC  X(10)    VALUE  "SO-GOKEI  ".
    02  LST-KIN   PIC  ¥¥¥,¥¥¥,¥¥¥,¥¥9.
PROCEDURE      DIVISION.
HAJIME.
    PERFORM  PROC1.
    PERFORM  PROC2  UNTIL  END-SW  =  1.
    PERFORM  PROC3.
    STOP     RUN.
PROC1.
    OPEN  INPUT  URI-FILE
          OUTPUT LST-FILE.
          MOVE  25  TO  LINE-CNT.
    READ  URI-FILE  AT  END
                    MOVE  1  TO  END-SW.
PROC2.
    IF  LINE-CNT  =  25
        MOVE  ZERO  TO  LINE-CNT
        MOVE  SPACE  TO  LST-DATA
        WRITE  LST-DATA  AFTER  PAGE
        WRITE  LST-DATA  FROM  LST-HEAD1  AFTER 3
        WRITE  LST-DATA  FROM  LST-HEA21  AFTER 3.
    MOVE  HIN-CODE  TO  LST-CODE.
    MOVE  HIN-NAME  TO  LST-NAME.
    MOVE  URI-SUU   TO  LST-SUU.
    MOVE  TANKA     TO  LST-TANKA.
    COMPUTE  KINGAKU  =  URI-SUU  *  TANKA.
```

```
    MOVE  KINGAKU   TO  LST-KIN.
    ADD  1  TO  LINE-CNT.
    WRITE  LST-DATA  FROM  LST-BODY  AFTER  2.
    ADD  KINGAKU TO  KIN-TOTAL.
    READ  URI-FILE  AT  END
                    MOVE  1  TO  END-SW.
PROC3.
    IF  LINE-CNT  =  25
        MOVE  ZERO  TO  LINE-CNT
        MOVE  SPACE  TO  LST-DATA
        WRITE  LST-DATA  AFTER  PAGE
        WRITE  LST-DATA  FROM  LST-HEAD1  AFTER 3
        WRITE  LST-DATA  FROM  LST-HEA21  AFTER 3.
    MOVE  KIN-TOTAL  TO  LST-TOT.
    WRITE  LST-DATA  FROM  LST-TOTAL  AFTER  2.
     CLOSE  URI-FILE  LST-FILE.
```

＊データ例

```
 ·····1·····2·····3·····4·····5·····6·····7··
 0102ﾃﾞﾝｷｶﾞﾏ ｼｮｳ    000312008000
 0104ﾃﾞﾝｷｶﾞﾏﾀﾞｲ    000085009480
    ·
    ·
    ·
```