# 「プログラム構造の最適化アプローチ」について

鍬　塚　泰　亨



## 1．はじめに

毎日のコンピュータ学習・実務の中で、学生・プログラマは，方法論としてのプログラミングの本質を，つまりプログラムの論理構造を正しく組立てる方法を知ることが大へん重要であると考える。

プログラミング技術は経験から生まれるものであるが，経験による表面的な知識の多様さより，むしろ，ものの考え方，いいかえれば，明確な論理基盤に立った，論理思考の方法・思考の過程を正しく表現する手段を身につける必要を感じる。そこで通常，教室学習あるいは職場実務では，あまり積極的には表面に出ない考えで，プログラムの組立て方にアプローチしていきたい。

## 2．データの階層構造を知る

まず，情報処理システムでは，インプット・データ，処理プログラム，アウトプット・データは「情報集合」としてとらえられる。通常インプットおよびアウトプット・データはフォーマッティング（製定化）されており，例えば実務における伝票・帳表などは，いろいろな項目がデータあるいは情報

として階層的に記載されている。したがって，インプットおよびアウトプット・データは，フォーマット上に含まれる「項目群の集合」としてとらえることができる。

一方の処理プログラムは，よく知られているように「命令群の順序集合」としてとらえられる。

まず，インプットおよびアウトプット・データの情報集合としての論理構造を明確にするために，

◦集合の「細分化」を行ない，

◦その結果を「構造図」で書き表わす。

以下にその手順を示す。

① インプットおよびアウトプット・データ・フォーマットに記載されている「項目群の集合」をまず，はっきり認識する。

② 「部分集合間の階層関係（どの項目群が上位レベルの集合，下位レベルの集合か）」をとらえる。

③ 各部分集合を，「1回だけ現れるもの」と，「繰返し現われるもの」に分けて，フォーマット上に記載されている項目の順序に合せて，上部から順次記述していく。

④ 「繰返し現われる」部分集合は，さらに，細分化の対象になり，「1回だけ現われる」部分集合になるまで，③の要領で記述していく。

次に，フォーマティングされた表を例に，集合の細分化の結果と，構造図の例を示す。

＜例1＞

◦給与支給額一覧表フォーマット

| 営業所コード | 部門コード | 従業員コード | 給与総額 |
|---|---|---|---|
| | | | |
| | | | |
| | | | 部門別合計 |
| | 部門コード | 従業員コード | 給与総額 |
| | | | |
| | | | 部門別合計 |
| | | | 営業所会計 |
| 営業所コード | 部門コード | 従業員コード | 給与総額 |
| | | | |
| | | | 総合計 |

（図1）

◦アウトプット・データの構造図

```
{給与支給額一覧表ファイル｜          <-------- 全体集合
  01  Ω {営業所}                   <-------- 一覧表レベル
    02    営業所コード
    02  Ω {部門}                   <-------- 営業所レベル
      03    部門コード
      03  Ω {従業員}               <-------- 部門レベル
        04    従業員コード          <-------- 従業員レベル
        04    給与総額
      03    部門別合計
    02    営業所別合計
  01    総合計
}
```

（図2）

（注）1．Ω｛ ｝は，繰返部分集合（細分化の対象となる部分集合）を表わす。
2．01～04は，レベル（階層）番号を表わす。

◦インプット・データの構造図

```
{インプット・データ・ファイル｜   ←-------- 全体集合
   01  Ω {営業所}               ←-------- ファイル・レベル
     02  Ω {部門}               ←-------- 営業所レベル
       03  Ω {従業員}           ←-------- 部門レベル
         04   営業所コード
         04   部門コード        ←-------- 従業員レベル
         04   従業員コード
         04   給与総額
}
```

（図３）

## ３．プログラムの基本構造を知る

インプットおよびアウトプット・データの論理構造が明らかになり，

① アウトプット・データの構造図を作成し，

② アウトプット・データの構造図にもとづいて，インプット・データの構造図が作成されると，

次にプログラムの構造図を作成しなければならない。よく知られている基本原則，

◦インプット・データは処理プログラムの対象となる。

◦処理プログラムはインプット・データを処理する。

◦アウトプット・データは処理プログラムより作られる。

から，結論づけられることは，

③「プログラムの構造は，インプット・データにもとづいて作成し，アウトプット・データでチェックされなければならない」ということである。

また，一般的にプログラム論理は，

◦繰返部分の構造　（繰返構造）

◦分岐部分の構造　（分岐構造）

で構成されていることはよく知られている。したがって，この２つの基本構造の特徴と，構造図の形式を定義し，例を示す。

(1)　繰返構造

インプット・データ集合が繰返構造であるとは，「同じ性質をもつデータ集合を複数個含む１個の集合」をいう。したがって，上述③から，インプット・データ集合が繰返構造であれば，プログラムも繰返構造となることがわかる。

繰返構造プログラムが，ｎ個の要素を含むデータ集合を処理するときの基本構造図を次のように定義する。

◦基本構造図

$\pi$　{プログラム｜

　01　{前処理}

　01　Ω{$\alpha$，{１要素の処理}}

　01　{後処理}

}

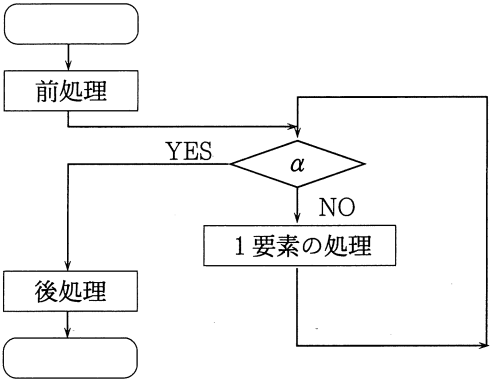
（図４）

（注）３．$\pi$｛　｝は，プログラムの処理の流れが各レベルで上から下へ直線的に流れることを表す。

４．Ω｛　｝は，条件$\alpha$を満足するまで｛１要素の処理｝を繰返し実行することを表わす。

５．｛　｝は命令群の順序集合を表わす。

◦基本流れ図




（図5）

繰返構造プログラムの例を示す。

（図3）のインプット・データ　構造図から，プログラム構造図を作成すると以下のようになる。

◦構造図

π {プログラム |

01　{前処理}

01　Ω {α,　{営業所} }　　<--------- ファイル・レベルの処理

02　{前処理}

02　Ω {β,　{部門} }　　<--------- 営業所レベルの処理

03　{前処理}

03　Ω {γ,　{従業員} } <--------- 部門レベルの処理

03　{後処理}

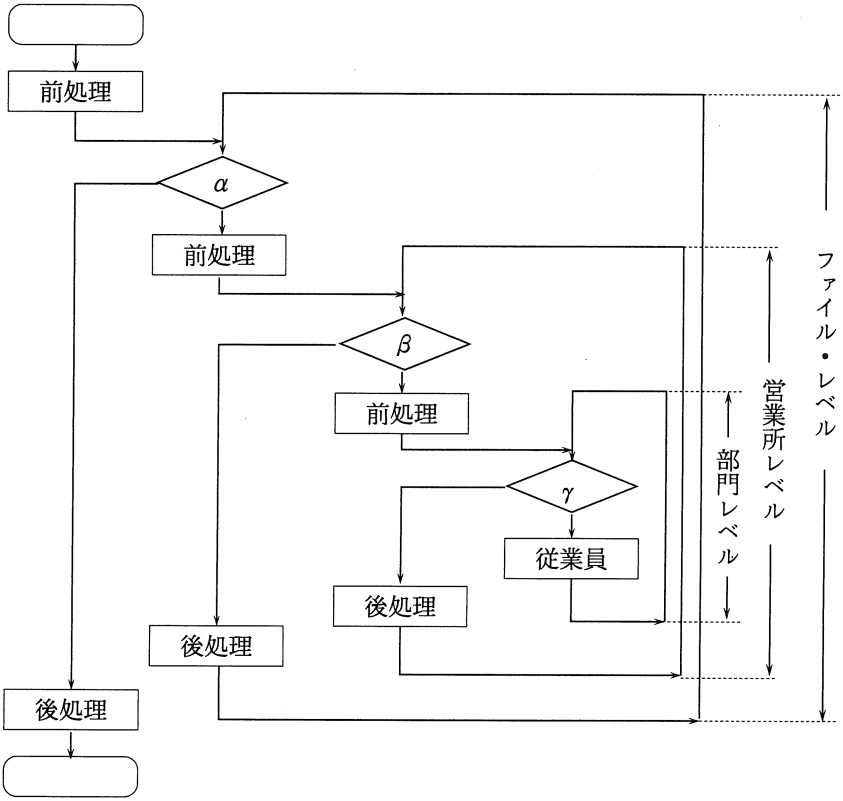
02　{後処理}

01　{後処理}

}

（図6）

◦流れ図




（図７）

⑵　分岐構造

インプット・データ集合が分岐構造であるとは，「現われたり，現われなかったりする１個以上の部分集合が存在する」ことをいう。前述⑴と同様，インプット・データ集合が分岐構造であれば，プログラムも分岐構造となる。

基本構造図を次のように定義する。

◦基本構造図

π {プログラム |

01 {前処理}

01 △ {A, {処理X} }

01 △ {$\overline{A}$, {処理Y} }

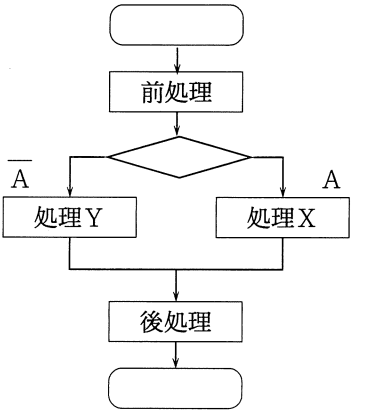
01 {後処理}

}

（図8）

（注）6．△ {A, {処理X} } は，条件結果がAであれば， {処理X} を実行することを表わす。

◦流れ図




（図9）

分岐構造プログラムの例を示す。

＜例2＞

得意先に関する残高一覧表を作成する業務がある。アウトプット一覧表およびインプット・ファイルは次のとおりとする。

◦アウトプット一覧表フォーマット

| 得意先コード | 得意先名 | 伝票No. | 借方 | 貸方 |
|---|---|---|---|---|
| | | | | |
| | | | | |
| 旧残高 | 新残高 | | 借方合計 | 貸方合計 |
| 得意先コード | 得意先名 | | | |
| 旧残高 | 新残高 | | | |
| 得意先コード | 得意先名 | | | |

（図10）

◦インプット・ファイル・フォーマット

1本のインプット・ファイルに，得意先レコード（マスター）と，その後に続く変動レコード（トランザクション）が含まれているとする。

・得意先レコードのフォーマット

| 得意先コード | 得意先名 | 旧残高 |
|---|---|---|

・変動レコードのフォーマット

| 得意先コード | 伝票NO. | 金　額 | 区分コード |
|---|---|---|---|

（図11）

区分コードは，金額を「借方」と「貸方」に分ける2通りの値からなるとする。

◦アウトプット・データの構造図

```
{アウトプット一覧表 |
  01  Ω {得意先}                    <------ 一覧表レベル
    02  得意先コード
    02  得意先名
    02  △ {変動データ}
      03  Ω {変動ライン}  <------ 得意先レベル
        04  伝票NO.
        04  △ {借方}
                ⊕
            △ {貸方}
      03  借方合計
      03  貸方合計
    02  旧残高
    02  新残高
}
```

(図12)

(注) 7．02 △ {変動データ} は，変動レコードが繰返し現われたり，全く現われない場合を表わす。

8．04 △ {借方} は，借方が現われたり，現われなかったりすることを表す。

9．04 △ {借方}
⊕
△ {貸方}
は，借方，貸方が排他的であることを表わす。

◦インプット・データの構造図

```
{インプット・ファイル |
 01  Ω {得意先}              <------ ファイル・レベル
   02    得意先コード
   02    得意先名
   02    旧残高
   02    △ {変動データ}
     03  Ω {変動レコード} <------ 得意先レベル
       04    得意先コード
       04    伝票NO.
       04    金額
       04  △ {コード, VALUE ▼借方▼}
}
```

(図13)

(注) 10. △ {コード, VALUE▼借方▼} は, コードの値が▼借方▼であったり▼$\overline{借方}$▼=▼貸方▼であったりすることを表わす。

◦プログラムの構造図

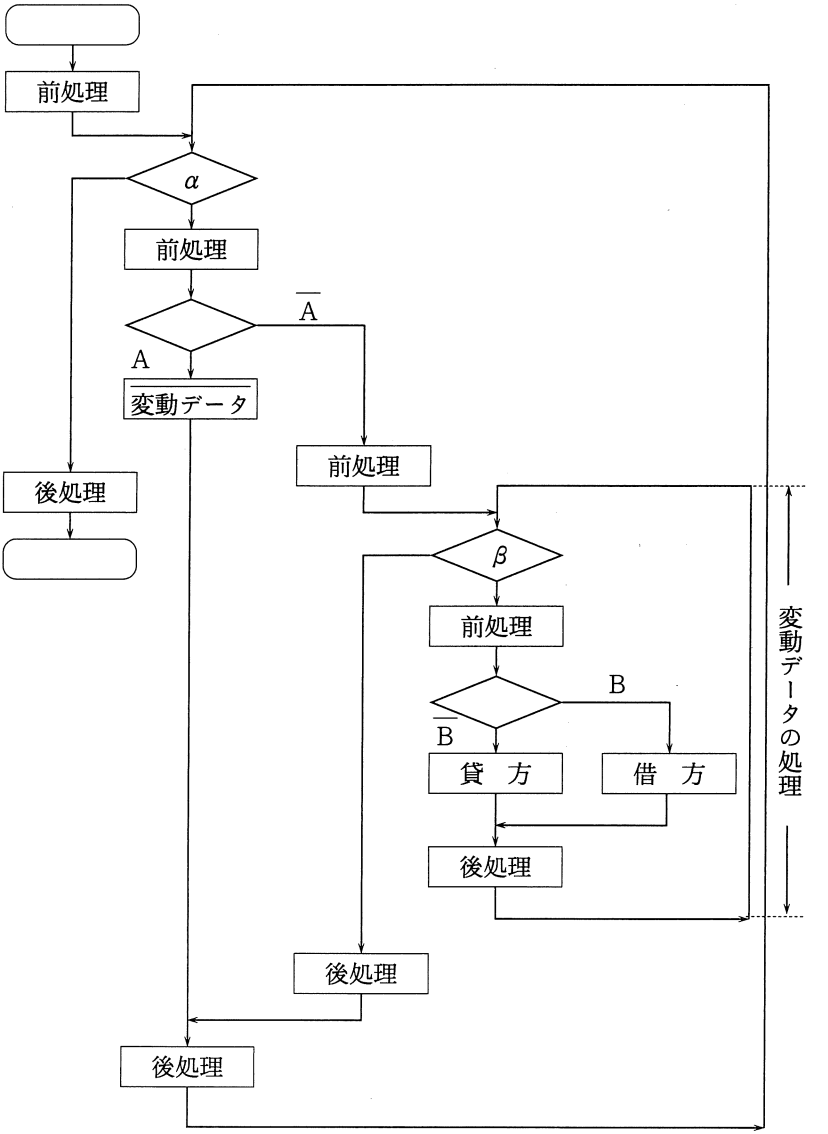
```
π {プログラム |
   01   {前処理}
   01   Ω {α,  {得意先} }
     02   {前処理}
     02   △ {A,  {変動データ} }
          △ {A, }
       03   {前処理}
       03   Ω {β, }
         04   {前処理}
         04   △ {B,  {借方} }
              △ {B,  {貸方} }
         04   {後処理}
       03   {後処理}
     02   {後処理}
   01   {後処理}
 }
```

(図14)

(注) 11. 02 { $\overline{A}$, } は, 条件結果$\overline{A}$のとき, 03レベルの変動データ処理を行う。

(注) 12. 03Ω { β, } は, 条件βを満足するまで, 04レベルの処理を行う。

◦プログラムの流れ図




（図15）

## 4．プログラム構造の最適化を求める

インプット・データ集合の構造図から，プログラム構造図を作成していくとき，最適化のためのアプローチとして，以下のステップを考える。

① 真理値表を作成する。

② この真理値表から，論理式（処理条件）を作成し，簡素化して（ブール代数演算により），論理式を単純化する。

③ 簡素化された論理式を処理別に記述する。

④ 各論理式内の変数（インプット・データ集合）を現われる頻度に応じて並びかえる。このときの結果（論理式全体）が「最適化」の状態である。

その最適化の状態は，次のように3つ考えられる。

i） ツリー構造の状態で最適な場合

「簡素化された論理式のすべてに共通して含まれる変数が，少なくとも1個あるとき」である。

（真理値表の各処理は，互いに「交わらない」または「含まれる」関係にあるインプット・データ集合を処理対象にしているとき）

ii） 分岐構造の状態で最適な場合

「簡素化された論理式のすべてに共通して含まれる変数がないとき」である。

（真理値表の各処理は互いに「交わり」かつ「含まれない」関係にあるインプット・データ集合を処理対象にしているとき）

iii） i）およびii）の組合せの状態（組合せ構造）で最適な場合

⑤ ④の結果から，頻度の高い変数から順次，構造図を書いていく。

以下に最適化アプローチの例を示す。

＜例3＞

「互いに排他的でない」関係の部分集合A，B，Cを含むインプット・データ集合Dと，5種類の処理V，W，X，Y，Zからなる真理値表があるとする。

真理値表

| A | D B | C | V | W | X | Y | Z |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | × | | | | |
| 0 | 0 | 1 | × | | × | | |
| 0 | 1 | 0 | | | | × | |
| 0 | 1 | 1 | | | × | × | |
| 1 | 0 | 0 | | × | | | |
| 1 | 0 | 1 | | × | × | | × |
| 1 | 1 | 0 | | × | | | |
| 1 | 1 | 1 | | × | × | | × |

（図16）

簡素化の例

・論理式　$V=\overline{A}\cdot\overline{B}\cdot\overline{C}+\overline{A}\cdot\overline{B}\cdot C$

・簡素化　$=\overline{A}\cdot\overline{B}\cdot(\overline{C}+C)$

〃　$=\overline{A}\cdot\overline{B}\cdot(1)$

論理式を簡素化し，並びかえをした結果（論理式全体）は，

・$V=\overline{A}\cdot\overline{B}$

・$Y=\overline{A}\cdot B$

・$W=A$

・$Z=A\cdot C$

・$X=C$

となる。

V，Y，Wは交わらない互いに独立した集合であり，ZはWに含まれる集合である。したがってこの部分を「ツリー構造」とする。Xについては「分岐構造」をとらせる。したがってこれは，「組合せ構造」の状態で最適化となる例である。

このときの3つの状態 i ） ii ） iii ）のプログラムの構造図は，各々次のように書ける。

「組合せ構造」の図

```
π {プログラム |
    01   {前処理}
    01  △{Ā, }
      02   {前処理}
      02  △{B̄,  {処理V} }
          △{B,  {処理Y} }
      02   {後処理}
  △{A, }
      02   {前処理}
      02   {処理W}
      02  △{C̄, }
          △{C,  {処理Z} }
      02   {後処理}
    01   {中間処理}
    01  △{C̄, }
        △{C,  {処理X} }
    01   {後処理}
  }
```

ツリー構造

分岐構造

（図17）

（注）13. 01△{Ā, } は，条件結果がĀのとき，直後の02レベルの処理を行う。
（注）14. 02△{C̄, } は，条件結果がC̄のとき，直後の03レベルの処理がないため無処理とする。

「ツリー構造」で解いた場合

「ツリー構造」の図

```
π {プログラム |
    01   {前処理}
    01  △ {A̅, }
      02   {前処理}
      02  △ {B̅, }
            03   {前処理}
            03   {処理V}
            03  △ {C̅, }
                △ {C,  {処理X} }
            03   {後処理}
          △ {B, }
            03   {前処理}
            03   {処理Y}
            03  △ {C̅, }
                △ {C,  {処理X} }
            03   {後処理}
      02   {後処理}
          △ {A, }
            02   {前処理}
            02   {処理W}
            02  △ {C̅, }
                △ {C,  {処理Z} }
                       {処理X} }
            02   {後処理}
    01   {後処理}
}
```

（図18）

さらに，「分岐構造」で解いた場合

「分岐構造」の図

```
π {プログラム |
    01   {前処理}
    01  △ {¬A・¬B,  {処理V} }
        △ {¬¬A・¬¬B, }
    01   {中間処理}
    01  △ {¬A・B,  {処理Y} }
        △ {¬¬A・B, }
    01   {中間処理}
    01  △ {A,  {処理W} }
        △ {¬A, }
    01   {中間処理}
    01  △ {A・C,  {処理Z} }
        △ {¬(A・C), }
    01   {中間処理}
    01  △ {C,  {処理X} }
        △ {¬C, }
    01   {後処理}
 }
```

（図19）

以上の３つの構造図から，「分岐命令△」の数のみに注目して比較するならば，（実際は他の要因も考慮する必要がある。）

- 組合せ構造　　4
- ツリー構造　　5
- 分岐構造　　8

したがって，この例は組合せ構造がプログラム構造として最適であること

が分かる。

実際のところ，上述④の論理式の並びかえ後の状態ですでに最適化構造の解は出ているのであるから，３つの構造図を作って比較してみる必要はなく，簡素化された論理式から最初に作成される構造図が「最適化されたプログラム構造」として求められるのである。

おわりに，今後の課題として，プログラム構造図を「２次元プログラミング言語」と考えて，その理論的基礎を探る方法を見つければ，コンピュータ技術の１つの主要なテーマである，

- 自動フローチャーティング
- 自動フローチャーティングにリンクして特定コンピュータのオブジェクト・プログラムの生成

等の開発も十分可能となり，成果は実り多いものと予想される。まだこの分野は未開拓なところでもあり，興味多きところである。

**参考文献**

(1) 遠山啓「現代数学対話」（岩波新書）1967. 5
(2) 赤摂也「集合論入門」（培風館）1959. 7
(3) シャーマン（関根智明訳）
「電子計算プログラミングとコーデング」（竹内書店）1964. 9
(4) Ｊ・Ｄ・ワーニエ（鈴木君子訳）
「ワーニエ・プログラミング法則集」（日本能率協会）1975. 7
(5) 岸田孝一・三田守久「フローチャート」（日本生産性本部）1973. 6
(6) アドラー（宮本敏雄訳）「新しい数学」（ダイヤモンド社）1966. 1