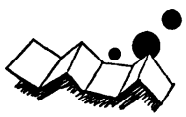
解　説



データ構造指向の設計技法†

紫合　治††

1. はじめに

ソフトウェアの設計を進めるとき，そこで扱う入出力データや内部データの設定の仕方によって，システムの構造が大きく異なってくる．また，事務処理などでは，画面，帳票，ファイル，データベースなどのデータを設計することが設計者の重要な作業であり，一旦それらのデータが定まると，プログラム作成は比較的簡単な作業になることも多い．このように，データはソフトウェア設計の中核をなす重要な要素になる．

1970年代に盛んに研究されたソフトウェア設計方法論は，当初は構造化プログラミング，段階的詳細化など，機能や制御の適切な定め方が中心となっていたが，その後，データ抽象化，データ構造化指向技法，データフロー指向技法など，データのとらえ方が研究の重要なテーマになってきている．最近のソフトウェア工学分野での話題の一つとして盛んに研究されているオブジェクト指向技法も，データ抽象化やデータフロー技法の延長線上にあるととらえることもできる[1]．

本稿では，「データ構造」特集の一環として，データ構造指向のソフトウェア設計技法を解説する．ここでは，データ構造指向の代表的な技法としてジャクソン法（JSP：ジャクソン構造化プログラミング法）[2]を中心に述べる．JSPはすでに10年の歴史があり，本学会誌にも解説がでている[3]．本稿の内容も，それと重複するかもしれないが，できるだけ「データ構造」の視点から解説を試みる．ジャクソン法については，JSPを含む技法として最近JSD（ジャクソンシステム開発法）[4]が提案され注目されているが，「データ構造」指向という見方からはJSDは広すぎるので，ここではJSPに限る．また，データ構造指向技法としては，ジャクソン法の他にワーニエ法などもあるが，データ構造指向の基本的な考え方はJSPに含まれていると思われるので，それらの説明は省略した．

2. データ構造指向技法とは

1970年代の初め，良いプログラムの作成技法として「構造化プログラミング」が提唱され，その後ソフトウェア設計技法やツールが盛んに研究され出した．構造化プログラミングの基本的な考え方は，プログラムの制御構造の規律化（順次，選択，繰り返しの三基本構造の組み合わせでプログラムを記述）と機能の段階的詳細化を通して，理解しやすい，誤りのないプログラムを作成するというものである．

構造化プログラミングの原理は，比較的分かりやすく，効果も大きいので，現在も広く使われている．しかし，プログラムをどのように構造化するかは設計者にまかされており，良い構造にすることは依然として設計者の経験と勘に頼るしかない．では，良い構造とはなにか？　実行速度やメモリ効率などは定量的に測定できるが，ここでの良い構造には，理解しやすいとか，修正しやすいなど，定性的な性質も含まれる．以下に，これらの定性的な性質について検討してみる．

プログラムを理解するとは，もとの問題の記述（そのプログラムが果たすべき機能の記述）とプログラムリストをもとに，問題を理解して，プログラムがその問題の実現になっていることを確かめることである．このように，プログラムを読みながら，プログラムのある部分が問題のある部分を実現していることを確かめる作業が，プログラムの理解であると考えられる．これは，プログラムの誤りを見つける作業（レビュ）でもある．この場合，プログラムの部分と問題の部分の対応が付きやすいこと，すなわち，プログラムの構造が問題の構造を素直に反映していることが，プログラムの理解しやすさ，誤りの見つけやすさの重要な要因になる．

一方，プログラムが修正しやすいとは，どんな変更も簡単にできることではなく，もとの問題の小さな変

† Data Structure Oriented Program Design Method by Osamu SHIGO (Software Product Engineering Laboratory, NEC Corporation).

†† 日本電気(株)ソフトウェア生産技術研究所

更に対しては，対応するプログラムの変更も少なくてすむということである．問題の変更量と対応するプログラムの変更量がほぼ比例することが，修正容易性の条件といえる．したがって，プログラムの構造が問題の構造を反映していることが，修正しやすさにとっても重要になる．

このように，「良い構造」の重要な条件として，プログラムが問題の構造を反映していることが挙げられる．問題の構造，またはプログラムの機能の表現の仕方は，問題を説明する背景によって注目される側面が変わってくるが，多くの場合，そこで扱うデータによって決まる．たとえば，事務処理システムでは，画面，帳票，ファイル，データベースなどによって，機能の説明がなされることが多い．また，システムプログラムでも，コンパイラはソースとオブジェクトによって，リンカはオブジェクトやリンク指示とロードモジュール形式によって，その機能が説明される．すなわち，問題の構造は，一般にそこで扱われるデータの構造に反映されることが多い．したがって，「良い構造」のプログラムはデータの構造を反映している必要がある．これが，データ構造指向技法の基本的な考え方である．

3. 問題の構造とデータ構造

データ構造指向技法では，問題の構造を適切に反映したデータ構造を設定することが，良い構造のプログラムを得る第一歩になる．後で述べるように，一旦データ構造が定まれば，プログラム構造はそのデータ構造に沿って系統的に定めることができる．したがって，問題の表現から適切なデータ構造を選択することが，データ構造指向技法の最も重要なステップであるといえる．

ここで扱うデータは，なんらかの要素の集合とする．データ構造とは，データ集合の要素の配置の順序，要素間の関係の付け方の規則を定めたものである．これらのデータ集合は，データベースやポインタによるリスト構造などの蓄積型データ（空間配置型データ）と，端末やプリンタのようなストリーム型データ（時間配置型データ）に分けられる．蓄積型データも，局所的には，ストリーム型と同様に順次アクセスを行うことが多い．データ構造指向技法では，順次アクセスのデータの構造を主に扱う．以下，蓄積型データとストリーム型データにつき，問題構造を反映したデータ構造について述べる．

3.1 問題構造と蓄積型データの構造

蓄積型データは，システムの内部データになることが多いが，もとの問題の背景になる外界のモデルをなんらかの形で反映している．例として，山崎の共通問題（倉庫管理問題）[5] を考える．これは，酒類販売会社の倉庫管理の問題である．倉庫には複数銘柄の酒ビンを混載したコンテナが搬入される．受付係は顧客からの注文（銘柄と本数の指示）を受け，どのコンテナからどれだけ搬出するかを記した出庫指示と，在庫不足の場合の記録を作成する．問題は受付係の仕事を自動化するシステムを設計することである．

問題の外界モデルとして，種々の酒ビンの入ったコンテナの集合からなる倉庫があり，それに対応して，たとえば図-1 のようなコンテナ管理表（蓄積型データ）が設定される．図-1 の構造は，コンテナの入庫処理やコンテナごとに本数の合計を求めるような問題に向いている．

一方，出庫処理では，銘柄と本数を指示されて，出庫指示（指定銘柄の酒をどのコンテナからなん本ずつ出すか）や在庫不足リスト（指定銘柄の酒の在庫不足による未出荷本数）を出力する．このとき，図-1 のデータ構造での処理を考えると，コンテナを順に見ながらその中身を調べ，指定銘柄の酒があれば必要なだけ取り出すという処理になる．この場合，図-1 のデータを並べ変えて，図-2 のように銘柄を第一キーとした

コンテナ番号	銘　柄	本　数
C 1	H A	20
	H B	15
	H C	8
C 2	H B	32
	H D	3
……	……	……

図-1 コンテナ管理表

銘　柄	コンテナ番号	本　数
H A	C 1	20
	C 3	28
H B	C 1	15
	C 2	32
	C 4	16
……	……	……

図-2 銘柄管理表

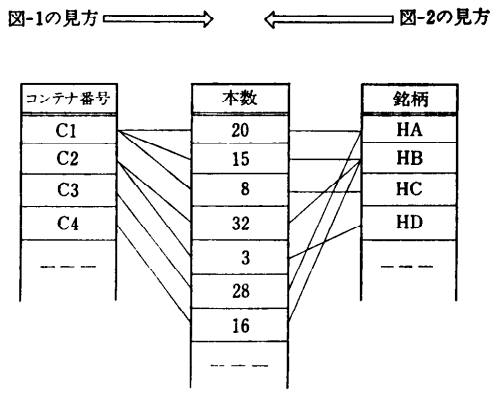


図-3　複数の見方のできるデータ構造

データ構造（銘柄管理表）にすると，まず指定銘柄を見つけて，必要なだけまとめて取り出すことができ，効率がよくなる．さらに，出力である出庫指示のデータ構造（銘柄につき，コンテナ，本数が繰り返す）にも合っており，図-1 よりも図-2 のほうが出庫処理問題を反映しているといえる．

もとの共通問題では，出庫指示にコンテナが空かどうかの印を付けることになっている．コンテナが空かどうかは，図-1 のデータ構造のほうが判定しやすい．また，入庫処理も図-1 が適している．したがって，この問題では，図-3 のように，図-1 と図-2 のふたつの見方のできるデータ構造が望まれる．図-3 のデータ構造を使えば，出庫処理ではまず銘柄管理表の見方で指定銘柄を見つけコンテナから必要な本数だけ取り出し（データのアップデートも行う），同時にコンテナ管理表の見方でそのコンテナが空かどうかを判定すればよい．

一つの蓄積型データに対して，なん通りもの見方をしたい場合，次のような解決策が使われる．

（1）　それぞれの見方ごとにデータを用意しておき，一カ所の変更のごとにすべてのデータを書き直す．

（2）　一つの基本的なデータだけ用意しておき，別の見方をしたいときはそのつど作成する．

（3）　なん通りものデータ構造が表現できるように，データ要素間に関係を付ける（データベース，ポインタによる構造付けなど）．

これらは，場合によって組み合わせて使われる．

このように，一見複雑そうにみえるデータ構造も，比較的単純なデータ構造をいくつか合わせたものと考えることができる．言い換えると，問題をみて，図-1 や図-2 のような単純なデータ構造をいくつか設定し，それらを合わせ持った図-3 のようなデータ構造を考えればよい．

3.2　問題構造とストリーム型データの構造

ストリーム型のデータは，システムの外部データとして現れることが多く，その仕様も問題の記述に含まれていることが多い．蓄積型データでは，データベースやポインタによる構造など，データの入れ物の構造が問題の構造を反映しているが，ストリーム型では，入れ物の構造は単に順次列データであり，その列の要素の並び方の規則（文法）が重要になってくる．すなわち，ストリーム型データの場合，言語の文法に対応する構造を，データ構造としてとらえる．

たとえば，図-2 のデータが図-4 のような形式でストリーム型データに入っているものとする．図-4 で，銘柄，コンテナ番号，本数は，それぞれ，M, C, N文

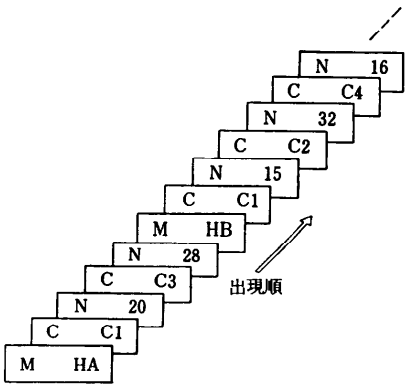
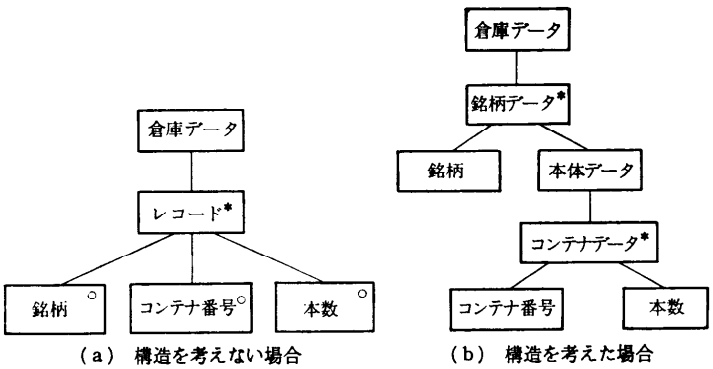


図-4　図-2 のストリーム型データによる表現例



（a）　構造を考えない場合　　（b）　構造を考えた場合

図-5　JSP によるデータの木構造図
（図中，○は選択を，＊は繰り返しを表す）

字を先頭に持つレコードに書かれているとする．通常のプログラム言語（COBOL など）ではレコードの形式は宣言文に書くが，その並び方（現れ方）はプログラム中には明示しない．このため，図-4 のデータ構造を「銘柄，コンテナ番号，本数のいずれかのレコードの繰り返し」ととらえがちである．これは，たとえば倉庫内の酒ビンの本数の合計を求めるような問題には合っているが，銘柄ごとの本数の合計を求める問題に対しては，「倉庫データは，銘柄データの繰り返しで，銘柄データは一つの銘柄レコードの後に0個以上のコンテナデータが繰り返す．またコンテナデータは一つのコンテナレコードの後に一つの本数レコードがくる」というような構造を知らないと処理できない．JSP ではこのようなデータの構造を図-5 のような木構造図で表す．

さらに出庫処理まで考えると，図-5(b)のデータ構造は，出庫指示銘柄でないデータが0個以上繰り返した後，出庫指示銘柄のデータが（もしあれば）一つ来て，その後出庫指示銘柄でないデータが0個以上繰り返すととらえたほうが情報が多くなる．図-6 に，その木構造図を示す．図-5(b)と図-6 のデータ構造は，文法としては同じ言語を表している．しかし，処理されるデータ値の性質まで考慮すると，図-6 のほうがより正確であるといえる．このように，データ構造指向技法では，できるだけ問題の意味を多くとらえて，それを文法として表現することが重要である．

4. データ構造とプログラム構造

データ構造指向技法では，前節で述べたデータ構造にしたがってプログラム構造を定める．データ構造は，木構造図で表されるように，構造化プログラミングの三基本構造と同じく，順次，選択，繰り返しの組み合わせになるので，それに合わせたプログラムも三基本構造の組み合わせで表現される．

4.1 入力データと出力データの対応とプログラム構造

プログラム構造の設定では，まず入力データと出力データの構造の対応を調べ，その対応に従ってそれらのデータ構造を重ね合わせた構造を定め，それをもと

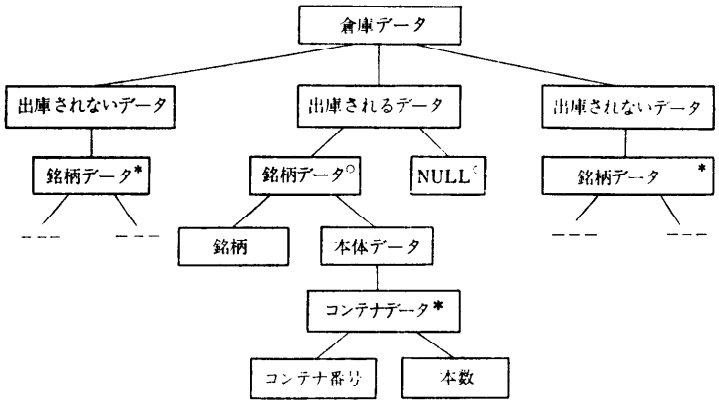
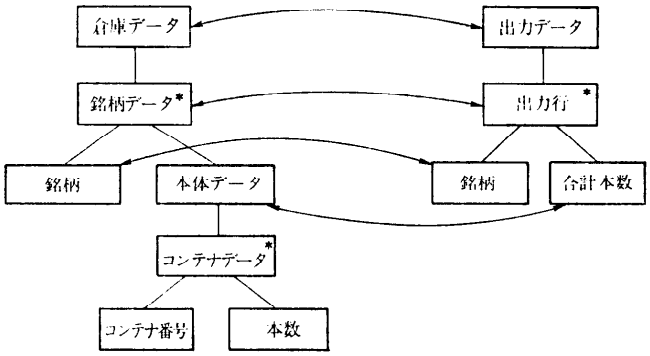
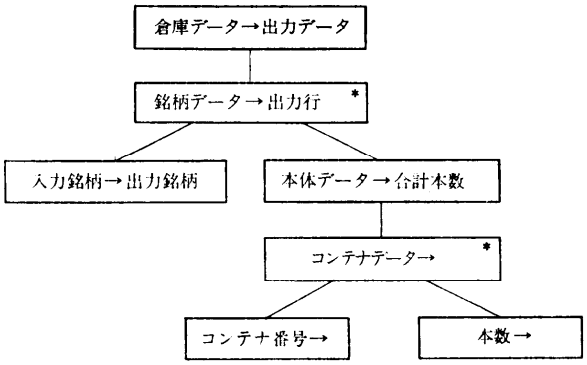


図-6 出庫処理を考慮したデータ構造図



（a） 入力と出力の対応関係



（b） 入力と出力を重ね合わせた構造

図-7 入力と出力の構造の対応と重ね合わせ

```
入力データと出力データのオープン
1レコード入力.
繰り返し 入力が銘柄レコードの間
    銘柄を出力
    合計=0
    1レコード入力
    繰り返し 入力がコンテナ番号レコードの間
        1レコード入力
        もし 入力が本数レコードなら
            合計=合計+本数
        そうでなければ
            エラーメッセージ出力
        1レコード入力
    合計を出力
入力データと出力データのクローズ
```

図-8 図-7(b)の構造に従ったプログラム

にプログラム構造を定める．ストリーム型データの場合は，3.2 で述べた文法構造の対応関係を調べる．たとえば，図-5(b)のデータ構造を入力して，銘柄ごとに本数の合計を出力する問題を考える．出力は，銘柄と合計本数の対の繰り返しで，入力の銘柄データごとに，出力の一対が対応する．このような，入力と出力の対応を図-7(a)のように表す．この対応をもとに，対応関係にあるノードを重ねて，図-7(b)の構造が得られる．この重ね合わせた構造は，入力に関するノードのみに注目すると入力と同じ構造になり，出力に関するノードのみに注目すると出力と同じ構造になる．

```
SW 1=SW 2=NO
入力データと出力データのオープン
1レコード入力
繰り返し 入力がある間
    選択 入力レコードの種類で
    場合 銘柄レコード
        もし SW 1が NO なら
            SW 1=YES
        そうでなければ
            合計を出力
        合計=0
        銘柄を出力
        SW 2=YES
    場合 コンテナ番号レコード
        なにもしない
    場合 本数レコード
        合計=合計+本数
もし SW 2 が YES なら
    合計を出力
入力データと出力データのクローズ
```

図-9 分かりにくいプログラムの例

このようにして得られた図-7(b)の構造をプログラムの構造とする．最終的なプログラムは，処理に必要な命令文（入力文，出力文，合計=0，合計=合計+本数など）を，その構造に適切に配置することによって求まる．図-8 に，結果のプログラムを示す．

同じ問題を，問題に合わないデータ構造，たとえば図-5(a)の構造に合わせると，図-9 のように図-8 に

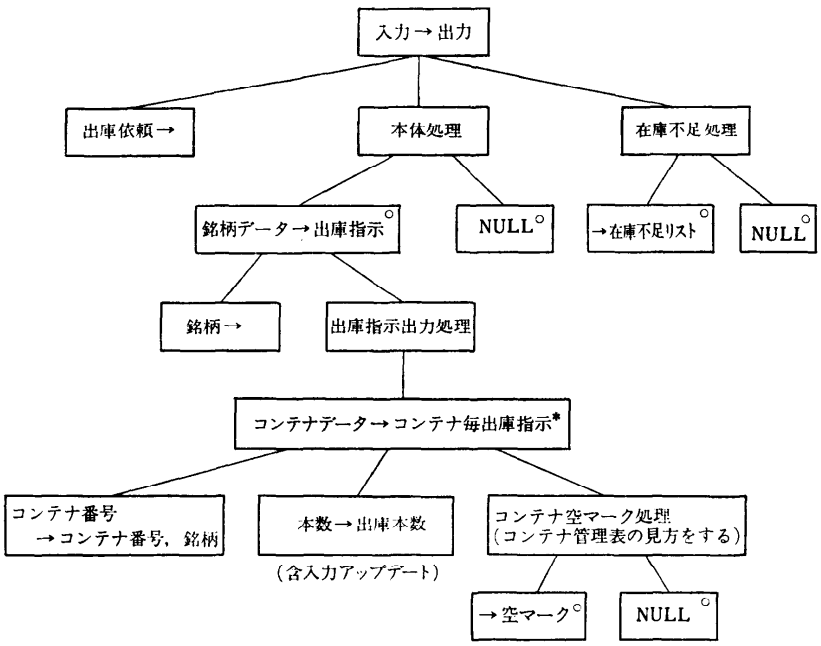


図-10 図-3 のデータ構造による出庫処理のプログラム構造

```
ファイルオープン
入力　出庫依頼（指定銘柄，出庫本数）
残本数＝出庫本数
入力　銘柄管理表の銘柄データで　銘柄＝指定銘柄のもの
もし　見つかったなら
    入力　銘柄データのコンテナデータ（コンテナ番号，
          本数）
    繰り返し　入力終りでなく，かつ残本数≠0
        出力　コンテナ番号，指定銘柄
        もし　残本数≧本数　なら
            コンテナ出庫数＝本数
        そうでなければ
            コンテナ出庫数＝残本数
        残本数＝残本数－コンテナ出庫数
        出力　コンテナ出庫数
        変更　銘柄データのコンテナデータの本数＝
              本数－コンテナ出庫数
        もし　コンテナ管理表の見方でコンテナ番号の
              コンテナが空なら
            出力　空マーク
        入力　銘柄データのコンテナデータ（コンテナ
              番号，本数）
もし　残本数≠0　なら
    出力　在庫不足リストとして，指定品名と残本数
ファイルクローズ
```

図-11　出庫処理のプログラム例

比べてかなり分かりにくいプログラムになってしまう．図-8 では，同一銘柄の本数の合計を求める処理がテキスト上にもまとまっているが，図-9 では分散してしまっている．これは，前に述べた理解しやすさの規準から外れているといえる．

一方蓄積型データを扱う場合も，プログラム構造を定めるには時間的順序が必要になるので，順次アクセスをする部分をもとにプログラム構造を設定する．たとえば共通問題の出庫処理で，図-3 のデータと出庫依頼（銘柄と出庫本数）を入力して，出庫指示と在庫不足リストを出力する問題を考える．この場合，3.1 で述べたように，まず図-2 の銘柄管理表の見方で指定品名を見つけ，その中でコンテナデータを順次アクセスしていく．したがってそのデータ構造は図-6 と同様になる．ただし，蓄積型データではキー項目による直接指定もできるので，図-6 の中で出庫されないデータの部分はプログラム構造には反映されない．これと，出庫依頼，コンテナ管理表，出力データ（出庫指示と在庫不足リスト）を重ね合わせると，図-10 のようなプログラム構造が得られる．図-10 をもとにしたプログラムの例を図-11 に示す．

4.2　データ構造の対応がつかない場合

入出力のデータ構造の間に上で述べたような対応が付かないと，それらを重ね合わせた構造を設定することができない．JSP では，このような構造の不一致として，順序不一致（対応するデータ要素の順序が異なる場合），境界不一致（データの木構造図の間で，上位のノード間と下位のノード間には対応があるが，中間のノード間に対応が見出せない場合），入り交じり不一致（論理的にまとまるべきデータ要素が入り交じっている場合）に分けて，それぞれの解決法を示している．

JSP による構造不一致の解決策は，基本的にはデータ構造間の対応がつく中間データを新に導入する方式である．データ構造Aとデータ構造Bとの対応がつかないとすると，Bとの対応がつく A' をAから作成し，A' とBとの対応をとってそれらを重ね合わせた構造を設定する．ここで，中間データ A' は，もとのデータAの見方を変えたものであり，3.1 で述べた，一つのデータで複数の見方をしたい場合に相当する．すなわち，中間データ A' の導入は，データBと関係した問題（AからBを出力など）にとってより適切な見方を新に追加することである．このようにして，もとの問題は，(1)もとのデータと中間データの変換の問題と，(2)中間データを使った問題に分かれる．後者はこれまで述べた JSP の技法がそのまま使える．

JSP では，ストリーム型データを扱っているので，(1)と(2)の処理の結合は，ストリームを介した並行プロセスでモデル化される．このモデルを効率的に実現するため，プロセスをサブルーチンに変換する技法

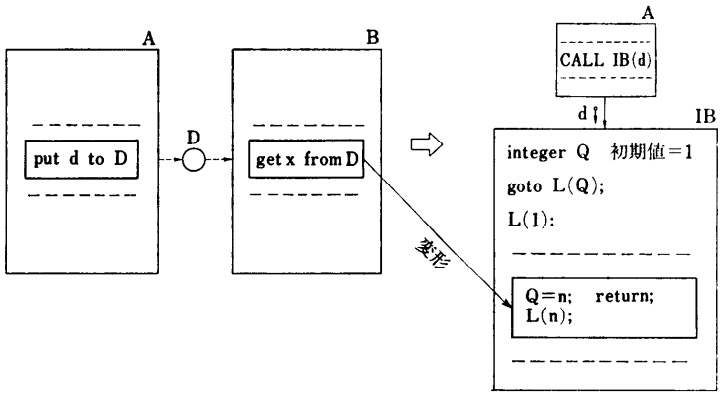


図-12　プログラムインバージョン

(プログラムインバージョン) が使える．これは，サブルーチン内に，次に実行再開すべき場所を覚えておく内部変数を持って，コルーチンの制御を模擬する技法である (図-12)．

構造不一致の解決は JSP の重要な部分であり，他の設計技法にない具体的な提案がなされている．本稿では，データ構造に関する技法に限るため，これらの説明は省略するが，詳細は参考文献 2)，4) を参照されたい．

5. おわりに

データ構造指向の設計技法は，従来事務処理分野を中心に適用されてきたが，データ構造に合わせてプログラムの構造を定めるという方式は，どんな分野にも適用できるといえる．本稿ではジャクソン法 (JSP) を中心に，データ構造指向の考え方を述べたが，JSP のすべてを紹介したわけではない．JSP についての詳細は，文献[2),3)] を参照されたい．

ここでは触れなかったが，データ構造としては再帰的構造も考えられる．その場合も，再帰的構造のデータを扱うプログラムは再帰的構造にするのが最も分かりやすいといえる．ここでは，データ構造のとらえ方，見方が重要になる．リスト構造のデータを，要素の繰り返しと見る人は繰り返しプログラムが，再帰的と見る人は再帰的プログラムが理解しやすいであろう．データ構造のとらえ方，見方を重視することは，データ抽象化[6)] の考えにも一致する．3.1 で述べた複数の見方ができるデータ構造を実現する場合，それぞれの見方に対してのオペレーション群を持った抽象データ (または，抽象データ型) が使える．この抽象データは，オペレーション全体としては複雑で理解しにくいが，それぞれの見方のオペレーション群ごとに見ると，比較的簡単なデータ構造の集まりとして理解しやすくなる．

データ構造指向技法は，問題の分析から始まるので，解法指向ではなく，課題指向であるといえる．しかし，一般には，問題の分析の過程で，解法の概要も並行して思い浮かべることが多い．3 章で述べた例題に対するデータ構造も，このような解法の概要を考えながら設定したものである．しかしながら，この解法の概要は，詳細な実現方式までは意識しないで，問題分析で得られたモデル上での解法である．このように，最終的な実現方式とは独立に，問題分析で得られたモデル上での解法を設計する方式は「オペレーショナルアプローチ」[7)] と呼ばれ，従来のウォータフォール型ライフサイクルモデルに替わる方式として最近注目されている．データ構造指向技法も，今後は JSD やオペレーショナルアプローチなどのより広い技法の一部として，普及していくであろう．

参考文献

1) 紫合 治：ソフトウェア設計法について，コンピュータソフトウェア，Vol. 1, No. 2, pp. 55-68 (1984).
2) Jackson, M. A.: Principles of Program Design, Academic Press, London (1975).
鳥居訳：構造的プログラム設計の原理，日本コンピュータ協会 (1980).
3) 峰尾欽二：プログラミング方法論 (ジャクソン法)，情報処理，Vol. 23, No. 11, pp. 1063-1074 (1982).
4) Jackson, M. A.: System Development, Prentice-Hall (1983).
5) 山崎利治：共通問題によるプログラム設計技法解説，情報処理，Vol. 25, No. 9, pp. 934 (1984).
6) 久野 靖：データ抽象向けプログラム設計技法，情報処理，Vol. 26, No. 11, pp. 1310-1318 (1985).
7) Zave, P: An Operational Approach to Requirement Specification for Embedded Systems, IEEE Trans. on Software Engineering, Vol. SE-8, No. 3, pp. 250-269 (1982).

(昭和 60 年 11 月 27 日受付)