上級情報処理技術者育成指針

各　　論

第3部　コンピュータおよび情報処理技術

財団法人 日本情報処理開発協会
情報処理研修センター


007828

ま　え　が　き

当財団では，昭和４３年以来，６年にわたり，情報処理育成事業の一環として，初級，中級，上級の「情報処理技術者育成指針」（ただし，上級のみ標題は「情報処理技術者研修ガイドブック」となっている）を作成してきた。このうち上級情報処理技術者育成のための指針については，情報処理技術の著しい進展により，全般的な見直しが必要となってきた。

このため，昭和５１年度から３カ年計画で，これまでに作成した初級，中級，両指針との関連を考慮し，同時に，現在および今後数年間に想定される環境にも調和できるようにとの考え方にもとづいて，新しく「上級情報処理技術者育成指針」（以下「本育成指針」とする）の開発を進めてきたが，これが完成のはこびとなった。

本育成指針は，個々の企業内や，一般の企業人向け教育機関などにおいて，さまざまな形で行なわれている上級情報処理技術者の教育に対して，標準的なカリキュラムを示し，情報処理技術者の養成を効果的に推進することを，そのねらいとしている。

本育成指針の内容は，「総論」および，分野別に編集された各論，すなわち「組織システムの分析」，「システム開発運用の背景」，「コンピュータおよび情報処理技術」，「情報システムの開発」の，合計５分冊から構成されている。この利用に当っては，まず総論を参照し，必要に応じて各論の関連部分を参照されるよう希望したい。

なお，この事業実施にご尽力いただいた関係各位に心から感謝の意を表わすとともに，この指針が各方面で利用され，情報処理技術者育成の一助となることを念願する次第である。

昭和５３年３月

財団法人　日本情報処理開発協会
情報処理研修センター

所長　山　内　二　郎

I

36

~~2~~ 4

上級情報処理技術者育成指針委員会

（敬称略，50音順）

委員長	松田武彦	東京工業大学
委員	安達俊雄	通商産業省
〃	魚木五夫	広島修道大学
〃	江村潤朗	日本アイ・ビー・エム株式会社
〃	岡本行二	東京芝浦電気株式会社
〃	小佐井純正	株式会社三菱銀行
〃	関　収	通商産業省
〃	台　文彦	新日本製鉄株式会社
〃	土居範久	慶応義塾大学
〃	西村恕彦	電子技術総合研究所
〃	服部幸英	日本鋼管株式会社
〃	原田睦明	株式会社ソーシャル・サイエンス・ラボラトリー
〃	前川良博	横浜商科大学
〃	間野浩太郎	青山学院大学
〃	三浦大亮	東レ株式会社
〃	吉野元之助	(財) 日本情報処理開発協会

上級情報処理技術者育成指針作成小委員会

（敬称略，50音順）

委員長	魚木五夫	広島修道大学
委員	岡本行二	東京芝浦電気株式会社
〃	小佐井純正	株式会社三菱銀行
〃	土居範久	慶応義塾大学
〃	服部幸英	日本鋼管株式会社
〃	三浦大亮	東レ株式会社

序

本書は，上級情報処理技術者育成指針（以下「本育成指針」という）のうち，「コンピュータおよび情報処理技術」に関連した分野の教育に有用と考えられる資料類を，整理・編集したものである。本書の内容は，次の４つの単元に分かれている。

単元Ｃ１：情報構造

単元Ｃ２：コンピュータ・システム

単元Ｃ３：ファイルおよびコミュニケーション・システム

単元Ｃ４：ソフトウェア設計

これらの単元は，教育の計画や実施などにおける一つの作業単位であり，本書ではこれをさらに幾つかの章に分けて解説している。

上級情報処理技術者の育成を，実際に計画し，また実施しようとする場合には，本育成指針のうち「総論」をまず参照されたい。総論は，本育成指針利用のための案内書であり，技術者育成の体系的な計画，教育の目標，それに至るための教育内容や教育方法，などに関して方向づけを与えるものである。

本書は，下記の方々の執筆または協力と，本育成指針作成小委員会の監修を得て作成されたものである。

（敬称略，５０音順）

主査	土居範久	（慶応義塾大学）
委員	犬伏茂之	（日本ユニバック株式会社）
〃	今江泰	（日本ユニバック株式会社）
〃	宇都宮公訓	（筑波大学）
〃	大野義夫	（慶応義塾大学）
〃	金田雄次	（株式会社ソーシャル・サイエンス・ラボラトリー）
〃	久保秀士	（日本電気株式会社）
〃	斉藤信男	（筑波大学）
〃	菅田光	（株式会社ソーシャル・サイエンス・ラボラトリー）
〃	箱崎勝也	（日本電気株式会社）
〃	原潔	（日本ユニバック株式会社）
〃	原田賢一	（慶応義塾大学）
〃	原田睦明	（株式会社ソーシャル・サイエンス・ラボラトリー）
〃	藤野喜一	（日本電気株式会社）
〃	穂鷹良介	（筑波大学）
〃	山本喜一	（慶応義塾大学）

第 3 部
コンピュータおよび情報処理技術

総 目 次

単元Ｃ１

情　報　構　造

単元Ｃ１

情　報　構　造

単 元 C 1
情　報　構　造

目　次

第1章　情報に関する基礎概念

キー・ワード

情報(information), データ(data), 情報量(information content), ビット(bit), コード化(coding), バイト(byte), 語(word), 検査ビット(check bit), 誤り訂正符号(error-correcting code), データの型(data type) 数(number), 字(character), 論理(logic), アドレス(address), 名前(name), 値(value), 記憶場所(storage location), 主記憶装置(main storage) 補助記憶装置(auxiliay memory), ブロック(block), ファイル(file), レコード(record), ハッシュ表(hash table), 写像(mapping), 結合(link), 関係(relation), ポインタ(pointer), 論理構造(logical struclure), 物理構造(plhysical structure), 固定小数点表現(fixed point representation), 浮動小数点表現(floating point representation), 補数(complement), 変換(conversion), 丸め(round), 誤差(error), あふれ(overflow), 下位けたあふれ(underflow)

目　　標

第2章以降のために基礎的概念を整理する。人間は，道具としてコンピュータを利用するため，情報を形式化してデータにし，コンピュータに与えて処理させ，その結果を意味づけて利用する。コンピュータ内では，データを効率よく，正しく処理するために，種々の符号化方法，記憶方法を使い分ける。とくに，互いに関連するデータの記憶には高度な手法が開発されている。以上に関して大づかみに理解させ，後続の章で内容を展開するための導入部とする。

内　　容

1.1　情報とデータ

人間が関わる情報をデータにしてコンピュータに与えて処理させ，その結果を加工された情報として利用する。それゆえ，コンピュータはデータ処理機械である，という見方を明確にし徹底させる(図1－1参照)。

データは，「人間，もしくはコンピュータなどの自動的手段によって行なわれる解析，処理，通信に適するように形式化された事象，概念または指令の表現」であり，情報は，「ある表現に

適用される一定の約束にもとづいて人間がデータに割り当てた意味」である。

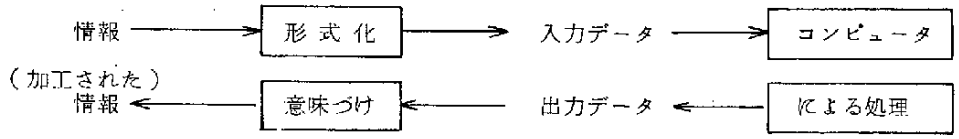


図1－1　情報，データとコンピュータの関係

情報がどのようにデータに形式化されるかをより深く理解させるために，情報の量についてごく簡単にふれるとよい。

たとえば，8進数のある桁が6であるとすると，「0から7までの8つの候補のうち，6である」といっているわけであるから，その情報量は「8分の1」である。一般に，

$$\text{情報量} = -log_2 P \quad 〔ビット〕$$

と定義する。ただし，Pは情報に示された事象が起こる確率である。この場合の情報量は

$$-log_2 \frac{1}{8} = 3 \quad 〔ビット〕$$

である。

したがって，A，B二つの場合のいずれかであることを区別するデータは

$$-log_2 \frac{1}{2} = 1 \quad 〔ビット〕$$

となり，1ビットで済む。

1.2　データの表現

コンピュータは2値（0と1）を基本としてデータを表現するので，コンピュータ中では，データをコード化し，1ビット以上のビット列として表現する。この節では以下の各項目について，理解させる。

(1)　ビット列の長さ

データの表現であるビット列の長さは，本質的に制限がない。ビット列は，最終的にはハードウェアの操作をうける。その際，バイト（byte），語（word）など一定の長さのビット列を単位とした処理を行なうことによって，処理速度の向上などを図っている。

しかし，応用プログラム（application program）などのソフトウエアからは，通常，そのような差を意識しなくてもよいようになっている。

(2)　検査ビット

コンピュータ中で，データの処理や転送が常時正しく行なわれるということは絶対に保証できない。ハードウェアが誤まって動作したときそれを検出し，場合によっては，再試行や訂正

するため，データ・ビットの組に検査ビット（check bit）をつけることがある。奇偶検査（parity check）のための検査ビットは，その一つの例である。他にも，巡回冗長検査（cyclic redundancy check）やハミング・コード（Hamming code）のための検査ビットなどがある。（図1−2参照）。ハミング・コードは，代表的な誤り訂正符号（error correcting code）である。

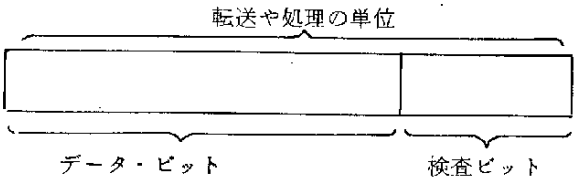


図1−2　検査ビットのつけ方

(3) データの型

コンピュータ処理の対象となるデータは

数　（number）

字　（character）

論理　（logic）

アドレス　（address）

などの型に分類することができる。

このうち，もっとも頻繁に現われる数や字は，共通的なコード化が行なわれている。たとえば，字については，ＪＩＳコードがある。

1.3　データの記憶

データを効率よく利用できるようにするため，データの記憶に種々の工夫がなされている。この節では，以下の項目について理解させる。

(1) 記憶のしくみ

データは記憶装置に記憶される。プログラムでデータを操作するとき

データの名前

データの値

データの記憶場所

を混同しないように注意しなければならない。

(2) 記憶装置

処理装置（processing unit）と強く結びついている主記憶装置（main storage）と，割合低速で容量が大きい，磁気ディスクなど，補助記憶装置（auxiliary storage）

では，ハードウェア的にデータの記憶の単位が異なる。主記憶装置中ではバイト，ワードが単位であるが，補助記憶装置中では，ブロック（block），ファイル（file）が単位である。応用プログラムから見た補助記憶装置に対する入出力の単位であるレコード（record）をいくつかまとめた，ハードウェア的な入出力の単位がブロックである。（図1－3参照）

入力装置は書換えができない，出力装置はそのままではコンピュータに入力できない，特殊な補助記憶装置と考えることができる。

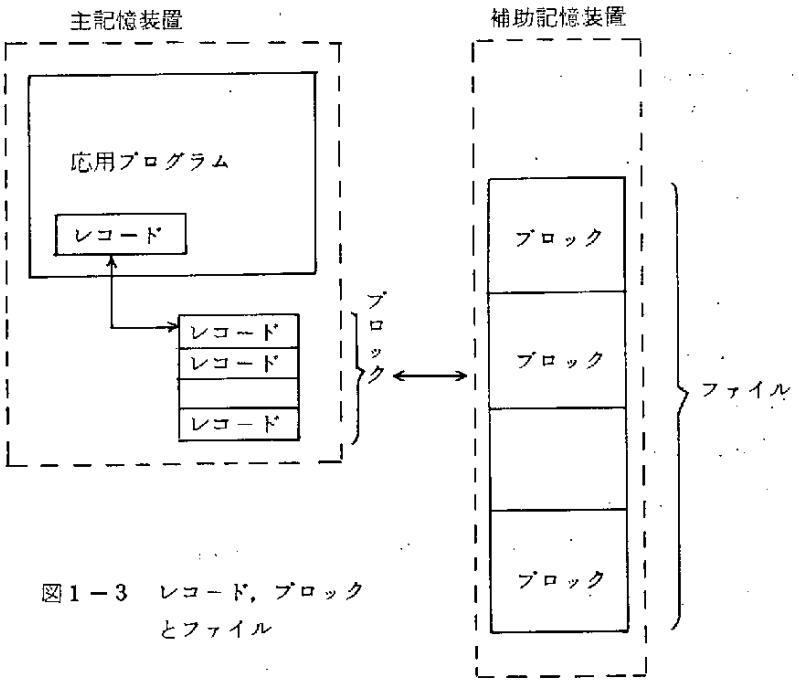


図1－3　レコード，ブロックとファイル

(3)　記憶場所

データを記憶装置のどこに記憶するか，記憶した場所をどのように知るか，は処理速度に強く影響する。装置中の記憶場所は

①　アドレス

②　順　　番

③　アドレスと順番の組合せ

のいずれかで識別する。

値によって関連するデータを捜し出したいときは，値が記入されている表の中の項目を一つ一つ比べるよりも，

①　値を引数とした写像によって記憶場所が一意的に決まるか，

②　一意的に決まらなくても，その場所から始めて少ない比較で捜し出せる

ようにした方が効率がよい。例としては，ハッシュ表（hash table）がある。

1.4 データの結合

いくつかのデータを集めて，それ全体にあらためて名前をつけて一つのデータとして扱ったり，互いに関連するデータを結合しておいて，次々にたぐるよせて使ったりすることがよくある。この節では，以下の項目について理解させる。

(1) データの関係

いくつかのデータを結合して得られたデータを記憶するときは，それらのデータの関係も何らかの物理的方法によって記憶しなければならない。たとえば，

① 全体として1組になっているが，個々のデータ項目は独立である

② ①と同じであるが，データ項目が発生した時間に差がある

③ データ項目の間に，従属関係がある

などの関係を表現する必要が生じる。

①に対しては，データ項目を次々と並べて記憶すればよい。

②に対しては，データ項目が発生した時間的順序を，そのまま並べる順序に反映させて記憶すればよい。

③は簡単ではなく，ポインタ(pointer)を用いた表現などで記憶する。

(2) ポインタ

基本的には，ポインタは従属するデータ項目を示すものであり，コンピュータ中では，データ項目が記憶されているアドレスなど，記憶場所を示す値である。(1)の③の関係をポインタで表わす簡単な例を図1－4に示す。

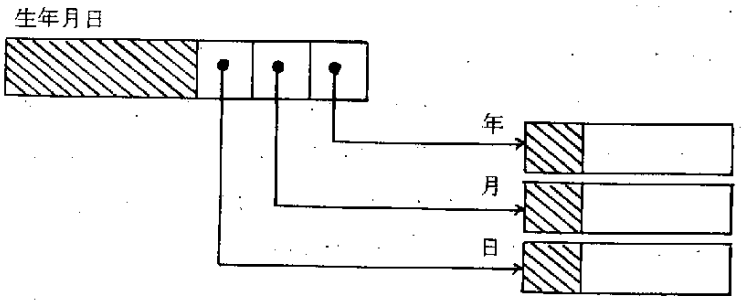


図1－4 関係を表わすポインタの例

また，データの大きな集まりを分割して記憶し，分割されたものをポインタで接続するようにした使い方もできる。

以上のように，ポインタは，データの論理構造をハードウェアの中で表現するための代表的

な手法である。また，このハードウェアの中での表現を物理構造という。

1.5 数

コンピュータ中では，数を処理することがもっとも多い。数をどのように符号化し，処理するかを簡単にまとめる。この節では，以下の項目について理解させる。

(1) 数の表現

コンピュータ内では，数を限られた長さのビット列として表現しなければならない。そのため，次の二つに分けて表現する

① 固定小数点表現

小数点を固定して，数を2進数にしてビット列として表現する。通常，小数点を右端に固定し，整数として表現する。

② 浮動小数点表現

数を指数部と仮数部の組合せとして表現するので，絶対値が大きい数，小さい数の表現に適している。

例 $16.777,216 = 2^{25} \times 0.5$ の表現

2を底とした表現をすれば，指数部は25，小数部は0.5となるので，これらを2進の固定小数点として表現する。

10進の小数を2進小数に変換すると，ほとんど循環小数となるので，ワード境界で打切ったり，丸めたりする。そのため，必ず誤差が生じることに注意する。

符号（正または負）つきの数の表現では，通常，符号のために1ビットをさき，残りのビットで絶対値を表わす。負の数の絶対値の表現に

① 真数による表現

② 補数（complement）による表現

の二つがあり，加算の計算速度が速いという理由から，固定小数点表現では，補数表現を用いることが多い。補数表現は，$-x$（xは正）の絶対値を$m-x$として表現する手法である。ただし，mは記憶できる最大値に1を加えたものである。補数表現されていれば，符号も一緒に単純に加算することによって，符号つきの数の加算ができる。

(2) 桁あふれと桁落ち

数に対する基本的な操作は，加減乗除であるが，これらの計算の結果得られたデータが記憶できない程度に大きかったり，小さかったりすることがある。これらは，それぞれ，あふれ（overflow），下位けたあふれ（underflow）といわれる。

指導上の留意点

基本的な考え方を述べるにとどめ，具体的詳細は後続の展開に委ねること。

参 考 文 献

〔1〕 D.E.Knuth,"The Art of Computer Programming,Vol.1", Fundamental Algorithms",Addison-Wesley,1968

〔2〕 ベルティス（Berztiss A.T.）著，古川康一他訳「データ構造Ⅰ，Ⅱ」日本コンピュータ協会，1974

〔3〕 藤田広一「基礎情報理論」昭晃堂，1972

〔4〕 野崎昭弘「スイッチング理論」共立出版，1972

第2章　論理構造——線型リスト

キーワード

論理構造(logical structure), 物理構造(physical structure), 外部構造(external structure), 内部構造(internal structure), 線型リスト(linear list), スタック(stack), キュー(queue), デキュー(dequeue), (線型リスト, キューの)ヘッド(head), テイル(tail), (スタックの)トップ(top), ボトム(bottom), 深さ(depth), プッシュ・ダウン(push down), ポップ・アップ(pop up), スタック・ポインタ(stack pointer), 先入れ先出し(FIFO; first-in-first-out), 後入れ先出し(LIFO; last-in-first-out), 終端ポインタ(terminal pointer, null), 単純連結(single linkage), 両方向連結(double linkage), 循環連結(circular linkage), 一方向リスト(one-way list), 両方向リスト(two-way list), 前進ポインタ(forward pointer), 後進ポインタ(backward pointer), マジック・リスト(magic list), (リスト要素の) 追加(addition), 挿入(insertion), 削除(deletion), アクセス(access), 文字列(string), 部分文字列(substring), 文字列編集(text editing), 文字列照合(matching), (文字列の) 接続(concatenation),

目　標

コンピュータを使って問題解決をするためには, コンピュータに処理手順(すなわちプログラム)とデータとを与える必要がある。このプログラムもデータもただ"ベッタリ"としたものがあるわけではなく, それぞれがある構造を持っていることが多い。プログラムが持っている構造はループとかプログラム単位への分割, ブロック構造などとして, 自然に外面にあらわれてくる。しかし, データの構造は一意に定まるものではないので, データが本来持っている種々の構造のうち, 当面の問題解決に必要なものだけを取り出すことが必要である。

データの構造を考えるにあたっては, 次の二つの側面を考察する必要がある。

- データが本来持っている構造

  これを論理構造とか外部構造とか呼ぶ

- コンピュータ内でのデータ間の関係の表現

  これを物理構造とか内部構造とか呼ぶ

この章の主題は，論理構造を再認識させることであって，物理構造の扱いは第6章に譲るべきである。しかし，現実には論理構造と物理構造との間には密接な関係がある上，マイクロ・プログラミングの発達，マイクロ・プロセッサの開発による各種装置のインテリジェント化に伴ない，その境界は変動するとともに，明確でなくなってきている。そこで，この章では，論理構造と物理構造の大ざっぱな区別を認識しうるように指導しつつ，物理構造へスムーズに入れるように共通の概念は第2～5章で扱うこととしたい。

特にこの章では，枝わかれのない構造を取り扱う。基礎概念を理解させた後，若干の応用を事例によって示す。

この章の目標をより具体的に示すと次のようになる。

まず，論理構造として典型的な構造のうち，枝わかれのない構造についての基本概念を与える。

つぎに，論理構造としての線型リストを物理構造としてのリストとして表現したときに使われる手法の代表的なものを理解させる。

さらに，各種の線型リストに対する基本的な操作のアルゴリズムを与え，理解させる。

最後に，論理構造のうちの線型リストのまとめとして，線型リストの操作例を示す。これによって，線型リストがどのような場面に応用できるかを理解させるとともに，それらの応用場面で，線型リストに対する操作が具体的にどのような意味を持っているのかを理解させることができる。

内　　容

2.1 基本用語

はじめに，序論として，データにもプログラムと同様に構造があること，その構造のうち，当面の問題解決に必要な構造だけを取り出して扱うことを説明する。また，これに伴なって，論理構造，物理構造の概念を次のように与える。

論理構造　データが本来持っている構造で，当面の問題解決にとって必要ありと認められたもの。外部構造ともいう。

物理構造　コンピュータ内でデータの構造を表現するために作られた構造。内部構造ともいう。

この二つの概念に関連して，次の事を説明しておく必要がある。

① 論理構造と物理構造とは1対1に対応しているものではない。すなわち，たとえば論理構造として木構造が考えられる場合であっても，それを表現するための物理構造として，配列が採用されることもあろうし，2進木リスト，一般の木，ハッシングを利用したランダムな構造などが採用されることもあり得る（図2－1）。

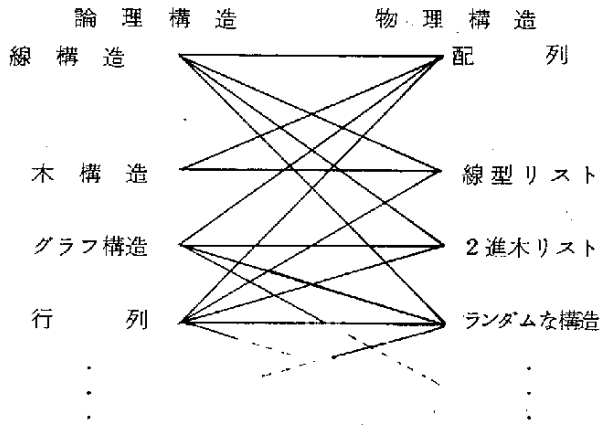


図２－１　論理構造と物理構造の対応

② 物理構造を決めるにあたっては，次のような事を考慮しなければならない（図２－２）。

- どのような論理構造の表現としてその物理構造を用いるのか
- 主記憶におくか外部記憶におくか
- 記憶スペースをたっぷり利用できるのか
- 次のような各種の作業がどのような頻度でどのような緊急度をもって発生するか

  項目の追加

  項目の削除

  項目へのアクセス

  { その項目があるかどうかわからない場合
  { その項目が存在することがあらかじめわかっている場合

  項目内のデータの変更

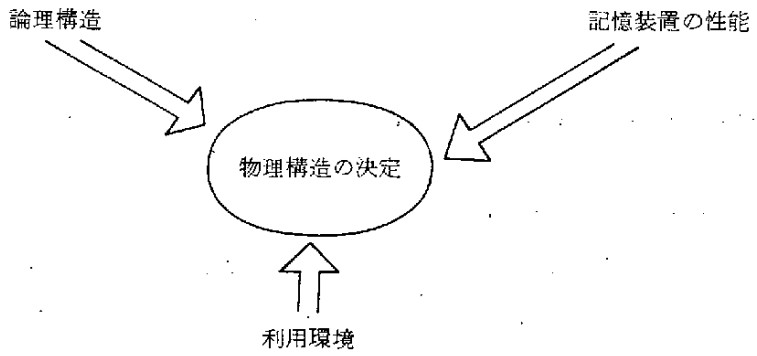
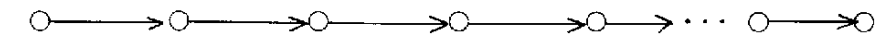


図２－２　物理構造を決定する因子

これらの説明は，論理構造，物理構造の具体例があらわれた段階で折りにふれて，より具体的な形で何度もくり返す必要がある。

次に，論理構造のもっとも簡単なものとして線型リストを導入する。線型リストは次のように定義できよう（図２－３）。

- 「ヘッド」と呼ばれる項目がただ一つある
- 「テイル」と呼ばれる項目がただ一つある
- テイルを除くすべての項目それぞれに対して，「次の項目」がただ一つある
- ヘッドから順に次の項目をたどっていくことにより，すべての項目をたどることができる。



ヘッド　　テイル

線型リストはこのような有向グラフとして表現されることもある。

図２－３　線型リスト

例としては次のようなものを示すのがよいであろう。

① 枝わかれのない鉄道（たとえば新幹線）。このときには，駅の隣接関係だけを取り出しており，駅と駅との間の距離，線路のまがり具合，まわりの風景などのデータをすべて捨てていることを注意する。山手線は線型リストでない例である。

② 名簿。１人分の記述を一つの項目とし，その名簿中に書かれた順序のみに着目するとき，線型リストと考えることができる。ただし，この場合には，図２－３のような有向グラフで表現するよりは表の表現の方が普通である。

③ 待ち行列（キュー）。高速道路の料金所，宝くじの販売窓口，理髪店などや，コンピュータ内に作られるジョブの処理開始待ちの行列，コア・スペース要求の待ち行列などの例を示す。また，これらの線型リストは入り方・出方（すなわち項目の追加・削除の仕方）に制約があることを説明し，このような線型リストが「待ち行列」とか呼ばれることを紹介する。

④ スタック。カフェテリアに積みあげられた皿，ＫＩＯＳＫの週刊誌，行き止まりの引込線などや，再帰的な呼出しを許すサブルーチンでの帰り先番地待避の手法〔６〕などの例を示す。また，これらの線型リストにも入り方・出方に制約があることを述べ，このような線型リストが「棚」とか「スタック」とか呼ばれることを紹介する。さらに，キューとの対照としてＦＩＦＯ，ＬＩＦＯの言葉も説明する。また図２－４に示すような，スタックに関連した用語，また，図２－５に示すスタックに伴なう操作も解説する。特に図２－５の説明では，「スタック・ポインタ」（あるいは略記して「ＳＰ」と書いてもよい）と書いた札の裏に磁石をつけて，

黒板上の任意の位置に貼れるようなものを作っておくと役に立つ。

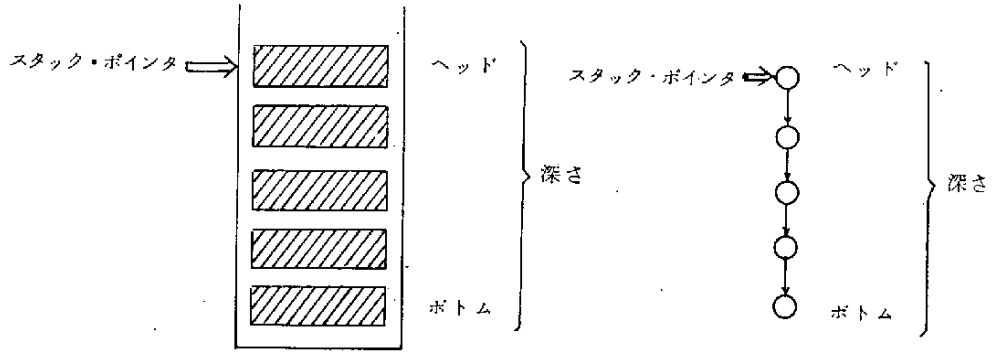


図２－４　　スタックに関連した用語

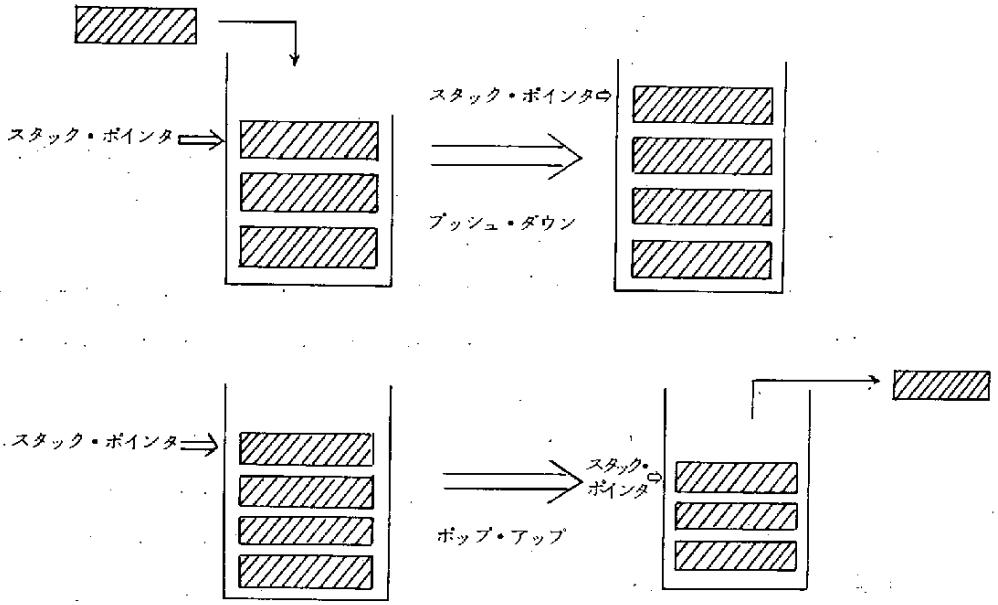


図２－５　　スタックに伴う操作

⑤．デキュー。図２－６が説明に役立つだろう。この図からａの線路を取り除いたものが「一方からしか入れないデキュー」，ｂの線路を取り除いたものが「一方からしか出られないデキュー」と呼ばれることを紹介し，ａ，ｂ両方を取り除くとキューになることを説明して，デキューとキューとの関連を明らかにしておく。

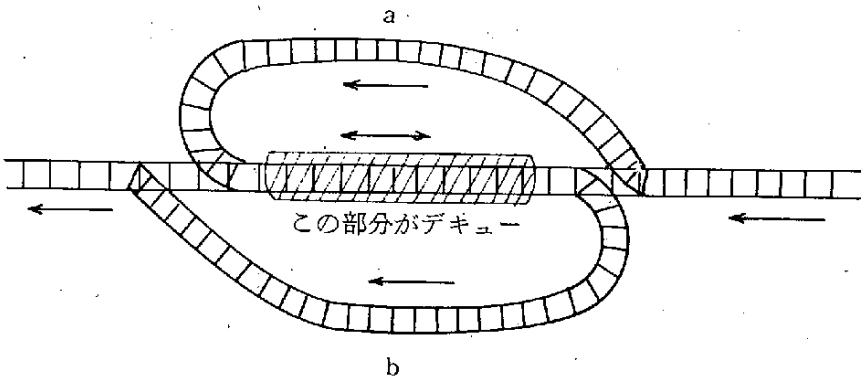


図２－６　　デ　キ　ュ　ー

最後に，線型リストや一般のリスト構造を図で表現する場合に，図２－７のように，項目（レコード）を箱で示し，その箱をデータ部とポインタ部に分け，ポインタは矢印で表現すること，および終端ポインタは⊠で示す習慣が確立していることも説明する。さらに，物理構造としては，ポインタとして０を使うことが多いということも簡単にふれておいた方がよいだろう。

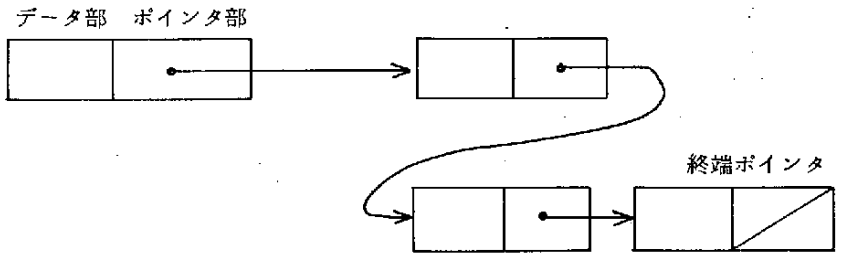


図２－７　線型リストの図表現

2.2　線型リストの基礎技術

次の項目をさすポインタつまり前進ポインタだけを持つ線型リストでは，「一つ前の項目」を知ることができないので，そのような項目を知りたい場合には，何らかの工夫が必要であることを説明する。

具体的な工夫としては，

- 「一つ前の項目」をさす後進ポインタを各項目に持たせる（図２－８）
- 先頭の項目をさすヘッド・ポインタを各項目に持たせる（図２－９）

がただちに考えられる。これらの構造を説明するとともに，ポインタの名称を説明する。なお，これらの構造で，項目を追加したり削除したりするアルゴリズムは次章に説明するので，ここでは説明する必要はない。

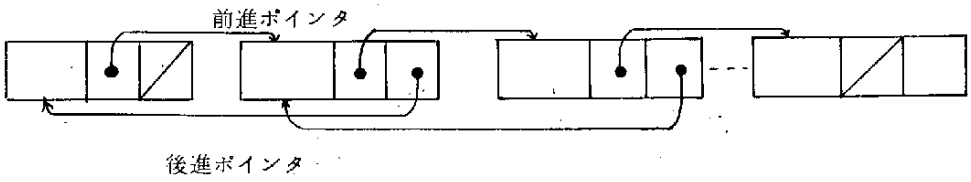


図２－８　両方向リスト

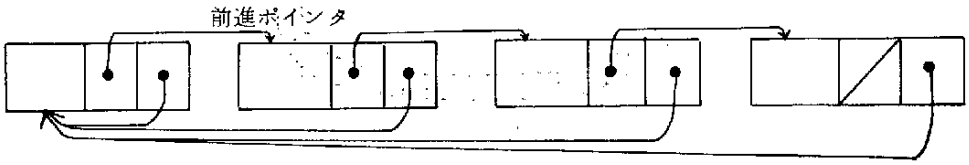


図２－９　ヘッド・ポインタを持つ線型リスト

これらはいずれも各項目にポインタを二つずつ持たせる構造であるが，ポインタを一つだけにしておいて，しかも前の項目を得られるようにするため，終端ポインタでヘッドをさすようにした循環連結を説明する（図２－１０）。

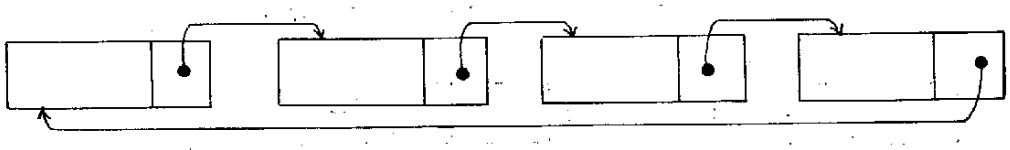


図２－１０　循環リスト

このとき，リストの先頭の項目を示すフラグを設けておいた方が便利であることを理解させる。

他にも，これらの基本的な構造を組み合わせた構造が考えられるので，時間があれば，それらもざっと紹介するとよいだろう。図２－１１は各項目に前進ポインタ，後進ポインタ，ヘッド・ポインタのすべてを持たせたフルポインタのリスト構造である。図２－１２はすべての項目に前進ポインタを持たせるとともに，一つおきに後進ポインタとヘッド・ポインタとを交互に持たせた構造で，ＣＯＲＡＬで採用されているものである〔９〕。

ＣＯＲＡＬに関連して，このようなリスト構造を扱うための処理系が存在すること，特に，両方向リストを扱うものとしてＳＬＩＰがあることも紹介しておく〔７〕，〔８　〕。リスト構造処理言語としては，他にもＬＩＳＰ，L^6 などたくさんあるが，それらは5.3節にまわした方がよい。

最後に，いささかトリッキーではあるが，マジック・リストを紹介しておくと面白いだろう〔11〕これは前進ポインタと後進ポインタとを一つのポインタですませてしまう手法で，図２－１３のような原理でできている。ただし，次の項目を知るためには，前の項目の位置がわかっていなければならず，前の項目を知るためには，次の項目の位置がわかっていなければならないという制

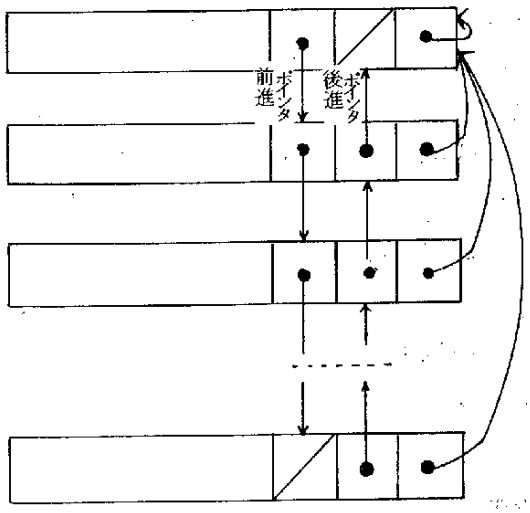


図２－11　フルポインタのリスト構造

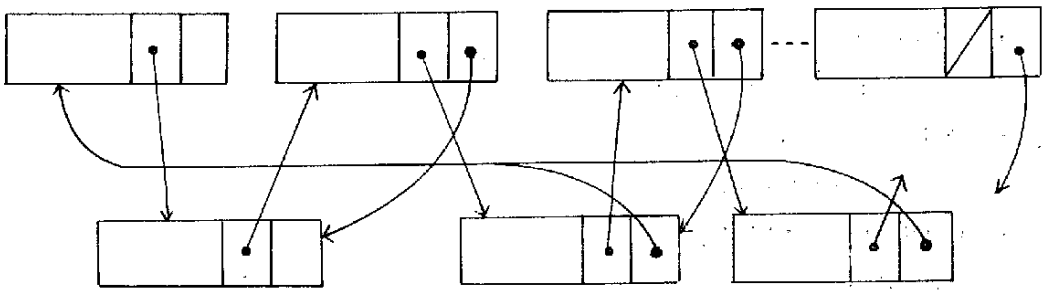
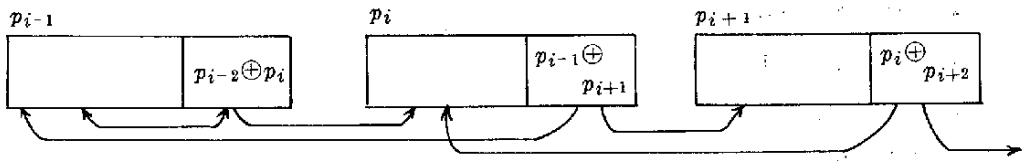


図２－12　CORALのリスト構造



$p\oplus q$はpとqの排他的論理和。あるいはpとqの和などを使ってもよい。
p_{i-1}の値とp_iのポインタ部$p_{i-1}\oplus p_{i+1}$がわかっていると$p_{i-1}\oplus(p_{i-1}\oplus p_{i+1})=p_{i+1}$であるから，$p_{i+1}$の値がわかる。

図２－13　マジック・リスト

約を説明することを忘れてはならない。

演習問題としては，

d：　各項目のデータ部の大きさ

p：　一つのポインタの大きさ

N： リスト中の項目の数

t： ポインタを1回たどるのに要する時間

を与えて，次の各構造で，それを格納するのに要するスペースと，任意の項目から一つ前の項目を得るのに要する時間の期待値とを計算させる。

- 両方向リスト
- ヘッド・ポインタを持つ線型リスト
- 循環リスト
- フルポインタのリスト構造
- CORALのリスト構造

2.3 線型リストの操作

次の各種の線型リスト

- 単純連結の線型リスト
- 両方向リスト
- ヘッド・ポインタを持つ線型リスト
- 循環リスト
- フルポインタのリスト構造
- CORALのリスト構造
- マジック・リスト

のそれぞれについて，

- 新しい項目の追加
- 新しい項目の挿入
- 項目の削除
- 項目のアクセス
  - その項目があるかどうかわからない場合
  - その項目が存在することがあらかじめわかっている場合

の各作業を行なう場合に，どのポインタをどのようにたどればよいのか，また，どのポインタをどのように書き換えればよいのかを，理解させる。項目のアクセスについては，リスト内の各項目が順不同で連結されている場合，あるキーに従って分類されている場合，さらに，分類された上，目次（ディレクトリ）を備えている場合の3通りを説明する必要がある。

一般に，構造を複雑にした場合とそうでない場合の長所・欠点が表2－1のようになることを理解させる。この長所・欠点は，第3～5章の多重連結構造にもそのままあてはまるので，その

時点でもう一度確認するとよい。

表2－1　構造の複雑さと特徴

構造が簡単 ⟷ 構造が複雑	
記憶域は少なくてすむ	ポインタの分だけ余分の記憶域が必要
アクセスに時間がかかる	アクセスは高速でできる
挿入・削除は簡単なアルゴリズムでできる	挿入・削除に複雑な手順が必要

また，リスト上の項目を一定のキーに従って分類し，これに目次をつけた構造は第3単元のファイル・システムで扱う索引順編成のファイルへのつなぎともなる。

以上の説明で，それぞれの構造の定性的な比較は理解できる。さらに厳密に定量的な比較を行なうためには，まず，項目の追加・挿入・削除・アクセスなどのアルゴリズムをよりきちんとした形で与えてやる必要がある。それには，

- 流れ図
- FORTRAN 〔13〕
- ALGOL 〔1〕,〔4〕
- アセンブリ言語 〔2, pp·120−160〕

などの手段が考えられ，対象とする人たちの予備知識に応じて適当なものを利用すればよい。一般には，ALGOL風の記述がコンパクトで理解しやすいと思われる。

たとえば，単純連結の線型リストxとyを，ポインタを書きかえてつなぐ手順を示すと次のようになる〔1, p·64〕。

```
pointer procedure Conc (x,y)
   value x,y; pointer x,y;
      begin
        Conc: = x;
        if x = " " then go to A;
C:      if lk(x) = " " then go to B;
        x: = lk(x);
        go to C;
B:      rl(x,y); go to E;
A:      Conc: = y;
```

E : end

ここでlk(x)はxのポインタ部の値を与える関係であり，rl(x,y) はxのポインタ部をyに書きかえる手続きである。

FORTRANでは第3～5章で多重連結構造を扱う場合などに手続きの再帰的呼出しができないという不便さがある。アセンブリ言語ではプログラムが非常に長くなってしまう。また，流れ図による表示も，手続きの再帰的呼出しをうまく表現しにくい。

いずれの表現方法を採用するとしても，構造の操作手順をはっきりと規定することによって，その手順の効率を定量的に比較することができる。

パラメータとしては，前回と同じく，次のものが必要となる。

d : 各項目のデータ部の大きさ

p : 一つのポインタの大きさ

N : リスト中の項目の数

t : ポインタを1回たどるのに要する間

tは一定としてよい。すなわち，構造全体がコア記憶装置などの等速呼出しの記憶装置に入っているものと仮定する。目次を設ける場合には，たとえば，項目はABC順に分類されており，項目のデータ部に入るキーの値の頭文字はA～Zに一様に分布し，さらに，目次はこの頭文字一つについて1エントリずつ，すなわち目次全体が26エントリから構成されているなどといった仮定が必要になってくる。

2.4 線型リストの応用

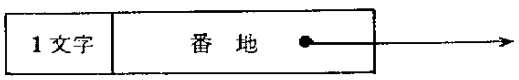
とりあげる話題はいろいろと考えられるが，ここでは，文字列を単純連結の線型リストで表現した場合の例と，高精度の数値を両方向リストとして表現した場合の例を示しておこう。

(1) 文字列の照合

文字列を単純連結の線型リストとして表現し，その文字列の中に別の文字列が部分文字列として含まれているかどうかを照合する手順を掲げておこう。詳しくは〔1〕などを参照のこと。

このような文字列照合は，SNOBOLや編集系などではきわめて重要な作業であることも合わせて説明しておくとよい。

① 1項目（1レコード）で1文字を次のようにあらわす。



データ部　　ポインタ部

② 文字列は上のレコードをつなげた形で表現する。(図２－１４)

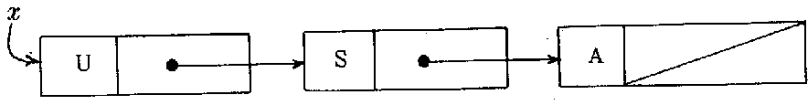


図２－１４　文字列の線型リストによる表現

このような表現をとると，文字列への部分文字列の追加・削除などがポインタ部の書きかえだけで実現できることを確認しておく。

③ 逆に，図２－１４のような線型リストのデータ部をつないでできる文字列を，その線型リストの"表現"と呼ぶ。この例ではリストxの表現は"ＵＳ･Ａ"である。

④ 次の三つの手続きを導入する。

v(x)　項目xのデータ部の値

lk(x)　項目xのポインタがさしている文字列

c(x,y)　一つの項目を作り，そのデータ部にx，ポインタ部にyを入れる

⑤ 文字列xが文字列yを部分文字列として含むかどうかは図２－１５の手順によって決定できることを説明する。このとき，(＊)の箱でリストyの値をzとして保存しておくこと

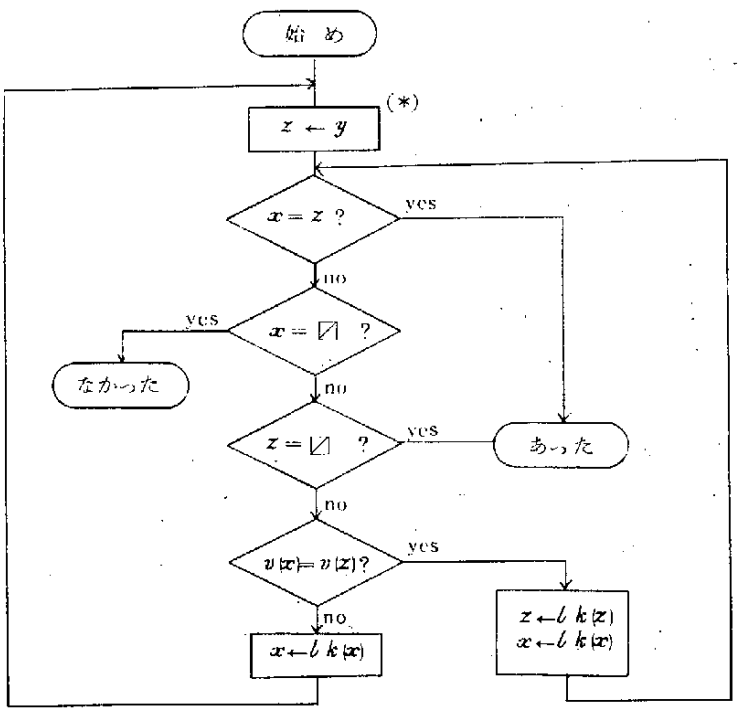


図２－１５　文字列の照合手順

がなぜ必要なのか（文字列yのはじめの方だけうまく照合できたのに結局y全体としてはうまく照合できなかったときのため）も実例をあげて説明しておく。

例：　x ＝ "URIURIGAURIURINIKURU"

　　y ＝ "URINI"

文字列の扱いに関連する演習問題の例を示す。

- 文字列x，yの接続。

  xの最後のポインタを書きかえて接続する方法

  x，yが表現する文字列を表現する線型リストを別に作って接続する方法

- 文字列x中に文字列yが含まれているかを照合し，含まれていたら，その部分を文字列zで置き換える。
- 上と同じで，置き換えてできた文字列x'中に文字列yが含まれていたら，置き換えをくり返す。
- 文字列x中に文字列yと文字列zにはさまれる文字列があるかどうかを照合する。
- 文字列x中に文字列yまたは文字列zが含まれているかどうかを照合する。

この他にも，ＳＮＯＢＯＬや編集系が行なう作業のほとんどが演習問題として使える。ただし，ＳＮＯＢＯＬでは，文字列の内部構造を規定していないことは知っておいた方がよい。

ＳＮＯＢＯＬ言語についての文献は，第５章の文献リスト参照のこと。

(2) 高精度演算 〔7〕,〔15〕,〔17〕

有効桁数に制限のない数値を扱う場合，その数値を何桁かずつに区切って，配列や線型リストの形で表現するのが普通であることを説明する。また，配列で表現したときには，配列の大きさによって，扱える桁数に制限がつくので，線型リストで表現した方が，スペース・時間の両面で効率は落ちるが，より完全な高精度演算が実現できることを簡単に説明する。

また，線型リストを物理構造で実現するにあたって，上位の桁から行なう演算（例：除算）と下位の桁から行なう演算（例：桁あがり，加減算）とがあるため，両方向リストを採用すると便利であることを示す。

なお，桁を区切るにあたっては，２進法で区切った方が演算速度や記憶スペースの効率がよく，１０進法で区切ると入出力に便利であることも簡単にふれておいた方がよいだろう。

例としては，円周率$\pi = 3.14159265358979323846\cdots_{(10)} = 3.1103755242102643021514230\cdots_{(8)}$をそれぞれ１０進４桁，８進５桁で区切った図２－１６のようなものを示すとよい。

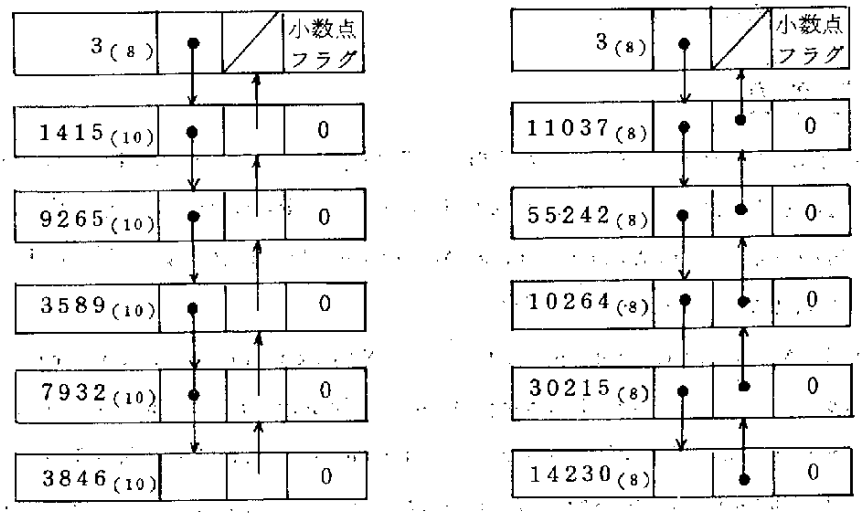
このような形であらわした数値M，Nは

$$M = A_0 + A_1 x + A_2 x^2 + \cdots\cdots$$

$$N = B_0 + B_1 x + B_2 x^2 + \cdots\cdots$$

のように書くことができる。xは10^{-4}, 8^{-5}などである。このとき，このM, Nに対する加減乗除演算は次のように行なうことができる〔15〕ことを示す。

$$M + N = (A_0 + B_0) + (A_1 + B_1)x + (A_2 + B_2)x^2 + \cdots\cdots$$



(a) 10進4桁で区切った場合　　(b) 8進5桁で区切った場合

図2－16　円周率πの両方向リストによる表現

$$M - N = (A_0 - B_0) + (A_1 - B_1)x + (A_2 - B_2)x^2 + \cdots\cdots$$

$$M \times N = A_0 B_0 + (A_0 B_1 + A_1 B_0)x + (A_0 B_2 + A_1 B_1 + A_2 B_0)x^2 + \cdots\cdots$$

$$M \div N = \alpha + \beta x + \gamma x^2 + \cdots\cdots$$

ここで

$$A_0 + A_1 x + A_2 x^2 + \cdots\cdots = B_0\alpha + (B_1\alpha + B_0\beta)x + (B_2\alpha + B_1\beta + B_0\gamma)x^2 + \cdots$$

これらの演算アルゴリズムを作らせたり，さらにこれらを組合わせ，ニュートン法を利用して，平方根や立方根などを求めるアルゴリズムを作らせるとよい。

ここでの考え方は，多項式の計算にも応用できる〔4, pp・251－295〕。

なお，この演習にあたって，主要な定数の8進表現が必要な場合には，〔2, p・614〕が便利である。

指導上の留意点

(1) 2.2では，複雑な構造を図示する場合に，ポインタの種類に応じて線の色を変えたり，ポインタの線がなるべく交差しないように配置を工夫したりする必要がある。

(2) 2.2や2.3では，いろいろの操作に要する時間の期待値を計算させる場合には，単に式を誘導させるだけでなく，手近に利用できるコンピュータの演算時間，記憶容量，語長などをもとにして，d，p，N，tの値を適宜定めて，期待時間，記憶容量が実際にどれくらいの値になるのかを理解させるのが望ましい。時間の余裕があれば，Nの関数としてグラフを書くのも理解を深める手だてとなろう。

参考文献

〔1〕浦昭二編「データ構造」電子計算機基礎講座6，共立出版，pp.7—16，55—70，1974

〔2〕D.E.Knuth, "The Art of Computer Programming, Vol.1", Fundamental Algorithms, Addison-Wesley, pp.120-160, 234-239, 251-295, 614, 1972

〔3〕P.A.V.Hall, "Computational Structures—An Introduction to Non—numerical Computing", Computer Monographs, Macdonald and Jane's, American Elsevier, pp.14—15, 61—86, 1975

〔4〕E.Horowitz, S.Sahni, "Fundamentals of Data Structures", Computer Science Press, pp.77—128, 140—169, 182—197, 1976

〔5〕小林孝次郎「情報構造」計算機ライブラリ5，サイエンス社，pp.8—33，1977

〔6〕藤野精一「リスト処理と言語解析」プログラミング演習シリーズ3，朝倉書店，pp.1—30，1970

〔7〕浅井清，中村康弘，石黒美佐子，稲見泰生，斉藤真之「記号処理言語」ＩＣＳライブラリ3，総合図書，pp.109—158，270—307，1971

〔8〕J.Weizenbaum, "Symmetric list processor", CACM, vol.6, pp.524—536, 1963

〔9〕W.R.Sutherland, "The CORAL language and data structure", Appendix C, Lincoln Laboratory Technical Report 405, 1965

〔10〕大須賀節雄「データ構造」bit，共立出版，vol.8，pp.19—24，1976(①：データ構造入門)，pp.519—524(②：データ型と基本オペレーション)，pp.595—600(③：データの性質と表現)，pp.857—861(④：ハッシュ・コーディング)，pp.999—1005(⑤：データ構造の形態1)，pp.1041—1047(⑥：データ構造の形態2)，pp.1243—1249(⑦：リスト，ガーベジ・コレクションおよび動的記憶割当て)，pp.1303—1308(⑧：まとめ)

〔11〕古川康一「マジック・リスト── 両方向リストを1個のポインタですませる方法 ──」情報処理，vol.12, №4, pp.248, 1971

〔12〕G.G.Dodd, "Elements of data management systems", Computing Surveys, vol.1, pp.117—133, 1969

〔13〕A.T.Berztiss, "Data Structures:Theory and Practice", Academic Press, New York, 1971.訳：古川康一，杉藤芳雄，宮川正弘，島田俊夫「データ構造 I，II」コンピュータ・サイエンス研究書シリーズ，日本コンピュータ協会，1974

〔14〕角田博保「べたづめ方式のSNOBOL3処理系における性能測定」第16回プログラミングシンポジウム報告集，pp.197—207, 1975

〔15〕谷本勉之助「実用数値計算法」森北出版，pp.262—274, 1958

〔16〕W.H.Burge, "Recursive Progamming Techniques", Addison—Wesley pp.46—50, 1975

〔17〕和田秀男「大きな数の処理」数理科学，vol.15, №4，サイエンス社pp.64—68, 1977年4月

〔18〕古川康一「データ構造」情報処理，vol.13, pp.302—310(I)，pp.554—562(II)，1972

〔19〕M.J.R.Shave, "Data Structures", Mc Graw—Hill, 1975

〔20〕H.S.Stone, "Introduction to Computer Organization and Data Structures", McGraw—Hill Computer Science Series, McGraw—Hill, 1972

〔21〕I.Flores, "Data Structure and Management", Prentice—Hall, 1970．訳：久保寛彦「データ構造」，「データ管理」竹内書店，1972

第3章　論理構造——木構造

キーワード

木(tree), 森林(forest), 部分木(subtree), 自由木(free tree), 有向木(oriented tree), 順序木(ordered tree), 木の数えあげ(enumeration of trees), 根(root), 葉(leaf, terminal node), 親(father), 子(son), 兄弟(sibling, brother), 長男(heir), 祖先(ancestor), 子孫(descendant), 節, 頂点(node, vertex), 枝, 辺(branch, edge), (木の)高さ(height), 重心(centroid), 平衡木(balanced tree), 分岐ノード(branch node), (ノードの)レベル(level), 木の配列表現(array representation of trees), 2進木(binary tree), 完全2進木(complete binary tree), データつき木(labeled tree), データつき2進木(labeled binary tree), 3進木(ternary tree), n進木(n-ary tree), 一般木の2進表現(binary tree representation of general trees), LISP(LISt processor), (木の)登り方(traversal method), 前置法(preorder), 挿入法(inorder), 後置法(post order), つなぎ木(threaded tree), 両方向2進木(two-way binary tree), 循環リスト(circular list), 構文解析(parsing), 代数式(algebraic formula), 辞書(dictionary), ゲーム(game), 家系図(family tree), ポーランド記法(Polish notation)

目　標

前章に続いて，論理構造を扱うが，この章では特に枝わかれのある構造の最初のものとして木構造を扱い，その必要性，扱う手法を説明し，応用例を与える。

この章の目標と，それを達成するためにとりあげる題材は次のようになっている。

まず，枝わかれのない線型リストが扱える構造の限界を述べ，枝わかれの必要性を認識させる。このため，辺に方向がない木について，最も基本的な概念・用語をいくつか与え，この章の導入とする。

次に，木の辺に方向を持たせた場合，すなわち有向木に関連してあらわれる多数の用語を家系図，植物の木と対応させながら導入する。コンピュータに関連する分野に広く応用される2進木には特に重点を置く。

また，既在の木構造に対する基本的な操作（要素の追加，削除，検索）のアルゴリズムを与える。

さらに，2進木の構造ができあがっているとき，そのノード全部を走査する手法，すなわち「木の登り方」がどのような場面で必要となるかを理解させ，木の登り方の3種類の方法を説明し，その3種類の登り方とポーランド記法との関連を理解させる。

最後に，コンピュータのまわりの分野を中心に，木構造の事例を示す。各自にテーマを与えて，結果を発表させるのもよい。

内　容

3.1　木の概要

次のようなトピックスを題材にして，枝わかれの必要性を説明する。

① 鉄 道 網

前に線型リストの説明では，新幹線を例にとりあげたが，一般の鉄道網で駅と駅のつながり具合を表現するには，分岐やループをあらわすことが必要である。

② 道 路 網

③ 通 信 網，コンピュータ・ネットワーク

たとえば，ＡＲＰＡＮＥＴの構成が〔4，P. 849〕に示されている。

④ ゲームの木

ゲームの局面の変化は木やネットワークとして表現できる。たとえば，エイトというゲームを題材にした木が〔3,応用編PP・147－149〕に示されているので，参考にするとよい。

⑤ 流 れ 図

箱のつながり方が線型リストで表現できるような一本道の流れ図ではプログラムを作る意味がない。そこで分岐，ループが必要となる。

⑥ 従業員の職制（図3－1）

⑦ 系　　図（図3－2）

⑧ 本の目次（図3－3）

なお，詳しいことは5.2でとりあげるので，ここでは，木やネットワークがはっきりとした形で出ているわかりやすいものを中心に，簡単に説明すればよい。ただ，とりあげたトピックスについて，それがループを含むかどうか（すなわち，木で表現されるのか，木でないループを含むネットワークで表現されるのか）は，意識させておく必要がある。

次に，ループのない枝わかれ構造として木を導入する。この段階では，厳密な定義は必要でない。

さらに，木がいくつか集まったものとして森，木の一部分として部分木を説明する。木という名前が植物の木に由来していることも述べておいた方が，次回の用語の導入に都合がよい。

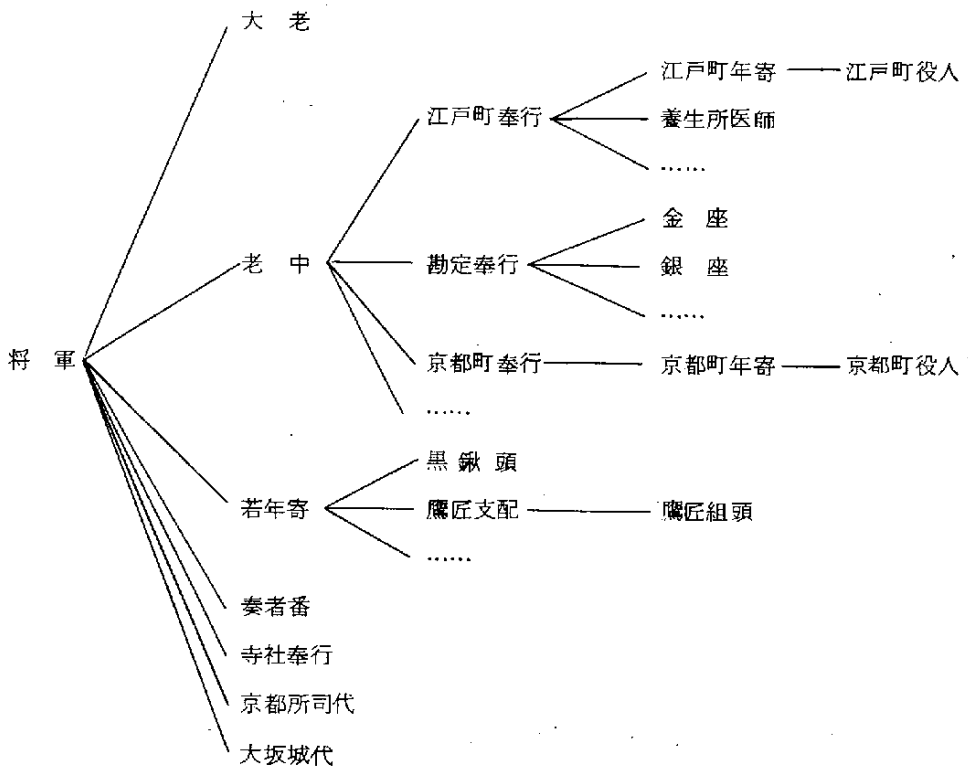


図３－１　武家官職（江戸時代）（「明解古語辞典」，三省堂より抜粋）

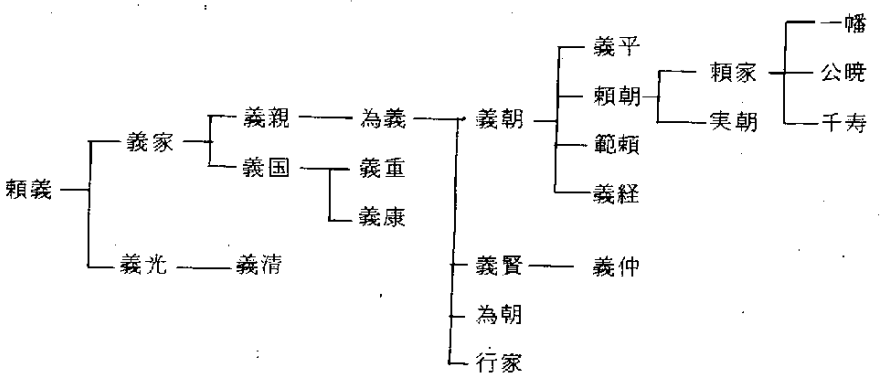


図 3.－２　清和源氏系図（「ポケット日本史」，昇龍堂出版より抜粋）

第一部

第１章

第１節

§１，§２

第２節

§３，§４，§５

第２章

第１節

§６，§７，§８，§９

第２節

§10，§11

第３節

§12，§13，§14

第二部

第１章

……

図３－３　本の目次の例

3.2　木の基礎技術

はじめに，前回示したトピックスについて，それを木やネットワークの形で書いたときに枝に方向がつくものと，そうする必要がないものとがあることを示す。ここで，枝に方向がある木，すなわち有向木と，枝に方向のない木，すなわち木あるいは自由木とを導入する。多くの木構造が有向木として表現した方が便利であることを説明する。

有向木はたとえば次のように定義できる。

①　はいってくる枝のない節がちょうど一つある。

②　節から出ていく枝の数についての制約はない。

③　①の節を除いたすべての節には，はいる枝がちょうど１つある。

④　ループがない。

有向木を導入すると，植物の木と対照しながら，次の用語を与えることができる。

・根

・葉

また，系図と対照して次の用語が与えられる。

・親

・子

・兄　弟

・長　男

・子　孫

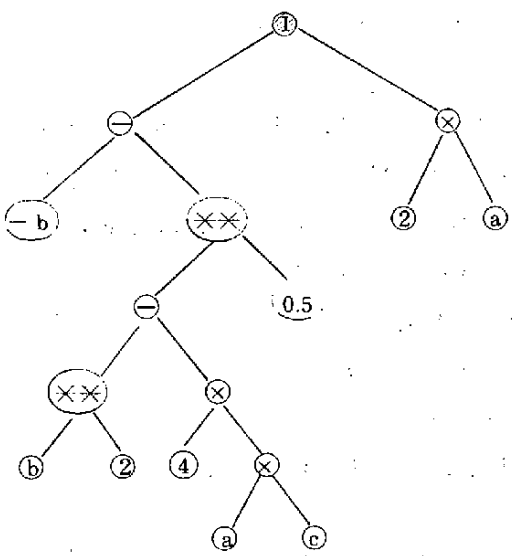
・祖　先

さらに，木の高さ，節のレベル，木の重心などグラフ理論からくる用語も必要に応じて適宜紹介しておく。

これらの用語を導入したあと，次の話題は２進木である。２進木には，節にデータのついたデータつき２進木と節にデータのつかない通常の２進木とがある。

データつき２進木は，たとえば数式の構文解析などに利用されていることを図を用いて説明する（図３－４）。

２進木は一般の木との関連で重要である。木の各節を，長男を示すポインタと次弟を示すポインタとによって書きかえると，一般の木がデータつき２進木で表現できることを，実例を使い，実際にポインタを書きかえながら説明する（図３－５(b)）。さらに，図３－５(c)のように，左の枝で親，右の枝で子供たちをつないだ木を持つようにすると，同じ木を２進木としてもあらわせることを示す。



$$\frac{-b-(b^2-4ac)^{\frac{1}{2}}}{2a}$$

図３－４　算術式のデータつき２進木による表現

データつき２進木も２進木も，一つの節から出る枝がたかだか２本しかないので，コンピュータ内で表現する場合に便利であることが２進木の大きな利点であるから，このことをよく説明しておかなければならない。

2進木の形にしておけば，表の形で表現するのも楽であることも述べる（表3－1）。木，2進木，表それぞれの表現で，親から子（たとえば図3－5，表3－1で頼義から義経）へとたどるにはどうすればよいかも確認しておくとよい。

表3－1　2進木に変換した家系図の表表現の一例

1	頼義	2	※	13	行家	※	※
2	義家	4	3	14	義平	※	15
3	義光	※	※	15	頼朝	19	16
4	義親	7	5	16	範頼	※	17
5	義国	8	※	17	義経	※	※
6	義清	※	※	18	義仲	※	※
7	為義	10	※	19	頼家	21	20
8	義重	※	9	20	実朝	※	※
9	義康	※	※	21	一幡	※	22
10	義朝	14	11	22	公暁	※	23
11	義賢	18	12	23	千寿	※	※
12	為朝	※	13				

2進木の形にしておけば，表の形で表現するのも楽であることも述べる（表3－1）。木，2進木，表それぞれの表現で，親から子（たとえば図3－5，表3－1で頼義から義経）へとたどるにはどうすればよいかも確認しておくとよい。

3進木，n進木，完全2進木は言葉だけ説明しておけばよいだろう。

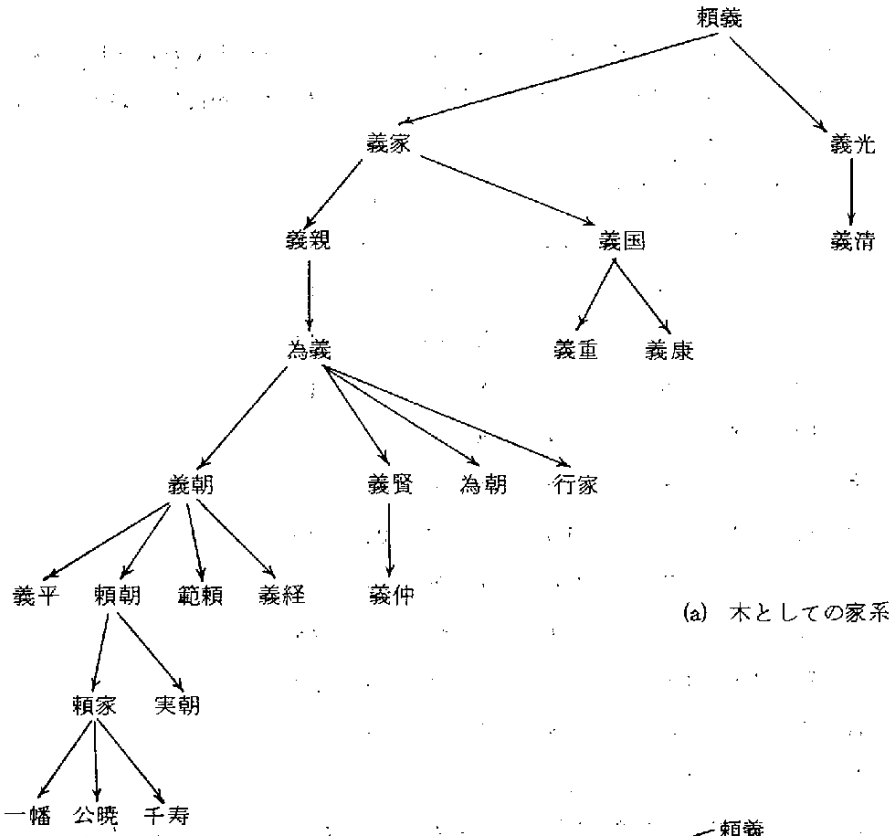
最後に2進木リストを内部構造として持つ処理系ＬＩＳＰの話を簡単にしておくとよい。内部構造の紹介，言語の基本思想（基本関数からのつみあげですべてを表現する）について述べ，さらに簡単なＬＩＳＰの関数をとりあげて，2進木がその関数でどのように処理されるのかを紹介する。ＬＩＳＰの詳細は5.3で扱う。

なお，ＬＩＳＰに関する文献は第5章の参考文献リストを参照のこと。

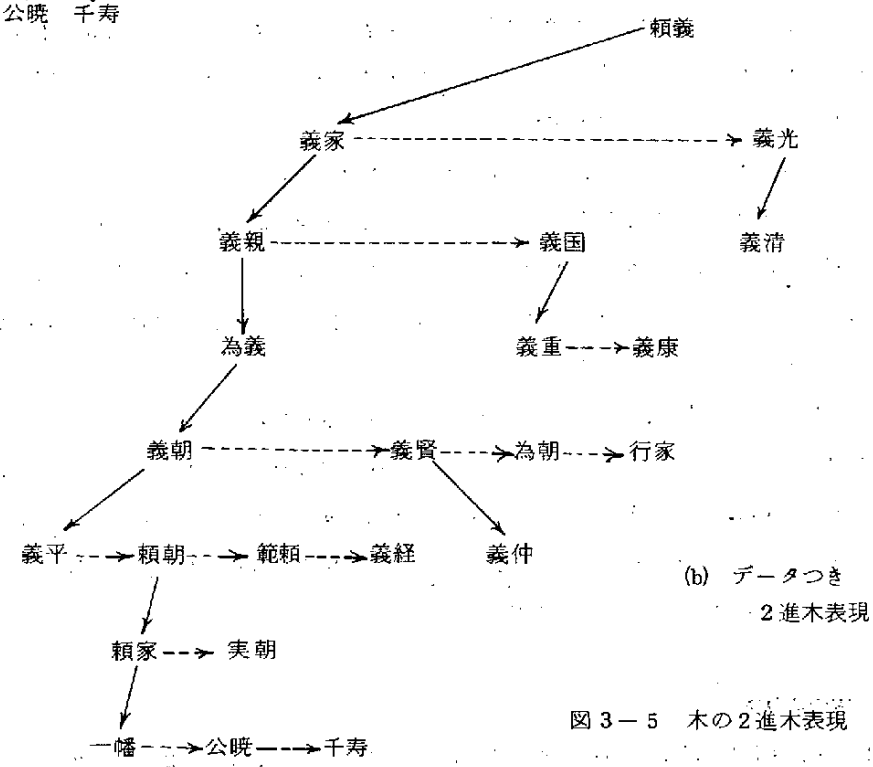
演習問題としては，いろいろの木をデータつき2進木や2進木に変換させたり，逆の変換をさせたりする。また，表3－1とは違った表表現を工夫させるとよい。それらの長所・欠点を互いに討論させるのも有益であろう。さらに，そうした表表現を採用したとき，節の検索，追加などがどのような手順で行なえるのかも検討しておく必要がある。

3.3　木の登り方

はじめに，木に登るとはどういうことなのかを説明する。すなわち，2進木の構造ができあが

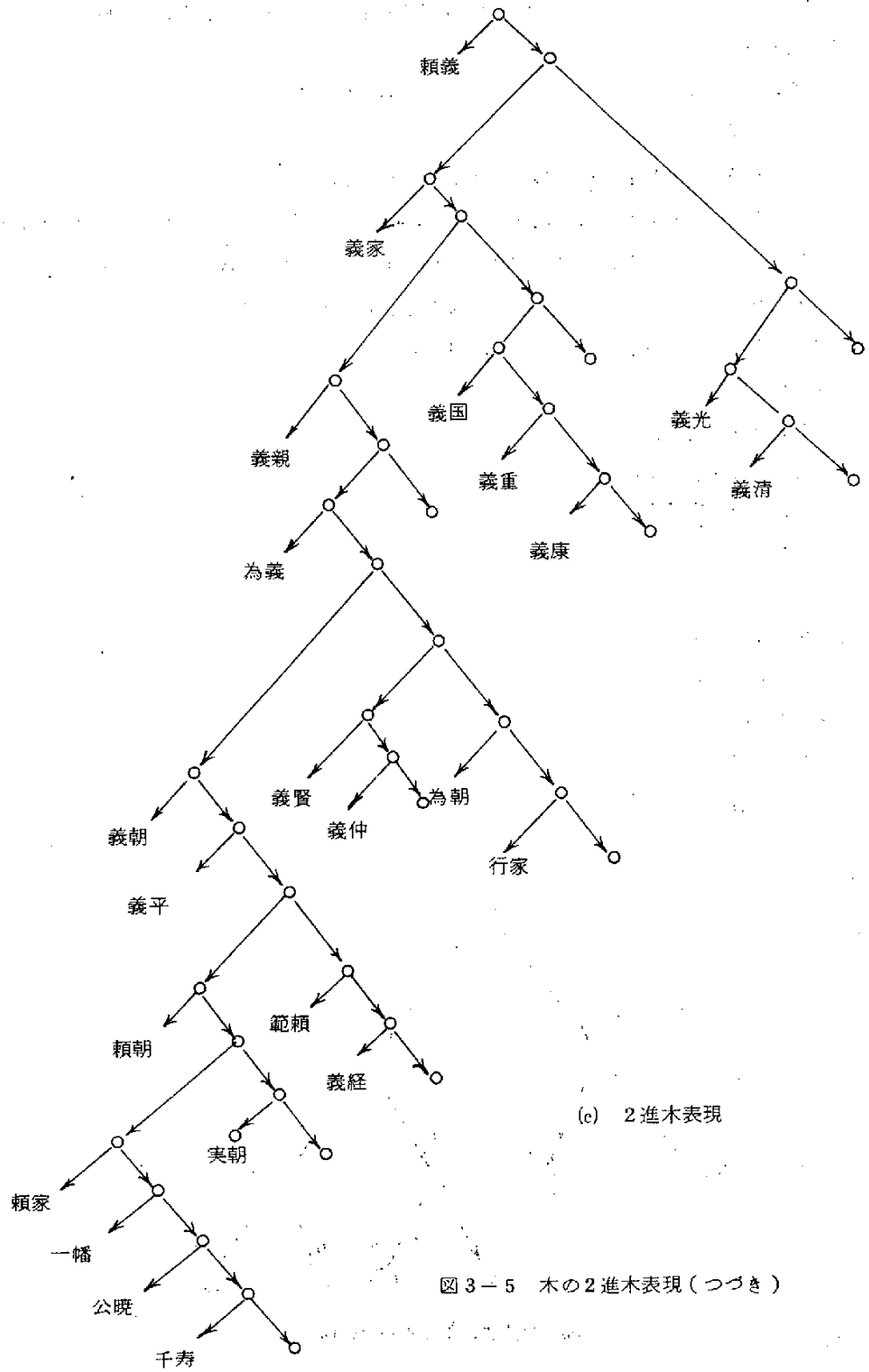


(a) 木としての家系図



(b) データつき
2進木表現

図 3－5　木の2進木表現



(c)　2進木表現

図3－5　木の2進木表現（つづき）

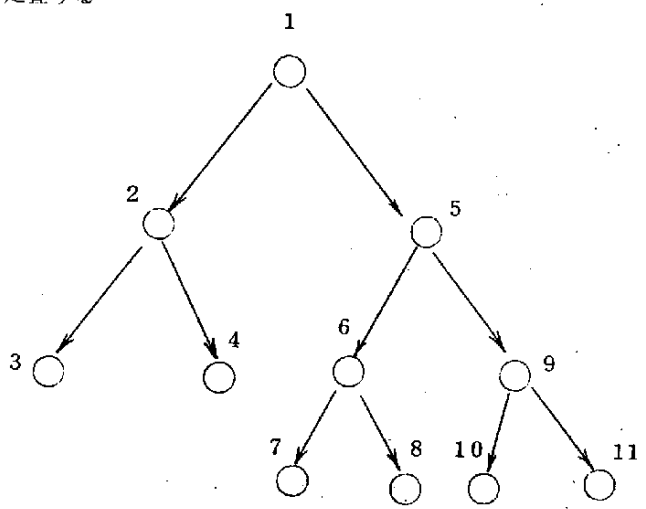
っているとき，そのすべてのノードを順次走査する作業のことを「木に登る」と呼んでいるということを説明する。

さらに，次のような場面で木に登る必要が生ずることを示す。

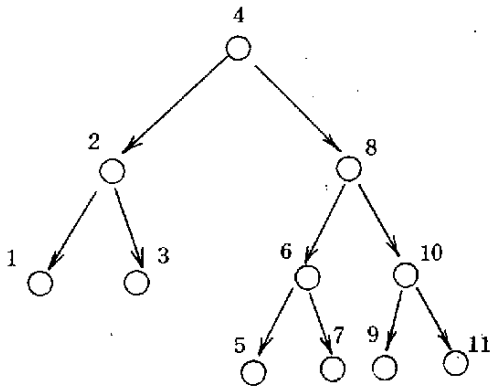
- ある頂点をみつけたいとき
- データつき2進木で表現された算術式をコンパイルするとき
- 一括型のちり集め（garbage collection）の作業の一部として（現在“生きている”リストにつながっているセルすべてにマークをつける段階で木に登る手法が必要になる。マークがつかなかったセルが“ちり”として自由リストにつながれる）

次に，3種類の木の登り方を説明する。すなわち，

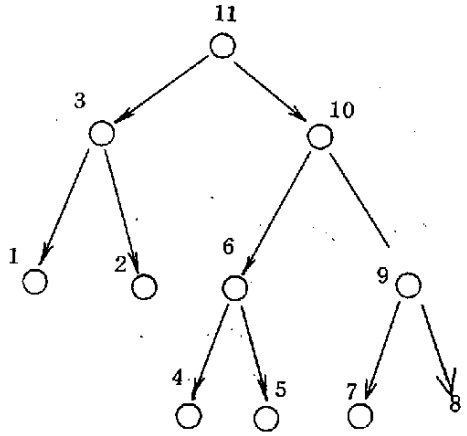
- 前置法
  - ア．節を走査する
  - イ．左部分木に登る
  - ウ．右部分木に登る
- 挿入法
  - ア．左部分木に登る
  - イ．節を走査する
  - ウ．右部分木に登る
- 後置法
  - ア．左部分木に登る
  - イ．右部分木に登る
  - ウ．節を走査する



(a) 前置法による走査の順序



(b) 挿入法による走査の順序



(c) 後置法による走査の順序

図３－６　木の登り方とノードを走査する順序

２進木の例を一つ示して，具体的にこれらの手法で，節がどのような順に走査されるかを示すとよい（図３－６）。なお，これらの手法はいずれも再帰的な形で書かれるので，時間の余裕があれば，これらの手法を説明する前に，再帰の概念を復習しておくとよい。

最後に，これらの木の登り方と算術式のポーランド記法との間に密接な関係があることを説明しておくと興味深いだろう。例としては，データつき２進木でいえば，木の登り方を説明するときに用いた２進木と同じ構造をもつような算術式を使うとよい（図３－７）。この２進木に３種類の木の登り方を適用して，節を走査するときに，その節に対応する演算子，被演算数を順に出すと，次のような文字列が得られる。

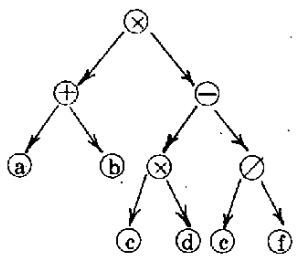


図３－７　データ付き２進木表現した算術式

（ａ＋ｂ）×（ｃ×ｄ－ｅ／ｆ）

- 前　置　法

  ×＋ａｂ－×ｃｄ／ｅｆ　：ポーランド記法

- 挿　入　法

  ａ＋ｂ×ｃ×ｄ－ｅ／ｆ

- 後　置　法

  ａｂ＋ｃｄ×ｅｆ／－×　：逆ポーランド記法

これらの文字列がそれぞれ，ポーランド記法，括弧を除いた式，逆ポーランド記法になっている

ことを説明する。

演習問題としては，３種類の木の登り方をＬＩＳＰなどの言語でプログラムさせたり，あるいはプログラムをあらかじめ用意しておいて，その動きを解析させると，再帰呼出しの様子をよく理解させることができる。

3.4　両方向の木構造

これまでに説明した木構造では，根から葉に向かってたどることはできても，逆に葉から根に向かってたどることはできないということをはじめに注意する。そして，葉から根をたどりたくなるようなケースとして次のような場合を説明する。

①　ゲームの木

ある局面まで読みを進めた段階で，その局面が具合が悪いことがわかって，もとの局面に戻したい場合。通常は読みを進める過程で局面をスタックに入れて逆戻りができるようにしておくが，スタックを使わずに，木構造そのものを逆戻り可能なように構成しておくことも考えられる。

②　従業員の職制

ある従業員の直接の上司が誰であるか知りたいときや同僚を知りたいとき。

③　系　　図

親を知りたいときや，兄や弟を知りたいとき。

このような場合に対処するため，木構造で葉から根をたどれるような工夫がいろいろと考えられている。ここでは，そのうちの次の手法を説明する。

- 両方向２進木
- 循環リスト
- つなぎリスト

これらの手法は，線型リストでテイルからヘッドをたどれるようにした手法と対応させながら説明するとよい。すなわち，通常の木構造は単純連結の線型リストに対応し，両方向２進木は両方向リストに，循環リストはそのまま循環リストに対応している。

(1)　両方向２進木

図３－８に示すように，親から子をさす通常のポインタの他に，子から親をさすポインタを各項目に持たせたものであることを説明する。各項目でポインタが三つずつ必要になることも確認しておく。

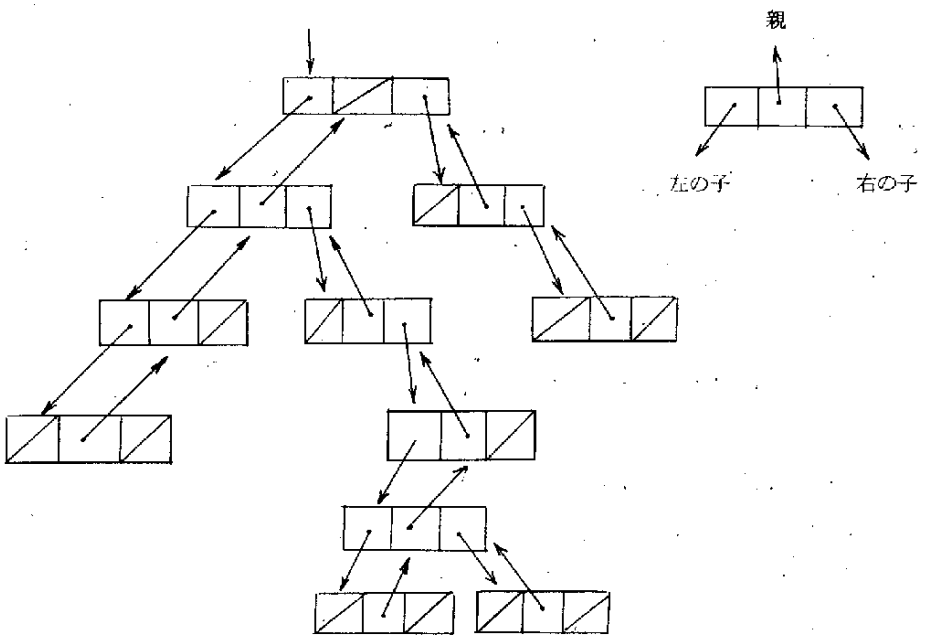


図 3－8　両方向 2 進木

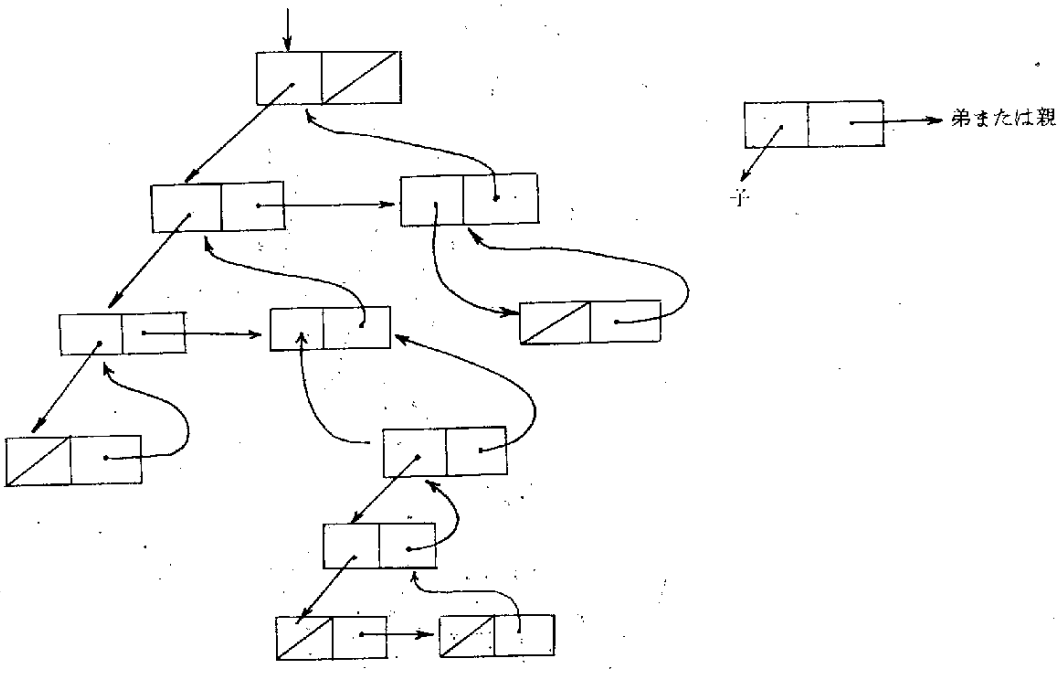


図 3－9　循環リスト

(2) 循環リスト

図３－９に示すように，弟をさすポインタ部の終端ポインタで親をさすようにすると，一つの項目のポインタは２個のままで，子から親を知ることができるようになることを図の上で確認する。ただし，この図のような不完全な２進木では，子をさすポインタが左の子をさしているのか右の子をさしているのかを示すフラグが必要であることも説明する必要がある。このフラグは，弟または親をさすポインタが弟と親のどちらをさしているのかを示すフラグをつけても同じであることを理解させる。

(3) つなぎリスト

次の規則で作られる木構造である〔１〕。この定義に忠実に従いながら，黒板上でポインタを一つ一つ書きこんでいく。ここで「左ポインタ」は左の子をさすポインタの意味とする。「右ポインタ」も同様（図３－１０）。

① 節Ａの左ポインタが終端ポインタのときは，Ａの親Ｂ，さらにその親Ｃと順に見ていって，親が左ポインタでさしている限り，左ポインタをさかのぼり，初めて右ポインタでさしている親Ｐに出会ったとき，Ａの左ポインタでそのＰをさすようにする。

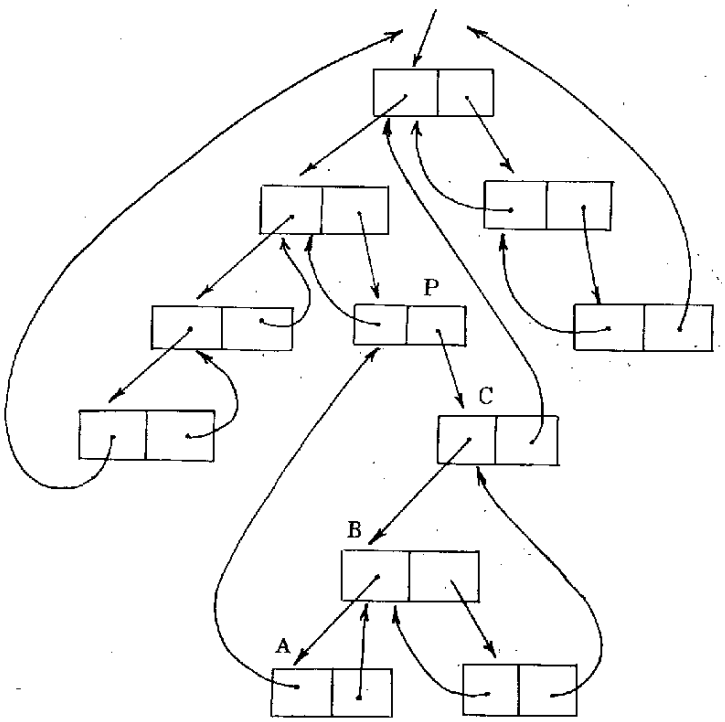


図３－１０　つなぎリスト

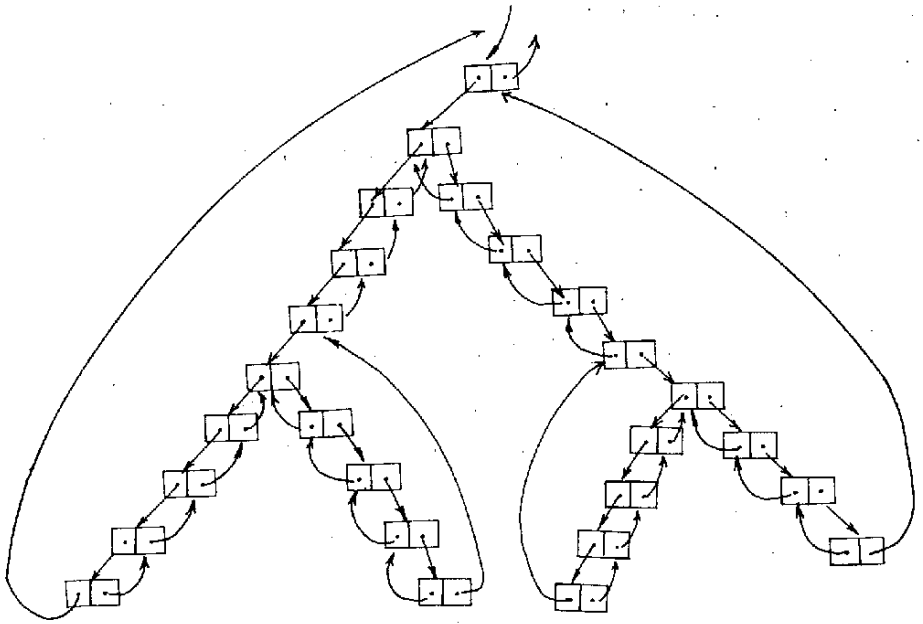


図３－１１　つなぎリストの概念図

② 同様に，右ポインタが終端ポインタのときは，その親を順に見ていって，親が右ポインタでさしている限り，右ポインタをさかのぼり，初めて左ポインタでさしている親に出会ったとき，右ポインタでその親をさすようにする。

この構造では，木登りのために，各項目に，左ポインタと右ポインタが親をさしているのか子をさしているのかを示すフラグを二つずつつけておく必要がある。

この構造は，図３－１１のような図を見せると，理解させやすい。

最後に，つなぎリストの木登りは次の手順でできることを説明する〔１〕

根から左ポインタをたどる。左ポインタが親に向かっているものに到達した場合，右ポインタをたどる。これが親に向かっているときは，そのまま親へもどるように進める。右ポインタが子に向かっているときは，その子へ到達した時点で左ポインタにきりかえて木登りを続ける。

この木登りではスタックを必要としないこともつけ加えておく。

演習問題としては，２進木構造を与えて，両方向２進木，循環リスト，つなぎリストを作らせ，また，つなぎリスト上で木登りの順序を追跡させる程度でよいであろう。

時間があれば，各種のリスト上で，節を追加したり削除したりするアルゴリズムを工夫させることもできる。

3.5　木の応用

次のようなテーマを適宜取りあげる。

(1)　構文解析

バッカス記法に従って，文からparsing　tree　が作れる。この木では，葉はすべて終端記号であり，その他の節はメタ記号に対応するデータ付きの木になる。文法があいまいであるとparsing　tree　が一意に定まらない（例を示す）。

(2)　代数式，ポーランド記法

演算子の優先順序を考えながら，代数式をデータつき2進木に変換する〔2〕。あるいは，データつき2進木をコンパイルして，目的コードを生成する。

データつき2進木であらわされた代数式をある変数に関して微分し，その結果をデータつき2進木であらわす。そのようにして得られたデータつき2進木に次のような簡単化を行なう（eは任意の式）。

e＋0 → e	0＋e → e
e－0 → e	0－e → (－e)
e×0 → 0	0×e → 0
e×1 → e	1×e → e
e／1 → e	

(3)　辞　書

索引順次ファイル（indexed　sequential　file）を磁気ディスク装置上に作ることを考える。このとき，通常は検索の便を考えて，目次をつける。この目次は，カバーする範囲と詳しさによって次のようなものが考えられる。

- マスタ・インデックス
- シリンダ・インデックス
- トラック・インデックス

これらの目次は全体として木構造を構成する。

(4)　ゲ　ー　ム〔3〕

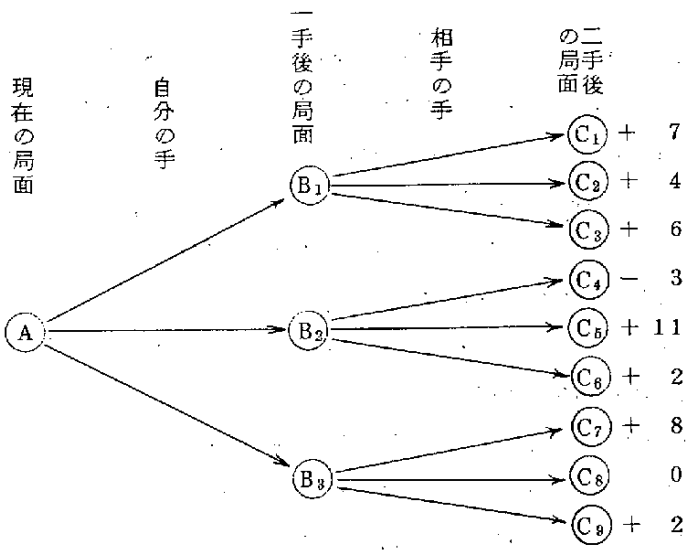
ゲームの局面を節と考え，一つの手を選ぶことによる局面の変化を枝と考えれば，ゲームは初期局面を根とする木構造と考えることができる。

簡単なゲームの木を示し，その木の上でミニマックス戦略や木のかりこみの原理を説明する（図3－12）

(5)　家　系　図

木の探索アルゴリズムの例として，ある人を基準にして，次のような人の集合を求める。

- 尊　　族



数字は局面の評価点，＋は自分に有利，－は相手に有利。局面B_1，B_2，B_3 の評価点は最小をとってそれぞれ＋4，－3，0となる。このうちの最大をとってB_1 の局面を選べばよい。なお，C_4 が－3であることがわかった時点で，C_5，C_6の評価は省略できる。

図３－12　ゲームの木とミニマックス戦略

- 卑　　族
- 甥・姪
- 伯父・叔父
- ５親等

このような操作を行なうためには，木を子から親へとたどれるようにしておく必要があることを説明し，構造を工夫させる。

指導上の留意点

(1)　この章の内容は特にグラフ理論と密接な関係を持つ分野なので，受講者が数学的な素養を持つ場合には，木の数，操作効率などに関して，式の誘導を含む厳密な形での定式化を提示することも可能である。

(2)　3.2では取り扱う分量が多いので，他の項にくらべると，余計に時間をかけた方がよい。

参考文献

〔1〕浦昭二編「データ構造」電子計算機基礎講座６，共立出版，pp.16－24，70－100，1974

〔2〕 D.E.Knuth, "The Art of Computer Programming Vol.1", Fundamental Algorithms, Addison-Wesley, pp.305—406, 1972

〔3〕 A.I.Forsythe, T.A.Keenan, E.I.Organick, W.Stenberg, "Computer Science—A First Course", John Wiley & Sons, 1969. 浦 昭二訳「コンピュータサイエンス入門1基礎編, 2応用編」培風館, pp.143—170, 187—217(2応用編), 1972

〔4〕 野口正一「コンピュータ・ネットワークの最近の研究・開発動向」情報処理, vol.18, №8, pp.838—850, 1977

〔5〕 大須賀節雄「データ構造」bit, 共立出版, vol.8, pp.19—24, 1976(①:データ構造入門), pp.519—524 (②:データ型と基本オペレーション), pp.595—600 (③:データの性質と表現), pp.857—861(④:ハッシュ・コーディング), pp.999—1005(⑤:データ構造の形態1), pp.1041—1047(⑥:データ構造の形態2), pp.1243—1249(⑦:リスト, ガーベジ・コレクションおよび動的記憶割当て), pp.1303—1308(⑧:まとめ)。

〔6〕 W.H.Burge, "Recursive Programming Techniques", Addison Wesley pp.119—129, 1975

〔7〕 P.A.V.Hall, "Computational Structures—An Introduction to Non—numerical Computing", Computer Monographs, Macdonald and Jane's, American Elsevier, pp.8—14, 1975

〔8〕 G.G.Dodd, "Elements of data management systems", Computing Surveys, vol.1, №2, pp.117—133, 1969

〔9〕 A.T.Berztiss, "Data Structures:Theory and Practice", Academic Press, 1971. 古川康一, 杉藤芳雄, 宮川正弘, 島田俊夫訳「データ構造 I, II」コンピュータ・サイエンス研究書シリーズ, 日本コンピュータ協会, pp.201—234, 332—335, 1974

〔10〕 古川康一「データ構造」情報処理, vol.13, pp.302—310(I), pp.554—562(II), 1972

〔11〕 E.Horowitz, S.Sahni, "Fundamentals of Data Structures", Computer Science Press, pp.155—169, 218—281, 1976

〔12〕 小林孝次郎「情報構造」計算機ライブラリ5, サイエンス社, pp.34—61, 1977

〔13〕 J.K.Rice, J.R.Rice, "Introduction to Computer Science", CBS Education International, 1969. 細井 勉, 関本年彦, 佐藤雅彦訳

「コンピュータサイエンスⅠ，Ⅱ，Ⅲ」サイエンスライブラリ情報電算機22，23，24，サイエンス社，pp.61—91(Ⅲ)，1974

〔14〕 M.J.R.Shave, "Data Structures", McGraw—Hill, 1975

〔15〕 H.S.Stone, "Introduction to Computer Organization and Data Structures", McGraw—Hill Computer Science Series, McGraw—Hill, 1972

〔16〕 I.Flores, "Data Structure and Management", Prentice—Hall, 1970.久保寛彦 訳「データの構造」，「データの管理」竹内書店，1972

〔17〕 小林孝次郎「情報構造」計算機ライブラリ5，サイエンス社，1977

第4章　論理構造——配列・行列

キーワード

配列(array), 表(table), 多次元配列(multi－dimensional array), 添字(subscript), 要素(element), 記憶変換関数(storage mapping function), 割当て(allocation), 行方向の割当て(row-wise allocation), 列方向の割当て(column-wise allocation), 逐次型割当て(sequential allocation), 三角行列の割当て(allocation for triangular matrix), 対称行列の割当て(allocation for symmetric matrix), 三重対角行列の割当て(allocation for tri-diagonal matrix), 疎な行列(sparse matrix), 疎な行列のリスト表現(linked representation of sparse matrices)

目　標

第3章で扱った木構造がデータ間の階層関係を表現する構造であったのと同様に，表や配列も，データ間の順序を表現する一つの構造であることを再認識させるのがこの章の第一の目標である。

なお，配列は論理構造としての表を実現する物理構造の一つにすぎないこともくり返し述べる。

次に，記憶変換関数，割当てなどの用語の意味を理解させるとともに，配列が記憶域上に具体的にどのように対応づけられるのかを説明する。配列は通常のものの他に，特殊な性質を持ったものもいくつか取り上げて，変換関数を誘導する。

最後に，疎な行列について，それがどのようなもので，どのような場面にあらわれるのかを理解させる。

また，疎な行列をリスト表現する手法を説明し，そうすることによる長所・短所を理解させる。

内　容

4.1　配　列

表や配列がどういうものであるのかはあらためて説明する必要はないであろう。ただし，1行あるいは1列だけからなる表や1次元配列が線型リストと同じ構造であることは説明しておいた方がよい。

さらに，このような表や配列と線型リストとの性質の違いも次のように対照させておく必要がある。

① 表や配列では，「一つ前の要素」もすぐに得られるが，線型リストではそのためのポインタを特に持たせない限り得られない。

② 表や配列では，はじめからn番目とか，ある要素からn個前（あるいは後）とかも直ちにアクセスできるが，線型リストではn回ポインタをたどらなければならない。

③ 線型リストでは，ポインタの分だけ余分の記憶域が必要となる。

④ 表や配列では，途中に要素を追加したり，途中の要素を削除したりすると，それ以降の要素をすべてずらさなければならないが 線型リストでは，ポインタのつなぎかえだけで実現できる。

次に，配列と表との違いを説明する。すなわち，普通は，「表」は論理構造を示す言葉であり，「配列」は物理構造を示す言葉である。表は配列の形で実現されることが多いが，必ずしも常にそうであるというわけではない。次のような例を簡単に示すとよい。

① 線構造の配列表現（待ち行列やスタックの要素を配列に入れる）

② 木構造の配列表現（表3－1を2次元配列として見る）

③ ネットワークの配列表現（連結行列（incidence matrix））。これについては5.1で詳しく扱う）

④ 表の線型リスト表現（追加・削除がひんぱんにおこる場合など）

⑤ 表の木構造表現（索引順次ファイルに何段もの目次をつけたような場合（3.5の「辞書」参照）。

⑥ 2次元の表（行列）を一般のリスト構造で表現する（4.3で説明する疎な行列の場合など）

演習問題としては，配列に対する次のような操作のアルゴリズムを作らせるのがよい。

- 要素の削除
- 要素の追加
- 要素の挿入
- 要素の検索

{ イ．要素が何らかのキーに従って並んでいる場合
{ ロ．要素が順不同で格納されている場合

また，できれば，そのアルゴリズムの効率も計算させる。なお，要素の検索でイのケースについてはさまざまの手法が工夫されており，第9章で詳しく扱う。

4.2 行列の割当て

はじめに，記憶変換関数とか割当てとかの意味を説明する。

次に，いくつかの配列について記憶変換関数の形を具体的に示し，それを誘導して見せる。*

* ここでは添字は1から始まるものとしている。

(1) 1次元配列の要素 $a(i)$

$$loc(a(i))=loc(a(1))+(i-1)S$$

ここで $loc(x)$ は x に対して割当てられるアドレス，S は一つの要素に対して割当てられる語数である。

1次元配列では，添字の値の上限がわからなくても記憶変換関数を計算できることを注意しておく。

(2) 2次元配列の要素 $a(i,j)$

添字の上限をそれぞれ I，J とする。

2次元配列については，2通りの割当てが行なわれていること，それぞれ「行方向の割当て」，「列方向の割当て」と呼ばれていることをまず説明する。

① 行方向の割当て

$$loc(a(i,j))=loc(a(1,1))+\{(i-1)J+j-1\}S$$

② 列方向の割当て

$$loc(a(i,j))=loc(a(1,1))+\{(i-1)+(j-1)I\}S$$

2次元配列の場合には，添字の上限 I または J がわからないと，記憶変換関数が計算できないことを説明する。

受講者がFORTRANを知っている場合には，FORTRANで列方向の割当てを採用していることも説明しておくとよい。

(3) 3次元配列の要素 $a(i,j,k)$

添字の上限をそれぞれ I，J，K とする。

全部で3！＝6通りの割当てが考えられるが，代表的なものはやはり次の2通りであるので，これらの式を誘導して見せる。

① 行方向の割当て

$$loc(a(i,j,k))=loc(a(1,1,1))+\{((i-1)J+(j-1))K+(k-1)\}S$$

② 列方向の割当て

$$loc(a(i,j,k))=loc(a(1,1,1))+\{(i-1)+I((j-1)+J(k-1))\}S$$

行方向の割当ての場合には，配列要素の添字の値を i，j，k の順に拾いながら，一時記憶を使わずに記憶変換関数が計算できることも説明しておく。

(4) 多次元配列の要素 $a(i_1,i_2,\cdots,i_m)$

添字の上限をそれぞれ $I_1, I_2, \cdots, I_m$ とする。全部で $m!$ 通りの割当てが考えられる。

① 行方向の割当て

$$\mathrm{loc}(a(i_1, i_2, \cdots, i_m)) = \mathrm{loc}(a(1, 1, \cdots, 1)) + \{((\cdots(i_1-1)I_2+(i_2-1))I_3+\cdots+(i_l-1))I_{l+1}+\cdots +(i_m-1)\}S$$

② 列方向の割当て

$$\mathrm{loc}(a(i_1, i_2, \cdots, i_m)) = \mathrm{loc}(a(1, 1, \cdots, 1)) + \{(i_1-1)+I_1((i_2-1)+I_2((i_3-1)+\cdots + I_{l-1}((i_l-1)+\cdots+I_{m-1}((i_m-1))\cdots)\}S$$

(5) 三角行列の要素 $a(i, j)$

$$\begin{bmatrix} a_{11} & & & & 0 \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ \cdots & \cdots & \cdots & \cdots & \\ a_{N1} & a_{N2} & a_{N3} & \cdots & a_{NN} \end{bmatrix}$$

N の値が大きいときには，記憶変換関数の計算が少々複雑になっても，三角行列の特徴を生かした割当てを行なって記憶域を節約した方がよいこともあるということを説明する。

① 行方向の割当て

$$\mathrm{loc}(a(i, j)) = \mathrm{loc}(a(1, 1)) + \left\{\frac{1}{2}i(i-1)+(j-1)\right\}S$$

② 列方向の割当て

$$\mathrm{loc}(a(i, j)) = \mathrm{loc}(a(1, 1)) + \left\{\left(N-\frac{j}{2}\right)(j-1)+(i-1)\right\}S$$

(6) 対 称 行 列

三角行列と同様であることを説明する。

(7) 対角行列の要素 $a(i, i)$

$$\begin{bmatrix} a_{11} & & & 0 \\ & a_{22} & & \\ & & \ddots & \\ 0 & & & a_{NN} \end{bmatrix}$$

$$loc(a(i,i)) = loc(a(1,1)) + (i-1)S$$

最後に，配列への領域の割りつけとか，記憶変換関数の計算とかがどのような時点で行なわれるのかも説明しておくとよい。

たとえば，ＦＯＲＴＲＡＮのプログラムの場合には配列はすべて静的であって，コンパイル時（あるいは連結編集時。いずれにしても実行時以前）に割当てが行なわれる。記憶変換関数の計算が行なわれるのは，配列要素の参照を含む文が実行される時点である。ただし，添字が定数の場合には，コンパイル時に計算してしまうこともできる。

なお，受講者がＦＯＲＴＲＡＮを知っている場合には，コンパイラの目的コードの最適化に関連して，次の話題について説明しておくのも興味深いと思われる。すなわち，ＦＯＲＴＲＡＮでは添字式の形が変数の１次式以下に限られ，その形も制限されている。このことを利用すると，たとえば次のようなプログラム

```
      DO 10 I = 1, N
       .....
      A( 3*I+1 ) = .....          (*)
       .....
   10 .....
```

で（＊）の代入文の左辺の配列要素$A(3*I+1)$に対する記憶変換関数の計算を簡単化することができる。つまりIの値が１のときの添字式$3*I+1$の値４を保存しておけば，Iが２のときにはあらためて$3*I+1$の値を計算しなくても，保存しておいた値４にIの係数３を加えればよい。以後も同様に，この値を保存しておいて，次々に３を加えていくようにしておけば，効率のよい目的コードが得られる。

このような最適化は添字式として任意の形の式を許すと困難になる。一見意味不明のように思われるこの制限にも，このような意義が含まれている。

演習問題としては，次のようなものが考えられる。

① ３次元配列について残り４種類の割当ての記憶変換関数を考える。

② 上三角行列の割当てを考える。行方向と列方向の２種類がある。

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots\cdots & a_{1N} \\ & a_{22} & a_{23} & \cdots\cdots & a_{2N} \\ & & a_{33} & \cdots\cdots & a_{3N} \\ & 0 & & \ddots & \\ & & & & a_{NN} \end{pmatrix}$$

③ 三重対角行列の割当てを考える。

$$\begin{pmatrix} a_{11} & a_{12} & 0 & & & 0 \\ a_{21} & a_{22} & a_{23} & 0 & & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 \\ 0 & & \ddots & \ddots & \ddots & \\ \vdots & 0 & & & & 0 \\ & & & & a_{N-1,N-1} & a_{N-1,N} \\ 0 & & & 0 & a_{N,N-1} & a_{NN} \end{pmatrix}$$

④ 1次元配列$a(1)$, …, $a(N)$ で次のような記憶変換関数を考える。

$$\mathrm{loc}\big(a(i)\big) = \mathrm{loc}\big(a(1)\big) + q\,S$$

$$q \equiv i \mod p$$

ただし$p < N$はNと互いに素であるような（なるべくNに近い）整数とする。このときどのようなことが起こるかを考えさせる。

この問題はハッシングへのつなぎとなる。

4.3 疎な行列

はじめに，疎な行列はどのようなもので，どのようなところにあらわれるのかを説明する。例としては，有向グラフを行列表現するのが典型的であろう。特に，行列が大きくなるほど0要素の割合が増加する傾向があるので，そのまま2次元配列として格納したのでは，記憶域がたりなくなるという事情も説明しておく。

次に，そのような疎な行列の性質を生かした格納の手法をいくつか与える。

0でない要素の位置に規則性がある場合には，それを利用することができる。前回扱った三角行列，対角行列，三重対角行列などがこの例であることを復習する。

0でない要素の位置が不規則である場合の格納法としては，次の2通りを説明する。図4－1

のような例を用いて説明するとわかりやすい。

$$\begin{pmatrix} 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2 \\ 4 & 0 & 0 & 0 \\ 10 & 0 & 5 & 0 \end{pmatrix}$$

図４－１　疎な行列の例

(1)　疎な行列の１次元配列表現

０でない要素のそれぞれについて図４－２のような形の項目を一つずつ１次元配列に登録しておく方法である。０要素に対しては何も作らない。

形はきわめて簡単で，記憶域もとらないが，たとえば第 i 行 j 列の要素の値はいくつかといったことを調べるのにも効率がよくないことを説明する。

行番号	列番号	値

図４－２　０でない要素の表現方法１

図４－１の行列はこの表現方法を使うと，たとえば図４－３のように表現されることも確認させる。

(2)　疎な行列のリスト構造表現

この方法では，０でない要素のそれぞれについて図４－４のような形の項目を一つずつ作って，ポインタで結ぶ。さらに，各行・各列をあらわす循環リストのヘッドをさすポインタを１次元配列に入れておく。０要素に対しては何も作らない。

1	1	3
4	3	5
2	4	−2
4	1	10
3	1	4

図４－３　図４－１の１次元配列表現

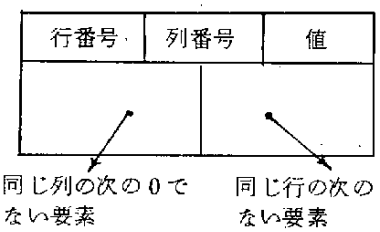


図４－４　０でない要素の表現方法２

この方法は，０でない要素一つずつについて，１次元配列表現よりはポインタ二つ分だけ記憶域を余分にとるが，ある位置の要素の値を知りたいときにはずっと効率がよいことを確認させる。

図４－１の行列をリスト構造表現したものを図４－５に示しておく。この図のように，この構造を説明するときには，０でない要素を表現する項目を行列の中のその要素に対応する位置

に書いてやると理解しやすい。

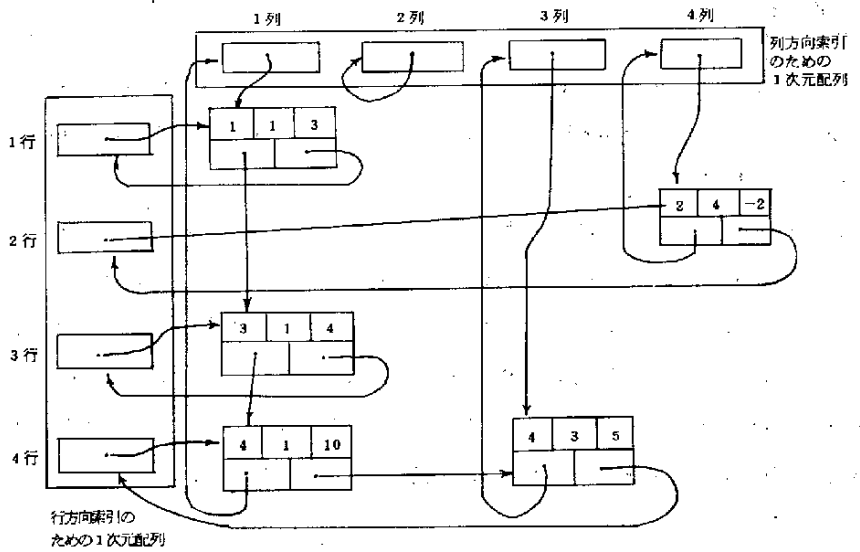


図4－5　図4－1のリスト構造表現

演習問題としては次のようなものが考えられる。

① 疎な行列を1次元配列表現するものとして，行列の加算・乗算のアルゴリズムを作る。

② 同様に，リスト構造表現した場合の加算・乗算のアルゴリズムを作る。

③ 配列表現・リスト構造表現のそれぞれについて，0要素がどれくらい以上あれば通常の2次元配列よりも得になるかを調べる。このときには，使用するスペースのみを考え，操作の効率は考えなくてよい。

④ リスト構造表現について次の問題を考える。

- 各項目に行番号・列番号を入れているが，どうせ行方向・列方向の索引によってわかるのだから，不要ではないか。
- 循環リストでなく，線型リストにしたら何か問題が生ずるか。

⑤ 1次元配列表現を採用したとして，操作の効率を向上させる手段を工夫する。

指導上の留意点

(1) 配列を記憶装置上でどう実現するかについては4.2で扱うので，4.1ではふれなくてよい。

(2) 4.1で配列の添字を0から始めるか1から始めるかは，受講者がどのような言語を習得しているかに応じて決めるとよい。

各種のプログラミング言語で配列をどう扱っているかについては第8章で扱う。

(3) 4.3の疎な行列は文献によって，「スパースな行列」，「がら空きマトリクス」，「疎行列」，「スパース行列」などの名で呼ばれているので，指導者は心得ておく必要がある。

参考文献

〔1〕D.E.Knuth, "The Art of Computer Programming, Vol.1", Fundamental Algorithms, Addison-Wesley, pp.240—251, 295—304, 1972

〔2〕E.Horowitz, S.Sahni, "Fundamentals of Data Structures", Computer Scinece Press, pp.40—76, 130—140, 1976

〔3〕情報処理学会編「情報処理ハンドブック」オーム社, pp.2.8—2.9, 2.11—2.12, 1972

〔4〕A.T.Berztiss, "Data Structures:Theory and Practice", Academic Press, 197.1．古川康一, 杉藤芳雄, 宮川正弘, 島田俊夫訳「データ構造Ⅰ; Ⅱ」コンピュータ・サイエンス研究書シリーズ; 日本コンピュータ協会, pp.309—315, 1974

〔5〕浦昭二編「データ構造」電子計算機基礎講座6, 共立出版, pp.35—39, 48—50,1974

〔6〕小林孝次郎「情報構造」計算機ライブラリ5, サイエンス社, 1977

〔7〕大附辰夫, 川北建次「スパース行列処理技法」情報処理, vol.17, (1): pp.42—49, (2): pp.142—154, (3): pp.229—238, 1976

〔8〕古川康一「データ構造」情報処理, vol.13, pp.302—310(Ⅰ), pp.554—562(Ⅱ), 1972

〔9〕G.G.Dodd, "Elements of data management systems", Computing Surveys, vol.1, pp.117—133, 1969

〔10〕P.A.V.Hall, "Computational Structures An Introduction to Non—numerical Computing", Computer Monographs, Macdonald and Jane's American Elsevier, 1975

〔11〕M.J.R.Shave, "Data Structures", McGraw—Hill, 1975

〔12〕H.S.Stone, "Introduction to Computer Organization and Data Structures", McGraw—Hill Computer Science Series, McGraw—Hill, 1972

〔13〕I.Flores, "Data Structure and Management", Prentice—Hall, 1970．久保寛彦訳「データの構造」,「データの管理」竹内書店, 1972

第5章　論理構造――一般の構造・リスト処理言語

キーワード

一般の構造（general structure, network structure）

一般の構造の配列表現（array representation of network structures）

一般の構造の木表現（tree representation of network structures）

一般の構造の三つ組表現（3-tuple representation of network structures）

（節の）次数（degree），入次数（in-degree），出次数（out-degree），有向グラフ（directed graph），平衡有向グラフ（balanced directed graph），連結グラフ（connected graph），ループ（loop, cycle），路（path）

リング構造（ring structure）

異質のフィールドや節を持つ多重連結構造（multilinked structures with heterogeneous fields and/or nodes）

プレックス（plex），可変長節（variable-size node），流れ図（flowchart），プログラム（program），図形（picture），輸送路（transportation network），化学式（chemical formula），電気回路（electric circuit）

LISP 1.5, SLIP, L^6, SNOBOL

目　標

まず，一般の構造を表現するいろいろの手法とその長所・短所を理解させる。また，それらの表現に対する操作アルゴリズムを考えさせる。

このときグラフ理論の用語のうちの基本的なものをコンピュータ上での表現と関連づけて紹介する。

次に，図形など，複雑な構造をもったデータの取り扱いにあたって，異質のフィールドや節を持った多重連結構造が出現する必然性を理解させるとともに，そうした構造を実現するにあたって問題となる点を説明する。

さらに，一般の構造の扱いが必要となる事例のうちで代表的なものについて，本来固有のものとして持っている論理構造，利用の環境，表現方法の例，その方法による表現例，その表現方法

の利点・欠点，操作アルゴリズムを工夫させる。

最後に，特徴のあるデータ構造を持つ言語について，そのデータ構造，データ構造の操作例とその記述方法，プログラム例を紹介する。

内　　容

5.1　一般の構造とグラフ理論の用語

はじめに，線型リストや木構造では表現できず，有向グラフとして表現するのが適したような構造を持ったものの例をいくつか簡単に示す。たとえば，図５－１のような道路網を示すのがよいであろう。これを利用して，この論理構造を配列で表現する手法を２通りほど説明する。

都市間のつながり具合だけに着目し，距離や道路の特徴をすべて無視してよい場合には，各節を行と列にとり，行方向の節から列方向の節に道があるとき１，そうでないとき０を要素とするような行列を作って，次のように表現することができることを述べる。

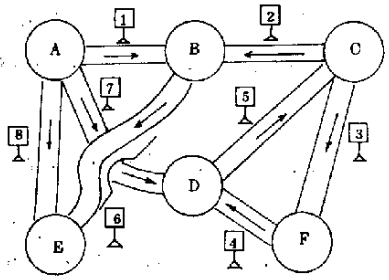


図５－１　　道　路　網

	A	B	C	D	E	F
A	0	1	0	1	1	0
B	0	0	0	0	1	0
C	0	1	0	0	0	1
D	0	0	1	0	0	0
E	0	0	0	0	0	0
F	0	0	0	1	0	0

道路の特徴なども捨てることができない場合には，上の行列で１が入っているところを道路のコードにして表現する方法も考えられるが，次のような行列で表現することもできることを説明する。すなわち，各節を行方向にとり，列方向にはそれぞれの道を対応させる。さらに，節から道が出ているときには＋１，入っているときには－１を入れて，次のような行列を作ればよい。

	1	2	3	4	5	6	7	8
A	＋1	0	0	0	0	0	＋1	＋1
B	－1	－1	0	0	0	＋1	0	0
C	0	＋1	＋1	0	－1	0	0	0
D	0	0	0	－1	＋1	0	－1	0
E	0	0	0	0	0	－1	0	－1
F	0	0	－1	＋1	0	0	0	0

この表現によると，列の要素の和が0になり，また，自己へのループが表現できないこともつけ加えておくとよい。

さらに，このように表現すると，要素に0の多い行列，すなわち疎な行列となり，4.3で述べたようなリスト構造表現などが使えることも説明しておくとよい。

出発地を中心とした構造を考えると，図5－2のような木構造表現も可能である。ただ，一般にはこの表現方法はあまり使われないので，説明を省略してもよい。

（A，V，O）の三つ組を使った表現も可能である。ただし，三つ組の構造は第　6章で扱うので，ここでは，物理的な構造の説明は無理で，論理的に次のような三つ組の集合として表現できるということだけを示しておけばよい。

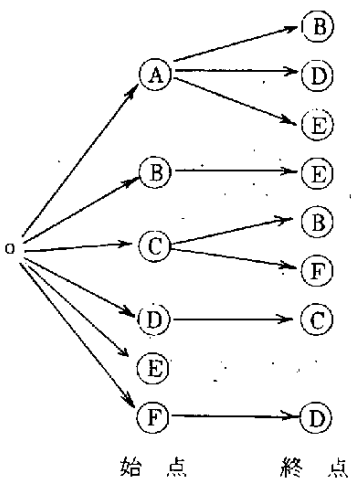


図5－2　図5－1の木構造表現

（始点，1，A），　（終点，1，B），
（始点，2，C），　（終点，2，B），
（始点，3，C），　（終点，3，F），
（始点，4，F），　（終点，4，D），
（始点，5，D），　（終点，5，C），
（始点，6，B），　（終点，6，E），
（始点，7，A），　（終点，7，D），
（始点，8，A），　（終点，8，E），

ポインタを利用して，もっと直接的な形でリスト構造として表現することも可能である。たとえば，わかりやすい表現として，図5－3のようなものを最初に示すとよい。

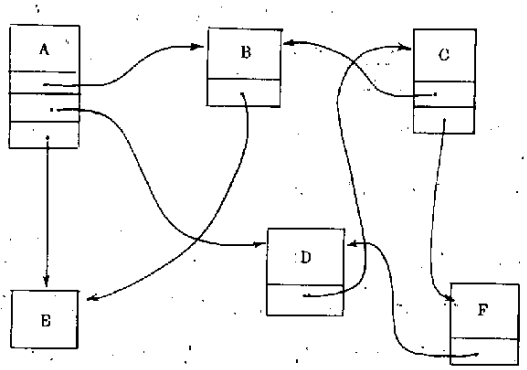
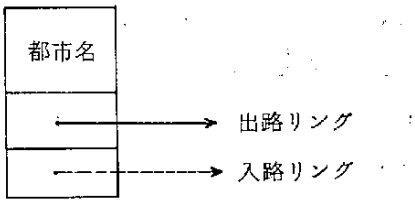
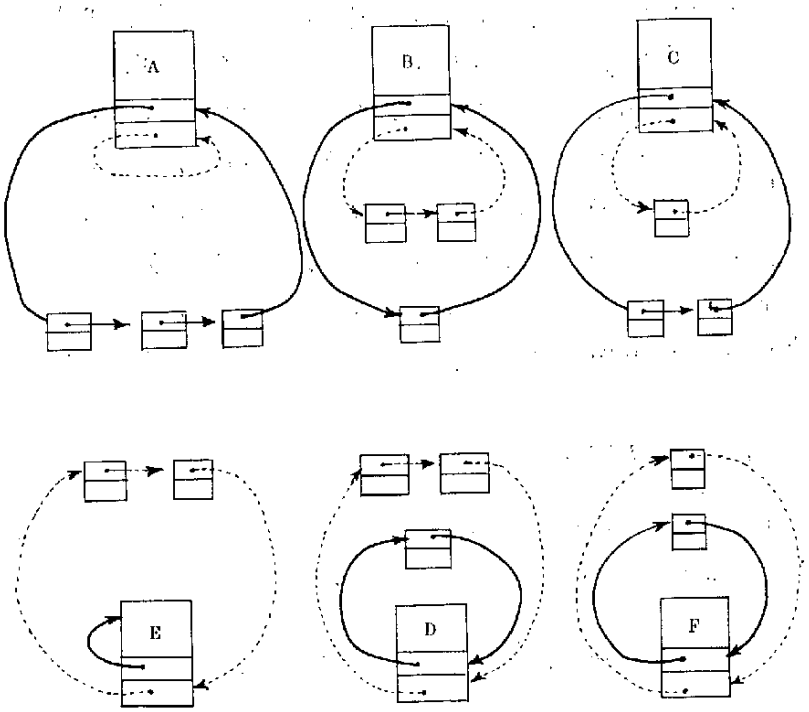


図5－3　図5－1のリスト構造表現（その1）

この構造では，都市を表現したブロックの大きさがまちまちになる。このようなことから生ずる記憶域管理上の問題を簡単に述べ，これを解決する手段として，図５－４のようなリング構造による表現も可能であることを示す。この表現では，ブロックの大きさが固定長のもの２種類だけでよいことを確認しておく。このような図を示すときには，ポインタの一つ一つを受講者たちに確認させながら色別けをして書いていくとよい。



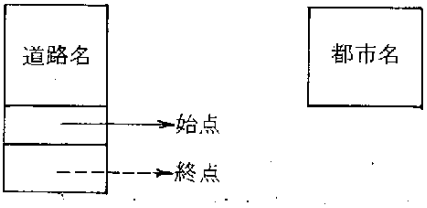
(a) 都市ブロックの構造



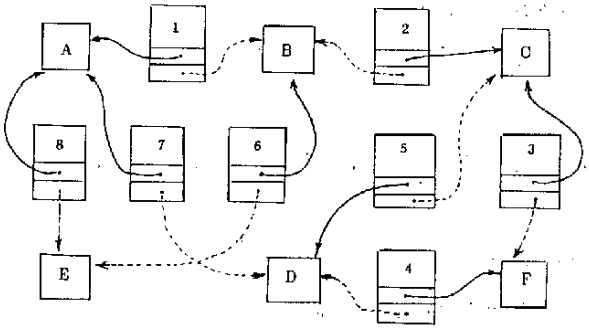
(b) リ ン グ 構 造

図５－４　　図５－１のリング構造表現

次に，図５－３は都市ブロックを中心にして，これを道をあらわすポインタでつないだが，視点を変えて，道を中心にしたリスト構造として表現すると，図５－５のような構造も作れることを説明する。これも，何を解こうとするかによって，論理構造がかわってくる例である。ただし，このままでは道の表を作って，そこに始点・終点を登録しておくのと大差ない。図５－４のよう



(a) 道路ブロックと都市ブロック



(b) リ ス ト 構 造

図５－５　　図５－１のリスト構造表現（その２）

な都市中心の構造を組合わせた場合に特に意味があることをつけ加えておくべきであろう。図５－５はむしろ，一つの構造の中にいろいろの種類のブロックが含まれた例として示すべきである。

最後に，一般的傾向として，いろいろのポインタをつけて構造を複雑にした場合と，そうでない場合について，表５－１のような性質があることを確認させるとよい。この表は2.3で示したものと同じであるが，この時点で再掲しておくことは意義深い。

表５－１　　構造の複雑さと特徴

構造が簡単 ⟷ 構造が複雑	
記憶域は少なくてすむ	ポインタの分だけ余分の記憶域が必要
アクセスに時間がかかる	アクセスは高速でできる
挿入・削除は簡単なアルゴリズムでできる	挿入・削除に複雑な手順が必要

演習問題としては，次のようなものが考えられる。

- ここで示した構造について，関係の変化，ブロックの追加・削除のアルゴリズムを工夫する。
- 図５－１以外の有向グラフについて，各種の構造がどうなるか考える。

・ ここで示した構造以外の構造を工夫する。

5.2 一般の構造の応用

この節は，2.4や3.5に続く内容を持っている。

以下のものは，その論理的な構造を有向グラフやグラフによって表現することができる。それぞれについて，それが使われる環境を設定して，それを論理構造として表現させ，さらに，どのような物理構造として実現させるのが望ましいかを検討させる。主記憶装置だけでなく。補助記憶装置の使用も考慮に入れた考察が望ましい。

次のようなテーマが考えられる。

① 流れ図，プログラム

会話型式でプログラムを入力，修正し，実行を行なうようなシステムの基本設計

② 図形〔11〕−〔18〕

図形表示装置やライトペンなどの装置を使い，表示画面上で図形を組立てたり修正したりするシステムの基本設計．特にデータ構造を中心に設計する。

③ 輸送路〔7〕

最短路問題，循回セールスマン問題

④ 化学式〔10〕

化学物質のデータベースとその検索システムの基本設計

⑤ 電 気 回 路

会話型式で電気回路を入力・修正し，その特性のシミュレーションを行なうようなシステムの基本設計

5.3 リスト処理言語

題材として適当なプログラムが掲載されている参考文献としては，次のようなものがある。

(1) ＬＩＳＰ 1.5

(a) 〔42, p. 456〕

・ リストを逆にする

・ 二つのリスト構造が等しいかどうかテストする。

(b) 〔47, pp. 139−318〕(いずれもかなり長いが，ていねいな解説がついている)

・ 分 数 計 算

・ 行列計算用プログラム言語とそのプロセッサ

・ 直交多項式生成のプログラム

- ルジャンドル多項式による展開
- 逆関数を定義するプログラム
- Clebsch-Gordan 係数

(c) 〔44, pp. 209-212〕

記号微分

(d) 〔39, pp. 103-108〕

- 集合演算（ある記号が記号の集合のなかに含まれているかどうかを知る）
- 多項式の微分を求める

(2) SLIP

(a) 〔39, p. 139〕

あるリストと全く同じリストを新しく作る

(b) 〔39, p. 151〕

セルの貯蔵所とパブリック・リストを作成する

(c) 〔39, pp. 154-158〕

次の行列の基本演算を行なう。

- スカラー積 S * A
- 加　法　A + B
- 乗　法　A * B
- べき乗　A^n
- 転　置　A^T

(3) L^6

(a) 〔42, p. 449〕

- バグWによって指され，A-フィールドによって結びつけられるリストを逆にする。
- 二つのリスト構造が等しいかどうかテストする。

(4) SNOBOL

① SNOBOL3

(a) 〔42, p. 489〕

文字列を読み込んで，各行がN文字以内に納まるように編集し，行中に適宜空白を挿入して行末をそろえてから印刷する。

② SNOBOL4

(a) 〔65, pp. 206-234〕

- トランプ・ゲームのブリッジを行なう。

- 階乗表を作る。
- 超高精度演算を行なう。
- トポロジカル・ソートを行なう。
- SNOBOL の文の文法チェックを行なう。
- 歌詞を作る。

(b) 〔44, pp.197−200〕

文字列を与え，その文字列からN個の文字をとってきたときの可能な組合せをすべて出力する。

指導上の留意点

5.3のSNOBOL では，データ構造はとくに決められていないので，特定のプロセッサで使われている構造を，そのことを明確にして紹介するか，あるいは，適当な構造を工夫させる。

参 考 文 献

〔1〕 D.E.Knuth, "The Art of Computer Programming, Vol.1" Fundamental Algorithms, Addison-Wesley, pp.355−359, 362−399, 1972

〔2〕 A.T.Berztiss, "Data Structures: Theory and Practice", Academic Press, 1971. 古川康一, 杉藤芳雄, 宮川正弘, 島田俊夫訳「データ構造 I, II」コンピュータ・サイエンス研究書シリーズ, 日本コンピュータ協会, pp.119−165, 1974

〔3〕 浦昭二編「データ構造」, 電子計算機基礎講座6, 共立出版, pp.24−29, 96−102, 181−281, 226−249, 1974

〔4〕 大須賀節雄「データ構造」bit, 共立出版, vol.8, pp.999−1005 (⑤: データ構造の形態1), pp.1041−1047 (⑥: データ構造の形態2), pp.1303−1318(⑧: まとめ), 1976

〔5〕 P.A.V.Hall, "Computational Structures−An Introduction to Non−numerical Computing", Computer Monographs, Macdonald and Jane's American Elsevier, pp.3−8, 26−34, 1975

〔6〕 I.Flores, "Data Structure and Management", Prentice−Hall 1970. 久保寛彦訳「データの構造」竹内書店, pp.37−65, 123−164, 1972

〔7〕 E.Horowitz, S.Sahni, "Fundamentals of Data Structures", Computer Science Press, pp.282−334, 1976

〔8〕 C.A.Lang, J.C.Gray, "ASP—aring implemented associative structure package", Communications of the ACM, vol.11, pp.550—555, 1968

〔9〕 小林孝次郎「情報構造」計算機ライブラリ5, サイエンス社, pp.62—79, 1977

〔10〕 R.E.Stobaugh, "Chemical structures: computer handling", Encyclopedia of computer Science and Technology, vol.4, Macel Dekker, pp.478—509, 1976

〔11〕 W.M.Newman, R.F.Sproull, "Principles of Interactive Computer Graphics", McGraw-Hill Computer Science Series, McGraw—Hill, pp.374—384, 1973
（近く翻訳が出る予定：大須賀節雄「コンピュータ・グラフィックス」科学技術出版社）

〔12〕 J.C.Gray, "Compound data structures for computer aided design, a survey", Proceedings of ACM 22nd National Conference, pp.355—365, 1967

〔13〕 穂坂衛「データ構造」電子通信学会誌, vol.53, №4, pp.515—521, 1970

〔14〕 古川康一「コンピュータ・グラフィックスにおけるデータ構造の問題」情報処理, vol.11, pp.523—532, 1970

〔15〕 三重野博司編「自動設計とグラフィックス」コンピュータ全書5010, 大河出版, pp.276—294, 1971

〔16〕 E.M.Thomas, "GRASP—a graphic service program", Proceedings of ACM 22nd National Conference, pp.395—402,1967

〔17〕 R.Williams, "A survey of data structures for computer graphics systems", Computing Surveys, vol.3, pp.1—21, 1971

〔18〕 穂坂衛「コンピュータ・グラフィックス」コンピュータ・サイエンス・シリーズ, 産業図書, pp.208—225, 1974

〔19〕 古川康一「データ構造」情報処理, vol.13, pp.302—310(I), pp.554—562(II), 1972

〔20〕 G.G.Dodd, "Elements of data management systems-", Computing Surveys, vol.1, pp.117—133, 1969

〔21〕 M.J.R.Shave, "Data Structures", McGraw—Hill, 1975

〔22〕 H.S.Stone, "Introduction to Computer Organization and Data Structures", McGraw—Hill Computer Science Series, McGraw—Hill, 1972

・ グラフ理論に関するもの

〔23〕小野寺力男「グラフ理論の基礎」数学ライブラリー6，森北出版

〔24〕F.Harary, "Graph Theory", Addison—Wesley, 1969．池田貞雄訳「グラフ理論」共立出版，1971

〔25〕尾崎弘，白川功「グラフとネットワークの理論」コロナ社，1973

〔26〕C.Berge, "Graphes et Hypergraphes", Dunod, 1970．伊理正夫，伊理由美，岩坪秀一，小林欣吾，佐藤創，星守訳「グラフの理論Ⅰ,Ⅱ,Ⅲ」サイエンスライブラリー数学15，16，17，サイエンス社，1976

〔27〕服部嘉雄，小沢孝夫「グラフ理論解説」昭晃堂，1974

〔28〕小野寺力男「グラフ理論の展開と応用」数学ライブラリー30，森北出版，1968

〔29〕R.G.Busacker, T.L.Saaty, "Finite Graphs and Networks：An Introduction with Applications", McGraw—Hill, 1965．矢野健太郎，伊理正夫訳「グラフ理論とネットワーク／基礎と応用」培風館，1970

〔30〕奥野隆史，高森寛「点と線の世界」三共科学選書5，三共出版，1976

〔31〕R.Bellman, K.L.Cooke, J.A.Lockett, "Algorithms, Graphs and Computers", Mathematics in Science and Engineering 62, Academic Press, 1970．渡辺 茂訳「計算機のためのグラフとアルゴリズム」共立出版，1972

〔32〕伊理正夫「グラフの理論とその応用」電子通信学会誌，vol.54, 55, №1～5,1971, 1972

〔33〕野崎昭弘「計算数学セミナー」数学セミナー増刊，数学セミナーリーディングス，pp.13—18, 1976

〔34〕C.Flament, "Applications of Graph Theory to Group Str-Structure", Prentice-Hall, 1963．山本国雄訳「社会行動の数理解析 —グラフ理論と社会構造」紀伊国屋書店，1974

〔35〕本間龍雄「グラフ理論入門 — 点と線の数学」ブルーバックスB255，講談社，1975

〔36〕C.L.Liu, "Introduction to Combinatorial Mathematics", McGraw—Hill, 1968．伊理正夫，伊理由美訳「組合せ数学入門Ⅰ，Ⅱ」共立全書541, 542，共立出版，pp.183—334, 1972

〔37〕野口広，釜江慶子「グラフ理論」数理科学シリーズ6，筑摩書房，1974

〔38〕伊理正夫，古林隆「ネットワーク理論」ORライブラリー12，日科技連，1976

・ リスト処理言語全般に関するもの

〔39〕 浅井清，中村康弘，石黒美佐子，稲見泰生，斉藤直之「記号処理言語」ＩＣＳライブラリ3，総合図書，1971
（ＩＰＬ−Ｖ，ＬＩＳＰ，ＳＬＩＰ，ＦＬＩＰＳ，ＳＰＭ）

〔40〕 藤野精一「記号処理言語」情報科学講座Ｃ・1・0・3，共立出版，1975
（ＬＩＳＰ，L^6，ＳＰＲＩＮＴ２）

〔41〕 情報処理学会編「情報処理ハンドブック」オーム社，１９編，1972
（ＬＩＳＰ，ＳＮＯＢＯＬ，L^6，ＳＬＩＰ）

〔42〕 J.E.Sammet, "Programming Languages:History and Fundamentals", Series in Automatic Computation,Prentice−Hall 1969.竹下 亨訳「プログラミング言語ハンドブック」日本経営出版会，1971
（ＩＰＬ−Ｖ，L^6，ＬＩＳＰ 1.5，ＣＯＭＩＴ，ＳＮＯＢＯＬ，ＴＲＡＣ，ＡＭＢＩＴ，ＴＲＥＥＴ，ＣＬＰ，ＣＯＲＡＬ，ＳＰＲＩＮＴ，ＬＯＬＩＴＡ）

〔43〕 J.M.Foster, "List Processing", Macdonald Computer Monographs 1, 1967．西村敏男訳「リスト処理」サイエンス・マクドナルド・コンピュータシリーズ1，サイエンス社，pp.53−66, 1973
（ＩＰＬ−Ｖ，ＬＩＳＰ，ＳＬＩＰ，ＦＬＰＬ，ＣＯＭＩＴ）

〔44〕 浦昭二編「データ構造」電子計算機基礎講座6，共立出版，pp.181−226, 1974
（ＣＯＭＩＴ，ＳＮＯＢＯＬ，ＬＩＳＰ，ＳＬＩＰ，L^6）

〔45〕 D.G.Bobrow, B.Raphael, "A comparison of list processing languages", Communications of the ACM, vol.7, pp.231−240, 1964
（ＣＯＭＩＴ，ＩＰＬ，ＬＩＳＰ，ＳＬＩＰ）

・ ＬＩＳＰに関するもの

〔46〕 中西正和「ＬＩＳＰ入門 ― システムとプログラミング」コンピュータサイエンス大学講座，近代科学社，1977

〔47〕 雨宮綾夫編「ＬＩＳＰとその応用例」コンピュータ・サイエンス・シリーズ，産業図書，1972

〔48〕 E.C.Berkeley, D.G.Bobrow, "The Programming Language LISP: It's Operation and Applications", M.I.T.Press, 1966

〔49〕 藤野喜一，松田正一，篠沢昭二他「情報処理言語ＬＩＳＰ」日本電子工業振興協会,1965

〔50〕 中西正和「ＫＬＩＳＰ説明書」計算機シリーズ7，慶応義塾大学情報科学研究所，１９７２

〔51〕 C.Weissman, “ＬＩＳＰ 1.5 Primer”, Dickenson, 1967．小林 達訳「ＬＩＳＰ入門」ダイヤモンド社，１９７１

〔52〕 後藤英一「ＬＩＳＰ入門」bit，共立出版，vol.6, 1974, pp.7—13（①：マッカーシーの条件式とM式），pp.109—117（②：関数の帰納的定義），pp.174—184(③：帰納的定義のＦＯＲＴＲＡＮ処理　プリプロセッサ，ＦＯＲＴＲＡＮ），pp.258—265（④：Lispのデータ言語S式），pp.338—344（⑤：ＭＳ変換，ＬＩＳＰの万能関数）pp.418—423（⑥：Lispの標準関数），pp.502—508（⑦：連想リストと属性リスト），pp.582—588（⑧：関数引数の処理，基本集合演算の高速化），pp.942—949（⑨：ＨＬＩＳＰ），pp.1022—1029（⑩：連想計算機能とＨＬＩＳＰ），pp.1106—1113（⑪：ガーベッジ・コレクター廃品回収），pp.1188—1194（⑫：仮想記憶と連想的処理の効用），vol.7, 1975, pp.6—11（⑬：Backtrack法とLisp），pp.94—100（⑭：Lispの入出力プログラム）。

〔53〕 黒川利明「ガーベジ・コレクション解説」bit，共立出版，vol.9, pp.63—65, 1977

〔54〕 中西正和「ＬＩＳＰを中心としたリスト処理言語」情報処理，vol.14, pp.961—967, 1973

〔55〕 J.McCarthy, P.W.Abrahams, D.J.Edwards, T.P.Hart, M.I.Levin, “ＬＩＳＰ1.5 Programmer's Manual”, ＭＩＴ Press, 1962

〔56〕 W.D.Maurer, “The Programmer's Introduction to ＬＩＳＰ”, Macdonald Computer Monographs 6, 1972．佐藤浩史訳「プログラマのためのＬＩＳＰ入門」サイエンス・マクドナルド・コンピュータシリーズ6，サイエンス社，1974

〔57〕 藤野精一「リスト処理と言語解析」プログラミング演習シリーズ3，朝倉書店，pp.31—130, 1970

〔58〕 小林孝次郎「情報構造」計算機ライブラリ5，サイエンス社，pp.80—114, 1977

・ ＳＬＩＰに関するもの

〔59〕 J.Weizenbaum, “Symmeric list processor”, Communications of the ACM, Vol.6, pp.524—536, 1963

・ L^6 に関するもの

〔60〕 C.K.Knowlton, "A programmer's description of L ", Communications of the ACM, vol.9, pp.616—625, 1966

〔61〕「T—L^6 によるリスト処理入門」電子計算機セミナー・テキスト，日本科学技術連盟，1970

〔62〕「T—L^6 使用者のための便覧」日本科学技術連盟，1970

・ SNOBOLに関するもの

〔63〕 A.Forte, "SNOBOL 3 Primer", M.I.T.Press, 1967

〔64〕 D.J.Farber, R.E.Griswold, I.P.Polonsky, "The SNOBOL 3 programming languages", The Bell System Tecnical Journal, vol.45, pp.895—944, 1966

〔65〕 R.E.Griswold, J.F.Poage, I.P.Polonsky, "The SNOBOL 4 Programming Languages", Prentice—Hall, 1968, 2nd ed, 1971

〔66〕 R.E.Griswold, M.T.Griswold, "A SNOBOL 4 Primer", Prentice—Hall, 1973

〔67〕 J.F.Gimpel, "Algorithms in SNOBOL 4", John Wiley & Sons, 1976

第6章　物　理　構　造

キー・ワード

パック（pack），アンパック（unpack），部分語（partword），部分語のアドレス付け（part-word addressing），順次割当て（sequential addressing），スタック（stack），キュー（queue），デキュー（dequeue），ポインタ（pointer），連結割当て（linked allocation），使用可能空間リスト（available list），ちり集め（garbage collection），高次元配列（higher dimensional array），木（tree），順次表現（sequential representation），結合行列（connection matrix），ネットワーク構造（network structure），構造データ（structure），多重連結構造（multilinked structure），分散記憶（scatter storage），ハッシュ表（hash table），ハッシング関数（hashing function），衝突（collision），直接アドレス付け（direct addressing），間接アドレス付け（indirect addressing），即値アドレス付け（immediate addressing），連想記憶（associative memory），内容によるアドレス付け（content addressing）

目　　標

データの論理構造を記憶装置中に保持させながら，いかにデータを記憶するかを指導する。本章では，主記憶装置中での記憶の物理構造を対象として説明するが，その多くは磁気ディスク・ファイルなどの中での記憶の構造に適用できる。記憶装置の割当ては，基本的には，順次割当てと連結割当てしかない。スタック，キュー，デキュー，配列，木，構造（多重連結構造）のようなデータに具体的にどのように記憶装置を割当てるか。また，割当て方によってそのデータに対する基本的な操作がどのようになるかを理解させる。さらに，ハッシュ表の原理，作り方，使い方と種々の番地付けについて理解させる。

本章では，後続の章で扱う実際的な技術の基礎になる部分を理解させればよい。

内　　容

6.1　記憶装置の割当て

データの論理構造を物理的な記憶装置中に保持しながら記憶するための，基本的な記憶装置の

割当てはついて，以下の項目を説明する。

(1) データのパック

データを圧縮して，記憶装置中の小さな空間にとじこめることをパック（pack）といい，逆に，パックされたデータを，使いやすいようにもとの形にもどすことを，アンパック（unpack）という。

① 個々に独立して意味をもつ論理値（0もしくは1の1ビット）をまとめて1ワード中に記憶する。

② 個々に意味をもつ数や文字を1ワード中にまとめて記憶する。

③ 入力装置から文字コード（1文字を8ビットにコード化した形式）で読んだ10進数を，1桁が4ビットの2進化10進数に変換して記憶する。

④ 多くの空白文字などを含む字データから，その空白文字を符号化し取り除いて記憶する。

⑤ スパース行列などにおいて，0でない要素だけに着目してリスト構造で記憶する。

(2) 部分語

データが，記憶装置中で2ワード以上にまたがって記憶されることがある。そのとき，データの一部を記憶するワードを，部分語（part-word）という。

部分語の場所を識別するための番地付けを部分語のアドレス付け（part-word addressing）という。データがどのような記憶割当て方法で記憶されているか，すなわち，順次割てか，連結割当てか，などによって，部分語のアドレス付けは異なる。

(3) 順次割当て

論理的な構造をもつデータに対して，記憶装置の連続した空間を割当てることを順次割当て（sequential allocation）という。論理的な構造をもつデータにどのような記憶割当てを行なうかは，そのデータをどのように操作するかを考え，その最適化を図るように決定しなければならない。

スタック（stack），キュー（queue），デキュー（dequeue）など線型リスト（linear list）のもっとも簡単，かつ自然な記憶割当ては，順次割当てである。たとえば，線型リストXのj番目の要素の主記憶中での番地$A(X[j])$は，

$$A(X[j])=A(X[j-1])+C$$
$$=A_0+Cj$$

ただし，

C：1つの要素の記憶に必要なワード数

A_0：仮想の要素X[0]のアドレス

となる。

スタックは，順次割当てで記憶すると操作が非常に簡単になる。Tをスタック・ポインタ

（stack pointer）とすれば，スタックXに新しい要素Yを入れる操作は

$$T:=T+1;\quad X[T]:=Y;$$

となり，スタックから取り出してYに代入する操作は

$$Y:=X[T];\quad T:=T-1;$$

でよい。

キューとデキューを順次割当てで記憶しているときのポインタ操作は，前方ポインタ（front pointer）Fと後方ポインタ（rear pointer）Rによって，次のように行なえばよい。すなわち，キューXの後方に要素を挿入するときは，

$$R:=R+1;,\quad X[R]:=Y;$$

前方の要素を取り出すときは

$$F:=F+1;\quad Y:=X[F];$$
$$\text{if } F=R \text{ then } F:=R:=0;$$

となる。

しかし，スタックやキューの割当てた記憶が不足しているにもかかわらず挿入しようとする，あふれ（overflow）や，スタックやキューがからであるにもかかわらず取り出そうとする，空読み（underflow）が生じることがある。これを考慮すれば，前述のポインタ操作は次のようになる。ただし，Mはスタックやキューの容量であり，X[M]の次はX[1]が続くように環状になっているとする。

スタックへの挿入

```
                    X:=Y
T:=T+1;
if T>M    then    OVERFLOW;
X[T]:=Y;
```

スタックからの取出し

```
                    Y:=X
if T=0    then    UNDERFLOW;
Y:=X[T];
T:=T-1;
```

キューへの挿入

```
                    X:=Y
if R=M    then    R:=1
          else    R:=R+1;
if R=F    then    OVERFLOW;
X[R]:=Y;
```

```
キューからの取出し                    Y:=X
    if R=F    then   UNDERFLOW;
    if F=M    then   F:=1
              else   F:=F+1;
    Y:=X〔F〕;
```

記憶装置中に二つ以上のスタックやキューを記憶しているとき，そのうちの一つがあふれを生じても，全体としてみれば記憶装置に余裕があることが多い。そのため，割当て方を変えて互いに都合をつけ合うようにする。

(4) 連結割当て

物理的に分散している記憶場所に記憶するよう記憶装置を割り当て，それをポインタで連結して使用する割り当て方を連結割当て（linked allocation）という。図6－1は順次割当てと連結割当ての違いを示している。

順次割当て

アドレス	内容
A_0+C	要素 1
A_0+2C	要素 2
A_0+3C	要素 3
A_0+4C	要素 4
A_0+5C	要素 5

連結割当て

アドレス	内容	
A	要素 1	B
B	要素 2	C
C	要素 3	D
D	要素 4	E
E	要素 5	∧

図６－１　順次割当てと連結割当ての比較

図6－1において，アドレスA，B，C，D，Eは記憶装置のどこであっても構わない。また∧は連結の終りを表わしている。このように連結された表は図６－２のように表わすことができる。

図６－２　　図６－１の連結割当ての構造

順次割当てと比較しながら連結割当ての特徴を説明する。

① 連結割当てではポインタのための余分の記憶装置を必要とする。しかし，たとえば，１ワード中にポインタ部を記憶するための余裕があったりして，この点は必ずしも欠点とはいえない。さらに，順次割当てに比べて，使わない小さな記憶空間が散在することがないなど，

記憶装置の利用効率はよくなる。

② 連結リスト中の要素の削除が容易である。たとえば，図6－1において，要素3の削除のためには，要素2のポインタをDに変えるだけでよいが，順次割当てでは，要素4，5を上に移動させなければならない。

③ ②と同様に，要素の挿入が容易である。要素2と3の間に新しい要素2.5を挿入したければ，要素2のポインタを要素2.5のアドレスに変え，要素2.5のポインタをCにすればよい。

④ 連結割当てでは，リスト中の任意の要素の参照は，ポインタをたぐる必要があり，順次割当てに比べて，多くの処理時間を要する。

⑤ 二つ以上のリストに分割したり，二つ以上のリストを一つにまとめたりすることが容易である。

⑥ 可変長リストが生じたり消滅したりする環境で記憶装置を有効に利用することができる。

⑦ 間接アドレス付けが使えれば，リストを次々にたぐる速度は，順次割当てよりも速い。

連結割当てでは，通常未使用の記憶装置の空間が，使用可能空間リスト（ available list ）としてまとめられている。このリストをAVAIL リストと表わし，その先頭要素のアドレスは連結変数（ link variable）AVLで示されているとする。AVAIL リストから一つの要素（分の記憶装置）を取り出し，その場所を連結変数Xで示すようにするときは，

```
if  AVL=Λ  then   OVERFLOW
           else   X:=AVL, AVL:=L(AVL);
```

とすればよい。この操作をX⇐AVAILと略記する。ただし，L（AVL）はAVLで示されたAVAILリストの要素のポインタ部である。逆に，連結変数Xで示された要素をリストから削除してAVAILリストに加える操作は

```
L(X):=AVL;
AVL:=X;
```

となる。これを，簡単に

AVAIL ⇐ X

で表わす。

不要になった記憶装置を集めてAVAILリストに加えることが行なわれる。このような操作をちり集め（ garbage collection ）という。

スタックを連結割当てで記憶しているとき，Yをスタックに挿入する操作は

P⇐AVAIL; D(P):=Y;

$$L(P) := T ; \quad T := P ;$$

となる。ただし，Pは補助ポインタ変数，$D(P)$，$L(P)$はそれぞれPで示された要素のデータ部，ポインタ部，Tはスタックの先頭の要素を示すポインタ変数である。逆にスタックから要素を取り出してYに与える操作は

```
if T= Γ  then    UNDERFLOW
else begin  P:=T ; T:= L(P) ;
            Y:=D(P) ; AVAIL:=P  end;
```

となる。

キューを連結割当てで記憶すると，キューの後方へのYの挿入の操作は

$$P \leftarrow \mathrm{AVAIL} ; \quad D(P) := Y ; \; L(P) := \Gamma ;$$
$$L(R) := P ; \; R := P ;$$

となり，前方からの要素の取出し操作は，

```
if F=    then   UNDERFLOW
     else begin P:=F, F:=L(P),
          Y:=D(P), AVAIL⇐P ;
          if F= Γ then R:=A(F)    end;
```

となる。ただし，F，Rはそれぞれキューの前方ポインタ，後方ポインタであり，$A(F)$はFのアドレスである。

(5) 配列の記憶

2次元以上の高次元配列（higher dimensional arry）は線型リストの一般化したものであり，通常，順次割当てによって記憶装置を割り当てる。

k次元配列Bの要素$B[I_1, I_2, \cdots, I_k]$のアドレス$A(B[I_1, I_2, \cdots, I_k])$は，$B[0, 0, \cdots, 0]$のアドレス$A(B[0, 0, \cdots, 0])$をもとにして，

$$\begin{aligned} A(B[I_1, I_2, \cdots, I_k]) &= A(B[0, 0, \cdots, 0]) \\ &\quad + c(d_2 + 1)\cdots(d_k + 1) I_1 + \cdots \\ &\quad + c(d_k + 1) I_{k-1} + c I_k \\ &= A(B[0, 0, \cdots, 0]) + \sum_{1 \le r \le k} a_r I_r \end{aligned}$$

ただし，

$$a_r = c \prod_{r < s \le k} (d_s + 1)$$

と計算される。cは一つの要素の記憶に必要なワード数であり，Bの各要素は，I_k，$\cdots$，I_2，I_1の順に変化するような順序で記憶されるものとしている。また，$d_1, d_2, \cdots, d_k$は

それぞれ対応する次元の大きさである。すなわち，

$$0 \leq I_1 \leq d_1,\ 0 \leq I_2 \leq d_2,\ \cdots,\ 0 \leq I_k \leq d_k$$

である。

高次元の配列を連結割当てで記憶することがある。各要素ごとに必要数のポインタ部を設け，値ごとのリストをそのポインタ部を用いて結ぶことによって記憶する。たとえば，人間ごとに一つの要素を対応させ，その要素に三つのポインタ部を設ける。それらは性別，年令，髪の色を与えるためのものである。男性リストをつくり，男性だけを性別ポインタ部で結ぶ。女性の性別ポインタ部を結び女性リストをつくる。年令，髪の色についても同様なリストをつくる。このようすを図6－3に示す。

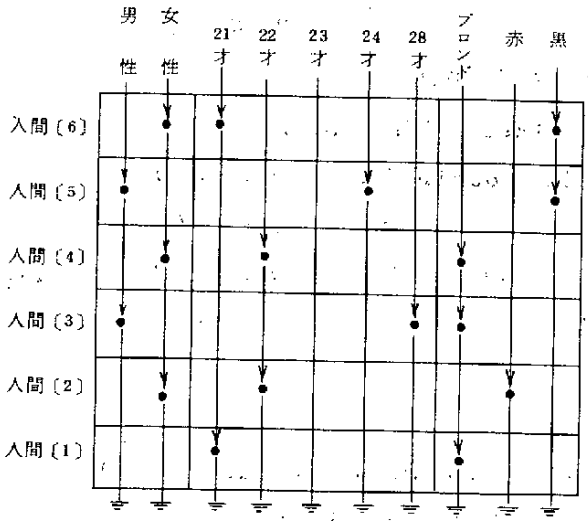


図6.3　高次列配列の連続割当ての例

このようにしておけば，「年令が２１才から２３才で髪の色がブロンドの女性を見つける」などの探索が高速に行なえるし，新しく人間を追加することも容易である。

スパース行列も連結割当てで記憶すると便利である。行列の各要素ごとに，図6－4のような五つのフィールドをもつ記憶単位をつくり，これらをポインタで結ぶことによってスパース行

左方向ポインタ	上方向ポインタ
行番号　列番号	要素の値

図６－４　スパース行列のための記憶単位

列を記憶する。その例を図６－５に示す。

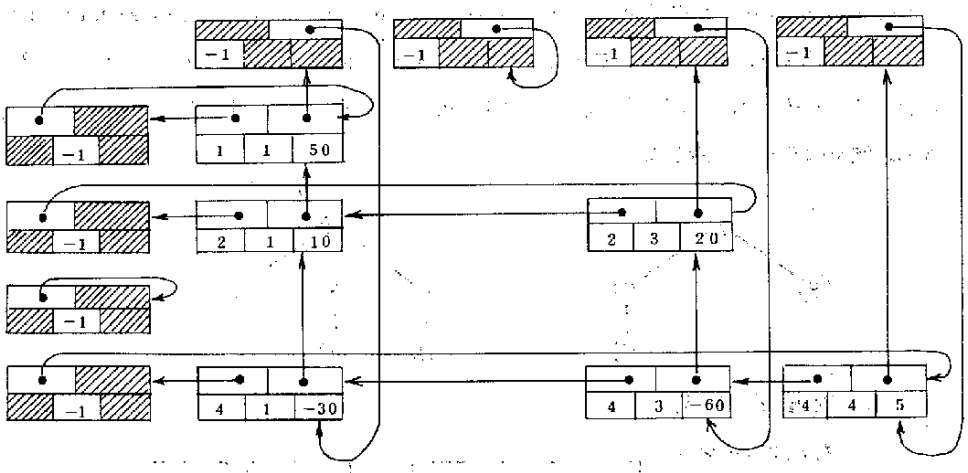


図６－５ $\begin{bmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{bmatrix}$ の連結割当てによる記憶

(6) 木 の 記 憶

木（ tree ）は非線型リストであるから，原則として，順次割当てはできない。一般に，木を連結リストとして表現し，そのまま連結割当てで記憶する。

木は２進木（ binary tree ）に変換できるので，２進木が記憶の基本になる。２進木の記憶のための連結リスト表現例を図６－６に示す。木登り（ tree traverse）を高速化するためのつなぎ木（ threaded tree ）表現も図６－６の記憶単位に１ビットのタグを加えるだけで記憶できる。

木を順次表現する方法がある。この方法には前置順次表現（ preorder sequential representation），家系順次表現（ family-order sequential repres-

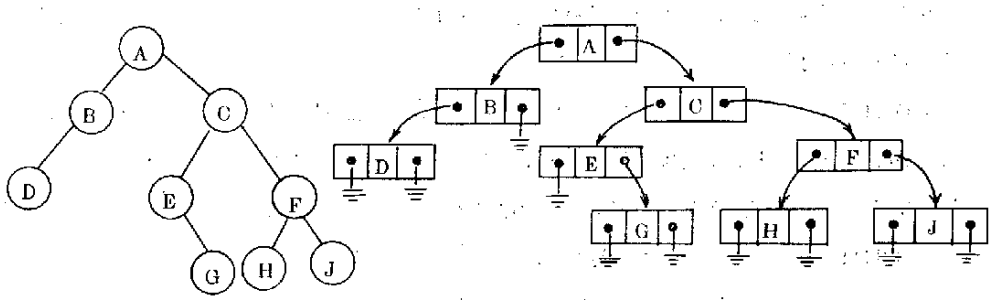
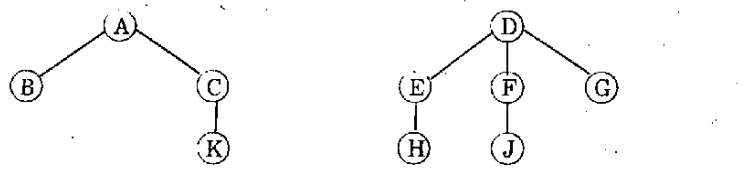


図６－６ ２進木とその連結リスト表現の例

entation），後置順次表現（postorder sequential representation）がある。ポインタ，タグ，次数（degree）を用いて木構造を表現する。この方法は2進木に限らず，一般の木を表現できる。この表現をそのまま反映させて木構造を記憶することができる。例を図6－7に示した。



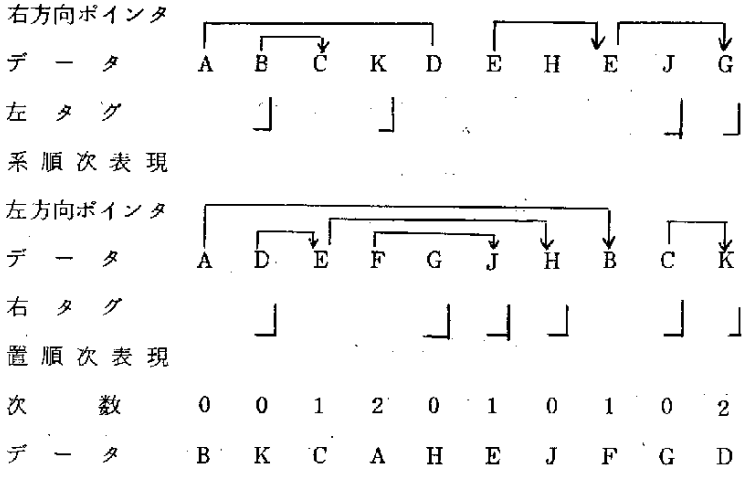


図6－7　木の順次表現の例

また，木構造は，結合行列（connection matrix）で表現できる。このとき，結合行列はスパースになることが多いので，図6－5のような連結リスト表現し，連結割当てで記憶するとよい。

結合行列による方法は，木構造に限らず，ネットワーク構造も表現できることに注意する。

(7) 構造の記憶

COBOL などで使うことができる構造データは，多重連結構造（multilinked structure）として表現される。すなわち，数個のポインタ部をもついくつかのデータの間の関係（構造）が，それらのポインタの連結によって表現されるデータである。

COBOL では，構造データに対して，

a_1 OF a_2 OF ……OF a_n

MOVE CORRESPONDING α TO β

などの操作ができる。ここで，a_1 は a_2 …，a_n によって修飾（ qualify ）されていることを表わし，α と β は同一の名前をもつデータ項目を含むグループの名前である。したがって，ＣＯＢＯＬコンパイラのコンパイル時における構造データの記憶は，上記のような操作を効率よくコンパイルできるようにするものでなければならない。

構造データに対しては，基本データ項目ごとに，次のような五つのポインタを用意する。

① ＰＲＥＶ

もし同じ名前をもつ項目があれば，そこに連結する。

② ＦＡＴＨＥＲ

この項目を含むグループがあれば，そのもっとも小さなものに連結する。

③ ＮＡＭＥ

この項目に対応する記号表中の項目に連結する。

④ ＳＯＮ

グループ中の最初の項目に連結する。

⑤ ＢＲＯＴＨＥＲ

この項目を含むグループ中の項目間を連結する。

これらをのポインタをデータ表中で互いに連結することによって構造を記憶する。例を図６－８に示す。

```
1  A                    1  H
   3  B                    5  F
      7  C                    8  G
      7  D                 5  B
   3  E                    5  C
   3  F                       9  E
      4  G                    9  D
                              9  G
```

記 号 表

	ポインタ
A :	A 1
B :	B 5
C :	C 5
D :	D 9
E :	E 9
F :	F 5
G :	G 9
H :	H 1

	PREV	FATHER	NAME	SON	BROTHER
A 1 :	∧	∧	∧	B 3	H 1
B 3 :	∧	A 1	B	C 7	E 3
C 7 :	∧	B 3	C	∧	D 7
D 7 :	∧	B 3	D	∧	∧
E 3 :	∧	A 1	E	∧	F 3
F 3 :	∧	A 1	F	G 4	∧
G 4 :	∧	F 3	G	∧	∧
H 1 :	∧	∧	H	F 5	∧
F 5 :	F 3	H 1	F	G 8	B 5
G 8 :	G 4	F 5	G	∧	∧
B 5 :	B 3	H 1	B	∧	C 5
C 5 :	C 7	H 1	C	E 9	∧
E 9 :	E 3	C 5	E	∧	D 9
D 9 :	D 7	C 5	D	∧	G 9
G 9 :	G 8	C 5	G	∧	∧

図6－8 構造データの多重連結表現の例

このようなポインタやデータ表による表現に対して，

① データ表をつくる。

② 修飾されたデータ名による参照をチェックする。

③ CORRESPONDING のために対応する対を決定する。

ための効率よいアルゴリズムが開発済みである。

6.2 分 散 記 憶

1組のデータを記憶するとき，出合う順番に対応させた番地を割当ててもよいが，表を探索しなければならないとき，所望のデータを見つけるまでに多くの時間がかかる。そのため，一定の規則によってデータを分散させて記憶することにより，探索時間の短縮を図るような記憶方法が考えられた。このような記憶の仕方を一般に分散記憶（scatter storage）といい，ハッシュ表（hash table）はその代表的なものである。

(1) ハッシュ表

記憶装置中でのアドレスを何らかの方法で決めることによって，できる限りデータのアドレスがデータごとに決まるようにした表である。通常，データ名を独立変数としたハッシング関数（hashing function）によってアドレスを決定する。このようすを図6－9に示す。

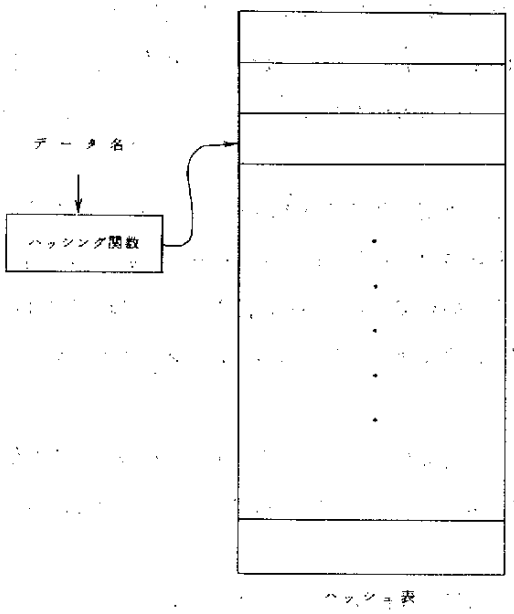


図6－9　ハッシュ表における番地の決定

(2) ハッシング関数

ハッシュ表では，関数値ができる限り異なるような関数を選ぶことが大切である。ハッシング関数には次のようなものがある。

① 除　算　法

正数値で割って，その剰余を関数値とする。

② 数 字 分 析

各桁における数字の現われ具合を調べ，一様性の高い桁を組み合わせて利用する。

③ 平方数の中央部

独立変数を2乗し，その中央部をもとにする。

④ 重　合　せ

独立変数をいくつかの部分に分け，それらの和をとる。

⑤ 基 数 変 換

独立変数の値を別の基数を使って表わし，それをもとにする。

⑥ 多項式の除算

独立変数の各桁の数字を線型多項式の係数とみなし，それを別に用意した多項式で割り，その剰余の式の係数をもとにする。

(3) ハッシュ表の記憶装置割当て

ハッシュ表では，異なるデータ名に対してハッシング関数値が同じになってしまうことがある。この現象を衝突（collision）という。衝突を生じた場合，記憶装置をどのように割当てるかには，基本的に次の二つの方法がある。

① 順次割当て

ハッシング関数値をもとに決定したアドレスAがすでに別のデータで使用されている場合，A＋1，A＋2，…とアドレスをずらして行って，最初に空いている場所を利用する。検索する場合は，アドレスAから始めて，見つかるか，空いている場所に到達するまで順次探して行く。したがって，記憶装置は連続した一連のアドレスを割当てておけばよい。

② 連結割当て

ハッシング関数値をもとに決定したアドレスAがすでに別のデータで使用されている場合，別に用意した未使用の記憶域から必要なだけの記憶装置を取り出し，ポインタによってそこを連結，使用する。

もちろん，衝突が生じるたびに連結リストは伸びて行く。

以上のようすを図6－10に示す。

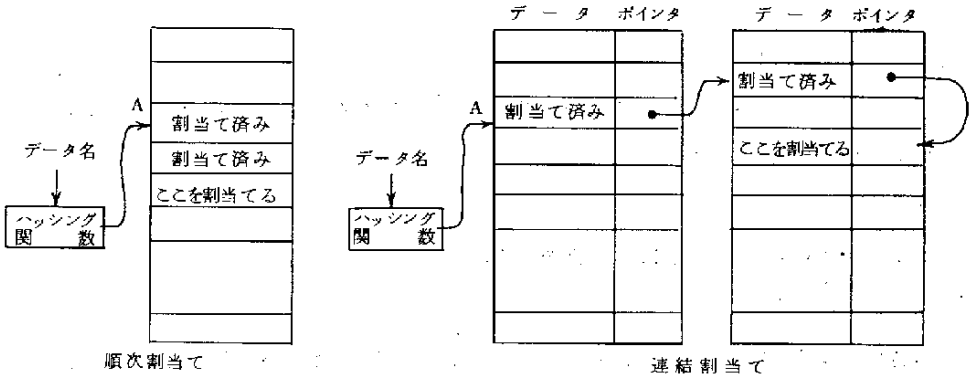


図6－10　ハッシュ表の記憶装置割当て

6.3 アドレスの計算

記憶装置はワード，トラックなどの記憶単位に分割され，それらの単位を識別するためにアドレス（address）がふられている。データを参照するときはそのアドレスを計算しなければならない。この節では，以下の項目について説明する。

(1) 直接アドレス付け

演算対象の記憶場所を直接指定するアドレス付けを直接アドレス付け（direct addressing）という。

命令語における演算対象のアドレスAの指定の仕方には次のようなものがある。

① 非修飾アドレス指定

$$A=(B)+D$$

② インデックス修飾アドレス指定

$$A=(B)+(X)+D$$

③ 相対アドレス指定

$$A=(P)+D$$

(B)は基底レジスタBの内容，(X)はインデックス・レジスタXの内容，(P)はプログラム・カウンタの内容，Dは命令語中で与えられる定数を表わしている。基底レジスタをもたないコンピュータでは，①，②において(B)の項がない。

(2) 間接アドレス付け

(1)の①～③で得られたアドレスの内容が演算対象のアドレスを示すアドレス付けを間接アドレス付け（indirect addressing）という。このようにして得られた，アドレスが再び間接アドレスになっていることもある。連結リスト処理にとっては非常に有効な機能である。

ハードウェアでスタックを実現しているコンピュータでは，スタックを参照する命令のアドレスの計算結果が，スタック・ポインタの記憶アドレスになっていることがある。それゆえ，特別な間接アドレス付けとみなすことができる。

(3) 即値アドレス付け

命令語の一部をそのままオペランドとして用いることを即値アドレス付け（immediate addressing）という。

(4) 連 想 記 憶

装置中での記憶場所を番地で指定する記憶装置に対して，引数（argument）のあるフィールドと一致する記憶装置中のワードやバイトにアクセスできるようにした記憶装置を連想記憶（associative memory），または内容によるアドレス付けの記憶（content addressable memory）という。連想記憶は，その機構の一部として，比較機能をもっている。

条件に合ったワードやバイトの場所はタグ・レジスタ（tag register）中のタグで示される。比較は，記憶装置中のすべてのワードに対してハードウェアによって並列的に実施されるので，非常に高速である。最近では，引数のフィールドとの一致だけでなく，大小関係や論理関係によってアクセスするワードを決定するような連想記憶も考えられている。

指導上の留意点

(1) 主として，論理構造を記憶装置中にどのように記憶するか，それをどのように操作するかについて，基本的な部分を理解させる。

(2) ファイルの物理構造に関する解説はファイルの章に委ねる。

(3) 連想記憶はアドレス付けの一種として解説するにとどめ，具体的な使い方や評価は別の章に譲る方がよい。場合によっては省略しても構わない。

参 考 文 献

〔1〕 D.E.Knuth,"The Art of Computer Programming, vol.1", Fundamental Algorithms, Addition-Wesley,1968

〔2〕 J.A.N.Lee,"The Anatomy of a Compiler",Reinhold, 1967

〔3〕 J.J.Donovan,"Systems Programming",McGraw-Hill, 1972

〔4〕 上条史彦「データ・システム」産業図書, 1972

〔5〕 Y.H.Chu,"Computer Organization and Microprogramming",Prontice-Hall,1972

〔6〕 D.W.Digby,"A Search Memory for Many-to-Many Comparisons", IEEE Trans.on Computers,vol.C-22,pp.768－772

第7章 記 憶 管 理

キーワード

ちり集め（ garbage collection ），動的割当て（ dynamic allocation ），使用可能空間リスト（ available space list ），隣接未使用空間（ adjacent free space），可変ブロック（ variable block ），内部断片化（ internal fragmentation ），外部断片化（ external fragmentation ），初適合（ first fit ），最適適合（ best fit ），最悪適合（ worst fit ），バディ・システム（ buddy system ），フィボナッチ・システム（ Fibonacci system ）

目 標

連結機構を用いることによって，表なども共通のプールされた記憶領域で各種の表を育てたり，縮小したりすることができる。アプリケーションによっては，ブロックの大きさを一定にしておき，その大きさを小さめにして連結記憶方式（ linked-memory philosophy ）をとることもある。しかし，多くの場合，ブロックの大きさは一定にせず，共通の記憶領域を共用する。そのためには，連続した記憶領域から可変のブロックを割当てたり，解放したりするためのアルゴリズムが必要である。これらのアルゴリズムを一般に動的記憶割当てアルゴリズムという。

本章では，ちり集め，動的記憶割当てなど，記憶管理に関する要点を理解させることを目標とする。

内 容

7.1 ちり集め

連結機構を用いることによって，表なども"プールされた"記憶領域で各種の表を育てたり縮小したりすることができることは既に説明した。リスト構造を使用したときには，未使用になった節（ node ）を使用可能空間リストに戻す方法が問題になる。未使用になるたびに使用可能空間リストに戻すこともするが，一般的な処理速度を上げるために"ちり集め"と呼ばれる方法が用いられることが多い。そこで，まず，ちり集めの方法を理解させる。

ちり集めの方法では，各節に"印ビット（ mark bit ）"と呼ばれる1ビットのフィールドを必要とする。この場合には，どの節も使用可能空間リストに戻さないようにほとんどのアルゴリズムを書く。そして，使用可能空間が全てなくなるまでプログラムを実行させるのである。使

用できるものがなくなったときに，"ちり集めルーチン（garbage collector）"で印ビットを用いて，そのとき使用していない節をすべて使用可能空間に戻し，再びプログラムを続行するのである。

ちり集めは，一般に，2フェーズから成る。各節の印ビットはゼロに初期設定されているものとすると，第1フェーズでは，主プログラムから直接アクセス可能な節から始めて，全てのちりでない節に印をつける。第2フェーズでは，無印の節を使用可能空間リストに載せる。

これらの代表的なアルゴリズムを説明する。節がいくつもの節から指されているときには，参照カウンタ（reference counter）を用いる必要があることを説明する。

さらに，節の大きさが可変の場合，単位となるブロックの大きさを小さめにし，一定にして"連結記憶方式"をとることもあることを説明する。この場合，節の大きさを一定にしないことにより，内部断片化を小さくしようとするものであることを理解させる。

7.2 動的記憶割当て

多くの場合，節の大きさは一定にせず，共通の記憶領域を共用する。共用するためには，連続した記憶領域から，可変のブロックを割当てたり，そのブロックを解放したりするためのアルゴリズムが必要である。これらのアルゴリズムを，一般に，"動的記憶割当てアルゴリズム"と呼ぶ。動的記憶割当ては，応用プログラムだけでなく，制御プログラム（オペレーティング・システム）でも行なう。そこで，動的記憶割当ての場合には，通常，節の代りにブロック（block）という用語を用いる。

(1) 割　当　て

動的記憶割当てを用いると，記憶は市松模様化（checkerboardin）することをまず理解させる必要がある（図7.1参照）。

動的記憶割当てにおける問題は，使用可能空間の維持の仕方と未使用空間を探す最良のアルゴリズムであることを説明する。

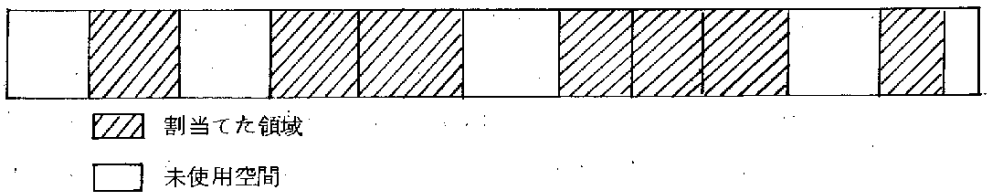


図 7.1　記憶空間の市松模様化

(a) 使用可能空間の維持の仕方

使用可能空間はリストとして維持する。最良の方法は，多少の例外を除いて，使用可能空間自身をリストにすることであることを説明する。

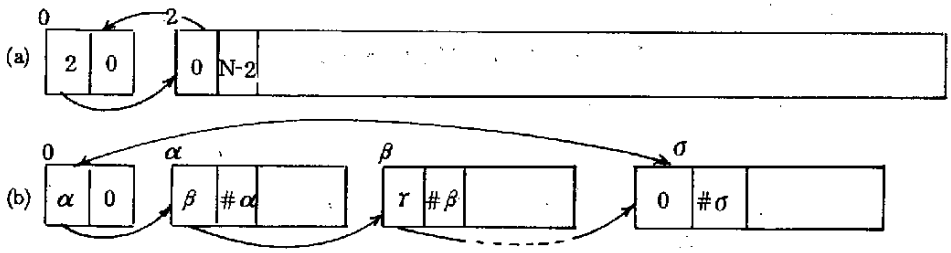


図 7.2　使用可能空間リスト　(a)初期状態　(b)一般的な状態

一般に，使用可能空間の最初に，そのブロックの大きさおよび次の未使用空間のアドレスをを表わす二つのフィールドを設ける。さらに，使用可能空間のプールの先頭に大きさ 0 の固定の節を設ければ，最初の未使用ブロックとなる（図7.2）。未使用ブロックは，大きさの降順，昇順あるいはアドレス順等に連結する（次に述べるアルゴリズムに依存する）。

(b)　アルゴリズム

基本的には，連続した n 語が必要な場合，m（$\geqq n$）語の未使用ブロックを探し，割当て，残りを $m-n$ 語として登録しておく。このとき，n 語以上のブロックとして，どれを選ぶかによって様々なアルゴリズムが考えられることを理解させる。

代表的なものとして，次の三種を説明する。

- 初　適　合

  未使用ブロックをアドレス順に連結し，n 語以上の最初の未使用ブロックを割当てる。

- 最 適 適 合

  未使用ブロックを大きさの降順，昇順またはアドレス順に連結し，$m-n$ が最小となるブロックを割当てる。

- 最 悪 適 合

  未使用ブロックを大きさの降順またはアドレス順に連結し，常に，最大の未使用ブロック（すなわち，$m-n$ が最大となるブロック）を割当てる。

これらのアルゴリズムを説明し，その利点および欠点，改良すべき点を理解させる。たとえば，改良初適合アルゴリズムは次のようになる。

```
First fit algorithm for given( size ) =
 own trail = 0 ;
 begin
   local space , dif ; local x = trail ;
   space := M[M[x]+1]                          $次の未使用ブロック
   if space < size then
```

```
        repeat
            x := M[x]; space := M[M[x]+1]
        until x = trail or space ≧ size
      fi
      trail := x ; dif := space − size
      if dif < 0 then report failure
      else if dif = 0 then M[trail] ← M[M[trail]]
           else M[M[trail]+1] ← dif fi ;
           return M[trail]+dif as starting address of the
           allocated block
      fi
    end
```

(2) 解　　放

未使用になったブロックを解放する場合には，ちり集め法を用いるべきでないことを説明する。特に，未使用空間が隣接する場合を例に上げて理解させる。このとき，詰め（compaction）と関係し，場合によっては，詰めと抱き合わせればちり集め法も利用できることを説明する。

解放の際の唯一の問題は，折りたたみ問題（collapsing problem）である。すなわち，二つの隣接未使用領域を一つにまとめることである。この結果，長時間にわたり，記憶領域の割当てと解放とが繰り返えされても，平衡が保たれることを理解させる。

解放のアルゴリズム例は，次のようになる。

```
release space (a,size) =
  begin
   local x = 0
   while M[x]< a and M[x]≠0 do x := M[x] od
   if x+M[x+1]=a then M[x+1]:=M[x+1]+size
   else M[a]:=M[x]; M[a+1]:=size ;
        M[x]:=a ; x:=a
   fi ;
  if x+M[x+1]=M[x] then
     M[x+1]:=M[x+1]+M[M[x]+1] ;
     M[x]:=M[M[x]]
```

```
        fi
    end
```

境界タグ(boundary tag)や2進表示を用いることにより，高速化できることも説明すべきであろう。

(3) バディ・システム

このアルゴリズムは 2^k 単位でブロックを管理しようとするものである($k \geqq 0$)。ある整数 k に対してブロックの大きさがちょうど 2^k 語でないときには，そのブロックより大きい 2^k 語のブロックのうちで最小のものを割当てる。この方法では，長い使用可能空間リストをたぐる必要がないので，処理が速いことを説明する。

この方法の概要は次のとおりである。まず，各 2^k ($0 \leqq k \leqq m$)の大きさの使用可能ブロックに対し，個々にリストを設ける。割当ての対象となる全空間は 2^m 語である。この空間を，便宜上，アドレス0からアドレス 2^m-1 までとする。最初は 2^m 語のブロック全体が使用可能である。その後，2^k 語のブロックが必要になったとき，その大きさの未使用ブロックがなければ，それより大きいブロックを2等分し，最終的に 2^k 語のブロックを作る。一つのブロックを2等分したとき，それらの二つのブロックをバディ(buddies)という。後刻，これらのバディが再び使用可能になったときには，一つのブロックとして折りたたむ。

この方法で重要なことは，ブロックのアドレスと大きさとがわかると，そのバディのアドレスがわかることであることを理解させる。これは，大きさが 2^k のブロックのアドレスが 2^k の整数倍であることによる。つまり，アドレス x が大きさ 2^k のブロックのバディのアドレスを $\mathrm{buddy}_k(x)$ と表わすとすると，

$$\mathrm{buddy}_k(x) \begin{cases} x + 2^k & : x \bmod 2^{k+1} = 0 \text{ のとき} \\ x - 2^k & : x \bmod 2^{k+1} = 2^k \text{ のとき} \end{cases}$$

となることによる。

バディ・システムでは，各ブロックについて，1ビットのタグ・フィールドTAGを用いることを説明する。ここで，

$$\mathrm{TAG}(p) = \begin{cases} 0 & : \text{アドレスpのブロックが割当て済み} \\ 1 & : \text{アドレスpのブロックが使用可能} \end{cases}$$

である。さらに，使用可能ブロックは二重連結リスト(double linked list)として管理することを説明する。

(4) フィボナッチ・システム

バディ・システムを一般化することによって，フィボナッチ・システムにできる。すなわち，フィボナッチ・システムでは，大きさF_n のブロックを作るのである。ここで，F_n は任意のkに対し，

$$F_n = F_{n-1} + F_{n-k-1}$$
$$F_0 = 0 \quad , \quad F_1 = F_2 = \cdots\cdots = F_{k+1} = 1$$

である。$k=0$の場合，$F_n = 2^n$ となりバディ・システムとなることを説明する。

フィボナッチ・システムで重要な問題は，隣接の"バディ"を折りたたむことである。そのために，"左バディ・カウンタ（left-buddy-counter）"を用いる必要があることを理解させる。左バディ・カウンタは，次のように維持する（図7.3）。

i）最大ブロック（根）のleft-buddy-counterはゼロにする。

ii）分割するたびに，左と右のブロックのleft-buddy-counterの値は次のようにする。

left-buddy-counter（右のブロック）＝0

left-buddy-counter（左のブロック）

＝left-buddy-counter（親）＋1

left-buddy-counterがゼロと非ゼロのバディが折りたたむ際の候補となるのである。

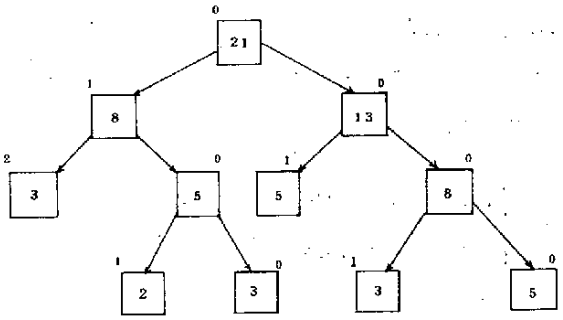


図7.3　$k=1$の場合のフィボナッチ木とleft-buddy-counter

(5) 各方式の比較

動的記憶割当てアルゴリズムの数学的解析は，さほど行なわれていない。有名な結果としては，５０パーセント則（fifty percent rule）がある。

そこで，実際には，シミュレーションの結果が報告されているので，それらの結果をもとに各方式を比較し，理解させるとよい。実際に，シミュレーションを行なわせてみると，さらによい。特に，使用可能空間の大きさとブロックの大きさとの関係がもたらす結果を理解させる。

また，各方式のアルゴリズムを十分に理解させることが重要である。

指導上の留意点

1. この章では，各方式のアルゴリズムを十分に把握させる必要がある。
2. 各方式を比較して，各方式の利点，欠点，効率等を十分に理解させなければならない。
3. 指導者は最新の学術文献に目を通し，最新の成果を積極的に取り上げるべきである。

参考文献

〔1〕 D.E.Knuth,"The Art of Computer Programming.Vol.1",Addison Wesley,1968

〔2〕 P.J.Denning,"Virtual Memory",Computing Servey ,Vol.2,1970

〔3〕 A.N.Habermann,"Introduction to Operating System Design", Science Research Associate,1976
土居範久訳，「オペレーティングシステムの基礎」培風館，1978

第8章　プログラム言語における情報構造の表現

キーワード

宣言(declaration), 型宣言(type declaration), データ型(data type), スカラ型(scalar type), 構造体型(structured type), 整数型(integer type) 実数型(real type), 文字型(character type), 文字列型(string type), 論理型(logical type, Boolean type), 部分範囲型(subrange type), ビット型(bit type), 文字列(string), 部分文字列(substring), 可変長文字列(varying string), 固定長文字列(fixed length string), ビット列(bit string), 配列(array), 配列要素(array element), 多次元配列(multidimensional array), 添字(subscript), 整合寸法(adjustable dimension), 整合配列(adjustable array), リスト(list), リスト要素(list element), ポインタ型(pointer type), 構造体変数(structured variable), レコード(record), 可変レコード (variant record), 階層構造 (hierarchical structure), 基本項目(elementary item), 表(table), 変数の有効範囲(scope of variables), 静的変数(static variable), 動的変数(dynamic variable), FORTRAN, ALGOL, PL/I, COBOL, SNOBOL, LISP, GPSS, SIMSCRIPT, IDS, PASCAL, 抽象データ型(abstract data type), データの抽象化(data abstraction), 具象化(instantiation), 実行時スタック(run time stack), ディスプレイ (display) 活動記録(activation record), ドープ・ベクトル(dope vector), 配列要素のアドレス計算(address computation of an array element), Iliffe ベクトル, 記憶域の動的割当て(dynamic memory allocation), 静的割当て(static allocation), 記憶域の解放(deallocation), パラメタの受授(parameter passing), オフセット(offset), テンプレート(template), ベース・アドレス(base address), 相対アドレス付け(relative addressing), 絶対アドレス付け(absolute addressing)。

目　標

情報構造の取扱いに関して特徴のある代表的なプログラム言語をいくつか選び, 各言語における情報構造の表現方法および記述方法について修得させるとともに, コンピュータ内部における

それら論理構造の実現について理解させることを目標とする。

プログラミング言語におけるデータ型と型宣言の役割，宣言の有効範囲などについて簡単に説明する。また，最も代表的な言語であるFORTRAN, ALGOL, COBOL, PL/Iにおける情報構造の特徴を説明し，配列，文字列，リスト，構造つき変数などの個々の情報構造について，各言語におけるそれらの取扱い方，記述方法，および構造の相違点を比較する。これらのまとめとして，データ構造の取扱いに関して，PASCALを例にとって，情報構造の表現について説明する。さらに，抽象データ型に触れる。ここでは，プログラミングの作成過程を考慮しながら，従来のプログラミング言語における情報構造の取扱いの問題点を明確にし，抽象データ型の定義と記述法について説明する。最後に，プログラミング言語における情報構造が，実際にはコンピュータ内部でどのようにして実現されるかを理解させる。そこで，各種の構造について，記憶の割当て，実行時の記憶構成（run time storage organization），データの参照方法について説明する。

内　容

8.1 プログラム言語と情報構造

まず序論として，プログラミング言語における情報構造のとらえ方について以下の項目を説明する。

(1) プログラムの構造

プログラムは一般にデータに関する記述と，それらに対する操作の記述とからなる。データに関する記述は各プログラミング言語によって，それぞれ用意された宣言を用いて行なう。それらの宣言によって，プログラムで使用する変数の名前，その変数に対する各種の属性（attribute）を指定する。変数はその値が単一のデータか，あるいは基本的なデータの複合体で構造をもっているかどうかによって，次の2種に大別される。

(a) スカラ型

多くの場合，スカラ型の変数はその値の型を指定するだけでよい。通常，高級プログラム言語には次のようなものが基本的データ型として用意されている。

- 整　数　型
- 実　数　型
- 論　理　型
- 文字列型
- 倍精度実数型
- 複素数型

(b) 構 造 型

配列や階層構造をもつものがある。それらを表現するためには，基本データ（ elementary data ）の型と，それらデータ間の関係を指定しなければならない。

(2) 代表的なプログラム言語における情報構造

プログラム言語のレベルで扱うことができる情報構造にはどのようなものがあり，そこにどのような工夫がされているかを理解させる。個々の情報構造についての言語間での比較は次節以降で行なうので，ここでは，簡単に説明すればよい。

(a) FORTRAN

次の項目について説明をする。

- データの型
- 型宣言文を用いた宣言と暗黙の型宣言（ implicit typing ）
- 配列の取扱い
- COMMON文およびEQUIVALENCE文による記憶割当ての指定

(b) ALGOL

ALGOLにおける情報構造の特徴として，変数の有効範囲と動的変数について，次の要点に沿って説明する。

① ブロック構造

- ブロック
- ブロックの宣言部分の構造

② 有効範囲

- 変数，配列，手続きなどの有効範囲
- 局所的（ local ）と非局所的（ nonlocal ）

③ 変数および配列の動的な性質

ブロックに実行が移ったとき，そこで宣言された配列や変数が使用可能状態となり，ブロックの終了によってそれらは無効となる。

④ 配列の取扱い

- 添字域の指定（上下限）
- 添 字

(c) COBOL

COBOLにおける階層構造をもつデータの取扱いについて説明する。

- データ部の構成
- 基本項目と集団項目（ group item）

- 反復句（OCCURS句）
- 修飾（qualification）と一意参照

(d) PL／I

PL／Iは，これまでに述べた情報構造以外に，ポインタ，文字列，あるいはビット列のデータが容易に扱えるようになっている。詳細は次節以降で触れるので，ここではそれらを簡単に説明する。

- データの属性たとえば，BINARY, DECIMAL, FIXED, FLOAT, BIT, CHARACTER, VARYING, PICTURE
- 静的割付け（STATIC）と自動割付け（AUTOMATIC）

8.2 情報構造の表現

前節では，プログラム言語における情報構造のとらえ方を理解するために，実際のプログラム言語を例にしながら基本的な概念や用語について説明してきた。この節では配列，文字列，リストというように情報構造ごとに，代表的な言語におけるその取扱いについて理解させる。

(1) 配　列

FORTRAN, PL／I，およびALGOLにおける配列について，次の事項について説明する。

- 配列要素の情報構造
- 上下限の指定
- 配列要素および配列の参照

  PL／Iの配列の断面（cross section）についても説明する。
- 動的配列と静的配列

(2) 文字列

SNOBOLおよびPL／Iにおける文字列について，次の事項について説明する。

- 可変長文字列と固定長文字列
- 部分文字列
- 文字列の基本操作
- SNOBOL－4におけるパターンと照合

(3) リスト

PL／I, GPSS, SIMSCRIPT, IDS, LISPにおけるリストの構造について説明する。

- 各言語におけるリストの構造とその記述法

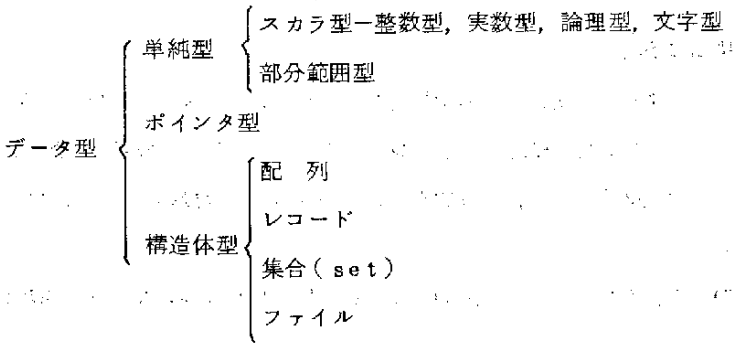
- リスト要素の参照とリスト操作
- リスト要素に対する記憶の管理

(4) 表とレコード

PL／IとCOBOLにおける構造体変数を用いて表やレコードの表現について説明する。

8.3 PASCALにおける情報構造

これまでのまとめとして，PASCALにおける情報構造について説明する。PASCALの特徴の一つは，問題レベルのデータ型が使用者自身で定義できるようになっていることである。PASCALのデータ型は次のように分類されている。その概要は参考の項に示す。



8.4 抽象データ型

問題レベルのアルゴリズムを，構造的にあるいは系統的に，スムーズに詳細化していくというプログラムの作成法が広く用いられるようになっている。抽象化（abstraction）とは，物事が複雑なときに，その中から本質的なものだけを抜き出して考えるということで，我々のもっている強力な道具の一つであるといえる。これまでの構造的プログラミングに関する議論では，機能的な面つまり操作の記述についての抽象化が強調されていた。しかし，データと操作とは分離して考えるべきものではなく，相互に関連しているものなので，次にそれらを一体化してとらえるというようになってきた。

(1) 抽象データ型

プログラミング方法論や設計論と関連づけながらデータの抽象化，抽象データ型について説明する。

(2) 抽象データ型の定義

スタックの定義とそれに関する操作などを例として，以下に示す抽象データの定義機能について，そこでの抽象データのとらえ方および記述法について説明する。

- SIMULA67のクラス（class）

- CLUのクラスタ(cluster)
- ALPHARDのフォーム(form)
- EUCLIDのレコード(record)

8.5 実行時におけるデータ領域の構成

コンパイラによって生成される目的プログラムは，通常，命令部とデータ部とからなる。命令部は原始プログラム(source program)の実行文から生成され，データ部は宣言の情報をもとに作られる。静的な言語の場合には，その目的プログラムのデータ部は実行時に固定した記憶場所が与えられ，プログラムが終了するまでその構造は変わらない。一方，動的な言語の場合には，プログラムの実行にともなって，必要な大きさの記憶場所が確保され，実行に必要なデータの構造が作られ，不要になるとその記憶場所は他の目的のために解放される。この節では，データ領域の基本的な構成やそれに関する用語を説明する。

(1) 記憶の割当て

本講の講義または演習に使用しているコンピュータにおける記憶装置の論理的構成，記憶単位(バイト，語)，およびアドレス付けについて，以後の説明が理解できる程度に簡単に説明する。

高級言語における情報構造は，目的プログラムにおいて，単純な構造をもつ記憶装置とそこへの単純なアクセス機構(access mechanism)を利用して実現しなければならない。つまり論理構造から物理構造へのマッピングが必要となる。

次の項目について説明する。

- 記憶割当ての必要性
- 静的割当てと動的割当て

(2) データ領域

割当ての最小単位は，個々の変数や配列である。通常はプログラムを構成する単位，たとえばFORTRANではプログラム単位，ALGOLの場合にはブロックという単位で，そこで使用される定数，変数，配列などのために，一括して必要な大きさの連続した記憶域が確保される。そのような領域をデータ領域という。ここでは次の事項について説明する。

- 活動記憶
- ディスプレイ
- テンプレートの必要性とその構成

8.6 論理構造の実現

基本的なデータ型の変数，配列，および構造体変数の実現方法について説明する。

(1) 基本的なデータ型の変数

整数型，実数型，論理型，文字型，あるいはポインタ型の単純変数に対するコンピュータ内での実現について説明する。これらは，通常，物理構造へ直接対応させることができるものである。ただし，バイト型のコンピュータの場合には境界（boundary）に注意しなければならないことがある。

(2) 文字列

文字列の場合には，値だけでなく，プログラムで指定された文字列の最大の長さ，現在の値の長さ，あるいは値が格納されている記憶場所のアドレスなどを実行時に保持しておかなければならない。任意の長さが扱えるような言語では，文字列空間（string space）と呼ぶ場所に文字列そのものを格納するようにし，その中で動的割当てをしなければならない。文字列空間の大きさと割当て方式の間には損得分岐点がある。文字列の転記および部分文字列の参照方法についても説明する。

(3) 配　列

ここでは，基本的なデータ型の要素からなる配列を考える。次の事項について説明する。

(a) 配列要素のアドレス計算

FORTRANの配列を例にして，ベクトル，行列，および多次元配列の要素に対するアドレスの計算法を示す。この場合にはアドレスの計算式は定数部と可変部に分割すると計算の効率がよくなる。可変部は多項式によって計算される。FORTRANの場合には静的配列であるので，アドレス計算の式を直接目的プログラム中に展開してしまうことができる。さらにその計算は最適化（optimization）によって高速化されることがある。

(b) ドープ・ベクトル

PL／IやALGOLのような動的配列の場合には，各寸法の上下限あるいはその配列のために，確保した記憶場所のアドレスを1ケ所に集めておく必要がある。そのような情報を収集したものがドープ・ベクトルである。配列要素のアドレス計算はドープ・ベクトルを参照しながら行なうので，上述の方法と比べると時間がかかる。

(c) Iliffeベクトル

配列要素へのポインタを要素とするベクトルを用いる方法であり，間接的に何重かそのベクトルを参照していくことによって，目的とする要素のアドレスを求めることができる。ただし，ポインタのベクトルを作らなければならないので，記憶場所を必要以上に多く使用する。

(4) 構造体変数

ＰＬ／Ｉの構造体変数またはＣＯＢＯＬの階層構造をもつ変数に対する記憶の割当てを説明する。論理構造が木状の関係で表現されていたとして，そこに含まれる各変数に対する記憶の割当てのアルゴリズムを示し，各変数の参照方法について説明するとよい。ＰＬ／ＩのＡＬＩＧＮＥＤおよびＰＡＣＫＥＤ属性，またはＣＯＢＯＬのＳＹＮＣＲＯＮＩＺＥＤ句の指定が記憶の割当てに与える影響とデータの参照の効率にも触れる。

8.7 ブロック構造におけるデータ領域の構成

ＡＬＧＯＬやＰＬ／Ｉのようにブロック構造をもつ言語では，コンパイル時にそのデータ領域のわく組は確定するが，実際の記憶域はブロックが実行される時に確保しなければならない。それと同時に配列や文字列の変数が使用されるときには，ドープ・ベクトルやテンプレートをそこに作り出さなければならない。このようなデータ領域（活動記憶）は，通常，実行時スタックと呼ばれるスタック上に確保される。

(1) 手続きに対するデータ領域

コンパイル時に各手続きについて，そこで必要とするデータ領域の大きさや構造は決定することができる。その中には次のような種類のデータが含まれる。

① ディスプレイ

② 実行時スタックのポインタ

③ 手続き呼出しにともなう暗黙的なパラメタ：レジスタの退避場所，戻り先，実パラメタのディスプレイ

④ 実パラメタ自身

⑤ 各ブロックに対しては，次のような情報を保持するための場所が必要である。

- 実行時スタックの最上段へのポインタのための記憶場所
- 単純変数の値を記憶するための場所
- ドープ・ベクトル
- 複雑な式を評価するときなどに必要となる中間結果を保持するための記憶場所

配列要素自身の値を格納するための場所はプログラムを実行してみないと，必要な大きさが定まらない。したがって，そのような場所もブロックを実行するときに，はじめて実行時スタック上に確保され，対応するドープ・ベクトルの中にそのアドレスがセットされる。

(2) 変数の参照方法

動的なデータ領域に割当てられた変数を参照するための方法について説明する。変数は一般に，それが属するデータ領域のオフセットの値と実行時スタック上に確保されたデータ領域の先頭アドレスの和で表現される。後者の値は，目的プログラムでは，レジスタに置くようにさ

れれるので，動的変数は目的コードではレジスタ修飾によってアドレス付けされる。

(3) 実行時スタックの制御機構

次の事項について説明する。

- 手続きの入口と出口における実行時スタックの制御
- ブロックの入口と出口，およびブロックの脱出時点における実行時スタックの制御
- ドープ・ベクトルの作成および文字列型の変数（ＰＬ／Ｉ）に対する記憶場所の初期設定

8.8 パラメタの受授

パラメタの型には以下に示すように数種類のものがある。手続き呼出しによって，実パラメタと仮パラメタとが各型によってどのように対応づけられるかについて説明する。

- 参照型呼出し（ call by reference ）
- アドレス型呼出し（ call by address ）
- 値型呼出し（ call by value ）
- 結果型呼出し（ call by result ）
- 名前型呼出し（ call by name ）

8.9 リスト処理言語におけるリスト構造の実現

これまでは，主として翻訳（ translation ）によって実行されるような言語における論理構造の実現を扱ってきた。この節では，通訳（ interpretation ）によって処理される言語の中でもリスト処理言語，主としてＬＩＳＰを中心として，そこにおける論理構造の実現方法を説明する。

(1) 属性リスト

ＬＩＳＰのＳ式（ Ｓ－expression ）に対するリスト構造について述べ，それを構成する原子記号（ atomic symbol ）はさらに属性リスト（ property list ）からなることを説明し，属性リストの構造を示す。

(2) 操　作

ＬＩＳＰの代表的な関数，car，cons，nconc などによって，内部的にリスト構造がどのように変化するかを説明する。

(3) 自由リストの管理

自由リスト（ free list ）の構成と管理方式について述べる。リスト処理および記号処理言語の処理系においてはちり集め（ garbege collection ）のタイミングとそのアルゴリ

ズムが実行効率に大きな影響を与える。個々のコンピュータの特徴を利用したリスト要素の内部表現とちり集めのアルゴリズムについて，効率の面から考察する。

指導上の留意点

(1) 本章では，問題レベルで扱っていた論理構造をプログラム言語でどのような手段によって実現することができるか，またどのような実現方法が望ましいかを理解させることが重要である。さらに，プログラミングにおけるデータの構造化の問題を認識させた上で，データの抽象化に関して問題の意味，考え方，データ型の定義方法を理解させなければならない。

(2) 各節では多数のプログラム言語を引用するので，ともすれば講義がまとまりのないものになってしまう恐れがある。言語仕様の細部についての説明はできるだけ避けるべきである。

(3) 言語によっては，概念的に同一のものであっても，異なった表現あるいは用語が使われることがある。指導者は用語の意味を正確に理解させ，用語の選択に注意しなければならない。

(4) 時間の配分に関して，最初の節は復習または以後の節への準備を目的としているので，説明は簡単でよい。第2節に重点を置くようにする。

(5) 時間的に余裕があれば，第2節で図形処理用の言語における情報構造を説明するとよい。その言語（ASP，CORAL，LEAPなど）では処理の必要性から複雑な構造をもったリストが扱えるようになっている。

(6) 第2節の説明では，言語による情報構造の単なる相違点を理解させるだけでなく，言語の応用分野，設計思想，または歴史などの背景も必然性として理解させるようにするとよい。

(7) 抽象データの取扱いに関しては，現在もまださかんに研究・議論が続けられている。このテーマは第4単元「ソフトウェア設計」とも関係している。本章では例題を中心として，その問題意識と考え方を理解させる程度でよい。

(8) PASCALにおけるデータ型の分類には批判（参考文献〔27〕）もある。第3節の終りでそれについて触れ，プログラム言語におけるデータ型の分類について検討するとよい。

(9) 一般にプログラム言語において扱える情報構造が複雑化するにつれて，その実現も複雑になり，データの参照が効率の悪いものとなる。動的配列を例として，使用上の便利さと，それを実現するための物理構造と機構をよく理解させるとよい。

(10) 問題レベルのデータ構造たとえばリストを扱うようなとき，プログラム言語のレベルでは一般にいく通りかの実現方法が考えられる。処理の効率，他の機種への互換性などの要因がとくに重要なときには，それらのコンピュータ内部での実現がある程度見透せることが必要である。論理構造と物理構造の対応づけは，実際のプログラミングにおいては，言語やそこで用いる情報構造を選択するときに考慮しなければならない。本章を通して，この点を強調すべきである。

(11) 記憶域の動的割当てに関しては，第7章「記憶管理」で説明しているので，本章では省略する。時間的に余裕があれば，第3節で簡単に復習するとよい。ファイル上での実現については第3単元にあるので，ここではとり上げていない。このほかに，本章に関する事項としては以下のものがある。時間があれば，これらについて説明するとよい。

- 仮想記憶をもつコンピュータでの効率的な論理構造の実現
- FORTRANにおけるCOMMON文およびEQUIVALENCE文の記憶割当てに関する処理
- データ・ベース用言語における論理構造の実現
- PL／IのALLOCATE文とFREE文の実行時における処理

参　考

8.3節で扱うPASCALのデータ型の定義について簡単に述べる。PASCALはN.Wirthによって設計されたALGOL風の言語である。構造的プログラミングを指向した言語としては，現在のところもっとも普及しており，多くの文献や説明書が出ている。

PASCALにおける情報構造の分類は，8.3節に示したとおりである。まず，どの言語にもみられるように，整数型，実数型，論理型および文字型が用意されている。これを標準スカラ型という。次に基本的な型として，スカラ型と部分範囲型がある。スカラ型は自分自身を値としてもつような名前を列挙することによって定義される。それらの各名前の間には，定義の順序に従って順序関係が成立つ。部分範囲型はすでに定義したスカラ型を用いて，その一部分を定義するものである。

スカラ型の例：**type** day ＝（sun,mon,tue,wed,thu,fri,sat）;

部分範囲型の例：**type** weekday ＝ mon‥fri ;

変数の宣言は一般に次のように書く。

var v_1 ,v_2 ,…,v_n : t_1 ; w_1 ,w_2 ,…,w_m : t_2 ;…;

vとwは変数名，t_1，t_2はデータ型の名前である。

以上のような型を総称して単純型と呼ぶ。以下に示すのは構造型と呼ばれるもので，配列，レコード，集合，およびファイルの四つがある。それらは単純型および構造体型を用いて定義される。

(1) 配列型　**type** a＝**array**［t_1］ **of** t_2;

aは定義しようとする型の名前であり，t_1は配列要素を指すときに用いる添字のとり得る範囲を示すスカラ型の名前，t_2は配列要素の型を指定するもので任意の型を書くことができる。

例：**type** month＝（jan , feb ,…, dec）;

```
calender = array [ month , 1 … 31 ] of day ;
```

(2) レコード型

レコードはいくつかの欄(field)の集りとして定義される。各欄には名前をつけ，手続きにおいては，レコード名と欄の名前を指定することによってデータを識別する。たとえば，複素数を扱うときには，次のような定義をする。

```
type complex = record realpart : real ;
                      imgpart  : real end ;
var x : complex
```

complex という型は二つの実数型の値をとる欄からなり，前半の欄を realpart，後半を imgpart と名づける。変数 x の実数部，虚数部を指定するときには，それぞれx.real-part，x.imgpart と書く。レコード型の定義を用いることによって，COBOLやPL/Iにあるような階層的な構造を定義することができる。

(3) 集合型　**type** S = **set of** t ;

スカラ型 t の値からなる部分集合を値としてとるような型Sを定義する。

例：
```
type color = set of ( red ,yellow , blue );
var x : color ;
```

集合型の変数 x は値として〔red〕，〔yellow〕，〔blue〕，〔red , blue〕，…，〔red, yellow , blue〕，および空集合〔 〕のいずれかをとる。集合型のデータについては，演算子＋，＊，－によって，それぞれ和集合，積集合，補集合を求めることができる。また同値性，包含関係，および集合要素を検査するための演算子も用意されている。

(4) ファイル型　**type** f = **file of** t ;

ファイルはレコード(記録)を逐次的に並べたものとして定義される。そのレコードの型は t で指定する。ファイル型の変数たとえば x を宣言すると，それにともなって1記録分のバッファが作られる。これをバッファ変数あるいはwindowと呼び，プログラムでは $x\uparrow$ で表わす。ファイルのデータに対する参照はこの変数を用いることによって行なう。文字型データからなるファイルはテキスト・ファイルと呼ばれ，通常の入出力文read やwrite は既定義のテキスト・ファイルに対する操作として定義される。

PASCALには単純型と構造体型のほかにポインタ型がある。データ型 t のデータを指すポインタ型 p は，

```
type p = ↑t
```

によって定義する。このような定義を用いると線形リストは次のように再帰的に定義できる。ここで，リスト要素は整数型の値をもつ欄とポインタをもつ欄とからなるものとする。

```
type link = ↑ listcell ;
     listcell = record valpart : integer ;
                       linkpart : link end ;
```

ポインタ型の変数に対しては，記憶の割当てと解放のために，それぞれnewとdisposeという標準手続きが用意されている。たとえば，new(x)と書くと，ポインタxの指すデータを格納するために記憶域が確保され，その領域に対するポインタの値がxに代入される。

dispose(x)はその逆の操作であり，xの指していた記憶域は解除される。

参考文献

〔1〕 日本規格協会「ＪＩＳ 電子計算機用プログラミング言語 ＦＯＲＴＲＡＮ」昭和42年

〔2〕 日本規格協会「ＪＩＳ 電子計算機用プログラミング言語 ＡＬＧＯＬ」昭和42年

〔3〕 日本規格協会「ＪＩＳ 電子計算機用プログラミング言語 ＣＯＢＯＬ」昭和47年

〔4〕 浦昭二「データ構造 電子計算機基礎講座」共立出版，昭和49年

〔5〕 菅忠義「標準言語ＦＯＲＴＲＡＮ」共立出版，昭和47年

〔6〕 西村恕彦「標準言語ＣＯＢＯＬ」共立出版，昭和49年

〔7〕 島内剛一「プログラミング言語論」共立出版，昭和47年

〔8〕 竹下亨「ＰＬ／１複合プログラミング言語」日本経営出版会，昭和46年

〔9〕 中西正和「ＬＩＳＰ入門」近代科学社，昭和52年

〔10〕 米田信夫・野下浩平「ＡＬＧＯＬ60講義」bit，共立出版，第8巻1号（昭和51年）～第9巻3号（昭和52年）

〔11〕 情報処理学会「情報処理・ソフトウェア・エンジニアリング特集号」第16巻10号，昭和50年

〔12〕 原田賢一「ストラクチャード・プログラミング用言語」数理科学6月号，サイエンス社，昭和51年

〔13〕 情報処理学会「構造的プログラミングとその経験・シンポジウム報告集」昭和50年

〔14〕 情報処理学会「パネル討論会 構造的プログラミング」情報処理，第17巻11号，昭和51年

〔15〕 O.J.Dahl,E.D.Dijkstra,and C.A.R.Hoare,"Structured Programming", Academic Press,1972 野下浩平他訳「構造化プログラミング」サイエンス社，昭和50年

〔16〕 N.Wirth,"Systematic Programming : An Introduction", Prentice-Hall,1973 野下浩平他訳「系統的プログラミング入門」近代科学社，昭和50年

〔17〕 K.Jensen and N.Wirth,"PASCAL User Manual and Report,2nd Ed.", Springer,1975

〔18〕 R.Conway and D.Gries,"An Introduction to Programming",Winthrop,1973

〔19〕 R.E.Griswold,et al.,"The SNOBOL 4 Programming Language, 2nd Ed.",Prentice-Hall,1971

〔20〕 J.McCarthy,et al,"LISP 1.5 Programmer's Manual",MIT Press, 1962

〔21〕 C.Weissman,"LISP 1.5 Primer,Dikenson",1969.小林達訳「LISP入門」ダイヤモンド社, 1971

〔22〕 J.E.Sammet,"Programming Languages:History and Fundamentals", Prentice-Hall, 1969. 竹下亨訳「プログラミング言語ハンドブック」日本経営出版会, 1971

〔23〕 D.Gries,"Compiler Construction for Digital Computers",John Wiley & Sons,1971

〔24〕 CODASYL Systems Commitee,"Feature Analysis of Generalized Data Base Management Systems", Technical Report,1971

〔25〕 G.Gordon,"Systems Simulation",Prentice-Hall,1969

〔26〕 H.W.Lawson Jr.,"PL/I List Processing",CACM, Vol.10, pp.358-367,1967

〔27〕 A.N.Harbermann,"Critical Comments on the Programming Language PASCAL", Acta Informatica, Vol.3, pp.47-57, 1973

〔28〕 P.Weger,"Programming Languages,Information Structures,and Machine Organization",McGraw-Hill, 1968

〔29〕 R.Williams,"A Survey of Data Structure for Computer Graphics Systems", Computing Surveys,Vol.3,№1, pp.1-21, 1971

〔30〕 W.A.Wulf,"ALPHARD:Toward a Language to support Structured Programs",Carnegie-Mellon Univ., 1974

〔31〕 L.Flon,"A Survey of Some Issues Concerning Abstract Data Types",Dept.of C.S.,Carnegie Mellon Univ., 1974

〔32〕 B.Liskov,"A Note on CLU",M.I.T., 1974

〔33〕 B.W.Lampson,et al.,"Report on The Programming Language Euclid",

SIGPLAN Notices, ACM, Vol. 12, №2, 1977

〔34〕 R.A.Hopgood, "Compiling Techniques", Macdonald Computer Monographs, Macdonald, 1969. 首藤勝, 他訳:「コンパイラの技法」サイエンス社, 昭和48年

〔35〕 J.M.Foster, "List Processing", Macdonald Computer Monographs, Macdonald, 1967. 西村敏男訳「リスト処理」サイエンス社, 昭和48年

〔36〕 J.K. Iliffe, "Basic Machine Principles", Macdonald Computer Monographs, Macdonald, 1968

〔37〕 D.E.Knuth, "The Art of Computer Programming, Vol. 1, Fundamntal Algorithms, 2nd Ed, Addison Wesley, 1974

〔38〕 B.Randell and D.J.Russell, "ALGOL 60 Implementation", Academic Press, 1964

〔39〕 R.E.Griswold, "The Macro Implementation of SNOBOL 4", Freeman, 1972

第9章　分類および探索

キーワード

内部分類法(internal sorting), 挿入分類法(insertion sort), 交換分類法(exchange sort), 選択分類法(selection sort), 併合分類法(merge sort),分配分類法(distribution sort), 外部分類法(external sort), バランス型分類法(balanced sort), アンバランス型分類法(unbalanced sort), 分類法の効率(efficiency of sorting methods), 逐次探索法(sequential search), キー比較探索法(searching by comparison of keys), 木探索(tree search), 見出し付探索(indexed search), ハッシュ法(hashing), 探索法の効率(efficiency of searching methods)

目　標

分類の技法に関しては内部分類法, 外部分類法の概要について説明し, それぞれの技法のアルゴリズムを理解させるとともに, その特徴および効率についての知識を与える。これらの知識を前提として, 与えられた問題に対してどの技法が最も適切であるかを判断できるようにすることを目標とする。

探索の技法に関しては, いろいろな方法の概要ならびに特徴について説明するとともに, 探索の効率についての知識も与える。さらに与えられた問題について技法を選択できるようにすることを目標とする。

一般の応用システムでは, 分類と探索の両方が必要とされる場合が多くみられる。したがってシステム全体を通じて見たときに, ファイルおよびデータの構造に最も適し, さらにデータの入力, 更新, 検索の方法, 頻度など多方面からの検討を加えて, 最適の技法を選び出し, 全体のシステムを構成できる能力を養うことを目標とする。

内　容

9.1 分　類

(1) 分類の重要性と基本用語の確認

大量のデータ処理における順次処理の利点をスペースと時間の節減という観点から説明する。順次処理における, 入力, 処理および出力の各段階において, 次の観点から分類の必要性を

確認させる。

- 入力段階……データの重複，欠落，順序のチェックなど。
- 処理段階……ファイルの突き合わせを行なう際のレコードの順序。
- 出力段階……要求された順序でのレコードの出力。

分類に関する基本的用語として次のものの意味を明確にする。

- ファイル，レコード，フィールド
- キー
- 照合順位
- 分類と併合
- 昇順と降順
- 系列
- パス
- 内部分類と外部分類

(2) 分類の技法

分類の対象となるデータ（レコード）の量が少なければ主記憶のなかで行なう内部分類だけですむし，データ量が多ければ，系列の作成のための内部分類と，系列の数を減らすための外部分類を組合せて用いる必要がある。データ量の多少，レコードの大きさと形およびキーの大きさと形さらに使用できる装置によって，さまざまな技法が考えられ効率に大きく影響することを理解させる。

(3) 内部分類法

代表的な内部分類法は次のように分けられる。

- 挿入分類法
- 交換分類法
- 選択分類法
- 併合分類法
- 分配分類法

それぞれの分類法に対して，二，三のアルゴリズムを取りあげ，例を示しながらパスの回数，比較・転送の回数，必要とする領域の大きさなどについても説明する。

(a) 挿入分類法

キーの値が順番に並ぶように，適切な場所にそのレコードを挿入してゆく方法である。取りあげるアルゴリズムには次のようなものがある。

- 跳び跳び法（diminishing incremental sort）

- 番地計算法(address calculation sort)
- 直接挿入法(straight insertion sort)
- リスト挿入法(list insertion sort)

(b) 交換分類法

二つのレコードのキーを比較し，順序が逆の場合にはレコードの交換を行なうことをくりかえしてゆく方法である。代表的なアルゴリズムには次のようなものがある。

- 泡立ち法(bubble sort)
- 改良泡立ち法(cocktail-shaker sort)
- クィック法(quick sort)
- 併合交換法(merge exchange sort)
- 基数交換法(radix exchange sort)

(c) 選択分類法

ファイルのレコードのうちから，キーの値が最大(または最小)のものから順番に取出してゆく方法である。つぎのようなアルゴリズムがある。

- 直接選択法(straight selection sort)
- トーナメント法(tournament sort)
- ヒープ法(heapsort)

(d) 併合分類法

複数個の整列ずみの系列を併合するときに，それぞれの系列の先頭にあるレコードのキーの値を比較して，順番に並ぶように併合して一つの系列を作る手続きを繰り返していく方法である。アルゴリズムとしては次のようなものがある。

- ナチュラル併合／2 (natural two-way merge sort)
- 直接併合／2 (straight two-way merge sort)
- リスト併合法(list merge sort)

(e) 分配分類法

キーを構成する数字 (あるいは文字) の桁ごとに，数字の値によって異なる位置にレコードを分配してゆくことを繰り返していく方法である。レコードが大きい場合にはレコードの転記のコストが大きくなるため実際にはポインタを転記するだけですむようにする。このアルゴリズムとしては，

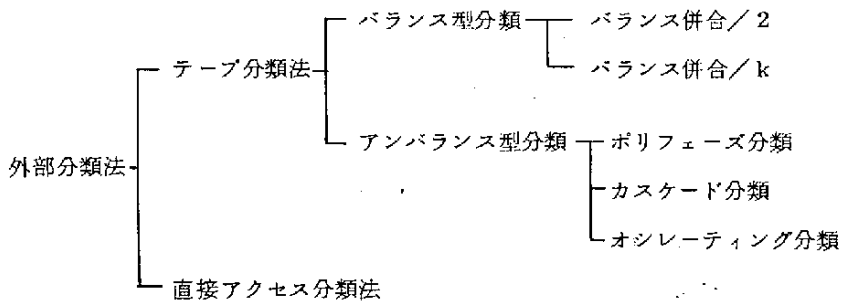
- 基数リスト法(radix list sort)

をとりあげる。

(4) 外部分類法

代表的な外部分類の技法は使用する装置によって大別され，表９－１に示すとおりである。

表９－１　代表的な外部分類法



以上のそれぞれの技法について，例を用いてアルゴリズムを説明し，長所や効率についても説明する。

(a)　バランス型分類

入出力装置として同数のテープ装置を用い，部分的に内部分類法を使って整列ずみの系列を作り，この系列を出力テープに分配し，次にはそれらのテープを入力テープとして使って併合を行なうということを繰り返していく方法である。入出力テープ装置としてそれぞれ２台の装置を用いるバランス併合／２（ balanced two-way merge sort ）と，それぞれｋ台の装置を用いるバランス併合／ｋ（ balanced k-way merge sort ）について説明する。

(b)　アンバランス型分類

入力テープ装置と出力テープ装置の台数が一致しない方法である。

・　ポリフェイズ分類（ polyphase merge sort ）

Ｎ－１本の入力テープを併合して１本のテープに出力するが，１本の入力テープが空になったときにそのテープを次のフェーズの出力テープとして併合を行なってゆく方法である。

・　カスケード分類（ cascade merge sort ）

Ｎ－１本の入力テープから併合／Ｎ－１を行ない，次にはＮ－２本の入力テープから併合／Ｎ－２を行なうというように，１回のパスごとに併合／Ｎ－１から併合／１（単なる転記）までを行ない，Ｎ－１本の入力テープが同時に空になるまでこのパスを繰り返す方法である。

・　オシレーティング分類（ oscillating sort ）

内部分類と外部分類を同時に行ないながらＮ本のテープを使ってＮ－１本のテープへの分配と，併合／Ｎ－１を交互に繰り返していく方法である。一般にはテープ装置に逆読みの機能

があるときにのみ使えることに注意する。

(c) 直接アクセス分類法

ディスク，ドラムなどの直接アクセス可能な装置の特徴を活かして，併合のときの次数を増したり，レコード全体を転送しないでポインタを効果的に使うなどして効率を高める方法がある。

直接アクセス分類法を用いるために，次の用語の意味を明確にし，アルゴリズムを考える際にその効率の検討に用いなければならない。

- シリンダ(cylinder)
- トラック(track)
- シーク時間(seek time)
- 回転待ち時間(latency time)
- 転送時間(transmission time)

(5) 分類法の効率比較

(a) 内部分類法の効率比較

効率比較の尺度として，パスの回数，作業領域の大きさ，比較の回数，転記の回数を考慮しなければならない。代表的な内部分類法についてこれらを比較したものが表９－２である。(比較・転記の回数は平均値である。)

表９－２　内部分類法の効率比較

		作業領域	パスの回数	比較の回数	転記の回数
挿入分類法	跳び跳び法	0	跳び方により異なる	$O(N^{1.2})$	$O(N^{1.2})$
	番地計算法	$2.5N$	1	$1.33N$	$1.66N$
	直接挿入法	0	$N-1$	$(N^2-N)/4$	$(N^2-N)/4$
	リスト挿入法	ポインタN語	$N-1$	$N^2/4$	$N^2/4$
交換分類法	泡立ち法	0	N	$(N^2-N)/2$	$(N^2-N)/4$
	クイック法	$\ell og_2 N$	$\ell og_2 N$	$N\ell og_2 N$	$\frac{N}{6}\ell og_2 N$
	併合分類法	0	$\ell og_2 N$	$N\ell og_2 N$	$N\ell og_2 N$
	基数交換法	$M-1$	N	$N\ell og_2 N$	$\frac{N}{4}\ell og_2 N$
選択分類法	直接選択法	0	N	$(N^2-N)/2$	$N(\ell n N+0.57)$
	ヒープ法	0	N	$N\ell og_2 N$	$\frac{N}{2}N\ell og_2 N$
併合分類法	ナチュラル併合/2	N	$\ell og_2 N-1$	$2N(\ell og_2 N-1)$	$N(\ell og_2 N-1)$
	直接併合／２	N	$\ell og_2 N$	$N\ell og_2 N$	$N\ell og_2 N$

N：レコード数，　M：キーのビット数

これらの効率を比較するにあたっては，入力レコードの分布を十分に考慮する必要があることを注意する。

(b) 外部分類法の効率比較

外部分類法の効率を比較するにあたり，系列の長さと個数，パスの回数，フェーズの回数，レコードの量と長さ，キーの長さなどの用語を明確にする。またテープ分類法と直接アクセス分類法は使用する装置の性質が異なるため別個に比較する。

テープ分類法の効率の尺度として処理量（処理される内部分類によって作られた整列ずみの単位系列の個数）を定義する。テープ分類法の効率比較は，表９－３のようになる。

表９－３　テープ分類法の効率比較

	パスの回数	処　理　量
バランス型分類	$\lceil \log_{T/2}(r-1)+1 \rceil$	$r\lceil \log_{T/2}(r-1) \rceil + r$
オシレィテイング分類	$\lceil \log_{T-2}(r-1) \rceil + 1$	$r\lceil \log_{T-2}(r-1) \rceil + r$
ポリフェーズ分類	$\lceil \log_q r - \log_q \sum_{i=1}^{T-1} i q^i + T + 1 \rceil$	$\frac{r \log_q r}{q} + r - \frac{T - \log_q \sum_{i=1}^{T-1} i q^i}{q}$

r：単位系列の個数
T：テープ台数
q：フィボナッチ根

表９－３より一般に次のようにいえる。

- オシレィテイング分類法は常にバランス型分類法よりも速い。
- $T \leq 10$ ならば，ポリフェイズ分類法は，バランス分類法よりも速い。
- $T < 6$ ならば，ポリフェイズ分類法は，オシレィティング分類法よりも速い。
- アルゴリズムはバランス型分類法，ポリフェイズ分類法，オシレィティング分類法の順に複雑になる。

直接アクセス分類法の効率は使用できる装置の特性によって大きく影響を受けるので単純に比較することはできない。一般には併合分類法が用いられるが，併合の次数と探索時間（シーク時間と回転待ち時間），転送時間の間には，図９－１の関係があるので，最適な併合次数を求める必要がある。

(6) 分類パッケージの利用

外部分類に関しては，COBOL，PL／Ⅰなどの高水準言語には分類パッケージが用意されているが，これらのパッケージの利用法および効率上注意すべき点について説明する。またア

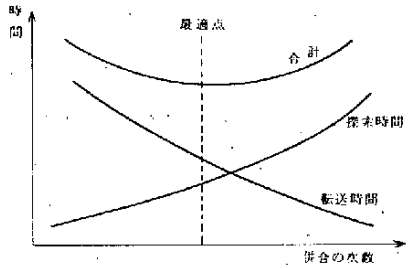


図９－１　併合の次数と，探索時間，転送時間との関係

センブラ，FORTRANなどからも使うことができるようになったユーティリティとして分類プログラムが用意されている場合もあるので，高水準言語の分類パッケージと同様の説明を加える。実際には自分でプログラムを作るよりもパッケージを利用することが多いので，特に効率面でのチェックの重要性を認識させる。

9.2 探　索

(1) 探索技法

多数のレコードのなかから，目的とするキーの値あるいは特定のフィールドを持つレコードを探し出すことを探索という。ここでは種々の探索技法のアルゴリズムについて説明し，探索法の効率の評価法として照合回数およびレコードの変更，挿入および削除のしやすさなどについて理解させる。

(a) 逐次探索法（ sequential search ）

対象となるレコードを先頭から順次調べてゆく方法である。レコードの順序がどうなっていても目的のレコードを捜すことができる点が特色である。

(b) キー比較による探索法

最も代表的な方法が２分探索法（ binary search ）で，対象となるレコードをキーの順に分類して整列しておき，まずその中央のレコードと比較して目的とするレコードがファイルの前半にあるか後半にあるかを判断し，つぎにはそのどちらかのサブファイルに対して同様の手続きを行なう方法を繰り返すものである。

そのほかの手法としてフィボナッチ数列を利用して２分探索と同様に探索を行なうフィボナッチ探索法（ Fibonacci search ）などがある。

(c) 木探索（ tree search ）

２分探索法などのキー比較による探索は，キーの順に整列された固定長の表に適している。レコードの追加，削除など表の構造を変える必要が頻発するような状況ではこの方法は表を

再構成する費用の点で不適当である。したがって表自身を2進木（ binary tree ）の形で持つことにすれば表の変更の効率も，探索の効率も良くすることができる。この考えに基いているのが2進木探索法（ binary tree search ）である。この技法はコンパイラの記号表などに多く用いられている。

主記憶に木構造を持ち，この木を探索したり変更したりする場合には，効率を保証するために平衡木（ balanced tree ）とする方が良い。

多数のレコードが直接アクセス装置に置かれている場合には，2進木の構造ではアクセス回数が多くなりすぎる。このようなときに有効なものとして，多進木（ multiway tree）とB－トリー（ B－tree ）がある。

(d) 見出し付き探索（ indexed search ）

キーを値として見ないで桁ごとに辞書式の見出しによって探索する方法である。桁ごとの文字そのものを見出しとして表を作るトライ探索法（ trie search ）と，桁ごとの文字を2進数で表わす計数木探索法（ digital tree search ）のアルゴリズムを説明する。

(e) ハッシュ法（ hashing ）

キーの値を引数として与えるとレコードの位置を返すハッシュ関数を用いて探索する方法である。ハッシュ法の利点，欠点について説明し，特にハッシュ関数の作り方，衝突が生じた場合の解決法などについて説明する。

(2) 探索技法の効率比較

探索技法の効率比較の尺度として，次のような項目が考えられる。

- 照合回数（ number of matching ）
- 挿　入
- 削　除
- 更　新
- 領　域

さらに探索および更新については次の四つを考える必要がある。

- 与えられたキーの値をもつ項目を探す。
- ファイルの先頭からk番目の項目を探す。
- 特定の場所に項目を挿入する。
- 特定の項目を削除する。

いろいろな探索技法の効率を比較すると表9－4のようになる。

表9－4　探索技法の効率比較

	平均照合回数	付加領域	更新の効率
逐次探索法	$(N+1)/2$	0	○
2分探索法	$\ell o g_2 N$	0	×
2進木探索法	$\ell o g_2 N$	レコードごとに左右のポインタ	○
トライ探索法	$\ell o g_2 N / \ell o g_2 M$	$CMN/\ell o g_2 M$	○

9.3　分類と探索の妥協点

分類と探索は互いに補いあう関係にあるので，これらの手法を用いるシステムに応じて，どちらに比重を置くべきかを決める必要がある。したがって分類と探索の妥協点を求めるためには次の事柄に注意する必要があることを説明する。

(1)　データの利用形態

データの入力の形態……ファイルを構成するときの処理形態（一括処理，時分割処理，即時処理）。

- データの探索の形態
- データの更新の形態……レコードの項目の更新だけ，あるいはレコードに対する更新も含めるかなど。

(2)　データの利用頻度

データの利用形態にもかかわることで，入力，探索，更新の頻度によって最適なファイル構成と分類，探索の技法を選択する必要がある。

(3)　データの保存

データの利用形態，頻度に大きく影響を受けるが，多量のデータの場合には当然主記憶だけではすまない。最も多く用いられる外部記憶装置として磁気テープ，ディスク，ドラムがあり，さらに穿孔カード，磁気カードなどの装置が考えられる。これらの装置の特徴を理解させ，利用形態に合った装置と手法が選択できるようにする。

9.4　分類と探索における情報構造の効果

情報をどのような構造のもとで処理，保存すべきかを決定するためには，次のような事柄について考慮する必要があることを説明する。

- レコードの量
- レコードの長さ

- 固定長レコードが可変長レコードか
- キーの長さ
- キーの種類
- キーとレコードの長さの比率
- 主記憶の大きさ
- 使用できる外部記憶装置の種類
- データの利用形態
- レコード内の項目の個数，内容，配置
- 表（ table ）の構造……単純な表，多段表，ハッシュ表，見出し付き表など
- リスト（ list ）の構造……一方向リスト，二方向リスト，環構造など，およびポインタの量

指導上の留意点

(1) すべてのキーの分布に関して最善な分類，探索技法はなく，利用形態によって効率が大きく変わることを理解させる。

(2) 分類，探索ともに，基本的な技法のアルゴリズムを理解させることを第一の目標とする。このときには必ず例を用いて理解を助けるようにする。

(3) 効率に関する理論的解析には深入りしない。多くの応用システムでは必ずしも理論的な効率が達成できるわけではないので，効率のオーダーが重要である。

(4) 分類，探索をそれぞれ単独に理解するのではなく，データの利用目的に合った技法を選択し，組合せて利用できるようにする。そのためには実際に出合う事例をとりあげ，これを例として説明することが望ましい。

(5) 分類，探索は単なる手法のアルゴリズムだけではなく，データ構造，データベース・システムなどとも密切な関連を持つことを理解させる。

(6) 使用できる装置によって分類，探索の効率は大きく変わってくるので，アルゴリズムだけで手法を決定すべきではない。

参　考

内部分類法のうちクイック法とヒープ法について解説する。どちらの場合にもレコードはキーだけから成るものとして配列$A[1:N]$に入っている。キーを昇順に分類することを考える。

(1) クイック法

クイック法のアルゴリズムは次のように書ける。

(i) (a) Aの任意の要素xを選ぶ。

(b) Aの左端から$A[i]>x$であるような要素が見つかるまで捜す。

(c) Aの右端から$A[j]<x$であるような要素が見つかるまで捜す。

(d) $A[i]$と$A[j]$を交換する。

(e) iとjが一致するまで(a)~(d)の手順をくりかえす（この結果Aは，xを境に左側にはxより小さな要素，右側にはxより大きな要素だけからなる二つの配列に分割されたことになる。）。

(ii) それぞれの部分配列に対して，部分配列の要素が1個だけになるまで(i)の手順を繰り返す。

このアルゴリズムは再帰的な手順であるが，これを展開して繰り返しとして表わすことができる。このとき配列を1回分割するたびに二つの部分配列ができるが，一方の部分配列について処理を続けるためには，残りの部分配列についての記録をスタックに保存しておく必要がある。このスタックに部分配列の左端と右端の添字を記録するとして，そのときのスタックの大きさは高さ$\ell og_2 N$だけあればよいことに注意する。

次にクイック法の効率を考える。上の手順(i)で行なう比較の回数はNであり，一方手順(i)で常にキーの中央値を選んで分割したとすれば，パスの回数は$\ell og_2 N$回である。したがって全体の比較の回数は，$N\ell og_2 N$回となる。手順(i)での交換回数は，

$$\frac{1}{N}\sum_{x=1}^{N}\frac{N-x}{N}(N-x+1)=\frac{N}{6}-\frac{1}{6N}\fallingdotseq\frac{N}{6}$$

となり，全体での交換回数は，$\frac{N}{6}\ell og_2 N$となる。この考察の基礎として分割のとき常に中央値を選ぶことを仮定しているが，実際に中央値が選ばれる確率は$1/N$にすぎない。しかし分割を任意に行なっても効率は，$2\ell n\ 2$倍悪くなるだけである。xとして配列の左端または右端の要素を選ぶようにすると，すでに整列ずみの配列が与えられたときに効率は最悪となる。このとき(i)の各パスごとに$N-1$個と1個の配列に分割されるので，N回のパスが必要となり，効率は$O(N^2)$となってしまう。したがってxの選び方に注意する必要がある（一つの方法として配列の端の要素ではなく，ちょうど真中の要素を選ぶようにするとよい。）。

(2) ヒープ法

配列Aが2進木を表わしていると考えると，$A[k]$の親は$A[k/2]$となり，$A[k]$の子は$A[2k]$と$A[2k+1]$となる。このとき$A[1]$, $A[2]$, …; $A[N]$がヒープであるということは，$1<j/2<j<N$に対して，$A[j/2]\geq A[j]$という関係，すなわちすべての要素に対して親$\geq$子という関係を満足することである。

ヒープ法のアルゴリズムは次のように書くことができる。

(i) $A[1], A[2], \cdots, A[N]$ からヒープを作る。

(ii) ヒープの要素の個数が1になるまで，次の手順を繰り返す。

(a) 根（$A[1]$）にあるデータをとりだし，最も右の葉にあるデータを根に持ってくる。この操作によって要素数はN－1になる。

(b) 再びヒープを作る。

このアルゴリズムは簡単に繰り返しとして書ける。ヒープ法の効率を考えてみると，(ii)(a)の選択の手順はN回繰り返さなければならない。そのために必要な比較回数は $\ell og_2 N$ 回である。したがって選択のための全体の手数は，$O(N\ell og_2 N)$ である。最悪の場合を考えると，(ii)(b)の手順を$N/2$回繰り返す必要があるが，その場合でも全体の手数は$O(N\ell og_2 N)$ですむ。一方交換の回数は平均的に$\frac{1}{2}N\ell og_2 N$であり，最悪の場合でもこれに近い回数ですむ。

参考文献

〔1〕I.Flores,"Computer Sorting," Prentice-Hall, 1969.
関根智明訳「コンピュータ・ソーティング」サイエンス社，サイエンス・ライブラリ情報電算機＝14，1972

〔2〕D.E.Knuth," The Art of Computer Programming Vol.3," Sorting and Searching, Addison-Wesley, 1973

〔3〕N.Wirth," Algorithms + Data Strnctures = Programs", Prentice-Hall, 1976.

第10章　情報構造の使用例

キーワード

記号表（symbol table），ハッシュ表（hash table），構文木（sytax tree），スタック（stack），ヒープ（heap），スケジューリング・リスト（scheduling list），先入れ先出し待ち行列（FIFO queue），優先順位待ち行列（priority queue），タスク制御ブロック（task control block），ファイル制御ブロック（file control block），ページ・テーブル（page table），セグメント・テーブル（segment table），コーリング・スタック（calling stack），ファイル・ディレクトリ（file directory），木構造ファイル（tree structured file），逆ファイル（inverted file），チェイン構造（chain structure），階層構造（hierarchical structure），ランダム化技法（randomize technique），関係形式モデル（relational model），TDMS，IDS，IMS，プレックス（plex），イベント・リスト（event lists），リストによる文字列表現（character string representation by licts），線形記憶による文字列表現（character string representation by linear memory），文字列操作（character string manipulation），連想三つ組（asspciative triple），ASP（associative set processor），LISP，SNOBOL，L^6，探索木（search tree），$\alpha-\beta$法（$\alpha-\beta$ method），パターン照合（pattern match）

目　　標

情報構造が具体的に使用される場面として，言語プロセッサや制御プログラムなどのシステム・プログラム，データベース，および，図形処理や記号処理などの応用プログラムを考える。

言語プロセッサでは，第一に，原始プログラムから目的プログラムまで各処理の段階においてあらわれる情報を表現するために用いる情報構造がある。第二に，それらの情報を処理して，言語プロセッサの機能を実現するために用いる情報構造がある。

制御プログラムでは，第一に，制御の対象となるもの，たとえば，タスクやファイルなどの表現に用いる情報構造がある。第二に，それらの情報を制御して，制御プログラムの機能を実現するために用いる情報構造がある。

データベースにおいては，代表的なデータベース管理システムで用いられる情報構造を考え，データベースの機能と関連づける。

その他の応用プログラムにおいては，各分野でよく使用される特殊目的の言語を例にして，それぞれの分野の特徴的な情報構造を説明する。

各事例において，そこで用いる情報構造を選択する理由を説明し，各情報構造の特徴を十分理解させる。また，異なった情報構造を用いた場合を想定し，システムに対する影響を検討することにより，各情報構造の比較を行なう。

内　　容

10.1　言語プロセッサにおける情報の表現

本節では，言語プロセッサの各段階においてあらわれるいろいろの処理対象の表現をするため，およびそれらを処理して言語プロセッサの機能を実現するために用いられる情報構造の例を説明する。[1]

(1)　文字例(character string)

言語プロセッサの第一段階である字句解析部は，原始プログラムの文字列を走査して，字句に関する単位に分解する。文字列の表現は，

①　一字づつの走査が効率よく行なわれる，

②　適当な長さの列が処理できる，

を考慮して行なわれることを理解させる。

具体的には，

①　連続した記憶領域

②　線形リスト

などによって文字列を表現する。文字列の分割や連結を頻繁に行なう場合には，リストを用いた方がよいが，一定方向に一字づつ走査するだけならば，連続した記憶領域を用いる方が簡単であることを理解させる。

一般に，原始プログラムは，カード読取り装置，ファイル装置，端末装置などを介して，ユーザから与えられる。この場合，それらの周辺装置に対する入力操作と，文句の走査との整合が必要になる。たとえば，論理レコードと物理レコードとの切れ目の問題，物理レコードの後半の空白の処理，コメントなど不必要な部分の飛び越しの処理などが必要になることを理解させる。また，字句走査の能率を上げるためには，バッファリング技法を用いればよい。

(2)　構　文　木

言語の構文を解析して，対応する構文木を作るのが構文解析部の仕事である。構文木が構成されれば，それを走査して意味解析ができ，コード生成を行なうことができることを説明する。また，構文木の等価変換を行なう全域的な最適化，レジスタの割当ての最適化なども可能にな

る[2]ことを理解させる。

構文木の実現方法は，次のような方式が考えられる。[3]

① リストを用いた2進木による表現

② ブレックスを用いたn分岐木による表現

③ 木の線形表現（カッコなどの区切り記号を利用した表現）

④ ポーランド記法または逆ポーランド記法

⑤ n組データ項目による表現

①および②は，リストまたはブレックスというデータ項目単位に対応する記憶領域の動的割り当てが必要になるのに対して，③，④，⑤では，その必要はない。一方，木の処理アルゴリズムは，①，②の方が簡単になる。それぞれの表現方式の特徴を，図10－1に示すような具体的な例を用いて充分に理解させることが望ましい。

(3) 中間コード

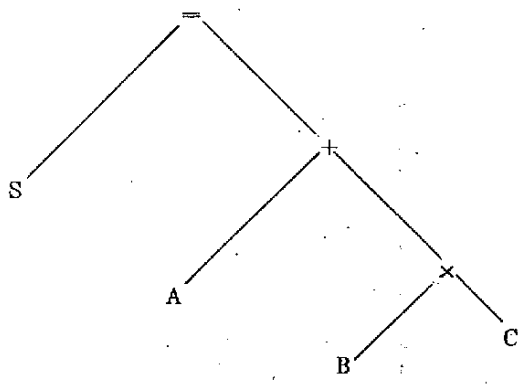


図10－1 構文木の例

S＝A＋B×C

言語プロセッサの各パス（字句解析部，構文解析部，最適化部など）の出力は，中間コード（intermediate code）として表現され，次のパスへの入力となる。構文木は，中間コードの一種と考える。コード化の方針は，処理する言語および言語プロセッサの設計方針により異なり，次のような中間コードの表現形態があることを理解させる。

① 列…… 中間コードをあらわすデータ項目を線形に並べたもの。

② リストまたはブレックス

…… データ項目をポインタを用いて関係づけたもの。

③ 表…… 表の記述項をポインタを用いて関連づけたもの。

表現形態は，それによってあらわされている情報の性質に合致したものを選択する。一つの

パスが出力した中間コードが，上述の形態の混合したものでもよいことを説明する。

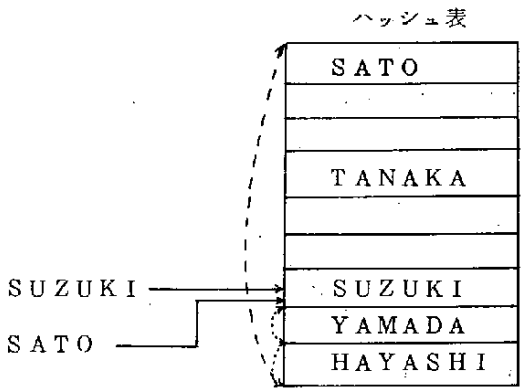
(4) ハッシュ表（hash table）〔1〕,〔4〕

言語プロセッサで処理するプログラムにあらわれる記号名を登録する記号表（symbol table）を効率よく実現するために，一般に，ハッシュ表が使用される。記号から表の記述項のアドレスを計算するために用いるハッシュ関数として，割り算法，中間二乗法，折り畳み法などを説明する。また，同一の記述項でぶつかり合うオーバーフローを処理するために，図10－2に示すように，オープン・ハッシュ法，チェインを利用する方法などを説明する。

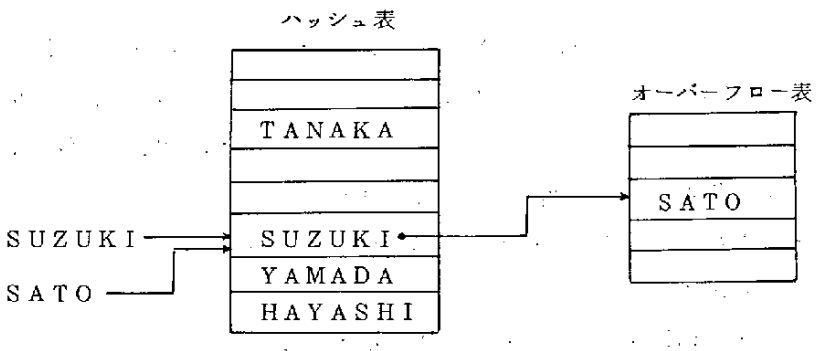
線形探索など，他のアルゴリズムを用いた記号表と効率の比較をして，ハッシュ表の長所や特徴を理解させる。このようなランダム化技法は，記憶管理の必要ないくつかの応用システムでも広く利用されている。

(5) スタック（stack）

スタックは，構文解析のように，再帰的な処理をする場面で利用される情報構造である。図10－3に示すような簡単な数式の処理を例にして，スタックの利用法を説明する。



(1) オープン・ハッシュ法



(2) チェインを利用する方法

図10－2 ハッシュ表のオーバーフローの処理

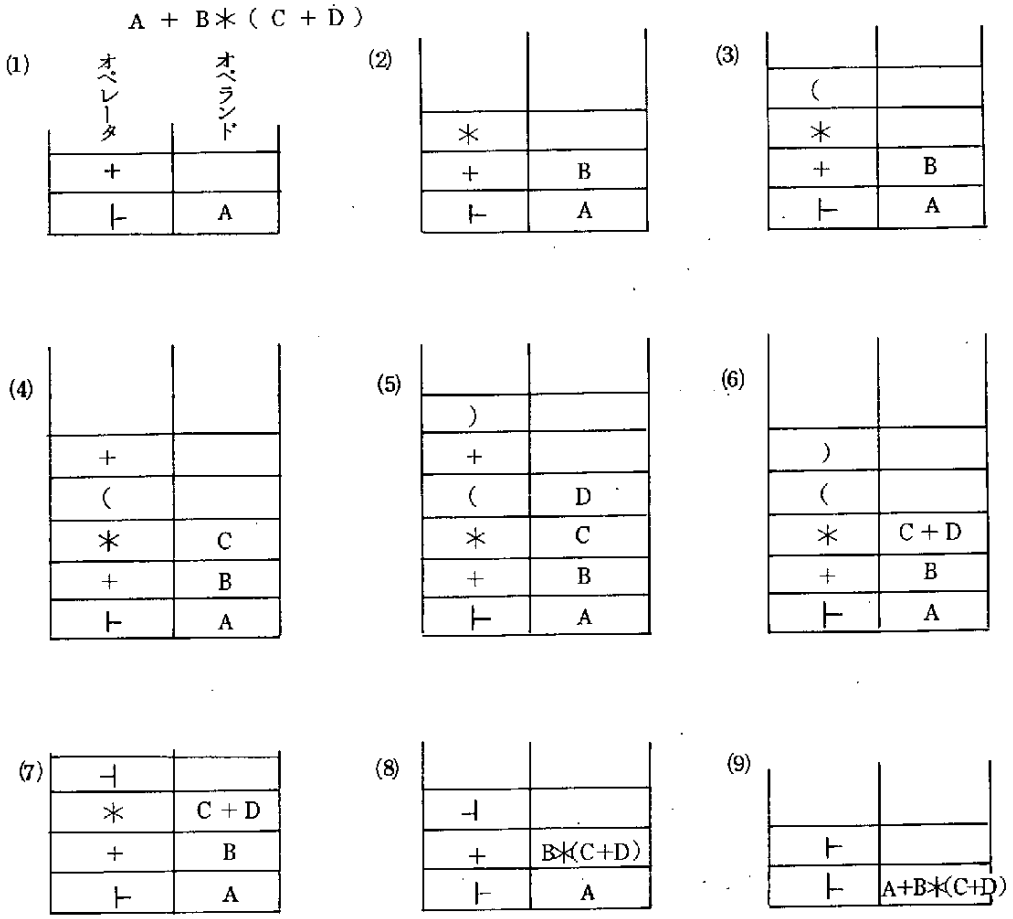


図１０－３　スタックを用いた数式の処理

ＡＬＧＯＬ系の言語では，実行時にスタックまたはヒープ（heap）が必要である。そのような言語の目的コードは，スタックまたはヒープを使用するように作られている。図１０－４に示すような簡単なプログラムを例として，実行時の状態の変化を理解させることが望ましい。

10.2　制御プログラムにおける情報の表現

本節では，オペレーティング・システムの制御プログラムにおいて，プロセス（タスク），ファイル，ハードウェアの装置など，制御の対象となるものを表現し，それらを管理するために必要な情報構造の例を説明する。[5]

(1)　プロセス（タスク）制御ブロック（process(task) control block ）

プログラムの独立な制御をプロセス（process）またはタスク（task）という概念であらわす，これを制御プログラムで扱うとき，具体的なデータ，すなわち，プロセス制御ブロックであらわす。ここには，プロセス識別子，優先度，ＰＳＷ退避領域など，プロセスの管理に必要な情報が含まれていることを理解させる。プロセス制御ブロックは，プロセスの管理の仕方

```
begin
    integer a, b ;
      ⋮
    begin
        integer  b , c ;
          ⋮
        begin
          integer c , d ;
            ⋮
          a := a+b × c ;
            ⋮
        end
      ⋮
    end
  ⋮
end
```

図10－4　ALGOL のプログラム

により，固定した領域に静的に割当てられるか，浮動する領域に動的に割当てられる。両者の得失を，簡単に説明する。

(2)　ファイル制御ブロック(file control block)

ユーザのデータの論理的なまとまりを，ファイル(file)とよぶ。これを，制御プログラムで管理するために，具体的なデータ，すなわち，ファイル制御ブロックであらわす。ここには，ファイル識別子，ファイルの属性，入出力領域へのポインタ，入出力状態語の退避領域など，ファイルの管理に必要な情報が含まれており，ユーザ・プログラムからファイルへの入出力操作は，ファイル制御ブロックを介して行なわれることを理解させる。ファイル制御ブロックは，一般に，ユーザ・プログラムの領域内にあらかじめ定義しておくが，これを動的に割当てると，その管理が難しくなることを理解させる。

(3)　デバイス・テーブル(device table)

コンピュータ・システムが備えている周辺装置を管理するために，デバイス・テーブルを用意する。これは，各装置の論理アドレス(記号装置名)，物理アドレスなどを含む一定の大き

さの表である。特に，ディスク装置では，オン・ライン・カタログと呼ばれるテーブルを用意し，現在使用可能なボリュームをそこに登録することを説明する。

(4) スケジューリング・リスト（scheduling list）

処理装置や周辺装置に対して，複数個のプログラムから使用の要求が出されたとき，待ち行列（または，スケジューリング・リスト）を用いて，その要求を管理する。また，ジョブの処理のスケジューリングも，この種のリストを用いて行なう。

待ち行列の並べ方には，

① 先 入 先 出

② 優 先 度 順

③ 多 段 リ ス ト

などがあることを説明する。このようなスケジューリング・リストを実現するためには，

① リ ス ト 構 造

② 表 構 造

などを用いる。図１０－５に示すように，優先度順に並べる行列を例にとり，

① 挿 入

② 削 除

③ 探 索

の操作を実現するアルゴリズムを具体的に理解させる。

(5) 仮想記憶におけるアドレス変換表（address translation table）

仮想記憶の管理をするためには，ページ・テーブル（page table）およびセグメント・テーブル（segment table）というアドレス変換用の表を用いる。セグメント・テーブルには，セグメントに対するページ・テーブルへのポインタが，ページ・テーブルには，ページに対す

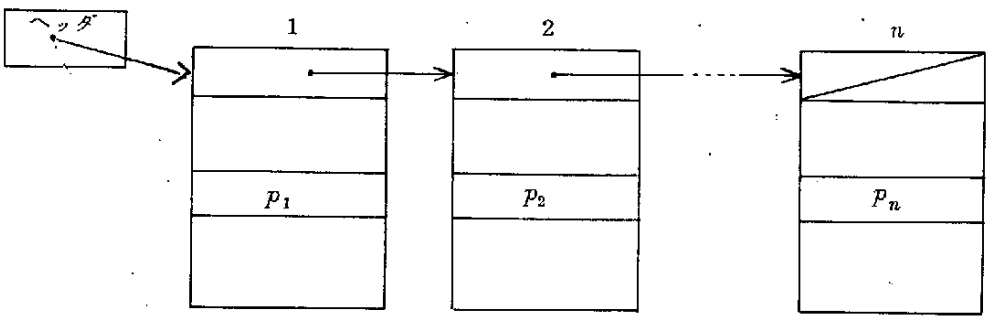


図10－5　優先度順に並べた待ち行列

る主記憶のブロック（ページ・フレーム）へのポインタが書かれている。図１０－６に示すように，一つの論理アドレスから，その実効アドレスを求める方法を具体的に説明し，この種の

テーブルを主記憶内に実現する方法を理解させる。

このようなテーブルを用いたアドレス変換の速度を向上させるためには，連想記憶（associative memory），または，表探索バッファ（Table Look-aside Buffer-TLB）を利用するのが一般的である。この種の連想記憶の動作と，それを用いる意味とを理解させる。ここで用いる連想記憶は，ハードウェアで実現されるが，連想記憶または内容アドレス付け可能記憶（content addressable memory)の機能は広く利用されるので，それをソフトウェアで実現する方法を考えさせるとよい。

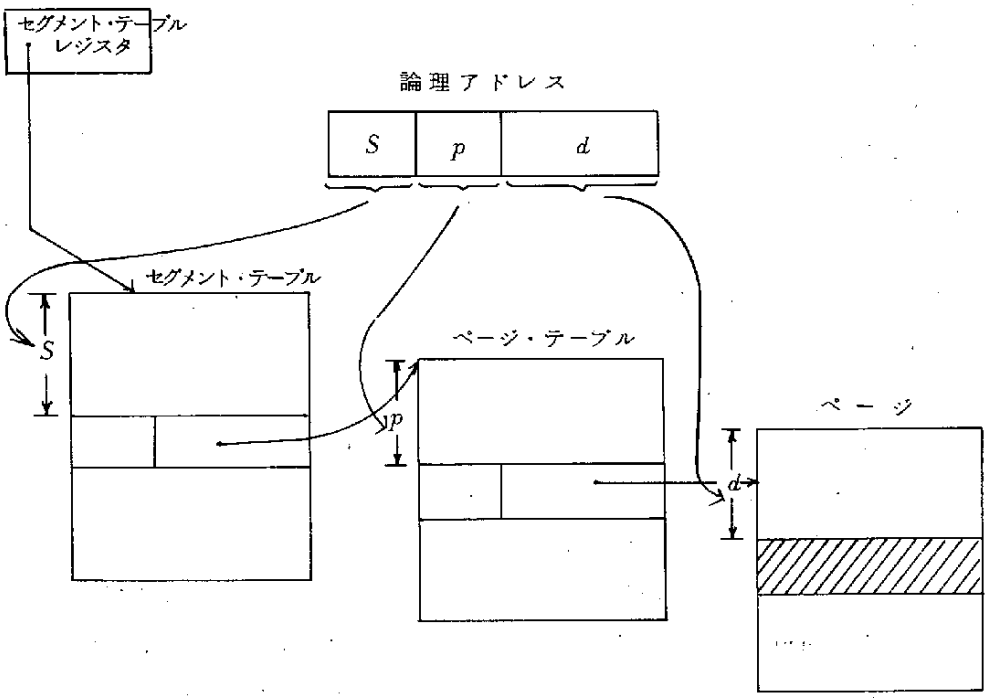


図１０－６　論理アドレスから物理アドレスへの変換

(6) 記憶管理表

記憶空間の分割割当てを行なうとき，記憶の使用状況をあらわすために，一般に，記憶管理表を用いる。記憶の管理では，未使用の空間は，なるべく大きくつなげておくことが原則である。空間の割当ては，

① 初　適　合（first fit）

② 最 適 適 合（best fit）

③ バディ・システム（buddy system）

など，いくつかのアルゴリズムを適用して行なわれることを説明する。

このような記憶管理表は，

① リ ス ト 構 造

② 表　構　造

などを用いて実現されることを理解させる。

(7) 階層構造ファイル（hierarchical structure file）

ファイルを管理するためには，図 10—7に示すように，ディレクトリの階層構造（または，木構造）を利用する。これは，オンライン・ファイルで特に必要であることを理解させる。木

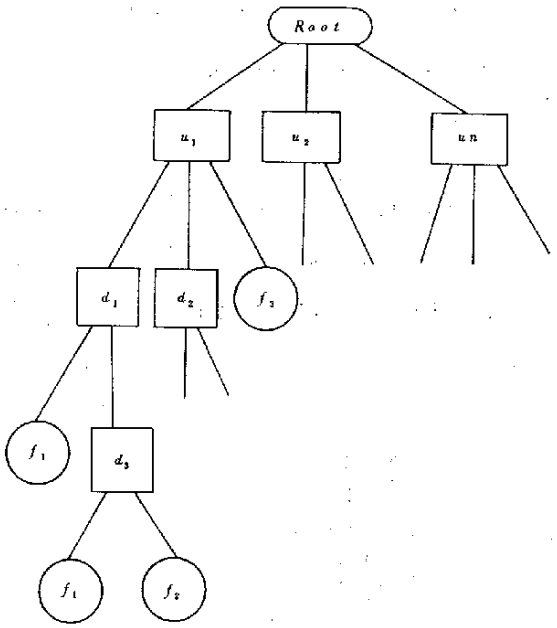


図１０－７　階層構造ファイル

構造は，根から任意の節までの路が一意的にきまるという性質をもつので，ファイルの識別名を一意的に与えることに役立ち，機密保護の実現ができる。一方，ファイルの公開をするためには，木構造の一部を変更して，網構造にしなければならない。

木構造を具体的に実現するために，ディレクトリがもつべき情報を考え，また，ファイルのオープン，クローズ，生成，消滅，公開などの操作を，木構造ディレクトリを用いて具体的に説明することが望ましい。

(8) コーリング・スタック（calling stack）

サブルーチンのリターン・アドレスは，一般に，コーリング・スタックを用いて制御できるので，リエントラント・ルーチンを管理するためには，プロセス制御ブロックにコーリング・スタックを設ける。このようなスタックの動きについて，具体的なプログラムと対応づけて

説明する。

10.3 データベースにおける情報構造[6]

この節では，データベースで用いられる基本的な情報構造を説明し，データベースの機能と関連づけてそれらの得失を理解させる。

(1) 階 層 構 造

階層構造（または木構造）の基本構造としたデータベース管理システムとして，ＩＭＳ（Infnrmation Management System）[7]がある。たとえば，図10.8に示す例を用いて，ＩＭＳのデータ構造の特徴を理解させる。アクセスの能率を上げるために，物理的な記憶方式を工夫し，階層順次編成（Hierarclnical Sequential）と階層直接編成（Hierarchicel Direct）の二つのアクセス方式が使える。

(2) チェイン構造

レコード内に任意個のポインタ領域を設け，同一クラスのレコードを環状リストにする構造を，チェイン構造という。レコードのクラス間の主従関係は，バックマン線図（Bach man Diagram）で表わされる。図10.9に示す具体例を用いて，チェイン構造と，バックマン線図との関係を理解させる。

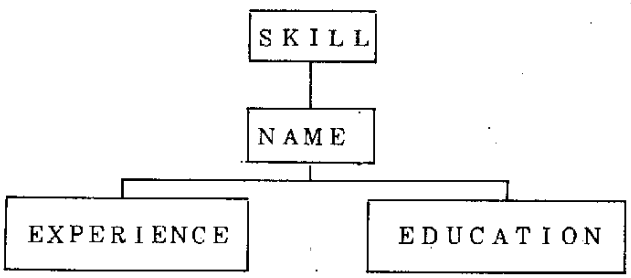


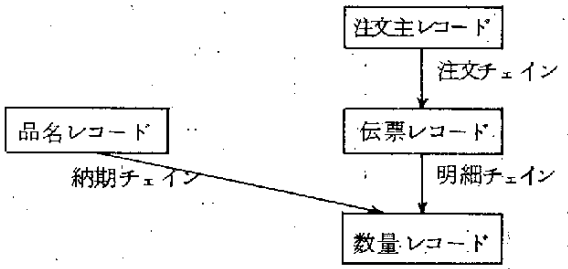
図１０－８　ＩＭＳのデータ構造の例

チェイン構造を基本としたデータベース管理システムとして，ＩＤＳ（Integrated Data Store）[8]がある。これは，CODASYL方式のデータベース・システム[9]の基礎となっている。ＩＤＳでは，アクセスの効率をあげるために，ランダム化技法を利用している。また，同一のチェインにつながるレコードは，隣接した記憶領域に格納する。論理構造と物理構造との関係を，システムの効率と関連づけて説明することが望ましい。

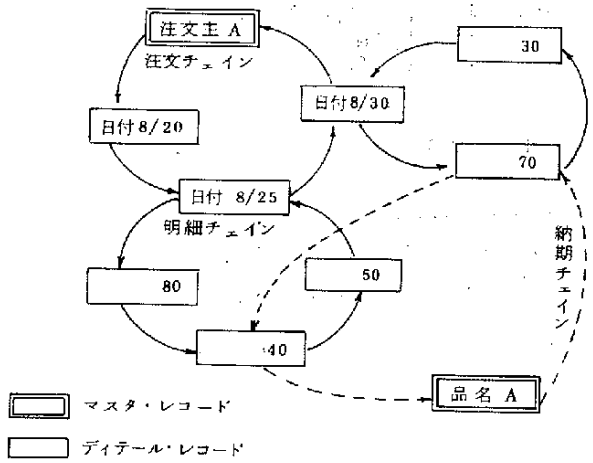
(3) 逆ファイル（inverted file）

逆ファイルは，レコードの任意のフィールドに対するキーと，そのキーをもつレコードのア

ドレスとの対応表を与えてある情報構造である。データベースに対する検索機能を効率よく行なうことを目指した構造であり，一種の連想記憶あるいは内容アドレス付け可能記憶が実現されている。逆ファイルを基本構造としたデータベース管理システムとして，ＴＤＭＳ（Time-shared Data Management System）[10]がある。図10.10 に示すＴＤＭＳの構成を例にし



(1) バックマン線図



(2) チェイン構造

図１０－９　ＩＤＳの構造

て，逆ファイルの特徴を理解させる。

(4) 関係形式モデル（relational model）

データベースで扱う情報を，表の形で表現し，物理的な表現を考慮せずモデル化した構造を，関係形式モデル[11]という。このようなモデルを確立すれば，データの独立性が実現でき，データベースの拡張，変換などが物理的構造や環境に依存せずに行なえることを説明する。

関係形式モデルの基本概念を，図１０－１１に示す例を参考にして，理解させる。表の各欄に対応するｎ個の集合A_1，A_2，……，A_n の上のn項関係，すなわち，直積$A_1 \times A_2 \times$

$\cdots\cdots \times A_n$ の任意の部分集合を，集合 A_1，A_2，$\cdots\cdots$，A_n 上の関係（relation）とよぶ。関係は集合と同一視してよいから，これに，和，差，積，商，射影などの集合演算が定義でき，それらを組み合わせて，照会，回答などの演算が可能になることを説明する。

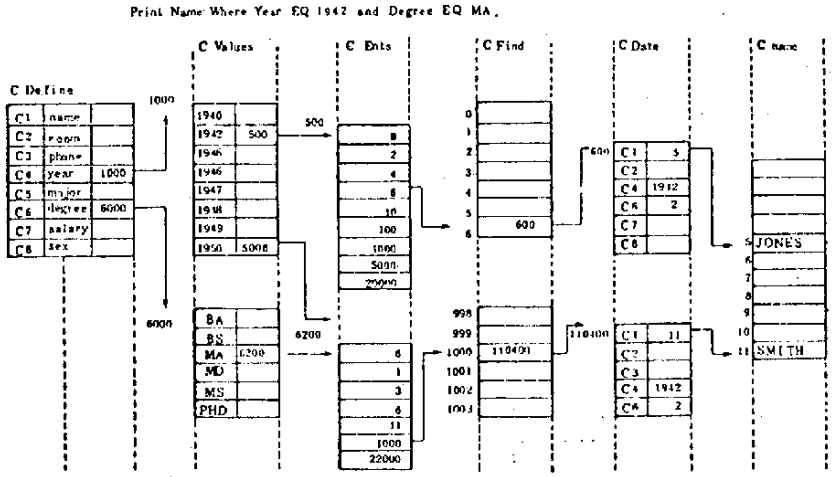


図１０－１０　ＴＤＭＳにおける逆ファイルの構成

(1) 図　　式

$$R:\quad \begin{array}{cccccc} A_1 & A_2 & \cdots\cdots & \cdots\cdots & A_n \\ \hline a_1^{1} & a_2^{1} & \cdots\cdots & & a_n^{1} \\ a_1^{2} & a_2^{2} & \cdots\cdots & & a_n^{2} \\ & & & & \\ a_1^{m} & a_2^{m} & \cdots\cdots & & a_n^{m} \end{array}$$

(2) 具 体 例

鈴　木	男	31	東　京
山　本	女	28	川　崎
⋮	⋮	⋮	⋮
田　中	男	25	横　浜

図１０－１１　関係形式モデルの例

10.4　その他の応用分野における情報の表現

この節では，前の三節で述べた分野以外で用いられる情報構造の例を説明する。

(1)　コンピュータ・グラフィックス

コンピュータ・グラフィックスで扱う図形は，点，線，面などの基本要素が，相互に関連して構成されている。これらを詳細に表現し，柔軟にその操作を行なうために，いろいろの情報構造が提案されてきたことを説明する。これらの情報構造は，一つのプログラム言語の機能として提案されている。

任意の大きさのデータ項目をポインタで関連づけるプレックス（plex）構造は，複雑な構造を柔軟に記述できる。図１０－１２の例を用いて，プレックス構造の特徴を理解させる。APL言語[12]は，プレックスを基本構造としており，また，前節に述べたＩＤＳデータベースは，同様の情報構造をファイル記憶上に実現している。また，データ項目の役割をあらかじめ定めて，それらをリング状に結合して情報構造をあらわすＡＳＰ（Associative Structure Package）[13]を，図１０－１３を用いて説明する。これは，ring，ring start，associator，element，dataの五つの型から成り立つ。これらの情報構造の利害得失を比較して理解させるとよい。

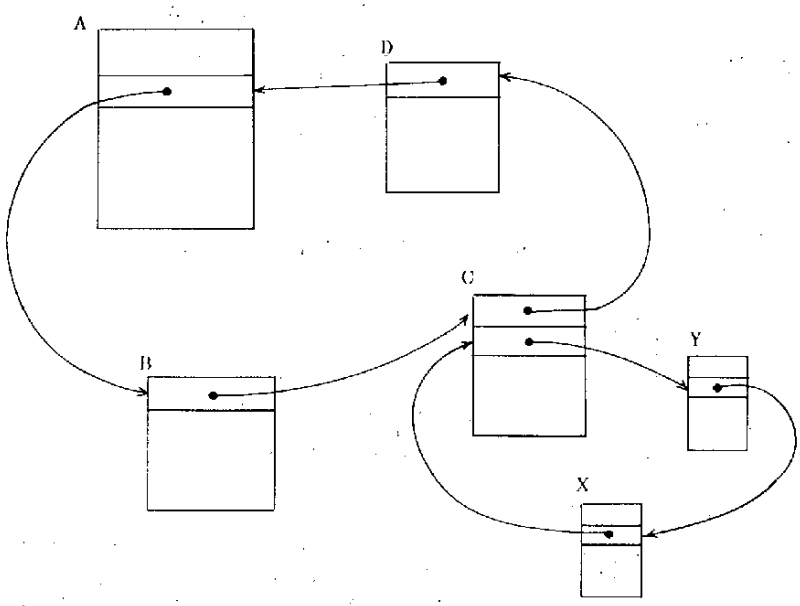


図１０－１２　プレックス構造

(2)　記　号　処　理

文字列であらわした記号の処理は，言語処理，編集システム，数式処理など多くの分野で利

用されている。文字列の連結，分割，パターン照合などの操作が，効率よくかつ柔軟に行なえるように，情報構造を工夫していることを理解させる。

ＬＩＳＰ[14]は，図１０－１４に示すように，一つのセルを二つの領域に分け，それぞれの領域の内容を参照する関数（car およびcdr）と，新しいセルの二つの領域に内容を設定する関数（cons）とを基本操作として用いるシステムである。これによって作られる構造は，２進木であ

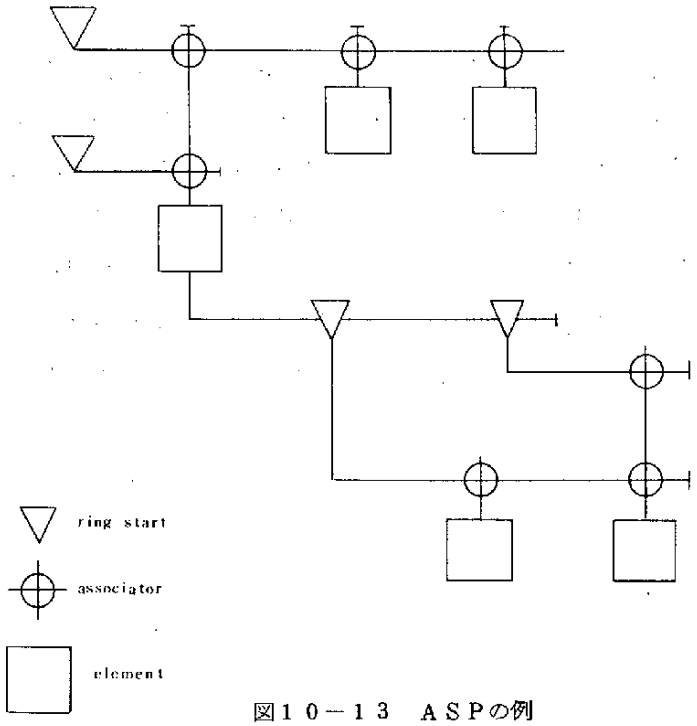
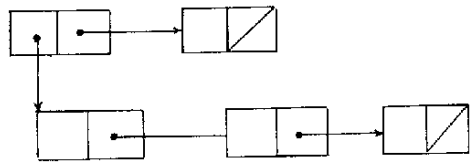
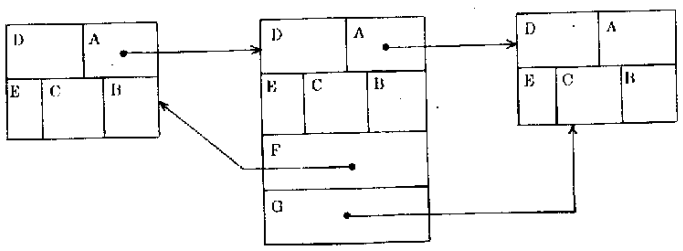


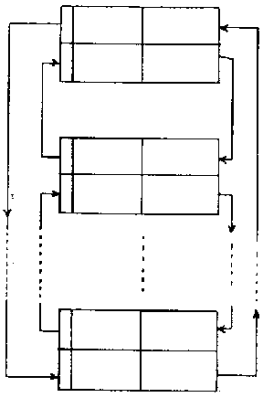
図１０－１３　ＡＳＰの例



(1)　ＬＩＳＰの構造



(2)　L^{A}の構造



(3) ＳＬＩＰの構造

図１０－１４　記号処理言語の構造

り，複雑な情報構造を表現できる。

L^{6}[15]は，2^{n} 語をまとめたブロックを基本単位とし，各ブロックには，二語にわたらない範囲で領域を設定し，これに，数，文字，ポインタなどを格納する。ブロックと領域の定義は，ユーザに任せられているので，問題に応じて柔軟な設定ができる。各領域は，英字大文字または数字で名前をつけるので，図１０－１４に示すように，文字列によって任意の領域の内容を指定できる。

ＳＬＩＰ[16]では，図１０－１４に示すように，一つのセルに前方向（ＬＮＫＬ）および後方向（ＬＮＫＲ）のポインタをもたせる。このシステムは，ＦＯＲＴＲＡＮで実現できるので，便利に使用される。

これらの情報構造の利害得失を，具体的な例を用いて理解させることが望ましい。

(3) 人　工　知　能

ゲームやパズルのような問題は，ゲーム木あるいはＡＮＤ／ＯＲ木という木構造を用いてその状態をあらわし，木を探索することによって解に到達する。ＡＮＤ／ＯＲ木の探索を，効率よく行なうために，$\alpha-\beta$ 法を用いて探索の一部を省略する。

定理の証明などのプログラムでは，n 組のデータ項目のパターンの照合が基本的な操作となり，それを効率よく行なえる様に情報構造を設計しなければならない。

上述のプログラムを記述するために，従来からＬＩＳＰおよびその拡張版が使用されてきたことを説明する。ＬＩＳＰは２進木が基本構造であり，また，再帰的プログラムの記述が容易

に行なえる。また，PLANNER[17]は，パターン駆動型の手続き呼び出しが可能である。

パターンの照合は，連想記憶の一つである。連想記憶を実現する一つの方法として，連想三つ組(associative triple)[18]が提案されていることを説明する。そこでは，データの項目間の関係を，次の三つの概念であわわす。

① 対象物(Object ; O)

② 属性(Attribute , A)

③ 値(Value , V)

このとき，〈A，O，V〉またはA(O)＝Vと書き，この三つ組がデータを代表する。そこで，A，O，Vの任意の一対に対して具体的な値を指定して，それに合致する組をすべて取り出すような連想機能を考える。

F0：A(O)＝V

F1：A(O)＝x

F2：A(x)＝V

F3：A(x)＝y

F4：x(O)＝V

F5：x(O)＝y

F6：x(y)＝V

ここで，x，yは変数。

連想三つ組構造では，上述の七つの関数が高速に処理できるように，ハッシュ表などを用いてデータを構成する。

以上に述べた情報構造の利害得失を，具体的な問題を記述しながら理解させることが望ましい。

指導上の留意点

(1) 情報構造の使用例は，具体的な問題を用いて説明するとよい。

(2) 同一の問題をいくつかの情報構造で記述するときは，それぞれの長所や短所がわかるように説明する

(3) 情報構造の選択は，与えられた機能をできるだけ効率よく実現できるように行なわれることを理解させる。

参考文献

〔1〕 F.R.Aホップグット著，首藤・関本訳「コンパイラの技法」サイエンス社，1973

〔2〕 中田育男著「コンパイラの技法」竹内書店新社, 1971

〔3〕 D.Gries,"Compiler Construction for Digital Compnters", John Wiley, 1971

〔4〕 R.Morris,"Scatter Storage Techniques", CACM, 11, 1968

〔5〕 マドニック／ドノバン著, 池田克夫訳「オペレーティング・システム」日本コンピュータ協会, 1976

〔6〕 データベース特集号, 情報処理, Vol 17, №10, 1976

〔7〕 IMS（情報処理システム）／360 概説書, N GH20—0765—2, IBM, 1973

〔8〕 インテグレーテッド・データ・ストア（IDS）プログラミング説明書, TOSBAC—5600, 東芝

〔9〕 CODASYL Systems Committee,"Feature Analysis of Generalized Data Base Management Systems", May 1971. 西村恕彦, 中川友康訳「データベースシステム」bit, 臨時増刊, 共立出版, 1973

〔10〕 R.E.Bleier,"Treating hierarchical data structures in the SDC time-shared data management system(TDMS)", ACM National Conference, 1967

〔11〕 E.F.Codd,"A relational model of data for shared data banks", CACM, Vol 13, №6, 1970

〔12〕 G.G.Dodd,"APL—A Language for Associative Data Handling in PL／I", FJCC, 1966

〔13〕 C.A.Lang and J.C.Gray,"ASP—A Ring Implemented Associative Structure Package", CACM, Vol. 11, №8, 1968

〔14〕 藤野精一「リスト処理と言語解析」朝倉書店, 1970

〔15〕 K.C.Knowlton,"A Programmer's Description of L^6", CACM, Vol 9, №8, 1966

〔16〕 J.Weizenbaum,"Symmetric List Processor", CACM, Vol 6, №9, 1963

〔17〕 C.Hewitt,"PLANNER, AI—Memo №168", Project MAC, M.I.T. 1968

〔18〕 J.Feldman and P.D.Rovner,"An ALGOL—Based Associative Language", CACM, Vol.12, №8, 1969

単元Ｃ２

コンピュータ・システム

単元 C 2
コンピュータ・システム

目　次

第1章　コンピュータ・システムの概要

キーワード

コンピュータ・システムの構成要素（units of computer system），アーキテクチャ（architecture），アーキテクチャのレベル（level of architecture），オペレーティング・システム（oprating system），応用プログラム（application），ハードウェア・システム（hardware system），ソフトウェア・システム（software system）

目　標

コンピュータ応用領域の拡大，利用形態の進歩によって，コンピュータ設置台数はますます増加すると同時に極めて多様な使われ方をするようになった。コンピュータの種類も超小形のオフィス・コンピュータから超大型コンピュータまでの規模の広がりと，汎用コンピュータの他にも専用の目的に使用されるコンピュータも増加しつつある。またオンライン／ＴＳＳの利用による端末の発達も著しいものがある。一方コンピュータをユーザの目的に適合させるためのソフトウェアの量は急速に増大しており，システムの効率的開発のための技術の確立が重要になっている。

コンピュータ・システムの単元ではコンピュータ内部の問題を詳細に習得させることよりも，コンピュータを利用したシステムの構成と開発に必要となる基本的問題についてなるべく巾広く理解させることを目標とする。

本章では，その目標に即した概要を説明する。

内　容

1.1 基 本 概 念

コンピュータ利用システムを作成する場合，もっとも重要なことは，そのシステムのユーザのニーズを明確にし，ユーザの経済的制約，その時点における技術的制約のもとで，できるだけ効果的にユーザのニーズに適合するシステムを設け開発することである。この場合，システムの設計者がシステムを作るために使用する構成要素がもっている設計者が利用できる機能の集合をその構成要素の**アーキテクチャ**とよぶ。たとえばあるコンピュータのマシン構造アーキテクチャとは，そのコンピュータの機械語命令の集合を表わしている。もしコンピュータをオペレーティング・システムを含めた形で考えれば，オペレーティング・システムの機能レベルでのアーキテク

チャを考えることもできる。より一般的に考えれば設計者に限らず，アプリケーションのユーザからみれば一つの応用システムは，その提供するアプリケーションの機能のレベルにおけるアーキテクチャをもつと考えることができる。アーキテクチャのレベルは図1—1に示されるが，この単元が対象とするレベルはその中のシステム・アーキテクチャのレベルと考えることができる。したがってまず始めに，システムのアーキテクチャとそのレベルについて理解させることはシステムの開発を効率的に行う上からも非常に大切な事である。

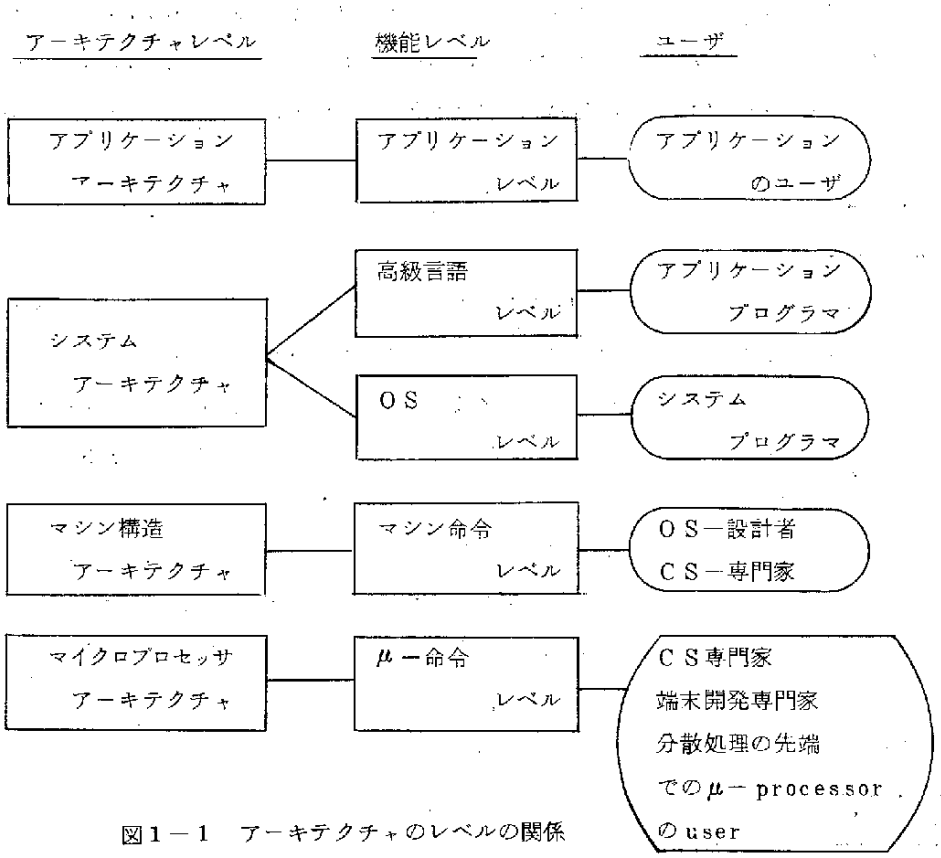


図1—1　アーキテクチャのレベルの関係

(1) システム・レベルのアーキテクチャ

あるオンライン応用システムを設計開発する場合，大きく見ればハードウェア要素とソフトウェア要素から構成されている。少し詳しく検討すれば図1—2にみられるようにシステムコンポーネントとして，コンピュータ・ハードウェア（CPU，主記憶など），端末装置，オペレーティング・システム，言語，データベース／データ・コミュケーション，ネットワーク／通信制御サブシステム，応用プログラムなどが考えられる。これらのコンポーネントのもっている多くの機能を利用して，目的とするユーザ・システムを構成するためにシステム構成技術とよ

ばれる技術の集合が必要であり，これらには応用モデル，システム分析評価技術，システム設計技術，システム開発／検査技術，システム運用保守技術，さらにシステム開発管理技術などが考えられる。これらの各種技術を使って，システム設計者は最適な応用システムを構成して行くことができれば，大変理想的である。しかし，現時点ではこのようなシステム・レベルにおけるアーキテクチャは十分発展しているとはいえないが，本単元ではこの様な考え方に立って講義をすすめることが望ましい。一般に，コンピュータ・システムのハードウェア，ソフトウェアの全体を理解することはかなり大変であり，上記の様な方向性をもって体系的に扱うことによって，理解度をますことができると考えられる。

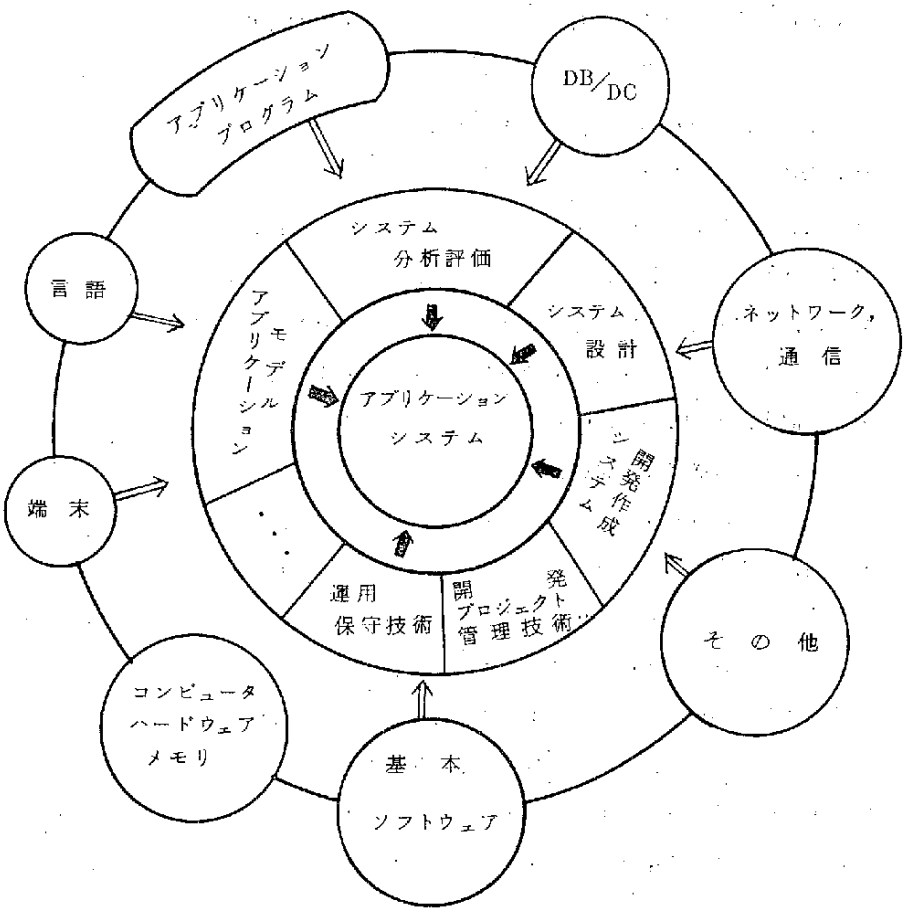


図１－２　システム・レベルのアーキテクチャ

(2) システム設計の考え方

(a) 要求と実現手段との整合性

システム設計を進める際に考慮すべき条件として使用目的からの要求と，それをシステムとして実現するために利用可能な各サブシステム（ソフトウェア，ハードウェア）の機能を十分理解して，すなわち要求と実現手段との整合をシステム分析やシステム設計開発支援技術並びにシステム評価技術を採用することによって行うことが重要である。この整合作業によって要求されるシステムの方式すなわちアーキテクチャを決定する方法論を各自で把握する様に指導することが大切である。

(b) 使い易く、理解しやすいシステムにする

これは条件というよりもむしろ当然考慮すべき基本的性質でもあるが，システムの効果を発揮させるためユーザにとって最も重要なものの一つである。このためマン・マシン・インタフェースには特に考慮が必要であり使いやすい端末の選択が要請される。またシステム自体の構造も明確な分りやすい構造に作っておくことが，システムを使う場合，さらに保守をする場合にも結局よい結果をもたらすことになる。特にＤＢ／ＤＣシステムの場合などでは，あまり複雑なデータ構造は好ましくなく，実際の情報の相互関係，利用度，情報の発生点，到着点，利用者との関係を考慮したネットワークの分析が大切である。結局作成される情報システムといっても，規定の組織の中で利用されるものであり，システムと組織との調和のとれた融合が大切な事を理解させることも重要である。

(c) システムの柔軟性、発展性

システムは開発終了後，保守運用体制に入ったあとも，利用環境条件その他の要請から改良発展がかなりの工数で行われる。レポートによっては長期に利用されるシステムでは保守改良作業が全体の６０～７０％に達する場合もあるといわれるので，改良保守作業がやりやすいようなシステム構造にしておくこと，すなわち柔軟性のある構造，たとえば十分な機能分析による階層構造，モジュール構造等によるシステム構成が望ましい場合が多い。

このためには，わかりやすい各種ドキュメント類の整備，設計，作成用の高水準言語の使用などが効果的であり，またシステムの要求分析段階から開発保守運用段階全体を通した，システムのライフサイクル的にみた一貫した考え方が必要になる。

1.2 ハードウェア体系

コンピュータ・システムを構成する要素をハードウェアとソフトウェアに大別したときハードウェア体系は次の様になる。

1. プロセッサ

2. 記憶装置

3. 入出力装置

4. 通信制御装置

5. 端末装置

の五つに分類され，さらに細かく分けられる。これらのサブシステムの組合せによって，すなわち各サブシステムの機能や，それらを構成する装置の種類によって多くの特徴や，性能の差があり，また，構成方式の差による相異もかなりある。さらにコスト的にもかなりの範囲があるので，これらとシステムの要求として組込んで，要求される機能，能力を発揮させるためには，各ハードウェア要素を動かして行くためのソフトウェアとの組合わせによって全体としての性能を発揮できる様にして行くことが重要問題である。このためには以下の各章で説明される様に性能を規定するパラメタの理解や，システムとしての性能評価方式の理解と利用が大切な問題であることを説明する。

1.3 ソフトウェア体系

開発作成されるユーザ・システムを考えるとき，ハードウェアよりもソフトウェア・システムはよりユーザに近い位置にあるものであり，同じ機能性能のハードウェアを使っても，ソフトウェアの方式の工夫によってかなり大きな性能の差異を生じる場合がある。また応用ソフトウェアの性能を上げるためにもオペレーティング・システムを中心とする基本ソフトウェアの機能性能をよく理解することが大切であり，さらにシステムの信頼性や運用方式などの理解も重要である。

この節では，このような観点からソフトウェア体系的に眺め利用できる様に理解させることを主目標にする。

一般にソフトウェア・システムは図1－3に示す様な体系をもっている。詳細については4章以下で取扱う。

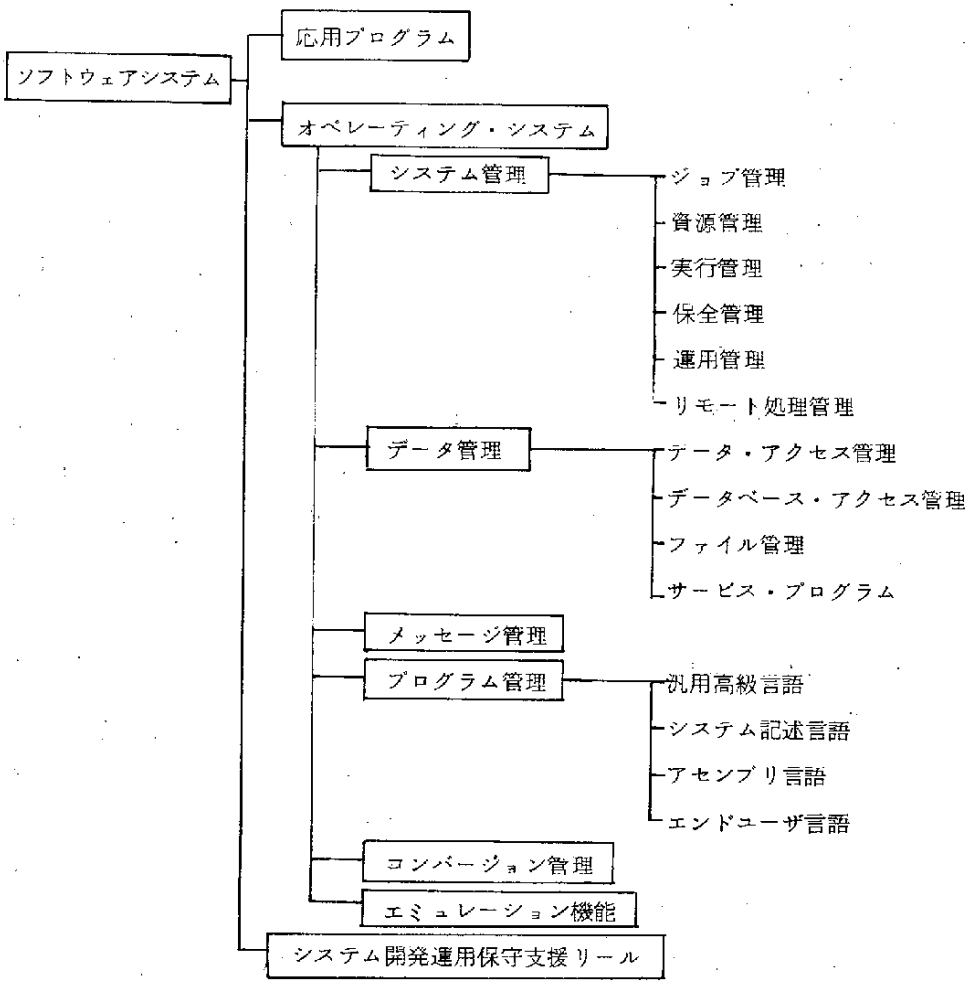


図1－3　ソフトウェア・システムの体系

指導上の留意点

(1) 本章は第3部単元2の最初の章であり，現時点におけるコンピュータ・システムの発展状況と，将来方向についての大よその展望を与えることが望ましい。

(2) 目標にものべた様に，コンピュータ・システム全体を詳細に講義するのではなく，どの様にしてユーザの要求に応じたシステムを構成し開発するかに指導の重点をおく。したがって本章では，システムの機能レベルとアーキテクチャ・レベルについて，体系的に把握できる様に指導する。

(3) ハードウェアおよびソフトウェアの体系を理解させると同時に，それらを構成する各種のコンポネントを，これらを組合わせてシステムを構成するためのシステム開発構成技術についての

重要性と使用法について大要を修得させる。

(4) 新しい方向としてコンピュータとコミュニケーションの融合によるシステムのネットワーク化，分散化に関連する各種技術および，通信，端末等のコンポネントについて十分理解させることも必要である。

(5) システムの性能についての考え方も，ハードウェアのコスト効率よりも，システムが組織に融合して利用される方向にあり，人間側からみた使いやすさなどを十分考慮した効率の追おという面からのシステム評価の考え方についてもふれておくことが望ましい。

(6) 今後システムにしめるソフトウェアの量が益々増大する傾向にあり，このため作りやすくかつ保守しやすいソフトウェア構造の重要なことを十分理解させることが大切である。

参 考 文 献

〔1〕石井善昭「計算機の基本方式」共立出版，1973

第2章　中央処理装置および記憶装置

キーワード

アーキテクチャ（architecture），演算論理装置（ALU；arithmetic logic unit），マイクロプログラム（microprogram），配線論理（wired-logic），エミュレーション（emulation），制御記憶（control storage），並列処理（parallel processing），前置プロセッサ（front-end processor），後置プロセッサ（back-end processor），データベース・マシン（database machine），高水準言語マシン（high-level language machine），信号処理（signal processing），アクセス時間（access time），サイクル時間（cycle time），記憶階層（storage hierarchy），高速記憶装置（high-speed storage），主記憶装置（main storage），補助記憶装置（anxiliary storage，secondary storage），外部記憶装置（external storage），内部記憶装置（internal storage），固定記憶装置（read-only storage），連想記憶装置（associative storage），半導体記憶素子（semiconductor storage element），磁気記憶素子（magnetic storage element），インタリーブ（interleaving），高速緩衝記憶装置（high-speed buffer storage），キャッシュ（cache），記憶保護（storage protection），誤り訂正符号（error-correcting code）

目　標

コンピュータ・システムの処理の中心であるプロセッサ（CPU及びその他の特殊プロセッサを含む）の機能，構造について理解させる。ここではプロセッサの基礎知識は充分に有するものとし，主として，従来の知識を体系的に整理することを目標とする。また，最近，システムの機能分散傾向によって，特殊目的のプロセッサが汎用プロセッサと共に使用される傾向にある。それらの特殊目的プロセッサについて理解を深め，システムを設計するときに役立たせることも狙いの一つである。

さらに，また，高性能でかつ大容量の記憶装置を実現するための技術，特に，バッファ記憶方式について理解を深め，類似の階層記憶システムとして，ディスクと低速大容量記憶装置とを組み合わせた仮想ディスクシステム等を理解させる。

記憶装置においては信頼性が極めて重要である。各種の信頼性向上のための方式と記憶保護の基本について理解させる。

内　容

2.1　プロセッサの基本機能

コンピュータ・システムにおけるプロセッサの役割を復習する。受講者は既にかなりの基礎知識を有するものと考えられるが，ここではそれを体系的に整理する。

(1) 機　　能

プロセッサの基本機能はそれに与えられる命令列（プログラム）を実行することである。実行するプログラムの性質によって，次の２種に分けられることを説明する。

① データ処理

記憶装置上に与えられたデータの演算処理，判断を行う。

② システム制御

コンピュータ・システムを構成する各種資源を制御する。システム・プログラムが通常これを行う。次のような機能をあげて説明する。

・入出力の起動

・割込制御（外部割込，タイマ，プログラム割込，入出力割込，障害割込）

・障害検出と障害修復

(2) プロセッサの特性

プロセッサを特徴づける基本項目を理解させる。次のような項目がある。

・アーキテクチャ（データ形式，命令セットなど）

・構成要素（ハードウェア）

・構成方式（レジスタ数，ビット巾など）

・制御方式（マイクロプログラム制御，先取り制御など）

・結合方式（記憶装置，入出力装置との結合方式など）

これらの項目により，命令実行速度のような性能と価格が定まる。

2.2　アーキテクチャ

プロセッサのアーキテクチャは使用者との機能的接点を定める重要なものである。ある応用から見るとき，アーキテクチャがその応用が要求する機能をどれだけ満足しているかによって性能に大きく影響することを理解させる。

(1) アーキテクチャの定義

さまざまな定義があるが，参考文献〔１〕の定義が一般的である。すなわち，アーキテクチャとは「プログラマから見たシステムの属性」であり「データの流れや制御，論理設計，物理

的な実現方法とは別の概念的構造及び機能的動作」と定義している。定義とともに，アーキテクチャの意義についても説明する。

(2) アーキテクチャの基本要素

アーキテクチャを規定する基本要素について説明する。具体的な例を参照しながら説明することが望ましい。

① データ形式

・2進数，10進数

・固定小数点，浮動小数点

・配列，リストなどのデータ構造の表現

・文字列，ビット列

など種々のデータ形式の内部表現があることを説明する。PL／Iなどのデータ宣言文と対比させて，高水準言語とプロセッサ・アーキテクチャとのギャップを理解させるのも一つの方法である。

② アドレス指定方式

種々のアドレス指定方式を説明する。

・スタック方式

・インデックス修飾

・間接アドレス方式

・相対アドレス方式

これらのアドレス指定方式によって，必要なデータへアクセスする効率が異なることを説明する。また，アドレス可能な範囲がプログラムの作成効率に影響すること，一方では命令語長の制約との関係があることも説明する。

③ 命令形式

各種命令形式を説明する。

・0，1，2，3アドレス方式

・命令語の構成

④ 命令レパートリ

基本的な命令レパートリは既に理解しているものと考えられる。したがって，応用目的により命令の使用比率が著しく異なること，応用指向の命令レパートリを持ったアーキテクチャ（たとえば高級言語マシンなど）が考えられていることなどを説明する。

⑤ システム制御機能

システム制御のための機能について説明する。

・入出力の起動

・割込処理

・タイマ

・その他（イニシアル・ローダ，障害検出，記憶保護など）

2.3 プロセッサの基本構成

プロセッサの基本構成要素である演算論理装置（ＡＬＵ），レジスタおよびフリップフロップ（register and flip-flop），制御回路について簡単に説明する。これらの構成要素の機能と動作について復習する。

2.4 マイクロプログラム制御

最近の多くのプロセッサでは配線論理（wired-logic）に代って，マイクロプログラム制御方式が使用されている。マイクロプログラム制御方式の特徴について理解させる。

(1) 特　　徴

次のような特徴を説明する。

① 不規則で複雑な制御論理の大部分が規則的なメモリ構造に置き換えられる。

② ハードウェアの細部をプログラム的に制御しうる。

③ 複雑な機能を持ったアーキテクチャを比較的低価格で実現できる。

④ 制御論理の変更・修正が容易である。

マイクロプログラム制御方式と配線論理方式のコスト比較を図２－１により説明する。

(2) マイクロプログラムの応用

マイクロプログラム方式を活用したいくつかの方式を説明する。

① エミュレーション

② 高水準言語処理のサポート

③ 制御プログラムの高速化

④ 診断と保守

⑤ その他（モニタリング等）

(3) 制御記憶

マイクロプログラムを格納する制御記憶装置は高速なものが要求される。固定記憶装置（ＲＯＭ）が使用されることが多かったが，最近では書き込み可能な制御記憶（ＷＣＳ）が用いられることが多い。制御記憶の機能とＷＣＳの利点について説明する。

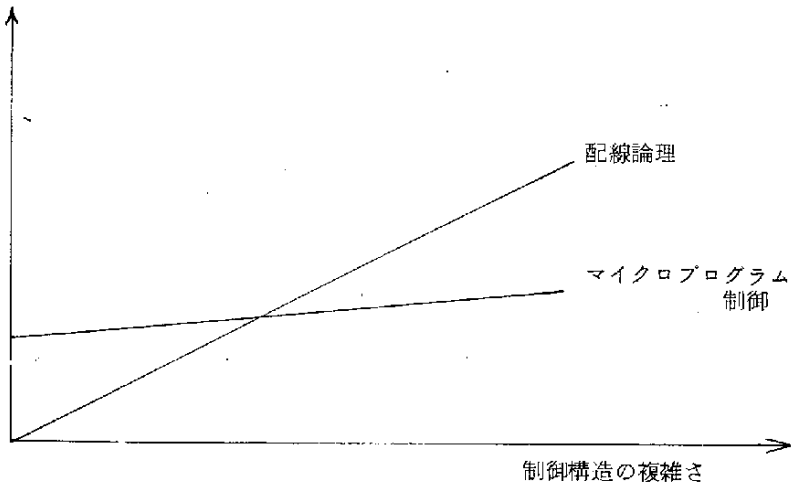


図２－１　　制御方式のコスト（概念図）

2.5　特殊目的プロセッサ

特定の応用目的を指向したプロセッサによる性能向上を狙いとする特殊目的プロセッサを汎用プロセッサと結合して使用することが多くなっている。種々の応用に対する特殊目的プロセッサの構造を理解させる。

(1)　並列処理プロセッサ

大規模な行列の処理や偏微分方程式の数値解析などでは極めて高性能な演算速度が要求される。多数のプロセッサを並列に動作させることによって，高性能を実現する並列処理プロセッサを説明する。並列処理の三つのタイプについて例をあげて説明する。

①　ＳＩＭＤ型（ＩＬＬＩＡＣ　Ⅳ，ＰＥＰＥ等）

②　ＭＩＳＤ〃（パイプライン制御，ＳＴＡＲ－１００　等）

③　ＭＩＭＤ〃（Ｃ．ｍｍｐ　）

(2)　前置プロセッサ

通信回線を含んだシステムにおいて，通信制御を専用に処理する前置プロセッサが採用されることがある。プロセッサとしてはミニコンピュータが使用されることが多い。代表的なものにＡＲＰＡ網のために開発されたＰＬＵＲＩＢＵＳシステムがある。

(3)　データベース・マシン

データベース管理を専門に処理するデータベース・マシンは前置プロセッサに対応して，後置プロセッサと呼ばれることがある。実験システムとしてベル研究所のＸＤＭＳシステムがある。

(4) 高水準言語マシン

高水準言語プログラムを直接的に処理する高水準言語マシンはマイクロプログラムを応用したものが多い。高水準言語マシンの概略を説明する。

(5) 信号処理プロセッサ

高速フーリエ変換を行うＦＦＴプロセッサなどが信号処理の分野で使われている。ＦＦＴプロセッサの基本概念を理解させる。

2.6 記憶装置の特性と種類

(1) 記憶装置の特性と種類

記憶装置はデータの格納（蓄積），保存，参照（読出し）の機能を持つが，記憶装置は次のような諸元により特性づけられることを説明する。

① 参照時間（アクセス時間）
② 書込時間
③ サイクル時間
④ 読取りビット数
⑤ 書込みビット数
⑥ 容量
⑦ コスト（ビット当りコスト）
⑧ データ転送速度
⑨ 揮発性／不揮発性
⑩ アクセス特性
　・ランダム
　・シークエンシャル

(2) 記憶装置の階層的分類

記憶装置はその特性によっていくつかの種類に分類され，図２－２のような階層をなすこの記憶階層の機能的役割を説明する。

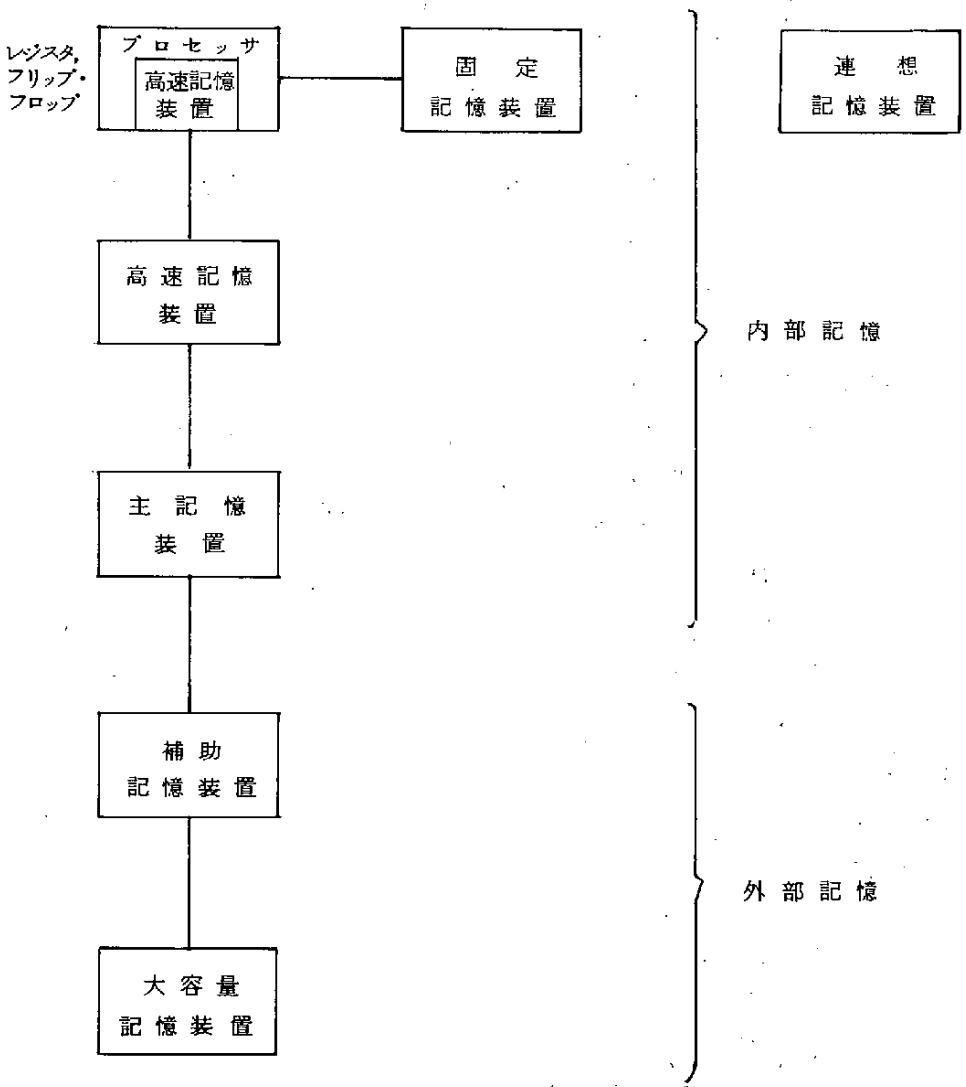


図2－2　記憶階層

① 高速記憶装置

プロセッサが演算のために使用する。レジスタ，フリップ・フロップ，制御記憶などを含むが高速バッファ（緩衝）記憶を指すことが多い。容量よりもアクセス時間のような性能が重視される。

② 主記憶装置

プロセッサが実行するプログラムとデータを格納する。プロセッサの規模に応じた速度，容量，ビット幅が要求される。システムの性能に大きい影響を及ぼす重要なものである。

③　補助記憶装置

主記憶装置を補うために使用される。中程度のアクセス時間，主記憶よりも１桁～２桁大きい容量とコストの安いことが必要である。

④　大容量記憶装置

ファイルの貯蔵に用いられる。容量が第一に要求され，次いでアクセス時間，データ転送速度が問題になる。補助記憶を大容量記憶の両方を外部記憶装置と呼び，高速記憶と主記憶とを内部記憶装置と呼ぶ。

⑤　読取専用記憶装置（固定記憶装置）

マイクロプログラムのように固定的に必要な情報を格納するために用いられ，書き込みは動作時にはできず，製造時あるいは使用時に特別の装置を用いて行われる。

⑥　連想記憶装置

通常の記憶装置が格納位置でアクセスされるのに対し，格納された内容によってアクセスされる特殊な記憶装置である。検索などの応用に極めて有効であるが，安価な素子がまだ見つかっていない。

(3)　性能と価格

各種の記憶装置の性能と価格の関係を図２－３のような図を用いて説明し，記憶階層構成が性能と価格のバランスをとるために必要であることを理解させる。

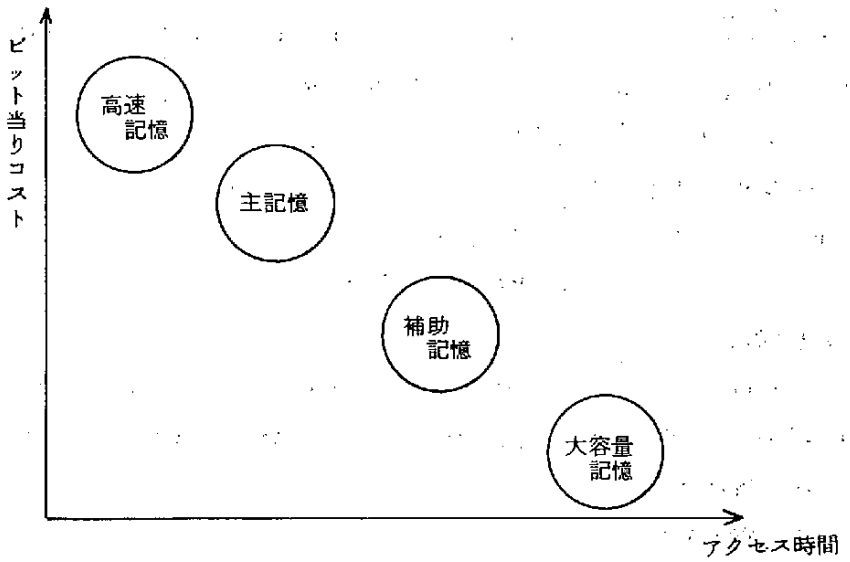


図２－３　　記憶装置の性能とコストの関係

2.7 記憶素子

前節の分類は機能面から見た分類であるが，記憶装置に用いられる素子について説明し，素子の特徴とその記憶階層における位置づけを理解させる。

(1) 半導体記憶素子

① 種　類

a) バイポーラ型

- ＴＴＬ
- ＥＣＬ
- I^2 Ｌ
- その他（ＲＴＬ，ＤＴＬ等）

b) ＭＯＳ型

- Ｐ－ＭＯＳ
- Ｎ－ＭＯＳ
- ＣＭＯＳ

c) 特殊記憶

- ＲＯＭ
- ＣＣＤ

② 特徴と記憶階層における位置

半導体記憶素子の特徴は高速性にある。その中でもバイポーラ型のものが最も高性能であるが，最も高価でもある。一方，ＭＯＳ型は性能面ではバイポーラよりも劣るが，価格はバイポーラよりも安価である。ＣＣＤ記憶はシフト・レジスタ型の構造であり，性能よりも安価な点に特徴がある。一般に，次のような記憶階層で使用される。

- バイポーラ型　高速記憶装置
- ＭＯＳ型　主記憶装置
- ＲＯＭ　マイクロプログラム用記憶装置
- ＣＣＤ　（高速）補助記憶装置

(2) 磁気記憶素子

① 種　類

a) ランダム・アクセス記憶装置

- 磁心（磁気コア）
- 磁性模
- 磁性線（ワイヤ）

b) シフト・レジスタ型

・磁気バブル

c) 回転型記憶装置

・磁気ディスク装置（可動ヘッド，固定ヘッド，フロッピィ・ディスク）

・磁気ドラム装置

d) その他

・磁気テープ装置

・磁気カード装置

・磁気テープ・カートリッジ

・カセットテープ

② 特徴と記憶階層における位置づけ

磁気記憶装置の特徴には情報の不揮発性がある。そのため，磁気記憶は外部記憶として大量のデータを貯蔵する目的に用いられることが多い。記憶階層との関係は次のような事項を説明する。

・ランダム・アクセス記憶装置 ── 主記憶，
　　└ 高速補助記憶

・シフト・レジスタ型 ── 高速補助記憶

・回転型 ── 外部記憶装置

・その他 ── 大容量記憶装置

(3) その他の記憶素子

・電子ビーム記憶（ＥＢＡＭ）

・光記憶（ホログラム・記憶）

2.8 主記憶の高速化技術

論理素子の速度に比較して主記憶装置の速度は遅い。そのため，主記憶装置を高速化するいくつかの技術が考えられている。それらの技術について説明する。

(1) 記憶装置構成

・インタリーブ

・モジュール化

・書込み／読取ビット数の増加

などの記憶装置構成による高速化について理解させる。

(2) 高速緩衝記憶装置

プロセッサ側に高速緩衝記憶装置を置き，主記憶から読み出されたデータを高速緩衝記憶装置に格納し，主記憶への参照回数を減少させるバッファ記憶方式あるいはキャッシュ方式を理解させる。

① 実効アクセス時間

高速緩衝記憶，主記憶のアクセス時間をそれぞれ t_B ，t_M とし，参照されるデータが高速緩衝記憶に存在する確率（ヒット率）を p とすると，記憶システムとしての実効アクセス時間 t_E は次式で表わされる。

$$t_E = p \cdot t_B + (1-p)\, t_M$$

p が1に近ければ実効アクセス時間は t_B に近づき，t_B が充分速ければ高速な主記憶と等価であることを理解させる。

② マッピング方式

高速緩衝記憶を用いる方式では，プログラムがその存在を意識しないですむようにすべてハードウェアで自動的に制御する方式がとられる。代表的な制御方式について説明し，制御方式によってバッファ容量とヒット率の関係，および制御ハードウェア量が異ることを理解させる。

・アソシアティブ方式

・セクタ方式

・セット・アソシアティブ方式

③ 技術的問題点

高速緩衝記憶方式ではマッピングの他に次のような問題があることを理解させ，その解決法を理解させる。

・主記憶・高速緩衝記憶間データ転送の高速化（インタリーブなど）

・書き込みデータの処理

・入出力動作の取扱い

・多重プロセッサ・システムにおける共用データの取扱い

2.9 外部記憶の高速化技術

主記憶装置だけでなく，外部記憶装置のアクセス時間を短縮することもシステム性能を向上させる上で重要である。高速化技術の代表的なものについて説明する。

(1) 階層記憶方式

高速緩衝記憶が主記憶とプロセッサの間に設けられた記憶階層であるのと同様に，外部記憶

においても主記憶と大容量記憶の間に補助記憶の階層を設け，高速化をはかる方式がある。次のような方式について説明する。

① 仮想ディスク

低速，大容量の外部記憶，たとえばカートリッジ・テープ装置と高速の磁気ディスクを用いて階層記憶構成をとる仮想ディスクシステムについて説明する。ＩＢＭ社の３８５０ 大容量記憶システム（ＭＳＳ）がその典型的な例である。

② ディスク・キャッシュ

まだあまり一般的ではないが，大容量コアやＣＣＤ，バブルのような高速補助記憶をディスクのようにアクセス時間の遅い記憶装置の緩衝記憶として用いる方式がある。

(2) 回転待時間の短縮

ＴＳＳのようにスワッピングが頻繁に行われるシステムにおいて，ドラムやディスクの回転待時間を最小にすることが重要である。

そのための方式として次のようなものを説明する。

・ページング・ドラム方式

・回転角検出機構

(3) 知能ディスク

外部記憶装置にデータ処理機能を持たせ，アクセスに関連した処理，たとえばキー値によるデータ検索を実行させることによって，プロセッサと外部記憶間のデータ転送量を減らす方式がある。通常，ディスク装置などの制御装置のデータ処理能力を強化することによって実現される。データベース処理を高速化する目的に使用されることが多く，知能ディスクと呼ばれる。知能ディスクの概念とシステム構成を説明し，その効果を理解させる。

2.10 記憶装置の高信頼化と記憶保護

記憶装置では格納されたデータを誤りなく再生することが重要である。そのために，次のような手法が用いられることを説明する。

① 誤り検出

・パリティ・チェック（水平，垂直）

・多重化による比較チェック

② 誤り訂正

・誤り訂正符号による自動訂正

・多数決論理による修正

また，書込みによってデータ内容が破壊されることを防ぐ記憶保護機構について，次のよう

な保護方式を説明する。

・キーによる保護

・アクセス権による保護（リング・プロテクション，境界レジスタ等）

・物理的手段による書込禁止（磁気テープのプロテクション・リング等）

指導上の留意点

(1) プロセッサの機能的側面であるアーキテクチャと構造的側面であるハードウェア構成とが混同されがちである。両者の区別と関係を明確に理解させるよう留意する。

(2) アーキテクチャの説明は一般的な説明では単調になりやすい。受講者になじみ深いシステムを例にとって説明し，それに含まれていない機能等を補足するとよい。

(3) マイクロプログラムに関連した話題は多いが，ここではその基本概念とその応用に対する考え方を理解させる程度に止めざるを得ないだろう。

(4) 特殊目的プロセッサについては主なプロセッサの特徴とその応用における位置づけを理解させることが重要であり，その内部構造にまで立入った説明は時間的に無理である。特定のプロセッサの構造的特徴まで理解できれば充分である。

(5) 記憶装置を機能的にとらえた分類と，物理的な素子としてとらえた分類とがあることに注意する必要がある。主記憶，外部記憶というのは機能的な見方であり，たとえばコアがその両方に使用されることもある。

(6) 記憶素子の詳細についての説明は不要であろう。むしろ，素子の種類とその特徴を理解させる方に重点を置く。

(7) キャッシュについては最近のほとんどの大形システムで採用されているため，その特徴を良く理解させることが必要である。特に，キャッシュ方式の問題点については詳しい説明をすることが望しい。また，この方式はプログラマから見えない点が仮想記憶システムと異ることも理解させる必要がある。

(8) 信頼性と保護については大体の概念を理解させる程度で良い。

参 考 文 献

〔1〕 Amdahl, G. M. et al, "Architecture of the IBM System 360", IBM Journal of Res. & Dev., vol. 8, pp. 87 1964

〔2〕 石井善昭「計算機の基本方式」共立出版，1973

〔3〕 藤野，箱崎，山本「マイクロプログラミング技術とその応用」電子通信学会誌，vol.

60, pp. 635－644, 1977

〔4〕 「情報処理（専用プロセッサの方式とシステム構成特集号）」vol. 18, no. 4, 1977

〔5〕 「電子通信学会誌（メモリ特集）」vol. 60, no. 11, 1977

〔6〕 「情報処理（メイン・メモリ特集号）」vol. 16, no. 4, 1975

〔7〕 日経エレクトロニクス編「半導体メモリ」日経マグロウヒル, 1976

第3章　入出力および通信制御

キワード

入出力チャネル（input-output channel），入出力制御装置（input-output control unit），入出力装置（input-output device），入出力インタフェース（input-output interface），選択チャネル（selector channel），多重チャネル（multiplexer channel），ブロック多重チャネル（block-multiplexer channel），チャネル・プログラム（channel program），入出力コマンド（input-output command），データ・チェイニング（data chaining），コマンド・チェイニング（command chaining），周辺装置（peripheral equipmeut），端末装置（terminal equipment）ネットワーク・システム（network system），通信制御装置（communications controller），遠隔集線装置（remote concentrator），メッセージ交換装置（message-exchanger），プロトコル（protocol），ネットワーク・アーキテクチャ（network architecture），知能端末（intelligent terminal）

目　標

最近のコンピュータ・システムにおいて，演算処理能力は当然ながら，入出力処理能力も重要な要素である。入出力アーキテクチャの主としてハードウェア的側面を正しく理解し，入出力チャネル，入出力制御装置，入出力装置などの機能と動作を把握させることを目標とする。
ここでは入出力の最も中心的な部分を充分理解させることが必要である。また，最近の機能分散化傾向により，入出力側により多くの機能を分担させる方向がある。この流れを理解させることも重要である。

さらに，コンピュータ・システムを通信回線と結合したネットワーク・システムは遠隔地の端末からコンピュータを使用するシステムやコンピュータ同志を結合したコンピュータ・ネットワーク・システムとして重要である。そこでネットワーク・システムを構成する主要な要素である通信制御装置と端末装置の機能と構成を理解させる。

内　容

3.1　入出力アーキテクチャ

入出力サブシステムの基本構成を複習する。一般に図３－１に示すように，

① 入出力チャネル（チャネル装置，データチャネル，入出力マルチプレクサ等いろいろな名称が使用される）

② 入出力制御装置（Ｉ／Ｏコントロールユニット，周辺制御装置などとも呼ばれる）

③ 入出力装置（Ｉ／Ｏデバイス，周辺装置）

で構成されるものとして説明するとよい。

この構成にもとづいて，入出力アーキテクチャを次のように説明する。

(1) 入出力アーキテクチャの狙い

入出力サブシステムは入出力装置の動作を制御し，入出力装置と主記憶装置との間のデータ転送を実行する。入出力装置のデータ転送速度は主記憶装置のサイクル時間に比べて遅いため，次のようなことが要求される。

① 入出力サブシステムと主記憶装置の間のデータ転送を中央処理装置と独立に実行する。

② 複数の入出力動作が同時に実行できるようにする。

これらのことを実例をあげて説明し，入出力サブシステムの目的を理解させる。

(2) インタフェース

入出力サブシステムのインタフェースには次の四つのレベルがあることを説明する。

① プロセッサー入出力チャネル間

中央処理装置などのプロセッサと入出力チャネルとのインタフェースである。

② 主記憶装置ー入出力チャネル間

主としてデータ転送に使用されるインタフェースであるが，入出力チャネルの動作に必要な情報の授受のためにも使用される。

③ 入出力チャネルー入出力制御装置間

通常入出力インタフェースといった場合，このインタフェースをいう，最も標準化が進んでいる。

④ 入出力制御装置ー入出力装置間

入出力装置個有の性質に依存するインタフェースであり，個々の入出力装置ごとに規定される。

以上の各レベルのインタフェースについて例をひいて具体的に説明する。標準化の必要性，その実現における問題点などについてもふれることが望しい。

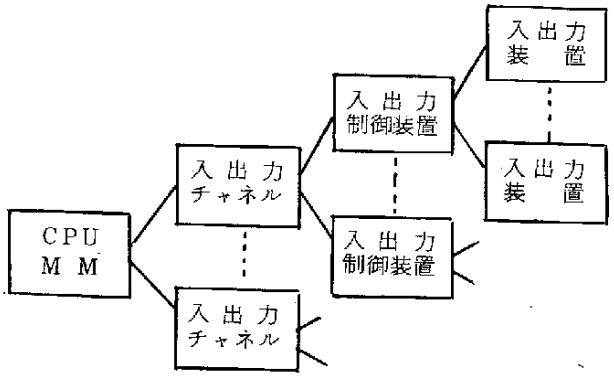


図3－1　　入出力サブシステムの構成

3.2　入出力チャネル

(1)　入出力チャネルの機能

入出力チャネルの機能はプロセッサによって起動され，入出力制御装置の制御を行う。次のような事項について説明する。

・入出力指令の解読

・入出力制御装置又は入出力装置に対する動作の指定

・データ転送の制御（入出力制御装置側及び主記憶装置側）

・入出力装置の状態の把握

・障害の検出と処理

上にあげた機能は中型以上の汎用コンピュータでは標準的な機能であるが，小型のコンピュータやミニコンピュータではこれらの機能の全てが入出力チャネルによって実行されないことが多い。そのような場合にはこれらの機能の一部あるいは全部がプロセッサによって実行されることを説明する。

(2)　入出力チャネルの分類

入出力チャネルに接続される装置には多様なものがある。磁気ディスクのように高速なデータ転送速度を持つものやカード読取装置やシリアル・プリンタのように低速なものもある。そのために，接続する装置に応じてチャネルのデータ転送機能が異るチャネル装置が使われる。代表的なものとして，次の２種を説明する。

・選択チャネル

・多重チャネル

補足説明として，ＩＢＭ３７０ のブロック多重チャネルについても説明しておく方が良い。

また，入出力チャネルと入出力制御装置との結合方法から分類すると，次の２種がある。

・スター型（並列接続，たこ足式）

・バス型（直列接続，いもづる式）

これらの両方式の利点・欠点についても説明する。

(3) 入出力チャネルの動作

入出力命令の実行に伴なう入出力チャネルの動作を説明する。

a）チャネル・プログラム

複雑な入出力動作を指定するためには多くの情報が必要である。そのために，入出力命令は単に入出力動作の起動だけを規定し，それ以降の動作は入出力コマンド（入出力指令）によって指定する方式が一般的である。

コマンドは単独でも用いることができるが，複数のコマンドを組合わせたプログラムとしても実行できるチャネル・プログラム方式を採用するものが多い。チャネル・プログラムに関連する次のような事項について説明する。

・入出力コマンドの形式

・入出力コマンドの機能

・チャネル・プログラム

・データ・チェイニング

・コマンド・チェイニング

b）チャネル・プログラムの実行

チャネル・プログラムを解読し，指定された入出力制御装置にコマンドを送ることにより，データ転送を制御するのがチャネルの動作である。これらの一連の動作をたとえば図４－２のような例について説明する。

c）入出力インタフェース

入出力チャネルが入出力制御装置との間で制御信号，データの受け渡しをするための標準的な手順が入出力インタフェースである。

入出力インタフェースは多様な入出力制御装置の特性によらない標準化が進められているが，実際にはメーカー毎に標準が定められている段階であり，国内，国際的な標準化の努力が続けられている。

入出力インタフェースの説明はＪＩＳインターフェース案，電子工業振興協会の論理仕様に基づいて次のような説明をするとよい。

・信号線の説明

・コマンド

・状態情報

・動作モードとシークエンス

・異常状態の処置

(4) チャネル装置のハードウェア

チャネル装置は一種のプロセッサである。その実現形態には次のようなものがある。

・中央処理装置がチャネル機能を実行する。

・中央処理装置のハードウェアを使用し，部分的に特別なハードウェアを付加して実現する。

・独立の装置として構成する。

・入出力プロセッサとして，チャネル機能だけでなく，入出力管理機能も行う。
（たとえば，CDC 6000シリーズの周辺プロセッサ，BCC500 システム）

3.3 入出力制御装置

入出力チャネルとの情報を交換し，入出力装置を直接制御するのが入出力制御装置である。チャネル装置が入出力装置の種類に必しも依存しないのに対し，入出力制御装置は入出力装置に依存した機能が要求される。

入出力制御装置の機能を一般的に説明する。次のようなことを理解させる。

① チャネルから受取ったコマンドを解読し，指定された入出力装置を動作させる。

② 入出力装置のデータ転送を制御する。

このとき，必要に応じて次のような処理を行う。

・符号変換

・直・並列変換（ビット←→バイト等）

・バッファ制御

・エラーの検出（パリティ検査，冗長コードによるエラー修正，検出）

③ 入出力装置の状態の監視を行い，状態情報をチャネル装置に送出する。

・動作の完了

・異常状態の検出

入出力制御装置の構造は入出力装置の性質により多様である。最近の傾向として入出力制御装置をマイクロプログラムで制御し，かなりのデータ処理機能を実行させるものがある。次のような例について説明する。

① 汎用入出力制御装置

② 仮想ディスク装置（ＩＢＭ３８５０ システムなど）

③ 前置プロセッサ（通信制御装置など）

④ 後置プロセッサ（データベース・マシンなど）

3.4 入出力装置

入出力装置は，広義には，プロセッサあるいは主記憶装置へ入出力するための全ての装置を総称するものとして使用される。狭義には，コンピュータ・システムと外部（人間または装置）との入出力を行う装置と定義され，たとえばシステム・ファイルとして使用される磁気ディスク装置は補助記憶装置と呼ばれる。ＪＩＳ情報処理用語では総称する場合は周辺装置という。用語としてはこの他に端末装置がある。これらの用語についてはまだ統一されていないが，表３－１，表３－２に夫々ＪＩＳ Ｃ６２７２ に示されている分類と，日本電子工業振興協会作成の周辺・端末装置分類案を示す。これをもとに用語の理解をはかる。

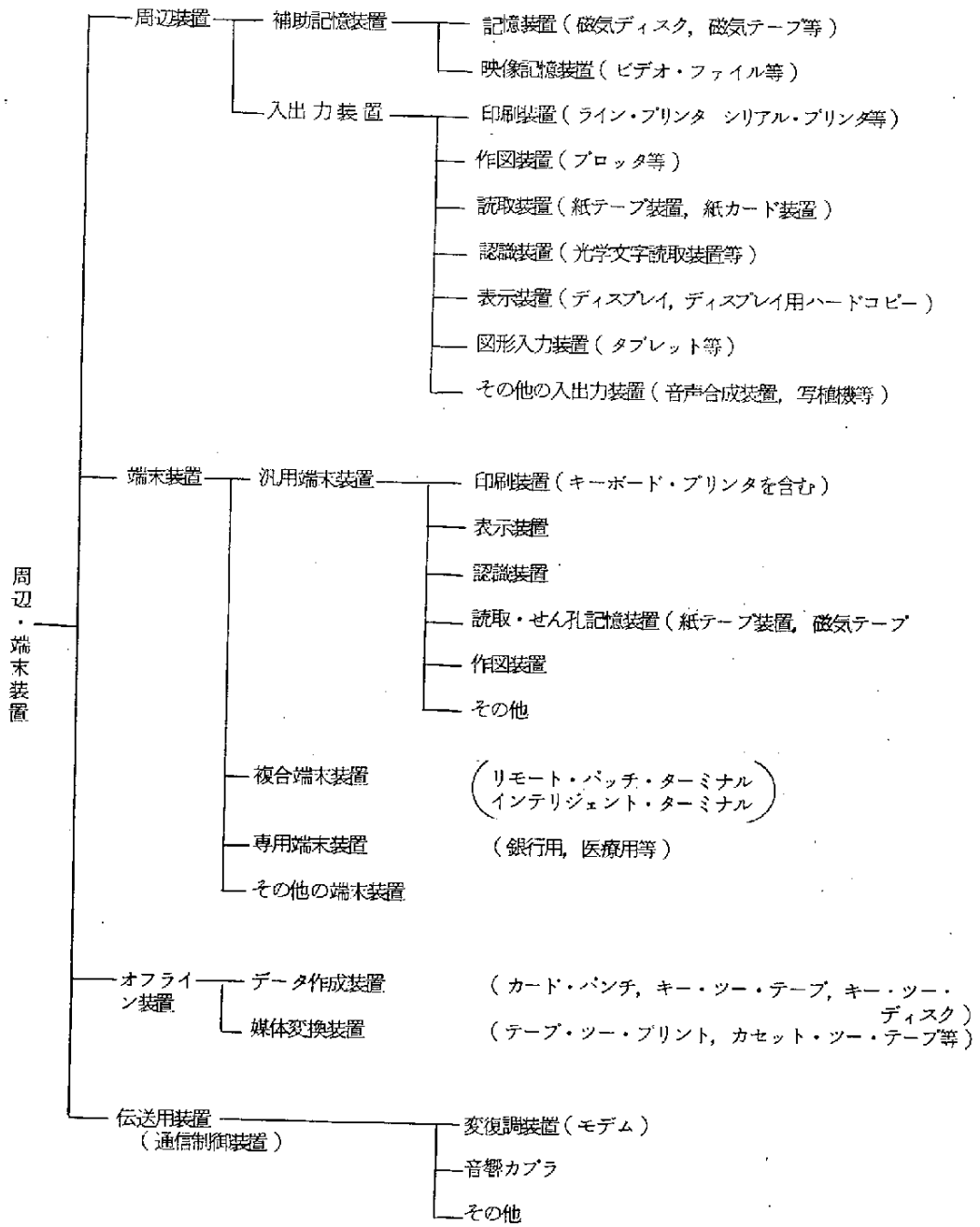
受講者は通常の入出力装置については充分な知識を有するものと考えられるので，主として最近の入出力装置を説明する。

表３－１　電子計算組織構成図（ＪＩＳ　Ｃ６２７２解説より）

電子計算

- イ．中央処理装置
  - 1. 演算処理装置
  - 2. 入出力制御装置
  - 3. 主記憶装置
- ロ．周辺装置・端末装置（各種制御装置を含む）
  - A．磁気的記憶装置（変換媒体）
    - 4. 磁気ドラム装置
    - 5. 磁気ディスク装置
    - 6. 磁気テープ装置
    - 7. 磁気カード装置
  - B．紙又はプラスチック（変換媒体）
    - 8. 紙テープ読取り装置
    - 9. 紙カード読取り装置
    - 10. 紙テープせん（さん）孔機
    - 11. 紙カードせん孔機
  - C．パターン表示，認識（変換媒体）
    - 12. 光学マーク読取り装置
    - 13. 磁気インキ文字読取り装置　E13B　CMC7
    - 14. 光学文字読取り装置　ドキュメント　ページ　ジャーナル
    - 15. X－Yプロッタ
    - 16. カーブ読取り
    - 17. A／D変換　D／A変換
    - 18. データ・ディスプレイ（CRT）
  - D．印　刷（出力）
    - 19. ライン・プリンタ
    - 20. タイプライタ
  - E．文字表示（出力又は入力）
    - 21. データ・ディスプレイ（KB／CRT）
    - 22. 文字パネル
  - F．データ通信（データの遠隔地転送）
    - 23. 通信制御装置
    - 24. 回線アダプタ
    - 25. モデム
- ハ．遠隔地端末装置
  - G．リモート・コントローラ（データの遠隔地転送）
    - 単　一
    - 複　合（多重）
  - H．リモート・ターミナル（遠隔地側操作端末）
    - 28. けん盤送受信装置（KB／PR）
    - 29. 紙テープ読取り装置／紙テープせん（さん）孔機
    - 30. 文字表示装置
    - 31. 金銭登録機
    - 32. 加算機
    - 33. 会計機
    - 34. 音声応答装置
    - 35. プッシュホン
    - 36. 磁気テープ伝送装置
    - 37. その他

表３－２　周辺・端末装置分類案（日本電子工業振興協会作成）



(1) 特殊な入力装置

特殊な入力装置として，次のような装置の機能，利害得失について説明する。

a) 文字入力

・光学文字読取装置（活字，手書文字）

・磁気インク文字読取装置

・漢字入力装置

b) 音声入力

・制限語いの音声入力

・話者識別

これらは現在研究段階であるが，近い将来実用化される可能性が高い。応用を念頭において利害得失を説明するとよい。

c) 図形入力

・タブレット（tablet）

・ディジタイザ（digitizer）

・飛点走査装置（flying-spot scanner）

d) アナログ信号入力

・A－D変換器

(2) 特殊な出力装置

次のような出力装置を説明する。

a) ノン・インパクト・プリンタ

・感熱印刷装置

・静電印刷装置

・インク・ジェット・プリンタ

b) 作図装置

・X－Yプロッタ

・ドラフタ

c) 表示装置

・プラズマ・ディスプレイ

・液晶ティスプレイ

d) 音声応答装置

e) アナログ信号出力

・D－A変換器

3.5 ネットワーク・システム

(1) システム構成

遠隔地の端末やコンピュータを通信回線を介してコンピュータと結合したネットワーク・システムには種々の形態がある。図3－2に示すネットワーク・システムの代表的な構成を説明し，その特徴を理解させる。

ネットワーク・システムの形態として，

① プロセッサと端末の結合（オンライン・システム，ＴＳＳ，ＲＪＥなど）

② プロセッサ同志の結合（コンピュータ・ネットワーク）

③ 端末同志のメッセージ通信

の三つがあることを説明する。

(2) 構成要素

ネットワーク・システムの構成要素として次のようなものを挙げ，夫々の役割について説明する。

① プロセッサ（ホストと呼ばれる）

② 通信制御装置

通信回線の制御，メッセージの処理，ホストとのデータ授受を行う。プログラム制御のものは前置プロセッサ（ＦＥＰ）と呼ばれる。

③ 遠隔集線装置

通信回線の使用効率をあげるために使用され，遠隔地のいくつかの端末からのデータをまとめて伝送する。

④ メッセージ交換装置

多数のホストを結合するネットワークで，全てのホスト間を回線で結合する代りにメッセージ交換装置を介して相互接続する方式がとられる。ＡＲＰＡ網のＩＭＰ（interface message processor）はその例である。

⑤ 端末装置

⑥ 通信回線

(3) プロトコル

システムっユーザ間，あるいはシステム構成要素間で協定した通信形式と手順はプロトコルといい，次のように分類できる。

① 処理系プロトコル

・ユーザ・レベル・プロトコル

・システム・レベル・プロトコル

② 通信系プロトコル

・論理的プロトコル

・物理的プロトコル

プロトコルの必要性，各種プロトコルの意味を説明する。システム・レベル・プロトコルはＦＥＰのネットワーク制御プログラム（ＮＣＰ）で，ユーザ・レベル・プロトコルはホストのＯＳで処理されることが多い。

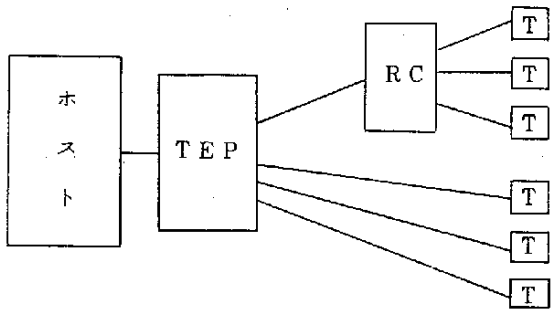
ネットワーク制御の標準化がメーカ各社ごとに進められており，各種ネットワーク・アーキテクチャが開発されている。各社のネットワーク・アーキテクチャ（ＳＮＡ，ＤＩＮＡ，ＦＮＡ，ＤＣＮ，ＡＮＳＡ，等）を簡単に説明するとよい。

ＦＥＰ：通信制御装置

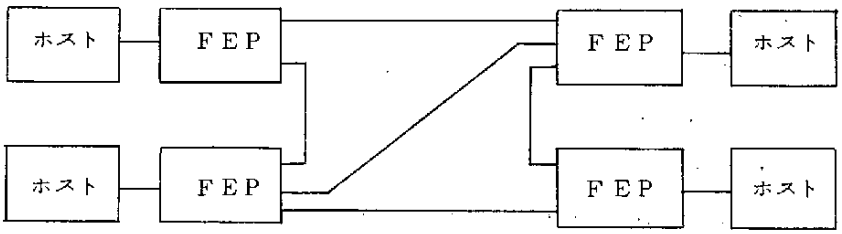
ＲＣ　：遠隔集線装置

Ｔ　：端末装置

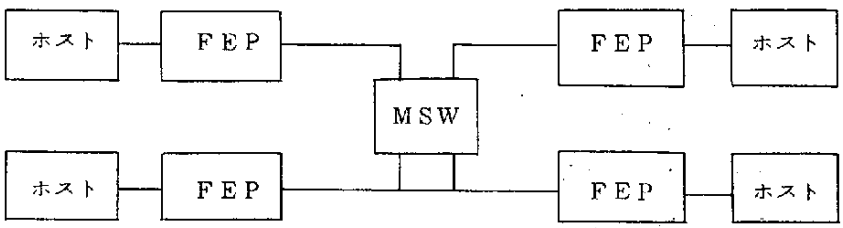
ＭＳＷ：メッセージ交換装置



a）単一ホスト・システム



b）コンピュータ・ネットワーク　(1)



c）コンピュータ・ネットワーク　(2)

図３－２　　ネットワーク・システム構成

① 配線論理による専用装置

② ミニコンピュータを利用したプログラム制御のタイプ

の二種類がある。最近の多くのシステムではプログラム制御のものが使用されている。その理由として次のような点を説明する。

① 柔軟性

回線の種類や使用されるコードに対する適応性が高い。

② 機能の向上

データ処理機能を通信制御装置に含ませることが容易であり，ホストの負荷を軽減しうる。

一般的なハードウェア構成をたとえば図3－3を用いて説明する。また，遠隔集線装置やメッセージ交換装置も基本的にはこれと同様であるが，ホスト・インタフェースが不要である。

3.6 通信制御装置

通信制御装置はコンピュータと通信回線という異質な系を結合する基本機能を持つ。

(1) 機 能

通信制御装置の主な機能について説明し，その役割を理解させる。装置によってはこれら機能の一部をホストで実行することもある。

① 回線制御

・伝送制御（シリアル・パラレル変換も含む）

・回線状態の監視

・結合路の制御（通信リンクの設定）

・エラー処理

② データ制御

・コード変換

・メッセージ形成

③ メッセージ制御

・メッセージのバッファリング処理

④ ホストとの通信制御

・チャネルとのインタフェース制御

これらの機能のどこまでを通信制御装置で分担するかによって，ホストの負荷が変る。

(2) ハードウェアの構成

通信制御装置のハードウェア構成には，

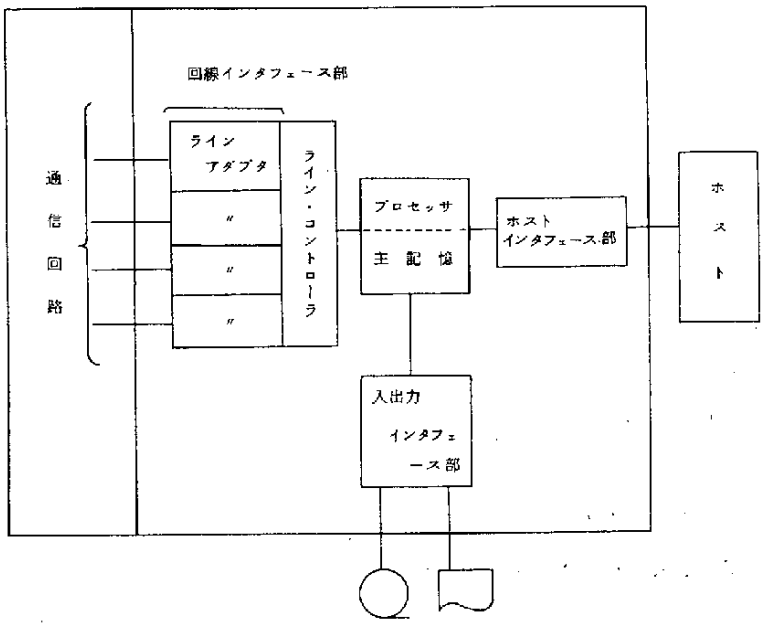


図３－３　通信制御装置の構成

3.7　端末装置

ネットワーク・システムにおける端末装置は次の機能が要求される。

①　人間にとって使いやすい

②　端末である程度のデータ処理を行える

③　通信回線の制御ができる

これらの要求に対して，知能端末（ intelligent terminal ）と呼ばれる処理機能を持った端末装置が使われることを説明する。知能端末の多くはミニコンピュータや更に安価なマイクロコンピュータが使用される。

(1)　機　能

端末装置の機能を説明し，拡張機能を含むものが知能端末であることを理解させる。

①　基本機能

・回線制御

・通信手順制御

・誤り制御

・入出力機器の制御

・同時動作の制御

・データの入出力

② 拡張機能

・オペレータに対する指示（案内）

・編　集

・チェック

・ファイル処理

・コード変換

・符号による特殊制御

・演算処理

(2) 端末装置の構成

端末装置の構成には次の二種がある。

① 自立型

1台の端末制御装置が1セットのキーボードとＣＲＴ表示装置またはプリンタ（ＫＢ／ＣＲＴ，ＫＢ／ＰＲと略す）で構成されるもの。

② クラスタ型

1台の端末制御装置が複数セットのＫＢ／ＣＲＴあるいはＫＢ／ＰＲを制御するもの。

クラスタ型は端末の拡張が容易であることを説明する。

指導上の留意点

(1) 入出力チャネルの機能はかなり複雑であり，概念的な説明だけでは理解が難しい。受講者が使用しうるコンピュータ・システムを例にとって説明し，それを補足する方法が望しい。

(2) 入出力に関しては特に用語の統一がなされていないため誤解を招かないように注意が必要である。そのためには入出力アーキテクチャの説明時に多様な用語を整理しておくとよい。

(3) 入出力アーキテクチャのソフトウェア的部分は第5章で説明するので，ここではハードウェア面に重点を置いて説明する。

(4) 入出力装置は対象者のレベルによって説明の増減を行ってもよい。

(5) 前置プロセッサおよび後置プロセッサについては多少詳しく説明することが望しい。

(6) ネットワーク・システムの構成は抽象的な説明の後に具体的なシステムの構成をとりあげ，抽象的なモデルと対比させるとよい。

(7) プロトコルの概念は比較的理解しやすいものであるが，実際にどのようなプロトコルが使用されているかについて実例をあげることが望ましい。

(8) 通信制御については機能面を十分理解させ，ホストとの機能分担における利点・欠点を理解させる。

(9) 端末装置は応用システムに対応して，多くのものがある。ここでは個別の端末装置よりも，端末に対する考え方を理解させることを主眼とする。

参 考 文 献

〔1〕 山下英男監修 「共立総合コンピュータ辞典」共立出版，pp. 750－754，pp. 762－770，1967

〔2〕 日本電子工業振興協会 「Ｉ／Ｏインターフェース －論理仕様－」45－Ｃ－213－2，電子工業振興協会 1971

〔3〕 H. カッツアン・ジュニア著，犬日方真訳 「コンピュータ・オーガニゼーションとＩＢＭシステム／370」ＴＢＳ出版会，1972

〔4〕 「情報処理（専用プロセッサの方式とシステム特集号）」vol. 18，no. 4，1977

第4章　プログラム管理およびデータ管理

キーワード

プロセス（process），資源（resource），並列処理（parallel processing），擬似プロセッサ（pseudo-processor），プロセス制御ブロック（process control block），プロセス管理プログラム（process management program），プロセスの同期（process synchronization），クリティカル・セクション（critical section），ロック（lock），セマフォ（semaphore），ディスパッチング（dispatching），実行状態（running state），実行待ち状態（ready state），待機状態（wait state），ペトリ・ネット（Petri net）

ファイル（file），データベース（data base），論理レコード（logical record），物理レコード（physical record），データ構造（data structure），ファイル編成（file organization），アクセス手法（access method），データベース管理システム（data base management system），データの独立性（data independency），データの完全性（data integrity）記憶構造（storage structure）

ジョブ（job），ジョブ・ステップ（job step），バッチ処理（batch processing），リモート・バッチ（remote batch），タイムシェアリング処理（timesharing processing），スプーリング（spooling），ジョブ・スケジューリング（job scheduling），ジョブ制御言語（job control language），ＪＣＬ，ＪＣＬマクロ（ＪＣＬ macro），オペレータ・コマンド（operator command），料金計算（accounting）

言語処理プログラム（language processor），サービス・プログラム（service programm），コンパイラ（compiler），アセンブラ（assembler），ロード・モジュール（load module），マクロ・プロセッサ（macro processor），インタプリタ（interpreter），リンケージ・エディタ（linkage editor），ローダ（loader），デバッグ支援プログラム（debugging support program），エンドユーザ言語（end user language）

目　　標

本章では，オペレーティング・システムの中核となるプロセス（またはタスク），データ管理，ジョブ管理およびプログラム管理について理解させることを目標とする。

まず，プロセスの概念について整理し，その背景となる制御構造，ディスパッチング・アルゴ

リズム，同期問題について説明する。

データ管理については，一般的な汎用オペレーティング・システム，提供しているデータ管理機能の整理を行なうと同時に，個別ファイルから一歩視野を広め，ファイルの統合化を目指したデータベース管理システムに対する基本的な知識の習得を目標とする。

ジョブ管理については，既知の知識の整理が目標であり，各種処理方式に対して統一的な立場から説明するとともに，ジョブ制御言語，ジョブ・スケジューリング方式等について説明する。

最後に，一般にコンピュータシステムを利用して，情報処理を行う場合必要なデータおよびアルゴリズムを表現するプログラム言語と，処理プロセッサ，すなわち言語処理プログラムの概要とその有効な利用法について理解させる。さらに質のよいプログラムを効率的に作成するための手法である構造的プログラミングについても，その有効性を理解させることが望ましい。

4.1 プロセスの概念

プロセス（またはタスク）という概念が出現した背景，定義，プロセスのもつ性質について説明する。

(1) 背　　景

ここでは，プロセス（タスクとも呼ばれる）という概念について，その存在の必要性を説明する。プロセスはコンピュータ・システムに依頼された「仕事」を，システム内部で代表し，システム資源の割当てを要求する主体となる，スーパバイザによって実現される概念的実体である。このような概念が導入された背景を次の三つの観点から，具体例を引用しながら説明する。

① 並列処理による処理量向上

ソフトウェア・レベルにおける並列処理を実現するために，ハードウェアのサポートの下に多重プログラミングが導入された。

② 優先処理の必要性

リアルタイム・システムなどでは後から到着した要求でも，既に処理中の仕事を中断して優先的に実行しなければならない場合がある。

③ システム動作の見通しのよさ

全体として一つの仕事を実行する場合でも，独立に並行的に動くいくつかのプログラムの集りとして記述した方が理解しやすい場合が多い。

これらの要求を実現するために，独立して並列に走行しうるプログラム実行体の概念がソフトウェア的に実体化される必要がでてきた。

(2) プロセスの定義

プロセスの概念は多くの文献で様々な，少しずつ異った定義を与えられている。大まかには次の二つの立場に集約できるので，これらについて説明する。

① 擬似プロセサ・システム

プログラムが走りうるための環境を，資源の仮想化によって提供する。

② 擬似プロセサによって実行しているプログラム

形式的には，プロセスはそのとりうる状態空間，その中における作用関数，初期状態から規規定される。状態はメモリの内容に，作用関数はプログラムに対応する。

文献では②のタイプの定義が多いが，これを実際に支えて実現するためにハードウェアとソフトウェアによって提供されているのが①で言うプロセスである。一般には①と②は区別せずにプロセス（またはタスク）と呼ばれているが，区別して名称を変えているシステムもある。

(3) プロセスの性質

並列に走行する実体としてのプロセスがもつ次のような性質について説明する。

① 互いにほとんど独立である。

② まちまちの速度で，見かけ上並行的に進行する。

③ 相互通信のための手段が必要である。

④ 子プロセスを生成しうる。

(4) プロセスが必要とする資源

実際にコンピュータの上でプロセスが走行するためには資源が割当てられなければならない。資源の割当て方にはその資源の性質によって，種々のタイプがあることを具体的に説明する。これにより，プロセスのイメージが明確になる。

① 専 有 資 源

- 専有機器：磁気テープ，端末，専有するディスクなど
- ソフトウェア：ＳＲＲ（Serially Reusable Routine）
- プロセスを代表するプロセス制御ブロック（ＰＣＢと略記）

② 共用可能な資源

a 本質的に共用可能なプログラムおよびファイル

b 時分割で共用可能なプロセッサ，入出力チャネルなど

c 空間分割で共用可能な主記憶装置，共用のディスクなど

これらの資源をプロセスに割当てるのがスーパバイザの中心的な仕事であり，特に②のｂ，ｃのタイプの割当てによってハードウェアを仮想化し，擬似プロセッサとしてのプロセスを実現可能にしている。資源の割当ての詳細については以下の各章で説明する。本章ではプロ

セッサの割当てのみを説明する。

4.2 ＰＣＢとプロセスの生成／消滅

(1) Ｐ　Ｃ　Ｂ

プロセスを一意的に識別し，代表させるために，オペレーティング・システムのプロセス（タスク）管理プログラムはＰＣＢまたはＴＣＢ（Task Control Block）と呼ぶ制御テーブルを用意している。ここではＰＣＢの代表的な内容と，それによってプロセスを代表できることを説明する。

ＰＣＢは次のような内容をもっている。

① プロセスの状態を定義する状態変数

② レジスタの退避用領域——実際にプロセッサが与えられて走行しているのでない場合，この領域の内容がそのプロセスの擬似プロセッサの状態を表わしている。

③ プロセスが所持しているか，使用することを宣言している資源に関する情報（アドレス空間の情報を含む）

(2) プロセスの生成／消滅

プロセスを生成する方法は，動的性に関していくつかのレベルに分けられる。

① システム生成と同時に作られており，計算を実行するには存在するプロセスを割当ててもらう。

② 必要に応じてプロセスを生成できる。プロセスが子供を作ることができる。

③ この中間として，ジョブステップ開始時点で対応するプロセス（群）の構造が確定される。

①は小型，専用システムに適しており，②は大型，汎用システムに適している。これらの方法の利点，欠点，適性について説明する。また，バッチ処理ではジョブ・ステップに対応してプロセス（群）が生成／割当てられることを説明する。

プロセスの消滅は一般には生成と裏返しのタイミングで行われるわけであるが，異常による消滅の際におけるプロセス管理プログラムの仕事について，特に説明することが望ましい。

4.3 プロセスの同期

(1) プロセス同期の問題の背景

システムが互いにほとんど独立に並列に動く多数のプロセスの集りとして実現されているため，各プロセスは相互に直接的，間接的に連絡をとりながら仕事を進めていく必要がある。連絡の必要性は次の二つの場合に生ずるが，その各々について具体例を上げて説明する。

① プロセス間の協力のため。

プロセス機能の専用化に伴い，他のプロセス（特にシステム・プロセス）に仕事を依頼する必要が生ずる。この場合，依頼を示す事象と依頼の内容を示すメッセージの送受のためのメカニズムが必要である。生産者一消費者問題（たとえば入出力処理プロセスとユーザ・プロセスとの関係）で典型的に現れる。

② プロセス間での資源，競合を解決するため。

複数プロセスが同一のデータ集合（主記憶上，ファイル上）を基盤として動いているが，同時更新により内容の正しさが保証できなくなる危険がある。このような資源にアクセスする部分をクリティカル・セクションといい，相互排除的（mutual exclusive）に実行されねばならない。

また，同期の問題はペトリ・ネットを用いて図式表現できるので，これについて概説するとよい。

(2) 非衝突（conflict free）型の同期

プロセス間の協力のための同期のとり方で，待ち合せ型とも呼ばれる。

① WAKEUP−BLOCK

② POST−WAIT

③ P−V

などの（マクロ）命令について，その機能を説明する。特に複数個の処理要求が到着した場合の記憶方法，メッセージの受渡し方法などに注目して説明するとよい。また，メッセージの転送に重点を置いた高水準の同期システムとして郵便受け（mailbox）システムが知られている。

(3) 衝突（conflict）型の同期

資源の競合を解決するための同期のとり方である。クリティカル・セクションを走行中は対象資源をロックしておかなければならない。

① 短期的なロック

単一プロセッサでは割込禁止にするだけでよいが，多重プロセッサでは資源の状態を表現するスイッチを設けて管理する必要がある。このスイッチの状態の確認とセットとを非可分に実行できるようなハードウェア機構が要求される。ここでは，これらのことを念頭に置き，

- テスト・アンド・セット（test and set）
- セマフォ

について説明する。既にロック中であった場合のプロセスの動作についても説明する（スピン・ロック）

② 長期的なロック

ファイルへのアクセスを含むような場合には，ロック権がとれなかったプロセスはプロセッサを離して待つべきである。ＥＮＱ，ＤＥＱマクロ命令について説明する。セマフォはこの場面でも使用できる。これらの機能を実現するには短期のロックを使用する必要があることに注意するべきである。

4.4 プロセスの状態

プロセスの状態とは，プロセスを制御するためにプロセス管理プログラムが設定している状態であり，ＰＣＢを用いて管理される。代表的な状態に次のものがある。

① 実行

② 実行待ち

③ 待機状態

④ スワップ・アウト状態

資源としてプロセッサだけを意識する場合は①～③で充分であるが，主記憶も動的に管理するＴＳＳなどでは④をも扱う必要がある。ここでは各状態の定義を与え，状態間の遷移がどのような条件で起るかについて説明する。待機状態と他の状態との間の遷移はプロセスの同期に伴って生ずる。

4.5 プロセッサの割当て

実行待ち状態のプロセスが複数個ある時，どれにプロセッサを割当てるか，がここでの問題である。この動作はディスパッチングと呼ばれ，そのアルゴリズムはシステムの性能に大きな影響を及ぼす。考慮するべきことはスループットの向上とユーザの応答時間の要求に答えることである。この観点から以下の各種アルゴリズムについて説明する。

(1) 優先度方式

① プロセス（ジョブ）としての優先度をそのまま反映する方式

② 最短ジョブを最優先する——平均応答時間が最小になる

③ ダイナミック・ディスパッチング——入出力動作の多いプロセス程優先する。

各々について実現方式として，横取り（preemption）ありとなし，タイムスライスと組合せる場合と組合せない場合がある。

(2) ラウンド・ロビン方式

平等なサービスとジョブの長さに応じた応答時間の保証を狙う。

(3) フィードバック方式

①と②組合せで，タイムスライスを使い切るごとに，優先度を落し，タイムスライスを長くする。これにより，等価的に(1)の②の実現を計る。

この中でスループット向上を狙っているのはダイナミック・ディスパッチングだけであるが，約１０～３０％向上すると言われている。実際のシステムでは，これらのいくつかが組合せて採用されている例が多い。

4.6 データ管理の基本概念

ここでは，ファイルの基本的な構成要素を説明する。

① データ

② ファイル，ファイル名

③ 論理レコード

④ フィールド，フィールド値

⑤ ボリューム

⑥ ラベル（ボリューム・ラベル，ファイルラベル）

⑦ エヤステント

⑧ ブロック（またはページ）

⑨ 物理レコード

4.7 ファイル・カタログ管理

オペレーティング・システムはデータを２次記憶上でファイルを基本単位として管理している。ここでは，そのファイルがいかに管理されるか，その基本事項について説明する。

(1) ファイルの割り当て

ファイル管理はジョブやジョブステップが実行できるようにファイルを割り当てる。ファイルの割り当ては，基本的には，内部ファイル名と外部ファイル名の対応付けを行うことをいう。ファイルの割り当て期間にはジョブレベル，ジョブステップレベルがある。また，ファイルは，その使用のされ方によって，永久ファイルと一時ファイルに分類される。

(2) ファイルの共用

ファイルの共用は同時に実行される複数のジョブステップに同一のファイルを割り当てることである。ここでは，ファイルの共用を制御するために準備される共用モードと競合が生じた場合の制御方式，デッドロックの問題について簡単に説明する。

(3) ファイル・スペースの管理

ファイルを作るにはボリューム上にスペースを確保する必要がある。スペースの管理はＶＴＯＣを通して行われ，エクステントを基本単位として割り当てられる。ここでは，二次記憶媒体上へのスペースの割り当て，解除，空きスペースの管理機能について説明する。

(4) ファイル・カタログ

カタログは，ファイルを管理するための情報の集りで，オペレーティング・システムはその情報を利用して，ファイル呼び出しの自動化，ファイルの世代管理の自動化，ファイルの機密保護，アクセス権チェック等を行う。ここでは，カタログの構造とその管理機能について説明する。

(5) ファイルの障害管理

装置や媒体の障害，プログラムの誤まり等で生じるファイルの破壊に対処するための「更新世代の管理」，「保存」，「復元」機能について説明する。

(6) ファイル定義のオーバライディング

オーバライディングとは，ファイルや装置の性質を実行時に与えたり，変更したりすることである。その目的は，ファイルや装置からの独立性を高め，それらに何らかの変更が生じてもその影響を極力少くすることにある。ここでは，プログラム，ジョブ制御言語，ファイル・ラベル情報との間でのオーバライディングの規則について説明する。

(7) ファイルの再編成と再配置

データは，一般に流動する。すなわち情報の追加，削除，更新などによりファイルの形が変化する。この変化による二次記憶スペースの効率低下，アクセス効率の低下を免れるための再編成の必要性を説明する。また，ファイルの生成，削除に伴って生じるボリューム上のスペースの細分化を防ぐための再配置の必要性を説明する。

4.8 標準ファイル編成

ファイル設計では，データを物理的にどのような編成にするか，が大きな問題である。

ここでは，ＩＢＭ３６０時代に確立された個別ファイル中心の標準ファイル編成を復習し，その選定基準を与えるため，各ファイル編成の長所，欠点を明確にする。

(1) ファイル編成の種類

ファイル編成の名前とそのファイル構造について説明する。

- 順ファイル編成
- 順区分ファイル編成
- 直接ファイル編成

- 乱ファイル編成
- 索引ファイル編成

(2) アクセス・モードと処理モード

上記のファイル編成をアクセス・モード，処理モードと関連付けて説明する。

(3) レコードの形式とアドレス表現

論理レコードと物理レコードとの関係を示すレコード形式，および，レコードのアドレス形式を定義し，各ファイル編成との関係を説明する。

レコード形式には，「固定長非ブロック」，「固定長ブロック」，「可変長ブロック」，「可変長ブロック」，「不定長」等がある。

4.9 VSAMファイル編成

前節で説明した伝統的なファイル編成の機能を集約し，総合的に再構成し，改良しようとするきざしがある。IBMのVSAM（Virtual Storage Access Method）がその一例である。ここでは，VSAMを参考にしながら，その意図と特徴を説明する。

(1) VSAMファイル編成の狙い

- 標準的なファイル編成機能の包含
- ファイル用仮想記憶の概念の導入による装置からの独立性の強化
- 自動再編成機能（これにより，ユーザはオーバフロー等を意識する必要がない）の提供
- 多角的なアクセス機能の提供
- データベース・システムへの適用性

(2) VSAM仮想記憶の概念

VSAMファイル編成では，プログラム空間とは独立に，ファイル専用の仮想記憶の概念を導入している。ここでは，その導入意義を説明し，その基本概念を標準ファイルと対応付けて説明する。

(a) 記憶スペースの構成要素

- エクステント
- コントロール・エリア
- コントロール・インタバル
- レコード

(b) レコード形式とアドレス表現

上記の仮想記憶上に，データはレコードを基本単位として割り当てられる

- ここでは，そのレコードの形式とレコード・アドレスの表現方式について説明する。

(3) ファイル編成

上記の仮想記憶上で構成されるファイル編成の名前とその構造上の特徴および標準ファイル編成との対応関係について説明する。

(a) ファイル編成

- キー順（key sequenced）編成
- 入力順（entry sequenced）編成
- 相対レコード（relative record）編成

(b) キーとインデックスの構成

キー値によるレコード・アクセスを実現するために設けられる「基本インデックス」，「代替インデックス」の構成について説明する。インデックスはB・トリーで実現される。

(4) アクセス手法

ファイル編成と対比させながら次の観点からアクセス手法を説明する。

- アクセスモード
- 処理モード
- 参照モード

アクセス手法は，標準ファイル編成と比較すると，豊富でより柔軟性を持っていることを理解させる。

(5) ＶＳＡＭの有効性

ＶＳＡＭの処理効率向上の要因となるいくつかの事柄について説明する。

(a) バッファ管理

複数のユーザ間でバッファを共用するためのバッファ・プールの概念とバッファ管理方式について説明する。

(b) 拡張再編成

- コントロール・エリアを単位とした記憶スペースの動的割り当て機能
- インデックスのB・トリーでの維持
- B・トリーで維持するため，レコードの追加，削除時の処理方式

4.10 データベース・システムの概念

前節までは主に個別ファイル中心のデータ管理機能について説明してきた。そこで，以降では，データベース・システムに焦点を合わせて説明する。ここでは，データベース・システムの概念について簡単に説明する。

(1) データベースの定義

(2) データベース・システムの出現の背景とデータベース化によるメリット

(3) データベース・システムに対する要請

- データの独立性
- データの非冗長性
- データの関連性，整合性
- データの完全性，機密保護，信頼性
- データの効率性

(4) 代表的なデータベース管理システム

4.11 データベース管理システム

データベース管理システム（ＤＢＭＳ（Data Base Management System））は，従来のオペレーティング・システムのデータ管理とデータ処理部分とを統合したものと考えられる。ここでは，ＤＢＭＳの基本的な機能を簡単に説明し，データベース設計上のキーポイントとなるデータ構造の表現と記憶構造への対応付け，および，オペレーティング・システムへの影響について概説する。

(1) ＤＢＭＳの基本機能

(a) 言語インタフェース機能

- データベース完義言語（ＤＤＬ）
- データベース処理言語（ＤＭＬ）

(b) データベース機能

- データベース定義機能
- データベース・アクセス機能
- データベース・メインテナンス機能
- データベースの共用と保護機能
- データベース・リカバリ機能

(2) データ構造

ＤＢＭＳでは，複数のユーザが共通のデータベースをアクセスするため，柔軟，かつ，統一的なデータ構造を与える必要がある。ここでは次の三つの基本構造を定義し，その具体的なイメージを植え付けるため，ＣＯＤＡＳＹＬ ＤＢＴＧ案に従って，概要を説明する。

- 単 純 構 造
- 階 層 構 造
- ネットワーク構造

(3) 記 憶 構 造

記憶構造の基本単位である蓄積レコードを定義し，蓄積レコード間の関係を示す基本的な編成技法を説明する。

(a) 順　　編　　成

(b) 直 接 編 成

- 直 接 編 成
- 乱　　編　　成
- 索引順編成，索引編成

(c) リスト編成

- 単純リスト編成
- 多重リスト編成
- リ ン グ 編 成

(4) オペレーティング・システムへの影響

ＤＢＭＳを構築する上で，従来のオペレーティング・システムに与えるインパクトについて説明する。

(a) 排 他 制 御

複数のユーザによって生じる「二重更新」による矛盾とそれを防ぐための排他制御方式，監視方式等を説明する。たとえば，ＣＯＤＡＳＹＬ　ＤＢＴＧ案のＫＥＥＰ，ＦＲＥＥ等が参考になる。

(b) チェックポイント／リスタート

汎用オペレーティング・システムが提供するチェックポイント，リスタート機能ではカバーできないケースについて説明し，どこに技術的な困難さがあるかを説明する。

(c) アクセス・インタフェースの一本化

4.12 ジョブの形態

ユーザから見て，コンピュータ・システムで処理される一まとまりの仕事であるジョブの，与え方とそれに対応する処理形態について説明する。ここでは各処理形態におけるジョブ管理の位置づけとそれに伴う諸概念を中心として説明する。各処理形態自体の意味づけはあとで扱う。

(1) バ ッ チ 処 理

① 背 景，特 徴

② 入 力 装 置

バッチ・ジョブは一般に次の各装置から並行的に入力可能である。

- カード読取装置
- 磁気ディスク，磁気テープ
- 遠 隔 端 末

特に遠隔端末から入力する場合を，ＲＪＥ（Remote Job Entry）またはリモート・バッチと呼ぶ。この方式の特徴についても説明する。

③ ジョブの概念

ジョブの定義について説明する。まとまった仕事であって，他と独立のものとされていたが，最近ではジョブ間の関係を意識できるシステムも現れている。

④ ジョブ・ステップ

ジョブを構成する要素であって，使用される資源，処理の機能の違いにより分割された処理プログラムに対応づけられる。また，実行時にはシステム内ではプロセス（群）として存在し，各種サービスを受ける。これらを具体例を上げて説明する。

(2) タイムシェアリング処理

次の各項目について説明する。

① 背 景 ， 特 徴

② ジョブの入出力装置

③ 会話（セション）の概念

バッチにおけるジョブに相当する。

④ コ マ ン ド

バッチにおけるジョブ制御言語に相当し，ジョブ・ステップの実行指令も含まれる。

⑤ インタラクション

コマンドの実行は，いくつかのインタラクションを含む。この定義と，ＴＳＳにおける応答時間の定義について説明する。

(3) リアルタイム処理

次の各項目について説明する。

① 背 景， 特 徴

② ジョブの概念との関係

一般にリアルタイム・システムは一つのバッチのジョブ・ステップとして起動され，リアルタイム・サービスの期間中そのジョブ・ステップが存在し続けるという形をとる。この場合，ジョブ管理としては特別の配慮は不要である。

4.1.3 ジョブの制御

ここではジョブが入力されてから実行され出力が行われるまでの過程を追って，ジョブ管理プログラムがどのように働くか説明する。

(1) ジョブの入力

バッチ処理ではジョブの定義（ＪＣＬとデータ）が一括してシステムへ入力され，ディスク上に蓄えられる。この仕事は他のジョブの実行と並行して行われる（スプーリング）。次の事項について説明する。

① ジョブの定義を直接アクセス装置へ蓄えておく必要性

② インプット・リーダーの起動方法

③ ジョブ入力時のチェック（システム負荷とユーザの権利の確認）

④ ＪＣＬのエラー・チェックと翻訳，ジョブ入力ファイルとデータ・ファイルの作成

⑤ ジョブ・スケジュール待ち行列への登録

(2) ジョブ・スケジューリング

スプールされているジョブから始動すべきジョブを選択する。スケジューリングの目標はスループットの向上と個々のジョブのターンアラウンド時間の要求を満すことにある。スケジューリング方式は大別して二つある。

- ＦＩＦＯ方式
- 優先度方式

これらについて説明する。後者では選択のためのパラメータとして各ジョブに，通常

- ジョブ・クラス
- クラス内優先度

をもたせ，クラス間優先度に従ってクラス別最大スケジューリング・ジョブ数の制限の下でクラスを選び，その中で優先度順に選ぶ。クラスの設定に際して考慮すべき要因について例を上げて説明する。また，クラスより直接的なスケジューリング要因として，

- ジョブの大きさ（主記憶，専有機器の台数）
- プログラム特性（ＣＰＵ使用型／ＩＯ使用型）
- ジョブを始動すべき時刻
- ジョブを終了すべき時刻
- ジョブが待たされていた時間（たとえばprority aging 機能による配慮）

などがある。これらの要因がどのように扱われるかについて，その概要を説明する。

さらに，スケジューリングの対象になる条件として

- 先行すべき複数ジョブの完了を待つ

- 実行中のジョブから起動が要求される
- 時刻の到達，オペレータの指示による

などを許すシステムもある。これらの機能の意義について簡単に説明する。

タイムシェアリング・ジョブのスケジューリングについても簡単に説明する。開始前の待ちは許されないので，負荷が高いと会話の開始を認めないことになる。

(3) 資源の割当て

スケジュールされたジョブは静的な割当ての対象となっている資源の割当てを受けて走行可能となる。対象となる資源は主記憶，専有する入出力機器，ファイルなどであるが，ジョブ開始前に一括して割当てるやり方とジョブ・ステップ単位で必要なだけ割当てるやり方がある。資源割当てについて，その概要を説明する。

(4) ジョブ・ステップの実行

ここでは

① プロセスの生成または起動

② ライブラリの探索とプログラムのローディング

について説明する。仮想記憶を採用している場合，ロードはバッキング・ストアへなされる

(5) ジョブ・ステップの終了処理

ジョブ・ステップの終了に伴って行われる後処理について説明する。

- ジョブ・ステップ単位で確保した資源の解放
- 料金計算情報の収集
- ファイルの登録，抹消
- 次のジョブ・ステップの実行開始準備

(6) ジョブの終了処理

ジョブの終了に伴って行われる後処理について説明する。

- 確保されている資源の解放
- 料金計算情報の収集
- 各種制御ブロック，ファイルの解放
- アウトプット・ライターへのジョブ出力の引渡し

(7) ジョブの出力

バッチでは一般に，タイムシェアリングでも要求により，ジョブ出力はディスクへ蓄えておき，スプーリングにより最終的に出力される。次の事項を説明する。

① ジョブ出力の内容

② ジョブ入力の場所と別の場所へも出力可能であること

③　出力クラスの利用による効率向上

4.14　ジョブ制御言語

ここでは，個々のＪＣＬ文について細い説明は省き，おもに言語としての形式とそれに関係した言語能力の面からジョブ制御言語を見直して比較，説明する。

(1)　言語としてのＪＣＬ

現在用いられているＪＣＬを文法体系から分類すると，大きく二つに分けられる。

①　汎用プログラミング言語形

②　逐次実行型（串刺し形）

②はさらに次のように分類される。

②－a　コマンド方式（各文が独立）

②－b　バッチ方式（ジョブ・ステップ単位で一まとまり）

これらの各タイプについて，次の観点から具体例を上げて説明する。

- タイムシェアリング用コマンド，オペレータ・コマンドとの共通性
- 条件判定，ジャンプの柔軟さ
- 資源割当のタイミングとの関係

(2)　基本ＪＣＬ

代表的なＪＣＬとして，

- ジョブ定義文
- ジョブ・ステップ定義文
- ファイル定義文

について，各々で指定されるパラメータについて解説し，ジョブ・スケジューリング，資源割当てとの関係について説明する。

(3)　実行制御文

- 制御変数の定義，値のセット
- 無条件／条件つきジャンプ
- ＪＣＬプログラムの呼出し

などの機能について，(1)と関連させて説明する。特に条件つきジャンプ機能の大きさと，これらの文を置ける位置の制限が(1)における分類の基準となっている。

(4)　ＪＣＬマクロ

ＪＣＬマクロ（またはカタログド・プロシージャ）の実現方式にはマクロ展開方式と，呼び出されたＪＣＬ列が条件によって自分自身を修飾していく方式とがある。この両方式について

簡単に説明する。

4.15 システム運用のための機能

(1) システム使用者の種類

システムの使用者は次のように分類できる。各々の役割と権限は使用できる制御指示の種類によって規定される(一般利用者はＪＣＬのみ)。この関係を説明する。

- システム管理責任者
- 利用責任者(使用者グループの責任者)
- 上級利用者(システム・プログラムの保守を行う)
- 一般利用者
- オペレータ

(2) オペレータ・コマンド

ここでは次の種類のコマンドについて，ジョブ管理における役割について説明する。

- ジョブ入力，出力指示コマンド
- ジョブスケジュール用コマンド
- デバイス管理用コマンド
- システムの状態表示要求コマンド

(3) オペレータ・メッセージ

ジョブ管理に関係する代表的なメッセージについて説明する。

- オペレータに対する情報通知
- オペレータに対する動作指示
- オペレータに対する応答要求

(4) 料金計算機能

システム管理責任者が課金処理や運用計画を行うために必要な情報の収集が，ジョブ，ジョブ・ステップ単位で行われる。課金の方式はシステム別に定められるべきものなので，ジョブ管理は使用した資源の量のデータのみを提供することを説明する。また，収集されるデータの種類について説明する。

4.16 プログラム管理の概要

プログラム管理とは，プログラムの作成から，実行および保守にいたるまでの，各段階において，プログラマを支援する機能の集合をいう。一般にプログラム管理は，言語処理プログラムと，サービスプログラムとから成立っている。

言語処理プログラムは，ユーザが書いた原始プログラムを実行可能なレベルである目的プログラムへ変換するためのプログラムで，事務処理向きのＣＯＢＯＬコンパイラや技術計算向きのＦＯＲＴＲＡＮコンパイラなどは代表的な言語処理プログラムである。このほかに，マクロを処理するプログラムや，データの編成や編集等を含めて定常処理向きの言語であるＲＰＧやＩＲ等の問題向き言語がある。またデータベースなどにあらかじめ蓄積されているデータを中心にして，非定形的な業務を行いたいユーザにとって使いやすいエンドユーザ言語がある。サービスプログラムとは，ソースプログラムの作成や，プログラムの結合（ｌｉｎｋ），デバッグ，保守を支援するためのプログラムのことをいう。

ここで，これらについて体系的に説明する。

4.17　言語処理プログラム

現在のコンピュータ・システムでは，ユーザの目的に応じて適切な言語を利用できる様に，多くのプログラム言語が開発されている。

たえとば．ＣＯＢＯＬ，ＦＯＲＴＲＡＮ，ＰＩ／Ｉ，ＡＬＧＯＬ，ＲＰＧ，マクロアセンブラ，システム記述用言語（ＳＹＳＬ，ＨＰＬ）などがある。また，コンピュータ・システムの利用環境によっては，ＴＳＳで使用される会話型ＢＡＳＩＣなどもある。ＣＯＢＯＬ，ＦＯＲＴＲＡＮ，ＰＬ／Ｉなどは一般に汎用高水準言語とよばれ，大ていの問題に適用できる。しかし複雑な情報をもっている問題，たとえば，輸送問題，部品展開，図形処理，言語処理，などデータが一般にネットワーク構造またはリスト構造とよばれるデータ構造をもつ問題には，ネットワーク構造を取扱い得る様な言語の利用が効率的であることを理解させる。この様な言語を問題向き言語と呼んでいる。

プログラムの品質を上げることと，プログラム開発の生産性を向上するためには，対象とする問題を含んでいる専門領域の用語で，問題を記述できるような専用言語の有効度についても検討されている。専用言語の採用は，かなりの効果があることがわかってきている。

4.18　アセンブラ言語

原始プログラムと，目的プログラムの命令コードが大体１：１に対応する言語をアセンブラ言語と呼んでいる。アセンブラ言語は機械語を記号化にしたものなので，大型のシステム・プログラムなど演算速度の速いことが要求されるところでは，アセンブラを利用することが必要になる場合もある。一般にアセンブラ言語によるプログラミングはコンパイラ言語によるそれよりも３～５倍の時間が多くかかるので，作成する対象によって，どちらを選ぶべきかを充分検討する必要がある。このためオペレーティング・システムによっては，特別に開発されたシステム・プロ

グラム作成用言語を用意している。

最近は大型のシステム・プログラムでも，このような言語を全体の９０％位まで使用し，残り１０％前後をアセンブラ言語で記述する方式が実用化してきていることに注意する必要がある。

アセンブラ言語は従来から，記述を容易にするためマクロ機能（macro facility）を含んでいる。最近は，特定の言語に依存した形でのマクロ機能でなく，プログラム言語の他に，ＪＣＬなどのようなプロセッサ用言語にも適用できるようなマクロ機能を備えている汎用マクロ・プロセッサ（generalized macro processor）が作成されているケースもあるので，その機能や使用法，特長などについてふれておくことも重要である。

4.19 コンパイラ

コンパイラはよく知られている様に，現在プログラミングに使用される言語の大部分である高水準言語を機械語に翻訳するためのプログラムであり，言語処理プログラムの中心的存在である。通常コンパイラ言語とよばれるものには，ＣＯＢＯＬ，ＦＯＲＴＲＡＮ，ＰＬ／Ｉ，ＡＬＧＯＬなどが代表的なものとしてある。会話型言語としてのＢＡＳＩＣは，一般にコンパイル方式をとるよりもインタプリタ方式で使用されるケースが多いのでコンパイラ言語には通常入っていない。コンピュータの利用形態が多様化するのに従って，コンパイラ言語もバッチ・モードだけでなく，タイムシエアリング環境，遠隔バッチ環境でも利用できるようになっている。

コンパイラ言語を利用するとき注意すべきことは，やはり記述の対象となる問題によって，すなわち事務用か科学計算用か，複雑なデータ構造をもつかもたないか，いろいろ多様で複雑な処理を必要とするか，目的プログラムが高い効率を要求するか，必要主記憶量をなるべく小さくすることを要求するかなどの諸条件を吟味して，どのコンパイラ言語を選択すべきかを検討する様に心がけることを注意すべきである。言語の選択の仕方によって問題解決の全体的な時間にかなりの差が生じることもある。

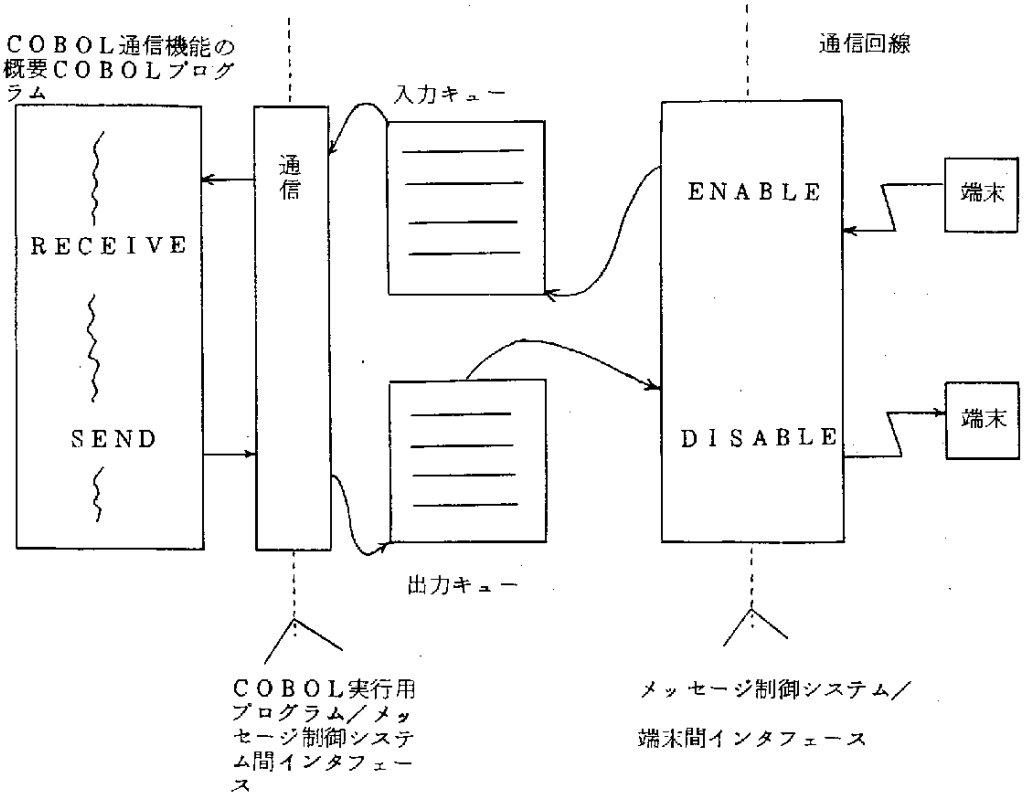
コンパイラ言語を利用してプログラミングを行い，実際の解答を得るまでの過程，すなわち，原始プログラムの作成，翻訳，リンク，プログラムのテスト，デバッグ，保守の各過程と，その際にどのようなサービス・プログラムが利用されるかを理解しておくことは，質のよい，効率の高いプログラム作成にとって大変有効である。

またデータベースの普及の進展に関連してデータベースとコンパイラ言語との関係についても検討しておくことが大切である。ＣＯＢＯＬは各種のデータベース・システムのホストとして使用されている。特にＣＯＤＡＳＹＬデータベースとＣＯＢＯＬ言語との関係が最も緊密になっている。

次に，コンパイラを使用者からみたときの問題として，最適化（オプティミゼーション）機能

とプログラム・ミスに対する診断機能の問題がある。自動的な最適化は人手によるプログラムにできるだけ近い効率を実現することが目標であるが，コンパイラのもっている最適化機能をできるだけ有効に利用する様な方法を修得するように，指導することが望ましい。また診断機能も同様に有効に利用することによって，プログラミング工数の節数と，時間的な短縮が可能になる。

さらに，オンラインシステムが普及するのに従って端末とＣＯＢＯＬの実行用プログラムとの通信が行われるが，これはメッセージ制御システム管理下で実行される。メッセージ制御機能として，ＣＯＢＯＬ実行用プログラムと端末間のインタフェース処理，メッセージの送受信のための通信回線の制御，各端末装置特有の処理（例えばコード変換など）などがある。このようなオンライン・プログラムに関連する命令（Recieve，Send）と端末とメッセージ制御システムとのインタフェース管理のための命令（ＥＮＡＢＬＥ，ＤＩＳＡＢＬＥなど）の説明が必要である。



4.20 インタプリタ

タイムシエアリング・システム（ＴＳＳ）の普及にしたがって，会話型によるプログラム作成や，会話型でプログラムの実行を行う場合が多くなってきている。この様な利用法の代表的言語として，ＢＡＳＩＣとＡＰＬがある。前者は初め小規模の科学技術計算用の会話型言語として開発されたが，その簡便さにより，次第にユーザ数をふやし，現在ではＢＡＳＩＣの基本的機能の他に事務処理用の機能をつけ加えたりした型のものまで出現しており，今後一層進展が期待される会話型処理用の言語として利用されていくものと考えられる。

ＢＡＳＩＣの処理形態の特長としては，原始プログラムを目的プログラムに翻訳してから実行するコンパイラ形式でなく，原始プログラムを一行ごとに実行時に解釈しながら実行していくいわゆるインタプリタ方式をとっている。この場合完全に原始ステートメントから実行するのではなく，ある程度中間言語レベルまで翻訳しておく方法も効率上からとられる場合がある。ＢＡＳＩＣ言語の利用は，簡単に覚えられ，簡単に使用できるということが大きな特長となっている。

一方ＡＰＬも，プログラムについての専門的知識をあまり必要としないで，必要な情報を必要な時点で短い時間で作成したり，取り出したりできる言語として最近注目される様になった。ＡＰＬ言語自身は既に古い歴史をもっているが，会話型モードによるコンピュータ利用の発展にしたがってより一般的になるものといわれている。この言語の特長の一つは他のコンパイラ言語にくらべるとずっと短かいプログラム・ステップで問題を表現できるという点にあり，さらにいわゆるプログラム的な色彩が少いという特長もある。一方記号の多様さと複雑さは一つの問題点である。

今後インタプリタ方式の言語処理プログラムはユーザ志向の言語（あるいはエンドユーザ言語）がふえるのに従って，利用される度合が高まるものと考えられる。

4.21 サービス・プログラム

実行可能な目的プログラムを作成するためのステップとして一般に，原始プログラムの作成と翻訳（コンパイラによる），翻訳の結果作られた複数個のコンパイル・ユニットをリンクして，実行可能なロード・モジュールを作る。また実行可能なプログラムのデバッグやテスト及び保守作業がある。これらの作業の中で，翻訳以外の作業の支援をするプログラムを，サービス・プログラムという。

サービス・プログラムを機能によって分類すると次のようになる。

原始プログラムの作成支援用・原始ライブラリ保守原始ライブラリの作成，保守が目的である。またテキストエデイター原始言語テキストの修正編集を行う，コンパイルユニットのリンク（結

合）の支援を行うものとしてリンケージ・エディタがある。このプログラムは，モジュールのリンクと編集，プログラムへのアドレスの割当て，プログラムの構造すなわち，オーバレイ構造や，動的プログラムの構造，動的リンク構造などを考慮して，ロード・モジュールの結合を行う機能をもつ。

またロード・モジュールに対する構造として，再生可能（プログラムは自分自身又は他のプログラムから変更されることはない）再入可能（reenterable），逐次的な再使用可能（一つのプロセスからしか使用できない），再生可能でない（一度使われると，改めてロードしないと使用できないプログラム）などを検討してよく理解させることが必要である。

指導上の留意点

(1) 本章で扱っているプロセスの概念とその実現法は，システムによって少しづつ異っている。色々な実例を対比しつつ説明することが望ましい。

(2) プロセスの実現機構（同期機構，ディスパッチング機構など）は最近ハードウェア，ファームウェアで作られるシステムが現れている。この場合，ソフトウェアからは，プロセスは完全に擬似プロセッサとして見れる。このような傾向にも触れることが望ましい。

(3) データベース，ファイルを設計する上でのファイル編成技術の選定基準がうえつけられること。

(4) ＤＢＭＳが従来のオペレーティング・システムのデータ管理に及ぼす影響について理解させる。ＶＳＡＭファイル編成は，その影響を考慮しつつ，再構成したものと云える。

(5) データベースは，何の必要性があって，生まれてきたのか，また，ＤＢＭＳがもつべき基本機能は何か，を十分認識させること。

(6) ファイルおよびデータベースの詳細は第３単元で扱う。本章ではそれらの概要がわかる程度にとどめるべきである。

(7) 受講者は少くとも一種類のオペレーティング・システムのジョブ管理機能については熟知していると考えられるので，ここでは他のオペレーティング・システムへも目を広めて既知のオペレーティング・システムの位置づけを行い，知識の整理ができるように配慮する。

(8) コンピュータ用の各種言語については，機能および利用形態について大要を理解させる。

(9) 作成すべきプログラムの種類によって，どのような言語が適切であるかを十分理解した上で使用する様に心がけさせる。

(10) 使いやすい言語は，コンピュータ・システムの利用者が増加するのに従ってますます重要性をましているので，開発すべきシステムによっては，それに適したエンド・ユーザ用言語を新たに作成することも大切なことを指摘しておくことも望ましい。

(11) プログラム作成と保守の生産性の向上，およびプログラムの品質を高める面から，構造的プログラミング，データの抽象化やウォークスルーなどのプログラム作成上の新しい考え方を十分理解させる様指導することも大切である。

参考文献

〔1〕 D.C. Tschritzis，P.A.Bernstein 著，斎藤信男訳「オペレーティング・システムの基礎」日本コンピュータ協会，１９７６

〔2〕 藤野喜一他「計算機システム基礎論」，共立出版，１９７３

〔3〕 斎藤信男「ＯＳの基礎理論(1)」，情報処理，vol. 15，pp. 887～899，1974

〔4〕 亀田寿夫「オペレーティングシステムの構成原理について」オペレーティング・システムズ・シンポジウム報告集（情報処理学会），１９７０

〔5〕 朴木実「データベース・導入と設計」企画センタ，1975

〔6〕 コンピュータ・メーカのＯＳおよび言語に関する各種ユーザ・マニュアル

〔7〕 "Data Base Task Group Report to the CODASYL Programming Language Committee April 1971"，ACM，1971

〔8〕「ジョブ制御特集号」情報処理，vol. 15，pp. 731～830，1974

〔9〕 藤井純，鈴木伸夫「オペレーティング・システム」産業図書(株)，1970

〔10〕 藤野喜一，紫合治「ソフトウェア開発と構造的プログラミング」電子通信学会誌，vol. 59，pp. 32－44，1976

第5章　システム資源管理

キ　ー　ワ　ー　ド

オーバーレイ（overlay），論理空間（logical space），記憶空間（storage space），アドレス空間（address space），アドレス変換（address mapping），スワッピング（swapping），仮想記憶システム（virtual storage system），セグメンテーション（segmentation），ページング（paging），断片化（fragmentation），ミシング・ページ・フォールト（missing page fault），デマンド方式（demand paging），先取り方式（prefetching），LRU（least recently used），ワーキング・セット（working set）論理入出力（logical input-output，LIO），物理入出力（physical input-output，PIO），論理レコード（logical record），物理レコード（physical record），バッファ（buffer），ブロッキング（blocking），デブロッキング（deblocking），入出力割込み（input-output interrupt），制御ブロック（control block），静的な資源割当て（static resource allocation），動的な資源割当て（dynamic resource allocation），デッドロック（deadlock），資源の共用（resource sharing），保護（protection），主記憶当て（storage allocation），ファイルの割当て（file assignment），入出力負荷平衡（I/O load balancing），入出力スケジューリング（I/O scheduling），性能目標（performance objective），応答時間（response time），スループット（throughput），サービス率（service rate），システム資源（system resources）

目　　標

本章では，主記憶管理，入出力管理について理解させてから，システム資源全般に関し，管理上の問題点を理解させることを目標とする。

まず，仮想記憶システムを中心とした記憶管理の意義，そのしくみ，そこで用いられる各種制御方式について理解させる。さらに，仮想記憶システムのもとになるプログラムの動特性（プログラムの局所性）について理解させ，局所性の高いプログラムを作成することの意義を理解させることも重要である。

次に，ハードウエアを制御するシステム・プログラムの機能を理解させる。プログラムが入出力動作が行なわれるまでに，いくつかのシステム機能を使用する。ここでは物理入出力からバッ

ファ管理までを理解させるものとする。入出力管理はハードウエアに密着したソフトウエアであるため，アーキテクチャに強く影響される。アーキテクチャと入出力管理との間のトレード・オフについても理解させることが望ましい。

システム資源管理はオペレーティング・システムの行う最も基本的な機能の一つである。ジョブ，ジョブ・ステップ，プロセスなどの仕事の実行体に対して，それらの必要とするハードウェア／ソフトウェア資源を，

① 論理的矛盾を起さないように

② システムの持つ能力を十分に引出し，高い性能を得られるように

割当てを行う。そこで，最後に，ここでは，単にオペレーティング・システムが行なっている資源割当て機能を個々のタイミング，資源に即して見るだけでなく，上記の目的がどのようにして実現されるかという問題意識の下に，資源割当て機能を整理し直し，理解させる。特に，今後大規模システムにおける処理形態として主流になる多次元処理の下では動的なシステム資源の管理が非常に重要な役割を演ずるので，この点を十分に理解させる。

内　　容

5.1 記憶管理の概念

主記憶装置はコンピュータ・システムの最も高価で，かつ最も重要な資源の一つである。主記憶装置を効率良く使用するか否かはシステムの性能に大きく影響する。記憶管理は主記憶装置をできるだけ効率良く使用するためのシステム機能である。

(1) 目　　的

複数のプログラムを並行して動作させる多重プログラム・システムでは，個々のプロセスに与えられる主記憶の量は限られる。一方，大規模なプログラムの作成においては次のようなことが要求される。

① モジュラリティ　　個々に作成したプログラムを実行時に結合して走らせる。

② 動的な記憶域の変更　　実行時に必要に応じてデータ領域を変化させる。

③ 物理的容量からの独立性　　プログラマがハードウェアを意識することなしにプログラムしうる。

このような要求を満足するために動的な主記憶管理が行なわれることを説明する。

記憶管理の方法は次の二つに分類される。

① プログラマがオーバーレイ等の手法を用いて管理する（マニュアル）。

② 記憶管理をシステム側が自動的に行なう（オートマティック）。

最近の多くのコンピュータ・システムでは②の方法を採用しており，仮想記憶システムと呼

ばれる。

オーバーレイ構造の問題点としては次のような点をあげて説明する。

① プログラムの作成が困難であり，プログラムの生産性が低い。

② オーバーレイされる最大の単位以上の記憶領域が必要。

③ 実行開始以前にオーバーレイ構造を決定する必要がある。したがって，実行時点で使用可能な記憶容量と無関係に構造が決定される。

④ 動的なデータ域の拡大・縮少に対処できない。

(2) 基 本 概 念

記憶管理の基本概念を説明し，仮想記憶システムに至る歴史的流れを理解させる。基本概念としては次のような事項について説明する。

a) 論理空間と記憶空間

論理空間と物理的な記憶空間の相違を理解することが基本的に必要である。論理空間はアドレス空間(address space)，名称空間(name space)とも呼ばれ，プログラムから見た記憶空間である。一方，記憶空間は物理的な格納位置(location)で表わされる実体の集合である。

b) アドレス変換

論理空間はプログラムが実行される時点には記憶空間に対応づけられなければならない。この対応づけをアドレス変換(address mapping, address translation)という。

c) 主記憶装置と外部記憶装置

プロセッサが命令で直接参照しうる記憶装置を主記憶装置といい，そうでない記憶装置を外部記憶装置という。この区別は機能的なものであることを理解させる必要がある。

d) アドレス変換の時期

アドレス変換がなされる時期について説明する。次のような時期がある。

① プログラム作成時

② コンパイル・アセンブル時

③ ロ ー ド 時

④ 手続き呼出し時(起動時)

⑤ 実行時における参照時(動的配置)

記憶管理の立場から見ると，変換はできるだけ最終的に必要な時期にまで延ばすことが望しい，上記①～⑤はそのまま記憶管理の歴史的変遷と一致していることを説明する。

e) スワッピング

主記憶装置に置かれていないプログラムやデータは外部記憶装置に格納される。主記憶装

置，外部装置間のプログラム，データの転送をスワッピングという。ロール・イン，ロール・アウトと呼ばれることもある。

f） 仮想記憶システム

仮想記憶システムはアドレス変換を実行時に行い，スワッピングを自動的に行なうものである。仮想記憶システムの基本概念を説明する。

5.2 仮想記憶システム

ここでは仮想記憶システムの各種方式について理解させる。

(1) リロケーション・レジスタ方式

本格的な仮想記憶システムが実現される以前に，動的再配置（dynamic relocation）を可能にするために考えられたリロケーション・レジスタ方式について説明する。

① アドレス変換

相対番地で与えられる論理空間にリロケーション・レジスタの内容を加えることにより，物理位置を得る。最も単純なアドレス変換である。

② 利点と欠点

この方式の利点・欠点を理解させ，後に述べるセグメンテーション，ページング方式が生み出された背景を理解させる。

利点として次のような点を説明する。

- 動的再配置ができる。
- アドレス変換に要するオーバーヘッドが小さい。
- リロケーション・レジスタと対をなす記憶領域の上限を示すレジスタ（境界レジスタ）を使用することにより，動的な記憶域の拡大・縮少ができる。

主な欠点としては次の点がある。

- 連続した記憶空間が必要
- 外部断片化（external fragmentation）（空き領域の断片化）による記憶空間の使用効率低下
- 空き領域を結合する操作（コンパクション又はちり集め）が必要

(2) セグメンテーション

プログラムとデータを論理的な単位（セグメント）に分割し，論理空間をセグメント名とセグメント内相対番地で表わす方式である。二次元アドレス指定と呼ばれる。

① アドレス変換：セグメント名からそのセグメントの記憶開始位置を求め，さらにそれにセグメント内相対番地を求める。

セグメント・テーブルによるアドレス変換を説明するとよい。記憶空間が割当てられていない場合のミシング・セグメント・フォールト（missing segment fault）についても述べる。

② 特徴：セグメンテーション方式の特徴として，次のような点を説明する。

- プログラムのモジュラリティ
- 動的に変化するデータ構造に対する適応性
- 保護（protection）
- プログラム，データの共用の管理

セグメンテーション方式は論理空間の管理に有効であることを理解させる

欠点として，次の点を説明する。

- 外部断片化
- アドレス変換のオーバーヘッド

(3) ページング方式

セグメンテーションが論理空間の管理を主な狙いとしているのに対し，ページング方式は記憶空間の管理を主目的としている。この相違を理解させることが重要である。

記憶空間を固定長のページに分割し，論理空間を固定長のページに変換するページング方式を説明する。

① アドレス変換：セグメンテーション方式と同様にページング方式のアドレス変換を説明する。次の事柄を理解させる。

- ページ・テーブル
- ミシング・ページ・フォールト
- ミシング・ページ・フォールト後の処理

② 特　　徴：ページング方式の最大の特徴は記憶空間が固定長のページに分割されているため，論理空間上では連続したアドレスが記憶空間上では不連続な位置でも良い点である。セグメンテーション方式との比較を行ない，ページング方式の特徴を理解させる。

(4) セグメント化ページング

セグメンテーションによる論理空間の管理とページングによる記憶空間の管理の両方の長所を採り入れた方式である。ＭＵＬＴＩＣＳシステムを例にとり，この方式を理解させる。

欠点として，アドレス変換の複雑さがあげられる。このような方式を必要とする環境についても述べる。

5.3 アドレス変換の高速化

アドレス変換の高速化のためにいくつかのハードウェア機構が知られている。代表的なものとして，次の二つの機構を説明する。

- 連想メモリの利用
- 動的アドレス変換機構－高速メモリによる実現（ table look-aside buffer , TLB）

5.4 仮想記憶の管理

仮想記憶の管理を行うシステム・ソフトウェアの機能を説明する。

(1) 主記憶への読み込み

プログラムの実行において実行すべきプログラムとデータを主記憶装置へ取り込まねばならない。主記憶への読み込み方式には次の三つがある。

- オン・デマンド方式
- 一括読み込み方式
- 先取り方式

それぞれの方式を説明し，その利点，欠点を理解させる。最近の大型コンピュータ・システムではオン・デマンド方式を採用しているものが多い。

(2) 主記憶領域の割当て

ページング方式では主記憶のどこに割り当てるかは問題にならないが，連続した可変長の領域割当てを必要とするセグメンテーション方式では主記憶のどこに要求されたセグメントを割り当てるかは重要な問題である。次のような方式を復習する。

- 要求領域より大きい未使用領域の中で最も小さいものを割当てる（ 最適適合 ）
- 未使用領域を探し，最初に見つかった領域へ割り当てる（ 先適合 ）
- バディ・システム（ buddy system）

割り当て方式の良否は主記憶の使用率に大きく影響することを主記憶の断片化の観点から説明し，寄せ集め（ compaction, or garbage collection）が必要になることを復習する。

(3) 置き換え（ replacement）

主記憶領域が要求されたとき，未使用領域がその要求を満足できないときには，主記憶上の領域を外部記憶装置に移送し，空きを作ることが必要である。このとき，どれを外部記憶装置に移すかを決定するかによって，システムの性能に大きい差がでてくる。オン・デマンド方式においてはこの置き換えの方式が特に重要である。次のような方式について説明する。

- ＬＲＵ（ least recently used）方式

- 到着順（ＦＩＦＯ，first－in first－out）
- ランダム（random）
- ワーキング・セット方式
- 最適置き換えアルゴリズム（optimal）

このうち，最適置き換えアルゴリズムは将来のページ（セグメント）参照系列が判っていることが必要であり，実際には実現できないが，方式間の評価基準として使われる。

(3) プログラムの動特性

比較的小さい主記憶装置で大きい仮想記憶システムを実現してもある程度の効率を得られるのは，プログラムが局所性（locality）を持つことによる。プログラムの局所性について説明し，仮想記憶システムにおいて局所性の強いプログラムが望しいことを理解させる。

プログラムの局所性を表現するモデルには下記のものがある。

- ライフタイム・モデル（lifetime model）
- ワーキング・セット・モデル
- スタック・モデル

プログラムの局所性を増すために，プログラムのページ参照系列を求め，解析することにより，再構成する手法もある。

5.5 入出力管理の基本概念

プログラムが入出力装置上のファイルを使用するまでには多くのシステム機能の助けが必要である。入出力管理はそのうちのチャネルを介した入出力ハードウエアの動作を管理するシステム・ソフトウエアである。

(1) 論理入出力と物理入出力

プログラム上で指定する入出力の動作（ＧＥＴ，ＰＵＴなど）は必しも物理的な入出力動作と一対一に対応しないことを説明し，次のような基本的概念を理解させる。

① 論理レコード

プログラムから見た入出力単位であり，論理入出力（logical input-output，LIO）は論理レコードに対して行われる。

② 物理レコード

実際の入出力装置上での入出力単位である。複数の論理レコードが一つの物理レコードに対応する場合，論理レコードが物理レコードと一対一で対応する場合，一つの論理レコードが複数の物理レコードにまたがる場合などがある。

③ バ ッ フ ァ

論理レコードに対する論理入出力と物理レコードに対する物理入出力（physical input-output，ＰＩＯ）の並行動作の同期をとるために使用される主記憶上の領域である。

④ ブロッキングとデブロッキング

複数の論理レコードを一まとまりとして一つの物理レコードにする操作をブロッキング，その逆の操作をデブロッキングという。

⑤ 論理入出力

⑥ 物理入出力

⑦ レコードの形式

- 固定長レコード
- 可変長レコード

(2) 入出力管理の機能

入出力管理の機能を説明する。次のような機能を含む。

① チャネルに対する入出力動作の起動

② 入出力割込み処理

③ エラー回復処理

④ 入出力要求の待合せ処理

これらの機能は物理入出力に関する機能であるが，論理入出力の管理として，次の機能を含めることもある。

⑤ バッファの管理

⑥ ブロッキングとデブロッキング

プログラムが使用するファイルの管理も含める場合もあるが，最近のオペレーティング・システムではアクセス技法と呼ばれるデータ管理を別に用意する方が普通である。

入出力管理機能をシステムが集中的に制御することの利点について理解させることが望ましい。

(3) 入出力管理のインタフェース

入出力管理は次の二つのインタフェースを持つ。

- データ管理システムやユーザ・プログラムのようなソフトウエア
- チャネル，入出力装置などのハードウエア

ソフトウエアに対するインタフェースにはＥＸＣＰなどに代表されるシステム・マクロが用意される。また，ハードウエアとのインタフェースは入出力命令，チャネル・プログラムや割込み機構などを使用する。

COBOL，FORTRANなどの高水準言語で書かれたプログラムではコンパイラがデータ管理や入出力管理のシステム・マクロを用いて入出力動作が行われるが，その過程を説明する。

5.6 バッファの管理

バッファはCPUの高速な演算速度と入出力のように速度が異なり，かつ同時に動作する装置の同期のために用いられる。バッファの制御方式について説明し，入出力動作がバッファの制御と密接な関連があることを理解させる。

(1) バッファの種類

バッファの種類とその用法について説明する

次のような種類がある。

- 固定バッファ（fixed buffer）
- 二重バッファ（double buffer）
- 動的バッファ（dynamic buffer）

(2) バッファの制御動作

各種のバッファに対する制御動作を説明し，その効果を理解させる

5.7 入出力管理の構成

物理入出力管理プログラムの構成を説明し，入出力管理の動作を理解させる。主に，次のプログラムで構成させる。

- 入出力動作の起動（イニシエータ）
- 入出力割込み処理
- エラー処理

(1) 入出力動作の起動

入出力要求を受け取り，入出力チャネルや入出力装置の状態に応じて，入出力動作の起動あるいは待合せ処理を行うのが基本機能である。システムのアーキテクチャやオペレーティング・システムによりかなり差異があるが，一般には次のような処理を行う。

① 入力データを検証する。

② チャネル，入出力制御装置や入出力装置の状態をチェックする。

③ 入出力動作が実行可能であれば，入出力動作を行うことを登録し，チャネルを起動する。

④ 他の入出力動作が行われているため，直ちに実行できない場合，その要求を入出力要求待ち行列に登録する。

⑤ 入出力割込みをプログラムに通知するための処理を行いユーザ・プログラムに戻る。

以上のほかに，仮想記憶システムを採用しているシステムでは，入出力動作中に対象となるページやセグメントの追い出しが起らないようにする処理，あるいは論理アドレスを物理アドレスに変換する処理も行われる。

(2) 入出力割込み処理

入出力割込みが起ると割込み処理ルーチンにより，次のような処理がなされる。

① 入出力割込みが正常な入出力動作の終了によるものが，エラーによるものか，アテンションによるものかを調べる。

② 正常な動作終了によるものであれば，プログラムに終了を通知する処理を行ない，その入出力動作の終了を待っているプログラムがあるか否かを調べ，待っているものがあればその処理を実行する。

③ エラーによる割込みの場合には，エラー回復処理を行なう。

④ アテンション割込みの場合は，それの処理ルーチンが起動される。

⑤ 割込み処理が終了すると，ユーザ・プログラムに制御が戻される。

(3) 制御ブロック

入出力管理では多くの制御情報が必要である。それらの制御情報をまとめて制御ブロックを作り，管理する方法が用いられる。制御ブロックの構成を具体例により説明し，主要な制御ブロックについて説明する。

- イベント制御ブロック
- ユニット制御ブロック

5.8 入出力管理の高速化

入出力管理は各入出力命令ごとに動作するため使用頻度が極めて高いシステム機能である。それの高速化はシステム性能を向上させるのに大きい効果がある。そのための手法について説明し，その利害得失を理解させる。

(1) 専用プロセッサ化

入出力管理機能をＣＰＵと独立に動作する専用プロセッサに行わせる。次のような例がある。

- 入出力プロセッサ（ＣＤＣ　６６００）
- データベース・マシン

(2) Ｓ／Ｆ／Ｈトレード・オフ

ソフトウエア機能をマイクロプログラム（ファームウエア）やハードウエアに置き換えることをＳ／Ｆ／Ｈトレード・オフという入出力管理機能の一部をファームウエアやハードウエア

に移すことにより高速化をはかることが最近多くなってきている。この傾向とその背景として，ハードウエア価格の低下があることを理解させる。

- 割込み処理のハードウエア（ＡＣＯＳシリーズ）
- チャネルにおけるアドレス変換（Ｍシリーズ）
- デバイス待ち行列のチャネルによる制御（ＡＣＯＳシリーズ）
- ファームウェア化（ＩＢＭ／３７０ 拡張機構）

(3) 入出力制御装置の機能向上

入出力制御装置の機能を向上することにより，入出力回数の減少をはかる。

- ディスクのキー・サーチ
- 回転角検出機構
- 知 能 端 末

5.9 システム資源

ジョブ，ジョブ・ステップ，プロセスなどの仕事の実行体は，各々が必要とするシステム資源を割当てられて初めてシステム上に存在しうるかあるいは実行可能になる。ここではシステム資源の概念について説明する。

(1) システム資源の分類

システム資源は割当ての観点から次のように分類される。

① 専有資源

a 専有機器：磁気テープ装置，端末，専有されるディスク装置など

b ソフトウェア資源：ＳＲＲ，ＰＣＢ，ＪＣＢ（Job Control Block）など

② 共用可能な資源

a 本質的に共用可能：プログラム，ファイルなど

b 時分割で共用：プロセッサ，入出力チャネル，ディスク装置など

c 空間分割で共用：主記憶領域，共用のディスク，ドラムなどの領域

③ 仮 想 資 源

スプールされているカード，リーダ，ディスクでシミュレートされている磁気テープなど専有資源と時分割で共用される資源との差は専有している時間の長さの差だけであり，どちらに関しても割当て問題が存在することに注意すべきである。

(2) システム資源の割当てのタイミング

割当てのタイミングとしては次の３種類があることを説明する。動的とはプログラム実行中の必要時点で割当てることを指している。

① 静的な資源割当て

a ジョブ開始時

b ジョブ・ステップ開始時

② 動的な資源割当てをする程，資源の使用効率は良くなりうるが，デッドロックの問題などが生ずる可能性があり，複雑な制御が必要となることを説明する。

(3) 共用と保護

システム資源は原則として，すべてのプロセスから共用されうる。そこで資源の保護が必要となるが，要求される保護の性格は次の二つに分類される。

① 共用は許されているがアクセスのタイミングが問題となる場合（排他制御）

② アクセスの権利が問題になる場合（プロセスと対象との静的な関係）

これらについて，具体例を引用しつつ説明する。

5.10 デッドロック

(1) デッドロックの定義

資源割当てにおいて起りうる論理的矛盾のうち最大の問題はデッドロックである。資源を要求しているプロセスが，どれもシステムに存在する以上の量を要求していないのに，互いの実行の進展を妨げあっている時，システムはデッドロック状態にあるといわれる。ここでは具体例を引用してデッドロック状態の説明を行う。

(2) デッドロック対策

デッドロックが生ずる必要条件について説明する。この内一つを破れば防止できる。

- 専 有 条 件
- 待 ち 条 件
- 非プリエンプト条件
- チェイン条件

また，デッドロック対策としては

- デッドロックの検知と回復
- デッドロックの静的／動的防止

があることを説明する。非常に稀にしか起らない場合は，検知と回復によるのが実用的なこともありうる。

(3) デッドロックの静的防止

以下のような解が知られ，実際に適用されている。これらについて説明する。

- 一括要求（all or nothing）

- 定 順 要 求
- 待ち条件の解除　資源を途中で手離すことを可能にしておく（後に割当てを受ければ続行できる）

(4) デッドロックの動的防止

システムの状態を観測しつつ，割当てを行いうるので，資源の使用効率がよい。次の方法のうち一つについて説明する。

- Habermann の方法
- Hebalkar の方法

資源割当ての要求の度にチェックし，割当てるか待たせるかを決定する。資源の種類が多い場合は実用的とは言えない。

5.11 静的な資源割当て

ジョブまたはジョブ・ステップの開始時に行われる資源割当の方法について説明する。同一の資源が，システムによりまたは要求の方法によっては動的にも割当てられうる。

(1) ＪＣＢ，ＰＣＢなど

ジョブ，ジョブ・ステップ，プロセスがシステム上に存在するために必要な資源である。一般に個数に制限がある（ジョブの最大多重度，最大プログラム多重度などとして）ため，競争の対象となる。ＰＣＢは動的に割当てられるシステムもある。

(2) 主記憶領域

① 仮想記憶方式のシステム

ジョブ・ステップ開始時に，多重度，空きメモリサイズの確認を行う程度の場合が多い。

② 仮想記憶方式でないシステム

- パーティション方式：ジョブ開始時に最大のジョブ・ステップ分を確保する。
- リージョン方式：ジョブ・ステップ単位に必要サイズを確保。スワップアウト，コンパクションを伴う場合もある。

両者の利害得失について説明する。

(3) ファイル

ファイルの割当ては一般にＪＣＬで要求される。要求の内容は大別して次の二つになる。

a）既存ファイルの使用要求

b）新しいファイルの定義の要求

a）に関しては論理的には排他制御だけを考慮すればよく，物理媒体に関してはカタログを通して知られる対象ボリュームがマウントされていない場合に装置割当ての問題が生ずる。

b）に関してはボリュームが指定されている場合に装置割当ての問題が起り，その他にディスク領域割当ても行なわなければならない。これらファイル割当の機能について資源管理の立場から説明する。なお，ファイルはジョブ・ステップ内だけで使われる場合と，ジョブ・ステップ間で受渡す必要のある場合とがある。

(4) 装　　置

専用ボリュームまたは非常駐ボリュームのために要求される。次の事項について説明する。

① 装置・グループの概念とその有効性

② 装置の確保方法

a）ジョブ・スケジュールの前／後に確保

a－1） 全ジョブ・ステップ要求数の和

a－2） 各ジョブ・ステップ中の最大要求数だけ

b）ジョブ・ステップ開始時に確保

③ 自動ボリューム認識機能

(5) ディスク領域

一時ファイル，新しい永久ファイル，既存ファイルの拡張のために要求される。割当て対象に自由度がある場合，その時点におけるシステム全体としての入出力負荷をバランスさせるように対象ボリュームを選ぶのがスループットの面から有効であることが知られている（入出力負荷平衡）。

(6) 静的割当ての順序

割当ての順序を規定する方法として，

- 一括割当て（1レベル）
- 多段割当て（多レベル）

がある。多段割当てでは，たとえば周辺機器の割当てが完了しないと主記憶領域の割当てを行なわない，などの制御を行う。いずれの方法においても，レベル内の必要資源が一括しては割当て不可能であった場合，

- 一部分でも確保して残りの空きを待つ
- 全部確保できなければ全然とらない

の二つの対処がある。以上について，効率とデッドロック対策およびジョブの緊急性に対する配慮の面から説明する。

5.12 動的な資源割当て

(1) ファイル，装置，ディスク領域

これらについては，ＪＣＬによる静的な割当ての他に，プログラム実行中にマクロ命令などによって割当てを要求できるようになっているシステムも多い。この動的割当て機能について説明する。

ＪＣＬで予約しておき実行中の必要時点で割当てを受ける方式と，予約なしの方式とがある。前者の方式はデッドロック対策上およびファイルに関してプログラム独立性の上から好ましく，後者は資源使用効率と柔軟性において勝る。ファイル用のディスク領域の拡張は大部分のシステムで動的に可能になっている。

(2) 主記憶領域

次の三つの事項について説明する。

① 基本的に静的な割当てを行っている場合

マクロ命令によって，システム領域あるいは自分のユーザ領域から作業領域などを割当ててもらえる。

② 仮想記憶方式におけるページ置換え

ユーザは主記憶領域を意識しなくなっている。

③ スワッピング

プロセスまたはジョブ・ステップを単位として主記憶とバッキング・ストアの間で出し入れする。この手段の適切な利用により緊急なジョブの応答時間の保証，全体のスループットの向上を計りうる。

(3) 入出力スケジュール

入出力要求が競合する場合の実行順序の決定が問題となる。チャネル，ディスクなどは多くのプロセスから共用されるので，競合が起る。スループット向上のための次の方式について説明する。

- Smallest Rate First 法　　その装置を使う比率の小さいプロセスからの要求ほど優先する
- ディスクのシーク最適化法　　シーク時間が最短になる順序に要求を並べる

(4) ＣＰＵ

レディ状態のプロセスに対するＣＰＵのディスパッチングの問題である。

5.13 性能目標達成のための制御

大規模システムにおいて主流となりつつある多次元処理においては，個々のジョブの応答性要求とシステムに与えられている性能目標を，変動する負荷状況に適応しつつ実現するような動的

な制御が必要とされる。

(1) 性 能 目 標

次のような目標が考えられる。各々について具体的な目標の与え方を説明する。

① 応答時間またはバッチ処理時間の保証

a 相対的な優先度で指定

b サービス率*を指定

c 希望応答時間を指定

② スループットの向上

目標を陽に指定させることはない

③ サービス配分

たとえばバッチとＴＳＳのサービス比率を規定するなど

(2) ジョブ・スケジュールにおける制御

次の代表的制御方法について説明する。

① 応答時間の保証

- スケジュール目標時刻の設定
- 優先度の指定と時間経過に伴うその上昇

② スループットの向上

- ジョブ・クラス機能による，ジョブ・ミックスのバランス
- Ｉ／Ｏ型，ＣＰＵ型の宣言によるジョブ・ミックスのバランス

(3) スワッピングによる制御

次の代表的制御方式について説明する。

① 応答時間の保証

- サービス率実現のためのスワッピング
- 優先度とタイムスライス
- デッドライン制御

② スループットの向上

- ＣＰＵ，チャネル使用率が不適切な場合，これを回復するスワッピング

③ サービス配分

- タイプ別多重度の監視と制御

(注)* サービス率は，そのプロセスが単位時間当りに受けるサービスの量（ＣＰＵ時間，Ｉ／Ｏ回数，メモリ占有面積から求める）で規定される。

- タイプ別サービス率の監視と制御

(4) ＣＰＵ，入出力スケジュールにおける制御

スループット向上のため，方法は既に説明した。応答時間の保証のためには，プロセスの優先度も配慮に入れる必要がある。

(3)，(4)で扱った制御はシステム特性，プログラム特性などの測定に基づいて，動的にフィードバックをかけていくことにより実現される。目標①，②，③の達成度の間でバランスをとるための機能も必要とされる。

指導上の留意点

(1) 記憶管理を理解するためには論理空間と記憶空間およびその間の変換の概念を明確に理解させることが必要である。

(2) セグメンテーションとページングについてはＭＵＬＴＩＣＳシステムを例にとって説明するとよい。MULTICSに関しては多くの文献があり，本質的な目的を理解するのに都合がよい。

(3) 時間的な余裕があれば，局所性を高めるためのプログラム再構成についても説明することが望ましい。また，局所性の高いプログラムを作成するにはどうすれば良いかを考えさせるのも効果があると思われる。

(4) 入出力管理の用語やその範囲はオペレーティング・システムによりさまざまである。まず，基本的な用語を明確にすることが大切である。

(5) データ管理との関係を理解させ，プログラム言語のレベル（高水準言語，アセンブラ）により，直接使用する機能レベルが異なることを判らせる。

(6) 入出力管理の機能はそれ程複雑ではないが，実際には多くのチェックを含んだかなり複雑なものになっている。処理フローを理解させた後，余裕があれば実際の入出力管理プログラムの一部を追跡することは非常に効果がある。

(7) 制御ブロックは具体的なシステムを例示した方が判りやすい。

(8) デッドロックの問題は現実のシステム設計上でも無視できない，やっかいな問題である。ここで理論的な形式化と取扱い法を把握させておくことが望ましい。

(9) 性能目標達成のための制御に対する要請は今後増々強くなり，その重要性を増してくる。この節についてはＩＢＭのＯＳ／ＭＶＳにおけるＳＲＭ（System Resources Manager）を具体例として取上げることが望ましい。

参考文献

〔1〕吉沢，黒沢「バーチャルメモリについて」情報処理，vol. 14，no.10，pp.701－707，778－785，1973

〔2〕T. Kilburn et al，"One－level storage system"，IRE Trans. EC－11，pp.223－235，1962

〔3〕P. J. Denning，"Virtual memory"，Computing Surveys，vol.2，pp.153－189，1970

〔4〕「ジョブ制御特集号」情報処理，vol. 15，pp. 731～830，1974

〔5〕斉藤信男「OSの基礎理論(2)」情報処理，vol. 15，pp. 989～998，1974

〔6〕亀田，鳴田「処理装置資源の一適応制御方式のシミュレーションによる評価」情報処理学会17回大会予稿集，pp.191

〔7〕H.W. Lynch，J. B. Page，The OS／VS2 Release 2 System Resources Manager，IBM Systems Journal，vol. 13，pp.274－291，1974

第6章　メッセージ管理

キーワード

オンライン・システム(on-line system), 支援プログラム(support program), プロセス(process), 前置プロセッサ(front end processor), メッセージ・キュー(message queue), キューイング・ファイル(queuing file), 転送制御(transmission control), メッセージ制御(message control), ドライバ(driver), 伝送制御手順(transmission control procedure), 障害管理(failure management), 復旧(recovery), チェック・ポイント(check point), システム・リスタート(system restart)

目　標

メッセージ管理は汎用オペレーティング・システムの管理機能の一部として，オンライン・システムを実現するために用意された，端末とのメッセージのやりとりを管理するための機能である。オンライン・システムはメッセージ管理以外のオペレーティング・システム機能も活用して構築されるが，ここでは既知のこれらの機能がオンライン・システムの実現において，どのように用いられるかについて理解させることを一つの目標とする。さらに，メッセージ管理の各機能が，システム内におけるメッセージの処理の過程においてどのように介在するかという観点から，ソフトウェアの構成，機能，特徴について理解させることを目標とする。

内　容

6.1　オンライン・システム

メッセージ管理機能はオンライン・システムを実現するためにオペレーティング・システムの一部として用意されている機能である。ここでは主にオンライン・システムを構成するのに必要な，メッセージ管理以外の機能について概要を把握させる。

(1)　オンライン環境の特徴

バッチ処理に比較して複雑な制御ソフトウェアが要求される理由を説明する。

①　応答時間に対する要求の厳しさ

②　処理要求のランダムな発生

③　多数の端末と多数の処理要求の並行処理の必要性

④　信頼性に対する要求の厳しさ

(2) オンライン・システムのソフトウェア構成

オンライン・システムのソフトウェアはその機能面から分類すると次のようになる。ここでは，これらの概要を説明する。

① 管理プログラム

一般のバッチ処理用管理プログラム以外にメッセージ管理プログラムが必要である。

② 処理プログラム

各オンライン・システム固有の業務を行うプログラムで，メッセージ入出力を除いては特にオンラインを意識して作成する必要はない。メッセージ入出力についてはマクロ命令が提供されている。

③ 支援プログラム

- システム診断プログラム
- デバッグ用ユーティリティ
- システム・メンテナンス・プログラム
- 制御構造生成プログラム
- 障害回復プログラム

(3) オンライン・システムとジョブの概念の関係

汎用オペレーティング・システムの上に構築されるオンライン・システムは，ジョブ管理に対しては，一つまたは複数のジョブ・ステップの集りとして定義される。バッチ・ジョブと同様にスケジューリングされ，資源の割当てを受けてシステムが開始されることを説明する。また，メッセージ管理プログラムの共通部分はオペレーティング・システムの一部として存在し，各々のオンライン・システムに固有の部分は処理プログラムと独立な，あるいは同一の，ジョブ・ステップとして起動される。これにより，複数のオンライン・システムが同一のコンピュータ・システム上に共存できる。

(4) トランザクションとプロセスの関係

プロセスのとらえ方には，汎用の実行体としてと，特定の機能の実行体としての二通りがある。オンライン・システムの実現法にもこれに対応して，明瞭に異なる2種類が存在することを説明する。

① トランザクション対応にプロセスが生成または割当てられて，プロセスが必要なプログラムを渡り歩く。

② 処理プログラム対応にプロセスが用意されていて，トランザクションが処理に必要なプロセスからプロセスへと受渡される。

(5) システム内通信機構

(3)における①，②に対応して用意される次のシステム内通信機構について説明する。

①の場合：プロセス内部の問題。プロセス固有の作業領域を用いる。

②の場合：通信領域と同期機構が必要になる。外部メッセージ用の待ち行列を経由する方法と，メッセージつきセマフォなどを利用する方法とがある。

(6) 前置プロセッサとメッセージ管理の関係

前置プロセッサはメッセージ管理機能の一部をミニコンで置き換えて，ホストのＣＰＵ時間と主記憶を節約しようとするものである。ここでは前置プロセッサとホスト側との機能分担の実現法，インタフェースについて説明する。

6.2 端末アクセス方式

(1) メッセージ管理の構成とデータの流れ

図６－１を引用して，メッセージの流れに沿ってメッセージ管理を構成する各プログラムの機能の概略を説明し，以下の詳細な説明への展望を与える。

(2) ネットワーク制御

仮想リンク（通信線を時分割により仮想化したもの）の概念について概説し，この概念に基づくネットワーク制御機能を説明する。

- 仮想リンクの設定と解放
- 端末の開設と閉鎖
- ネットワーク状態の報告
- 通信回線の開設と閉鎖

(3) 処理プログラム・インタフェース

次のマクロ命令について説明する。扱う単位はメッセージである。

- ＳＥＮＤ
- ＲＥＣＥＩＶＥ

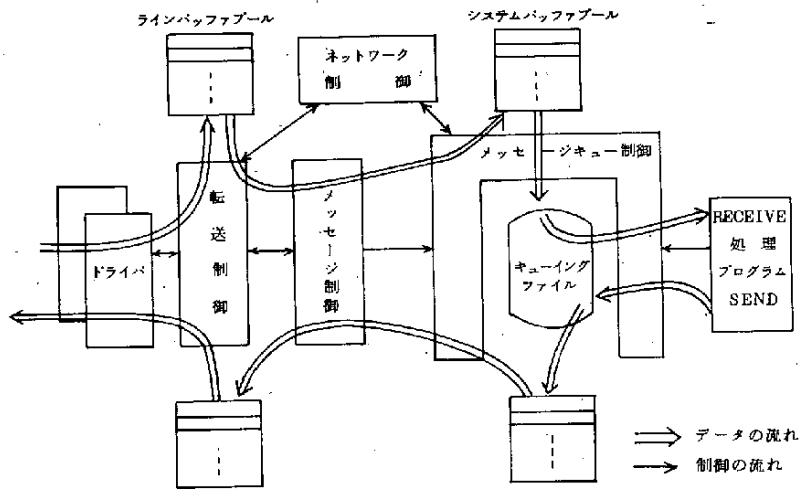


図 6.1　メッセージ管理の構成とデータの流れ

(4)　メッセージ・キューの構造

処理プログラムと転送制御プログラムの中間にメッセージのキューイング・ファイルが置かれ，これを介するメッセージ交換機能（メールボックスと呼ばれることもある）が提供される。一般的な形としては，

①　データ・キュー端子

②　キューイング・ファイル

③　プロセス・キュー端子

からなり，①から②へ向って，また，③から②へ向って木構造をとりうる。この構造の意味と各端子の役割，キューイング・ファイルの実現法について説明する。

(5)　メッセージ・キューの制御

以下の制御機能について，上記のキュー構造と関連づけて説明する。

①　送信受信の優先度制御

②　メッセージの同報扱い（broadcasting）

③　メッセージの代表送信扱い（ラウンド・ロビンまたは最少キュー方式）

6.3　メッセージ転送制御

(1)　転送スケジュール

回線が送受信の間で，またはいくつかの端末の間で共用されている場合の制御機能である。次の機能について説明する。

① スケジューリングの開始と終了処理

端末の開設，閉鎖に伴う。

② 送受信間の優先度制御

③ ポーリング端末に対する受信順序制御

リニア，ラウンドロビンなど

④ 送信端末の順序制御

リニア，ラウンドロビンなど

⑤ 端末ビジーによる一時受信の実行

⑥ メッセージ排他制御（送受信間）

⑦ 特定端末の長時間回線占有の防止

(2) 端末／回線共通制御

種々の通信ハードウェアに関する制御に共通した機能として次のものを説明する。

① 回線／端末の状態の管理

② 受信処理

ドライバの起動，メッセージ制御への登録依頼

③ 送信処理

メッセージ制御への取出し依頼，ドライバの起動

④ メッセージ・キューへの登録単位の認識

(3) メッセージ制御（ＭＣＰ）

転送制御の機能の中で最も処理プログラム寄りの機能であり，一般にシステムごとに作られる。所有すべき機能を説明する。

① 入力メッセージ制御

- メッセージの編集
- 登録すべきデータ・キュー端子の決定
- キュー・ファイルへの登録単位に編集
- データ・キュー端子への登録

② 出力メッセージ制御

- プロセス・キュー端子からの読出し
- メッセージの編集（出力端末に合せて）
- ラインバッファへのメッセージ移送

6.4 ドライバ

ドライバは，端末への送信，端末からの受信，端末／回線動作の制御と監視を伝送制御手順，回線／端末の特性に従って制御する。端末と回線の組合せごとに作成され，端末／回線共通制御の管理の下で動く。以下の基本的機能について説明する。

① 伝送制御手順の実行

伝送制御手順について，基本手順，高水準手順の代表的なものをとり上げて説明する。両者の利害得失にも触れる。

② エラーの検出と復旧

伝送制御手順上で検出できるエラー，ハードウェアの異常を管理し，再試行による回復を試みる。

③ メッセージ発信源の識別

回線―端末制御部―端末を識別し，入力メッセージ制御へ通知する。

④ 特殊文字受信の識別

ハードウェアで自動的に検出されて知らされるが，これを入力メッセージ制御へ通知する。

⑤ メッセージ転送終結文字の識別

ＥＴＢ，ＥＴＸなどを検出して入力メッセージ制御へ通知する。

⑥ 端末／回線共通制御とのインタフェース

メッセージ送受信の要求を受けて動作し，その結果をリターン・コードとして通知する。

⑦ 入出力制御とのインタフェース

端末との送受信のために，チャネル・プログラムを作成し，入出力制御へ実行を要求する。入出力完了コードを受取って分析し，処置を行う。

6.5 障害管理

オンライン・システムにおいては信頼性に対する要求が極めて強い。コスト面からの制約も考慮して，障害が起ってもなるべく早く回復可能なシステムとしておかなければならない。ここでは障害対策の方法について概説する。

(1) 障害の種類

① ハードウェアの障害

端末装置，通信回線，中央処理装置，周辺機器などの障害

② ソフトウェアの障害

③ 運用上の障害

④ システム設計の不備に伴う障害

トラフィック，機器能力の見積りのミスから生ずる障害

(2) 障害の救済処理

障害が検出された場合の処理について説明する。

- 再試行を行う（相当に有効）
- 切換えてあるいは切離して続行可能なら，そうする（2重化の効果）
- 不可能なら，復旧に必要なデータを出力して停止し，修理を受ける。

(3) 復旧処理

障害により停止すると，システムを中断時点の状態に戻して再開する必要がある。

① メッセージの復旧

ロギングに基づいてキュー・ファイルへ復旧する。

② ファイルの復旧

ファイルの破壊更新を行っている場合は，最新のサイクリック・ダンプとファイル更新時のイメージ（ビフォア・イメージ，アフタ・イメージ）から復旧する。これを避けるには2重書き，更新のタイミングを遅らせること，が有効である。

③ メモリ上の各種テーブルの復旧

最新のチェックポイント情報とロギングに基づく復旧を行う。

④ システム・リスタート

障害によるシステム停止後のリスタートの手順について説明を行う。

指導上の留意点

(1) メッセージ管理の体系は，まだ十分に確立されておらず，またオンライン・パッケージなどの提供も進められているが，システム制御機能でもユーザが個々に作成しなければならない場合が多い。したがって，各機能について十分理解させることが望ましい。

(2) システム内におけるメッセージの流れおよびそれに伴って受けるメッセージ管理機能の処理を明確に把握できるように説明する。

(3) オンライン・システムの設計に当っては性能面からの考慮が非常に重要となることに触れる。

(4) ネットワーク・アーキテクチャの諸概念との対応についても触れることが望ましい。

参 考 文 献

〔1〕大野豊「オンライン・リアルタイム・システムの設計」産業図書，1970

〔2〕 J. Martin 著，北原安定訳「リアルタイムーシステムの設計・管理・運用」日本経営出版会，1975

〔3〕 コンピュータ・メーカ各社のメッセージ管理，リアルタイム・パッケージ関係の各種ユーザ・マニュアル

第7章　運用管理および利用環境

キーワード

システム生成（system generation），システムの初期化（system initialization），オペレータ・インタフェース（operator interface），オペレータ言語（operator language），システムの中断と再開（system down, system restart），アカウンティング（accounting），システム生成言語（system generation lauguage）

スタンド・アローン（stand alone），オンライン（on-line），ネットワーク・システム（network system），一括処理（batch processing），分散処理（distributed processing），リモート処理（remote processing），タイムシェアリング・システム（timesharing system），多次元処理システム（multi-dimensional processing system）

目　標

システム運用管理とは，システムを効率的に運用するため，オペレータやシステム運用管理者の作業を支援する機能を集めたものである。本章では，あらかじめ設定されたシステム運用の目標の達成に必要な各種統計情報や，システム管理者による利用法，オペレータとシステムとのインタフェースを支援する機能，システムの運用開始／終了に必要な作業などについてできるだけ幅広い立場から理解させることを目標とする。

さらに，コンピュータの利用環境は，コンピュータを利用する分野の拡大にしたがって，超小型コンピュータや端末の普及などにより，急速に多様化が進展している。本章では，コンピュータを利用をする環境と目的に応じて，ユーザ側からみて最も効率よく，コンピュータおよびその関連システム（ソフト，ハードを含めて）を利用する形態について考察させることを目標とする。

内　容

7.1　運用管理の概要

運用管理は，与えられたシステムの目標をできるだけ低い運転コストで達成することを目的とする。一般に次の機能をもつ。

① システムの稼動開始時／停止時に動作する機能

② 課金管理，システム運用計画に必要なアカウンティング情報の提供

③ オペレータのシステム制御を支援する機能

またコンピュータ応用面の進歩に従って，システム運用コストの節約と省力化のため，さらに夜間のオペレーションの自動化をはかる自動ランニング・システムも開発されるようになった。

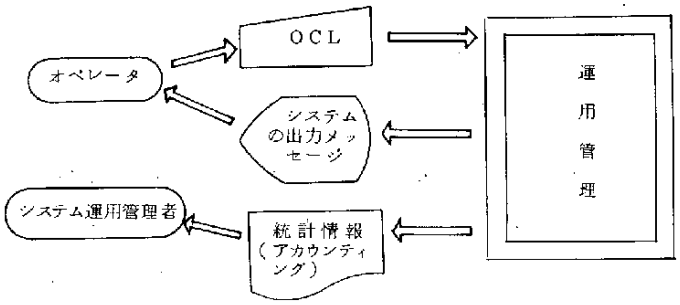


図7－1　運用管理の情報

7.2　システムの生成

システム運用開始前に必要な準備作業をシステム生成の作業という。オペレーティング・システムを主記憶装置に移したり，時間に従って更新が必要なシステム全体の情報の入力などがオペレータの指令によって行われる。

一般に通常のオペレーティング・システムでは，オペレータの使いやすい操作卓とともに，システム初期化機能として，ハードウェアの初期化と，オペレーティング・システムの初期化（ソフトウェアの初期化）が含まれる。

(1)　ハードウェアの初期化

各ハードウェア装置の電源投入後，周辺処理装置が必要とするファームウェアの格納，装置の初期診断の方法，システム状態表（装置の管理表，ファイルと装置アドレスの対応など），の更新，端末と回線の状態表の管理などについて理解させる。

(2)　ソフトウェアの初期化

主としてオペレーティング・システムの必要な更新，制御用記憶のサイズの変更指定，日付と時刻の設定とそれらの使用法，またオンライン・システムなどでは，ファイルと装置の関係が運用中に変る場合には，システム・テーブル・ファイルに登録しておく方法，バッチ処理の場合はシステム・パラメタとして与えることができる。

オンライン処理の場合には，入出力機器（ＭＴ，ＬＰ等）の故障の際のバックアップのための装置切換について装置アドレスをプログラム中で固定にするか可変にするかの二つの方法の長短などについても指摘する。

(3)　オンラインシステムの運用モードの設定

複数のＣＰＵをもつシステムを運転する場合，ＣＰＵと主記憶装置，チャネルの結合方式を必要に応じて設定する。

7.3 オペレータ・インタフェース

オペレータがシステムを効率よく運転するためには，オペレータの判断に参考となる情報が適切に利用できることが重要である。

ジョブの処理状況，負荷状況のデータをみて，必要に応じて任意の時点でジョブを投入したり，スケジューリング方式を変更したりする際の判断の方法についてよく理解させる。

またジョブ進行のため，ファイルのボリュームのマウントなど直接オペレータが実行すべき作業の他に，操作卓からオペレータに与えられる各種の情報についても検討する。

ユーザはシステムに対して，ジョブの動作指示，入力装置の起動停止，ジョブのアボートやジョブ出力の開始や，装置，またはジョブの処理／稼動状態の情報の取り出しなどをＯＣＬ（オペレータ制御言語）のコマンドを用いて指令できる。またプログラムとオペレータ間の通信を制御する機能をもつオペレータ・メッセージ・ハンドラのコマンド／マクロの理解が大切な事を理解させる。

7.4 中断と再開

(1) 故障の情報と中断

システムが正常の運転状態になったあと，何らかの変化の情報はオペレータに判りやすい様にする各種の方法がある。たとえば操作卓上のランプ表示，表示装置による表示，コンソール・タイプライタ，ＬＰなどへの打出しなどがある。またブザーでしらせるシステムもある。

これらの情報は，

① 装置の異常

② 回線，通信制御装置関係の警告／停止／停止解除の情報

③ 運転モードの変更

④ システム・ダウンではない異常

⑤ ファイル中の異常データ

⑥ 入出力キューの情報

⑦ 端末の異常

⑧ 業務の処理状態（受信呼数，ファイル　　能力など）

があり，これらの情報の適切な判断により，中断が必要ならばオペレータはそれを行うコマンドを投入することになるので，十分理解できる様に説明する必要がある。

(2) 運用状態の記録

システムの運転状況の記録は重要であり，このため，ハードウェア・エラーについては，軽度なものでも，多大な故障につながるものもあるので，正確な記録が大切なことをよく説明す

る。同様にデータ・エラー，処理エラーの記録，運用状態の変化の記録，ジョブの記録などを作成し，分析することによってシステム全体の効率的運用に役立てることができる。なるべくデータの取りやすい，また分析しやすい形式を考える必要があることを説明する。

(3) オペレータ，端末ユーザなどからの運用状態チェック

システムの運用状態は既にのべたように各種の方法でシステム側から表示されるが，オペレータ／ユーザ側から積極的に，直接に，システム状態をチェックできることが望ましい。ＴＳＳシステム等では端末からシステムの稼動状況をチェックできるのでこのような機能は効果的運用に役立つ。

(4) システムダウンの種類と再開

システムをダウンさせる場合も異常状態による場合と正常状態でのジョブ終了による場合とでは区別した手続きが行われる。また再開手続きも異なっている。さらにバッチとオンラインでも異なる。とくにオンラインの場合には，直ちにシステム・ダウンするのではなく，まず必要最小限の処置を行って回復させ，あとで原因を究明する等により，できるだけダウンの波及効果を少なくする。とくにハードウェアの一時的障害，プログラム・エラーなどで同種類の障害があまりおこらないことが判明した時には，すぐに回復するか否かの判断はオペレータにとって非常に重要である。

オペレータとしては，

① 障害データを収集する

② 障害対策としての機器の取換，システムの縮少を行うと同時に必要なデータを回復する。

③ ソフトウェアのエラーはすぐには直せないので，障害に関係するソフトやファイル・データを除外してシステムを回復する。すなわちフェイル・ソフト（fail soft）による回復を行う。同時にファイルの正常化，ファイルと接続している端末に対する後処理の問題についても注意する必要がある。

7.5 アカウンティング

ジョブやジョブ・ステップが実行時に使用した資源の種類，量，使用時間などの記録情報で，システム運用管理者が課金処理，運用計画を行う上で必要な情報であり，これを作成する機能をアカウンティング機能という。通常ログ・ファイルとジョブ・オカランス・レポート・ファイルに記録されている。これらの情報の有効利用についてはシステム運営管理者にとっても，またシステム設計開発グループにとってもシステムの改善上重要なことを理解させる。

7.6 システム生成

システムの対象とする応用および利用形態によって，最適なシステムを構成することは，効率的な運用管理の面からも非常に重要な問題である。このようなシステムの構成を行い，ユーザ用の新しいオペレーティング・システムを作成することをシステム生成という。

システム生成（SG, system generation）を行うためのツールとして，システム生成言語SGL（System Generation Language）を用意しているシステムが多い。

システムの更新はシステムの一部を変更するものである。SGLの機能としては，次のものがある。

① システム生成の実行開始

② システム生成文　システムの基本構成を作る。

③ ハードウェア構成文　ハードウェアを指定する。

④ ソフトウェア構成文　システムプログラム，サービスプログラム，共同プログラムの指定

生成はファームウェア・ファイルの生成と，既存のオペレーティング・システムを用いて，新しいシステムを生成する。

7.7 処理形態

ユーザがコンピュータ・システムを利用する場合，その目的，処理すべき情報の量，情報の加工度，企業規模，情報処理にかけることとのできる予算，アプリケーションの種類（業種，業務の種類），ユーザ組織の構成やユーザ・サイトの特徴（広域性，集中型か，分散構造型の種類），必要とする情報の許容処理時間の緩急など，使用ユーザの組織内のレベルなどの各種の制約や条件によって，処理形態はかなり異ったものになる。この節は，現在実用されている処理形態の種類について，できるだけ体系的に説明し，新しく応用システムを開発利用する場合の基本的な設計／評価基準をもてる様に理解させることが望ましい。

処理形態を考える場合，コンピュータの結合方式による分類として，**スタンド・アローン**（独立型），**オンライン・システム**，**ネットワーク・システム**（network system）の構成的分類について説明する。これと同時に情報の処理機能面からみた分類として，**集中処理型**，**分散処理型**の特徴と適用領域等についても説明するとよい。また実際のジョブの処理方式からみて，**一括処理**，**リモート・バッチ処理**，**会話型システム**（TSS），**トランザクション処理**などについて説明し理解させる。

処理形態に大きな影響を与える要素として人間と機械（コンピュータ）とのインタフェースの問題があり，このため端末の種類，機能についてふれること，またユーザが使用する言語の問題，

特にエンドユーザ言語，データベース／データ・コミュニケーションとその利用との関係などについてもふれることが望ましい。

7.8 一括処理とリモート・バッチ

ユーザのジョブを処理する場合，一括処理はコンピュータ・システムの有効利用の面から各ユーザのプログラムを一括してシステムに投入し，多重プログラミング方式によって，実行する方式でもっとも一般的な方式である。

この際，オペレーティング・システムが持っているスケジューリング・アルゴリズムによって処理コストが全体として最小化される様に設けられている。一括処理の場合でも，ユーザ側のジョブ処理結果を必要とする緊急度によって，ユーザは優先的実施を要求するためのクラスをつけてジョブを提出できる。一般にオペレーティング・システムはこのような情報を加味して最適スケジューリングを行っていくアルゴリズムを持っている。

リモート・バッチ処理は，本質的には一括処理と変らないが，ジョブ処理を行うセンタ・コンピュータからはなれた位置にあり，センタ・コンピュータに接続された，小型コンピュータまたはリモート・ジョブ端末／ステーションから一括ジョブのカードを投入して，センタで通常のジョブと同じ様に処理を行い，その結果を，離れた距離におかれたジョブ投入ステーション(端末)に送り返して，そこに設置されたＬＰから出力する方式である。

この処理を実現するためには，一括処理の全機能の他に，センタ・システムの前置プロセッサ（FNP）とのインタフェース機能を付加すればよい。

この節では，一括処理とリモート・バッチ処理の効率的使い方，および使用するオペレーティング・システムのジョブ処理アルゴリズムについても説明することが望ましい。

7.9 会話型システム

会話型システムは，端末の　コンピュータと対話をしながら，プログラムの作成や，ジョブの実行，データベースの検索，更新などを行っていくシステムである。タイムシェアリング・システム（ＴＳＳ）とよばれることが多い。

ＴＳＳは遠隔地または組織の構内におかれた多数の端末から，ユーザからみるとあたかもセンタ・コンピュータを占有している様に利用できるシステムであり，このことは，高速コンピュータの処理時間を細かく分割して，各利用者のジョブに少しずつ時間を与えることによって多くのユーザのジョブがあたかも同時に処理されているようにみえるように実行される。

この節では，ＴＳＳの特徴としての使いやすい言語インタフェースの種類と機能，コマンド体系など，また会話的なプログラムやデータの編集機能，ファイル・システムの構造と利用方法な

どについて説明する。

さらに応用システムをＴＳＳで実現する場合の端末数，端末能力，回線スピード／容量，FNP，センタ・システムの規模や機能，データベースの種類と機能，会話型言語の種類能力などを総合的に検討する必要があり，そのための設計と評価の重要性について理解させることが望ましい。

7.10　オンライン・システム

オンライン・システムは，一般に遠隔地にあるデータを，通信回線を通じて，センタにある処理用のコンピュータに，送り，処理を行って，送り返す方式，すなわち遠隔地からデータの入出力を行うことのできる方式である。この定義によれば，あとで扱っているリモート・バッチ処理と，オンライン・リアルタイム方式とがこの分類に入る。

オンライン・リアルタイム処理は，データが発生する毎に，処理コンピュータに送りこみ，直ちに必要な処理を行うシステムを実現する方式であり，このようなシステムをオンライン・リアルタイム・システムとよぶ。

この節ではオンライン・リアルタイム処理の応用例をあげて説明し，またこの方式を構成する基本的技術はコンピュータによる情報処理技術とデータの送受信に使用する通信回線に関係するデータ伝送技術の二つであるが，ここでは，まずデータ伝送技術について必要な事項を理解させる。すなわち，通信回線について，単信，半２重，全２重などの種類と性質について簡単に説明する。また通信回線上のデータ（ビット）の伝送方式について説明し，関連する通信機器装置（モデムなど）の理解が必要なことを説明する。さらに，通信を行っていく上でのプロトコルの問題とレベル，性質について理解させる。

またオンライン・システムの開発には，プログラムの作成，すなわちオンライン・プログラムとよばれるものの開発と検証の問題がありこの面についても，時間を割くことが特に重要な意味をもっている。このためオンライン・プログラムの基本的性質として，リアルタイム処理，待機処理，即時処理，同時処理，でプログラム作成上注意しなければならない各種の問題点について整理し，説明することが望ましい。また回線に対する入出力処理や，エラー処理の方法についての注意が必要である。

オンライン・プログラムは，一般に，回線制御部分，ファイルの入出力処理部分，メッセージ処理部分，同時処理を行う部分，システム全体を監視する機能をもつ端末処理部分等から構成されるので，これらの各部分についての性質や注意事項について理解させる。

さらに効率のよいオンライン・プログラムを作成し，また保守作業をなるべく効率的に行えるようにするための考え方として，モジュール構造の適用，データとプログラムの分離，必要部分の最適化，採用する各種の処理方式などについて，プログラムの目的からみた時の優劣の判断す

なわち評価方式などの考え方などについてふれることができれば望ましい。

7.11 多次元処理

最近のコンピュータ・システムでは，各種の処理方式を同時に利用できるようにした，複合処理形態のオペレーティング・システムを提供するようにしたものが現われている。

このようなオペレーティング・システムは多次元オペレーティング・システムとよばれており，たとえば，端末を通じて中央装置とのインタフェースをとりながら，リモート・バッチ，会話型リモート・バッチ，トランザクション処理（オンライン処理），ＴＳＳおよびメッセージ交換（以上の五つはリモート処理であり）さらに通常の一括処理を加えて六つの処理を実行できるシステムもある。

この様なシステムになると，ある程度大型のコンピュータを基本とすることが必要になってくるが，たくさんのユーザの仕事の性質に応じて，また処理時間の緩急の必要度に応じて，最適の処理形態を選択して利用できる点で，トータルなシステム利用効率化ユーザからみた利用効率を最大限に発揮させることができるといわれる。

この様なオペレーティング・システムの利用法の説明を行うことが必要であり，さらにネットワーク処理，分散処理システムなどについても，その特長を説明することが望ましい。

指導上の留意点

(1) 運用管理の問題を理解するためには，本章における講義の他に，実際に運用されているシステムの運用状況の観察と，実際の運用を経験してみることは，全体的な理解に大きく役立つので，運用管理の実習を行ってみることもよいであろう。

(2) 運用管理の最終目標の一つとしては，システムの自動運転があり，演習問題の一つとして，その様な環境において必要な条件や，どのようなことを準備すべきかなどについて考えさせるのも望ましい。

(3) 運用管理をできるだけ効率よく行うためには，オペレータのシステムとのインタフェースを使いやすいものにすることも重要であり，この面について，種々の検討をさせることも理解を深めることに役立つであろう。

(4) 開発および利用すべきユーザ・システムの性格，要求に応じて最適の効果を発揮できる様なシステムに仕上げるのには，どのような利用形態を採用すべきかという見方に立って，各種の問題，技術を理解，修得させる様に心がけることが望ましい。

(5) システムの利用環境に応じた利用形態を使用する場合，システム評価の立場から，第１０章におけるシステム性能評価尺度や技術と関連をもって指導することが望ましい。

(6) 作成されるユーザ・システムにとって，ユーザからみた使い易さの点も今後一層重要性をますので，各利用環境におけるユーザ・インタフェースの面からの評価も重要なことを指摘することが望ましい。

参考文献

〔1〕「分散システムに関する調査」電子協（52－C－324），1977

〔2〕岩城三郎「オンライン・プログラム設計入門」日刊工業新聞社，1973

〔3〕藤野喜一他「計算機システム基礎論」共立出版，1973

〔4〕「計算機システムの最近の動向」電子協（49－C－277），1974

〔5〕「データベース・システムに関する調査」電子協（52－C－276），1977

〔6〕大野豊「オンライン・システムのソフトウェア」産業図書，1977

〔7〕各社のオペレーティング・システムのユーザーマニュアル（運用管理，利用環境）

第8章　多重プロセッサ・システム

キーワード

疎結合システム（loosely－coupled system），密結合システム（tightly－coupled system），対称形システム（symmetric system or homogeneous system），非対称形システム（asymmetric system or heterogeneous system），共通バス（common bus），マルチポート（multiport），クロスバー（crossbar），アクセスの競合（access confliction），メールボックス（mailbox），システム再構成（system reconfiguration）

目　標

多重プロセッサ・システムは最近のコンピュータ・システムにおいて一般的になってきている。その目的を正しく理解し，システム設計における種々の問題を理解することによって，利用目的に合ったシステム構成を決定しうることを目標とする。

多重プロセッサ・システムには種々の構成があるが，ここでは最も中心的な主記憶を共有する密結合の多重プロセッサ・システムを中心に理解させる。密結合以外にも重要なものがあるが，あまり範囲を広げて，内容が浅くなるよりも，範囲を狭くし，深い知識を与える方が望ましい。

内　容

8.1　多重プロセッサ・システムの目的

多重プロセッサ・システムは複数のプロセッサによって構成され，一つの有機体として動作するシステムと定義される。多重プロセッサ・システムの定義と，その目的，歴史的背景について説明する。

(1) 目　的

多重プロセッサ・システムの目的はシステムにより異なるが，一般に次のような目的があげられる。

① システム性能の向上

- 並列処理による処理速度の向上
- 負荷の分散によるスループットの向上
- 機能の分散による性能向上
- 専門プロセッサ化

② 信頼性の向上

- フェイル・ソフト・システム
- フェイル・セーフ・システム

③ 資源の共用

- プログラム・データの共用
- ハードウェア資源の共用

通常，これらのうちの複数の項目がシステムの目的として選ばれることを理解させる。

(2) 歴史的背景

多重プロセッサ・システムの歴史的流れを年表を用いて説明し，最近の多くのコンピュータ・システムでこの方式が採用されていることを理解させる。主な多重プロセッサ・システムを表8－1に示す。

8.2 システム構成

多重プロセッサ・システムの構成は次のように分類される。

① 疎結合（loosely coupled）システム

表8－1 主な多重プロセッサ・システム

年	システム名	メーカ	備考
1958	SAGE システム (AN/FSQ-31,32)	IBM	デュプレックス・システム 軍用
1962	D-825	バロース	プロセッサ 4台（対称形） メモリ 16モジュール
1963	B-5000	バロース	プロセッサ 2台迄 OSはMCP
1963	704×/709×	IBM	チャネル結合システム
1964	CDC 6600	CDC	並列演算ユニット(10) 周辺プロセッサ(10) 非対称形
1965	GE 645	GE	MULTICS システム
1965	UNIVAC 1108	ユニバック	CP 3＋IOP 2
1966	360/67	IBM	CP 2台 TSS用
1969	360/65 MP	IBM	360/65の多重プロセッサ（2台）
1971	H6000シリーズ	ハネウェル	
1971	System 1055,1077	DEC	
1972	ILLIAC Ⅳ	イリノイ大	64台のプロセッサ・エレメントアレイプロセッサ
1974	370/158：168MP	IBM	

複数のプロセッサが共通にアクセスしうるファイル装置やチャネル結合装置を介して結合されたシステムである。

図８－１に示すように，計算を主として行なうプロセッサと入出力処理を中心に行なうプロセッサで構成することが多い。

この構成は各プロセッサを独立のコンピュータ・システムとして動作させることができるため，多重コンピュータ・システムと呼ばれることもある。

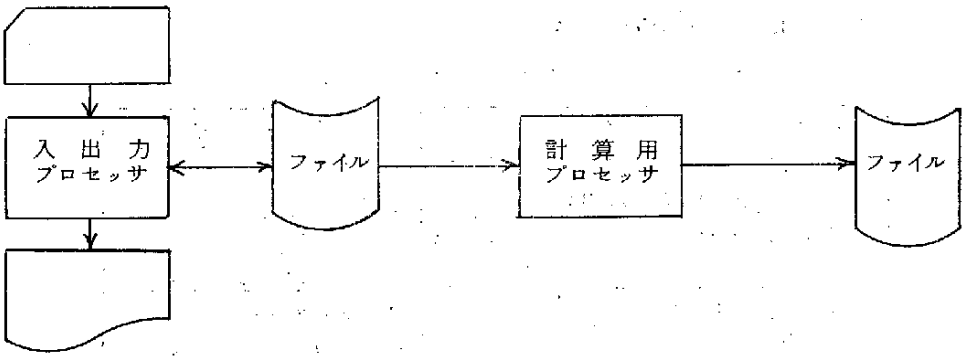
② 密結合（tightly-coupled）システム

複数のプロセッサが主記憶装置を共有して結合されたもので，狭義の多重プロセッサ・システムは密結合システムをいう。

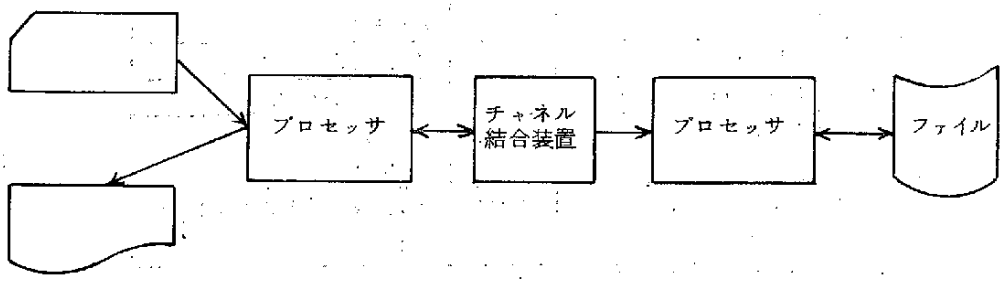
図８－２のような図を用いて，密結合システムの構成を説明する。

③ コンピュータ・ネットワーク

複数のコンピュータ・システムを通信回線を介して結合したコンピュータ・ネットワークも広義の多重プロセッサ・システムである。ここではあまり立入った説明は行わず，一応このカテゴリに入ることを説明する程度で良い。



(a) ファイル結合システム



(b) チャネル結合システム

図８－１ 疎結合システム

④ 並列処理システム

ILLIAC Ⅳで代表されるアレイ・プロセッサ・システムやCDC STAR-100のようなパイプライン制御方式のコンピュータ・システムも多重プロセッサ・システムの一種である。これらは並列処理による処理速度の向上を強く狙うものであり，単一プロセッサの実現形態の一種とみなしうる。

これらのシステム構成と目的との関係を説明し，①，②が広く使用される構成であることを理解させる。

構成要素となるプロセッサの観点からは次のような分類があることを説明する。

- 対称形（均質形）システム
- 非対称形（異質形）システム

その他に最近では安価なミニコンピュータやマイクロコンピュータを多数用いた，

- マルチ・ミニコンピュータ・システム
- マルチ・マイクロコンピュータ・システム

も注目されている。これらはまだ研究段階のものが多いが，代表的なシステム（たとえばカーネギー・メロン大学のC. mmpシステム，日本電気中研のMICSシステム，電子技術総合研究所のACEシステムなど）についてふれる。

8.3 結合方式

密結合システムにおいては複数のプロセッサ，主記憶装置，入出力制御装置や入出力装置の結合方式が問題となる。これらの結合方式，その利点・欠点について説明する。

① 共通バス方式

- 単一バス方式
- 複数バス方式

② マルチポート方式

③ クロスバー方式

これらの結合方式については文献〔1〕に詳しい。図8－3に各結合方式を示す。

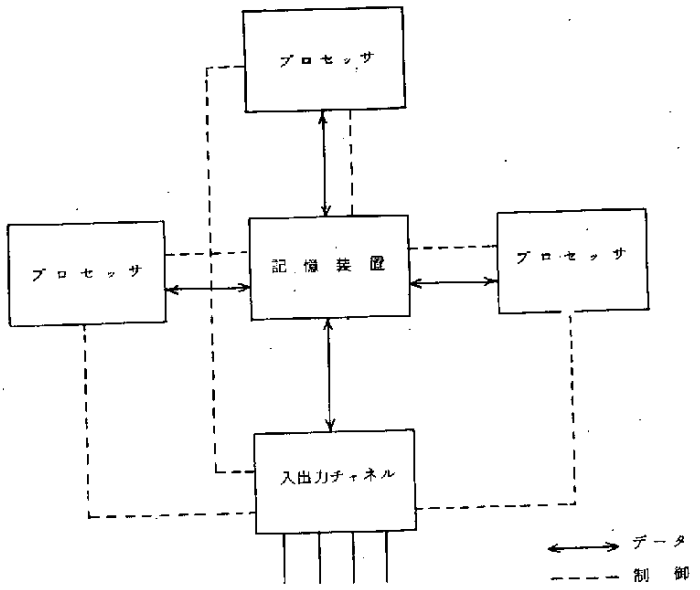


図８－２　密結合システム

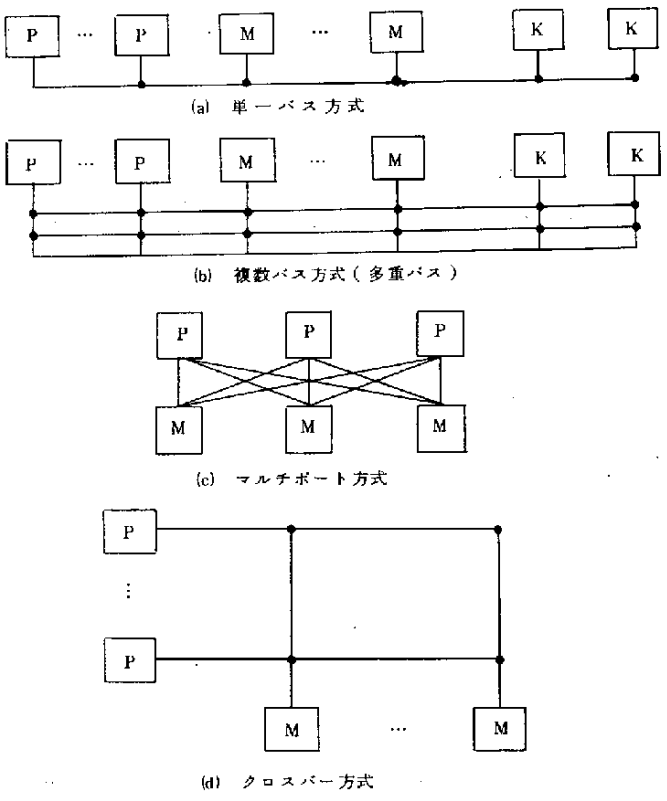


図８－３　結合方式

8.4 多重プロセッサ制御方式

多重プロセッサ・システムを構成するためには特別なハードウェア／ソフトウェア機構が必要である。ここでは多重プロセッサ・システム個有の問題とそれを解決するための機構を理解させる。

単一プロセッサ・システムでは一つのオペレーティング・システムがただ一つのプロセッサの上で動作するが，多重プロセッサ・システムではいろいろな制御方式がある。システム制御機能とそれを実行するプロセッサとの関係から，次のような制御方式があることを説明する。

(1) マスタ・スレーブ方式

システム制御機能は特定のプロセッサ（マスタ）が主導権を持ち，他のプロセッサ（スレーブ）はマスタの指示のもとに動作する。制御プログラムは比較的単純である。次のような特徴がある。

- 常に特定のプロセッサ上でシステム制御が行われる。
- 必ずしもリエントラント形式のプログラムでなくても良い。
- システム・テーブル類のロックが不要
- 特定目的のシステムや非対称システム構成に適す。
- ハードウェアとソフトウェアは比較的単純である。
- マスタとなっているプロセッサのダウンはシステム・ダウンにつながる。
- 融通性に乏しい。
- マスタの性能によってシステム性能が左右される。

このタイプの例としては，IBM S/360 TSS（360/67），B5500 MCPなどがある

(2) 各プロセッサごとの制御

各プロセッサが必要に応じて自分自身の制御を行なう。場合によっては異なるプロセッサが異なったオペレーティング・システムを持つ，次のような特徴がある。

- システム制御機能は各々のプロセッサの必要に応じて各自が処理する。
- システム・プログラムが共用されるときはリエントラント形式であることが必要。
- 各プロセッサは自分のテーブルを持つ。
- いずれかのプロセッサがダウンしても他のプロセッサは動作可能（フェイル・ソフト）。
- 入出力装置は各プロセッサごとに持つ。
- 入出力装置の再構成は手動。

このようなシステムは複数のコンピュータ・システムが並行して動作しているに近く，多重プロセッサ・システムとしての結合度は非常に弱い。初期の多重プロセッサ・システムであるSAGEシステムはこのタイプに属する。

(3) 分散制御方式

対称形のシステムではプロセッサは全てハードウェア的には同等である。このようなシステムではシステム制御機能はどのプロセッサでも実行できる方式がとれる。

- 任意のプロセッサがシステム制御機能を実行しうる
- 負荷均衡が容易
- 優先度などによるサービス要求の順序づけが必要
- リエントラント形式のコーディングが必要
- システム・テーブル類のロックが必要
- 信頼性の向上（しとやかな縮退(graceful degradation)）
- 資源の有効利用が可能

このカテゴリーに属するものは比較的最近のシステムが多く，IBMのMVS，カーネギー・メロン大のC. mmpシステムなどがある。

8.5 多重プロセッサ制御機構

多重プロセッサ・システムのオペレーティング・システムは単一プロセッサで必要な機能の他に次のような制御機構が必要であることを理解させる。

(1) 記憶管理

共有の記憶装置を持つ多重プロセッサ・システムでは次のような問題がある。

① 記憶装置におけるアクセスの競合

複数のプロセッサからのアクセスが重なるアクセス競合はシステム性能を低下させる。これの解決策としては次のようなものがある。

- 記憶装置をいくつかの独立したモジュールに分割し，競合を少くする。
- インタリーブ方式
- 共有記憶の他にプロセッサ専用記憶を持つ（ローカル・メモリ）
- 高速バッファ装置（キャッシュ）をプロセッサごとに持つ

ローカル・メモリやキャッシュは共有記憶の写しであるため，共有，専用記憶装置間のデータの食い違いをさける工夫が必要である。

② 共用データのロック

あるプロセッサが共有記憶のデータを処理中に他のプロセッサからの参照を禁止するロック機構が必要である。次のような機構を説明する。

- テスト・アンド・セット命令
- IBM 370のコンペア・アンド・スワップ命令によるロック機構

・ Dijkstra のセマフォ機構（又はP，V命令）

このような機構はシステム制御のためばかりでなく，ユーザ間の排他制御にも必要である。

(2) スケジューリング

多重プロセッサ・システムではジョブのスケジューリングによって性能に差がでることを図8－4のような例を用いて理解させる。

実際にスケジューリングやディスパッチングの最適化を行なうためには，将来の状況を見通すことが必要である。現在のオペレーティング・システムはスケジューリングの最適化はまだ不完全である。

(3) プロセッサ間の通信

プロセッサ間の通信にはハードウェアとソフトウェアの両方が用いられる。

・ 割込機構

・ メールボックス

通常，通信すべき情報を共通の記憶領域（メールボックス）に用意し，入出力割込と同様に，他のプロセッサへ割込む方式が使われる。

プロセス間の同期メカニズムであるセマフォ機構を利用する方法もある。

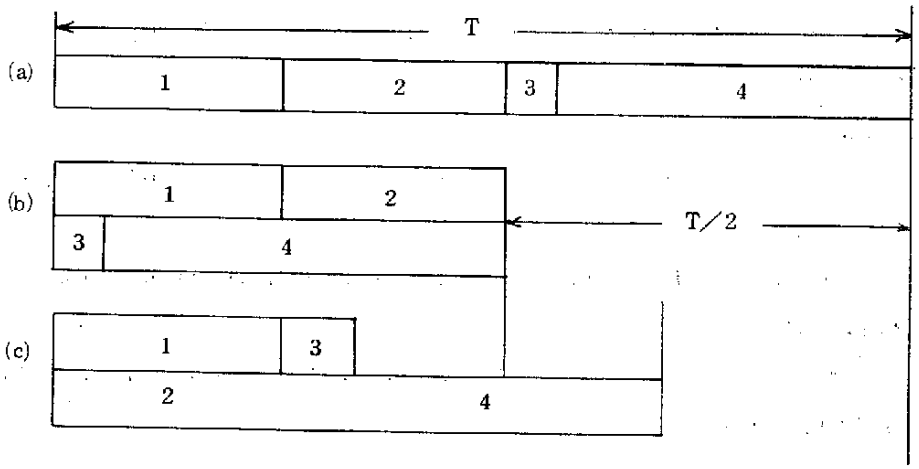


図8－4 スケジューリングの効果

(4) 障害管理

多重プロセッサ・システムではあるプロセッサが障害を起こしても全システムが止まらないようにすることが重要である。そのために次のような機構が使われる。

・ 相互診断

・ 障害ユニットの切離し（手動，プログラム制御）

・ システム再構成（手動，プログラム）

システムに障害が発生しても，できるだけ回復できるようにするための手法として，次のようなものがある。

- システム・ファイルの多重化
- システム資源の変動に対応しうる資源管理方式
- 障害検出機能（自己検出，他からの検出）
- 結合路の冗長構成

指導上の留意点

(1) 多重プロセッサ構成を採用したシステムは数多くあるが，その目的は様々である。表8－1のような代表的システムについて，その主な目的を理解させることが望ましい。

(2) システム構成は疎結合，密結合システムを中心に説明し，他の構成は簡単な説明にとどめてよい。

(3) 結合方式ではバスの競合の問題，システム構成の柔軟性，結合のために要する遅延等の観点から利点，欠点を述べる。

(4) 多重プロセッサの制御機構は複数のプロセッサが矛盾なく処理を行うための重要な機構である。そのような制御機構の必要性とその動作について十分理解させることが必要である。

参考文献

〔1〕「マイクロプロセッサ　設計のための予備的検討」PIPS－R－№3 電子技術総合研究所，1973

〔2〕P. H. Enslow Jr.（Ed），"Multiprocessors and parallel processing", John Wiley & Sons，1974

〔3〕日本ユニバック総合研究所編「共立総合コンピュータ辞典」共立出版，pp.351－352，756－761，1976

第9章　システムの信頼性および情報の保護

キーワード

信頼性（reliability），ＭＴＢＦ（mean time between failure），アベイラビリティ（availability），信頼度機能ダイヤグラム（reliability function diagram），入力データのチェック（input data check），ファイルの障害（file failure），ＥＣＣ（error correcting cord），巡回符号（cyclic redunduney code），プログラム例外（program exception），回復（recovery），故障診断（fault diagnosis），チェックポイント（check point），故障辞書（fault dictionary），ＦＬＴ（fault locating technique），オンライン診断（online diagnosis），リモート・メンテナンス（remote maintenance），ソフトウェアの信頼性（software reliability），保護（protection），記憶保護（storage protection），アクセス制御（access control），情報のプライバシー（information privacy），情報の機密性（information confidentiality），情報の安全性（information security），リング保護（ring protection）　資格（capability），ドメイン（domain）

目　標

オンライン・システムの普及と大型化，高度化，データ・ベースの発展，コンピュータの社会生活，企業活動との密着の方向により，コンピュータ・システムの信頼性に対する要求は非常に強いものとなってきている。

ここでは，システム設計に当って心得ておくべき信頼性に関する知識，高信頼化の技術を身につけさせるとともに，情報の保護に対する一般的な知識を習得させることを目標とする。

システム信頼性の計算方法，冗長構成の方法，障害回復の手法だけでなく，情報の信頼性を高める方法，ハードウェア内部で行われている障害の検出と訂正の技術，現在の故障診断，保守の技術の水準についても理解させる。さらに，ハードウェアと同様に，ソフトウェアの信頼性がシステムにとって重要であることを認識させ，その向上策についての方向づけを与える。

また，コンピュータ・システムにおける技術的な問題に範囲をしぼり，一般の汎用計算機が備えている保護機構，および，応用システムを構築する上で注意すべき問題点，あるいは，その解決方法などについて理解させる。

内　容

9.1　高信頼化技術の基礎

(1)　高信頼化技術の分類

コンピュータ・システムの信頼性を高めるために用いられる技術は次のように分類される。これらについて簡単に説明し，以下の内容への手引とする。

①　素子の高信頼化

a　高信頼性素子の開発

- 故障の物理的究明とそれに基づく改善
- 大規模集積技術による信頼度向上

b　環境条件の改善

- 温度，湿度，振動，衝撃など物理的条件の改善
- アンダーレーティング（定格以下で使う）

②　システムとしての高信頼化

a　システムとしての信頼性向上

- 冗長構成
- 修理期間の短縮
- システム回復技術の開発
- ソフトウェアの信頼性の向上

b　情報の信頼性の向上

- 情報への冗長性の付加
- 情報の保護

この章では②について理解させる。

(2)　信頼性の評価尺度

信頼性を表わす次の指標について説明する。

①　信頼度（reliability）

②　ＭＴＢＦ（Mean Time Between Failure）

③　ＭＴＴＲ（Mean Time To Repair）

④　アベイラビリティ（availability）

また，ＭＴＢＦは平均値であって，偶発故障期においてはたとえば信頼度９０％で予測すると故障間隔は１桁下ることを理解させる。

(3)　信頼性の計算

システムの信頼度機能ダイヤグラムの書き方について例を上げて説明し，それに基づく信頼性の計算方法を説明する。計算の基礎になるのは以下の公式である。各構成装置のＭＴＢＦをU_i，ＭＴＴＲをD_i，アベイラビリティをA_i（$=D_i/(U_i+D_i)$）とする。

① 直列系の場合

アベイラビリティ$=a_1 \times a_2 \times \cdots\cdots \times a_n$

$$MTBF \fallingdotseq 1/\left(\frac{1}{U_1}+\frac{1}{U_2}+\cdots\cdots+\frac{1}{U_n}\right)$$

② 並列系の場合

アベイラビリティ$=1-(1-a_1)\times\cdots\cdots\times(1-a_n)$ 同一装置がn台あり，少なくともr台稼動していれば故障でないシステムでは，

$$MTBF \fallingdotseq \frac{(n-r)!\,(r-1)!}{n!}\cdot\frac{U^{n-r+1}}{D^{n-r}}$$

(4) 信頼性を考慮したシステム構成

障害に備えた次のシステム構成の特徴について説明する。

① フォールバック・システム

② バックアップ・システム

③ デュプレックス・システム

④ デュアル・システム

⑤ 多重プロセッサ・システム

(5) 障害処理の手順

次の各ステップについて簡単に説明する。

① 障害の検出

② 障害救済処理

続行，機能低下状態で続行，停止のいずれかになる。続行以外の場合は次のステップを踏む。

③ 故障診断，修理

④ システムの回復

9.2 情報の信頼性

この節では主に人手およびソフトウェアで行う情報の高信頼化の方法について紹介する。

(1) 入力データのチェック

① チェック・ディジット・チェック

② レンジ・チェック

③ パリディティ・チェック

④ バランス・チェック

⑤ フォーマット・チェック

⑥ シーケンス・ナンバー・チェック

⑦ ハッシュ・トータル・チェック

(2) メッセージの紛失防止

① 通番の付加

② メッセージのロギング

(3) ファイルの障害対策

① ファイルの二重化

重要なファイルについてのみ行う。

② 仮空間方式

更新は仮ファイルに対して行い，処理単位（１メッセージの処理など）の終了後実ファイルの更新を行う。

③ チェック・ポイント方式

一定時間ごとにシステム回復に必要な基本情報をダンプする。

④ インクリメンタル・ダンプ方式

ビフォア・イメージ，アフター・イメージ

⑤ トータルダンプ方式

これらは独立した方法であるため，必要に応じて組合せて使うことができる。

(4) 情報の保護

保護の問題は広い意味の信頼性上重要な位置を占める。これらについては，後述する。

9.3 障害の検出と訂正

ハードウェア，ソフトウェアの障害を自動的に検出し，場合によっては自動的に訂正するための方法について紹介する。

(1) データの誤り検出と訂正

① パリティ・チェック

今日でも一番よく使用されている。適用対象としては，主記憶，磁気テープ，通信回線，加算回路などがある。

② ハミング・コード

最近は主記憶に多く採用されている。単一誤りの訂正と二重誤りの検出が可能なＥＣＣ（Error Correcting Code）である。

③　巡回符号（ＣＲＣ*）

ランダム誤りの他にバースト誤りの検出と訂正が可能で，磁気ディスク，磁気テープ，通信回線などに適用されている。

④　二重照合チェック

磁気テープにおける書込み後読取みの他，カード機器，紙テープ機器などで採用されている。

(2)　演算，制御の誤り検出

①　多重化法

②　nCmコード法

論理回路の出力m本中に常にn本の1があるようにする。

③　パリティ・チェック

④　タイムアウト・チェック

また，一般に誤りの訂正には自動再試行の手段が用いられる。

(3)　ソフトウェア・エラーのハードウェアによる検出（プログラム例外）

- 不正命令コード
- 記憶保護侵害
- オーバフロー，アンダフロー

など

(4)　ソフトウェアによる障害検出

前節の(1)，(2)で述べたようなデータのチェックの他に，次のようなシステム・チェックが可能である。

- デュアル・システムにおける結果の照合
- デュプレックス，多重プロセッサ・システムにおける相互監視
- 監視タイマの利用

9.4　システム障害の救済と回復

(1)　障害の救済処理

前節で扱った手段によって検出され，ハードウェアによる自動訂正が不可能であった障害の処理はソフトウェアに依頼される。ここで行われる処理について説明する。

(注) *　Cyclic Redundancy Code

① 再試行（何回か行う）

周辺装置，通信系に対して相当に有効

② 障害の切分け

③ 障害部分の切離し，再構成

- 該当ジョブ，メッセージのみの廃棄
- 冗長構成を生かす
- デグレード・モードへの移行

④ 再構成が不可能な場合は停止

(2) システムの回復

再構成を行って再開する場合，停止状態における修理の終了に伴う再開の場合，デグレード・モードからの復帰の場合に，システムの回復が必要となる。以下の回復手法について説明する。

① メッセージの回復

戻すべき時点におけるシステム内のメッセージの状態をロギングに基づいて再現する。端末に再投入を要求することもできる。

② 主記憶上の各種テーブルの回復

最新のチェックポイント・データに基づいて回復し，それ以後の処理を再現して停止の直前の状態に戻す。

③ ファイルの回復

- トータル・ダンプとインクリメンタル・ダンプによる回復。時間がかかる。
- 応急復旧　障害発生時に仕掛り中であったトランザクションの更新対象であったファイルは放棄し，それ以外に対するサービスは再開する。

④ バッチ・システムにおける回復

- ジョブ・ステップ・リスタート
- チェックポイント・リスタート

について，ファイルの回復との関係も含めて説明する。

9.5 診断と保守

障害が検出されると，故障個所を決定し修理（一般にはプリント板の取替え）を行う。また，定期的な保守によって障害の予防を行うが，いずれの場面でも故障診断は中心的な役割を果す。

(1) 故障診断

故障診断とは，一連の入力データを加え対応する出力データを観測し，この結果を利用して

故障位置を指摘することをいう。

① 診断方式

代表的な診断方式として,

- ＦＬＴ(Fault Locating Technique)
- マイクロ診断方式
- ソフトウェア診断方式

について，各々の長所，精度，適用範囲を含めて簡単に説明する。いずれの方式においても，予め診断用データを用意しておきそれに対する出力結果を故障辞書（出力の誤りの集合と故障点の対応表）と照合して故障点を決定する。

② 試験データの作成

- 人手による作成（機能的／非機能的）
- 自動作成　論理回路図に基づき，Ｄ－アルゴリズム，ランダム法などを用いて発生させる。

③ 故障辞書の作成

各々の試験データによって検出しうる故障を知るためには，ソフトウェアまたはハードウェアによるシミュレーションが用いられる。この結果を分類すれば故障辞書が得られる。

(2) 保　守

次の事項について概要を説明する。

- 定期保守
- システム初期化時における予期診断
- オンライン診断　ジョブ・プログラムの実行と並行する周辺装置のテスト
- リモート・メンテナンス　保守センターから通信回線を介して保守

9.6 ソフトウェアの信頼性

ハードウェアの障害はいわば予期されうるものであり，備えをしておくことは可能である。一般にソフトウェアは障害を内包して動作しており，これが顕在化した場合，特にオペレーティング・システム，オンライン・システムにおいては重大な影響を及ぼす。ここではソフトウェアの高信頼化手法を概観する。

(1) ソフトウェア自体の高信頼化

- 文書化とその機械化，自動化
- モジュール化
- 高水準言語の使用

- プログラミングの標準化，構造化
- 開発体制の整備（チーフ・プログラマ・チーム制など）
- レビューの徹底（ウォークスルー）
- ボトムアップ・テスト
- トップダウン・テスト

など，いわゆるソフトウェア工学的手法が有効であることを説明する。

(2) アベイラビリティの向上

実行時に発生するエラー対策について簡単に説明する。基本は障害の影響をシステム全体に波及させず，システム・ダウンを避けることである。

- 記憶保護機能，リング保護の活用
- 該当プロセス，該当メッセージのみの廃棄。システムは続行。
- 機能障害回復モジュールの組込み

9.7 情報の保護に対する基本理念

(1) 情報の保護の必要性

コンピュータの利用技術の発展に伴って，コンピュータに貯えられる情報は，より高価なものになり，利用者層も増大してきた。このような環境では，コンピュータ・システムにおける情報の保護が大きな問題になる。ここでは，保護の必要性を説明する。

(2) 情報の漏洩

PetersonとTurnの分析〔7〕を参照しながらコンピュータ・システムにおいて，情報の秘密性を犯す要因について説明するとよい。

(3) 情報の保護の限界

コンピュータ・システムにおける情報の保護は，技術的な面，あるいは，運用面からみても限界がある。ここでは，その理由を述べ，最終的には，それにかける費用と，効果とのバランスで決定されることを説明する。

(4) 情報の保護に関する用語

ここでは，情報の保護に関する一般的な用語〔6〕の説明を行う。

- 情報のプライバシー（information privacy）
- 情報の機密性（information confidentiality）
- 情報のセキュリティ（information security）
- 保護（protection）
- 保全性（integrity）

9.8 基本的な情報の保護方式

ここでは，コンピュータ・システムで取り扱われる基本的な情報の保護方式を説明する。

(1) 情報の暗号化[1]

傍受や不当なダンプにより情報がもれるのを防ぐ方法として暗号化がある。暗号化，復元化アルゴリズムは，ソフトウェアによる方法とハードウェアによる方法がある。

(2) 資格検査[1]

情報への不正なアクセスを防ぐために，アクセス権をチェックすることが行われる。この方式では保護のためのオーバヘッドが少い。アクセス権チェックは次のような形で行われる。

- パスワードによる検査
- 利用者の権限レベルによる検査
- プログラムの権限レベルによる検査

(3) 情報の保護モデル

Lampson[11]，Graham[8]等は，コンピュータ・システム内部において，複数ユーザ間の干渉によって生じる矛盾を回避するための標準化した保護モデルを提唱している。このモデルを利用すれば，現在実在する保護機構，あるいは，提案されている保護機構の大部分は説明できる。ここでは，この保護モデルを紹介し，保護機構に対する基本的な概念を理解させる。

(a) 保護モデルの狙い

- ソフトウェア・システムが真に対象としている情報対象を完全に保護すること。
- ソフトウェア・システムの開発者が最も理解し易く，かつ柔軟性をもった保護機構を提供すること。
- 効率のよい保護機構を提供すること。

(b) 保護モデルの基本要素

ここでは，本モデルの基本的構成要素について説明する。

- 対象（オブジェクト）
- 主体（サブジェクト）
- アクセス行列とアクセス規則
- ドメインと資格（capability）

対象の例としては，端末，ファイル，ページ，プログラム，補助記憶などの，ソフトウェア，ハードウェア資源が考えられる。主体の例には，プロセスがある。

(c) アクセス属性とアクセス権制御属性

アクセス属性は，主体が対象に対するアクセスを規定するための属性であるが，これらは対象の種類により，いろいろなものがある。通常オペレーティング・システムの中では，「読

み」，「書き」，「実行」がある。アクセス権制御属性は，ある主体があるドメインから別のドメインに移り変わる時の制約条件を規定するものである。アクセス権制御属性として，「所有者」，「制御」，「転記フラグ」などが提案されている。

(d) 保護の実現法

保護状態を表わすためのアクセス行列はスパース行列であるため，実現が困難である。ここでは，保護状態の具体的な実現方法として，次の方式を説明する。

- 資格リスト
- アクセス制御リスト
- ロック／キー機構

9.9 内部記憶保護[1]

多重プログラム環境では，同時に実行される複数の利用者プログラムや制御プログラムが主記憶装置内に混在している。これらの複数のプログラム，およびデータをお互の不当なアクセス，または破壊から保護するのが主記憶保護の目的である。ここでは，主記憶保護の実現方式について説明する。

(1) 主記憶へのアクセス保護属性

主記憶へのアクセス保護属性は，「書き込み禁止」，「読み出し禁止」，「実行禁止」がある。

(2) 主記憶へのアクセス保護方式

一般の商用コンピュータが採用している記憶アクセス保護方式について説明する。

(a) 領域レジスタ方式

プロセッサは，マスタ／スレーブ・モードで動き，スレーブ・モードのプログラムは，メモリの上限，下限を示す領域レジスタで示される領域にのみ書き込み可能で，マスタ・モードのプログラムは，どの領域にも書き込み可能である。

(b) 保護キー方式

主記憶の一定サイズ毎にアクセス保護ビットと，何ビットかのキーを持つ。

制御レジスタにも，同じ長さのキーを持つ。そして，記憶アクセスの許可は，アクセス保護ビットの状態と，両方のキーの一致性を基準にして決定される。

(c) リング保護

リング保護はプログラム間に階層構造を持たせた保護方式で，(a)の領域レジスタ方式をより一般化したものと云える。主記憶領域は，いずれかのリングに属し，リングの内側になるほど，強く保護される。ここでは，リング保護の特徴を示すため，次の項目について説明す

る。

- アクセス属性とリング番号
- 内向けアクセスと外向けアクセス
- 仮想記憶方式とリング保護の関係

9.10 ファイルの保護

ファイルはコンピュータ・システムにおける情報の蓄積の中心である。したがって，情報の保護が必須な部分である。ここでは，ファイルに関する保護がどのような方法でなされているか説明する。

(1) ファイルに対する保護

ここでは，ファイルを一つの単位とした保護方式について述べる。

(a) ファイルのアクセス保護属性

ファイルのアクセス保護属性には，一般に，「参照禁止」，「更新禁止」，「消去禁止」などがある。

(b) カタログによる保護

カタログは，ファイルを管理するための情報の集まりで，ファイルの自動アクセスを行うために使用される。このカタログにファイルに対するアクセス保護情報を登録し，ファイルアクセス時に，アクセス権のチェックを行う。

(c) パスワードによる保護

ファイルごとにパスワードを登録しておき，ファイルの使用開始時に，ファイル名とパスワードのチェックを行う。

(3) データベースの保護

データベース・システムの出現により，ただ単に，ファイル単位での保護機能では不十分になり，よりきめ細い単位での情報の保護が必要になってきた。ここでは，その一例として，CODASYLのDDLC*，PLC** 提案[9][10]のデータベースに対する機密保護について説明する。

CODASYL案のアクセス管理は，プライバシー・ロックとプライバシー・キーによる方式を，基本原則として採用している。プライバシー・ロックは，スキーマ，あるいは，サブスキーマで指定される。プライバシー・キーは，プログラム，サブスキーマで指定される。ロックの対象は，スキーマ，サブスキーマ，エリアレコード，データアイテム，セットなど多種多

(注) *DDLC：Data Description Language Committee

**PLC：Programming Language Committee

様である。また，アクセス保護属性も，スキーマ，サブスキーマの「参照」，「更新」，「転記」，「ロックの変更」や，各DML*コマンドが指定できる。

9.11 端末レベルでの保護[1]

ここでは，センタ・システムから離れた端末や通信回線上で生じる侵害に対するアクセス保護対策について説明する。

(1) 端末での保護対策

端末利用者が正当な利用者であるか否かを判定するため，一般にはパスワードを利用して行われる。また端末自体に識別子を与え，登録されていない端末からの使用を拒否する方法などが取られる。管理面からは，端末の設置場所を限定することが考えられる。

(2) 伝送上の保護

端末とコンピュータを結ぶ通信回線からの情報の傍受を防ぐため，暗号による情報の伝送が行われる。

9.12 運用管理[1]

コンピュータ・システム内の情報を破壊から守るために，システムの運用面からの情報の保護対策を立てなければならない。ここでは，運用面から考慮すべき問題を示し，その対策手段を説明する。

- システム設計上の問題
- 人事管理
- 災害からの保護

指導上の留意点

(1) 信頼性の問題は関係する範囲が広く，かつ技術的に深いものを含んでいる。ここでは全体の概要をバランスよく摑めるようにもっていくことが望まれる。

(2) システム作りの上で，信頼性への配慮は今後ますます重要になってくる。性能，機能と並行して当初から考慮すべき要素である。この重要性をよく認識させる。

(3) ハードウェア原価の低減に伴って，商用機でもハードウェア・レベルでの高信頼化が一段と進みつつある。たとえば演算ユニットの二重化，保守，診断用プロセッサの装備など。これらの動向についても触れることが望ましい。

(4) 今迄あまり注目されていなかったが，ソフトウェアの信頼性がシステムの信頼性の中で占め

(注) *DML：Data Manipulation Language

る位置は非常に大きい。ソフトウェアの信頼性の重要性を認識させると共に，最近諸所で行れている高信頼化の試みに注目させるようにする。

(5) 情報の保護の議論はひろく，社会的あるいは法的な問題にも及ぶが，ここでは，コンピュータの技術的な問題に留める。

(6) 情報の保護機能は，従来のコンピュータ・システムにおいて必ずしも十分であるとはいえない。また技術的にも解決されていない問題が多く，情報の保護には限界があり，保護にかける費用と，その効果とのバランスを考えた上でシステム設計を行うべきであることを，十分に認識させる。

(7) 同一データに対し，複数プロセスからの更新処理によって生じる矛盾を避けるための排他制御機能は，ここでは除く。

参考文献

〔1〕猪瀬博編「コンピュータ・システムの高信頼化」情報処理学会，1977

〔2〕「信頼性技術特集」電子通信学会誌，vol. 59，pp. 339－463，1977

〔3〕大島裕「ソフトウェアの信頼性」情報処理，vol. 16，pp. 887－894，1975

〔4〕吉村鉄太郎「情報の機密保護(1)」bit，vol. 5，no. 13，pp. 32－37，1973

〔5〕朴木実「データベース・導入と設計」企画センタ，1975

〔6〕D. C. Tschritzis, P. A. Bernstein著，斉藤信男訳「オペレーティング・システムの基礎」日本コンピュータ協会，1976

〔7〕H. E. Peterren and R. Turn, "System Implications of Information Privacy", Proc. AFIPS SJCC, pp. 291－300, 1967

〔8〕G. S. Graham, P. J. Denning, "Protection－Principles and Practice", Proc. AFIPS SJCC, pp. 417－429, 1972

〔9〕"CODASYL Data Description Language of Development June 1973", NBS Handbook 113, 1974

〔10〕"CODASYL COBOL Journal of Development", Canadian Goverment 110－GP－1d, 1976

〔11〕B. W. Lampson, "Protection" Proc. Fifth Princeton Symposium on Information Sciences and Systems, Princeton University, pp. 437－443, 1971

第10章　システムの性能評価および互換性

キーワード

スループット（throughput），応答時間（response time），ターンアラウンド時間（turn-around time），命令ミックス（instruction mix），高水準言語ミックス（high-level language mix），カーネル（kernel），標準問題（benchmark problem），合成プログラム（synthetic program），待ち行列モデル（queueing model），シミュレーション（simulation），モニタリング（monitoring），互換性（compatibility），移植性（portability，transferability），移行（migration），上位方向互換性（upward compatibility），プログラム変換（program conversion），標準化（standardization），データの互換性（data compatibility），符号体系（code system），ファイルの移行（file migration），エミュレーション（emulation），統合型エミュレータ（integrated emulator），可変構造型機械（variable structure machine），仮想機械（virtual machine），仮想機械モニタ（virtual machine monitor）

目　標

コンピュータ・システムの性能を評価するための手法には数多くのものがある。評価の目的に応じてそれらの手法の中から適切なものを選択し，評価を行うことが必要である。

この章ではまずシステムの性能評価に用いられる各種の手法の概要を理解させ，性能評価を行うとき，その目的によってどの手法をどのように適用すればよいかを理解させる。

コンピュータ・システムの性能は多くの要因が複雑にからむものであるが，今後，一層その重要さが増すものである。システムを設計する上で性能を常に意識する態度を育てることが大切である。

互換性，移行性が必要とされるのは，異るシステム間でのデータ交換，プログラム共用のためだけでなく，ユーザでシステムの増設，置き換えを行う場合がある。特に後者の場合，過去に膨大な投資を行って蓄積してきたソフトウェア財産（プログラム，データ）を，できるだけ低いコストで新システムへ移行できなければならず，移行性は非常に重要なポイントとなる。この章では，次に，互換性，移行性における問題点の所在を理解させ，これを解決するための標準化，一般に採用されている技術の現状についての一般的知識の習得をはかる。また，最近諸所で試みられている，この問題の解決につながると思われる新しい技術について紹介し，将来への展望を得

られるようにする。

内　容

10.1　システム性能の評価尺度

システム性能を評価する尺度について説明し，システムの利用形態に応じて評価尺度を選択すべきことを教える。

(1)　スループット

単位時間当りの処理量で定義される。通常

- 実行ジョブ数／単位時間
- メッセージ処理数／単位時間

などで表わされる。システムの処理能力を評価する基本的尺度である。

(2)　応答時間

主としてオンライン・システムやタイムシェアリング・システム（ＴＳＳ）のように実時間性の高いシステムの評価に用いられる。端末ユーザが処理要求を入力してから出力応答が返ってくる迄の時間で定義される。

バッチ処理システムではジョブの入力から結果の出力までの時間をターンアラウンド時間（turn-around time）あるいはＴＡＴと呼ぶことがある。

応答時間の表現法としては次のものがある。

- 平均応答時間
- 応答時間の分布又はある応答時間以内である割合

(3)　システム資源の使用率

スループット，応答時間，ＴＡＴなどがシステムの外部から見た評価尺度であるのに対し，システムの内部に立入った評価尺度として資源使用率（resource utilization）がある。ＣＰＵ使用率，ファイル装置の使用率，チャネル使用率などが特に重要である。

(4)　評価尺度間の関係

前述の評価尺度の間には一般に次のような関係があることを説明する。

- スループット　：　小→大
- 応答時間　　：　短→長
- 資源使用率　：　低→高

つまり，短い応答時間を保証する必要があるときはスループット，資源使用率を低くすることになり，スループットが最大のとき，応答時間は長くなる。

性能評価はこれらがバランスした動作点を見つけだすことが目的であることを理解させる。

10.2 評価の目的

性能評価はその目的で分類すると次の三つに分けられる。

- 機種選択
- 性能予測
- 性能改善

それぞれについて説明し，評価のポイントについて理解させる。

10.3 性能評価手法

(1) 手法の種類と目的との関係

性能評価のための手法には種々のものがある。手法の概略を説明し，評価目的とそれに用いられる手法との関係を理解させる。

主な性能評価手法には次のようなものがある。

- 競合比較手法
- 解析的手法
- シミュレーション
- モニタリング

目的とそれに用いられる手法との関係を図10－1に示す。

(2) 性能評価の階層構造

コンピュータ・システムのように複雑な構造体をいきなり評価するのは極めて困難である。また，重要なパラメータを見すごすおそれもある。そのため，システムをいくつかのサブシステムに分割して評価する方法をとることがある。図10－2に示すような階層構造モデルを例にとり，システムを階層的なサブシステムに分割しうることを説明する。

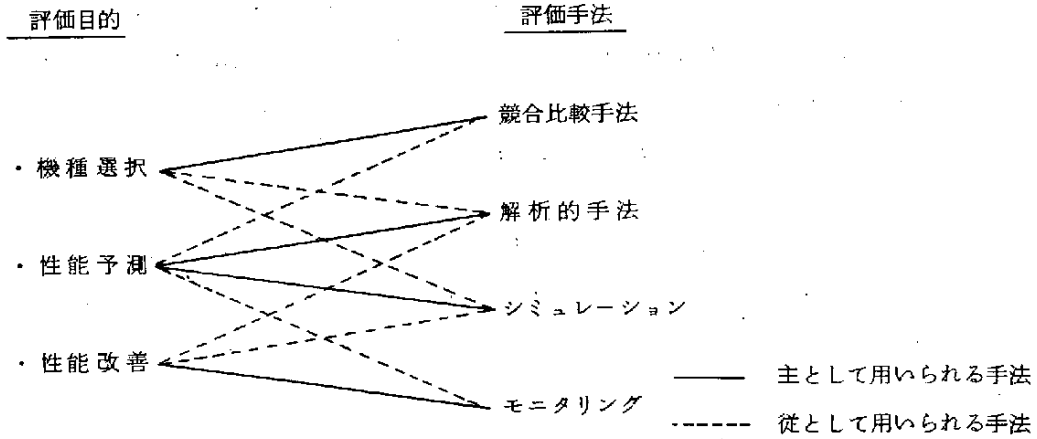


図10－1　評価の目的と手法

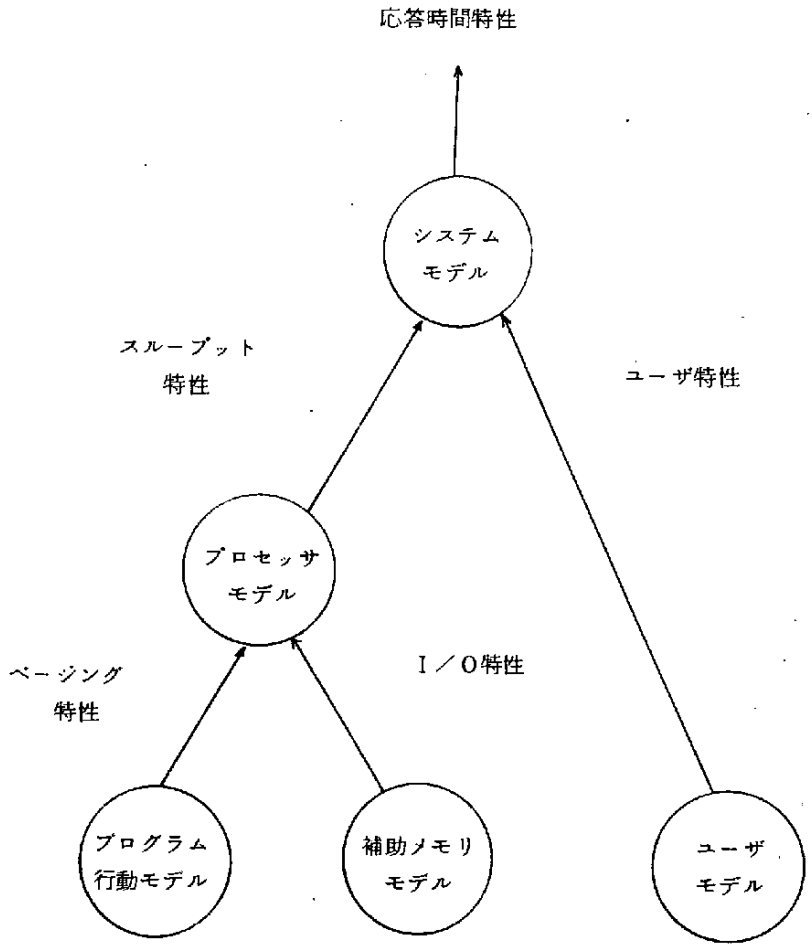


図１０－２　　階層構造モデルの一例

１０.４　競合比較手法

機種選択のための評価のように，複数の候補システムが既に存在する場合には，できる限り同等に近い環境条件を設定し，システム性能を相対評価する方法が用いられる。これは競合比率手法と総称される。次の方法について説明する。

(1)　命令ミックス

ＣＰＵの性能を比較するために用いられ，科学技術計算，事務計算用などの命令ミックスがある。

- ギブソン・ミックス（科学技術計算）
- 事務用ミックス

これを用いて平均命令実行時間や単位時間当りの命令実行回数（ＭＩＰＳ：million instructions per second で表わされる）を求める。この方法ではシステム性能をＣＰＵ性能で代表させようとするものであり，参考値として考えるべきものである。次のような問題を説明する。

- 命令ミックスと実際の使用状況とが合わないことがある。
- 命令ミックスの定義と対象機の命令レパートリーの対応が明確でない。
- 先取り制御やキャッシュのようなハードウェア機構の効果を反映しにくい。
- 複合命令などの効果が表現しにくい。
- ソフトウェアの特性が無視される。

(2) 高水準言語ミックス〔３〕

命令ミックスの代りに高水準言語の文の出現頻度を用い，それに対する機械語命令の実行時間から平均文実行時間を求める方法がある。この方法ではコンパイラの特性も含めたＣＰＵ性能を求めることができる利点がある。しかし，標準的なミックスが定義されておらず，平均文実行時間の算出もかなり面倒である。

(3) カーネル

評価用の小さいプログラム（カーネル）を実行させ，その実行時間を比較する方法である。

(4) 標準問題

カーネルよりも大規模な，実際のシステムで用いられる典型的なプログラム（標準問題，benchmark problem）を実行させる方法がある。標準問題が実際のシステム特性を反映している場合にはかなり良い評価ができる。しかし，評価にかなり大きい費用がかかる欠点がある。

(5) 合成プログラム又は合成ジョブ

標準問題として実際のプログラムを使用する代りに，求める負荷特性を持つプログラムやジョブを作りだすプログラムを用いる方法である。ＣＰＵの処理時間，入出力回数，主記憶の占有特性などをパラメータとして与える。あまり一般的ではないが有効な方法である。

１０.５ 解析的手法〔４〕

コンピュータ・システムを数学モデルで表現し，システム性能を解析的に求める方法である。主として，待ち行列モデルにもとづくものが多い。これはシステムの資源，サービスを提供する窓口，ジョブ・プログラムをサービスを要求する客としてモデル化し，待ち行列の長さ，待ち時間の分布を用いてシステム性能を求める方法である。

多重プログラミング・システムの待ち行列モデルの一例として図１０－３の有限母集団二段窓口モデルを説明する。システムが比較的単純なモデルで表現しうること，このようなモデルから

ＣＰＵ使用率，平均応答時間などが解析的に求められることを理解させる。

解析的手法はシステム設計の初期に，大まかなシステム性能を予測するのに適しているが，モデル化が適切であればかなりの精度でシステム性能を評価しうる。

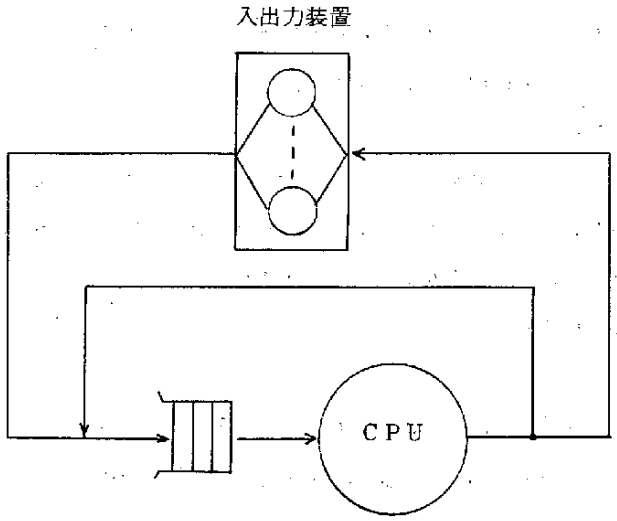


図１０－３　有限母集団二段窓口モデル

10.6 シミュレーション〔５〕

シミュレーションによる方法は他の手法に比べて柔軟性があり，特に，詳細なシステム動作の評価が可能な点に特徴がある。

シミュレーション・モデルをプログラムで作成し，評価データを得る方法である。シミュレーション言語には次のようなものがある。

- ＧＰＳＳ
- ＳＩＭＳＣＲＩＰＴ
- ＳＩＭＵＬＡ

コンピュータ・システム用の商用シミュレーション・システムには次のようなものがある。

- ＳＣＥＲＴ
- ＣＡＳＥ

シミュレーションは多くの性能評価問題に適用することができるが，シミュレーション・モデルの作成，シミュレーションの実行などに多大な労力と経費を要することが欠点であることを説明する。

ソフトウェア・シミュレーションに要するシミュレーション時間を短縮するために，モデルを

ハードウェアで実現する方法もある。主として研究用のものが多く，あまり一般的ではない。簡単にふれる程度で良い。

10.7 モニタリング

実際に稼動中のコンピュータ・システムから直接評価データを収集する方法をモニタリングという。主として性能改善のために用いることが多い。

(1) ソフトウェア・モニタ

対象となるコンピュータ・システムの上でデータ収集用のプログラムを走らせ，評価データを得る方法である。最近の多くのオペレーティング・システムに用意されている料金計算用データを記録する機能は一種のソフトウェア・モニタである。

次のような手法が用いられる。

① 事象起動

システム内の事象を検出し，それによって測定プログラムに制御を渡す。事象の起動にはオペレーティング・システムのルーチンを改造するなどの方法が用いられる。

② サンプリング

一定周期ごとのタイマ割込みなどを用いて測定プログラムを実行し，システムの状態を記録する。システムの状態はオペレーティング・システムのテーブル類を参照することによって求める。

(2) ハードウェア・モニタ〔6〕

システムの動作を測定する特殊な装置（ハードウェア・モニタ）を対象システムに接続し，評価データを収集する方法である。ハードウェア・モニタの基本的な機能を図10－4を用いて説明する。ハードウェア・モニタは対象システムの動作に干渉することなくデータを得ることができるが，かなり高価なハードウェア装置が必要である。

(3) 測定項目

モニタリングによって種々のデータを得ることができるが，主なものには次のようなものがある。

- システム資源使用率（CPU，チャネル等）
- プログラム使用頻度，実行時間
- OS時間／ユーザ時間
- ファイル使用率

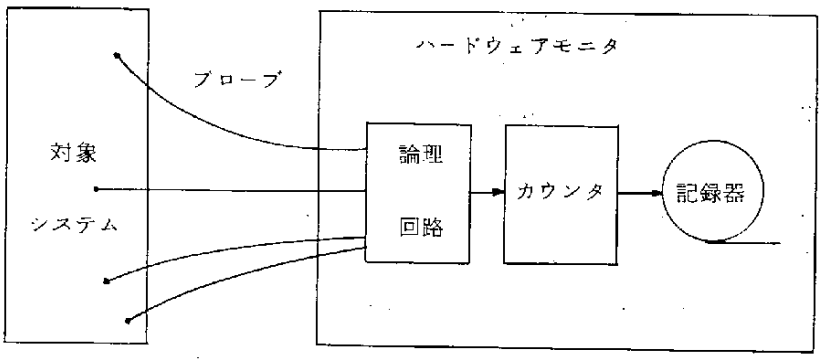


図１０－４　　ハードウェアモニタの基本構成

１０.８　システムの互換性および移行性の概要

(1)　互換性，移植性，移行

互換性（ compatibility ），移植性（ portabilityまたは transferability ），移行（ migration ）について，各々の概念を説明し，その重要性を理解させる。

互換性とは，システムに含まれるある要素が異なるシステムでも共通に使用できることを意味している。移植性は一般にプログラムに関して用いられる言葉で，同一のプログラムを異なるシステム上で実行させられる場合に，そのプログラムは移植性があることになる。移行とは既存のシステムから新しいシステムへ業務を移すことを意味する。

コンピュータの置換え，グレード・アップ時などに起る。互換性はなくても，移行のための種々の手段を用意しておくことにより，移行性を高めることはできる。

これらの性質が必要になる場面としては，

- コンピュータの置換え，グレード・アップ時
- 同一プログラムを別のシステムで使いたい場合
- 異なるシステム間でデータを交換したい場合（仕事の分担）

がある。

ソフトウェアおよびデータがユーザにとって非常にコストのかかった財産であることを認識すると，移行性の重要性が理解できる。第３世代以降，ほとんどのメーカーが大幅なアーキテクチャの変更を避けているのはこの理由によると考えられる。今後は，新機種導入時のシステム中断が許されないケースが増えると考えられ，この面の配慮も必要となることを説明する。

(2)　互換性，移行性の対象

互換性を考える際に対象とすべき重要な項目には以下のものがある〔７〕。

① 応用プログラムの互換性，移行性

② オペレーティング・システムの移行性

③ データ／ファイルの互換性，移行性

④ 記憶媒体の互換性

⑤ 周辺装置の互換性

⑥ システムの操作法の移行性

これらの互換性，移行性が要求される場面について説明する。②の移行性は，それ自身が膨大なソフトウェアであることと共に，応用プログラムとインタフェースをもつため，これを変えたくないことから要求される。④，⑤はハードウェアの互換性である。

①～④については以下の節で取扱うが，⑤，⑥の内容についてはここで説明する。

⑤は同一の周辺装置が異なるシステムで使用できるか否かの互換性で，中央処理装置と周辺装置の間のインタフェースの統一の問題となる。⑥はオペレータの操作卓からの操作法，ジョブ制御言語などが問題となり，設計思想が似ていることが望まれる。

10.9 プログラムの移行

ソフトウェアを他のコンピュータへ移行する場合は，中央処理装置とのインタフェース（命令体系など）だけでなく，オペレーティング・システムとのインタフェース，システム構成の違いなども問題になってくる。ユーザから見た移行の形態としては次の3種類がある。これらについて，実現法を説明する。

① 目的プログラムでの互換性

② 原始プログラムでの互換性

③ 変換による移行性

(1) 目的プログラムでの互換性

次のような実現法がある。これらの利害得失について説明する。なお，エミュレーションと仮想機械については別の節で詳しく取扱う。

① アーキテクチャを一致させる。

最も確実な方法であり，シリーズ・マシンはこの考えの下に提供されている。大型の場合は高速化だけでなく機能拡張されていることが多く，また最適のオペレーティング・システムも一般に異なる。このため上位方向互換性のみが保たれているのが一般的である。

② シミュレーション

個々の命令ごとにシミュレートする。速度が遅く，オペレーティング・システムの扱いにも難点がある。開発中のコンピュータに対するソフト開発用として用いられる程度で，あま

り使われない。

③ エミュレーション

マイクロプログラムでシミュレーションを行う。実用化の例は多い。

④ 仮想機械

同一アーキテクチャ上における，オペレーティング・システムの違い，システム構成の違いを吸収できる。

(2) 原始プログラムでの互換性〔8〕〔9〕〔10〕

① 高水準言語の標準化

一般に使われている汎用言語の標準化の動向について説明する。原則的には再コンパイルで済むはずであるが，実際には言語仕様の細かい差異があり，変換プログラムが用意されている。しかし，これを用いても人手による部分修正が必要であるのが一般的であることを説明する。

② アセンブラ言語の互換性

汎用アセンブラの試みはあったが成功したとは言えない。同一アーキテクチャでオペレーティング・システムが異る場合は変換プログラムで変えられる。

③ 言語プロセッサの移植

ソフトウェアとしての言語プロセッサ自身の移植という意味と，原始プログラムの互換性が得られるという二つの意味がある。最近，色々な手法が試みられている。代表的な手法について説明する。

- 中間言語の使用
- プロセッサ記述言語の使用
- マクロ・プロセッサの使用
- ブートストラップ方式

(3) プログラム変換による移行

① 機械語→アセンブラ言語

② アセンブラ言語→アセンブラ言語

③ アセンブラ言語→コンパイラ言語

④ コンパイラ言語→コンパイラ言語

いずれも実現例はあるが①，②は対象機械の制限が厳しく，③は第2→3世代の移行期に相当使われたが欠点が多かった。

10.10 データ／ファイルの互換性と移行

ハードウェア・レベルにおけるデータの互換性と，それを基礎にしたソフトウェア・レベルでのファイルの移行性が問題となる。

(1) データの互換性

同一シリーズ内ではデータの互換性は保たれている。他シリーズとの間のデータ記録媒体における互換性について説明する。

① 磁気テープ

媒体そのものはＪＩＳが制定されて，互換性がある。符号体系と記録方式については必ずしも統一されていない。しかし，ハードウェアに選択機構をもたせる，ソフトウェアで変換プログラムを用意するなどして，異機種間の情報交換手段として最もよく用いられている。

② カード

媒体としては標準化が行われている。コードは統一されていないが，ソフトウェア的に変換することにより互換性を保ちうることを説明する。

③ 紙テープ

規格，符号体系の現状について説明する。

④ 磁気ディスク・パック

標準化は進んでいない。情報交換用には使用されておらず，システム置換え時に問題となることを説明する。

⑤ 通信回線

回線自体は完全に規格化されているが，伝送制御手順，プロトコル，符号体系などほとんど統一されておらず，システムごとに設定しなければならない現状を説明する。

(2) ファイルの移行

媒体の互換性がない場合，あるいはファイル形態が異る場合はディスク上のファイルであっても，一度磁気テープへダンプし，必要ならば磁気テープ上での変換を行った後に新しいディスクへロードするという形がとられている現状を説明する。移行時にシステムの中断が許されない場合は，新旧システムを共存させておいて逐次データを新システムに移行していくような方式も必要となる。

10.11 エミュレーション技術

エミュレーションはアーキテクチャの異るコンピュータ間のプログラム移行を実現するための道具として広く用いられている。エミュレータの採用はプログラム移行作業を集中的にではなく徐々に進めることを可能にする。ここではエミュレータの実現技術について紹介する。

(1) エミュレーションのレベル

エミュレーション対象に含めるレベルとして次の二つがある。

(a) 機械語命令

(b) 制御プログラム

(a)では旧制御プログラムをそのままエミュレータ上で走らせる。(b)の方が効率は良いが開発が大変でまた融通性に乏しい。

(2) ＣＰＵ命令のエミュレート方式

(a) ソフトウェア制御

(b) ファームウェア制御

(c) 補助プロセッサの採用

これらの方式について説明する。現在は(a)の方式が多い（一命令エミュレートごとにソフトウェアへ制御が戻る）。

(3) エミュレーション環境

次の二つのタイプについて説明する。特に統合型が登場した背景について説明する。

(a) スタンドアロン型エミュレータ

(b) 統合型エミュレータ

(4) 入出力要求の処理

入出力要求はソフトウェアに渡されて処理される場合が多い。この場合，ホストの制御プログラムを利用するのが一般的である。この間のインタフェースのとり方を説明する。

(5) システム資源の変換

以下の資源の変換法について，具体例を上げて説明する。

- 周辺装置
- 主記憶
- レジスタ
- コンソール

(6) 可変構造コンピュータ〔１２〕

汎用エミュレーション・マシンとしてＭＥＴＡ－４，ＭＬＰ９００などを紹介し，そこで用いられている技術について説明する。

- レシデュアル制御方式
- ランゲージ・ボード

また，問題適応型可変構造コンピュータとしてＱＭ－１，Ｂ１７００を紹介し，そこで用いられている技術，思想の概要を説明する。

10.12 仮想機械〔13〕

オペレーティング・システム，機器構成の異なるシステムへの移行をスムーズに行わせる手段としても用いられる仮想機械の概念について説明する。

(1) 仮想機械の概念

仮想機械の定義を与え，その性質について説明する。仮想機械はハードウェアとソフトウェアで実現された実在システムの複製であり，大部分の命令はホスト・マシン上で直接実行される。仮想機械モニタによって複数のハードウェア・ソフトウェア・インタフェースが作り出され，従って，同一のハードウェア上で複数のオペレーティング・システムを同時に動かすことができる。

(2) 仮想機械の実現方法

次のシステム資源を仮想化することによって実現される。仮想化の方法について簡単に説明する。

① システム制御パネル，操作卓

② 中央処理装置

- ＣＰＵタイムの割当て
- 仮想機械の状態の管理
- 入出力要求の実行

③ 入出力システム

- 装置，アドレスの変換

④ 主記憶装置

- shadow page table

(3) 有効性

- オペレーティング・システムの違うプログラムの同時処理
- オペレーティング・システム移行時における新旧の並行処理
- 古いオペレーティング・システムで新しい入出力機器のサポート
- バック・アップ用として有効
- 特権プログラムのデバックを通常業務と並行に可能
- 中規模システムで大規模システムの開発やテストが可能

現在は同一シリーズ上のオペレーティング・システムの違いを吸収しているだけであるが，将来はエミュレータとの組合せで移行性の問題を解決することが期待されている。

指導上の留意点

(1) 個々の評価手法については手法の内容を理解させる程度にとどめる。簡単な手法，たとえば，カーネルや単純な待ち行列モデルなどはできるだけ具体例をあげて説明することが望しい。

(2) 本コースに深く関連するものには，第2部のシステム開発運用の背景，第4部の情報システムの開発などがある。待ち行列モデルはオペレーションズ・リサーチで，ソフトウェアの評価などはシステム設計で修得するものとする。

(3) システムの増設，置き換えを行う場合，単に価格／性能比だけでなく旧システムからの移行性が重要なポイントであることをよく認識させる。

(4) 機種が変る場合に，一時にすべてのソフトウェアを変換するのは無理があり，新旧システムの並行処理を行いつつ逐次移行していくのが望ましい。現状ではエミュレーション，仮想機械はこのための道具として位置づけられていることを理解させる。

(5) 移行のツールは100％の移行性を保証するものではない。各々の限界についてできるだけ説明を行うことが望ましい。

参考文献

〔1〕三上，箱崎，関野「計算機システムの評価技術〔Ⅰ〕，〔Ⅱ〕」電子通信学会誌，vol.59，pp.1083－1090，1369－1376，1976

〔2〕情報処理，vol.13，no.11，1972

〔3〕北村，金森，徳永「高水準言語指向のＣＰＵ性能指標について」信学会電子計算機研資ＥＣ－74－20，pp.59－69，1974

〔4〕橋田温「情報システムにおける待ち行列理論の応用（1），（2）」情報処理，vol.18，pp.184－195，289－297，1977

〔5〕三上，亀田「コンピュータと情報システムにおけるシミュレーション技術」電子通信学会誌，vol.60，pp.761－769，1977

〔6〕箱崎，小野「ハードウェアモニタの技法について」情報処理学会SE研資，74－1，1974

〔7〕石井善昭「計算機の基本方式」共立出版，pp.114－129，1973

〔8〕松下温，山崎晴明，丹下栄二「プログラム・トランスファラビリティ」情報処理，vol.16，pp.871－879，1975

〔9〕正田輝雄「コンパイラのキットを用いたＰＡＳＣＡＬの移植」日経エレクトロニクス，no.149，pp.100－131，1976

〔10〕 P. J. Brown著，鳥居，杉藤，真野訳「マクロ・プロセッサとソフトウェアの移植性」近代科学社，1977

〔11〕 石井善昭「エミュレーション」電気四学会連合大会予稿，pp. 1354－1357，1973

〔12〕 所真理雄「可変構造計算機とソフトウェア・エンジニアリング」情報処理，vol. 16，pp. 906－912，1975

〔13〕 R. P. Goldberg "Survey of Virtual Machine Research", Computer, pp. 34－45, June 1974

単 元 C 3

ファイルおよびコミュニケーション・システム

単 元 C 3

ファイルおよびコミュニケーション・システム

目　次

第1章　ファイルおよびコミュニケーション・システムの機能

キーワード

情報管理(information management)，情報システム(information system)，ファイル管理(file management)，コミュニケーション機能(communication facility)，障害対策(fault countermeasure)，機密保護(security protection)，データベース／データ・コミュニケーション(DB／DC)(data base／data communication)，ファイル構造(file structure)，ファイル操作(file manipulation)，データベース(data base)

目　　標

コンピュータ利用を基礎とした総合情報システムでは，ファイル管理の機能やコミュニケーション制御の機能が重要な位置を占めている。

ファイル管理はデータベースの概念を中心としたデータベース管理システムや，ファイルの記憶媒体としてのディスクなどの記憶装置の発展，さらにデータベース・マシン化の研究などにより，これからの発展が期待される。

また，コミュニケーション制御の分野においても，コンピュータ・ネットワークなどにみられるように，システムの様々な構成方法や運用方法が追求されている。

この章では，ファイル管理およびコミュニケーション制御の基本的機能を概観し，総合的な情報システムを構成する上で，両者の位置づけ及び統合化について理解させることを目標とする。

内　　容

1.1　総合情報システムへの期待

この節では，現存するコンピュータの利用形態を調べて，情報の利用と管理がどのように行なわれていたかを明らかにする。

これらの調査結果をもとに，総合情報システムの必要性や期待がどのようなところにあるのかを導き出す。

(1)　情報の利用と管理

これまでの適用業務ごとに独立していたシステム(ここでは業務別システムと呼ぶ)の特徴を調べ，その利点，欠点を明らかにすることを目的とする。

次のような項目に沿って説明するとよい。

① マスタ・ファイルの意味

② サブシステムごとのファイルの必要性

③ システムの寿命

④ システム化のアプローチ法

⑤ ファイルの保守性

⑥ 必要とする情報の範囲

⑦ 必要とする情報の即時性

⑧ 必要とする情報の形態

業務別システムの利点と欠点を明らかにし，ファイルの統合化を必要とする総合情報システムについて考察する。この場合，次のようなことがポイントとなるであろう。

① ファイルの統合化の必要性

② データの品質

③ システムの開発・保守のコスト

④ 利用者の層

(2) 情報の機密保護

総合情報システムにおいては必然的にファイル（データ）が共用されることになる。このため品質の高いシステムを維持する上で，意図的なデータベースの破壊や盗難からデータを守ることが必要となる。そのような機密保護の必要性を認識させ，このための方法を説明する。

機密保護は，残念ながら現在のコンピュータ技術では完全に行なえるとはいえないが，どのような問題があるかを明らかにし，総合情報システムの機密保護の限界について説明する。また，暗号化を含む機密保護についての最近の動向にも着目させる。

① 機密保護に伴う各種のセーフガード

物理的セーフガード，ハードウェア上のセーフガード，ソフトウェアのセーフガード，通信面でのセーフガード，要員面でのセーフガード，行政経営面からのセーフガード

② プライバシーの保護

(3) 障害対策

総合情報システムでは，ファイル（データ）が共用されているので，ファイルの破壊により引き起こされる影響は非常に大きい。ここでは，障害対策の重要性を強調し，そのためにどのような配慮がなされるかを説明する。

まず，障害の種類は以下のようになる。

① ハードウェア上の障害

- ＣＰＵ（主記憶装置を含む）
- ファイル媒体
- 回　　線
- 周辺機器

② ソフトウェア上の障害

- オペレーティング・システムのバグ（制御プログラム・言語処理系を含む）
- 応用プログラムのバグ

③ 運用上の障害

- 操作の誤り

これらの障害に対する復旧手段としては，ファイルの復旧と処理の復旧の二つに分けて説明するのがよい。

① ファイルの復旧

- ファイルのバック・アップ
- データの変更履歴

② 処理の復旧

- チェックポイント／リスタート
- ソウトウェア上の考慮

障害対策はシステム構築上，その経済性とトレードオフの関係にあり，どのような対策手段を講じるかは十分に検討しなければならないことを説明する。

1.2 ファイル管理の基礎

この節では情報システムの基礎となるファイル管理について，どのような構成や操作があるかを概念用語の説明をしながら概説する

(1) ファイルの構成

ファイルの構成要素として次のことを説明する。

① データ項目

② 集団項目

③ 繰り返し集団

④ レコード

⑤ レコード間の関係

⑥ ファイル

⑦ ファイル編成

(2) ファイルの操作

ファイルの操作について次のことを説明する。

① ファイルの生成

② 索引，多重リスト構造，逆ファイル

③ 直接呼び出し，順呼び出し

④ 分　類（sorting）

⑤ 照合（matchiny）

⑥ ファイルの更新

(3) ファイル・システムの機能

ファイル・システムのもつ基本的な機能や操作について概説する。次のような観点からファイル・システムの機能をまとめるとよい。

① データ構造とその表現方式

② データ独立性の程度

③ データ検索の機能

④ データの整合性

⑤ システムの操作モード

これらの機能の整理を通して，データベース管理システムの必要性が示せればよい。

1.3 データ通信の基礎

この節ではコミュニケーション・システムを理解するための基礎となるデータ通信の知識を理解させる。

(1) データ通信の定義

データ通信を正しく理解させるためには，その用語の定義を明確にする必要がある。データ通信は「データ伝送」と「データ処理」とが結合したものと言える。

(2) データ通信システムの概要

データ通信システムは，データを能率良く正確に伝送するデータ伝送系とより速く正確にデータを処理するためのデータ処理系とから構成されている。代表的な構成機器には次のものがある。

・コンピュータ

・通信制御装置

・変復調装置

・通信回線

・端末装置

図1・1を参考にして，以上のことを説明するとともに，各々の機器の概要についても理解させる。

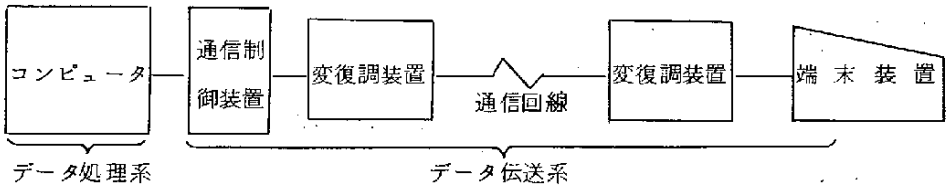


図1－1 データ通信システムの構成例

(3) 情報の符号化

データ通信ではコンピュータと端末装置との間でデータの交換が行われるため，情報の符号化が行われる。情報の符号化は国際的な標準化が行われており，ＩＳＯとＣＣＩＴＴが協同でＩＳＯコード（国際標準符号）を定めている。我国ではＩＳＯコードにカナ文字を追加したＪＩＳコードが用いられている。これらコード体系について理解させる。

(4) 通信速度

単位時間に伝送できる能力を示す尺度として，通信速度がある。通信速度の表わし方として次の三つがあり，各々について説明する。

- 変調速度（ボー）
- データ信号速度（ＢＰＳ）
- データ伝送速度（字／分）

(5) 通信方式

データ通信における通信方式は，データの流れる方向によって類別すると，次の三つがある。各々について説明する。

- 単向通信方式
- 半二重通信方式
- 全二重通信方式

(6) 通信回線の構成と伝送制御方式

通信回線の構成には，直通方式と分岐方式とがある。またコンピュータと端末装置との間で，より正確に，かつ効率良くデータのやり取りを行うための伝送制御方式がある。これらについて簡単な例をあげて説明する。

1.4 総合情報システムの概念

この節では，総合情報システムの概念としてデータベース／データ・コミュニケーション機能について概観し，総合情報システムの全体像を理解させる。

① 総合情報システムの必要性

② データベースの必要性

③ コミュニケーション機能の必要性

この節では1.1から1.3までの説明をまとめ，第2章以降の各論への導入として位置づけられればよい。

(1) 総合情報システム

この節ではコミニュケーション・システムとファイル・システムとが結合した総合情報システムの概念について理解させる。

総合情報システムのハードウェア構成は，図1・2のような例をあげて説明する。

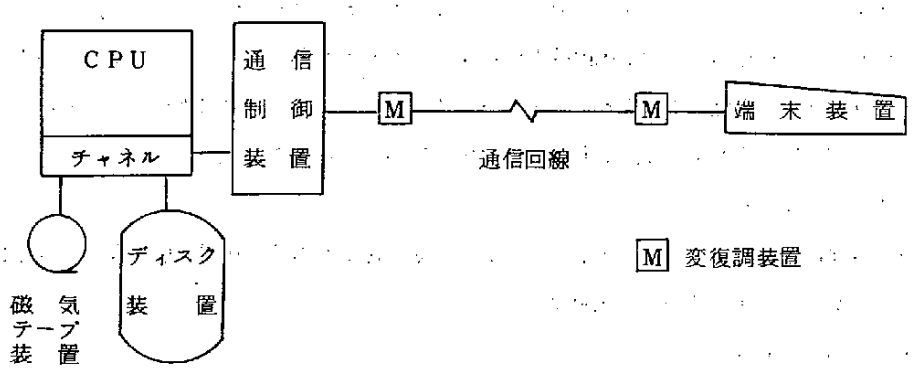


図1－2 総合情報システムのハードウェア構成

(2) 総合情報システムにおけるメッセージの流れ

ここでは総合情報システムにおけるメッセージの流れについて図1・3のような例をあげて説明する。

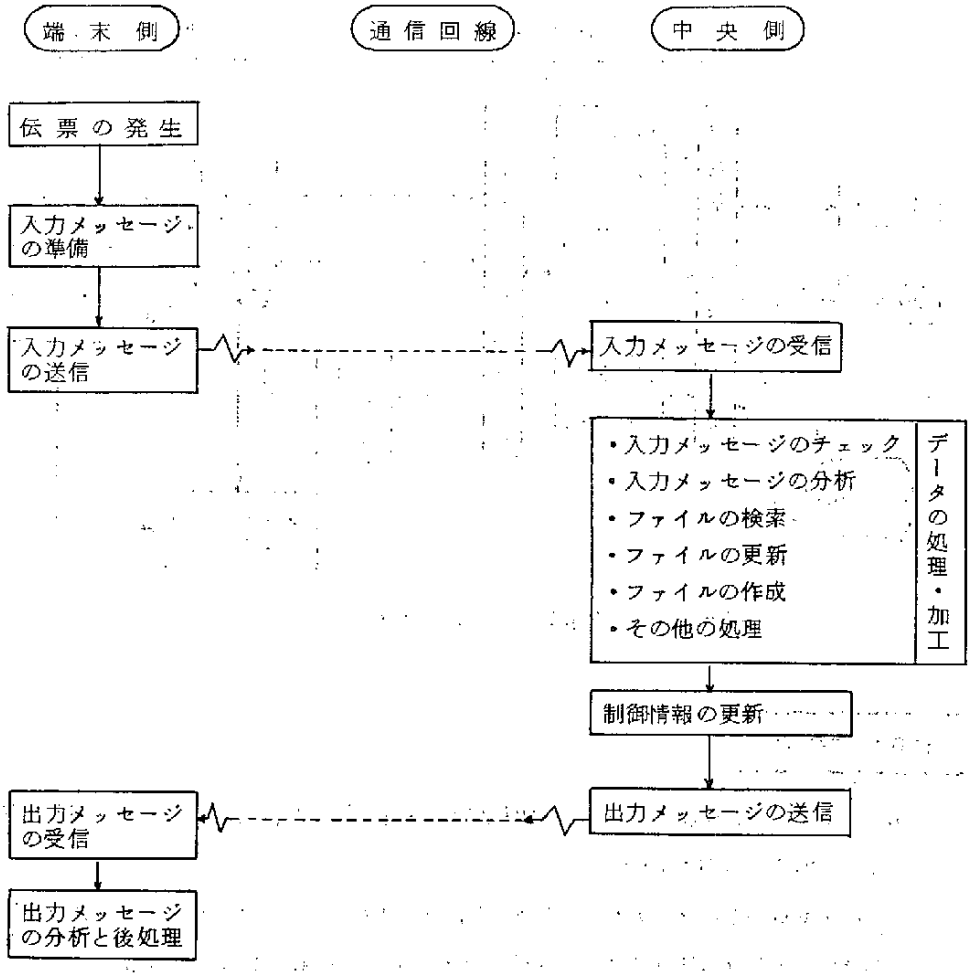


図1－3　メッセージの流れ

(3)　総合情報システムのソフトウェア構成

総合情報システムのソフトウェア構成は，機能的に管理プログラムと処理プログラムの二つに分類することができる。どちらのプログラムも開発の容易さ，機能変更に伴う修正のしやすさなどから機能あるいは業務単位のモジュール構造になっている。図1－4のような例をあげて，各モジュールの機能，データの流れなどについて説明する。

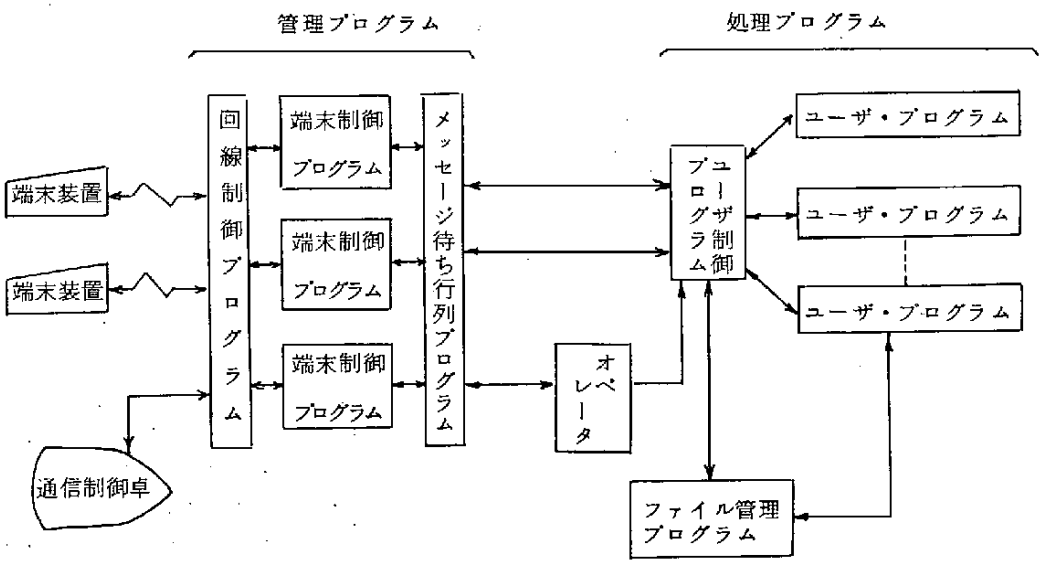


図1－4　総合情報システムのソフトウェア構成

指導上の留意点

(1) 本章は，第2章以降の説明の問題提起の部分となるため，第2章以降の展開を考慮した上で，説明するのが望ましい。

(2) 第10章の総合システムとの関連から，ファイルとコミュニケーションの諸機能についての基礎を十二分に理解させ，総合情報システムへの展開が開けるよう留意する。

参考文献

[1] 小林孝次郎 「情報構造」 サイエンス，1977

[2] K.Samuelson et al., "Information Systems and Networks", North-Holland, 1977

[3] Buckley, "Communications and Data Management", John-Wiley, 1976

[4] Brabb, "Computers and Information Systems in Business", Houghton-Mifflin, 1976

[5] J.Martin, "Telecommunications and the Computer", Prentice-Hall, 1976

[6] Murdick, Ross, "Information Systems for Modern Management", Prentice-Hall, 1975

[7] Rothman , Mosmann , "Computers and Society", Science Research Associates , 1976

第2章　ファイル・システムのためのハードウェア

キーワード

順次アクセス記憶装置(sequential access storage device)，直接アクセス記憶装置(direct access storage device)，磁気テープ(magnetic tape)，磁気ディスク装置(magnetic disc drive)，磁気ドラム装置(magnetic drum system)，IRG(inter-record gap)，磁気ヘッド(magnetic head)，トラック(truck)，シリンダ(cylinder)，アクセス機構(access mechanism)

目　　標

データの記憶媒体には，大別して二つの記憶装置がある。一つはレコードの記憶や検索において，一定の順序で処理するのに適したものである。もう一つは，レコードの順序が不定であっても，目的のレコードを読み出したり，更新したりするのに適したものである。これらは，それぞれ順次アクセス記憶装置，直接アクセス記憶装置と呼ばれている。

この章では，データの記憶媒体としての記憶装置を類別し，各装置の特徴を解説する。また各記憶装置では，どのような形でレコードが記録されるかについて述べ，各記憶装置におけるファイルの記憶容量やアクセス・コストなどの性格を説明する。

内　　容

2.1　記憶装置の分類

記憶装置には，定まった順序で処理するのに適したものがある。磁気テープ，紙テープ，カードなどは，順次アクセス記憶装置と呼ばれている。これに対し磁気ドラム装置，磁気ディスク装置などは，直接アクセス記憶装置と呼ばれる。これらの記憶装置は，順不同のレコードを直接読み出したり，更新したりするのに適している。

記憶装置のなかには，ファイル・システムのための記憶媒体としてはあまり使われなくなったものもあり，ここでは記憶装置を次のように分類し，その特徴を説明する。

(1)　磁気テープ

磁気テープに関し，図2－1などを用い次のことを説明し，その物理的な構造について理解させる。アクセスの特性や記憶容量の問題については，次の2.2で述べるのでここでは触れない。

① 記録形式（トラック）

9トラック方式，7トラック方式

② データの記録法

ISOコード，JISコード，ANSコード，EBCDI，Cコード，BCD，2進表示など

③ 記 録 密 度

④ チェック・ビット

⑤ ブ ロ ッ ク

⑥ IRG（inter-record gap）

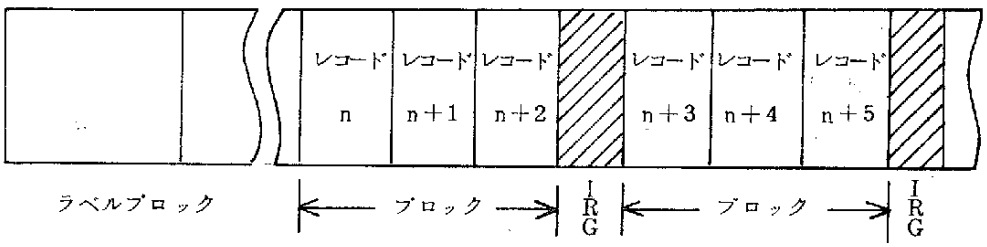


図2－1　磁気テープ

(2) 磁気ディスク装置

磁気ディスク装置に関し，次のことを説明し，その物理的な構造について理解させる。

① 磁気ディスク

② アクセス機構

③ 磁気ヘッド

④ ア ー ム

⑤ トラック

⑥ シリンダ

⑦ シーク（seek）動作とサーチ（search）動作

ディスク装置は，装置により異なっている事があるので，上記の概念の説明にともなうディスクの使用法については，具体的な装置を例に説明するのがよいであろう。これには，図2－2のような磁気ディスク装置の概念図が参考になるであろう。

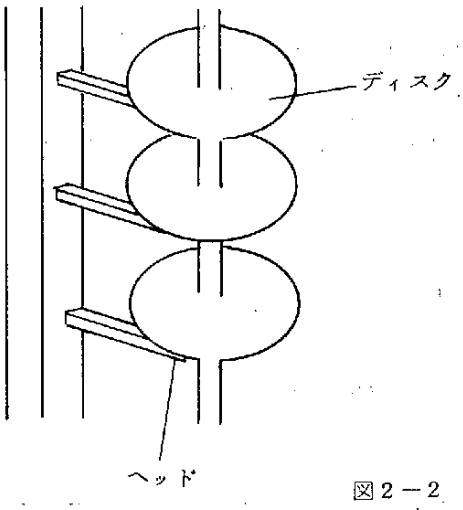


図２－２　磁気ディスク装置

(3)　磁気ドラム装置

磁気ドラム装置は，磁気ディスク装置と類似しており，その物理的な構造は，磁気ディスク装置の説明から類推できるであろう。

次の観点から，磁気ドラム装置の説明を行なう。

①　磁気ディスク装置との相異

・トラックの概念

・シリンダの概念

・固定ヘッドと浮動ヘッド

②　磁気ディスク装置との性能上の相異

・高　速　性

・容量，交換可能性の面

(4)　そ　の　他

記憶装置には，この他に次のようなものがある。これらは必ずしもファイル・システムのための記憶媒体としては適したものとはいえないが，入出力媒体として現在も利用されているので，簡単に説明する。

①　紙　テ　ー　プ

②　カ　ー　ド

特殊な場合として，磁心記憶装置がファイル装置，特に索引ファイル用として使われることがあることにも触れる。

2.2 記憶装置のアクセス

この節では，2.1 で説明した各記憶装置に関し，そのアクセスの特性や，記憶容量，アクセス・コストなどについて説明する。これらの説明を通して，ファイル・システムの設計時における適切な装置環境を決定するのに役立つ知識を与える。記憶装置のアクセスの問題はファイル編成とも密接な関係をもっているが，これについての問題は第3章で取り扱うので，ここでは触れない。

(1) アクセスの特性

各記憶装置について，次の観点から，そのアクセスの特性を説明する。

① 順次アクセス記憶装置

- 経　済　性
- デ　ー　タ　量
- アクセス速度

② 直接アクセス記憶装置

- レコードの直接読み出し
- デ　ー　タ　量
- アクセス速度

(2) 記　憶　容　量

記憶容量の決定要因について説明し，具体的な装置で容量計算を例示するのがよい。

① 磁気テープの記憶容量

- テープの全長
- ギャップ（ＩＲＧ）の長さ
- 記　録　密　度
- ブロックの大きさ

② 磁気ディスクの記憶容量

磁気ディスクの容量は装置によって異なった計算法を必要とするので，一般的な決定要因を説明し，具体的な装置で容量計算を例示するとよいであろう。容量の決定要因として，次のようなことを説明する。

- キーの有無
- トラックの容量（実効容量）
- レコードの大きさ

(3) アクセス・コスト

アクセス・コストに関し重要な要因を説明する。

① 磁気テープ

アクセス・コストの要因として次のことを説明する。

- ブロック化係数
- 入出力回数
- バッファ領域の大きさと個数
- データ転送速度
- テープ送り速度
- 記録密度
- 巻戻し時間

② 磁気ディスク

アクセス・コストの要因として次のことを説明する。

- シーク・タイム(seek time)
- サーチ・タイム(search time)
- 記憶容量

指導上の留意点

(1) ファイル・システムを構成するためのハードウェアの原理を理解させながら，さらにシステムの効率や経済性についての問題意識を高める。

(2) ファイルに関係するハードウェアの技術動向にも注目させる配慮も必要である。

参考文献

[1] 喜安善市編 「入出力装置」 共立出版，1967

[2] 森宗正ほか 「ハードウェア入門」 共立出版，1971

[3] Aba-alla, Meltzer, "Prinelples of Digital Computer Design", Prentice Hall, 1976

[4] Robinson, "Basic Principles of Digital Computers", Reston, 1974

[5] Wells, "Computer Systems Hardware", Cambridge University, 1976

[6] L．ナシェルスキー著 北川節訳 「電子計算機の基礎 ―電子計算機入門1」，培風館，1965

[7] L．ナシェルスキー著 北川節訳 「電子計算機の基礎 ― 電子計算機入門2」，培風館，1978

第3章　ファイル・システムの構成および構造

キーワード

データ構造（data structure），データ項目（data item），フィールド（field），集団項目（group item），繰返し項目（repeated item），集団関係（group relation），レコード（record），ファイル（file），線型構造（linear structure），木構造（tree structure），階層構造（hierarchical structure），ネットワーク構造（network structure），順編成ファイル（sequential file），索引順編成ファイル（indexed sequential file），直接編成ファイル（direct file），ファイル・ディレクトリ（file directory），論理構造（logical structure），物理構造（physical structure），ページ（page），ブロック（block），分類（sort），併合（merge），索引（index），索引ブロック（Index block），順次アクセス（sequential access），直接アクセス（random access），ランダマイジング（randomizing），同義語（synonym），中央二乗法（mean squarc method），除算法（dividing method），トランケーション法（truncation method），おりたたみ法（foiding and adding method），基数変換法（radix transformation method），開放アドレス方式（open addressing），バケット分割法（bucket partiton），オーバフロー・チェイン方式（overflow chaining），リスト表現（list representation），ポインタ（pointer），単純リスト（simple list），逆リスト（inverted list），多重リスト（multi list），リング構造（ring structure）

目　標

ファイル・システムは大量のデータを効率的に編成し，統一的に管理する機能を有する。これは，オペレーティング・システムのデータ管理機能を補完し，データベース管理システム（DBMS）の物理的な側面における基本的な機能を構成する。情報の論理的構造は，ファイル・システムの管理する物理的構造に変換される。

この章では，情報の論理的な構造を整理しファイル編成，記憶構造との対応を説明する。ファイル編成は，ファイル装置の使用法を標準化したものであり，利用者は，ファイルの生成や更新の目的に合うファイル編成を選択する。本章では，順編成ファイル，索引順編成ファイルの特徴を明らかにする。

また，この章では直接編成ファイルについて説明する。直接編成ファイルとは，レコード中のキー値から，そのレコードをファイル装置中のどの位置にレコードを蓄積するかを，索引やディレクトリなどを使わずに決められるファイル編成である。これは，ファイルの処理をランダムに行ないたい場合，とくにオンライン処理のような場合に最も適している。なお，直接編成で最も大きな問題は，同義語（synonym）と呼ばれる現象であるが，これについても説明する。

内　　容

3.1 データ構造

(1) データ構造の構成要素

ここではデータ構造を構成する基本的な要素を説明する。

データ構造を構成する基本的な構成要素は次のとおりである。

(1) データ項目

(2) 集団項目

(3) 集団関係

(2) データの論理的構造

データの基本的な構成要素は，多様に結合されて複雑な構造を表現する。データの結合のしかたにより，データ構造を分類し，明確にする。データ項目間の結びつきは，表現しようとする現実世界の対象（事象：entity）の性質を記述するものとして考えられ，通常，記録内関係（intra-record relationship）と呼ばれる。

これに対し，事象間の関係は記録間関係（intra-record relationship）と呼ばれ，通常は集団関係で表現され，多くのデータベース管理システム（DBMS）で中心的役割を果たす。

事象内関係は，COBOLのデータ記述法の中などに例としてみられる。

事象間の関係は次のような構造に分類される。

① 線形構造

② 木 構 造

③ ネットワーク（グラフ）構造

④ 階層構造

ここでの説明には，図3－1のようなデータ構造図を提示すると明確になるであろう。

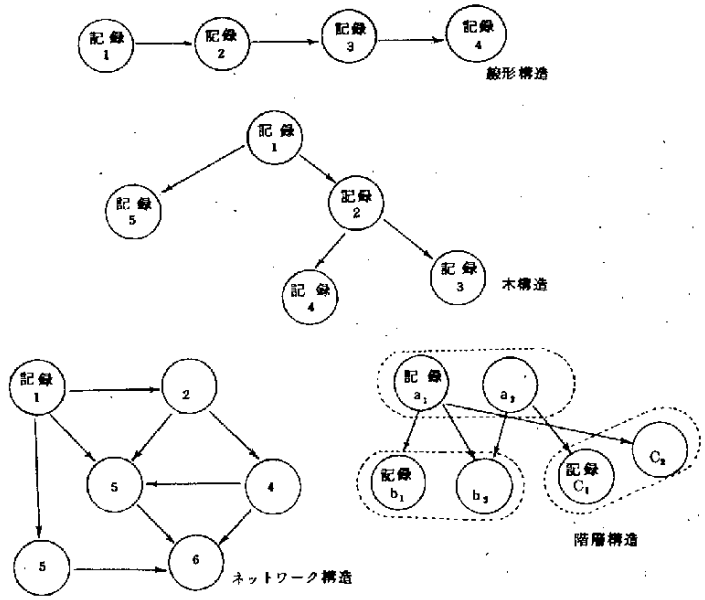


図3－1　データ構造

3.2　ファイルの構成

(1)　ファイルの構成方法

ファイルの構成のしかたを学ぶ方法の一つとして，データがどのように発生し，どのように処理されるかを知る必要がある。ここでは具体的なファイルを一つの例として，ファイルの構成要素と構成の方法を説明する。

ファイルの構成要素は，論理的なレコードである。レコードは，フィールド（データ項目）からなっており，レコード間には陰に陽に関係がある。

ファイルの編成は，順編成ファイル，索引順編成ファイル，直接編成ファイルに大別される。

また，ファイルには，直接の構成要素となるデータの他にファイルの属性を示す情報が必要であり，これはメタ・データと呼ばれる。このメタ・データの集まりを，ファイル・ディレクトリと呼ぶ。

ファイル・ディレクトリで管理される情報には次のようなものがある。

① ファイルの名前

② ファイルの作成者

③ ファイルのアクセス：キー情報

④ ファイルの作成日付，有効期限

メタ・データの取り扱いについては，第5章のデータ・ディクショナリ／ディレクトリ（DD／D）で詳しく説明する。

(2) ファイルの物理的構造

記憶装置上に設定された何らかのまとまりのあるデータの格納領域をファイルという。

オペレーティング・システムのデータ管理では，物理的な入出力単位で記憶装置に読み書きを実行する。また，いくつかの概念を用いて，物理的ファイルを利用者が制御しやすいような構造を導入することが必要となる。

ファイルを構成するレコードの概念，レコードの要素であるフィールドとの関係を明確にし，データの論理的構造と，物理的記憶構造の対応を考える。

なお，物理的記憶領域のページ分割，ブロック化などの基礎的な概念も提示し，ファイル編成への導入説明を行なう。

3.3 順編成ファイル

順編成ファイルは，従来からよく使われてきたファイル編成である。順編成ファイルに含まれるレコードは，レコードの持つ何らかの制御フィールドの値と同じ順で並べられている。ここでは，順編成ファイルの構造を説明し，典型的な例としてテープ・ファイルを取りあげる。

(1) 順編成ファイルの生成と更新

ここでファイルの生成とは，入力データからファイルを構成することをいい，ファイルの更新とは，フィールドやレコードの変更，追加，削除などのファイルの修正処理をいう。

ここでは順編成ファイルの生成や更新における問題点をとりあげ，順編成ファイルの特徴を理解させる。

① 順編成ファイルの生成

レコードに含まれている情報に従って並べられた順番に，レコードをファイル媒体に蓄積する。順編成ファイルの生成や更新にあたって基本的な操作となる。分類／併合の説明をもあわせ行なう。

② 順編成ファイルの更新

トランザクションを同じ順番に並べかえて突き合せを行なう。

(2) 順編成ファイルの読み出し

順次アクセスは，レコードの並んでいる順に読む方法であり，バッファリング技法の用いられることが特色であろう。順次アクセスを行なう以外は，一般にファイルの読み出しは遅い。

ここではこのような順編成ファイルの読み出し処理時の問題について説明する。

(3) 順編成ファイルの特徴

これまでの説明を総合して，順編成ファイルの利点，欠点をまとめる。

① 利　　点

・生成・更新が容易

・アクセスが速い

・記憶容量が少なくてすむ

② 欠　点

・順次アクセスしかできない

3.4 索引順編成ファイル

索引順編成ファイルと順編成ファイルとは

・索引順編成ファイルは主に磁気ディスク装置におかれること

・レコードの検索は索引を通してなされること

の二つの点で異なる。図３－２のような例で索引順編成ファイルを説明する。

ファイル処理にあたっては，一つのレコードを取り出すために，最小限２回の入出力命令が必要となる。また，多くのシステムでは，索引を主記憶装置上に常駐させることにしている。

(1) 索引順編成ファイルの生成と更新

索引順編成ファイルの生成時に，データ・ブロックとともに一般に索引ブロックも作成される。生成時に作成された索引ブロックの構造が，その後のファイルの更新処理において重要な役割を持つ。

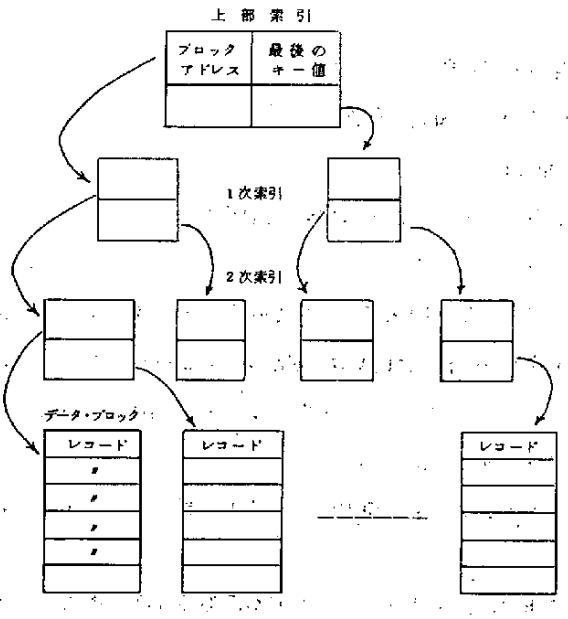


図３－２　索引順編成ファイル

索引ブロックが作成されながらファイルが生成されていく過程を例で示す。

索引ブロックを作成するためには

① 初期生成時のデータをもとにして作る

② 何らかのパラメタ指示によりあらかじめ作成される

③ ファイルの追加／削除時に常に作成する

など，いくつかの手法があり，これら手法について，その特徴を説明する。

索引順編成ファイルの更新は，フィールドの値を変えるようなレコードの変更に対しては，容易に行なわれる。

レコードの削除は，ファイル中のレコードに削除印を記入するような方法がとられる場合と，物理的に空間をあけわたしてしまう場合とがあり，これらの特徴を説明する。

レコードの変更によってレコードの長さが変るような場合には，そのレコードをもとの場所に書き戻すことはできないが，このような場合の処理について，次の取り扱いを説明する。

・追加レコードとしての取り扱い

・別の領域に書いておき，リンクを持たせる。

・ファイル中のレコードを再配置する。

索引順編成ファイルでは，レコードの追加は面倒であるが，レコードはキー値の順に配置するのが原則であるから，追加レコードも適当な位置に入れる必要がある。

また，あふれ領域が，頻繁に参照されるようになってきた場合は，ファイルの再編成の必要がある。

ファイルの再編成を最小限にするには，

・ファイル生成時の空き領域の確保

・あふれ領域の配置

などを考慮する必要がある。これらについて説明する。

(2) 索引順編成ファイルのアクセス

索引順編成ファイルでは，順次アクセスと直接アクセスの両方ができる。レコードのアクセスに際しては，索引レコードと目的レコードとの最小2回のアクセスが必要となる。このために索引レコードのアクセスができるだけ少なくなるような配慮が必要であり，これを説明する。

(3) 索引順編成ファイルの特徴

これまでの説明を総合し，索引順編成ファイルの利点，欠点をまとめる。

① 利　点

・順次アクセス，直接アクセスの両方が，ある程度効率よくできる。

・生成の効率がよい。

・部分的に更新できる

② 欠　　点

・あふれが多くなると，更新やアクセスが遅くなる

・再構成が必要

・レコードのアクセスに，2回以上の入力操作が必要

3.5 直接編成ファイル

レコード中のキー値からファイル中のレコードの位置が決められる直接編成は，キーから直接入出力命令に結びつけられるため，ファイルの中から特定のレコードをアクセスする手段として，最も優れている。また，乱処理を行ないたいような場合には適しているが，一般に順次処理には向いていない。

順次処理の効率をあげる手段としては，ブロッキングがあるが，直接編成ファイルにはブロック内のレコードが，キー順を守るという保証はない。

(1) キー変換方式

ここでは，レコードのキー値からアドレスを決めるキー変換方式について代表的なものを説明する。

① 中央二乗法

キー値を二乗して，その中央部を取り出し，これを格納アドレスとする方法。この方法は各桁の効果がいくつかの桁の範囲にひろげられるので，おりたたみ法よりも一様化にかなり有効であり，多くの場合，満足すべき結果が得られる。

② 常数をかける法

キー値にある常数をかけて，その結果の中央部分を取り出して，それを格納アドレスとする方法。

③ 除　算　法

キー値を，ある数で除算して，その余りをアドレスとして使用する方法。この方法はキーの長さが不定だったり，かたよりが大きい場合に良い結果が得られることがある。

④ トランケーション法

除算法と違って商を利用する方法である。除算法やおりたたみ法は，いずれももとのキーいかんによって，必ずしも一様に分布するという保証はない。実際に行なう場合には，除数をいろいろ変えてみるとか，いくつかの方法を組み合せるとかして，最も良い方法を選ばなければならない。

⑤ おりたたみ法

乗除算を使わずに加算とシフトだけでキーを圧縮するランダム化の方法。中央二乗法より有効でないが，簡単な方法である。

以上のほか，基数変換法，テーブル・ルック・アップ法，抽出法などがあることを触れる。

(2) 同 義 語

直接編成で最も大きな問題は同義語（synonym）が生まれる現象である。

キーを一様化したとき，格納アドレスが重複することは当然起り得るため，その処理は十分検討しておかなければならない。これらの重複は一般にさけられないものである。これに対処する方法として次の方法を説明する。

① 開放アドレス方式

キー値を無作為化し，各レコードを圧縮キーで指定されるアドレスに，一つずつレコードをいれる。重複があれば，このアドレスに続く最初の空アドレスに格納する。このようにして，全レコードを格納する。

② バケット（bueket）に分割する法

記憶装置全体をバケットと名づける一定大きさのブロックに分割し，バケット番号の選択を圧縮キーで決める。バケットの中は順次にさがす。

③ オーバフロー・チェイン（overflow chain）方式

オーバフロー領域に格納し，アドレス・チェインをつける。

同義語の発生率に影響を及ぼす因子としては，次のものがあることを説明する。

・キーの分布

・変換方法

・ファイル使用率

・バケットの大きさ

(3) 直接編成ファイルの特徴

① 利　　点

・アクセスが早い

・更新が早い

② 欠　　点

・空き領域ができる

・あふれが多くなると，アクセスが遅くなる

・キーの変換式を選ぶことがむずかしい

(4) リスト表現

ここでは，その他のファイル編成法としてのリスト表現についてその技法，特徴を理解させ

る。

① リスト構造

リスト構造とは何か，図3－3のような例を用いて復習する。

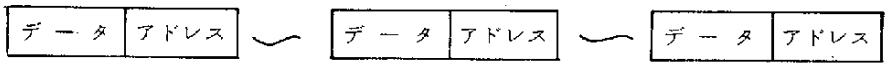


図3－3 リスト構造

② リスト構造の種類

リスト構造の種類について簡単に復習する。

① 単純リスト構造

② 両方向リスト構造

③ リング構造

各リスト構造について，どのような使用ができるかを復習する。また，リング構造の組み合せにより，複雑なレコード間の関係が表現できることを示す。

③ リスト構造の操作

リスト構造になっているデータを取り扱う時に，どのような操作が必要かについて，データの生成，追加，更新，削除に分けて簡単に復習する。

④ 多重リスト構造

多重リスト構造およびその特徴について復習する。

指導上の留意点

効率のよいファイル・システムの編成に必要なデータの論理構造，ファイル編成方法の選択などの基礎的事項を理解させながら，ファイル・システムの分析やデータベース管理システムとの関連に留意する。

参考文献

[1] 浦 昭二 「データ構造」 共立出版

[2] 上條史彦 「データベース・システム」 産業図書

[3] J. Martin, "Computer Data Base Organization", Prentice-Hall, 1977
（江村潤朗訳「データベース」，日本コンピュータ協会，1977）

[4] Horowitz, Sahni, "Fundamentals of Data Structures", Computer Science, 1976

[5] Cagan, "Data Management Systems", Melville, 1973 shing, 1973
(西村恕彦訳 「データベース入門」, 近代科学, 1975)
[6] Rustin, ed., "Data Description, Access and Control", ACM, 1974
[7] Curtis, "Access Mechanisms and Data Structure Support", QED Information Sciences, 1975

第4章　ファイル・システムの分析

キーワード

記憶単価(cost per memory unit)，アクセス時間(access time)，ファイル容量(file capacity)，ファイル編成(file organization)，多重索引(Multi-index)，多重リスト(Muti-List)

目　標

ファイル・システムを設計する時には，アプリケーションの種類とファイル装置の評価が重要である。

この章では，いかなるファイル編成をとるべきか，あるいは，いかなる装置を選ぶべきかについての評価方法について理解させることを目標とする。

内　容

4.1　ファイル・アクティビティの評価

(1)　考察すべき要因

いかなる装置を使うかを決める場合には，次のような要因を考えなければならない。

①　ビットあたりの記憶単価

②　アクセス時間

③　転送速度

④　容　量

⑤　オペレーション・タイム

このほか，制御プログラムや言語処理系，さらには信頼性なども考慮に入れなければならない。

この節では，このようなファイル・システムのための装置を選択する際に考慮すべき点について説明する。

(2)　ファイル装置の選択のステップ

装置の選択に当たっては，次のステップが必要であることを説明する。

①　ファイルのレイアウトと構成

アプリケーションに最適だと思われるファイルのレイアウトや構成法を決める。第3章で学んだ各ファイル編成の知識を基礎としてレイアウトや構成の決定について説明する。

② アクティビティ

アクティビティの分析について説明する。

- トランザクション量の見積り
- 入出力処理の回数
- アクセス時間

③ 効率上の必要条件

効率上の必要条件の設定について，説明する。

- 単位時間当りの処理量
- 将来の拡張性の予測

④ シミュレーション

ファイル・システムのシミュレーションによる評価について説明する。

4.2 応答時間

ここでは各ファイル装置の応答時間の計算方法について説明する。

① テープの読み書き時間

② 磁気ディスクのアクセス時間

4.3 その他の考慮

ファイル・システムにおいては，ファイルの共用や機密保護のために，次のような考慮が必要となってくる。

① ファイル利用の衝突

② セキュリティ・チェック

③ 復旧の為の情報保守

これらの要求は効率の観点に相反するものであり，この間のトレード・オフをどう決めるかも重要である。この事に関する基本的な考え方を説明する。

4.4 記憶管理技法の分析

ファイルの編成方式の違いによる記憶管理技法の相異を説明するとともに，アクセスの即時性のための考慮事項について説明する。

関係するレコードのアドレスをすぐに得ることができる索引やリストについて，その技法を分析説明する。

利用者の幅広いデータ要求に対しては，多重索引や多重リストなどの技法があり，これらの技

法についてその利点，欠点を説明する。

速いアクセスを要求する事とデータの完全性，同期性を保つ事とは，相反する要求になる場合がある。なお，多重リストや多重索引の保守上の問題点等について説明する。

4.5 ファイル・システムの評価

この節では，統合されたファイル・システムの評価の要素について説明する。

① 費　用

ファイル編成，生成，維持の費用

② 機　器

③ ファイル処理の為の言語

④ 要求されるファイル構造，形式

⑤ 実行効率

⑥ 保　守

⑦ 修　正

⑧ 利用者の経験

評価の要素に対する重みづけ，定量化の手法についても説明するとよい。

指導上の留意点

ファイル・システムを構成するハードウェアの評価を基礎にしながら，経済性または効率のよいファイル・システムを設計することの重要性が認識できるよう配慮する。

参考文献

[1] CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management System", 1971
（西村恕彦ほか訳「データベース・システム」, bit, 臨時増刊, 共立出版, 1973）

[2] Cagan, "Data Management Systems", Melville, 1973
（西村恕彦「データベース入門」近代科学, 1975）

[3] J. Martin, "Principles of Data Base Management", Prentice-Hall, 1976

[4] Kaimann, "Structured Information Files", Melville, 1973

[5] London, "Techniques for Direct Access", Mason Charter, 1973

第5章　データベース管理システム

キーワード

データベース管理システム（DBMS：data base management system），データ記述言語（DDL：data description language），スキーマ（schema），サブスキーマ（sub- schema），データ操作言語（DML：data manipulation language），ディレクトリ管理（directory management），データ定義（data definition），データの生成（data generation），問合せ処理（query Processing），更新処理（updating），データの独立性（data independency），汎用データ管理システム（generalized data management system），親言語方式（host language system），専用言語方式（independent language system），ネットワーク型DMS（network-type data management system），階層型DMS（hierarchical data management system），逆型DMS（inverted data management system）

目　標

データベース管理システム（data base management system，DBMS）は，ファイルの生成やファイルの維持，報告書作成のためのソフトウエア・システムであって，これらをすべて一般化した方法で行なうものである。このためにそのシステムは，一般性，融通性，適応性，制御性をそなえていなければならない。

データベース管理システムは，これまでのプログラム（アプリケーション）中心のシステム作りから，管理されるデータ中心のシステム作りを指向するものであり，そのためにいくつかの特徴を持っている。

まず，データ記述が，データ操作とは独立して行なわれる。またデータ操作は，はば広い利用者の使用を前提とし，使いやすさが工夫されている。また，独立したデータ記述はさらに一般的なディレクトリ管理の技術の一環としてとらえられている。

本章では，データベース管理システムについて，その機能を説明する。

現在，世に出ているデータベース管理システム（ＤＢＭＳ）にはいろいろなものがある。ファイルを検索して簡単な報告書を作成するものから，複雑なファイル構造を処理してオンライン方式で動くものまで多岐にわたっている。このような情況下では利用者が，利用可能なシステムを

評価選択することが困難ともなる。

したがって，現在市場に出ている汎用データベース管理システムの代表的なものを取りあげて，分類し説明することを目標とする。

なお，本章の説明を，第10章の「総合システム」において，さらに補足する。

内　　容

5.1　データベース管理システムの機能

データベース管理システムの持つ機能について説明し，それらの機能と意義を理解させる。

データベース管理システムの機能を説明する上で，図5－1のようなシステム概念図を示すとよい。これはCODASYL の報告書に示されているものである。

まず，データベース管理システムにみられる主要な機能モジュールを説明する。次のようなことがらを，簡単に解説するとよい。

① データ定義

② データ（ファイル）の生成

③ データ（ファイル）の修正

④ 問合せ

⑤ 更　新

⑥ 完全性

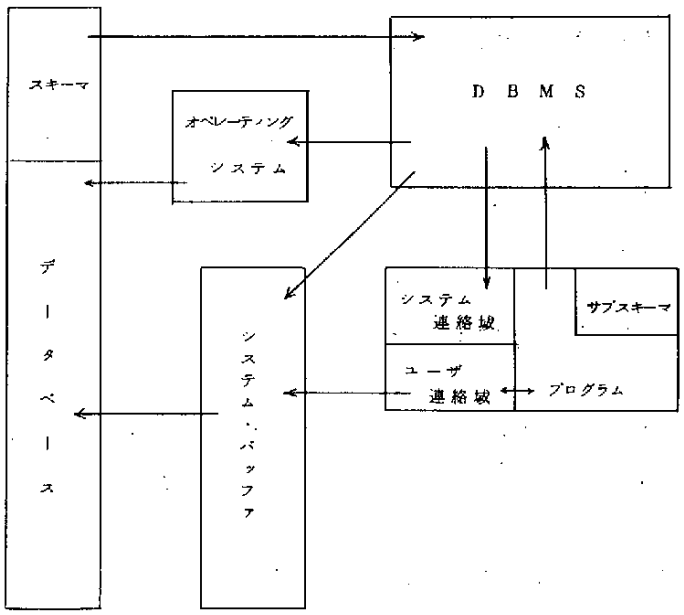


図5－1．DBMS の概念

5.2 データ記述

この節ではデータベース管理システムの最も特徴的な位置を占めるデータ記述（定義）についてその考え方や機能について説明する。

データ記述の考え方やその記述言語は，個別のシステムによっていろいろあるが，ここでは，CODASYL 提案のデータ記述言語（ＤＤＬ）を具体例として説明するのがよい。

データ記述（定義）の普遍的な考え方について，次のような主題に従って説明するとよい。

① データ独立性とは

② データベースの定義

・スキーマ

・サブスキーマ

③ スキーマ，サブスキーマが必要な理由

④ データ記述言語の機能

・データ項目

・レコード

・セット（レコードの集まりに名前をつけたもの）

・領　域

⑤ データ構造の表現に使うBachman 線図

5.3 データ操作

この節ではデータベースを利用し，処理する機能を分担するデータ操作について説明をする。データ操作機能は，専用言語システムと親言語システムで表面的には異なったユーザ・インタフェースを提供している。

データベース・システムの動きをよく理解するためには，データの基本操作が目に見える親言語の命令語で説明するのがよい。

代表的な例としては図5－2のようなCODASYL のデータ操作言語（ＤＭＬ）をとりあげる。ここでいうデータ操作言語はそれ自体で完結した言語ではなく，データベースの操作に関するサブ言語と呼ばれるものである。このデータベース操作サブ言語は，次の要素からなることを説明する。

(1) 動　作

動作には基本的に次の三つがあることを説明する。

① 検　索

・データベース中のデータの位置ぎめ

・データのアクセス

② 変　更

・データの追加

・データの削除

・データの置き換え

・データ相互間の関係の変更

③ 制　御

・開始指示

・終了指示

・スキーマとの関連領域

(2) 目的データやサブ・スキーマの関連領域

データ操作言語の作用過程における目的データやサブ・スキーマの関連を説明する。

プログラムとデータベースとの間での目的データの受け渡しに関する関連領域を説明する。

(3) 選択基準

① 位置による選択基準

・ファイルの先頭

・ファイルの最後

・レコード型

・レコード間の関係（セット）

② 内容による選択基準

・論理選択式

(4) データ操作命令の実行中に，異常が発見された場合の処理

① 検出された誤りに関する十分な情報の入手

② 適切な行動をとるための制御の復帰法

(5) 機密保護

① 機密保護の手段

② 機密保護のレベル

5.4 ディレクトリ管理

この節では，総合情報システム，とくにデータベース・システムの運用管理手段として要求されてきているデータ・ディクショナリ／ディレクトリ（ＤＤ／Ｄ）について説明する。

ＤＤ／Ｄは，現在ではデータベース管理システムに統合される傾向にある。ここではそのよう

ACCEPT
CONNECT
DISCONNECT
ERASE
FIND
FINISH
FREE
GET
KEEP
MODIFY
ORDER
READY
REMONITOR
STORE
USE

図5－2　CODASYLデータ操作言語

な傾向を念頭に置いて，次のようなことを説明する。

(1) データベース・システムとDD／D

DD／Dはメタ・データと呼ばれるものの集中管理をめざしている。

メタ・データとは次のようなものである。

① データの名前

② データの表現，構造

③ データの所在と使い方

④ データの意味

⑤ データの発生源

⑥ データの収集責任者，使用権，有効期限

このようなメタ・データを管理する必要が何故起きてきたのかをデータベース・システムの必要性とともに明らかにする。

次のようなことがポイントとなろう。

① データの共用性

② データの編成の多様性

③ データの使用の多様性

④ 共用データの完全性，整合性

⑤ システムの仕様の変更に対する対処

ＤＤ／Ｄの持つデータ・ディクショナリ機能とデータ・ディレクトリ機能を明確にし，ＤＤ／Ｄの目標を具体的に説明する。

(2) ＤＤ／Ｄの構成

ＤＤ／Ｄが管理するメタ・データに関しその特性を分類して説明する。

① データの識別属性

② データの表現に関する情報

③ データの物理的情報

④ データの関連を示す情報

⑤ 機密保護に関する情報

⑥ 完全性に関する情報

(3) ＤＤ／Ｄシステムの機能

ＤＤ／Ｄシステムの機能には，およそ二つの側面がある。

① 定義情報の収集，提供

② データの使用に関する標準化

また，ＤＤ／Ｄがもつべき機能を説明する。

① データに関する種々の定義づけ

② データの矛盾性の発見と削除

③ データの冗長性の発見と解決

④ 管理情報の一元性の保証

⑤ 問合せ機能

⑥ ドキュメンテーションの標準化

⑦ データベース管理システムとのインタフェース

⑧ 安全性，完全性の保証

(4) ＤＤ／Ｄの現状と将来

ＤＤ／Ｄは，データ管理システムや言語プロセッサあるいはオペレーティング・システム等と深く関係しており，ＤＤ／Ｄの管理するメタ・データは共通の資源として利用できなければならない。現在は，そのような目的に沿ったシステムはまだない。

ここでは特にデータベース管理システムとの共存関係を中心にＤＤ／Ｄのシステム全体の説明をする。

ＤＤ／ＤとデータベースＤ管理システムとの共存関係の型や，ＡＮＳＩ／Ｘ３／ＳＰＡＲＣ の出し

ているデータベース・システムのアーキテクチャに関する説明などをするとよい。

① DD／DとDBMSの関係

・分　離　型

・統　合　型

・中　間　型

② ANSI／X3／SPARC の概念

・概念スキーマ

・外部スキーマ

・内部スキーマ

情報システム全般を，DD／Dの視点からみなおすことは，大きな意味があろう。

5.5 データベース管理システムの分類

現在，市場に出ている汎用データベース管理システムは多種あるが，これらの分類の基準は明白でない。ここでは，いくつかの分類の基準を示す。

(1) 機能による分類

データベース管理システムが提供している機能の観点から分類すると，次のようになる。

① データ検索システム

② ファイル管理システム

③ 複合ファイル・システム

④ テレ・プロセッシング・モニタ

⑤ データベース管理システム

⑥ オンライン・データベース・システム

⑦ 特別な目的のシステム

(2) 言語体系による分類

汎用データベース管理システムが提供している言語の体系から分類すると，次のようになる。

① 親言語方式（host language system）

・コンパイラ拡張型

・プリ・コンパイラ型

・サブルーチン・コール型

② 専用言語方式（independent language system）

・インタプリタ型

・プログラム生成型

・プログラム・コンパイル型

・固定ルーチン型

(3) データ構造による分類

システムが提供するデータ構造からの分類を示す。

① ネットワーク型（CODASYL／DBTG型）

② 階層型（IMS型）

③ 逆　型（ADABAS型）

(4) その他の分類

次のような，その他の分類の観点を説明する。

① システムが開発されている環境

② システムの操作方式

③ 利用者に提供される言語のタイプ

④ サービスされるデータの論理構造

⑤ データ独立性の程度

(5) 汎用データベース管理システムの現状と将来

データベース管理システムの開発の歴史の説明を含めて，以上の分類の観点を説明するとよい。

データ管理システムの開発史には大きく次のような系統がある。

① ＲＰＧの流れ

② MARKⅣの流れ

③ ＧＩＳの流れ

④ CODASYL／DBTGの流れ

⑤ ＩＭＳの流れ

⑥ 逆ファイルの流れ

しかし，現在は，これらの汎用データベース管理システムの分類はあまり意味がなくなりつつある。お互いのデータ管理システムは各々相手の良い機能を取り込んでいく傾向にあり，システムの根底となるデータの構造やアクセス手法の違いによる効率の面に，差が現われる傾向にある。

このようなデータベース管理システムの将来動向が説明できるとよい。

(6) 汎用データベース管理システムの機能

次に汎用データ管理システムを理解する上でポイントとなる機能について説明する。それらの機能の意味，なぜその機能が問題となるかなどを次のような点から説明する。

① データ構造

② データ定義機能

③ 問合せ機能

④ 更新機能

⑤ 生成機能

⑥ プログラム機能

⑦ データ管理者機能

⑧ 記憶構造

⑨ 操作上の特性

5.6 汎用データベース管理システムの特徴

ここでは，現在市場に出ている汎用データ管理システムの中で代表的なものを取りあげてそのシステムの特徴を説明する。説明する際の項目としては，次のようなものがある。

① 開発メーカ

② ファイル構造

③ 応用言語

④ データベース言語

⑤ アクセス手法

⑥ 可変長レコード

⑦ データベース・セキュリティ

⑧ システム・アカウンティング

⑨ チェックポイント・リスタート

⑩ 互換性

⑪ バッチ／オンラインの並行処理の可能性

⑫ 応用プログラムの並行処理の可能性

⑬ 問合せ機能

⑭ 報告書作成機能

⑮ データ・ディクショナリ

⑯ テレコミュニケーション機能

1. ADABAS

① ソフトウェアAG社

② ネットワーク構造，逆構造

③ COBOL, PL／I, FORTRAN, ASSEMBLER

④ ADABAS

⑤ BDAM

⑥ 可

⑦ パスワード

⑧ 統計レポート

⑨ 可

⑩ 15 レベルまでのデータ保護

⑪ 可

⑫ 可

⑬ ADASCRIPT問合せ言語

⑭ ADAWRITER

⑮ DATAMANAGERとのインターフェース

⑯ TPモニタ

2. IMS

① IBM

② 階層構造

③ COBOL, PL／I, ASSEMBLER

④ DL／I

⑤ VSAM, SAM, ISAM, OSAM, BSAM

⑥ 可

⑦ パスワード

⑧ システム・ログ

⑨ IMS／DCのみ

⑩ バックアウト，リカバリの為のロギング

⑪ 可

⑫ 可，DB／DC モード

⑬ IQF, GIS／VS

⑭ GIS／VS, GIS－2

⑮ FDP

⑯ CICS, IMS／DC

3. DMS1100

① UNIVAC

② ネットワーク構造（CODASYL型）

③ COBOL，FORTRAN，PL／I

④ DDL，DML

⑤ CODASYLアクセス手法

⑥ 可

⑦ 有

⑧ 有

⑨ 有

⑩ オーディット・トレイル（Audit Trail），ビフォア／アフタイメージ（Before／After Image）

⑪ 可

⑫ 可

⑬ Query Language Processor

⑭ COBOL Report Writer，QLP／Report Writer

⑮ 可

⑯ 可

4. SYSTEM2000

① 三菱総研

② 階層構造，ネットワーク構造

③ FORTRAN，COBOL，PL／I，ASSEMBLER

④ DDL，IMMEDIATE

⑤ 標準のＩＢＭアクセス手法

⑥ 可

⑦ パスワード

⑧ ログ

⑨ 可

⑩ トランザクション・ログ

⑪ 可

⑫ 可

⑬ システム2000 Query／Update 機能

⑭ 有

⑮ DDL

⑯ TP2000, CICS, TSO

5. TOTAL

① Cincom Systems

② ネットワーク構造

③ COBOL, FORTRAN, PL/I, ASSEMBLER, RPG II

④ DBDL, DML

⑤ BDAM, DAM

⑥ 可

⑦ 無

⑧ 無

⑨ TPモニタ

⑩ 可

⑪ ロギング

⑫ 可

⑬ 可

⑭ 無

⑮ SOCRATER

⑯ ENVIRUN/1, CICS, TASK/MASTER

指導上の留意点

(1) データベース管理システムの特徴を理解させながら，データベース分野における技術的動向や標準化の動きを取り入れて説明し，より有効なデータベース・システムの設計が行なえるよう留意する。

(2) 市場に出ている汎用データベース管理システムの資料は入手の困難なものもあるので，ここでは，身近に存在するシステムで例示してもよい。

(3) ここであげたシステム以外にも次のようなものもみてみるとよい。

・MARKⅣ

・GIS

・IDS

・IDMS

参考文献

〔1〕 CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management System", 1971
（西村恕彦ほか訳「データベース・システム」 bit 臨時増刊, 共立出版, 1973）

〔2〕 CODASYL DBTG, "Data Description Language", Journal of Development, 1973
（データベース言語研究会訳 「データベース用データ記述言語」 情報処理学会, 1977）

〔3〕 CODASYL Systems Committee, "Selection and Acquisition of Data Base Management Systems", 1976
（西村恕彦監訳 「データベースの選び方」 bit 臨時増刊, 共立出版, 1977）

〔4〕 Cagan, "Data Management Systems", Melville, 1973
（西村恕彦訳 「データベース入門」 近代科学, 1975）

〔5〕 J. Martin, "Computer Data Base Organization", Prentice-Hall, 1977
（江村潤朗訳「データベース」 日本コンピュータ協会, 1977）

〔6〕 C. J. Date, "An Introduction to Data Base Systems The systems Programming Serieg", Addison-Wesley, 1975

〔7〕 Lefkovitz, "Data Management for On-line Systems", Prentice-Hall, 1976

〔8〕 "Guide to Data Base Management", Auerbach, 1975

第6章　コミュニケーション・システムの基本構成

キーワード

コンピュータ・システム(computer system)，通信制御装置(communication control unit)，通信回線(communication line)，端末装置(terminal device)，デュプレックス・システム(duplex system)，デュアル・システム(dual system)，バッファリング(buffering)，誤り制御(error control)，コード変換(code conversion)，伝達制御(transmisson control)，変復調装置(MODEM)，網制御装置(network control unit)，分岐装置(branch unit)，回線インタフェース(line interface)，回線制御(line control)，入出力制御(input/output control)，入出力装置(input/output device) データ伝送回線(data transmission circut)，単向通信方式(simplex communication)，半二重通信方式(half-duplex communication)，全二重通信方式(full-duplex communication)，2線式回線(two-wires circuit) BPS(bet per second)，ボー(baud)，変調(modulation)，変調速度(modulation rate)，位相変調(phase modulation)，振幅変調(amplitude modulation)，同期(synchronization)，同期式(synchronous)，非同期式(asynchronous)

目　標

コミュニケーション・システムを構成するハードウェアは，一般に，センタ機器，通信回線系各種装置(通信回線，変復調装置等)，端末装置に大別される。また，コミュニケーション・システムを別の観点から眺めると，データの処理を行なうデータ処理系と，データの伝送を行なうデータ伝送系とに分類することができる。データ処理系には，センタ機器のうちの中央処理装置や周辺装置が含まれる。近年，端末装置のインテリジェント化に伴い，端末装置の機能のうちの一部をデータ処理系に含めることもある。データ伝送系には，センタ機器の通信制御装置のほか，通信回線，変復調装置，端末装置などが含まれる。

コミュニケーション・システムでは，情報をある場所から他の場所へ送るためのデータ伝送路が重要な役割を果すが，データ伝送路上の情報の流れ方としては，基本的には三つの通信方式がある。現在，データ伝送路の主流はアナログ伝送路であり，コンピュータまたは端末装置とデー

タ伝送路との間で情報の授受のために変復調装置が使用される。変復調方式にはいくつかの種類がある。通信制御装置との間のインタフェースを理解することも必要である。

本章では，コミュニケーション・システムの基本構成をハードウエアの面からと，データ伝送系の面から理解させることを目標とする。

内　　容

6.1　コミュニケーション・システム

コミュニケーション・システムは，次のようなハードウエアから構成される。

(1) コンピュータ・システム

(2) 通信制御装置

(3) 通　信　回　線

(4) データ通信用諸装置

(5) 端　末　装　置

各々の機能の概要および位置付けについて，図6－1のような例をあげて説明する。

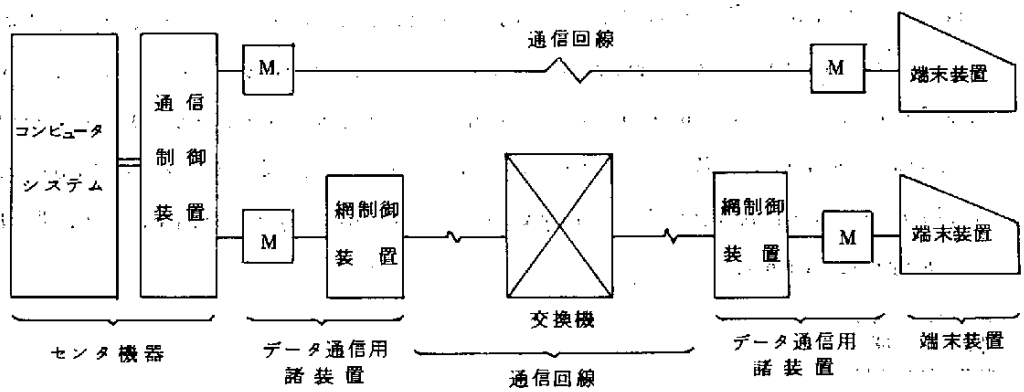


図6－1　コミュニケーション・システムの構成

(1) コンピュータ・システム

コミュニケーション・システムにおけるコンピュータ・システムは，データの処理，通信制御などを行なっている。これらは実際には，応用プログラム，通信制御プログラムにより実行されているためソフトウエアについて理解する必要があるが，ここでは，ハードウエアを中心に説明する。

コミュニケーション・システムで使用されるコンピュータ・システムのハードウエアとしては，以下の装置がある。

① 中央処理装置

② 直接アクセス記憶装置

③ システム制御卓（通信制御卓）

④ その他周辺装置（磁気テープ装置等）

各々について，その機能，用途を理解させる。また，コミュニケーション・システムの信頼性を向上させるためのコンピュータ・システムの障害対策についても次の事項を説明する。

- デュプレックス・システム
- デュアル・システム
- 多重プロセッサ・システム
- そ　の　他

(2) 通信制御装置

コミュニケーション・システムにおいてコンピュータに通信回線を接続するためには，通信制御装置が必要である。ここでは通信制御装置の役割，構成，機能について理解させる。

① 通信制御装置の役割

以下のような通信制御装置の役割について，理解させる。

- コンピュータ内部処理速度とデータ信号速度（ＢＰＳ）の差の吸収
- 通信回線上のデータの多重化（チャネルの節約）
- 直並列変換
- 通信回線の接続と監視，変復調装置の制御
- そ　の　他

② 通信制御装置の機能

以下のような通信制御装置の機能を理解させる。

- 文字の組み立て／分解
- バッファリング
- メッセージ処理
- 誤り制御
- コード変換
- 伝送制御
- 回線の監視および制御

ただし，これらの機能のすべてを通信制御装置が受け持つわけではなく，中央処理装置内のオンライン・プログラムが分担する場合もあるので，いくつかの事例をあげて説明するとよい。

また，通信制御装置の能力に応じて，バッファリングには，次の四つの方式があること

を理解させる。

- ビット・バッファ方式
- キャラクタ・バッファ方式
- ブロック・バッファ方式
- メッセージ・バッファ方式

③ 通信制御装置の構成

通信制御装置は図６－２のように一般に制御部，回線制御部，回線インタフェース部から構成される。

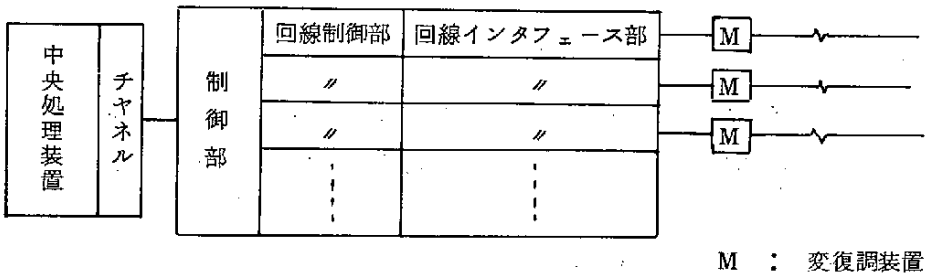


図６－２　通信制御装置の構成例

制御部，回線制御部，回線インタフェース部の機能について理解させる。

回線インタフェース部は，回線の規格に応じて，次のものがあり，用途に応じた選択が必要となる。

- 直流回線用
- モデム用（同期式，非同期式）
- データ・セット用
- ダイヤル用

また，近年，通信制御装置にプログラム機能を持たせ，通信制御処理装置として使用される場合も増加しつつある。そこで，通信制御装置についても説明すると良い。

(3) データ通信用諸装置

コンピュータ・システムと端末装置との間には，通信回線の他に，データ通信用の諸装置が設置されることがある。コミュニケーション・システムを設計する場合，データ通信用諸装置に関する知識は必須であり，その概要を理解させる。データ通信用諸装置としては，次のものがある。

① 変復調装置

ここでは，変復調装置の種類と特徴について理解させる。

② 中 継 装 置

中継装置には，時分割中継装置と周波数分割中継装置とがある。各々について，次のことを説明する。

- 機　　能
- 動作原理
- 種　　類
- 用　　途

③ 網 制 御 装 置

交換回線を使用する場合，交換機を動作させるために網制御装置（network control unit）が必要である。

- 機　　能
- 種　　類
- 関連機器との接続系統
- インタフェース
- 動作手順

④ 分 岐 装 置

通信回線を有効に利用するため，1本の通信回線に，複数の端末装置を接続することがある。この場合は，分岐装置が使用され，電電公社の局にある分岐装置を使用する場合と利用者が設置する場合とがある。

ここでは利用者が設置する分岐装置を中心に，次の事項について説明する。

- 機　　能
- 種　　類
- 関連機器との接続

(4) 端 末 装 置

端末装置は，通信回線網の末端に置かれ，人間とシステムとの接点にあたる。ここでは端末装置の構成，機能，種類について説明する。

① 端末装置の構成

端末装置は一般に以下の部分から構成される。

- 回線インタフェース
- 回 線 制 御 部
- 入 出 力 制 御 部
- 入 出 力 装 置

近年普及しはじめたインテリジェント・ターミナルは，このほかに小規模の中央処理装置や各種周辺装置を持っている。したがって，ここでは上記の各部分の機能について理解させるとともに，インテリジェント・ターミナルについても簡単に説明をする。

② 端末装置の機能

端末装置の機能としては次のものがあり，これらについて説明する。

- データの入出力
- 回線とのインターフェース
- 入出力の制御と操作手順の制御
- 伝送制御と誤り制御
- データのバッファリング
- データの処理

③ 端 末 の 種 類

端末装置を機能に応じて分類すると次のようになるが，これらについて説明する。

- データ作成機器
- データ入力機器
- データ出力機器
- データ入出力機器
- データ検出機器
- データ変換機器
- データ加工機器

また，処理形態から分類すると次のようになる。

a）マニュアル・エントリ

b）メディア・エントリ

c）ディスプレイ装置

d）バッチ・データ・メディア

e）プロセス制御用装置

最後に図6－3のような例をあげて端末装置の全体を復習する。

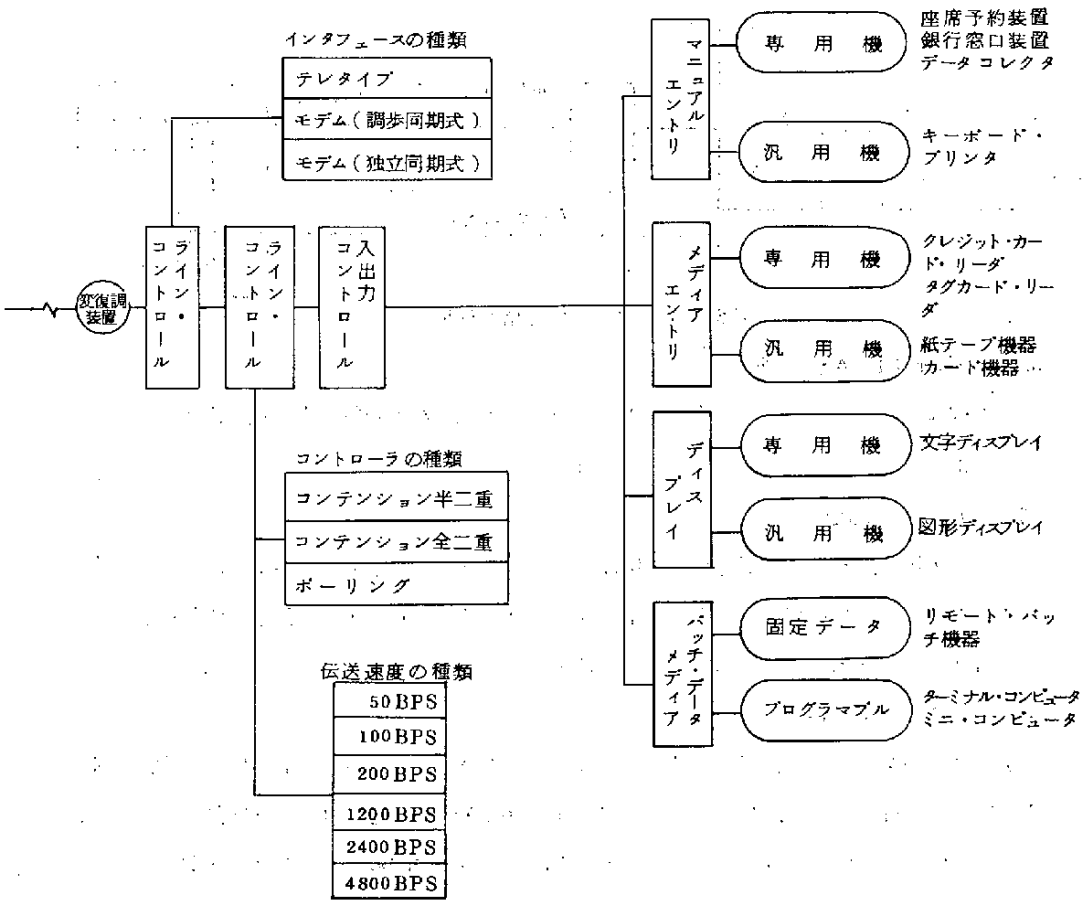


図6－3　端末装置の基本系統図

6.2　データ伝送路

コミュニケーション・システムにおけるデータ伝送路としては，利用者が通信回線を自ら設置してそれを使用する場合と，コモン・キャリア（電信電話会社。日本では日本電信電話公社）の通信回線を使用する場合とがある。我が国においては，鉄道会社，電力会社の一部を除いて，大部分が日本電信電話公社（以下電電公社）の通信回線を使用している。

この節では，データ伝送路の構成，構内回線，電電公社のデータ伝送路について説明する。

(1)　データ伝送路の構成

データ伝送路は通信回線と各種伝送用機器から構成され，図6－4のような位置づけになることを理解させる。

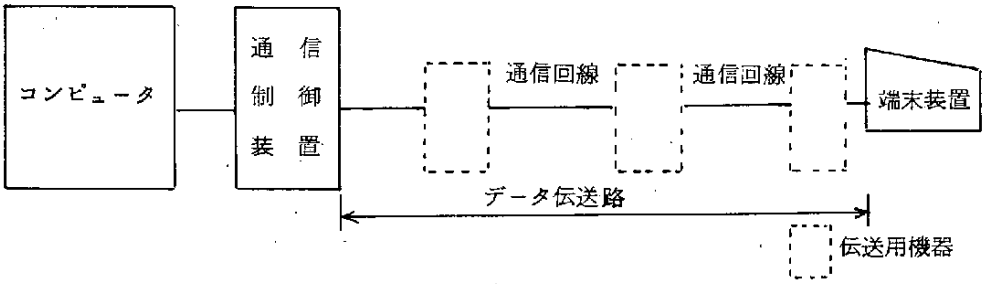


図６－４　データ伝送路の構成

(2)　２線式回線と４線式回線

通信回線には，２線式のものと４線式のものとがある。各々の原理と用途，両者の相違について理解させる。

(3)　直流伝送と交流伝送

データ伝送路上を情報が伝送される場合，コンピュータや端末装置の内部と同じように直流パルスで送る方法（直流伝送）と，直流パルスを交流に変換して送る方法（交流伝送）とがある。各々の原理と用途，相違について理解させる。

(4)　データ伝送路の種類

データ伝送路のうち，通信回線は，コンピュータや端末装置が設置されている建物内の部分と建物外の部分とがある。建物内の通信回線は一般に構内回線と呼ばれ，建物外の通信回線は一般に電電公社の回線が使用される。両者の関係については図６－５を参考に説明するとよい。

構内回線を構成している主端子盤，中間端子盤，および電電公社と利用者との保守の分界点となる切替端子盤についても説明する。また，データ通信に使用される電電公社の通信網としては，電話交換網と電信交換網とがある。各々の構成，特徴，データ通信用としての使用方法について理解させる。なお，データ通信回線サービスについては，次章で詳説する。

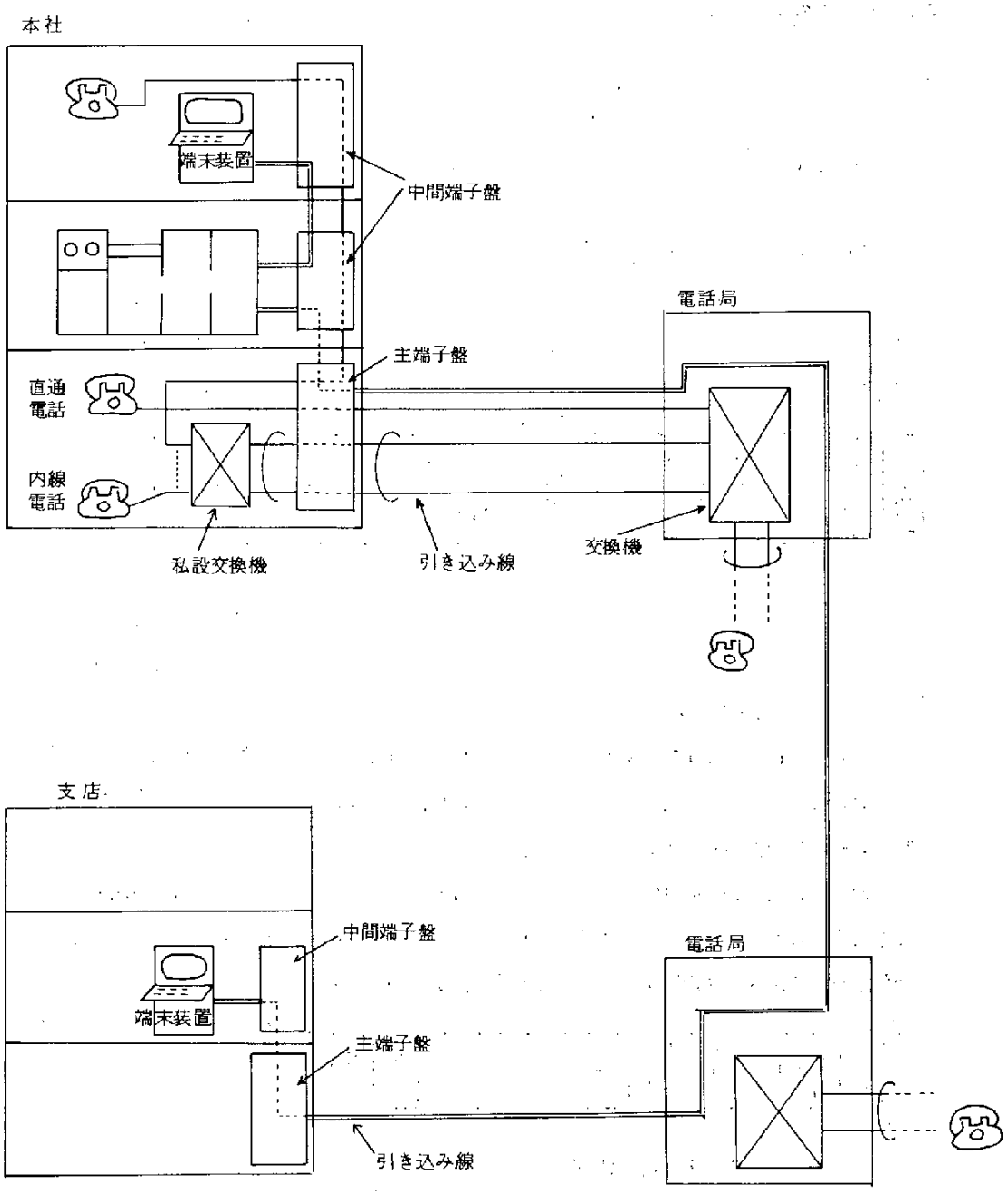


図6－5　データ伝送路の構成

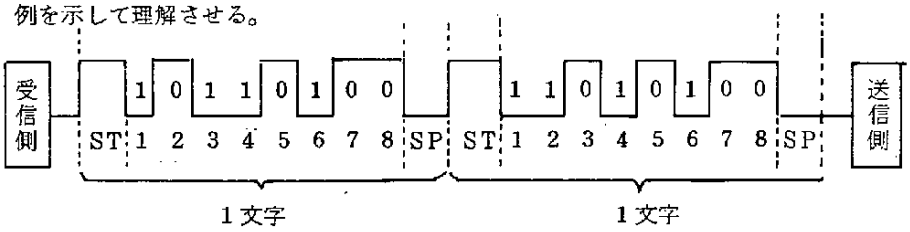
6.3 データの送受信

(1) 同 期 方 式

データの送受信を正確に行なうためには，送信側と受信側とで同期をとる必要がある。同期のとり方として，同期式（独立同期式）と非同期式（調歩同期式）とがあり，これらを説明する。

① 非同期式（調歩同期式；アシンクロナス）

非同期式では，通信回線上を伝送される1文字の前と後にスタート・ビットとストップ・ビットをつけ，送受信間で1文字ずつ同期をとりながら伝送することを，図6－6のような例を示して理解させる。



ST：スタート・ビット

SP：ストップ・ビット

図6－6 非同期式

ストップ・ビットは1.5ビット，2ビットの場合もあることを説明する。また，この方式が具体的にどのような原理で同期をとるのか，適用回線速度はどうかについても説明する。

② 同期式（独立同期式；シンクロナス）

同期式は1文字毎に同期符号を付加することはせず，データ・ビットを直列に順次連続して送信することを図6－7の例で理解させる。このときに，同期をとるために使用される同期信号キャラクタ（SYN）についてその使用方法を説明する。

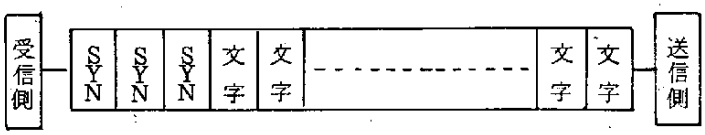


図6－7 同 期 式

同期式の場合，受信側の状態として同期モードとデータ・モードがあることとその違いについて説明する。また，この方式の適用回線速度，その理由についても理解させる。

(2) 変 調 方 式

直流信号を交流信号に変換する変調方式には，通信速度や回線の雑音に対する対応の仕方などから，振幅変調，周波数変調，位相変調の三つの変調方式があることを説明する。

① 振幅変調(Amplitude Modulation)

技術的には比較的簡単であるが，外界の影響に弱いため，一般には使用されていないが，簡単な原理の説明をする。

② 周波数変調(Frequency Modulation)

この方式は，電気信号(搬送波あるいは正弦波という)の周波数を高群，低群の二つに分け入力信号の１と０に対応させる方法であることを，図６－８のような例を用いて説明する。

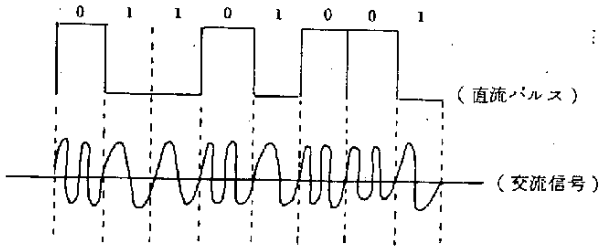


図６－８ 周波数変調

周波数変調の特徴，適用回線速度についても説明する。

③ 位相変調(Phase Modulation)

この方式は，入力データの１と０に対応して，搬送波の位相を変えて伝送する方式であることを説明する。次のような位相変調の特徴についても説明する。

- 周波数変調よりもさらに外界の影響に強い
- 周波数の帯域も少なくて良い

また，位相の切り換えには２位相，４位相，８位相があるが，各々の原理について図６－９のような例で理解させる。位相の切り換え，変調速度，回線速度との関係についても説明する。

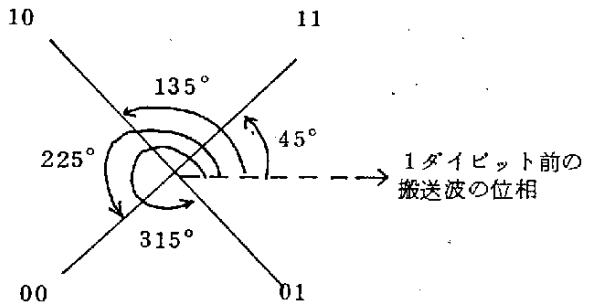


図６－９ ４位相変調の位相角度

(3) モデム・インタフェース

変復調装置は，通信制御装置や端末装置と接続され，種々のインタフェース・シグナルを制御しながら回線との間でデータの送受信を行なう。ここではインタフェース・シグナルの種類，その機能などについて理解させる。

① インタフェース・シグナルの種類

インタフェース・シグナルには，図６－１０のような種類があることを説明する。

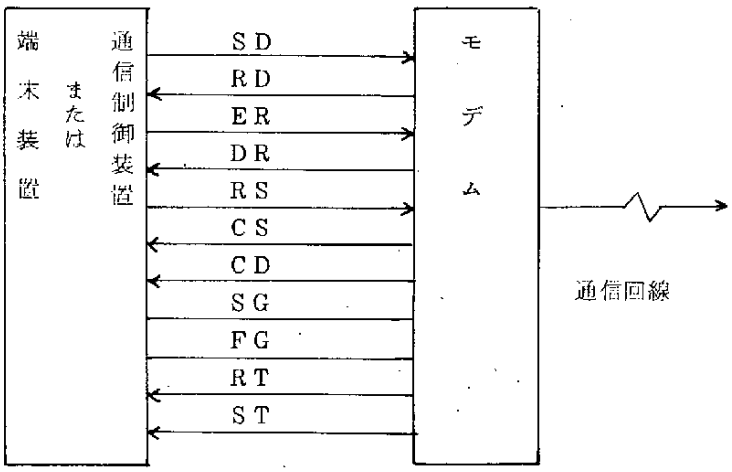


図６－１０ モデムと通信制御装置／端末装置のインタフェース

② インタフェース・シグナルの機能

以下のインタフェース・シグナルについてその機能を説明する。

- ＲＳ(Request To Send)送信要求
- ＣＳ(Clear To Send)送信可
- ＤＲ(Data Set Ready)モデム・レディ
- ＥＲ(Equipment Ready)端末装置レディ
- ＣＤ(Carrier Detect)キャリア検出
- ＦＧ(Frame Ground)保安用アース
- ＳＧ(Signal Ground)通信用アース
- ＳＤ(Send Data)データ送信
- ＲＤ(Receive Data)データ受信
- ＳＴ(Send Timing)送信タイミング
- ＲＴ(Receive Timing)受信タイミング

③ インタフェース・シグナルの動作

次の各段階におけるインタフェース・シグナルの動作を説明する。

- 初期動作 ― 通信制御装置，端末装置の電源をオンにした場合の動作
- データ通信の開始 ― データ通信を開始する場合の動作
- データの送受信 ― データを送受信する場合の動作
- データ通信の終了 ― データ通信を終了する場合の動作

6.4 通 信 方 式

一地点から他の地点への情報の流れは，一般に三つの基本的な方式に分類できる。ここでは，その各々について理解させる。

(1) 単向通信方式（simplex）

この方式は，送信側と受信側の2点間を図6－11 のような一対の通信回線で接続し，送信側と受信側を固定した形で行なうことを説明する。また，単向通信方式の適用分野についても理解させる。

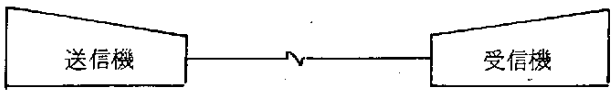


図6－11 単向通信方向

(2) 半二重通信方式（halt-duplex）

この方式は，図6－12 のように2方向の通信を交互に行なうデータ伝送方式であることや，その適用分野についても理解させる。

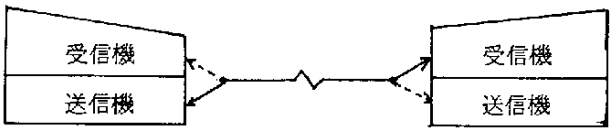


図6－12 半二重通信方式

(3) 全二重通信方式（full－duplex）

この方式は，図6－13 のように二方向の通信を同時に行なうデータ伝送方式であることを説明するとともに，この通信方式の適用分野についても理解させる。

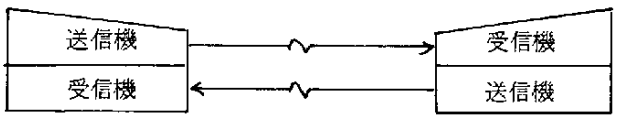


図6－13 全二重通信方式

指導上の留意点

(1) 本章では，コミュニケーション・システムを構成しているハードウエアを扱っているが，この分野は技術の進歩が著しいため，実際の指導では最新の資料を使用する必要がある。

(2) データ通信の基礎となる内容でもあるため，用語を理解させることも重要である。

参考文献

〔1〕山本　岩監　「データ通信」　産報，1971

〔2〕三原裕登ほか　「データ伝送入門」　技研，1966

〔3〕植田義明　「データ通信のための通信回線利用マニュアル」　企画センタ，1973

〔4〕「データ通信便覧」　データ通信協会，毎年刊行

〔5〕"Handbook of Data Communications", NCC Publications, 1975

〔6〕Karp ed., 'Basics of Data Communications", Mc Graw-Hill, 1976

〔7〕広田悳一郎　「データ伝送システム」　産業図書，1971

〔8〕斉藤正治　「ＴＳＳの計画と運営」　ＴＢＳ出版会，1977.

第7章　データ通信回線サービス

キーワード

特定通信回線（private line），公衆通信回線（switched line），回線交換（line switching），パケット交換（packet switching），他人使用（use by other persons），共同使用（multiple user lease），直営（public telecommunication equipment），自営（private telecommunication equipment），公衆電気通信法（public telecommunication law），有線電気通信法（wire telecommunication law），技術基準（technical criteria），型式審査（type Inspection），個別審査（Individual Inspection），開通検査（open channel Inspection），機器検査（device Inspection），パケット・プロトコル（packet protocol），パケット・プロトコル（packet procol），バーチャル・コール（virtual call），パーマネント・バーチャル・サーキット（parmanent virtual circuit），CCITT X.25

目　　標

データ通信には，必ず通信回線が存在する。この通信回線は，電力会社，鉄道等一部の利用者を除いてほとんどが，日本電信電話公社（以下電電公社と略す）および国際電信電話株式会社（以下国際電電と略す）から借りなければならない。電電公社および国際電電が提供する回線には多数の種類があり，また利用方法にも法律による種々の制約がある。データ通信システムを設計する場合，これらの回線の特徴，利用上の制約を十分理解し，回線の選択，ネットワークの設計をしなければならない。

本章では，電電公社，国際電電の回線の種類，法律による利用形態，データ通信関連法規，新データ網サービス，および，回線の効率的な利用法等，データ通信システム設計上必要な事項について理解させることを目標とする。

内　　容

7.1　電電公社のサービス

電電公社が提供する回線の種類，利用形態，申請方法，認定制度，等について説明する。

(1) 回線の種類

電電公社が提供するデータ通信回線には，特定通信回線，公衆通信回線がある。特定通信回線には表７－１の種類があり，公衆通信回線には表７－２のものがあることを理解させる。

表７－１　特定通信回線の種類

規　格	種　別	内　　容
A 規 格	A － 1	50 BPS 符号伝送（直流伝送）
B 規 格	B － 1	100 BPS 符号伝送（直流伝送）
C 規 格	C － 2	20,0, BPS 符号伝送（モデム直営）
D 規 格	D － 1	300 ～ 3400Hz 帯域使用（モデム自営）
	D － 5	1200 BPS 符号伝送（モデム直営）
	D － 7	2400 BPS 符号伝送（　〃　）
	D － 9	4800 BPS 符号伝送（　〃　）
I 規 格	I － 1	48 KHz 帯域使用（モデム自営）
	I － 3	48 KBPS 符号伝送（モデム直営）
J 規 格	J － 1	240 KHz 帯域使用（モデム自営）

表７－２　公衆通信回線の種類

種　別	内　　容
電 話 型	加入電話交換網を使うもの
電 信 型	加入電信交換網を使うもの

なお，特定通信回線と公衆通信回線のほかに，試行サービスとして東京，大阪で48 KB·PS交換回線サービスが提供されているので，これについても説明する。

(2) 利 用 形 態

電電公社回線の利用形態には，本人使用，他人使用，共同使用がありそれぞれ種々の利用上の制約がある。これらについて各々具体例をあげて説明する。

① 本 人 使 用

本人使用はセンタ側，端末側とも図７－１のように，同一のユーザが使用する形態であることを説明する。

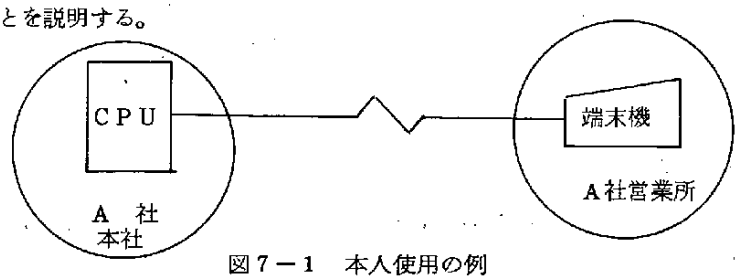


図７－１　本人使用の例

② 他 人 使 用

他人使用は，電電公社または国際電電の回線を使用して，第三者のデータ処理サービスをおこなう形態である。これを図7－2により説明する。

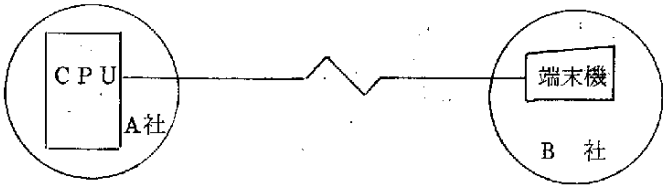


図7－2 他人使用の例

また，他人使用には利用上の形態に制限があるので，図7－3のような具体例をあげてCPUで処理された結果が入力された端末機にもどされるシステムであることを理解させる。

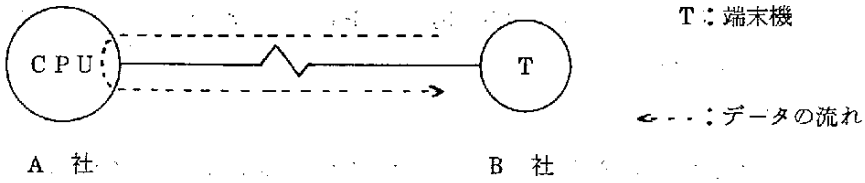


図7－3 端末機にもどされる例

また，国際特定通信回線では，この形態しか認められていないことも併せて説明する。

また，図7－3以外の形態で，代表的なものを，図7－4～図7－6を使用して説明する。さらに，制度的に認められない形態の例を，図7－6で説明する。

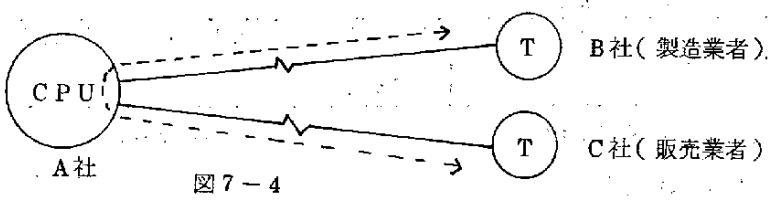


図7－4

・たとえばB社とC社の販売在庫管理のデータ処理を，A社が請け負うようなケース

B社とC社の間には業務上相当な関係が必要なこと，および，A社CPUがメッセージ交をしてはいけないことを説明する。

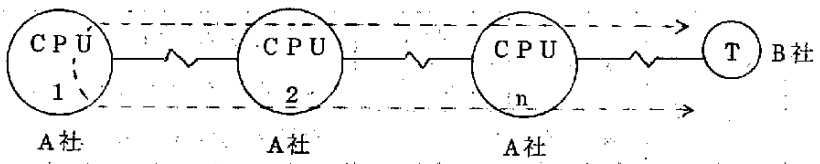


図7－5

• A社のＣＰＵが複数台あるケース

この場合，端末機からのデータは，どのＣＰＵで処理してもよいが，途中のＣＰＵでも，何らかのデータ処理が必要（メッセージ交換は認められない）なことを説明する。

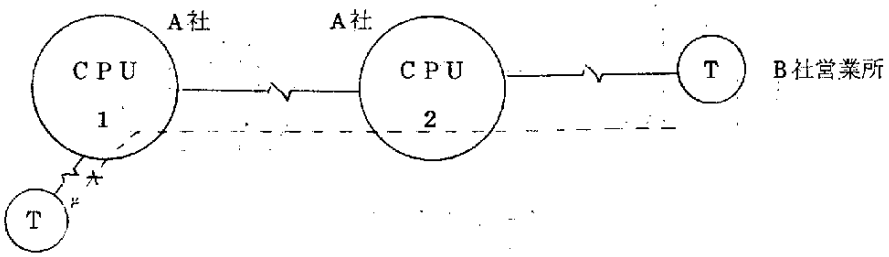


図７－６

• 図７－６のように，処理された結果を入力した端末機が収容されているＣＰＵ（例ではＣＰＵ２）と異なるＣＰＵ（例ではＣＰＵ１）に接続されている端末機に，出力することはできない。

③ 共 同 使 用

共同使用は，２以上の法人等が，共同して通信回線を使用するサービスである。共同使用には，基準認可と個別認可があることを，表７－３により説明する。

表７－３ 共同使用における基準認可と個別認可

基準認可	利用形態上のもの	ＣＰＵで処理した結果が，必ず入力した端末機に返される一つのＣＰＵと一つの端末機に終始する場合
	業務上の関係によるもの	郵政省令で定める業務上の関係があると認められる場合
個 別 認 可		上記基準認可に該当しないもので，郵政大臣の個別認可を受ける場合

また，共同使用においても，メッセージ交換は認められないことを説明する。

(3) 認定，適合審査

電電公社の回線に自営の端末機等を接続する場合，電電公社の技術基準に基づいた認定，審査を受けなければならないことを理解させる。また，この認定，審査の種類，内容について表７－４を使用して理解させる。

表７－４

適 合 審 査	機器審査	個別審査	電子計算機，端末機，モデム，ファクシミリ 等，網制御，通話機能を持たないもの
		型式審査	
	接続機能審査		
認 定	個 別 認 定		網制御装置，電話器 等の網制御および通話機能に関するもの
	型 式 認 定		

また，利用者が回線を申請して，それが電電公社から提供される時（回線開通），開通検査と機器検査がおこなわれることを理解させる。

(4) 関 連 法 規

データ通信に関連する法規には，

① 公衆電気通信法

② 有線電気通信法

があり，これにもとずいてサービスが提供されていることを説明する。

(5) 電電公社の新データ網サービス

電電公社は，昭和54年3月より新データ網サービスを開始する予定である。新データ網サービスには，回線交換サービスとパケット交換サービスがあることを理解させる。

① 回線交換サービス

回線交換サービスは，現在の公衆通信回線がディジタル化されたものと考えることができ，これには，200BPS以下，300BPS以下，1200BPS以下，2400BPS，4800BPS，9600BPS，48KBPSの各クラスがあることを説明する。

② パケット交換サービス

パケット交換サービスは，基本的に，データをパケットと呼ばれるブロックに区切って送受信する蓄積交換の一種であることを理解させる。また，パケット交換の方式として，バーチャル・コール，および，パーマネント・バーチャル・サーキットの2種類があること，および，パケット交換プロトコルなるものが存在しており，これにはCCITTX．25勧告があり，電電公社もこれに準拠しようとしていることを説明する。

③ 新データ網サービスの特長

新データ網のメリットとして，回線品質が良い，接続時間が短かい，合理的な料金体系である，特殊サービスが可能であることなどを理解させる。この特殊サービスについても，閉域接続，ダイレクト・コール，料金のセンター括払い，ID信号サービス，代行受信，同報通信等があることを説明する。

7.2 国際電電のサービス

国際電電が提供する回線の種類，利用形態等について理解させる。

(1) 回 線 の 種 類

国際電電が提供するデータ通信回線には，国際特定通信回線と，国際公衆通信回線があり，前者には表7－5の回線があり，後者には国際テレックスを使用した電信型があることを理解させる。

表７－５　国際特定通信回線の種類

クラス		速度
VOICE GRADE		使用する自営モデムによる。 国際電電も2400BPSモデムを提供する。
200 BAUD SPEED		200 BPS
100 BAUD SPEED		100 BPS
75 BAUD SPEED		75 BPS
50 BAUD SPEED	FULL SPEED	50 BPS
	HALF SPEED	25 BPS
	QUARTER SPEED	12.5 BPS

(2) 利用形態

国際電電の回線の場合も，電電公社の回線と同様に，本人使用，他人使用，共同使用の利用形態がある。これらは，他人使用の場合を除いて，電電公社の回線と同じであることを理解させる。

(3) その他

国際電電の回線の場合も，自営機器の認定審査があり，適用される法規も同じであることを理解させる。

7.3 通信回線の効率的利用

データ通信回線の効率的な利用例を図７－７のような例で説明し，理解させる。

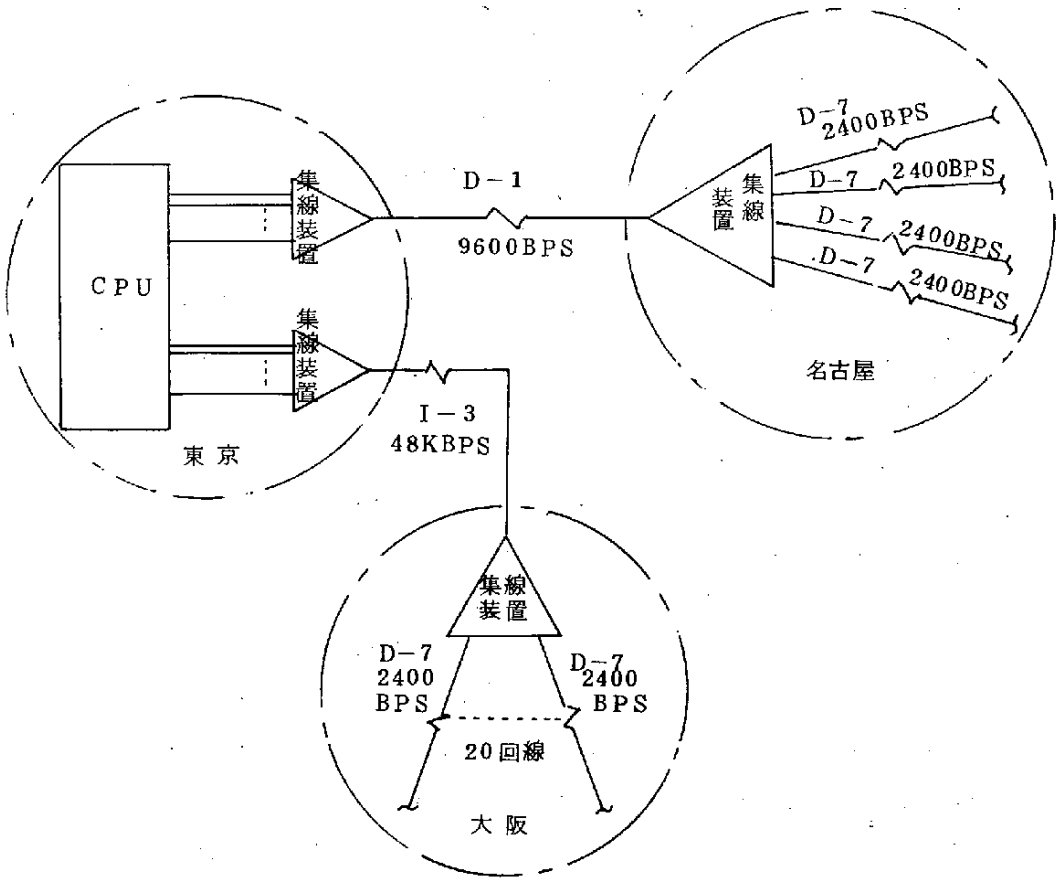


図７－７ 通信回線の効率的利用例

指導上の留意点

現実にコミュニケーション・システムを構築する場合，日本においては，日本電信電話公社と国際電信電話株式会社のサービス制度，技術基準などのほか，関連法規についても理解することが重要である。サービス制度，技術基準，関連法規は変更されることがあるので，講習に際しては最新のデータを入手しておく必要がある。

参考文献

〔1〕 植田義明「通信回線利用マニュアル」企画センター，1973

〔2〕「電気通信小六法」一二三書房，1978

第8章　コミュニケーション・システムと処理形態

キーワード

直通接続(point-to-point link), 分岐接続(multidrop link), コンピュータ・ネットワーク(computer network), 集中型ネットワーク(centralized network), 分散型ネットワーク(distributed network), 環状型ネットワーク(ring network), 網状型ネットワーク(network), リアルタイム・システム(real time system), リモート・バッチ・システム(remote batch system), 会話型システム(conversational processing system), 問合せ／応答型システム(Inquiry/answer system), メッセージ交換システム(message switching system), データ収集システム(data gathering system), データ分配システム(data dispatching system), タイムシェアリング・システム(timesharing system), 伝送制御(transmission control), 伝送制御手順(transmission control procedure), データ・リンク(data link), 主局(master station), 従局(slave station), 制御局(control station), 従属局(tributary station), 一次局(primary station), 二次局(secondary station), コンテンション(contention), ポーリング(polling), 伝送制御符号(transmission control character), ハイレベル・データリンク制御手順(high level data link control procedure), 不平衡型手順(unbalanced classes of procedure), 平衡型手順クラス(balanced classes of procedure), フレーム構成(frame structure), コマンド(command), レスポンス(response)

目　標

コミュニケーション・システムの構成は多様であり，その処理形態もまた多岐にわたる。近年，コンピュータ・ネットワークが出現するとともに，そのネットワーク形態も集中型，環状型，網状型などをとるようになってきている。

一方，コミュニケーション・システムの処理形態もリアルタイム，リモート・バッチ，会話型などの形態をとる。

本章では，通信回線の接続形態，ネットワークの形態，コミュニケーション・システムの処理

形態について理解させる。

また，データ通信では，相手方との情報の伝送を正確に行なうために，種々の制御を行なっている。たとえば，相手方との接続，相手方の確認，送受信可能か否かのチエック，伝送途中で発生する誤りの検出および訂正などである。このようなデータ通信のための制御を伝送制御とよび，これら一連の手続，規則を伝送制御手順と呼んでいる。本章では，伝送制御手順の用語を中心に，その全体像を理解させ，次に基本形データ伝送制御手順とハイレベル・データ・リンク制御手順とに分けて各々の違いを中心に理解させ，最後に誤り制御方式について説明する。

内　　容

8.1　通信回線の接続形態

この節では，通信回線の接続形態について，以下に示すような例をあげて説明し，併わせて，各々の接続形態の特徴，利点，問題点についても説明する。

(1)　直通接続（ポイント・ツー・ポイント接続）方式

コンピュータと端末装置とが図８－１のように一対一で接続され，データ量が多い場合に有効であることを理解させる。

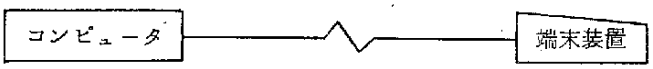


図８－１　直通接続方式

(2)　分岐接続（マルチ・ドロップ接続）方式

図８－２のように，一つの通信回線を複数の端末装置が共同で利用する方式であり，データ量が少ないときに，回線を経済的に有効に利用することができることを理解させる。

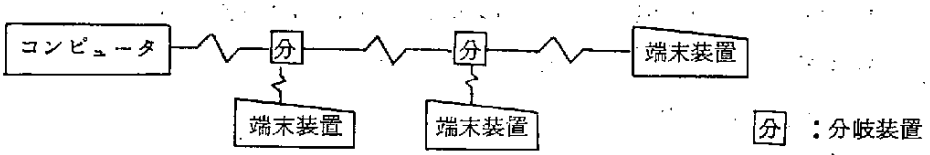


図８－２　分岐接続装置

(3)　交換接続方式

図８－３のように交換網を利用して，必要なときだけ，コンピュータと端末装置との間に接続を確立する方法であり，通信回線費用の低減，任意の相手との接続に有効であることを理解させる。

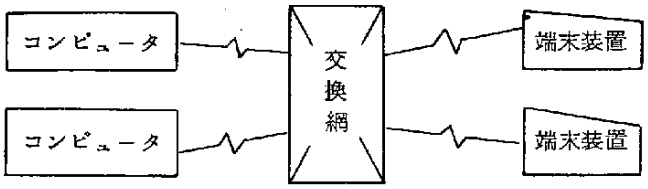


図８－３　交換接続方式

交換網としては電話交換網と電信交換網とがあるが，近く，ディジタル回線交換網，パケット交換網の利用も可能になる。

(4) 複　合　形　態

以上の接続方式のほかに，図８－４のようにコンピュータと端末装置との間に，集信装置を設置して，通信回線の集約を行なう場合もある。

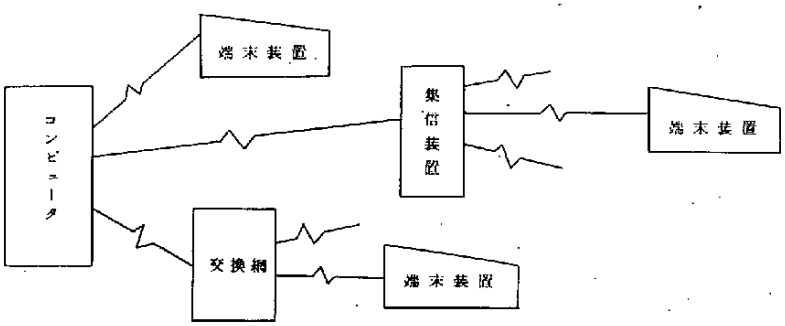


図８－４　　複　合　形　態

8.2 ネットワークの形態

従来のセンター集中型のシステムから，近年は技術の進歩と相まって，分散型システムが増加しつつある。それとともに，ネットワークも様々な形態をとるようになってきている。ここでは，様々なネットワーク形態について，コンピュータ・ネットワークを例にあげて説明するとともに，各々の特徴，問題点についても理解させる。

(1) 集　　中　　型

複数のコンピュータが，図８－５のように，中央のコンピュータを中心として，星状に接続されるネットワーク形態である。データの集約化が行ないやすく，回線を有効に利用できる反面，中央のコンピュータの障害が，システム全体に影響を与えるという点から信頼性に欠けることを説明する。

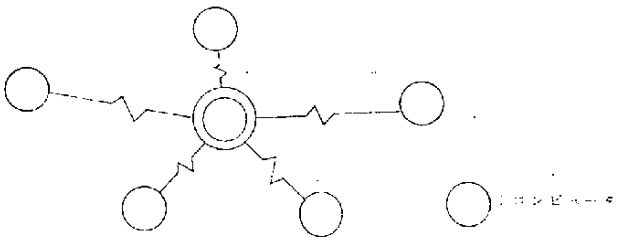


図8－5　集中型ネットワーク

(2) 環　状　型

コンピュータが，図8－6のように環状に接続されたネットワーク形態である。ネットワーク制御の機能は，コンピュータに分散している。回線の総延長距離は短かくて良いが，すべてのコンピュータで回線を共用するため，データ量の多いネットワークには向いていない。しかしながら，情報の流れが一定であることから経路制御が不要であり，コンピュータのネットワーク制御の負荷は少ないことを説明する。

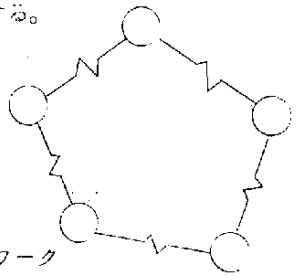


図8－6　環状型ネットワーク

(3) 網　状　型

複数のコンピュータが，図8－7のように網状に接続されているネットワーク形態である。ネットワーク制御機能は，各コンピュータに分散されている。この結果，あるコンピュータや回線などの障害が発しても網全体に影響を及ぼすことはなく，信頼性の高いネットワークとなる。

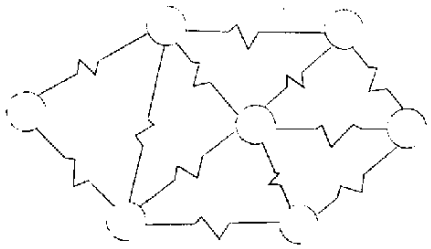


図8－7　網状型ネットワーク

8.3　コミュニケーション・システムの処理形態

コミュニケーション・システムの処理形態を分類すると，リアルタイム処理，リモート・バッチ処理，会話型処理がある。

この節では，これら三つの処理形態について説明する。

(1) リアルタイム処理

リアルタイム処理とは，銀行のオンライン・システムやみどりの窓口に見られる座席予約のシステムなど，即時性，広域性を必要とする分野で使用されている処理形態である。

リアルタイム処理は，データの流れによりさらに，次の四つの形態に分類して説明する。

① 問合せ／応答型処理

ある端末装置からデータを中央側のコンピュータに入力し，それを即時に処理し，再び同一端末装置に出力する形態である。

② メッセージ交換型処理

ある端末装置から入力された情報を交換局の役目を果すコンピュータを介して，他の端末装置へ出力する形態である。一般には，中央側のコンピュータで何らかの処理が行なわれることが多い。

③ データ収集型処理

端末装置からのデータを中央側のコンピュータに集める処理形態であり，収集されたデータは，一般には，オフラインで一括処理される。

④ データ分配型処理

データ収集型処理の反対で，中央側のコンピュータで処理した結果を，端末装置側へ送る形態である。

また，この節では，リアルタイム処理の持つ特性について，以下の項目を参考に説明する。

① 通信回線の使用

- 伝送制御の実行
- ビットと文字の変換

② データ発生がランダム

③ 待ち行列の発生

- 入力待ち行列
- チャネル待ち行列
- 出力待ち行列

④ 優先順位の設定

⑤ 応答時間

⑥ 信頼性の確保

なお，リアルタイム処理を行なうことによる一般的効果については，次のような項目をあげて説明する。

① 処理の即時性

② 省力化

③ 経営管理機能の強化

④ 企業競争力の強化

(2) リモート・バッチ処理

リモート・バッチ処理とは，遠隔地からバッチ処理を行う処理形態である。中央側のコンピュータ室で行うバッチ処理と同質のサービスを，遠隔地のユーザに提供するものである。オペレーティング・システム上ではリモート・バッチ処理と中央側のバッチ処理との区別はない。ただし，リモート・バッチ処理は，通信回線を介して行なうため，伝送制御などの機能が追加されている。図８－８のような例を示しながら，この処理形態の特徴を理解させる。

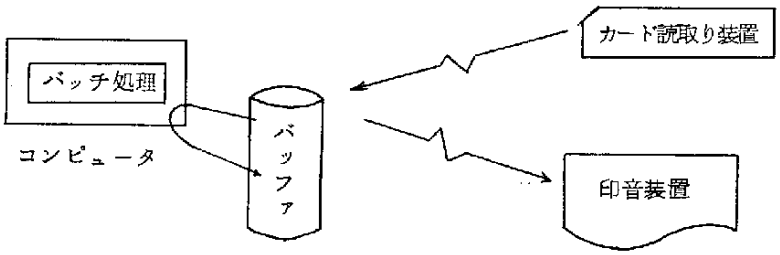


図８－８ リモート・バッチ処理の概要

(3) 会話型処理

会話型処理とは，人間が端末装置を使用して中央のコンピュータと直接，対話しながら作業を進めていく形態であることを，図８－９のように例示しながら説明する。

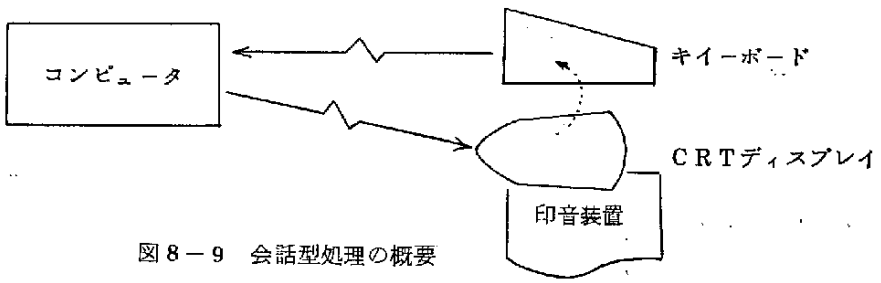


図８－９ 会話型処理の概要

リアルタイム処理と会話型処理とのちがいは，端末装置からプログラムを入力することができるか否かにある。つまり，会話型処理では，端末装置からプログラムを入力することが可能であるが，リアルタイム処理では，端末装置から入力できるのはデータのみであることを理解させる。

(4) タイムシェアリング・システム（ＴＳＳ）

会話型処理機能をもつシステムを，タイムシェアリング・システムと言うが，この場合に要求される次の諸機能について説明する。

① 待ち行列の制御

② 自動スケジューリング機能

③ 応答時間の保証

④ 利用形態による優先度

⑤ 機密保護機能

⑥ 共用ファイルのロック機能

⑦ アカウンティング機能

⑧ ロールイン／ロールアウト機能

また，タイムシェアリング・システムの効果について，次のような項目をあげて説明するとよい。

① 開発体制の変化

データの入力から処理結果の入手までの作業の大部分が現場で行なわれるため，コンピュータ室の負荷が大幅に軽減できる。

② コンピュータ利用人口の増加

③ 適用業務の高度化および拡大

8.4 伝 送 制 御

この節では，伝送制御の概要を次の諸点から理解させる。

(1) データ・リンク

送信側と受信側の間で情報の交換を可能にするため，ある一定の手順で確立された伝送経路のことをデータ・リンクと呼ぶ。

データ・リンクの端末側，コンピュータ側を"局"と呼び，次の種類があるが，これら局の機能や接続形態について理解させる。

① 主局と従局 ｝基本形データ伝送制御手順
② 制御局と従属局 ｝

③ 一次局と二次局 ｝ハイレベル・データ・リンク制御手順
④ 複合局 ｝

(2) 伝送制御手順の段階

回線を介して互いに通信を行うときの一定の手順を，伝送制御手順というが，これは，通常次の五つの段階（フエイズ）に分けられる。

第1段階（フェイズ1） 回線接続

第2段階（フェイズ2） データ・リンクの確立

第3段階（フェイズ3） 情報転送

第4段階（フェイズ4）　終　結

第5段階（フェイズ5）　回線切断

各々の段階について具体例をあげて説明する。また，通信回線の種類と使用される段階についても説明する。

(3) 伝送制御で定義される項目

伝送制御方式を定めるとともに決定すべき項目として，次のものがあるが，これらについて例をあげて説明する。

① 回線接続方式

- ポイント・ツー・ポイント方式（直通方式）
- マルチ・ドロップ接続方式（分岐方式）

② 通信方式

- 単向通信方式
- 半二重通信方式
- 全二重通信方式

③ 通信速度

④ 同期方式

- 調歩同期方式（アシンクロナス方式）
- 独立同期方式（シンクロナス方式）

⑤ 情報符号

- ＪＩＳコード
- ＡＳＣＩＩコード
- ＥＢＣＤＩＣコード
- その他

⑥ 誤り制御方式

- 誤り検出方式
  - 水平／垂直パリティ・チェック（ＬＲＣ／ＶＲＣ）
  - サイクリック・リダンダンシィ・チェック（ＣＲＣ）
  - その他
- 誤り訂正方式
  - 再送方式
  - その他

⑦ 電文長

⑧ 伝送制御文字

⑨ メッセージ・フォーマット

⑩ 伝送制御手順

8.5 基本形データ伝送制御手順

基本形データ伝送制御手順の方式，伝送制御符号，メッセージ方式について説明する。

(1) コンテンション方式

コンテンション方式は，データ・リンクのうち，ポイント・ツー・ポイント・リンクに用いられる伝送制御方式で，送信データが発生した時点で，随時，コンピュータまたは端末装置から「送信要求」を出すことができる。半二重通信回線の場合には送信権を得ようとして，コンピュータと端末装置との競合（コンテンション）が発生する。図８－１０を例に，コンテンション方式の伝送制御手順を説明する。

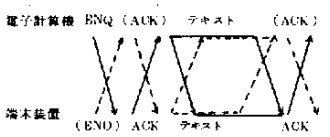
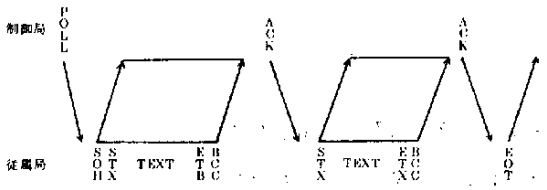


図８－１０　コンテンション方式（全二重）の例

(2) ポーリング方式

ポーリング方式は，データ・リンクのうちマルチ・ドロップ・リンクに主として用いられる伝送制御方式でコンピュータ（制御局）から各端末装置（従属局）へポーリングをかけて，データの送信を勧誘する。コンピュータから端末装置にデータを送出するときは，端末装置の受信状態を問い合せる（セレクティング）。図８－１１を例にポーリング方式について説明する。

（ポーリング）



（セレクティング）

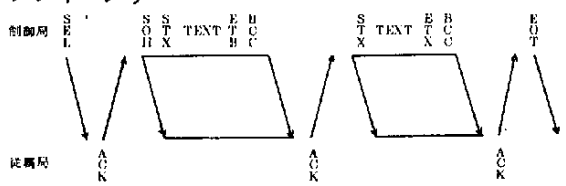


図８－１１　ポーリング方式の例

(3) 伝送制御符号

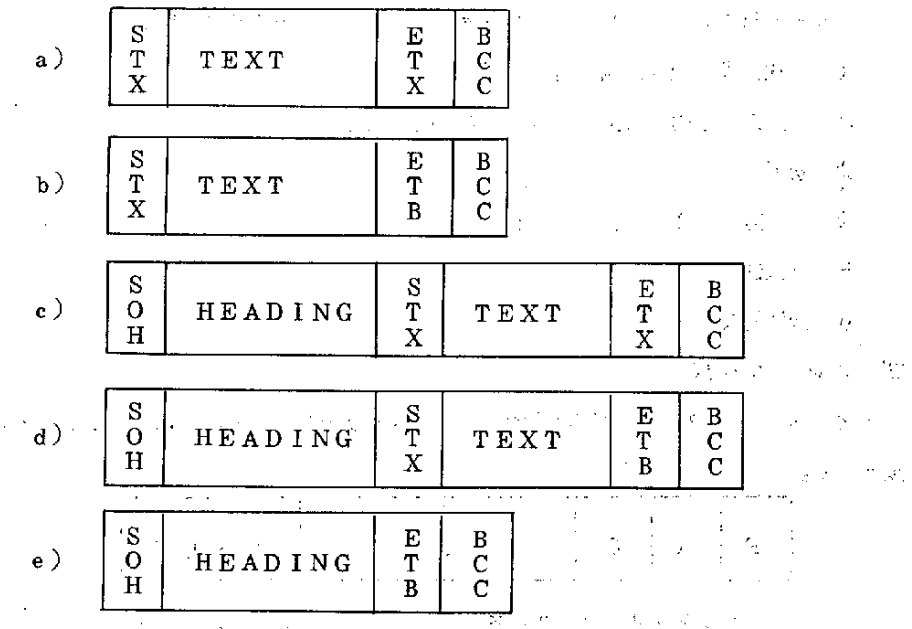
伝送制御を行うためには，制御用のコード（符号）が必要になる。ＩＳＯでは表８－１のような１０種類のコードを定義している。各々の符号の意味，使用法について説明する。

表８－１　伝送制御符号

符　号	名　　称
SOH	Start of Heading （ヘディング開始）
STX	Start of Text （テキスト開始）
ETX	End of Text （テキスト終結）
EOT	End of Transmission （伝送終結）
ENG	Enquiry （問合せ）
ACK	Acknowledgement （肯定符号）
NAK	Negative Acknowledgment （否定符号）
ETB	End of Transmission Block（伝送ブロック終結）
SYN	Synchronous Idle （同期符号）
DLE	Data Link Escape （伝送符号拡張）

(4) メッセージ形式

メッセージの形式には，図８－１２のような五つの形式があり，これについて説明する。



＊BCC：Block Check Character

図８－１２ メッセージの形式

8.6 ハイレベル・データ・リンク制御手順

ハイレベル・データ・リンク制御手順は，ＩＳＯを中心に標準化された新しい伝送制御手順である。これは，基本形データ伝送制御手順の問題点を解決するとともに，コンピュータとコンピュータの間，またはコンピュータとバッファ付端末装置の間の効率の良いデータ伝送を可能にするためのものである。この節では，ハイレベル・データ・リンク制御手順の概要，フレーム構成，手順要素，手順クラスのシーケンス例について説明する。

(1) ハイレベル・データ・リンク制御手順の概要

ここでは，ハイレベル・データ・リンク制御手順の概要を理解させるために，以下の項目について説明する。

① 1次局／2次局と複合局

データ・リンクは，1次局／2次局また複合局と通信回線から構成される。ここでは，1次局，2次局，複合局の機能について理解させる。

② 手順クラス

手順クラスは，データ・リンクの制御および情報の転送に関する局間の主導権の持ち方によって，平衡形手順クラスと不平衡形手順クラスが定義される。各々について説明する。

③ 基本形データ伝送制御手順とハイレベル・データ・リンク制御手順の比較

以下の各項目について，基本形データ伝送制御手順とハイレベル・データ・リンク制御手順との比較を行ない，ハイレベル・データ・リンク制御手順の特徴を理解させる。

① 監視方式（交互監視と同時監視）

② 伝送方式（両方向交互伝送と両方向同時伝送）

③ 透過性

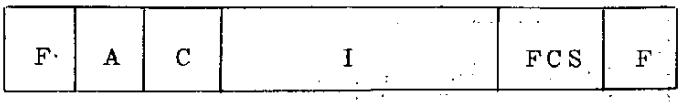
④ 応答方式（遂次応答と代表応答）

⑤ 拡張性

⑥ 接続構成

(2) フレーム構成

ハイレベル・データ・リンク制御手順では，すべての伝送は，図8－13のようなフレーム構成をとる。



F：フラグ・シーケンス　　I：情報フィールド

A：アドレス・フィールド　　FCS：フレーム・チエック・シーケンス

C：制御フィールド

図8－13　フレーム構成

ここでは，次の各々のフィールドについて説明する。

① フラグ・シーケンス

- 用途
- 透過性の実現方法

② アドレス・フィールド

- 用途
- アドレス・フィールドの拡張

③ 制御フィールド

- フォーマットの種類とビット構成
- シーケンス番号，P／Fビット
- 制御フィールドの拡張

④ フレーム・チェック・シーケンス（FCS）

- FCSの生成，誤り検出の方法

(3) 手順要素

モードとコマンド／レスポンスについて次の事項を説明する。

① モード

- 動作モード
- 切断モード
- 初期モード

② コマンドとレスポンス

コマンドとレスポンスは，制御フィールドの内容により

- 情報転送フォーマット
- 監視フォーマット
- 非番号制フォーマット

の三つに分類定義される。各フォーマットごとに，そこで定義されているコマンド／レスポンスの種類と使用方法について説明する。

(4) 手順クラスのシーケンス例

以下の各々のシーケンスについて，例をあげて説明する。

- NRM （両方向非同時伝送）の例
- NRM （両方向同時伝送）の例
- ARM （両方向同時伝送）の例
- ABM （両方向同時伝送）の例

8.7 誤り制御方式

データ通信では，伝送路の途中で誤りが発生することが考えられるが，伝送路で生じた誤りを検出して，訂正することを誤り制御という。この節では，誤り検出方式と誤り訂正方式について説明する。

(1) 誤り検出方式

誤り検出方式としては，以下のものがある。

① 2度送り方式

② エコー・チエック方式

③ パリティ・チエック方式（水平，垂直）

④ 群計数チエック方式

⑤ サイクリック・リダンダンシィ・チエック方式（CRC方式）

(2) 誤り訂正方式

誤り訂正方式としては，次のものがある。

① 再送方式

② 自己訂正方式

通常，再送方式が利用されるが，各々について例をあげて説明する。

指導上の留意点

コミュニケーション・システムの構成も，ディジタル交換網，パケット交換網などのサービスの開始に伴い，大きな変化を遂げるものと予想される。また，通信制御処理装置の発達，インテリジェント・ターミナルの発達も，コミュニケーション・システムの構成に，影響を与えることが予想される。

一方，処理形態では，コンピュータ・ネットワークを使った新しい処理形態の発達に期待が寄せられる。また，伝送制御手順については，ハイレベル・データ・リンク制御手順の実用化も進んでいる。

本章の指導では，コミュニケーション・システムの最新の技術動向や，伝送制御手順に見られる標準化動向を踏まえながら指導を進めたい。

参考文献

〔1〕喜田村善一監「コンピュータ・ネットワーク」 1975

〔2〕N.Abramson, F. K. Kuo, "Computer Communication Networks", Prentica-Hall, 1972

（猪瀬博訳「コンピュータ・ネットワーク」　オーム，1975）

〔3〕 Barber, "Communications Networks", Online, 1975

〔4〕 Green, Lucky, "Computer Communications", IEEE, 1975

〔5〕 猪方研二編　「オンライン・システム入門」，企画センター，1968

〔6〕 J．Martin, "Design of Real-time Computer Systems"; Prentice-Hall, 1976

（北原安定訳　「リアルタイム」　日本経営出版会）

〔7〕 J．Martin, "Telecommunications and the Computer", Prentice-Hall, 1976

〔8〕 斎藤正治　「ＴＳＳの計画と運用」　ＴＢＳ出版会，1977

第9章　コミュニケーション・システムの設計と評価

キーワード

平均到着率(average arrival rate)，平均使用率(average utilization rati)，隘路(bottle neck)，スループット(throughput)，応答時間(response time)，待ち行列(queue)，待ち行列理論(queing thery)，指数分布(exponential distribution)，アーラン分布(erlang distribution)，ポアソン分布(poisson distribution)，シミュレーション(simulation)

目　　標

コミュニケーション・システムを設計する場合，端末装置から始まってコンピュータの主記憶やファイル・レイアウトの見積りに至るまで多くの要素がある。本章では，はじめに端末／回線関係に重点を置き，端末装置の選定手順，回線網の設計手順について理解させる。

次に，コミュニケーション・システムにおいて，待ちの発生しやすい隘路について説明したあと，コミュニケーション・システムの評価手法について説明する。コミュニケーション・システムの評価手法として，“待ち行列理論”によるものと，“シミュレーション”によるものとを取り上げ，その適用方法などについて理解させる。

内　　容

9.1　コミュニケーション・システムの設計

この節では，コミュニケーション・システムの中で，コンピュータ側を除く回線網，端末装置に重点をおいて，コミュニケーション・システムの設計方法について理解させる。

(1)　設計の手順

コミュニケーション・システムの設計手順は，大よそ図9－1のようになる。

通信回線や端末装置には，多くの種類がありそれらの組み合せも多くのパターンが考えられる。したがって，コミュニケーション・システムの設計は，図9－1の手順を何度か繰り返すことが必要になることを理解させる。

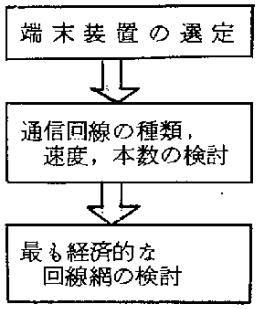


図9－1　設計手順

(2) 端末装置の選定

端末装置は人間とシステムとの接点に位置するため，重要な役割を担う。端末装置を選定する手順について，図９－２を参考に説明する。

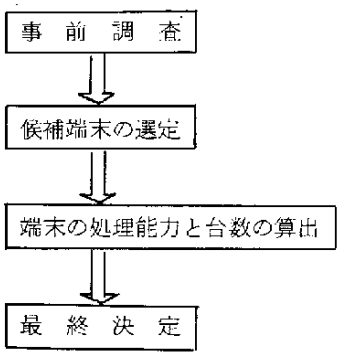


図９－２　端末装置の選定手順

① 事前調査

ここでは，事前調査の対象となる項目について説明する。調査項目としては，たとえば以下のようなものがある。

- 対象業務
- 端末の設置場所
- 端末設置場所の環境条件
- 専任オペレータの有無
- 発生データの種類
- データ量
- インプット媒体
- 予　算

② 候補端末の選定

事前調査の結果をもとに，候補となり得る端末装置を何種類か選定する。

③ 端末の処理能力の算出

端末装置の処理能力は，次式で算出できる。

$$x\text{件／時} = \frac{3600\ \text{秒}}{t\ \text{秒}}$$

x：処理件数（件／時）

t：１件の処理時間（秒）

一件の処理時間は，図９－３の合計として考えられる。各々の時間の算出方法についても，例をあげて説明するとよい。

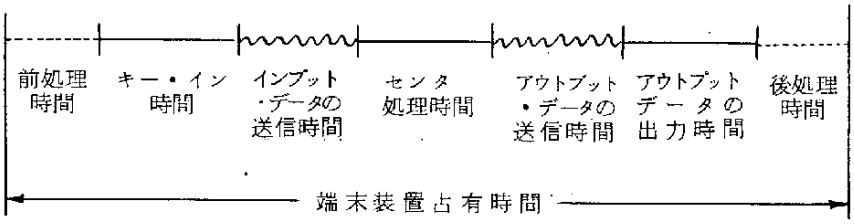


図９－３　端末装置占有時間

④　端末台数の算出

端末装置の必要台数は，次式で算出できる。

$$N\text{台} = \frac{K\ \text{件／時}}{x\ \text{件／時}}$$

N：端末装置台数（台）

K：１時間当りのデータ件数（件／時）

x：１時間当りの処理件数（端末装置１台当り）（件／時）

⑤　最終決定

以上の結果をもとに，次のような条件を考慮して最終的に端末装置を決定する。各条件についてそのポイントを説明する。

- 機　能
- 操作性
- 経済性
- 保守性

(3)　回線網の設計

ここでは，回線網を設計するときの調査項目と，設計手順について理解させる。

①　回線網設計のための調査項目

回線網を設計するときに，調査すべき項目として以下のものが考えられるが，各々について説明する。

- 端末装置
- データ量
- 端末装置，中継装置，コンピュータの設置場所
- 障害対策

②　回線網の設計手順

回線網の設計は，通常，以下の手順で行うが，各段階について例をあげて説明する。

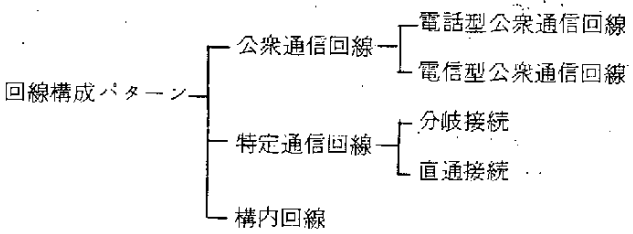
・各端末設置場所における回線使用時間と回線使用率を算出する。

・ネットワークの形態を検討する。（集中型か分散型か）

・各回線の構成パターンを検討する。

回線構成パターンとしては，表9－1のように考えられる。

表9－1　回線構成パターン



③　回線使用時間と回線使用率の算出

回線使用時間は，通常，次式で算出されるが，例をあげて説明する。

$$\text{回線使用時間}_{(秒)} = \frac{(\text{インプット文字数}+\text{アウトプット文字数})\times(\text{データ件数})\times(\text{1字当りのビット数})}{(\text{回線速度})\times(\text{伝送効率})}$$

回線使用率は，次式で算出できるが，例をあげて説明する。

$$\text{回線使用率}(\%) = \frac{\text{回線使用時間（秒）}}{3600\times\text{業務時間（時）}} \times 100$$

9.2　コミュニケーション・システムの評価

この節では，コミュニケーション・システムの評価に関し，隘路について説明したあと，評価手法として待ち行列理論による方法と，シミュレーションによる方法とについて理解させる。

(1)　コミュニケーション・システムの隘路

コミュニケーション・システムの設計／評価を行っていく場合，システムの処理能力がその評価対象となる。システムの処理能力は，一定時間当りの処理件数（スループット）と，応答時間（レスポンス・タイム）で表現されることが多い。たとえば，応答時間について考えてみると，コミュニケーション・システムで応答時間に影響を与える隘路（ボトルネック）は，図9－4のような場所に存在する。つまり，コミュニケーション・システムを評価する場合，システム内の隘路での待ちの状態を分析することが重要となる。以上のことを説明したのち，図9－4のような例をあげて，コミュニケーション・システムの隘路について理解させる。

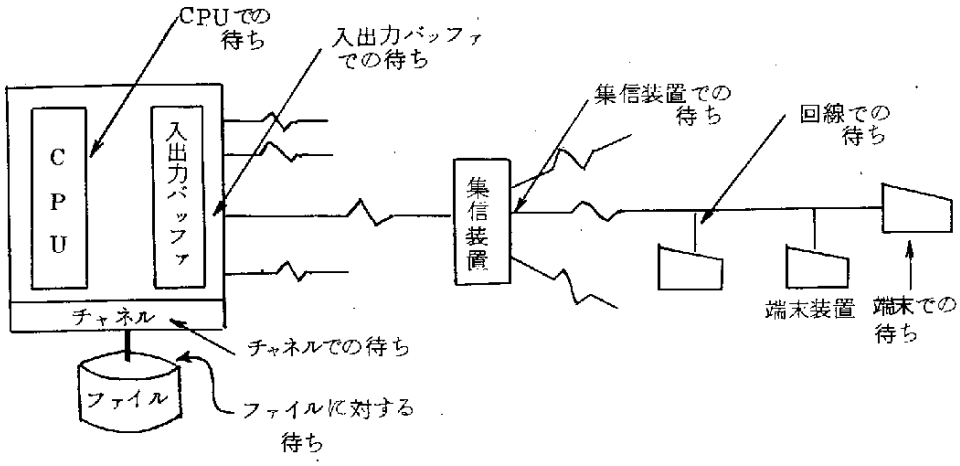


図9－4　コミュニケーション・システムの隘路

(2)　待ち行列理論による評価

システム設計の初期の段階では，待ち行列理論等を使用した数式モデルによる評価が有効であることを理解させる。

具体的な適用方法の説明の前に，到着の確率法則，指数分布とアーラン分布，窓口1個の待ち行列，複数窓口の待ち行列，平均値の法則，優先権などについて理解させる。

また，次の説明を行なう。

①　端末装置における待ち行列

端末装置における待ち行列の状態を推定することは，特にサービス業におけるコミュニケーション・システムを構築する場合，重要なポイントとなる。ここでは，費用とサービス性とのバランスのうえで，端末装置を検討する際，待ち行列理論をどのように適用したらよいかを説明する。

②　通信回線での待ち行列

通信回線での待ち行列を考える場合，伝送制御手順がコンテンション方式かポーリング方式かで，適用の仕方が異なる。

コンテンション方式の場合は，比較的，待ち行列理論の適用が容易である。具体例をあげて説明するとよい。

ポーリング方式の場合は，どちらかといえば，待ち行列理論よりもシミュレーションを適用した方がよい。しかし，回線使用率が低く，ポーリング制御に用する時間が無視できる場合は，単一サービス窓口として取扱うこともできる。ここでは，ポーリング方式の回線に対する待ち行列理論の適用について説明する。

③　システム全体に対する適用事例の紹介

コミュニケーション・システムを構成する各装置における待ち時間と処理時間とを算出することによって，応答時間を求め，そのシステムの評価を行うことができる。この時，待ち時間の算出のために，待ち行列理論を適用する。次のような順序で具体例をあげて説明するとよい。

- システムの構成
- 業務処理の流れ
- 処理のタイム・チャート
- 待ち行列モデル
- 待ち時間の算出
- 処理能力の算出

(3)　シミュレーションによる評価

コミュニケーション・システムの評価にあたって，シミュレーションは有効な手段である。シミュレーション・モデルは，システム全体またはシステムの一部について作成されるが，実際の動作に近い動きをし，各種の統計情報を出力してくれる。一度，作成されたモデルは，パラメータを変更するだけで様々の状態を再現し，その時々の統計情報を知ることができる。ここでは，シミュレーション・プログラムの概要を紹介したあと，具体的な例をあげて説明する。できれば，実際にプログラミングさせ，コンピュータで実行させてみるとよい。

①　シミュレーション・プログラムの種類

シミュレーションを実行するためのプログラム（シミュレータ）について，その種類，長所，欠点などを説明する。

②　汎用シミュレーション言語

汎用のシミュレーション言語としては，ＧＰＳＳ，ＳＩＭＳＣＲＩＰＴ，ＳＩＭＵＬＡなどがあるが，一般にはＧＰＳＳが最も多く使用されている。ここでは，各々のシミュレーション言語の概要について説明したあと，ＧＰＳＳの使用方法を簡潔に説明する。

③　ファイル・アクセスの例

ファイル・アクセスの問題を例にして，ＧＰＳＳでモデル化し，実行させる。

④　アウトプットの検討

実行した結果をもとに，アウトプットの見方，評価材料としての利用方法について，説明する。

⑤　待ち行列理論の結果との比較

同じ例に待ち行列理論を適用し，シミュレーションの結果と比較検討させる。

⑥ シミュレーションの問題点

最後にシミュレーションの問題点，危険性について説明し，正しい使用方法を理解させる。

指導上の留意点

(1) できる限り例題をもとにした実習を行なわせた方がよい。

(2) 現実に稼動中のシステムの評価には，モニタリングと呼ばれる手法が使用されているので，時間があるときはこれについても触れた方がよい。

参 考 文 献

〔1〕 大野 豊 「オンライン・リアルタイム・システムの設計」 産業図書，1970

〔2〕 E. Yourdon著／大野 豊訳 「オンライン・システムの設計」 丸善，1975

〔3〕 J. Martin, "Design of Real-time Computer Systems", Prentice-Hall, 1976（北原安定訳 「リアルタイム」 日本経営出版会）

〔4〕 大野 豊 「オンライン・システムのソフトウェア」 産業図書，1977

〔5〕 J. Martin, "Systems Analysis for Data Transmission" Prentice-Hall, 1972

第10章　総合システム

キーワード

データベース・システム（data base system），データの独立性（data indipendency），ホスト言語方式（host language system），専用言語方式（self contained language system），デンタベース管理者（data base administrator），D.B／D.C（data base／data communication），分散型データベース（distributed data base）

目　　標

総合システムの例として，データベース・システムをとりあげる。

総合システムの特徴としては，データの共有性があげられるが，データベース・システムは，そのようなデータの共同利用に関し適切な機能を利用者に提供するものといえる。このため，現在では，総合システムの構築において，データベース・システムの適用が不可欠なものとなっている。

ここでは，そのようなデータベース・システムの機能をまず説明する。次にデータベース・システムの分類と動向を考察し，総合システムの将来像を示すことを，第1の目標とする。

また，ファイルが個別の業務から独立して共用されるようになると，オンライン・システムの中央集中化がより可能になってくる。一方，情報発生の局地性やデータ管理の安全性の問題などから集中化されたデータベースに対して分散されたデータベースの必要性も出てきている。

本章の第2の目標は，総合システムの例として，データベース機能とデータ・コミュニケーション機能を総合的にとらえるDB／DCの概念について説明し，合わせてデータベースの分散化の問題をとりあげ分散型データベースについて理解させることにある。

内　　容

10.1 データベース・システムの機能

データベース・システムは，ファイル（データ）の取り扱いを中心として抽象化されたシステムである。このようなところから概念的には理解しやすくても，実際に適用する段になると，とまどうことが多い。

データベース・システムの機能を十分にいかすには，総合システムの設計の時に，十分，システム像としてデータベース・システムをとらえ，開発するシステムと組み合わせる必要がある。

このために，現存する具体的なデータ管理のパッケージを例として，まずデータベース・システムの全体像を説明する。CODASYL案や，ＩＭＳなどを参考とするとよいであろう。

データベース・システムの機能としては，次のようなことを理解させ，システムの利用者との関連性を重視して説明する。

(1) システムの利用者とそのための機能

データベース・システムの利用者を次の３種類に分け，そのためにどのような機能が提供されているかを説明する。

① エンド・ユーザ

② 専門のプログラマ

③ データ管理者

機能ポイントとなるのは次の二つである。

① 言語の形式

② 操作モード

(2) データの独立性

次の点から，データの独立性について理解させる。

① プログラムが，データの変更から受ける影響の度合い

② プログラムとデータの結合の行なわれる時点

③ ファイル装置からの独立性

(3) データの定義

表現できるデータ構造により，データベース・システムは特徴づけられる。ここでは，データ構造の種類および特定の応用に適応したデータ構造について説明する。

① 線型構造

② 階層木構造

③ ネットワーク構造

応用例としては，次のようなものを考えるとよい。

① 部品展開での部品関係

② 人事ファイル

③ 在庫管理ファイル

(4) 実行できる機能

どのレベルのシステム利用者に対し，どのような機能が提供されるかを説明する。

① データベースの定義，生成機能

② 問合せ言語

③ 報告書作成機能

④ 更新機能

⑤ プログラマの為の機能

10.2 データベース・システムの分類と動向

データベース・システムは，システムの総合化ということでいろいろな立場から検討，開発されてきているわけではない。

ここでは，データベース・システムの分類を行ない，各々の特徴から総合システムの目指すところを示す。各々のデータベース・システムは，必ずしも明確に分類できるわけでないが，次のような分類の仕方が参考になるであろう。

(1) ホスト言語方式と専用語語方式

これは使用する言語の面からの分類である。親言語方式（host language system）は既存のプログラム言語の拡張として開発され，プログラマを対象にしている。

専用言語方式（self contained language system）は，プログラマでない利用者のための機能を準備して，専用プログラマの必要性を少なくしようという方向で開発されてきた。

ここでは，両方式の代表的なシステムを具体的に取りあげ，システムが用意している機能やユーザとのインタフェースの取り方などを説明する。

両方式の区別は必ずしも明確ではなく，むしろ現在は，一つのシステムで両者のユーザ・インタフェースを用意しようという点を注意する必要がある。このことは，データベースの利用者の多様化ととらえるべきである。

(2) 目的別システム

データベース・システムに，目的を与えたものである。このタイプのデータベース・システムは，前述の分類では専用言語システムに属する。

ここでは，代表的な専用システムを具体的なシステムの例として説明する。

① データ検索を目的とするもの

② 報告書作成を目的とするもの

③ 情報検索を目的とするもの

④ 生産管理を目的とするもの

(3) その他

その他，データベース・システムは，オペレーション・モードやユーザとのインタフェースの取り方により，次のような特性をもっていることを理解させる。

① オン・ライン使用向けシステム

② 画面端末使用向けシステム

さらに，データベース・システムは，総合情報システム化の下で，種々のレベルの使用者に，種々の使い方をサービスするものとして期待されている。このようなニーズを理解させる。

10.3 データベースの共用化

データベースの共用についてはのことがあげられる。

① 複数使用者による共用

② 異なった応用による共用

これらの要求を満たすために，データベースはファイルと異なり，次のような特性を持ってい

① 複雑なデータ構造の表現

② データ独立性

また，データの論理構造と物理構造の区別によりこれらの要求を満たそうとしており，そのために次のような概念が重要になっている。

① データの論理構造

② データの物理構造

③ スキーマ

④ サブ・スキーマ

これらの話題を通して，データベースの共用化の意義，留意点，問題点についてまとめる。

データベースは，同時に多数の利用者から使えるようにデータを集中管理しているので，その正当性や整合性を維持するためには，業務別ファイル・システムの場合と比べ，より完全な管理が必要となる。このために，データベース・システムではデータ管理者（ＤＢＡ）と呼ばれるデータベースを管理するための技術をもった個人ないしは集団が欠かせない。

ここでは，データベース管理者の必要性を理解させ，データベース管理者の果すべき役割について説明する。

データベース管理者の役割として，次のことを説明する。

① データベースの定義

② データベースの改訂

③ データベースのアクセス法の決定

④ 機密保護

⑤ エラー回復の設計

⑥ 運用体制の設計

⑦ システム・モニタリング

⑧ ユーザ間のコンフリクトの解決

⑨ ユーザとデータベース間のコンフリクトの解決

⑩ データベースの再編成

10.4 データベース／データ・コミュニケーション

コンピュータの利用分野には，次のような傾向がみられる。

① 総合的な情報システム化の推進

② データベースに基づいたシステム開発

③ データベースのオンライン使用

④ データベースの分散

⑤ 対話形式の業務と端末装置の利用増加

⑥ バッチ，オンラインの共存

⑦ エンド・ユーザへのサービス向上

⑧ エンド・ユーザ向けの言語の増加

⑨ エンド・ユーザによるシステム開発

このように巾広い，ユーザを含む全社的な規模でのシステム化という面から，データベース／データ・コミュニケーション（DB／DC）の重要性は高いものになってきている。

ここでは総合情報システムと言う観点からデータベース／データ・コミュニケーションについて説明する。

(1) DB／DCの概略

DB／DCの利点について説明し，めざすところを明らかにする。

① 応用プログラムとデータの独立性

② データとファイル装置の独立性

③ データの管理の一元化

④ 遠隔地処理の効果

⑤ システムの開発，運営の容易性

⑥ コンピュータ投資面での効果

⑦ 生産性の向上

(2) DB／DCの定義

DB／DCを，次の諸点から説明する。

① データベース

- データの重複性の最小化
- 複雑なデータ構造
- 関係の表現
- 応用の多様性
- データの共用性
- データの同期性

② データ・コミュニケーション

- 即時性
- 同時処理
- 遠隔地利用
- 端末装置の共用
- データベースへのアクセス

(3) DB／DCの基本機能

DB／DCの持つべき基本機能を説明する。

① データベース機能

- データからの独立性
- ファイル装置からの独立性
- 共用データベースへの多数アクセス
- 柔軟なデータ構造

② 端末管理機能

- データからの独立性
- 端末装置の共用
- 柔軟なメッセージ制御機能

  メッセージ交換

  メッセージ編集

  会話処理

  機密保護

③ 言語インタフェース

- 標準的なインタフェース
- データベースの操作機能
- 問合せ言語機能

④ システム制御機能

- バッチ，オンラインの同時使用
- プログラムのスケジューリング
- 資源の割りあて
- チェックポイント／リスタート
- 統計情報の収集

10.5 分散型データベース

情報発生の局地性や安全性，コストの観点からみた場合，集中化されたデータベースに対して分散型データベースの導入も必要となる。ここでは，分散型データベースについて，その背景や構成，意義について説明する。

(1) 分散化の概念

① ファイルの分散

② プロセッサの分散

③ データ管理システムの分散

(2) 分散型データベースへの期待

① 多様化するユーザの要求への対処

② システム全体の処理効率

③ システムの信頼性の向上

④ システムの開発コストの低減

⑤ 通信コストの軽減

(3) 分散化の必要性

① 情報量の増大

② 情報の関係の複雑化

③ 情報発生源の広域化

④ 処理の即時性

⑤ 機密保護

⑥ 安全性

⑦ ユーザ要求の多様化

(4) 分散化の背景

① データベース・システムの発展

② コミュニケーション技術の発展

③ ファイル媒体の高速化，大容量化

④ ミニ・コンピュータの発展

⑤ システムの安全性，信頼性の要求

(5) 分散型データベースの構成

① 階層分散型

(i) 利点

- システム管理ソフトウェアが単純
- システム管理が容易
- 構築が容易

(ii) 欠点

- サブ・システム間の直接の通信ができない
- オーバヘッド
- 上位システムの障害の影響が大

② ネットワーク分散型

(i) 利点

- オーバヘッドが少ない
- 信頼性が高い
- 独立にシステム設計ができる

(ii) 欠点

- 通信の相手が多い
- システム構築がむずかしい

③ 機能構成

- コミュニケーション機能

  回線端末制御

  メッセージ伝送制御

  メッセージ編集

- データ管理機能

  データベースへのアクセス

  データ独立性

  データの整合性

  機密保護

- インタフェースの標準化機能

  データ操作言語の標準

データの抽象化

サブ・システム向けのデータ編成

- システム管理機能

メタ・データの管理

会話処理

変換機能

- データ管理者

データベースの管理

ネットワーク内でのファイルの運営

トランザクション特性の分析評価

資源利用の特性分析評価

復旧

(6) 分散型データベースの例

分散型データベースはまだ商用化されたものはない。ここではXDMSなどを説明するとよい。

(7) 分散型データベース・システムの展望

分散化の動向に関し将来を展望する。

① データベース・システムの将来

② コミュニケーションの将来

③ ミニ・コンピュータの将来

④ 記憶装置の将来

⑤ コスト・パフォーマンスの向上

⑥ 分散化へのアプローチ

指導上の留意点

データベース機能とデータ・コミュニケーション機能を統合した，総合システムについての理解を深めさせながら，これからのシステム構築がデータベースとコミュニケーションを中心にすえて行なわれることを理解できるよう留意する。

参 考 文 献

〔1〕 CODASYL Systems Committee, "Selection and Acquisition of Data Base Management Systems", 1976
（西村恕彦編「データベースの選び方」bit 臨時増刊, 共立出版, 1977）

〔2〕 Buckley, "Communications and Data Management", John Wiley, 1976

〔3〕 Lefkovitz, "Data Management for On-line Systems", Hayden Book, 1974

〔4〕 Gibbons, "Integrity and Recovery in Computer Systems", Hayden Book, 1976

〔5〕 Lyon, "The Data Base Administrator", John-Wiley, 1976

〔6〕 Clark, Redell, "Protection of Information in Computer Systems", IEEE, 1975

単元C4

ソフトウエア設計

単元C4
ソフトウエア設計

目次

第1章　プログラミング言語

キーワード

データ構造（data structure），名前の有効範囲（scope of name），静的記憶域（static storage），自動記憶域（automatic storage），被制御記憶域（controlled storage），基底付き記憶域（based storage），ブロック構造（block structure），省略時解釈（default interpretation），手続き（procedure），割込み処理（interrupt processing），リスト処理（list processing），翻訳時機能（compile−time−compilation），マクロ機能（macro function），前処理ルーチン（pre−processor）

目　　標

最近とみにシステム・プログラム記述言語として，ＰＬ／Ｉのサブセットからなる言語が多く開発され，実際にオペレーティング・システムやコンパイラ等の開発に使用されている。

本章では，まず，簡単なプログラムがＰＬ／Ｉで組めるようになること，プログラムを構造面から把握させることを目標とする。次に，ＰＬ／Ｉの概念を基盤にして，より高度な特徴機能を理解させること，特に，翻訳時機能を使用することにより，外部媒体から原始プログラムを複写，編集する効果（マクロ化，生産性，標準化）について十分把握させることを目標とする。

内　　容

1.1　高水準言語の利点

大規模プログラムの開発において高水準言語はとみに重要になってきている。これらの言語は，これまでアセンブリ言語でなければ効果的に遂行できなかった処理までをも行なうことができる。システム的な観点から，高水準言語の使用は全システムのプログラム製造工程（設計，コーディング，デバッグ，文書化等）や保守の段階の改善を図っている。たとえば，ＭＵＬＴＩＣＳシステムの開発におけるＰＬ／Ｉの利用がある。本節では，以下のような高水準言語の利点について説明する。

(1)　プログラマの生産性を高める。

高水準言語はしばしばアセンブリ言語より数倍も表現能力が大きい。

(2)　修得期間が短い。

(3) 文書化能力が優れている。

ＰＬ／Ｉプログラムはそれ自身かなり文書化されたものである。したがって，プログラムが読みやすく，保守が容易で，かつ変更しやすい。

(4) プログラムの信頼性が高い。

優れた文書化能力により，プログラマをこまごました問題から開放する。その結果，全般的な問題に注意を向けることを可能にし，信頼性が向上する。

(5) ハードウエアから独立している。

ＰＬ／Ｉのような高水準言語はアセンブリ言語に比較してハードウエアの依存度がきわめて小さい。

1.2 ＰＬ／Ｉの特徴

本節ではＰＬ／Ｉ言語の重要な特徴について簡単に説明し，用語や機能の概念を理解させる。

(1) 記述形式

- 文の形式
- 注釈行

について説明する。特に，ＰＬ／Ｉの文の記述は自由形式であること，注釈行はプログラムの空白が書ける任意の場所に書くことができ，これがプログラムの読み易さ（ｒｅａｄａｂｉｌｉｔｙ）に大きく貢献していることを実例をあげて説明する。

(2) 拡大されたデータの型とデータ構造

ＦＯＲＴＲＡＮで記述できない以下の項目を中心に説明する。

- 文字列
- ビット列
- データ構造

ＰＬ／Ｉの変数はいくつかまとめて配列や構造体に編成することができること，配列は同一の性質をもった要素からなること，構造体は変数と配列の集合であり，属性は必ずしも同じでなくてもよいこと，構造体が別の構造体を含んでもよいことといった概念を把握させる。

(3) 代入文と制御文

配列代入文，構造体代入文および名札変数付きＧＯ ＴＯ文等ＰＬ／Ｉの特徴的なものについて概念を理解させる。

(4) ブロック構造（ｂｌｏｃｋ ｓｔｒｕｃｔｕｒｅ）

ブロックには手続きブロックとＢＥＧＩＮブロックがあること，およびその相違を説明する。こうしたブロック構造により，プログラムの明瞭さ（独立性），デバッグ，保守の容易性を実

例をあげて理解させる。

ブロック構造をうまく利用するためには，名前の有効範囲（ scope of name ），ブロックの起動，終了，あるいはこれらのブロックとの情報の授受の仕方について，よく把握させることが必要である。

(5) 記憶域の割当て

記憶域の割当ては，通常，多くの言語が採用している静的（ statically ）に行うものと，プログラムの実行中に動的（ dynamically ）に行うものとがある。

動的記憶域はプログラマが割当てや解放を制御するものであり，これを活用して，オペレーティング・システムやコンパイラ等における記憶域の効率的な使用が可能となる。

ＰＬ／Ｉプログラムでは記憶域の割当てに，ＳＴＡＴＩＣ，ＡＵＴＯＭＡＴＩＣ，ＣＯＮＴＲＯＬＬＥＤ，ＢＡＳＥＤの４種類の異なった方法があるが，概念を上記との関連で説明する。

(6) リスト処理（ list processing ）

情報検索やオペレーティング・システム，コンパイラ等でよく使う表や制御表等はリスト構造をなしているものが多い。これらの領域を，動的な記憶域割当て方式を活用して連鎖したり削除するような操作をＰＬ／Ｉのリスト処理機能で行えることを理解させる。

(7) 翻訳時機能（ compile-time-compilation）

ＰＬ／Ｉはマクロ機能を持っている。これにより，しばしば使用する手順や宣言をあらかじめ，外部記憶装置にライブラリとして登録しておき，これを％ＩＮＣＬＵＤＥ文で原始プログラムの一部分に組み入れることができる。

また，プログラム中に現れた名前を変えたり，原始プログラムのある部分だけを編成翻訳するような機能を標準化，プログラムの生産性と関連づけて把握させる。

1.3 データ要素

ＰＬ／Ｉで取扱うデータについて，下記の項目に関して説明する。

- データ属性
- データの型，種類
- 省略時解釈
- データの構造

(1) データの属性

ＰＬ／Ｉの特徴の一つにデータ属性の豊富さがある。各種のデータ属性の意義と記述法について説明する。データの属性を大別すると次の四つの範疇に分類できる。

・型 指 定　名前の値の範囲と性質を指定する（ＦＩＸＥＤ，ＦＬＯＡＴ，ＤＥＣＩＭＡＬ，ＢＩＮＡＲＹ）。

・構造指定　配列等の次元，列データの長さを指定する（Ａ（3，3）ＣＨＡＲＡＣＴＥＲ（10））。

・範囲指定　名前が意味を持つ範囲を指定する（ＩＮＴＥＲＮＡＬ，ＥＸＴＥＲＮＡＬ）。

・記憶域割当て指定　割当てられた記憶域の期間を指定する（ＳＴＡＴＩＣ，ＡＵＴＯ－－ＭＡＴＩＣ，ＣＯＮＴＲＯＬＬＥＤ）。

(2) データの型と種類

ＰＬ／Ｌのデータは処理用データとプログラム制御用データの２種に大別できる。処理用データはまた算術データと列データの二つに分けられる。

(a) 算術データ

算術データは表現形式（ｆｏｒｍ），基底（ｂａｓｅ），表示（ｓｃａｌｅ），モード（ｍｏｄｅ），精度（ｐｒｅｃｉｓｉｏｎ）の５個の性質があって，これらの組合せにより，多種類のデータが作られることを各種の実例で説明する。

・表現形式：　数値フィールドとコード化形式。

・基　　底：　10進か２進。

・表　　示：　固定小数点か浮動小数点。

・モ　ー　ド：　実数か複素数。

・精　　度：　使用するコンピュータによって制限される。

(b) 列データ

列データには文字列データとビット列データがあり，その活用法を説明する。

(c) プログラム制御用データ

プログラム制御用データとして，名札データ，ポインタ・データ，領域データ，入口データ等がある。これらについて説明する。

(3) 省略時の解釈（ｄｅｆａｕｌｔ　ｉｎｔｅｒｐｒｅｔａｔｉｏｎ）

ＰＬ／Ｉには非常に多くの属性があるが，これら全てを毎回明記するのは面倒であり，適宜に省略時の解釈を活用することがプログラミングの生産性，読み易さ（ｒｅａｄａｂｉｌｉｔｙ）を高めることになる。また省略時のデータの精度については，利用するコンパイラごとにまちまちなので，コンピュータとの関連で説明する。

(4) データの構造（ｄａｔａ　ｓｔｒｕｃｔｕｒｅ）

データの集合である配列と構造体について，記述法と機能および効果的な使用法について説明する。

(a) 配　列

同一の属性をもったデータ項目の集りであるベクトルや行列に対して配列が使われる。配列の変換規則はＦＯＲＴＲＡＮと異なるので注意を要する。

例　ＤＥＣＬＡＲＥ　Ａ(３，３)は次の順に変換される。

Ａ(１，１)，Ａ(１，２)，Ａ(１，３)，Ａ(２，１)，Ａ(２，２)，Ａ(２，３)，・・・，Ａ(３，３)

また，ＰＬ／Ｉでは負の添字の上／下限が指定でき，アルゴリズムによっては０とか負の添字を使うと便利なことが多い。

配列の断面（ｃｒｏｓｓ　ｓｅｃｔｉｏｎ）―配列のある行，列あるいは面の上にあるすべての元一の表記法とその効果的使用法について説明する。

例　Ａ(３，＊)，　Ｂ(５，＊，＊)

(b)　構造体

構造体はスカラ項目，配列，構造体などからなる階層をもったデータの集りである。構造体の記述法，機能，効果的な使い方を説明する。

構造体の各要素は同じ型のデータである必要はない。構造体は一連のレベル付名前の宣言により定義される。構造体と配列を結びつけることは複雑なデータを取り扱うときによく行われる。構造体は木構造を形成したり，リスト構造を扱うデータ等に適している。たとえば，オペレーティング・システムやコンパイラあるいは高度な応用プログラム等での異なるタイプの関連したデータを格納した各種の制御表は構造体で宣言するとよい。

例１）　階層をもった構造体

```
DCL  1  ACCT-RECORD,
          2  NUMBER  CHAR(10),
          2  NAME,
             3  FIRST  CHAR(10),
             3  MID-INIT  CHAR(1),
             3  LAST  CHAR(10),
          2  ADDRESS,
             3  STREET,
                4  NUMBER  FIXED(3),
                4  NAME  CHAR(15),
             3  CITY,
                4  NAME  CHAR(15),
                4  ZIP-CODE  FIXED(5),
             3  STATE  CHAR(5),
          2  BAL  FIXED(7,2);
```

この構造体をわかりやすく図解すると次のようになる（同一名（この例ではＮＡＭＥ）に対する参照の仕方にも注意する）。

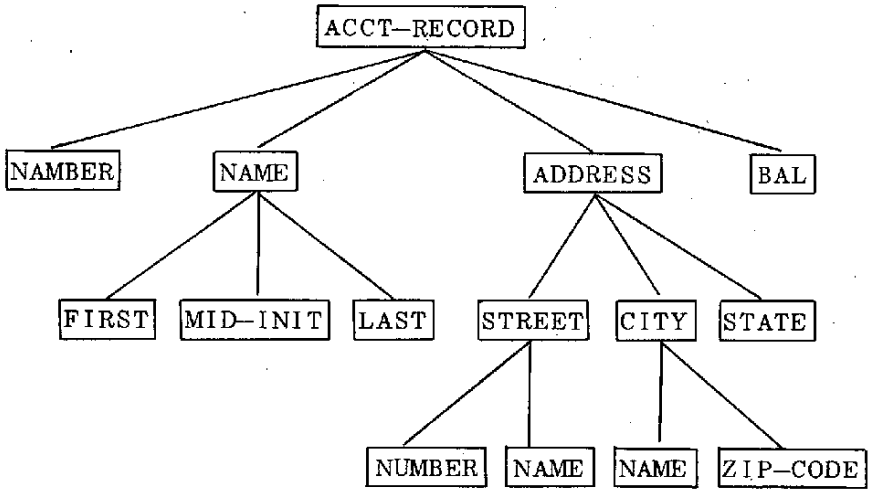


図１－１　階層をもった構造体

例２）　配列をもった構造体

```
DECLARE  1  TEAM(3),
           2  NAME,
           2  MEMBER,
              3  CAPTAIN,
              3  PLAYER(2);
```

表１－１　配列をもった構造体

←第１レベル	←第２レベル	←第３レベル
TEAM(1)	NAME(1)	
	MEMBER(1)	CAPTAIN(1)
		PLAYER(1,1)
		PLAYER(1,2)
TEAM(2)	NAME(2)	
	MEMBER(2)	CAPTAIN(2)
		PLAYER(2,1)
		PLAYER(2,2)
TEAM(3)	NAME(3)	
	MEMBER(3)	CAPTAIN(3)
		PLAYER(3,1)
		PLAYER(3,2)

1.4 式および代入文と制御文

ＰＬ／Ｉの式，代入文，制御文の概念および記述法と機能などを理解させる。

(1) 式と代入文

ＰＬ／Ｉにはスカラ式，配列式，構造体式の３種がある。これらの記述法と機能を理解させる。

(a) 算術データを含む式と代入文

式中の算術データの変換規則（表１－２）について説明する。

二項演算において一つの被演算子が定数であるならば，コンパイル時にもう一つの被演算子の属性に対応したものに変換される。もし両方とも定数または変数であり，異った属性を持っているならば，その中の一つは次のように変換される。

表１－２　算術式のデータの変換規則

種　　類	変　換　内　容
型　（ form ）	符号化された形（ coded form ）に変換される。
基底（ base ）	基底が異るときは，１０進の被演算子が２進に変換される。
表示（ scale ）	２個の被演算子の表示が異るときは，固定小数点の被演算子が浮動小数点に変換される。
モード（ mode ）	モードが異るときには，実数の被演算子が虚数部零の複素数に変換される。
精度（ precision）	精度が異っても変換は行われない。

厳密に，どのようにデータが変換されるかということは，個々のコンピュータのデータ項目の内部表現がどうなるかによって決るので，特定のコンピュータについて具体例で説明する。

関数の値は特に注意を要する。すなわち，ＰＬ／Ｉの組込み関数はＦＯＲＴＲＡＮのように一つのデータ型に限定されているわけではなく，結果としての関数の値は関数と引数の型によって定まる。ＦＯＲＴＲＡＮには，たとえば，ＳＱＲＴ，ＤＳＱＲＴ，ＣＳＱＲＴ，ＤＣＳＱＲＴ等があるが，ＰＬ／Ｉには平方根の関数としてＳＱＲＴ１個しかない。

(b) 列データを含む式と代入文

連結演算子（ concatenation operator（ || ））は列データに対してのみ作用する。

- 連結演算子の操作について説明する。
- 被演算子の型，列の長さの等しくない場合の扱いについて説明する。

さらに，ビット列の操作については以下のことを説明する。

・ビット列間の論理積

・ビット列間の論理和

・否　定

(c) 列データに対する組込み関数

列データに作用する以下の組込み関数の機能を理解させる。

・ＢＩＴ関数

・ＣＨＡＲ関数

・ＳＵＢＳＴＲ関数

・ＩＮＤＥＸ関数

・ＬＥＮＧＴＨ関数

・ＲＥＰＥＡＴ関数

・ＢＯＯＬ関数

(d) 擬変数としてのＳＵＢＳＴＲ

関数ＳＵＢＳＴＲは擬変数（ pseudovariable ）として使用することもできる。擬変数の概念を理解させる。

擬変数とは関数として使用されたり，値を受け取る変数として使用されたりする組込み関数名のことである。

たとえば，

Ｘ＝'ＡＢＣＤ'，　Ｙ＝'ＥＦＧＨ'

ＳＵＢＳＴＲ（Ｘ，２，２）＝ＳＵＢＳＴＲ（Ｙ，３，２）

によって列データＸは'ＡＧＨＤ'という値を持つことになる。

(e) 配列式と配列代入文

ＰＬ／Ｉの配列式と配列代入文の記述法と機能を理解させる。

通常の計算において配列の全部あるいは一部，たとえば２次元の配列の一つの行などに演算を施したい場合も多い。ＦＯＲＴＲＡＮでは，これはループ演算によってＤＯ文やＩＦ文を用いて実行される。ＰＬ／Ｉでは，そのための演算機能が備わっている。配列演算は一番右側の添字が優先して変化し，この順序で行なわれる。

例

Ａ＝－Ｂ＋３＊Ｃ；

Ａ，Ｂ，Ｃがそれぞれ二次元の配列で添字の最大値が（３，４）であるとすれば，この文は

```
DO I=1 TO 3;
DO J=1 TO 4;
A(I,J)=-B(I,J)+3*C(I,J);
END;
END;
```

と同じである。

(f) 構造体式と構造体代入文

構造体式と構造体代入文の記述法と機能について理解させる。特にby name の用い方について説明する。by name が用いられてない場合には，代入文に現れる構造体レベル番号は同一でなくてもよいが，構造体は同一でなければならない。構造体中の項目，または構造体自身が配列ならば，それと同じ次元のものでなければならない。

(2) 制御文

PL／Iの制御文として，GO TO文，DO文，IF文の記述法や機能についてFORTRANと対比して説明する。

(a) GO TO文

GO TO文の名札や名札変数の機能について説明する。名札変数の使い方はFORTRANの計算型GO TO文と似ていることに注意する。

(b) DO文

DO文の3種類の形態について，その記述法と処理機能を把握させる。また文のグループ化としての重要性を理解させる。

- 単純グループ　　例 DO; ; ; END;
- ループ　　例 DO I=1 TO 10; ; ;…END;
- 条件ループ　　例 DO I=1 TO 10 WHILE(式);

DOループの指標変数(index variable)の初期値はDO文の先頭で評価されることに十分注意させる。また，DO文はFORTRANのそれと異なり，多くの機能を含んでいる反面，まぎらわしい表現もある。次のような例で説明するとよい。

① DO K=1 TO 10 WHILE(A<B);

② DO K=1 TO 10,WHILE(A<B);

(c) IF文

IF文の機能および論理演算について説明する。特にIF文のTHEN，ELSE節ではDOグループかまたはBEGINブロックを用いることにより，プログラムの構造化が図れることを説明する。

1.5 ブロック構造

ＰＬ／Ｉには手続きブロック（ procedure block ）とＢＥＧＩＮブロックがあることについては1.2で説明した。

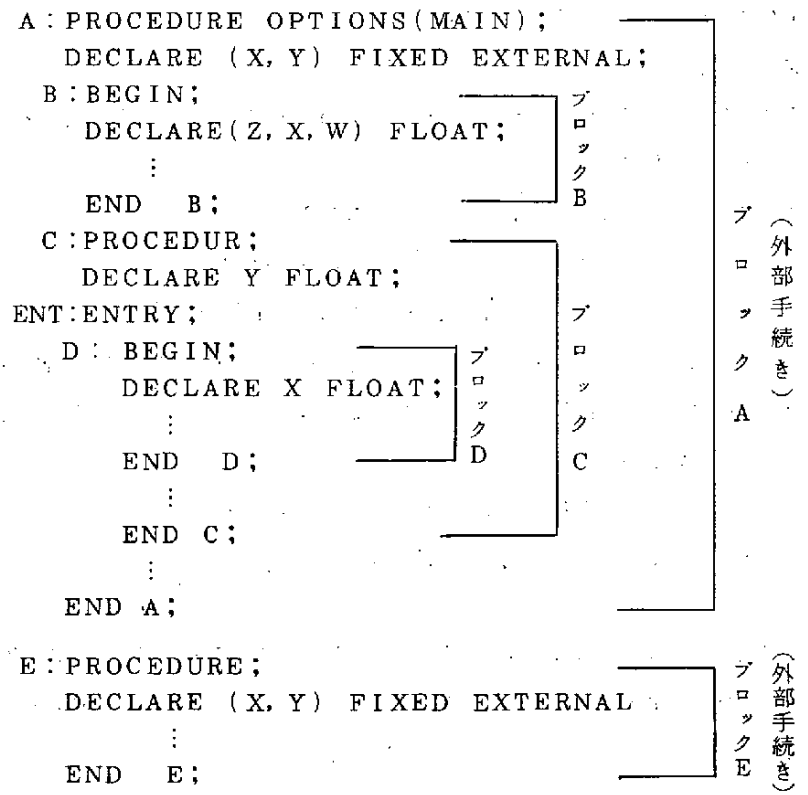
プログラムの構成を比較的少ない文の集りでまとめてブロック構造にすると，プログラムが読み易くなり，明瞭になるだけでなく，デバッグや保守が容易になる。ＰＬ／Ｉにはこの目的を満足させるようなブロック構造の機能があることを理解させる。また，ブロック構造の概念がＰＬ／Ｉプログラムで重要な意味を持つのは，そのブロックの中で宣言された名前およびブロック中の名札の有効範囲が決定されるからである。

ブロック構造に関して以下の項目について十分理解させる。

- 手続きブロックとＢＥＧＩＮブロックの形態と処理機能
- 個々のブロックの制御の流れ
- 手続きブロックとＢＥＧＩＮブロックの終了の契機

手続きの起動と終了の概念を十分に理解することは，ブロック間のデータ（名前）の受け渡しが，その名前が宣言された方法や属性だけでなく，この概念によっているだけに重要である。記憶域の割当てや解放もこの起動と終了の概念に基づくものであることを把握させる。

例　ブロック構造



1.6 記憶域の割当て

一般にプログラムの設計においては，プログラムの大きさとデータ領域を小さくすることが重要である。ＰＬ／Ｉには記憶域割当て方式に下記の４種類がある。

- 静　的（ static ）
- 自動的（ automatic ）
- 被制御的（ controlled ）
- 基底付き（ based ）

上記の記憶域割当てを用いて，効率的な割当て方と記憶域の管理方法について説明する。

静的記憶域（ static storage ）は一定不変で実行以前に割当てられ，その宣言がなされたプログラムの実行中は存在し続ける。ＦＯＲＴＲＡＮや他の多くのプログラム言語では静的記憶域割当てしかない。

自動記憶域（ automatic storage ）はそれを参照する手続きブロックやＢＩＧＩＮブロックが実行されている時にのみ割当てられる。すなわち，ブロックの入口点で記憶域が確保され出口点で解放される。

被制御記憶域（ controlled storage ）および基底付き記憶域（ based storage ）は実行時にプログラマが陽に記憶域の割当てを制御できるようにするものである。プログラマは実行中，必要な時に記憶域のブロックを割当てたり解放したりできる。

(1) 静的記憶域について実例をあげて説明する。

(2) 自動記憶域について実例をあげて説明する。

記憶域の宣言が何もしていないと省略時解釈により，自動記憶域が割当てられる。

(3) 被制御記憶域について実例をあげて説明をする。

ＣＯＮＴＲＯＬＬＥＤ属性を与えられた変数に対する記憶域は，ＡＬＬＯＣＡＴＥ文あるいはＦＲＥＥ文を使用することにより割当てたり，解放したりすることを理解させることができる。

例　Ｅ×１：

```
PROCEDURE;
    DECLARE   X(100)  FIXED  CONTROLLED,
              Y(100)  CHARACTER;
        :
    ALLOCATE   X;
        :
    FREE   X;
        :
    END  EX1;
```

ＡＬＬＯＣＡＴＥ文によりＣＯＮＴＲＯＬＬＥＤ変数に記憶域を割当てる時，この記憶域を指し示すポインタ（Ｐｏｉｎｔｅｒ）を同時にセットする機能について説明する。

例

```
ALLOCATE X SET(P);
```

同一変数に対していくつかの記憶域を割当てた場合，その変数は世代を持つといわれる。ＣＯＮＴＲＯＬＬＥＤ変数については最新世代しか参照できない。

(4) 基底付き記憶域

ＢＡＳＥＤ属性を持つ変数（基底付き変数）が割当てられる記憶域である。被制御記憶域と同じようにＡＬＬＯＣＡＴＥＤ文の実行で生成し，ＦＲＥＥ文の実行で解放する。基底付き記憶域は宣言内で指定されるポインタ変数を持ち，これは常にその構造の最新の実体を指すようにセットされる。

例

```
DECLARE   X   BASED(P); ..... (a)
    :
ALLOCATE  X   SET(P); ..... (b)
    :
```

Ｘをポインタ Ｐの基底となる基底付き変数（based variable）と呼ぶ。ＰにはＸが割付けられた記憶域のアドレスが設定される。

1.7 手続き処理

ＰＬ／Ｉの手続きには外部手続き(external procedure)と内部手続き（internal procedure）があり，それぞれに関数手続き（function procedure）とサブルーチン手続き（subroutine procedure）がある。これらの手続きについての情報と制御の授受の仕方について把握させる。

- 手続き
  - 外部手続き
    - 関数手続き
    - サブルーチン手続き
  - 内部手続き
    - 関数手続き
    - サブルーチン手続き

(1) 手続きの呼び出し方法

手続きは関数としてもサブルーチンとしても使用でき，これはＦＯＲＴＲＡＮの関数副プログラムとサブルーチン副プログラムによく似ている。

サブルーチン手続きはＣＡＬＬ文によって呼び出され，関数手続きは式の中の被演算子とし

て呼び出されることを把握させる。呼び出される手続きは，外部手続きでも内部手続きでもよい。

情報の授受は一般にはアーギュメント（ argument ）とパラメタ（ parameter ）を用いる。アーギュメント・リスト（ argument list ）とパラメタ・リスト（ parameter list ）の役割りは，その手続きを呼び出す方法によって決められる。

(2) アーギュメントとパラメタ

許されるアーギュメントとパラメタのデータ形式や属性およびそれらの関係等について理解させる。アーギュメントが定数，入口名，式で演算子を含むもの，式で括弧の中にあるもの，式でそのデータ属性がパラメタのデータ属性と一致しないもの，および関数の呼び出しでアーギュメントを持つ場合には，仮アーギュメント（ dummy argument ）が作られ，そこで作られた値の記憶アドレスが呼び出された手続きに渡されること（ call by reference）を理解させる。

アーギュメント（ argument ）として許されるのは以下の通りである。

- 式
- データ要素
- 入口名（プログラマが定義したもの）
- 組込み浮動算術総称関数名
- ファイル名

アーギュメント（ argument ）とパラメタ（ parameter ）の対応づけでは以下のことを特に注意させる。

- パラメタとアーギュメントの個数，データ属性は一致しなければならない。
- パラメタは，レベル１の名前でなければならない。
- パラメタは，呼び出された手続きの中で宣言されなければならない。
- パラメタとアーギュメントの構造体は一致させるか，相似的構造でなければならない。
- 配列アーギュメントの次元数と上下限および列の長さは，対応するパラメタのそれと同じでなければならない。
- パラメタがスカラ名札変数ならば，アーギュメントもスカラ名札定数か変数でなければならない。またパラメタが配列の名札変数ならば，アーギュメントも配列名札変数でなければならない。

1.8 リスト処理機能

(1) 基本概念

以下に示すリスト処理の基本概念を理解させる。

リストは一般にはある種の要素，構造体が1列に並んだものであって，リスト処理はこうした要素の並びをある関係で作成，連結，削除，追加するための環境作りをするものである。この機能により，情報検索やシステム・プログラム等で，高度に効果的なプログラム手法に活用したり，記憶装置を弾力的に利用したりすることが可能である。

リスト処理では原始プログラムにポインタが使われ，それによりデータのリストをアクセスしたり，その要素を指すことができる。次のような例を用いてポインタの概念を把握させるとよい。

```
DECLARE   P   POINTER,
          B   FLOAT,
          A   FLOAT BASED(P);
          ⋮
          B = P -> A;
```

上例ではPがポインタであり，Aをアクセスする際には，その位置がPによって指定されるものであることを宣言している。

(2) リスト処理機能

PL／Iでのリスト構造の処理を，次の機能を中心に説明する。

(a) ポインタ型の名前 … これにより情報構造間の連結を明白に指定し処理できる。

(b) 構造宣言 … これにより異なる型のポインタ・フィールドおよび値フィールドを持つリスト要素を明白に宣言できる。

(c) 制御記憶割当て … これにより構造を入れ子形にしたり削除したりできる。

```
DECLARE   1  A  BASED(P),
            2  PTR  POINTER,
            2  DATA  FIXED;
```

この宣言はPが構造Aにつくポインタであり，Aの構造はポインタ型のフィールド名PTRと，固定小数点型のフィールド名DATAから構成される。この宣言の有効範囲内に，ALLOCATEA があれば構造の実体Aが生成され，ポインタPがその最新の実体を指し示す。また，この構造中のポインタ・フィールドはALLOCATEされた他の構造を指すためにも使用できる。

ポインタ変数を活用すると，基底付きで記憶域を割当てられた構造リストに構成できる事

を例示して説明する。次の例は構造ＴＡＢＬＥの三つの実体を連鎖で結びリストにするものである。

例

```
DECLARE (Q,HEAD) POINTER;
DECLARE 1 TABLE BASED(P),
          2 TONEXT  POINTER,
          2 DATA    FIXED ;

① ALLOCATE TABLE;
  HEAD = P;
  Q = P ;
② ALLOCATE TABLE;
  Q->TONEXT = P ;
  Q = P ;
③ ALLOCATE TABLE;
  Q->TONEXT = P ;
  TONEXT = NULL ;
```

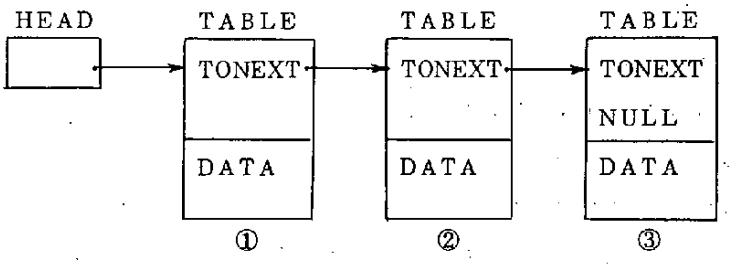
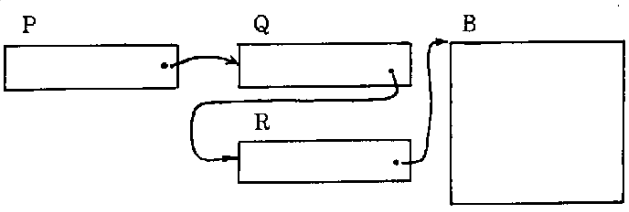


図２－１　リスト構造図

記号－>はポインタの修飾（ pointer qualifier ）を意味する。

ポインタ変数に値を設定するにはＡＤＤＲ組込み関数やＮＵＬＬ組込み関数を用いることもできることを説明する。またポインタ修飾はポインタ変数による一段の修飾だけでなく，多段の修飾ができることを説明する。

```
DECLARE   B(10,10) FLOAT BASED,
          P POINTER ,
          Q POINTER ,
          R POINTER ;
          ⋮
          P->Q->R->B(1,1) = 10;
```



1.9 割込み動作

割込み処理についての一般概念とその処理方式について，まず，理解させ，次にＰＬ／Ｉにおける次のような割込み処理の形態や機能を理解させる。

- 条件接頭語の目的
- 条件接頭語の範囲
- ＯＮ文の用法
- システムの割込み動作
- ＳＩＧＮＡＬ文，ＲＥＶＥＲＴ文の用法
- プログラムの制御の流れ

割込み条件として指定できるシステムの条件は次のように分類できる。

(1) 計算上の条件　：データの型変換あるいは式の計算において生じる条件。

(2) 入出力条件：　データの入出力時に発生する条件。

(3) プログラム検査用条件　：プログラムのデバッグと追跡のときに使われる。

(4) リスト処理条件：　記憶域割当てと関連して発生する条件。

ＯＮ文で宣言した手続きは，そのＯＮ文を実行した時点で有効となる。また，ＯＮ文で指定された条件に対する中断行動は，次の三通りの方法で中止あるいは終了されることを説明する。

(1) ＯＮ文を含むブロックから出る場合。

(2) 同じ条件に関して別のＯＮ文が実行される場合。

(3) その条件に関してＲＥＶＥＲＴ文が実行される場合。

1.10　翻訳時の機能

本節ではＰＬ／Ｉの翻訳時に扱える文や手続きの種類，形態および機能について説明し，さらにこの機能の効果的な使用法について説明する。

翻訳時の機能（マクロ機能）は通常のコンパイラへの前処理ルーチン（pre-processor）によって取り扱われる言語レベルで定義される。前処理ルーチンではコンパイル時の文を判別し，これを解釈し，それに従って原始プログラムを修正追加の操作を行う。

ＰＬ／Ｉの翻訳時の機能（マクロ機能）はアセンブリ言語での条件付きアセンブリ機能やマクロ機能に類似している。

プログラムの中で頻繁に扱う処理手順や，外部手続き間で共通な制御表（データ域）はライブラリに登録しておき，これをプログラム中の必要な所で％ＩＮＣＬＵＤＥ文によるマクロ呼び出しを行い，プログラム・テキストに編入するようにする。

```
％　INCLUDE　＝　登録名；
```

こうすることによりプログラムが明解になり，修正，保守等において誤りが少く，容易に行う事ができる。また，ライブラリ化する事により，多人数の人で共用でき，プログラム作成工程の短縮化を図ることができる。さらに，一定のパターンに従うテキストは毎回書かなくても，テキストを作り出す翻訳時文を翻訳時手続きとして定義しておけば，翻訳時にそれを呼ぶだけでよい。こうしたマクロ機能についての効果的な使用法を例事して説明する。

例１）　大構造体名だけが異なる２の構造体を原始プログラム上に作り出す。

ＰＡＹＲＬという名のファイルが次のような構造体の宣言文から構成されている場合，

```
DECLARE 1 PAYROLL,
          2 NAME,
            3 LAST CHARACTER(30),
            3 FIRST CHARACTER(15),
            3 MIDDLE CHARACTER(3),
          2 MAN-NO FIXED DECIMAL(6,0),
          2 HOURS,
            3 REGULAR FIXED DECIMAL(8,2),
            3 OVERTIME FIXED DECIMAL(8,2)。
          2 RATE,
            3 REGULAR FIXED DECIMAL(8,2),
            3 OVERTIME FIXED DECIMAL(8,2);
```

翻訳時機能により，次のプログラムはＣＵＭ－ＰＡＹとＰＡＹＲＯＬＬという大構造体名だ

け異なる2個の同じ構造体を原始プログラム中に作り出す。このように，ライブラリを利用できることを注意させる。

```
%  DECLARE  PAYROLL  CHARACTER;
%  PAYROLL='CUM-PAY';
%  INCLUDE  PAYRL;
%  DEACTIVATE  PAYROLL;
%  INCLUDE  PAYRL;
```

例2）　次の一連の文はDOループの実行時間を早くするために用いられる。

```
%DCL  I  FIXED;
%DO  I=1  TO  4;
A(I) = B(I)+3*C(I);
%END;
%DEACTIVE  I;
```

これにより，次の一連の文が生成される。

```
A(1)  =  B(1)  +  3*C(1)  ;
A(2)  =  B(2)  +  3*C(2)  ;
A(3)  =  B(3)  +  3*C(3)  ;
A(4)  =  B(4)  +  3*C(4)  ;
```

これは原始プログラム中の次のDO文と同じでないことに注意させる。すなわち以下の場合ではIの終りをチェックするような命令が必要となる。

```
DO  I=1  TO  4;
A(I)=B(I)  +  3*C(I);
END  ;
```

例3）　原始プログラムのある部分だけを編成翻訳する場合

```
%PLUS    =  '+';
%TIMES   =  '*';
%START   =  'PROCEDURE OPTION (MAIN)';
%FINISH  =  'END';
EX1  :  START;
          A=(B  PLUS  C)  TIMES  D;
          FINISH  EX1;
```

翻訳時機能により次のようになる。

```
EX1 : PROCEDURE OPTIONS (MAIN);
      A=(B+C) * D;
      END   EX1;
```

指導上の留意点

(1) ＰＬ／Ｉの細部まで理解させるのは困難なので，処理プログラムで扱う機能の範囲に抑えて説明する。

(2) １３章の総合演習を遂行するための最低限の知識を身につけさせる。

(3) 本章では，単にＰＬ／Ｉ言語を理解させるだけでなく，プログラムを構造的に把握させることが必要である。したがって，グループ，ＢＥＧＩＮブロックおよび手続きブロック等をプログラムの構造の面から説明する。

(4) 割込処理については，その概念を理解させる程度でよい。

(5) リスト処理についていくつか実例をあげて，効果的な使用法をよく理解させる。

(6) 翻訳時機能については，プログラムの静的な構造面からとらえて，マクロ機能の効果的な使用法を説明する。またこの翻訳時機能により，より強力かつ専門化した言語への拡張ができることを理解させる。

参 考 文 献

[1] John J. Donovan 著　池田克夫訳
「システム・プログラムⅠ，Ⅱ」日本コンピュータ協会，1974

[2] Peter Wegner, "Programming" Languages, Information Structures, and Machine Organization", McGraw-Hill, 1968
菅野宏訳「言語・情報構造・計算機の理論」日本経営出版会，1972

[3] 日本ＩＢＭ，ＩＢＭシステム／３６０　ＰＬ／１解説書，Form Number NC28-8201 1969

第2章　プログラム間のコミュニケーションと結合

キーワード

値とり（call by value），名前換え（call by name），番地取り（call by refer-erceあるいはcall by address），結果取り（call by resuet），コントロール・セクション（control section），外部記号（external symbol），呼び出し手順（calling sequence），パラメタ・リスト（parameter list），連係編集プログラム（linkage editor）

目　　標

大規模プログラムになればなるほど多数のサブルーチン（モジュール）に分割し，多人数でプログラムを作成しなければならない。したがって，自ずと各モジュール間のインタフェースで誤解やミスが起りがちであり，デバッグや保守が円滑に進まないケースが往々にしてある。各モジュール間のインタフェースに規則性を持たせ，繁雑な特殊処理をできるだけ排除するようにすることが必要である。そのためにプログラム作成規約を作り，モジュール間のインタフェース規定を定めるのが望ましい。

本章ではモジュール間のコミニケーションの方法と慣行等を修得させる。また，アセンブルあるいはコンパイルされたモジュールを相互に結合し，実行可能なプログラムに加工される過程等について理解させる。

内　　容

2.1　データの受け渡し

サブルーチン（subroutine）のデータ授受の仕方には，パラメタ，共通領域，外部名，ファイル等がある。これらにアクセスする場合のオーバヘッドおよびその各々の特徴と概念を理解させる。

また，パラメタがプログラム上でどのように実現されているか，サブルーチン側でパラメタをどう参照するかについて理解させる。

(1)　パラメタによる方法

(a)　パラメタの参照

パラメタによるデータの引渡し方法には次の四つの方法がある。これらについて説明す

る。特にパラメタが演算子を含んだ式の場合等について十分解説する。

- 値とり(call by value)

  データ値そのものを渡す。呼ばれた手続き側では，アーギュメントに対して局所的変数が与えられており，呼び出された時にパラメタの値が評価され格納される概念を把握させる。

- 名前換え（call by name）

  その名前もしくわそのアドレスが指定されること。呼び出される手続き内のアーギュメントをすべてパラメタの記号表現で置換したのと同じ効果を及ぼす。すなわち，パラメタの値（またはアドレス）はアーギュメントの登場のたびに評価されることを理解させる。

- 番地取り（call by reference，あるいは call by address）

  アーギュメント・リスト中には，パラメタの値を指すアドレスが入っており，呼ばれる手続き側では，このアドレスをそのまま対応するアーギュメントに対して使用することを説明する。

- 結果取り（call by result）

  ALGOLW言語において，アーギュメントXがRESULTアーギュメントであると宣言された場合，Xに対して，この手続きのデータ領域内に記憶域がとられ，手続き内のXの使用に際しては，この記憶域が使用される。値取りの場合と同様に，パラメタのアドレスが記憶域Xに格納されることを理解させる。

(b) パラメタの内容

パラメタについて以下のことを理解させる。

- 処理するデータや処理結果を格納するアドレス
- 使用する表名
- 表の長さ，配列の要素数
- サブルーチン名
- サブルーチンへの処理の指定
- ファイル名
- 特殊な分岐先のアドレス
- パラメタの数……異常処理に対処
- 完了コード

特に完了コードについてはデバッグ，保守のし易さなどから積極的に利用するよう次の理由を理解させる。

すなわち呼び出し元に制御を返却する場合には必ず正常，異常および警告等の完了コードを設定するのである。

なお，重大なエラーについてはそれを検出したサブルーチンで適切なエラーメッセージを出力する。たとえばファイル処理などのサブルーチンで適切なエラー・メッセージを出してくれなければ，そのサブルーチンを呼び出した側では，完了コードだけでは適切なエラー・メッセージを出すことはできない場合がある。

(c) パラメタによる方法のオーバヘッド

パラメタによる方法のオーバヘッドについて説明する。

- 各サブルーチンに対して，毎回パラメタの設定取り出しの命令が必要。
- レジスタの退避／回復操作。

(2) 共通領域による方法

コントロール・セクション，コモン・セクション等の定義命令に関する概念，機能を説明した後，実例をあげて更に理解を深めさせる。これらのセクションはシステムにより異なっているので，以下を参考にして，そのシステムに合った具体例で説明するとよい。

(a) コントロール・セクションとオペレーティング・システムとの関連

- ベース・レジスタとの関連
- プログラムの格納や連係編集との関連

(b) サブルーチン間での領域の相互参照との関連

特に高水準言語（ＦＯＲＴＲＡＮ）で書かれたプログラムとの相互参照（たとえばＦＯＲＴＲＡＮの名前付共通ブロックとコントロール・セクション，無名共通ブロックとコモン・セクションとの関係等）

(3) 外部記号（ｅｘｔｅｒｎａｌ ｓｙｍｂｏｌ）

独立にアセンブルされた他のプログラムから参照される記号，および他のプログラム中で定義されている記号（外部記号という）を参照するために以下のような命令が用意されている。下記のものについて機能，使用法を説明する。

(a) 外部から参照される記号

一般にＥＮＴＲＹ，ＧＬＯＢＡＬなどの命令で定義する。

(b) 外部で定義されている記号の参照

ＥＸＴＲＮ命令（ｉｄｅｎｔｉｆｙ ｅｘｔｅｒｎａｌ ｓｙｍｂｏｌ）などで宣言される。

なお，ベース・レジスタなどで実行時に命令の修飾を行なうシステムでは，外部記号で定義されたデータの参照，制御，受け渡し（特に入口点（エントリ・ポイント）が複数の場合）が複雑になる場合が多いので実例をあげ，プログラム間での問題も含めて説明する

とよい。

(4) レジスタによる方法

受け渡しデータをレジスタに設定する。

関数の値は一般にこの方法により行なわれる。

(5) ファイルによる方法

データの授受をファイルによる方法とその利点を説明する。

- インタフェースのミスが少くインタフェースの形式がきれいである。
- デバッグがし易い。
- 大量のデータを扱う場合に適している。

2.2 呼び出し手順

モジュール間の制御の授受の仕方について説明する。

(1) 分岐命令による方法

分岐命令による連係として，通常次のような方法がある。

① 外部記号の名前を使ってV形式のアドレス定数を作る。

② この定数を汎用レジスタにロードし，このレジスタを通して外部のコントロール・セクション（サブルーチン）へ分岐する。

たとえば図2－1は，ＳＩＮＥという名前のコントロール・セクションに関係する場合を示している。

```
MAIN    CSECT
BEGIN   BALR    2,0
        USING   2
          .
          .
          .
        L       3,VCON
        CNOP    2,4
        BALR    1,3
        パラメタ・リストあるいは
        パラメタ・リストへのアドレス
          .
          .
          .
VCON    DC      V(SINE)
```

図2－1 ＩＢＭ－370アセンブラの分岐命令の例

(2) マクロ命令による方法

モジュール間のインタフェースの誤りをなくすために呼び出しマクロを使うことが有効であることを理解させ，また次のような利点を説明する。

- このプログラムで統一的にエラー・チェックをする
- 省略時解釈によって統一的にパラメタ・リスト（parameter list）を整備する。

(3) 高水準言語（FORTRAN，PL／I等）におけるCALL文による方法は結局はコンパイラにより翻訳されて(1)の方法になる。

2.3 レジスタの割当て

一般に主プログラムとサブルーチンとの間の制御の受け渡しを行う場合，オーバヘッド（overhead）の減少，誤りの防止等のためレジスタの使用に関する規約を設ける必要がある。ここではそれらの目的のためどのような規約を設けているかを特定のオペレーティング・システムについて説明をする。

2.4 連係編集プログラム

この節ではコンパイラやアセンブラで生成された複数個の目的プログラムを連係編集プログラムで結合，編集し，1個のロード・モジュールを作成し，これを実行する過程を解説する。

(1) 機能

下記に示すような連係編集プログラムの機能を説明する。

- コンパイラで生成された目的モジュールをロード・モジュール（実行形成）に変換する。
- 複数個の目的モジュール（あるいはロード・モジュール）を結合して1個のロード・モジュールを作成する。
- プログラム・ライブラリから必要なプログラムを選択し，自動的にモジュールを組込む（自動呼び出しによる結合）
- 計画的オーバレイ構造のプログラムを編集する機能

(2) 入出力

具体的にあるシステムの連係編集プログラムの入出力について説明する。

(a) 入力

連係編集プログラムの入力として次のようなものがある。

- アセンブラが出力した目的プログラム
- 各種コンパイラが出力した目的プログラム
- 連係編集プログラムが出力したロード・モジュール

(b) 出力

ロード・モジュール，各種リストやメッセージについて説明する。

(3) 制御文

連係編集プログラムを起動するための制御文について，具体的にあるシステムのものについて，その記述法および機能について説明する。

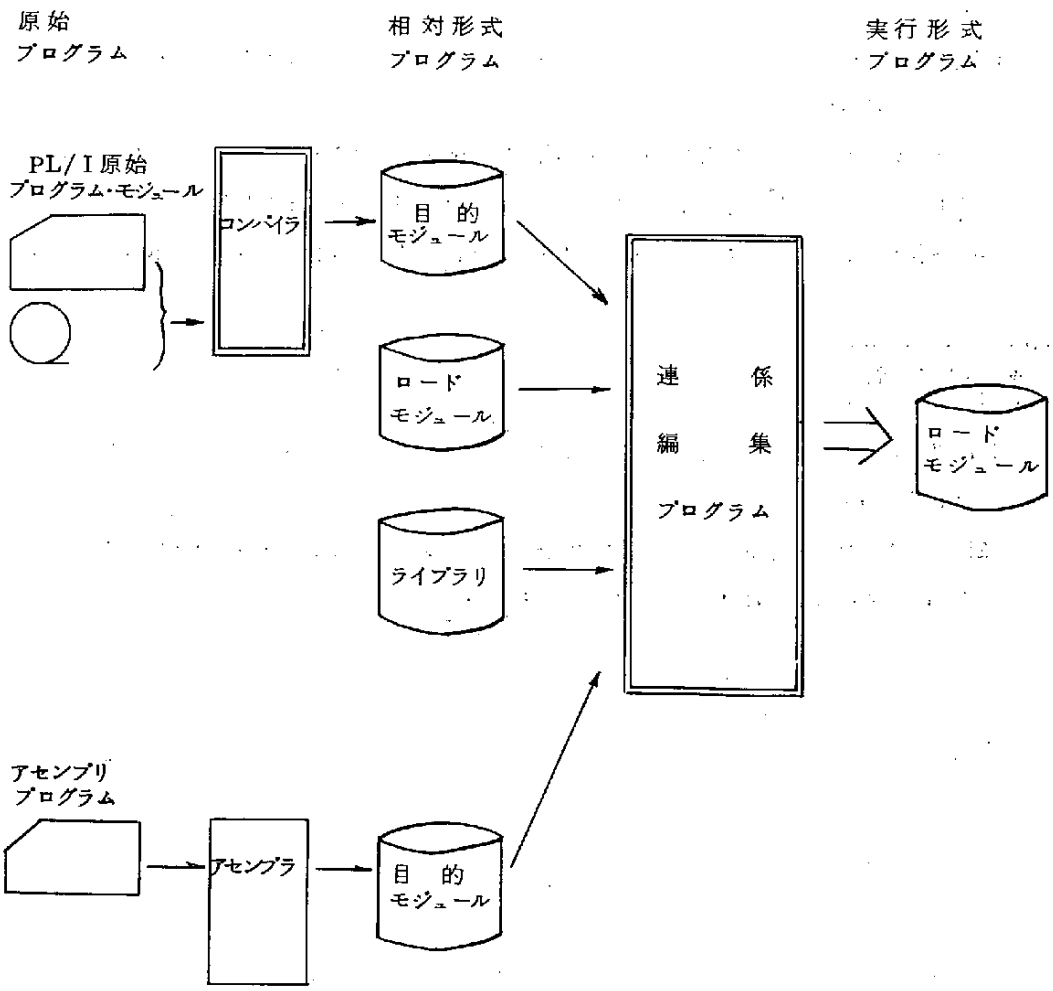


図2－2　連係編集プログラム機能

指導上の留意点

(1) 大規模プログラムになれば，プログラムを無数に分割して多人数の人によって作成される。この場合に，分割されたプログラム間のインタフェースが各種まちまちで輻湊していると，バグ発生の最大の原因となる。そのためにインタフェース規約を作成し，同一のパターンでプログラム間のデータや制御の授受を行いインタフェースを明解簡素にすることの重要性を理解させる。

(2) データの授受の仕方では，それぞれの場合に応じて最適な方法を選択することが重要であることを理解させる。

(3) 多数のモジュールを多人数で作成する場合に，制御の授受は入口マクロ，出口マクロを採用して，統一的な管理をすることにより，バグの発生を防止できることを把握させる。

(4) 具体的なシステム上でベース・レジスタの概念とアドレスの形式について，適宜に補足説明すること。

参 考 文 献

[1] 江村潤朗「オペレーティングシステムの構造的アプローチ（上巻）」日本コンピュータ協会，1972

[2] 日本IBM，IBMシステム／360 オペレーティング・システムアセンブラ言語，Form Number GC28—6514—5，1966

第3章　プログラムの実行時の構造と連結

キーワード

ローディング(loading)，プログラム・セグメント(program segment)，再配置可能性(relocatability)，オーバーレイ(overlay)，計画的オーバーレイ(planned overlay)，動的再配置(dynamic relocation)，動的リンク(dynamic link)，リンケージ・フォルト(linkage fault)，セグメンテーション(segmentation)，監視プログラム呼出し(supervisor call, SVC)，コルーチン(coroutine)

目　標

プログラムは最終的にはコンピュータの主記憶上にロード(load)されて実行されるが，主記憶の容量には限りがあるので，プログラム全体を一時に主記憶に格納して実行することが常に可能とは限らない。特に多重プログラミング(multiprogramming)を行っている環境においては，たとえ主記憶の容量が十分大きく，プログラム全体を格納する余地があったとしても，運用上，プログラムの実行時の大きさに上限を設けることが全体の効率上必要とされる場合が多い。したがって主記憶の効率的な使用方法について種々のテクニックが必要とされる。主記憶へのプログラムのローディング(loading)は専らオペレーティング・システムが司る役割であるから，オペレーティング・システムごとの特徴もよくわきまえて適切なローディング方法を選択するための知識を与えることを目標とする。

さらに，プログラムは連係編集プログラムによってリンク(link)を静的に行なうだけでなく，いろいろな場面で動的にリンクする必要がある。動的リンクの必要性を理解させ，どのような場合に効果的に用いることができるかを説明する。また，監視プログラム呼出し(supervisor call)の動作原理についてより深い理解をさせるようにする。割込み動作とそれによるユーザ・プログラムへの制御の渡し方についても説明する。並行型プロセスのシミュレーションや，シミュレーション用プログラムで取扱われる特別な形の制御の渡し方をするコルーチン(coroutine)について理解させる。

内　　容

3.1　プログラムの実行時の構造

(1)　プログラムの構造

プログラムのローディングとは結局のところ，データを主記憶に読込む操作であるということを理解させる。したがってプログラムの全部もしくは一部分のローディングを何回か繰返すときにはファイルからのデータの入出力の際生じるような問題，たとえば入出力機器の低速性，入出力命令を発信しデータのやりとりを行なうためのオーバヘッド等を考慮しなければならないことを説明する。

プログラムの実行時の一番簡単な構造は，全プログラムを一時に主記憶に格納して実行させるやり方でこれを単純構造（simple structure）と呼んでいる。

プログラムが大き過ぎたり，あるいは実行に際してすべての部分が実行されるとは限らない場合にはプログラムの一部分を主記憶に入れて実行を開始し，必要に応じて動的にプログラムの残りの部分を主記憶にロードしながら実行することが考えられる。このように分割されてローディングの対象となるプログラムの一部分をプログラム・セグメントと呼ぶ。プログラム・セグメントをこのように動的に主記憶に読み込んで，同じ主記憶に何回かプログラムをロードする方式としてオーバーレイがある。

オーバーレイについて次の点を説明する

(i)　プログラム・セグメントの配置の決め方。特にプログラム内で配置の位置までも意識する方式とプログラム内では位置について意識せず代りに連係編集プログラム（linkage editor）でプログラムを編集するときに位置関係を定めることができるいわゆる計画的オーバーレイ（planned overlay）の差について

(ii)　オーバーレイに適したプログラムの性質。実行時に何回も同じプログラム・セグメントが呼出されるようなプログラムはオーバーレイに適していない。これに対して初期設定処理あるいは後処理など実行に際して一回しか制御が渡らないようなプログラムの部分はオーバーレイに適す

(iii)　オーバーレイを利用したときに生ずるプログラムの制限。オーバーレイによって共通に使用される主記憶は何度も書き替えられるので情報の保存が効かなく思わぬエラーの原因となる。

(iv)　互いにオーバーレイされるプログラム・セグメント間の制御の移し方。特に計画的オーバーレイでＣＡＬＬ命令などのマクロ命令により，呼び出されるプログラム・セグメントの位置関係がどうあろうとプログラムをコーディングするときには同一の命令でよいこと

及びそれによる利点について。

(v) プログラムの複雑さが増加したときの欠点について，オーバーレイはプログラムの構造が複雑になったときには非常に繁雑になることを指摘する。そのためプログラマはプログラム・セグメント間の関係，呼び出し順序などについて注意深い設計をする必要がある。また，連係編集の手続きが大変面倒となる。前者を一挙に解決する手段としてページング（paging）による仮想記憶装置（virtual memory あるいは virtual storage）が出現し，後者のために動的リンク（dynamic link）のテクニックが出現したことを説明する。動的リンクについては後節で更に詳しく指導する。

(2) プログラムの再配置

プログラムはコンパイル後編集されてロード・モジュール（load module）となって利用されるが，毎回実行時の主記憶上の位置が変りうる。そのためローディングの際プログラムの格納場所を任意のアドレスから開始できるような工夫が必要である。このようにプログラムの格納位置をローディング時に変更し得る能力を再配置可能性（relocatability）という。

再配置可能性について次のことを説明する。

(i) アセンブリ言語を用いて，プログラムの再配置が可能であるためには機械命令（machine instruction）のオペランドが絶対アドレス（absolute address）であるか相対アドレス（relative address）であるかの識別が必要なことを教える。

したがって一般のロード・モジュールにはこれらの情報がテーブルとなって存在する必要があることを指摘する。

(ii) ハードウェアの機能によって再配置可能プログラムの実現方式がいろいろありうることを説明する。

(a) ベース・レジスタ（base register）を用いる方式のもの

(b) プログラム・カウンタ（program counter）によるアドレスの修飾を行なう方式のもの。この方式は自己相対アドレス指定（self-relative addressing）といわれる。

プログラムの再配置はプログラムのローディング時に必要なだけでなくプログラムの実行時に行なえると便利なことがある。プログラムを実行時に一時中断し，別の場所に格納しなおしてもなおかつ動作可能な機能を動的再配置（dynamic relocation）という。

動的再配置について次のことを説明する。

(i) 動的再配置の効用

多重プログラミングを行なっているとき

(a) ロール・アウト（roll-out）されたプログラムを再びロール・イン（roll-in）す

るとき任意のアドレスに置くことができる。

(b) 主記憶の割当てに無駄が生じたときプログラムの詰め合せ（compaction）を行い，連続した空き領域をつくり出す。

等の利点があること

(ii) 動的再配置の実現方法

(a) ハードウェア上の考慮

たとえばベース・レジスタの役割とかあるいは自己相対アドレス指定を可能にするハードウェア上の機能について説明する。

(b) ソフトウェア上の考慮

動的再配置を可能にするためにはオペレーティング・システムとの緊密な連係動作が必要である。ベース・レジスタを用いる方法においてどのようなメカニズムで動的再配置が実現されるかについてたとえば図3－1のような図を使って説明するとよい。

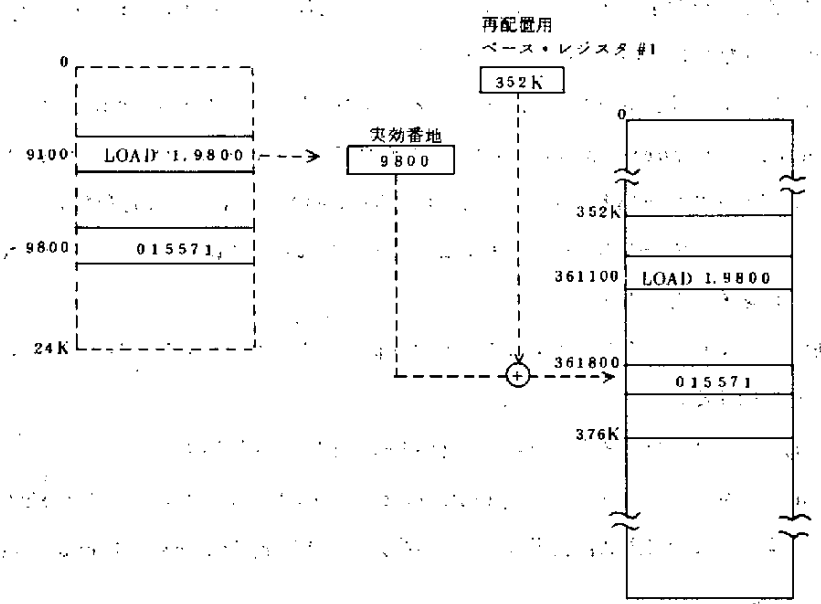


図3－1 ベース・レジスタを用いての動的再配置可能なプログラムの実行

3.2 プログラムの実行時の連結

(1) 動的リンク

現在最も普及しているプログラム同士のリンクは連係編集プログラムによって，実行に先立ってアドレス空間（address space）に命令またはデータを割当てる方式である。しかし，リンクすべきプログラムが実行時まで決定できない場合とかプログラムの数が非常に大きい場

合など上の方式が使用できないかあるいは使用したとしても効率が非常に悪い場合がある。このような場合，プログラム同士のリンクを実行時に行なう動的リンクのテクニックが使える。動的リンクが使用される次の二つのケースについてその典型的な使い方を説明する。

(i) ユーザが自分でジョブ起動プログラムを作る場合。

監視プログラムになり代ってユーザ・プログラムが任意のプログラムを起動したいと考える場合がある。たとえば原始プログラムの変更とそれにもとずいての再コンパイル，再リンク，実行などをジョブ制御言語の助けを借りて行なうのではなく，端末に坐っているユーザの応答を反映させながら行おうとするような場合である。

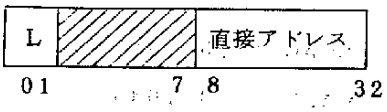
ユーザが指定するプログラムはあらかじめ予想がつかないから，ユーザのジョブ起動プログラムは動的リンク機能によって指定プログラムにリンクする。

(ii) システムを構成するサブルーチン等の数が多過ぎる場合。

多数のサブルーチンを使用する大きなプログラムの場合，連係編集プログラムを用いたプログラムの連結処理は指定方法が複雑であるだけでなく，処理自身に多大の時間を必要とする。たとえばオペレーティン・システムのシステム編集がその例である。動的リンクは巨大システムのシステム編集を容易にするのに役立つ。

動的リンク方式の例としてMULTICS流のセグメンテーション（segmentation）を利用したシステムを次の項目に従って説明する。

(i) 間接アドレス語の形式



L：リンケージ
フォルト表示ビット

図3－2　間接アドレス語の形式

(ii) 間接アドレス指定（indirect addressing）によって名前によるアドレス参照を行なうメカニズムの説明

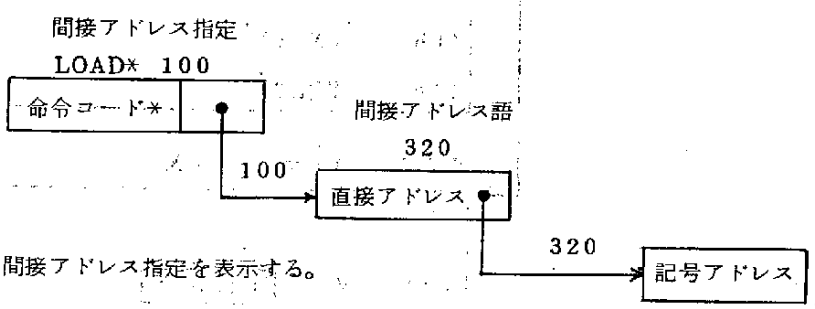


図3－3　間接アドレス参照メカニズム

(iii) 間接アドレス語にリンケージ・フォルト指示ビットが1とたてられていてかつその語が間接アドレス語として使用されるとリンケージ・フォルト(linkage fault)が発生し，オペレーティング・システムによって間接アドレス語がポイントする記号アドレス(symbolic address)を直接アドレスに変換する。

直接アドレスは間接アドレス語に書き込まれLビットは0にリセットされる。

(iv) 具体例として図3－4を用いるとよい。

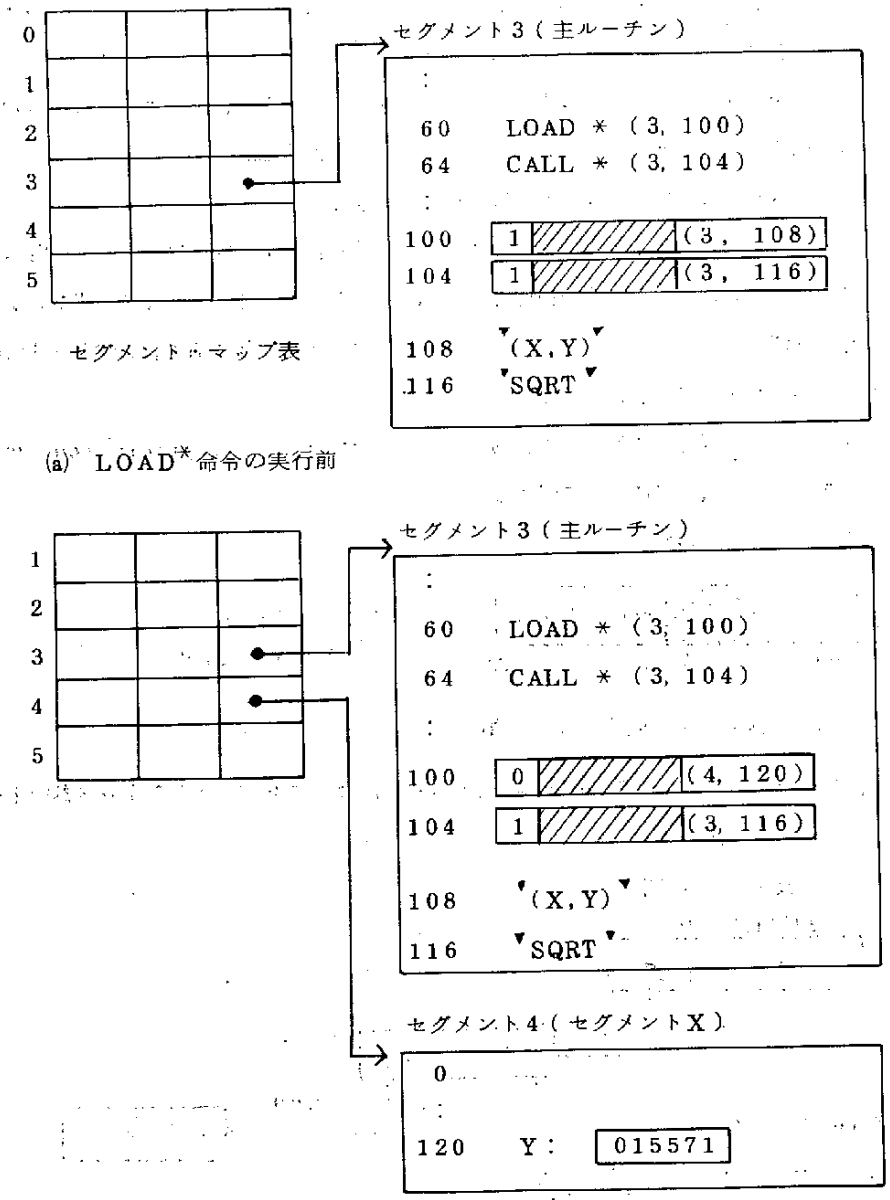


図3－4 セグメンテーションによる動的リンク

(2) 監視プログラム呼出し，割込動作

監視プログラム呼出し（ＳＶＣ）も動的リンクの一例と考えられる。この場合は一般のプログラムからＳＶＣによって指定されたコード（code）によって一意に定まる監視プログラム内の対応処理ルーチンに制御が渡る。

監視プログラム呼出しについて次の事項を説明する。

(i) ＳＶＣ命令が実行されるとＳＶＣ割込みが発生し，制御はＳＶＣハンドラに移る。

(ii) ＳＶＣハンドラはさらにＳＶＣ命令に付随するＳＶＣコードを解析し，正しい使い方がされているかどうかを検査した後，レジスタ等の退避を行って指定ルーチンに制御を渡す。

(iii) 処理終了後は通常の割込み処理と同様にレジスタ等の回復を行ってＳＶＣ実行プログラムに復帰する。

(iv) 一般プログラムから監視プログラムへあるいはその逆へ制御が移るときに監視プログラム状態（supervisor state）にあるか否かが監視プログラムによって容易に判別がつくようにシステムの状態の切り替えが通常行われる。

プログラム走行中に，ある定められた状態が出現したとき，一旦処理を中断してそれぞれの状態に対応した処理をさせたいときがある。このようなメカニズムがあれば命令実行のたびごとに一々ある定められた状態が成立しているかどうかをチェックしなくてもすむ。このようなメカニズムを割込み（interruption）という。割込み成立の条件としては複雑な条件を任意にセットすることはできなく，オペレーティング・システムでそのバラエティは定まっている。PL／ＩのON文を使って，割込み条件の種類，設定方法，割込み成立に伴った割込み処理ルーチンへの制御の移動方法，元のプログラムへの復帰方法について説明する。

(3) コルーチン

コルーチン（coroutine）は，個々のプログラムのどれが主ルーチンでどれがサブルーチンというように呼出しに関しての主従関係が存在しなくて互いに相手を自分のサブルーチンであるかのごとく呼出すことのできる制御方法を備えたプログラム群のことである。

一般のサブルーチンと異る点は，コルーチンへの入口点（entry point）はコルーチン呼出しの際に明示しないことである。実際には各コルーチンごとに自分は次回に呼出されたときにどこから処理を始めるべきかという活動記録（activation record）を有していて，そこから処理が実行される。

コルーチンの典型例としてはFORTRANプログラムにおける入出力文の実行時処理プロセッサを用いるとよい。簡単のためWRITE文の処理に話を限定して説明する。

(A) 変数リストを読みとり，呼出されるたびごとに(B)のコルーチンに変数の値を返してやるコルーチン

(B)　FORMAT 文を順に解読し，印字すべき文字を出力バッファに編集し，かつ1行文が完成したときに印字するコルーチン

(A), (B)二つのコルーチンをどのように動かせばWRITE文の処理が実行されるかその概略を説明する。

上の例でも分るようにコルーチンを正しく働かせるためには次の三つの内部操作が必要であることを説明する。

(i)　初期設定。コルーチンごとに活動記録を作り，適切な情報を与える

(ii)　起動。コルーチン名を指定して活動記録に従って制御を渡す

(iii)　後始末。コルーチンが走るとき作られた活動記録，派生したプログラムおよびその活動記録を削除する。

コルーチンの相互の関係の説明には次の図を用いるとよい。

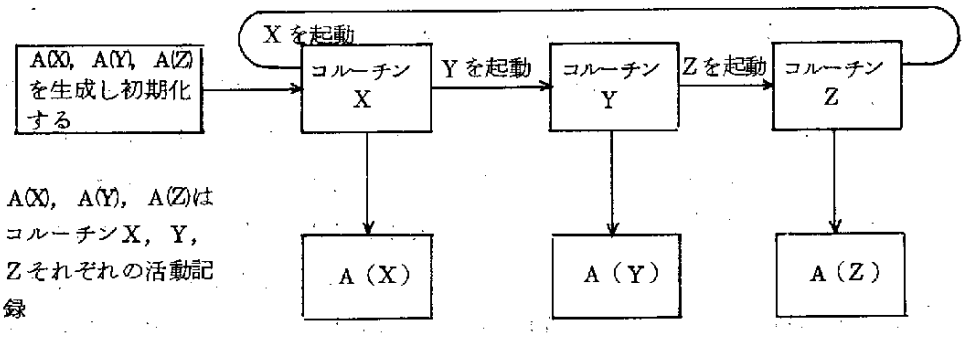


図3－5　コルーチン3個の制御

サブルーチンの起動とコルーチンの起動の差は活動記録の扱い方に差があることを説明する。すなわち，前者においては起動ごとに実行開始アドレスが指定されるから毎回活動記録の内容が新規に作られるのに対し，コルーチンでは一度初期設定した後は呼出されるコルーチンの活動記録の内容が呼出し側によって変更されることはない。サブルーチンでは毎回活動記録の内容が新規に設定されるから特に初期設定というような段階を設けずに毎回活動記録を生成し，初期設定するということを行なう。

コルーチンは概念上は直列（serial）に処理されていくものであるが，CPU1台での並列処理のシミュレーションに有用であることを説明する。

コルーチンの考え方はシミュレーション言語でも有効に使用されている事実を説明する。

指導上の留意点

(1) コンピュータによっては上で述べた方式と異るプログラムの実行時の構造を持つものがある。指導者は現実に利用できる，あるいは使用しなければならないプログラムの実行時の構造とは別に色々の方式を説明し，何故それらの方式が考え出されたかその理由を明確にする。

(2) 活動記録(activation record)という用語はあまり一般には使用されていない。しかし，これはコルーチンのみならず第5章に現れるプロセスの管理のためにも必須の情報であって，名前は異っても同種の制御情報がオペレーティング・システム内部などで使用されている。PCB(process control block) TCB(task control block) などといわれているものがこれに相当する。

この点については5章でも再度取扱う。

(3) シミュレーション言語SIMULAはコルーチンの考え方をたくみにシミュレーションのために応用している。もし余裕があるならばSIMULA(特にSIMULA67)を教えることは非常に価値がある。

参　考

(1) ローディングについては実際のコンピュータの例を用いて説明するのが説得力がある。各メーカのマニュアルもその意味で大いに参考になろう。

(2) ローディングの方法は一通りに限るわけではないから，毛色の異った複数種の方式があることに注意すべきである。

(3) 並列プロセスあるいは並列タスクのシミュレーションをコルーチンで行うとしたとき，プロセスに対応するものは何で，プロセス切替えに対応するものは何か

(答) 個々のコルーチン，コルーチンの起動にそれぞれ対応する。

(4) プロセス(あるいはタスク)については第5章で取扱われる。

(5) コルーチンといったとき，コルーチンどうしが同時に実行されることはなく，瞬時には一つのコルーチンしか実行されていない。これに対し，プロセスもしくはタスクといわれるものは，プロセッサが複数個ある場合には同時に複数個が実行されていることがある。このいみでコルーチンはプロセスをシミュレートしたものにある。

参 考 文 献

[1] Stuart E.Madnick, John J.Donovan 著，池田克夫訳「オペレーティング・システム」日本コンピュータ協会，1976

〔2〕 D.W.Barron, "Assemblers and Loaders", MacDonald London and American Elsevier", New York, 1969

〔3〕 IBMシステム／360　オペレーティング・システム　連係編集プログラムおよびローダー日本アイ・ビー・エム株式会社　N：GC28, 1970

〔4〕 Peter Wegner, "Programming Languages, Information Structures, and Machine Organization", McGraw-Hill, 1968

第4章　プログラム・コードの共用

キーワード

再帰（recursion），実行時スタック（run time stack），（レジスタ類の）退避/回復（save/restore），再入可能（reentrant），純手続き（pure procedure），テーブル駆動型プログラム（table driven program），インタープリタ（interpreter），逐次再使用可能（serially reusable），動的ネスティング（dynamic nesting）

目　標

プログラム・コードは作成するのに費用がかかる。またでき上ったコードを実行させるときには高価な主記憶を占有する。したがって無駄なコードはなるべく作らないようにすべきであるし，主記憶にあるコードは利用できる限り多くのジョブで共用すべきである。そのような工夫を凝らしたプログラム・コードは無駄な部分が省かれる結果，理論的にも無駄がなくプログラムの本質を浮き彫りにする。

本章ではこのようなプログラム・コードの実際上の応用技術を理解させるとともに背後にあるプログラムの本質を理解させる。

まず4.1節では単独のプロセス内で行なうプログラム・コードの共用について2種類の概念，再帰（recursion）とテーブル駆動型プログラム（table driven program）を中心に指導する。4.2節ではこれに対し単一プロセスではなく複数プロセス間で共用するプログラムについて理解させる。

内　容

4.1　単独プロセス内のプログラム・コードの共用

共通に使用できるサブルーチンを一つのプログラムのアチコチから呼出して用いるのは極くありふれたプログラム・コード共用化の手順であるが，時にはあるサブルーチンSの中で（Sの実行が終る前に）再びS自身を呼出して作業を行ないたい場合がある。このようにある一つのプログラムの実行が終了する以前に再び同一プログラムが呼出される現象を再帰（recursion）といい，これを可能にしているプログラムを再帰的プログラム（recursive program）といっている。あるプログラムSが再帰的に呼ばれるのはSが直接S自身を呼出す場合だけとは限らない。

図4－1は2重積分を1変数の積分に帰着する例であるが，この場合INTGRTという1変

数に関する積分を行なうサブルーチンが図4－2のように再帰的に呼出されている。このように再帰的呼出しは極く自然にプログラムの設計を行なったとき出現する現象であることを納得させる。

再帰的プログラムにおいてはプログラム・コードの共用はされるが，プログラムの実行に際して必要とされる他の情報は共用されないのでこれを別に維持する必要がある。

たとえば図4－1の再帰的なプログラムを例にとって，

- 再帰的プログラムが用いる入出力パラメタ
- 処理途中で必要とする作業域
- プログラムから別もしくは同一のプログラムを呼出すとき，呼出されたプログラムから制御が戻って来たときの戻り先アドレス（return address）

などの情報の維持方法について説明する。戻り先アドレスはプログラム呼出しのとき（厳密には再帰的に使用されるプログラムから別もしくは同一のプログラムを呼出すとき）に，スタック（stack）を用いて行なうのが一般的である。プログラムの呼出しはネスティングが動的に行なわれるために，動的ネスティング（dynamic nesting）と呼ばれる。

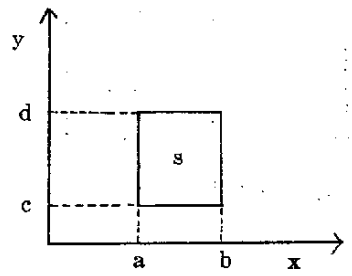
スタックに対してのプッシュ（push），ポップ（pop）操作及びこれを利用してのプログラムの呼出し（call），戻り（return）操作，レジスタ類の退避／回復（save/restore）について説明する。

```
        主ルーチン
          ⋮
    CALL  INTGRT(C, D, H, ANS)
          ⋮
    END

    SUBROUTINE  INTGRT(P, Q, F, V)
          ⋮
    CALL  F(T, VALF)
    END
    SUBROUTINE  H(ARGH, VALH)
    COMMON  Y
          ⋮
    Y=……
          ⋮
```

```
CALL  INTGRT(A, B, G, VALH)
      :
END

SUBROUTINE  G(ARGG, VALG)
COMMON  Y
      :
VALG=Z(ARGG, Y)
      :
END
```



左図に示す２次元の領域S上での関数Z(x, y)の積分

$$\iint_S Z(x, y)\,dxdy$$

をFORTRANで求める例

図４－１

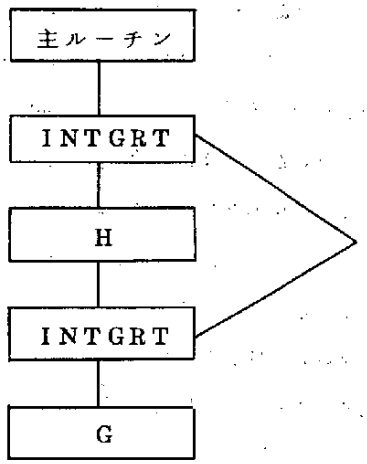


図４－２　プログラム間の呼出し関係図

```
CALL  SUBR1(A1)          A1を読む
CALL  SUBR1(B1)          B1を読む
CALL  SUBR2(A1, B1, C2)  A1+B1→C2
CALL  SUBR3(A1, B1, C3)  A1−B1→C3
CALL  SUBR4(C2)          C2を書く
CALL  SUBR4(C3)          C3を書く
```

(a) サブルーチン呼出しが多用された例

SUBR1	A1		
SUBR1	B1		
SUBR2	A1	B1	C2
SUBR3	A1	B1	C3
SUBR4	C2		
SUBR4	C3		

(b) サブルーチン呼出し命令を省き，操作指令をテーブルの形にまとめた例

図4－3　インタープリタの出現を理由づける例

単独プロセス内のプログラム・コードの共用化の第2の形としてインタープリタ（interpreter）あるいはテーブル駆動型プログラム（table-driven program）について取扱う。

あるプログラムのコードが実質的にサブルーチン呼出しの連続になることがある。たとえば単精度演算しか持たない計算機で倍精度もしくは4倍精度の計算を行なう場合を考えてみる。データの入力に始って，データの転送，4則演算，大小比較，データの出力に至るまでプログラムの殆どの部分はこれらの処理を司るサブルーチン呼出しの列になる（図4－3）。

図4－3(a)のようなプログラムは，図4－3(b)のテーブルの内容を解釈して実行するインタープリタによって効率よく行なえることを説明する。テーブル駆動型プログラムはインタープリタと同様の精神でプログラムの処理をテーブル状の入力によって制御する方式であるが，プログラムの柔軟性を増し，開発の生産性を上げるのに役立つことがあることを教える。

4.2　複数プロセス間のプログラム・コードの共用

プログラム・コードは単に一つのプロセス内で共用するだけでなく，多重プログラミング（multiprogramming）を行なっているときなどプロセス間あるいはジョブ間で共用するこ

とができる。

並行処理が行なわれているプロセス間で同時期にプログラム・コードを共用する問題について次の順番で考えさせ，理解させる。

(i) 共用されるプログラムが自分自身を書き替えているとどのような不都合が生じるか

(ii) 共用されるプログラムが使用する作業域についてはどうか，レジスタについてはどうか

(iii) 上の考察から共用されるプログラム・コードが，複数個のプロセスから同時に使用可能つまり再入可能（reentrant）であるためには，プログラムの実行の際にプログラム・コード自身を変更しないという純手続き（pure procedure）であることが分るが，では実際にコーディング作業を行なう場合にはどのようにする必要があるか

(a) 呼出し側ではどのようなことに気をつけなくてはならないか

(b) 純手続き側ではどのような考慮が必要か

の二つについて考えさせる。また，再入可能コードのコーディング方法については制御プログラム側の制約もしくは方式が密接にからむので，実例について説明する必要がある。再入可能プログラムの作り方の例としてはたとえば参考文献〔1〕を参照するとよい。

複数プロセスによるプログラム・コードの共用の例としてもう一つ，使用時間を互いにオーバラップしないようにして用いる再使用可能なプログラム・コードがある。これはプログラムが走行中は，再入不可能であるが処理が終って制御が一旦プログラムから外に戻ったときに再び他のプロセスのために同じプログラム・コードが使用可能となるものである。実際には複数個のプロセスがこの性質を有したプログラム・コードを同時に使用しようとしたときには，先着のプロセスがこのプログラム・コードを実行する権利を得，他のプロセスは前のプロセスの処理完了を待つかあるいは，後のプロセスのためにプログラム・コードの別コピーが主記憶内にロードされなくてはならない。この二つの制御方法の選択の方法と利害得失とを説明する。

再使用可能プログラムの作り方，守らなければならない規則について説明をする。

指導上の留意点

(1) 再帰性と再入性とはしばしば混乱されることがある。再帰的ではあっても再入的ではない例や逆に再入的ではあっても再帰的でない例を考えさせるなどして二つの差を明確に教える。

(2) インタープリタもしくはテーブル駆動型プログラムが生産性向上のために役立つのは，構造的プログラミングと同じく，低レベルの詳細はインタープリタへの指令を書くプログラマから隠してしまい，人間が一時に考える複雑さをへらす所にあることを納得させる。

(3) 本章では再帰性をもつプログラムではスタックを利用する，再入性をもつプログラムは純手続きに作る等各種の制約があることを教えるが，単に教条的に知識を詰めこむのではなく，そ

れらの約束に違反した場合にどのような不都合が生ずるのかというように，むしろ，反例的に教えることが大切である。これは，プログラマがウッカリ上の約束を守らないプログラム・コードを作ってしまい，それをデバッグするときに役立つ。

(4) 再入的プログラムの作り方についてはオペレーティング・システムとして何を使うかで方式がかなり異る。したがって，可能ならば方式の異る二つのオペレーティング・システムについて説明するのがよい。相違の主な点は，レジスタの退避／回復，作業域の確保の方式などについてである。

また，言語プロセサの能力も関係することがある。

参考文献

[1] IBM, "Operating System/360 Concepts and Facilities, in Programming Systems and Languages", 1967 McGraw-Hill, edited by Saul Rosen, pp 598－646,

[2] Peter Wegner, "Programming Languages, Information Structures and Machine Organization", McGraw-Hill, 1968

[3] C.William Gear, "Computer Organization and Programming", McGraw-Hill, 1968

第5章　並行処理における同期制御と排他制御

キーワード

協同型プロセス（cooperating process），並行型プロセス（concurrent process），多重タスク（multitask），非同期制御（asynchronous control），排他制御（exclusive control），（資源の）共用（share），（イベントの）待ち（wait），（イベントの）通知（post），セマフォ（semaphore），P命令（P－operation），V命令（V－operation）

目　標

単一タスク（single task）を扱っている限りでは現れてこない多重タスク（multitask）特有の問題について理解させる。多重タスクを使いこなすためには若干，システム・プログラムの内部ロジックについて理解することが有効なので極く簡単なオペレーティング・システムについて複数タスクをどのように制御しているかを説明する。

多重タスクを可能にする基本命令とそれを可能にするハードウェアのメカニズムについても理解させる。

内　容

5.1　多重タスク

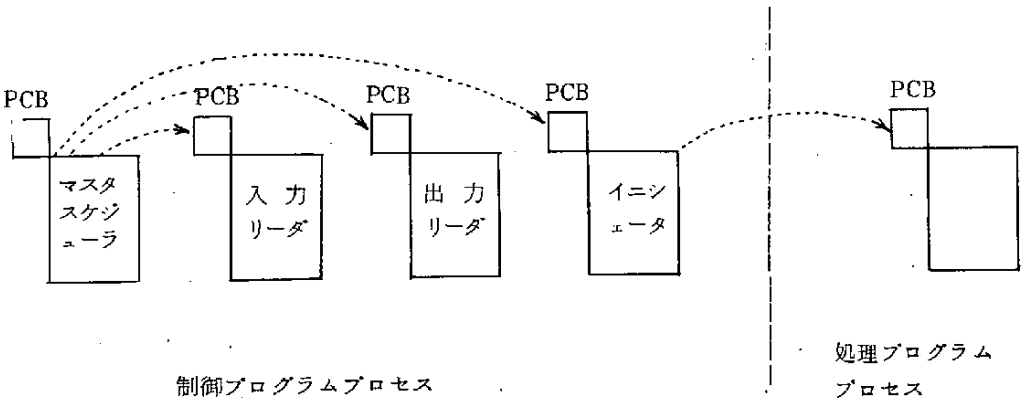
プロセス（process）またはタスク（task）とは他の処理と並列的に実行可能な処理をいう。この意味で並行型プロセス（concurrent process）ともいう。

プロセスはプログラムを実行することによってなされる処理もしくは仕事を指し，単なる命令の順序（sequence）であるプログラムとは異ることを理解させる。プログラムは一種のデータであるが，プロセスは違う。

コンピュータに投入されるジョブは，ジョブ・ステップ（job step）単位にプロセスとして実行される。また一つのジョブ・ステップの中で複数個のプロセスが生成されることもある。したがって，コンピュータ内部には一般に複数のプロセスが並行型プロセスとして存在する。プロセスはこのようにユーザのプログラムの実行のため生成されるだけでなくオペレーティング・システムそれ自身の各種のサービスを遂行するためにも利用されていることを説明する。たとえば入力ジョブ・ストリーム（input job strerm）を読取る入力リーダ（reader）と称するプ

ロセス，あるいは各種の出力情報を印字する出力ライタ（ output writer ）についてその並行型プロセスとしての動作を説明する。（図５－１）

並行型プロセスあるいは多重タスク（ multitask ）によって得られる利点につき説明する。個々のプログラムの設計が簡単化される点，一つのプロセスが入出力動作を行うとき生じる待ち時間を他のプロセスが無駄なく利用できる点などについて説明する。



破線はプロセスの生成を意味する。

ＰＣＢ：プロセス・コントロール・ブロック（ Process Control Block ）

図５－１　制御プログラムのプロセスと処理プログラム（ processing program ）のプロセスとの関係

5.2　資源の共用および割当て

互いに競合するプロセスによって多くのシステム内の資源（ resource）が要求される。制御プログラムはこれらの要求を矛盾のないように各プロセスに割当てプロセスの終了時には再び返却させる。

資源のあるものは要求時にすぐ割当てることができるが，現在他のプロセスがそれを使用している場合にはそのプロセスが資源を返却するかあるいは処理を終了するまで，資源要求プロセスを待たせておかなければならない。

同じ資源に対して数個のプロセスがその使用を待って（ wait ）いる場合，列（ queue）が生じる。列はシステムが管理している資源ごとに作られ，かつ列へのプロセスの登録には，一般に，優先順位（ priority）がつけられることを説明する。

中央処理装置（ central processing unit ）の使用を要求するプロセスの列はＰＣＢ（process control block）という制御表から成り，この資源割当を司る管理モジュールを

ディスパッチャ（dispatcher）と呼ぶ。ＰＣＢは第３章で述べた活動記録（activation record）の一種であり，一度待機状態にされたプロセスを再度昔の中断点から再開始させるために必要な情報を蓄えている。

プロセスもしくはタスクは一度発生させられた後，消滅する直前まで以下の三つの状態の一つにあることを説明し，かつその間の状態変化がどのような原因によって生じるかを考えさせる（図５－２）

・実行中（run）

・実行待ち（ready）　　ＣＰＵ時間以外の必要なすべての資源が割当てられ，いつでも実行可能な状態となっている。

・待機（wait）　　プロセスが何等かの条件によって処理を続行できなくなったとき，この状態にさせられる。ＷＡＩＴ状態になるのはＷＡＩＴなどというシステム・マクロ命令をプロセスが陽に実行した場合と，プログラム上では何の宣言がなくても，システム・マクロ命令を実行した場合に陰に待機状態にさせられるケースと２種類あることを説明する。

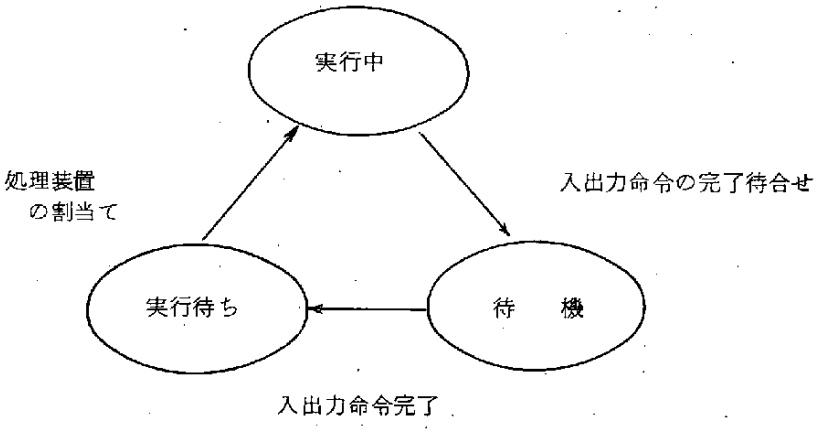


図５－２　　プロセスの三つの状態

5.3　事象の同期

(1)　待機（wait）および通知（post）

事象の同期（event synchronization）とはある特定の事象が生起するまでプロセスの実行を遅らせることである。

同期には二つの側面がある。

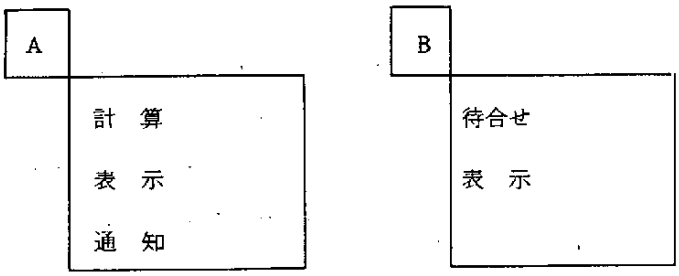
・同期の必要性をＷＡＩＴマクロ命令によって陽にあるいは他のシステム・マクロ命令によって陰に要求する。

・事象が生じたあと，その事象の生起を要求するプロセスにその旨を通知しＷＡＩＴを発信

した点を通過できるようにする。

後者の通知はたとえばＰＯＳＴマクロ命令でなされる。事象の生起が制御プログラムによって知られる場合（たとえば読込み動作の完了など）には制御プログラムが通知（ post ）する。

この待機と通知の機能は二つのプロセスが非同期に処理されているときに，たとえば一方のプロセスのある処理を他方のプロセスのある処理より先行させるという形で同期を互いに取るような場合に利用できる（図５－３）。このときプロセス間で共通のデータを参照することができると複数個のプロセスを一つの仕事に協同型プロセス（ cooperative process ）として利用することができる。



プロセスＡがプロセスＢの処理に先行する。

図５－３　　二つのプロセスで同期をとる

プロセスは数個の事象の生起を待合せることができる。たとえばＩＢＭ社のＯＳ／３６０について待機と通知のメカニズムについて説明する。

(2)　エンキュー（ＥＮＱ）及びデキュー（ＤＥＱ）

待機と通知機能を用いる事象の同期方法は，資源の共用の管理に利用できる。たとえばプロセスがシステムの資源を要求したときそのプロセス（あるいはタスク）のＰＣＢを適当な資源列につなぎ，そのプロセスを資源が利用可能となるまで待機させる。資源が利用可能となったときそのプロセスに制御プログラムが通知する。

逐次再使用可能な（ serially reusable ）資源の利用に関しては事象の同期をエンキュー（ＥＮＱ），デキュー（ＤＥＱ）によって行うことが可能である。プロセスは資源の要求をＥＮＱマクロによって行い，資源解放をＤＥＱマクロによって行う。ＥＮＱマクロが発信されたとき要求した資源が使用中であるならば，マクロを発信したプロセスは待機状態に置かれる。つまり待機列に入るわけである。プロセスが資源の使用を終りＤＥＱマクロを発信したとき，その資源の使用を待っている最初のプロセスに資源が使用できる状態になった旨通知される。

ＥＮＱマクロ命令により，並行的に動作しているプロセスが共通データを利用する際に，た

がいの干渉を避けることができる。ＥＮＱで対象とする資源はいかなるものでもよく，プログラミング上はそれが互いのプロセス間で共通の識別方法で識別さえできればよいことを説明する。

ＥＮＱ，ＤＥＱを用いることにより，資源の利用に関して排他制御（ exclusive control）ができることを説明する。排他制御は野放図に行うとデッドロック（ deadlock）の危険性があることを注意する。

同期の機構についてはほかにＬＯＣＫ，ＵＮＬＯＣＫを用いる方法，セマフォ（semaphore）に対するＰ命令（Ｐ－operation），Ｖ命令（Ｖ－operation）を用いる方法があることを説明する。

(3) 処理装置が複数個ある場合の同期処理について

普通，一つのプロセスが資源を要求してそれが使用中であったような場合には，待機状態におかれて（つまり処理は中断されて）処理装置は他のプロセスの処理を行う。同様な同期機構はオペレーティング・システム自体の内部でも用いられる。ところで"実行待ち"プロセスの管理は実行待ちプロセスの列の管理によってなされるが，実行待ちプロセスの中からプロセスを選びそれに処理装置を割当てるためにはこの実行待ちプロセス列を更新しなければならない。そのため，これに対して排他的制御を行うため，ロック等をかけなければならない。このようなとき，他のプロセスが再び実行待ちプロセス列にロックをかけようとしたときには，実行待ちプロセス列自身がロックされているので，他のプロセスを実行させることができない。したがって，プロセスはロックが解かれるまで絶えずロックを判定し続けなければならない。判定後再び判定命令にグルグル制御をまわし続けるので一名"スピン（ spin ）"ロック（あるいはbusy wait）ともいわれ，多重処理装置のオペレーティング・システムでよく用いられる。つまりロックには中断のロックとスピンのロックがある。

スピン・ロックを実現するときには，基本的には共用される資源ごとにその資源の状態を表わすための物理的な情報（ロック・バイトあるいはセマフォ）が必要である。ロック・バイト＝０は資源が使用可能であること，ロック・バイト＝１は資源がすでに使用中であることを示すこととする。このような約束の下ではプロセスは共用資源の処理を行う前に次の動作を行わなければならない。

(i) ロック・バイトを検査する（０か１か）

(ii) ロック・バイトを１にする

(iii) もしロック・バイトが１であったなら，手順(1)にもどる

プロセスが資源の利用を完了したときにはロック・バイトを０に設定する。注意すべきことは上記(i)，(ii)の手順の間で，ロック・バイトが他のプロセスによって変更されてはならないと

いうことである。たとえ処理装置が複数台存在したとしても(i)(ii)の手順は一動作で行わなければならない。ＩＢＭ370 でこれを行う命令はＴＳ（Test and Set ）で，他のハードウェアにおいても同様の命令が必ず存在することを説明する。

プログラムの実行中にこのように同時には複数個のプロセスが同一資源の使用ができない，いわゆる際どい部分（critical section）に入ることを説明し，その実現方法としてのＰＯＳＴ／ＷＡＩＴ，Ｐ／Ｖ命令について説明する。

指導上の留意点

(1) ＤＥＱ，ＬＯＣＫ／ＵＮＬＯＣＫ，Ｐ命令／Ｖ命令，ＷＡＩＴ／ＰＯＳＴ，ＥＮＱ／ＤＥＱ，などの微妙な差について説明する。

(2) 中断のロック（wait ）と"スピン"のロック（busy wait ）との差を明確にする。

参　考

(1) オペレーティング・システムに対して入力命令（たとえばRead操作）を発信したときのプロセス（タスク）の待合せの方法，それに対する通知の仕方については文献〔２〕P.641以下の例による説明などが適切であろう。

(2) Ｐ／Ｖ命令については文献〔１〕P. 293，P. 454に詳しい。

参考文献

〔1〕 Stuart E. Madnick， John J. Donovan 著，池田克夫訳，「オペレーティング・システム」日本コンピュータ協会　1976

〔2〕 IBM,"IBM Operating System／360 Concepts and Facilities", in Programming Systems and Languages edited by Saul Rosen, McGraw-Hill, 1967

〔3〕 Per Brinch Hansen,"Operating System Principles", Prentice-Hall, 1973
（田中穂積他訳「オペレーティング・システムの原理」近代科学社）

〔4〕 Leon Presser,"Multiprogramming Coordination, Computing Surveys", Vol,7,NO. 1,pp. 21−44, 1975

〔5〕 Leslie Lamport,"Concurrent Reading and Writing", CACM, vol. 20 №11 pp. 806−811, 1977

第6章　プログラムの文書化

キーワード

問題仕様書(problem specification)，設計仕様書(design specification)，コーディング仕様書(coding specification)，試験仕様書(testing specification)，利用者マニュアル(user's manual)，流れ図(flowchart)，決定表(decision table)，デシジョン・グリッド・チャート(decision grid chart)，状態遷移図(state diagram chart)，ＨＩＰＯチャート(HIPO chart)，構造化流れ図(structured flowchart)，自動流れ図作成(automatic flowcharting)，非手続き型言語(nonprocedural language)，疑似コード(pseudo code)

目　　標

文書化は，信頼性の高いプログラムを作成するための必要条件であることを理解させる。また，プログラム開発の各工程に於ける文書化の目的と内容を十分に把握し，開発の目的に合った文書化が自由にできるようになることが望ましい。ここでは，標準的な文書の形式と，それらの文書が満たさなければならない標準的な項目を説明する。

文書化は，プログラム開発の各段階で行なわれ，レビューされる。これらの文書は次の段階へと渡される，唯一のコミュニケーションのための手段である。また，正確で明瞭に書かれた文書は，作成されたプログラムを表現するための，ただ一つの方法であり，開発されたプログラムと，そのプログラムの利用者との間をつなぐ唯一のものである。プログラムやプログラムの開発と文書とのこれらの関係を理解することが大切である。

さらに，プログラムの内容を表現するための文書化の手法にはいろいろあり，プログラムの持つ個性に合わせて，それらの中から一つを選択するか，幾つかを組合せて使うことが必要である。ここでは，文書化の代表的な手法を幾つか選び，その特徴と適用の範囲などを説明し，プログラム開発の場合に応じて，最適な手法が選択できるようにする。

文書の自動作成や文書からのプログラムの生成については，それらが効果的である理由を説明し，かりにそれらの支援の道具が用意されていなくても，プログラムの表現がどうあるべきかが理解できるようになればよい。

内　容

6.1　文書化の重要性

開発の各工程をつなぐものは文書である。また開発したプログラムと，そのプログラムの利用者との間をつなぐものも文書である。文書化は，プログラム開発にあたっての重要な作業であるが，とかく軽視されがちである。

ここでは，表現のあいまいな文書や誤った内容が書かれている文書，簡潔に書かれていない読みにくい文書などが，プログラム開発とプログラム利用の上で，いかに悪い影響を及ぼし，悪い結果をまねくかを，例を挙げて説明し，文書化の重要性を十分に理解させる。

6.2　文書の標準化

文書はコミュニケーションの手段である。そのため，書かれた文書の解釈上の誤りや，形式の違いによる読みにくさ，用紙の不統一によるあいまいさ，などを無くすために，標準化が必要である。

ここでは，①形式の標準化，と②用字，用語の標準化，を取上げて，標準化が必要であることを説明する。

形式の標準化では，個々の文書の記述内容（項目）の統一のほか，次のような一般的な部分の標準も決めておかなければならない。

- 文書用の用紙の統一
- 章，節，項の割り方
- 表番，図番の入れ方
- ページのふり方や目次の作り方

用字，用語の標準化，すなわち表記法の統一にあたっては，とくにソフトウェアの文書には多い外国語の扱い方に注意しなければならない。正確さを望むとしても，外国語をそのまま記した，外国語まじり文は極めて読みにくいものであるから，外国語の発音をそのままカナ書きにする位までは考慮する。なお，標準となる辞書を決めておくのも，良い方法である。

6.3　標準的な文書

プログラムの開発計画を立てる際の重要な要因の一つは，文書計画である。プログラム開発の各段階で作られる，文書の種類と形式，記述内容（項目），およびその管理と保守の方法は，プログラム開発の初期の段階で決めなければならない。

この節では，標準的な文書の種類と形式，記述内容などを説明し，次の節で，文書の管理と保

守の方法について説明する。

プログラム開発の各段階に対応して，多くの文書が作られる。このそれぞれの文書の目的と，それらがどの段階で，どのように使われるかを説明する。

次に挙げるのは一つの例である。それぞれの開発によって，作られる文書の種類や分け方も違うが，ここでは一般的な場合として，次のように想定しておく。

- 問題仕様書（problem specification）
- 設計仕様書（design specification）
- コーディング仕様書（coding specification）
- 試験仕様書（testing specification）
- 利用者マニュアル（user's manual）

(1) 問題仕様書（problem specification）

利用者の要求している問題を分析し，正確に仕様として表現したものである。この仕様書には，要求される機能や性能が記述され，この仕様書に示された内容が，これから開発するプログラムのすべてを決める時の基となる。開発の初めの段階で作られ，開発の全工程を通しての基準となる仕様書であるから，利用者と十分に討議し，レビューをしておかなければならない。

(2) 設計仕様書（design specification）

プログラムの構造を決める仕様書であり，問題仕様書に示された問題の解決法を決め，プログラム設計の基本方針を示す。

設計仕様書には，次の内容を記述する。

- 全体的な設計概念
- プログラムの構成要素
- モジュールの階層構造
- モジュール間のインタフェース
- データの構造
- 各モジュールの設計
- ファイルの設計
- データの流れ

(3) コーディング仕様書（coding specification）

コーディング仕様書は，一つのモジュールの完全な記述である。

この仕様書には，次の内容を詳細に記述する。

- モジュールの機能
- 他モジュールとのインタフェース

・モジュールの論理構造

・データの記述

(4) 試験仕様書(testing specification)

試験には，単体試験，結合試験，総合試験，受入れ試験など，幾つかの段階がある。試験仕様書は，これらのそれぞれの段階について作らなければならない。

試験仕様書には，次の内容を記述する。

・試験の目的

・試験対象と試験の範囲

・試験方法，または試験手順

・試験のために必要とする環境

・試験結果の判定方法と，判定基準

(5) 利用者マニュアル(user's manual)

利用者マニュアルは，そのプログラムを利用者に使ってもらうための手引書である。プログラムの利用者は，ソフトウェアについての知識をもつ人達だけとは限らない。したがって，なるべく専門的な用語を避けるか，用語の解説を付けて，誰にでもわかりやすく，簡潔に，利用者の立場に立って書かれていなければならない。

利用者マニュアルには，次の内容を記す。

・プログラムの機能の概要

・使用手順

使用環境の整備方法

データの形式

プログラムの使い方(制御文の形式など)

異常状態となった時の処置

・メッセージの一覧と，その説明や対応の方法

・適切な目次と索引

6.4 文書の管理と保守

文書化を重要なこととして捕えることができるならば，作成された文書の管理と保守も同じように大切なこととして理解するのは，難かしくない。ここでは，文書の管理方法や保守方法の具体例を示して説明するとよい。

文書の管理，保守は，プログラム開発の初期の段階から必要なことである。文書は，開発時にコミュニケーションの役割を持つものであるから，とくに多くの人達が開発に参加する時には，

文書の管理や保守に欠陥があると，開発されたプログラムの欠陥として，そのまま現われる。したがって，大きなソフトウェアの開発では，文書の管理，保守などのために，専任のライブラリ管理人（librarian）を設けている。

プログラムの開発が終った後の文書の管理と保守は，開発したプログラムの管理や保守と同じように，また重要なことである。

プログラムの保守のためには，文書は欠かせない。開発されたプログラムは，そのプログラムの内容が文書化され，その文書が適切に管理，保守されることによって，永く使用に耐えられるものとなる。

プログラムの仕様の変更や改良にあたっては，文書を参照して変更の方式や変更内容を決め，プログラムを作り替える。そして，同時に文書も変更しておく。この時，修正の目的，内容，その修正に伴なう他への影響などを記述した文書（変更仕様書）を作成しておくことが望ましい。

文書の管理は各個人が行なうのではなく，統一して管理，保守することが重要である。とくに，文書間の記述上の矛盾がないよう，文書の修正は，プログラムの修正と同じように重要であることを理解しなければならない。

文書の管理，保守のためには，次のことを行なう。

- 文書には整理番号を付けて整理し，必要に応じて取出せ，コピーできるようにしておく。
- 需要の多い文書は，あらかじめ何部かコピーを取っておく。
- 全文書の目録を番号対応に作っておく。
- 定期的に，最新の文書の一覧表を関係者に配布する。一覧表には，作成者，作成・変更日，要約，登録番号などを記しておく。
- 火災やその他の災害が起った時のために，定期的に他の場所にコピーを保存する。

6.5 プログラムの完全な記述

プログラムの正確な設計と，作成されたプログラムの正しさの証明のためには，利用者の要求やプログラムの完全な，そして形式化された記述が必要である。この要求に従って，次の二つのことが試みられている。

① 利用者の要求を記述し，要求内容をプログラムの設計に結びつけるための言語などの道具と，そのサポートの開発。

② モジュールの完全な記述を行なうための言語と，その言語で記述された内容を使ってプログラムの正しさを証明すること。

この節は，文書化のもう一つの方向として理解できればよい。

①の例は，参考文献〔1〕,〔2〕に，②の例は，参考文献〔3〕,〔4〕にあるので，参考にす

るとよい。

6.6 流　れ　図

流れ図は，プログラムの制御の流れを表現するための，もっとも一般的な手法である。流れ図については，すでに学習しているので，ＪＩＳ規格を説明する程度でよい。ここでは他の文書化の手法との比較の基準とするために，流れ図の特徴を説明する。

流れ図の長所と短所は，次の通りである。

- どのような制御の流れでも表現できる。
- 1940年代から使われている手法なので，広く知られているし，標準化もされている。
- 高級言語での制御の流れを表現するためには，流れ図の要素の方が言語の要素よりも詳しすぎ，適さないことがある。（例：ＤＯ文など）
- 流れ図は，制御の１次元的な流れの表現を主としているので，制御の階層的な構造を表現するのには向かない。

流れ図の一つの形として，Ｄチャート（参考文献〔7〕）にも触れておくとよい。（図６－１）

また，プログラミングに詳細流れ図が必要であるか否かの議論があること（参考文献〔8〕）などを通して，詳細流れ図の役割を掴む指導もする。

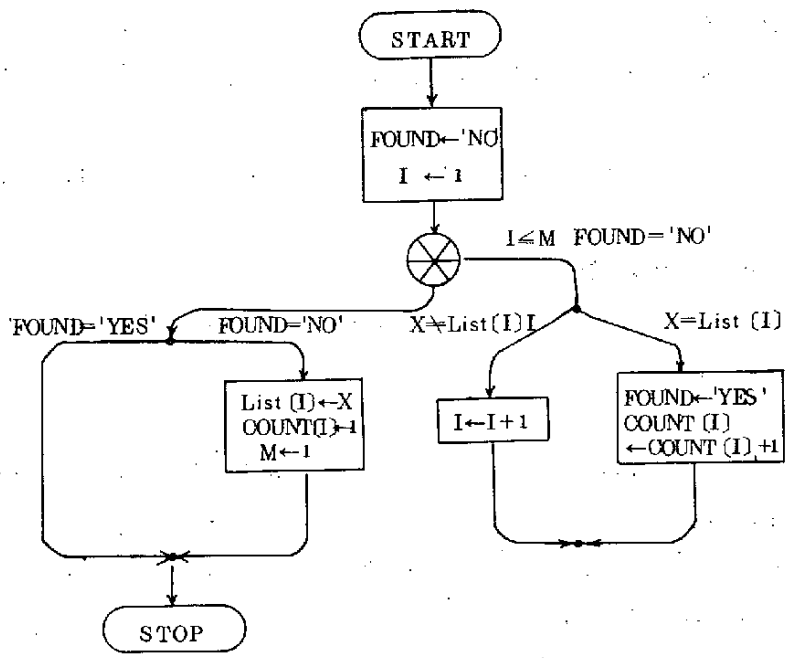


図６－１　　Ｄチャートの例（参考文献〔1〕より）

6.7 決　定　表

決定表(decision table)は，問題の論理を記述し，その解を表現する記法である。とくに，条件が複雑にからみ合った問題を整理し，各場合に対応する処理を完全に網羅した形で表現できることに特徴がある。ここでは，例題によって実際に決定表を作って説明し，決定表が自由に作れ，作った決定表の検定ができるようにする。

決定表は，次の特徴をもつ。

- 論理を，あいまいさ無しに表現することができる。
- 条件の組合せに落ちがない。
- コンパクトに表現できる。
- 完全性の検定や，条件の組合せ方の最適化ができる。
- 論理を整理するための，プログラム設計の補助手段として使える。
- 決定表からプログラムコードを生成することも可能である。

決定表は，次の四つの部分から成る。

- 条件スタブ(decision stub)
- 条件エントリ(decision entry)
- 動作スタブ(action stub)
- 動作エントリ(action entry)

これらの部分の記述方法によって，次の三つの表わし方がある。このそれぞれの特徴を説明する。

- 制限エントリ型
- 拡張エントリ型
- 混合エントリ型

複雑な論理を表現するためには，場合の数が多すぎて扱いにくくなることもある。このような場合には，幾つかの決定表に分けて，それらをGOTOやCALLで連結する。この分解や連結の方法，ループの表現方法などの説明もする。

決定表については，参考文献〔9〕を参照するとよい。

6.8 デシジョン・グリッド・チャート

決定表と関連深い文書化の手法として，デシジョン・グリッド・チャート(Decision Grid Chart)の説明をする。決定表が決定条件を主体に，ある条件が決った時に行なう動作(action)を表にしたものであるのに対し，デシジョン・グリッド・チャートは，動作を主体に，その動作を行なう時の条件を書いた表である(図6－2)。デシジョン・グリッド・チャートは，決定表に変換することが可能である。

複雑な，複数個の決定表をGOTOやCALLによって連結しなければならない場合でも，デシジョン・グリッド・チャートを使えば容易に表現できるので，そのような場合に，決定表を作成するための補助手段としても使える。また，デシジョン・グリッド・チャートは，動作を主体に表現するので，論理の意味を掴むには決定表よりもわかり易い場合が多い。したがって，論理の文書化としても有効な手法である。

デシジョン・グリッド・チャートも決定表と同じように，その記述内容によって，次の三つの型がある。

- 制限エントリ型
- 拡張エントリ型
- 混合エントリ型

ここでは，デシジョン・グリッド・チャートが作れるようになること，およびデシジョン・グリッド・チャートから決定表への変換についても説明する（参考文献〔10〕，〔11〕）。

grid chart A		SUM = SUM + 1	PERFORM SUB1	H = X ** 2	K = Y − Z	PERFORM SUB2	K = Y + Z	E = E − 100	GOTO GRID CHART B
		a1	a2	a3	a4	a5	a6	a7	a8
C1	SWITCH =	1	1	2	2	3	3		
C2	A = B	Y		Y	N				Y N
C3	D > 10	N			Y		Y		
C4	E < 100		Y			N		N	
C5	F < G	N		Y					

図 6 − 2　　デシジョン・グリッド・チャートの例（参考文献〔4〕より）

6.9 状態遷移図と状態遷移表

処理の開始と終了の状態を含めて，幾つかの取り得る状態を定義し，事象（イベント）の発生

によって，制御がそれらの状態間を移行する様子を図にしたものが状態遷移図であり，表形式にまとめたものが状態遷移表である。

状態遷移図は，外部からの事象が発生した時，それまでの処理手順に関係して次に行なう処理内容が決るような場合の表現方法として適している。たとえば，通信処理でのプロトコルの処理手順を表現する場合などによく使われるが，適用の範囲は広い。状態遷移表では，取り得るすべての状態について，すべての事象発生の場合の処理内容を記述するので，設計時の落ちがなくなり，設計の補助手段としても使われる。

ここでは，状態遷移図と状態遷移表の説明を例を挙げて行ない，流れ図による表現と比較して，場合による有用性を示す。

例題は，簡単な内容の方が理解しやすい。たとえば，３種類のカード入力――部名カード，課名カード，社員カードなどをイベントとし，入力カードの順序チェックや表題，課の合計，部の合計などを出力するプログラムなどが適当であろう。

6.10 ＨＩＰＯチャート

ＨＩＰＯチャート(Hierarchy plus Input-Process - Output Chart)は，設計のための文書化と，保守のための文書化の両方を考えて作られている。また，設計の補助手段ともなっている。ＨＩＰＯチャートの書き方などについては，参考文献〔12〕を参照するとよい。ＨＩＰＯチャート用の記述用紙やテンプレートなども作られている。

ＨＩＰＯチャートは，次の利点をもつ。

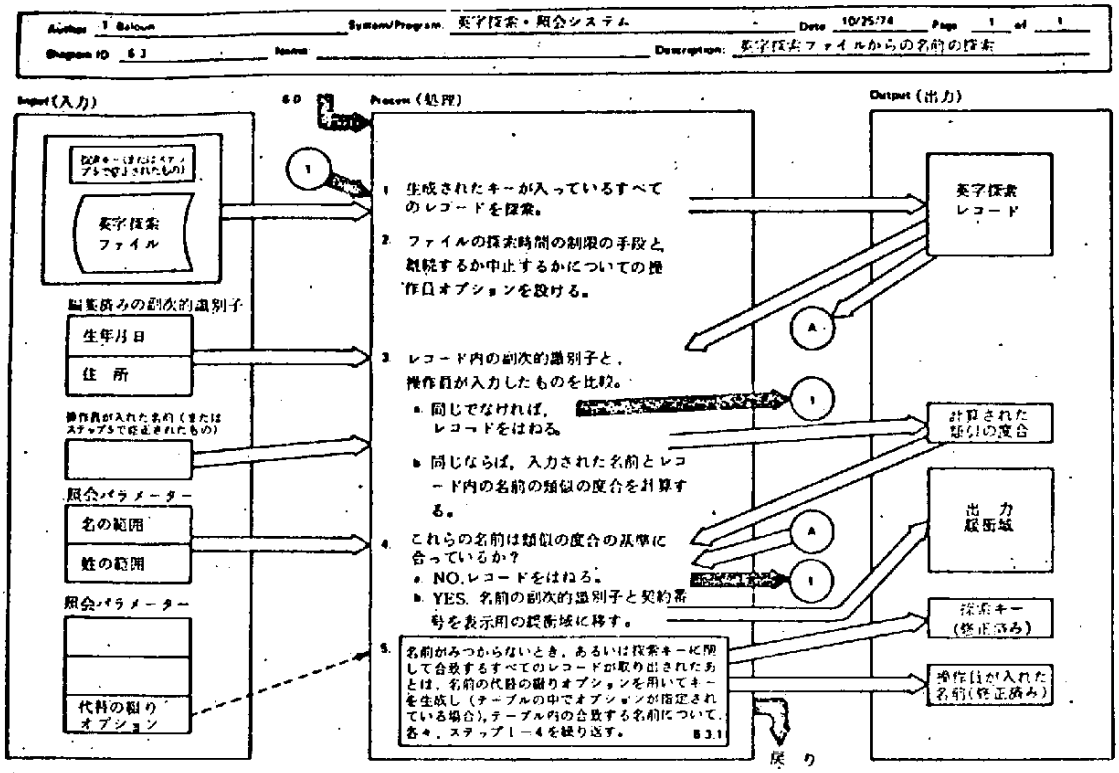
- システムの機能を階層的に表現するので，機能の構造がわかりやすい。
- 入力，処理，出力の関連が明らかに表現できる。したがって，制御の流れの中に，データの流れも同時に示すことができる。
- プログラム全体の中に個々の機能を位置づけるので，機能の理解や機能の変更が容易である。

ＨＩＰＯチャートは，次の三つの部分で構成する。(図６－3,4)

- 内容の一覧(目録)
- 総括ダイアグラム(機能の表現を中心とした図)
- 詳細ダイアグラム(コーディングができるレベルにまで詳細化した図)

ＨＩＰＯチャートは，下降型(top-down)プログラム設計によって作成する。

ここでは，適当な例題を選び，実際に簡単なプログラム設計を進めながらＨＩＰＯチャートを作っていく説明のとり方をするとよい。



Extended Description (補足説明)

Notes (注)	ルーチン	ラベル	Ref. (参照)
2 ファイルの探索時間を制限するために，CICSタイム・オプションを使用しなければならない。			
3b. 操作員が入力した名前が，英字探索ファイル上の名前と1文字ずつ比較される。そのあと，2つの名前のうち長い方の文字数を，等しかった文字の桁数で割ることによって，類似の度合を計算する。			
4. 計算された数が照会パラメーターの中の数より小さい場合，その名前は受け入れられる。			
5 このステップの開発の前に代料の掛りを含むテーブルが作られる。			

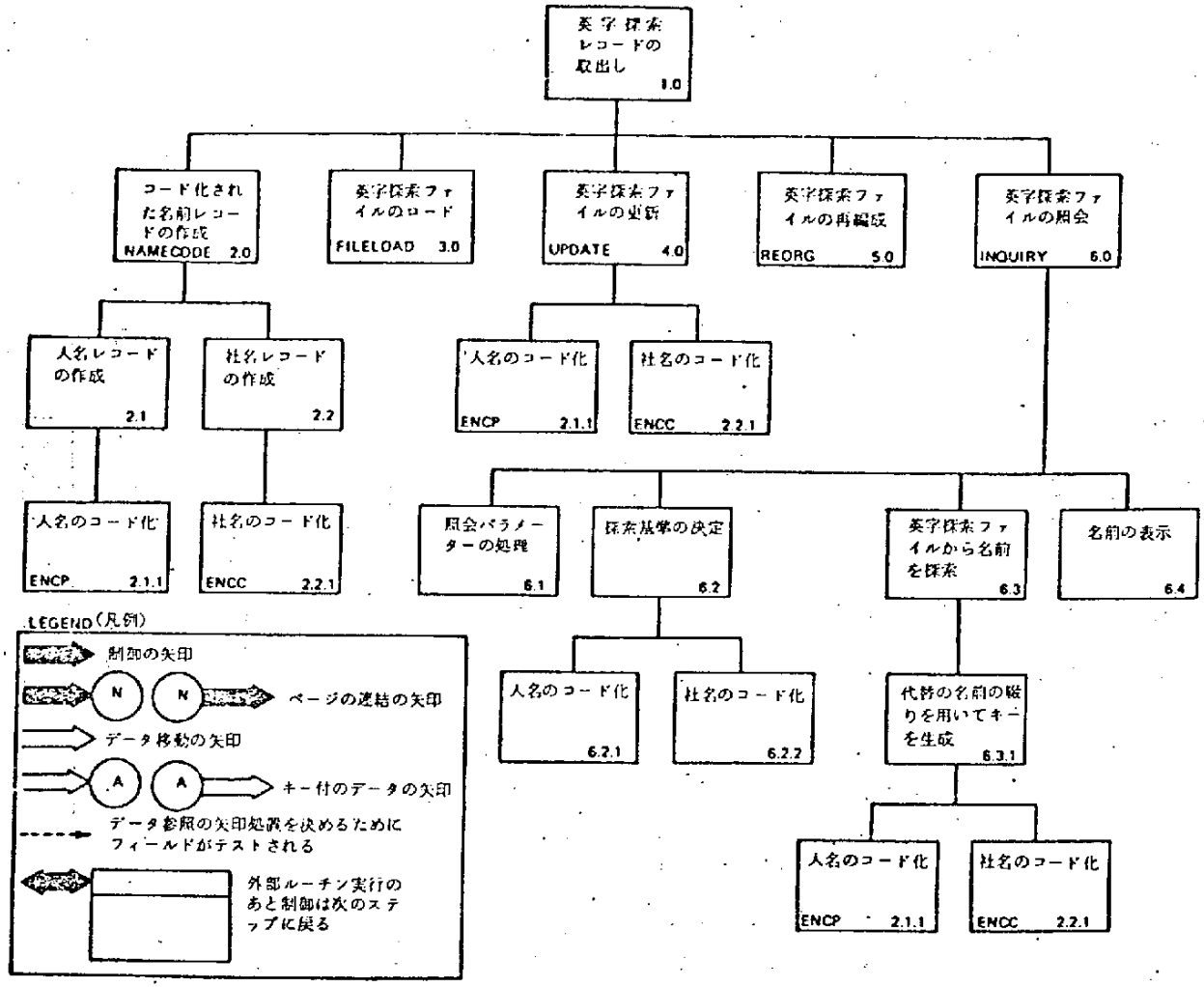
Extended Description (補足説明)

Notes (注)	ルーチン	ラベル	Ref. (参照)

英字探索ファイルの名前の探索，詳細ダイヤグラム（6.3）

補足説明

図6－4　HIPO チャート詳細ダイアグラムの例（参考文献〔7〕より）



英字探索・照会システムに対する初期設計レベルの図式目次

図6－3　HIPO チャート目録の例

（参考文献〔6〕より）

6.11 構造化流れ図

構造化流れ図（structured flowchart）は，構造的プログラミングのための流れ図として考えられたものである。したがって，構造的プログラミングの三つの要素――連結構造（sequence structure），選択構造（choice structure），繰返し構造（iteration structure）――と，それらの構造化を，モジュール全体の構造の中に位置付けて表現できるようにした図である（図６－５）。

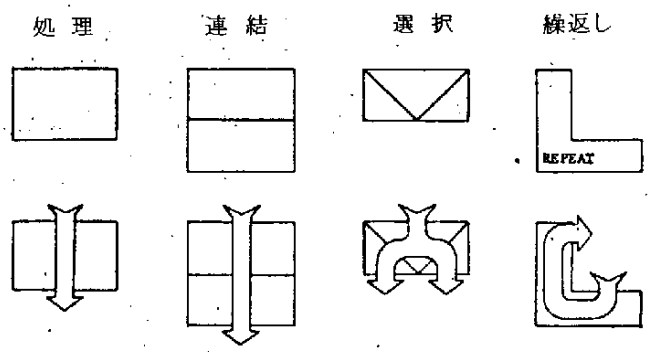
構造化流れ図を使うと，モジュールの設計は，下降型（top-down）で段階的に行なえる。また，このようにして作った構造化流れ図をもとにコード化すれば，きれいな形に構造化されたコードを作ることができる。

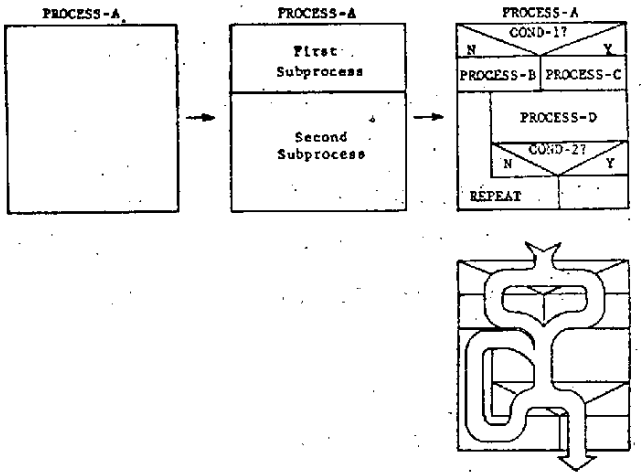
構造化流れ図は，次の特徴をもつ。

- 構造的プログラミングの設計思想に合わせて作成される図である。
- モジュール全体の構造が把握しやすい。
- ページを渡る結合子がないので，一覧性に優れる。
- コードとの対応付けが明らかである。
- 図の修正が行ないにくい。
- 注釈文が書きにくい。
- 複雑な構造をもつモジュールでは，図が大きくなり大きな用紙を必要とする。

構造化流れ図は，構造的プログラミングのための表現手段として優れた面をもつが，上に記したような欠点も持つため，まだ一般化していない。ここでは，適当なプログラムコードを例に，それを構造化流れ図で表わすとどうなるかを示すだけでよい。

構造化流れ図については，参考文献〔13〕，〔14〕を参照するとよい。





DECOMPOSE ROOT PROCESS

(Initially, independent and subordinate lists are empty.)

Define root process and put it on the "independent" list.

Select a process from independent list; put it on the "subordinate" list and remove it from the independent list.

Select a process from subordinate list; call it "current"; remove it from subordinate list.

Does current process require decomposition?
Y
N

DECOMPOSE CURRENT PROCESS ONE LEVEL

Add newly defined subordinate and independent processes (if any), to the appropriate lists.

More processes on subordinate list?
Y
N

REPEAT

Wrap up the independent process just decomposed.

More processes on independent list?
Y
N

REPEAT

Wrap up the decomposition.

図６－５　構造化流れ図の構成要素と完成図の例（参考文献[8]より）

6.12 プログラムからの文書の自動作成

この節では，プログラムから文書を自動的に作ることの意義と，具体例を説明する。

プログラムを文書から自動的に作成することは，次の利点をもつ。

- プログラム・コードと一致した正確な文書が得られ，保守やデバッグに役立つ。
- プログラムを修正すれば，文書も自動的に修正され，プログラムと文書の不一致が防げる。

流れ図の自動作成プログラムは各種作られ，実用化されている。また，構造的プログラミング用言語のための構造化流れ図（structured flowchart）を作成するプログラム（参考文献〔15〕）なども試みられている。

流れ図自動作成プログラムには，

① 自動作成のための指示部分をコーディングしておくもの

② とくに何の指示がなくても流れ図を作り出せるもの

の二つの型があるが，①はコーディング時に流れ図を多少意識してコード化するので，②よりも一般に見やすい図ができる。しかし，いずれにせよ流れ図上に表わされる文は，変数名や注釈文から拾われるので，注釈文の入れ方や量などを上手に与えないと，見やすい流れ図は作れない。

6.13 文書からのプログラムの生成

この節では，文書からプログラムを生成する意義と，文書とプログラムのつながりについて説明する。

文書からプログラムを生成することは，（文書より一般に下位レベルにある）プログラムから文書を作ることが，プログラムのデバッグや保守を主にしているのに比べ，次の利点をもつ。

- 入力となる文書には，「何を」行なうかを書き，「如何に」行なうかを与えることは，少ないか，または全く必要としない。
- したがって，計算のアルゴリズムを考える必要がないので，コード化作業を省くだけでなく，コード化時の誤りも生じない。
- プログラマでない人達でも，プログラムを作成することができる。

文書からプログラムを作成するシステムには，

- 非手続き型言語（nonprocedural language）
- 決定表コンパイラなど

がある。

非手続き型言語は，事務処理の分野に多く，問題に合わせて，入力データの形式やファイルの定義，印刷出力の形式などを与え，また必要な処理を実行の順序とは関係なく，条件を付けて記述することによってプログラムを生成することができるシステムである。代表的な例として，

RPG(Report Program Generator)が，またファイルの操作を主体としたMARKN(参考文献〔16〕)やその他のシステム(参考文献〔17〕，〔18〕)がある。

決定表からプログラムを生成する決定表コンパイラは，複雑な判定を含んだプログラムを作成する場合に役立つだけではない。一般の事務処理用プログラムでも，論理を決定表で表現して，決定表コンパイラでプログラムを生成すれば，プログラミングの作業を省き，コンパイラの最適化機能による最適化も行なわれ，良いプログラムができる。

プログラム設計言語(program design language)などと名付けられている，コンパイラは持たないが記述性に優れている言語で，プログラムをコード化するのも良い方法である。このようなコードを疑似コード(pseudo code)と呼び，疑似コードでコード化したのちに他の言語に置替える。疑似コードで書いたプログラムは，そのままプログラムの論理を表わす文書となって役立つ。

COBOL言語など，日常語に近い言語を使ってプログラミングできる言語では，プログラム・コードが文書の役割をはたし，詳細な文書を別に作る必要がない。また，構造的プログラミングは，人にとってわかりやすいプログラム構造とすることを目的に提案された手法であり，適切な注釈文を加えてプログラミングすることによって，文書としての役割もはたすことができる。

このように，文書とプログラムのつながりを良くすることは，文書からプログラムを自動的に生成できなくても，次の利点がある。

- コード化やテストが容易になる。
- 保守性が良い。
- 人とプログラムのつながりが良くなるので，信頼性の高いプログラムとなる。

指導上の留意点

(1) 全体を通して関連性をよくするためには，プログラム開発の例題を作り，その例題に基づいて一連の文書化したものを示しながら説明するとよい。

(2) (1)のような例題が作れない時でも，標準的な文書の説明では，それぞれの文書の見本を用意する。文書の形式の見本は，参考文献〔5〕や〔6〕にあるので，参照するとよい。

(3) それぞれの手法の説明では，必ず例題を示すこと。

(4) 文書化の特徴を教えるには，文書化の過程を含めて説明するのがわかりやすくてよい。

(5) それぞれの手法を説明するにあたっては，その手法に適したよい例題を選ばなければならない。

(6) 流れ図自動作成，その他自動化の項では，それぞれの印刷出力の結果を示して説明する。

参考文献

〔1〕 D.Teichroew and E.A.Hershey, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on software engineering, vol, SE-3 pp.41-48, 1977

〔2〕 D.T.Ross and K.E. Schoman,Jr.,
"Structured Analysis for Requirement Definition", IEEE Transactions on software engineering, vol. SE3 pp.6~15, 1977

〔3〕 D.L.Parnas,"A Technique for Software Module Specification with Examples", Communications of the vol.15, pp.330~336, 1972

〔4〕 B. Liskov and S. zilles, "Specification Techniques for Data Abstactions", Proceedings International Conference on Reliable Software, SIGPLAN Notices, vol.10 pp.72-87, 1975

〔5〕 P.W. Metzger 「プログラミング・プロジェクトの実践的管理手法(Managing a Programming Project)」, 日本ビジネスレポート(株), 1973

〔6〕 D. Walsh "A Guide for Software Documentation", Advanced computer techniques Co., 1969

〔7〕 J. Bruno and K. Steiglitz,
"The Expression of Algorithms by Charts", Journal of the ACM vol.19, pp.517-525, 1972

〔8〕 B. Shneiderman, R.Mayer, D.Mckey and P.Heller,
"Experimental Investigations of the Utility of Detailed Flowcharts in Programming", Communications of thec ACM, vol.20, pp. 373-381, 1977

〔9〕 ハーマン・マクダニエル, "デシジョン・テーブル入門", 日本経営出版会 1970

〔10〕 H. Strunz, "The Development of Decision Table via Parsing of Complex Decision Situations", CACM, vol.16, pp.366-369, 1973

〔11〕 守屋慎次「混合エントリ Decision Grid Chart のループの性質と変換アルゴリズム」電子通信学会論文誌, vol. J60-D, pp.347-354, 1977

〔12〕 「HIPO — 設計の補助手段と文書化の手法」 日本アイ・ビー・エム(株) N: GC20-1851, 1975

〔13〕 I. Nassi and B.Shneiderman, "Flowchart techniques for structured Programming", SIGPLAN Notices (ACM), vol.8, pp.12-26, 1973

〔14〕 A.V. Gelder, "Structured Programming in Cobol : An Approach for Application Programmers", CACM, vol.20, pp.2-12, 1977

〔15〕 P.Roy, "Linear Flowchart Generator For A Structured Language", SIGPLAN Notices (ACM) vol.11, pp.58-64, 1976

〔16〕 "MARK IV FILE MANAGEMENT SYSTEM REFERENCE MANUAL", Infomatics, 1969

〔17〕 N.S.Prywes, "Automatic generation of computer programs", National Computer Conference, pp.679-689, 1977

〔18〕 M. Hammer, W.G.Howe, V.J.Kruskal and I.Wladawsky, "A Very High Level Programming Language for Data Processing Applications", CACM, vol.20, pp.832-840, 1977

第7章　テスト計画

キーワード

テスト(test)，デバッグ(debug)，モジュール単体テスト(module unit test)，結合テスト(integration test)，システムテスト(system test)，受入れテスト(acceptance test)，設置テスト(installation test)，テスト・ケース(test case)，全ダンプ(post mortem dump)，スナップ・ショット(snap shot)，トレース(trace)，デバッグ・モード(debug mode)，オンライン・デバッギング(online debugging)，(テスト用)シミュレーション(simulation)，テスト・データ自動生成(test data generator)，プログラムの自動検証(automatic verification)

目　標

プログラムの開発の中で大きな部分を占めるテスト段階について，その目的と方法を理解させる。テスト計画やテストの実施は，それぞれの開発対象プログラムや開発チームの構成などによって異なるが，基本的な考え方には共通な部分も多い。ここでは，テストについてのできるだけ具体的な内容を示し，それぞれのプログラム開発の場合に応じて，実際にテスト計画を立てられるようになればよい。そのためには，経験に基づいた部分の多い一般的な注意事項を説明することも，その中からテストについての考え方を拾い出すために役立つ。

また，デバッグやテストの手法は数多くあり，デバッグやテストのそれぞれの段階に応じて使い分けられる。それらの中から基本的な手法を選び，その手法の目的や効果を知り，それぞれの場合に応じて適切な手法が選択できるようになることを目標にする。

さらに，オンライン・デバッギング機能やシミュレーション，テスト・データの自動生成などのテスト支援用ツール(tool)は，すべてのコンピュータ・システムに用意されているとは限らないが，このようなツール(tool)がどのように生かされるかを知ることも必要である。

内　容

7.1　テストの概要

はじめに，テストと関連して使われる，次の用語の意味と概要を説明する。

・テスト(test)：プログラムを実行させてエラーを見つけること。

・証明(proof)：プログラムの環境とは無関係に，プログラムだけを対象としてエラーを見

つけること。たとえば，プログラムのコード群を形式化した論理システムとして扱い，与えられた入力表明（input assertion）から決められた出力表明（output assertion）が得られることを，数学的に定理づけて，帰納的に検証する。テストと違って，プログラムを直接実行させることはない。

・実証：テスト用の環境でプログラムを実行させて，エラーを見つけること。

・確認（varidation）：実際の環境でプログラムを実行させて，エラーを見つけること。

・保証（certification）：標準として認められているテスト用データを使ってプログラムを実行させ，その基準に合うことを保証すること。標準テスト用として，COBOLコンパイラ用のものや数学関係のプログラムのテスト用のものなどが作られている。

・デバッグ（debug）：デバッグは，テストの結果見つかったエラーをより詳しく調べ，エラーを直すことを目的に行なう作業で，テストとは区別される。デバッグ作業は，テスト段階の中で，必要に応じて発生する作業である。

テストの目的は，プログラムから，より多くのエラーを見つけ出すことである。しかし，すべてのエラーを見つけ出したと保証することは難かしい。テストによって，プログラムが正しく働くことを保証するためには，すなわち，良いテストを行なうためには，効率良く，より多くのエラーを見つけ出さなければならない。これがテストを行なう上での基本となることを理解させる。

テストは，次の手順で行なう。

・テスト対象を明確にする。

・テスト・ケースを設計する。

・テスト・ケースを書く。

・テスト・ケースが正しいことをテストする。

・テストを実行する。

・テスト結果を判定する。

これらの中では，テスト・ケースの設計がもっとも難かしいことを説明する。すなわち，テストの実施にもスケジュールと予算が決められているので，経済性を考えて，次のようにテスト・ケースを選ばなければならないからである。

・各テスト・ケースは，完全で適切であること。

・選ばれたテスト・ケース群は，実行が可能な程に充分小さく，しかし，テストの実行結果は，全ケースが正しいかどうかを示すことができること。

7.2 テストの段階

一つのプログラムのテストは，幾つかの段階に分けて行なう。そして，各段階ごとに，テスト

対象やテスト・ケースの設計，……など，テストの手順に示したことを決めて，実施していかなければならない。この各段階のテストの目的や特徴を説明し，それぞれの段階がプログラム開発の各段階に対応付けられていることを説明する。

テストの段階と，プログラム開発の設計段階との対応付けは，たとえば図７－１に示すようになる。

- 単体テスト（module unit test）。環境からは独立に，一つのモジュールを単位として，そのモジュールに与えられた機能をテストする。通常はプログラム作成者がモジュールの構造を解析し，また与えられた機能を詳細化して，テストケースを設定する。
- 結合テスト（integration test）。システムの各部分間のインタフェースを中心にテストする。各モジュールを結合し，テストを繰返しながら順次大きくまとめ上げていき，一つのシステムのテストになるまで続ける。このテストは，疑似環境の下で実施する。
- システム・テスト（system test）。システムに要求された機能が満たされているかどうかをテストする。結合テストまでが，システムの内部構造のテストであるのに対し，システム・テストでは，システムを外側から見てテストする。通常は，システム作成グループとは別のグループが，なるべく実際の環境に近い環境の下でテストする。
- 受入れテスト（acceptance test）。システムが利用者の要求を満たしているかどうかを確かめるために行なう。したがって，決められた効率で動作することの確認も含めて，実際の環境の下でテストする。
- 設置テスト（installation test）。そのシステムのための装置が設置されたのち，その施設上でテストし，設置時のエラーを見つける。

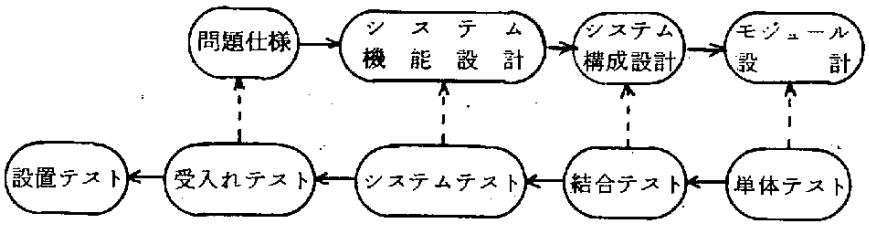


図７－１　テスト段階とプログラム設計段階との対応

7.3　結合テストの方式

モジュールを順次つなぎながらテストをする結合テストは，各テスト段階の中でも特に難かし

い段階で，幾つかの方式が考えられている。ここでは，次の代表的な方式について，その特徴を説明する。

- 上昇型（bottom-up）テスト
- 下降型（top-down）テスト
- 両方向型テスト
- 全部を一度に結合するテスト

(1) 上昇型（bottom-up）テスト

はじめに最下段のモジュールを単独にテストし，次にそれらのモジュールを呼出す，一段上のモジュールにテスト済みの最下段モジュールをつないでテストし，と，順に上位モジュールを加えてテストしていき，最上位のモジュールを加えたテストが終った時に終る。

上昇型（bottom-up）テストは，次の特徴をもつ。

- テスト・ケースの設定がしやすい。
- 幾つかのグループで同時にテストすることができる。
- テスト・データを，その時最上位となっているモジュールのインタフェースを通して与えなければならず，そのためのプログラムが必要となる。
- テスト・ケースを各モジュールごとに作ることになるので，テスト・ケースの数が多くなり，あまり大きなシステムには向かない。

(2) 下降型（top-down）テスト

最上位のモジュールから順に下位のモジュールをつなぎながらテストしていく。はじめに，最上位のモジュールの単体テストを行ない，次にそのモジュールが呼出すモジュールを直接つないでテストする。これを順に繰返し，すべてのモジュールを結合したテストが終った時に終る。結合の途中の段階では，下位の呼出すモジュールは，実際のモジュールの代りに，インタフェースだけを合わせた仮のモジュールをつないでテストする。

下降型（top-down）テストは，次の特徴をもつ。

- テスト・データは，常に最上位モジュールから幾つかのモジュールを通して与えなければならないので，テスト・ケースの設定が難かしい。
- 仮のモジュールの作成が負担になる。
- 下位に近いモジュールのテストや，エラー条件のテストが難かしい。

(3) 両方向型テスト

上昇型と下降型のそれぞれのもつ欠点を軽減するために，上からと下からの両方向からテストを進め，双方が合ったところでテストを終る。さらに，単体テストを終わらせたのちにこの型をとると，かなり改良された方式となり，大きなシステムのテストに向くことが多い。

(4) 全部を一度に結合するテスト

各モジュールの単体テスト後に全部を一度に結合してテストする方式である。この方式は簡単ではあるが，インタフェースの誤りは見つけにくく，またデバッグも容易ではない。広く行なわれているが，小さく，上手に設計されたプログラムでない限り，テストはなかなか完了せず，うまくいくことはない。

7.4 テストの計画と実施

テストの計画と準備は，プログラムの作成が終った時に始まるのではなく，プログラムの開発と並行して行なわなければならない。

テスト計画では，次のことを考えておく。このそれぞれについて，基本となる点を説明する。

- テストグループの構成と人員
- テスト仕様書の作成
- 計算機時間数やその他の費用の見積り
- テスト期間内のスケジューリング
- テスト準備とテスト実施開始の時期
- テストの進行管理方法
- テスト支援のための道具について

テストの実施は，テスト計画に基づいて行なう。したがって，テストの実施時に不都合なことが起るとすれば，それはテスト計画の立て方に起因する。テストの実行でエラーが見つかると，デバッグ作業が開始される。この作業と並行して，他のテストや他のデバッグ作業が行なわれることもあるので，これらの関連を考えて，モジュールの修正の時期など，テスト実施中の正確な管理が重要であることを理解させる。

テストの計画や実施にあたっては，次の事項に注意しなければならないことを理解させる。

- 「完全な」テストは実施不可能である。どこかで打切らなければならないが，どこで終らせるかの判断が難かしい。あらかじめ，妥当な点を設定しておかなければならない。
- テスト仕様の作成者は，プログラムの作成者と同じ人ではいけない。プログラムの作成の時に犯した誤りと同じ誤りをテストの時も犯したのでは，エラーは見つからない。
- テスト仕様書には，正しく実行された時の結果を明示しておく。実行の結果を見てから正しいかどうかの推定をするのでは，判断を誤りやすい。
- テスト仕様の作成には，一番優秀な人をあてるべきである。
- プログラムの設計時に，テストの容易さも考えておく。
- テストを容易にするという理由で，対象となるプログラムを変形させてはいけない。

・プログラムを修正したら，もう一度テストをやり直さなければならない。

・テストは実行させることが主ではなく，結果をよく検討することが主である。

・テストによって見つかるエラーが増加している時は，内在しているエラーも増加していると見るべきで，内在しているエラーが減少したと見てはいけない。

7.5 テスト・ケースの選択

テスト・ケースの選択が，良いテストのために重要なことはすでに説明した。ここでは，テスト・ケースの選択方法について理解させる。

テスト・ケースの選択方法として，次の各方法の特徴や長所，短所，どの範囲までのテストができるか，などを説明する。

・全数テスト（exhausive test）

・全経路（path）のテスト

・入力範囲をクラス分けする方法

・要求を分析する方法

(1) 全数テスト（exhausive test）

入力として可能な，全データをテスト・ケースとする方法である。この方法には次の欠点があるので，特殊なプログラム以外では使えない。

・実数値や整数値を入力とするプログラムでは，ケース数が無限にできるので実行不可能である。

・一般に，一つの入力データの取り得る値の数でもかなり多く，複数個の入力データについてでは，その組合せの数は膨大な値となって，実行不可能となる。

・テスト・ケースには，エラーとなる場合も含めなければならないが，これを含めると，ケース数はさらに増大する。

(2) 全経路（path）のテスト

プログラムの論理を分析して，テスト・ケースを選択する。この代表的な方法が，条件付分岐に注目して，①各条件単位に，その経路をテストする，②組合せが可能な全条件の組合せについてその経路をテストする，の二つである。これらには，次の欠点がある。

・②は，組合せの数が多くなり，実行不可能となることが多い。

・仕様とプログラムの論理との対応がテストできない。

・テストで与えたデータ値以外の入力値に対しては，何の保証もない。

(3) 入力範囲をクラス分けする方法

各入力の値の範囲を展開し，それらをクラス分けする。そして，一つのクラスの範囲内にあ

る一組のデータ値でプログラムが正しく実行されたならば，そのクラス内の他の値でも正しく実行されるであろうと推定してテスト・ケースを各クラスから一つずつ選ぶ方法である。

この方法は，次の長所をもつ。

・クラス分けをすることによって，仕様との対応が付く。

・テスト・ケース数が妥当な値となる。

この方法は，クラス分けのためにプログラムの論理や構造を解析しなければならず，そのためには，プログラムがわかりやすく作られていることが条件となる。

(4) 要求を分析する方法

プログラムの論理とは無関係に，仕様を分析してテスト・ケースを決めていく方法である。この方法では，要求をプログラムと対応付けて分析していないので，満足なテストをするためには入力として可能な値の全組合せが必要となり，全数テストと変わらなくなる。

7.6 デバッグ手法の概要

デバッグ手法の概要を理解させる。

①コード化の段階から，デバッグのための配慮が必要である。ガードのためのコードや，エラーの早期発見，デバッグしやすい情報の出力などに注意する。ガードのためのコードとは，モジュールに制御が移った時点でエラーを見つけ，エラーの範囲を局所的なものとするために組込む，チェック用のコードのことである。

②プログラムを実行させてデバッグする前に，机上デバッグを行なう。

③コード化の時，デバッグ文を挿入してプログラムの動作の記録をとっておく。誤った結果が出るか，動かなくなるまでプログラムを走らせてから調べるということはしない。

④コンパイラの持つ，デバッグ・モード指定を使って，不注意による誤った処理を防ぐ。

⑤デバッグ文による情報だけでは情報が不足する時には，全ダンプ（post mortem dump）をとる。ただし，全ダンプによってデバッグするためには，そのための準備も必要である。

⑥限定された実行の範囲の中で，変数の値など逐次変化する情報を詳細に得たい時には，トレースをとる。

⑦制御の移り方だけを知りたい時には，そのためのトレースを取る。

デバッグ支援とは，人と計算機の間をつなぐための道具である。より良いデバッグを行なうためには，このコミュニケーションの良いデバッグ支援を使うことや，そのための用意をしておかなければならないことを理解させる。

7.7 デバッグのためのコード化

デバッグを効果的に行なうためには，コード化の時から行なっておくべきことがある。ここでは，それらについて説明する。

デバッグのためのコード化の一つは，プログラム設計の段階から考慮しておくべきことである。それらを次に挙げる。

- 他のモジュールや，システムの外側から受取るデータは，必ずしも正しいデータばかりではないとしてコード化する。
- エラーは，可能な限り早く検出する。
- テーブルやレコード，制御ブロックなどには，ドッグ・タグ(dog tag)を付ける。ドッグ・タグとは，自分自身を識別するための文字列である。
- 誤った入力によって著るしい誤動作（無限回のループなど）が起る時には，必ず入力のチェックをする。
- システム内で共通に使う重要なテーブル類には，チェックサム（check-sum）を付けて，不注意による内容の変更を検出する。

他の一つには，コード化の時に加えるデバッグ文がある。デバッグ文は，プログラムがどう動作したかを調べるために，あらかじめコード化して入れておく文であり，それぞれの言語に用意されている。

デバッグが終った時，デバッグ文を手で取り除くのは大変である。また，プログラムが使われている時でも，デバッグ文を残しておき，プログラムに新たなエラーが発生した時，それらが使えるようにしておくことが望ましい。このためには，次のことが考えられる。

- コンパイラによっては，選択的コンパイルの機能をもっているので，この機能を使う。
- 選択的コンパイルの機能をもっていない時には，デバッグ文をマクロ化し，マクロ文の展開形を替える。

7.8 机上デバッグ

机上デバッグは，重要なデバッグ手法の一つである。机上デバッグに対しての認識と，その方法について理解させる。

プログラムをコンパイルし，コンパイラが検出したエラーを修正したのち，そのプログラムを実行させてテストする前に行なう作業が，机上デバッグである。机上デバッグとは，プログラム・コードを読む作業である。

机上デバッグには，次の利点がある。

- 早期にバグが発見できる。
- コンピュータの無駄な使用を防げる。

・論理的な誤りや，テスト・プログラムでは見つけにくい誤りを見つけることができる。

・言語の解釈上の間違いに起因する，論理上の誤りを防げる。

・外部仕様（そのプログラムへの要求を示す仕様）との食い違いを見つけられる。

机上デバッグには，次の二つの方法があり，その両方を用いるとよい。

①プログラム・コードを上から順に読んで行きチェックする。3～4人が集まり，1人が1行ずつ読んで行く。ここでは，ＩＦ文やＤＯ文の条件や，ＤＯ文の実行の範囲なども，注意してチェックする。

②2，3の簡単なテスト・ケースを用意し，データに値を与えて，そのテスト・ケースに従って各文の実行を机上でシミュレートする。3～4人が集まり，1人がプログラム・コードを実行順に読み上げ，他の人が変数の値の記録を取りながらチェックする。

7.9 全ダンプ

全ダンプ（post mortem dump）についての説明と，全ダンプで有効な情報を得るための準備について理解させる。

プログラムから要求があった時や，復旧が不可能なエラーが起った時，主記憶上のそのジョブに関連する内容を，すべて印刷出力することができる。これが全ダンプである。

全ダンプは，内容が主記憶上のイメージで出力されるので，

・オペレーティング・システムについての知識が必要である。

・高級言語の場合，コンパイル結果の目的プログラムの形を知らなければならない。など，相当な解析力が必要である。

また，全ダンプでより有効な情報を得るためには，次の準備をしておかなければならない。

①ドッグ・タグ（dog tag）を付けておく。

②実行に伴なって変ってしまう情報は，主記憶上の別の場所に保持しておく。

②では，たとえばサブルーチン呼出しなどの場合，次の内容を保持しておく。

・呼出したルーチンの名前

・呼出したルーチン内の呼出したアドレス

・各入力パラメタの値

・そのルーチンが変更する，共用データの値

・アセンブリ・プログラムでは，各レジスタの内容

・そのルーチンを呼出した回数

呼出しの回数が少なければ，これらを直接出力することも可能であるが（スナップ・ショット）呼出しの回数が多い時には，循環的に使うようなバッファを用意し，そこに記録する。こうすれ

ば，全ダンプの時に，最後の呼出しから遡って，幾つかの記録を得ることができる。

7.10 スナップ・ショット

スナップ・ショット（snap shot）の使い方を説明する。

スナップ・ショットとは，プログラム・コードに挿入されたデバッグ文によって，要求された内容を印刷出力する機能である。

全ダンプとは，次の点が異なる。

- 印刷出力したのちも，プログラムの実行は続けられる。
- スナップ・ショットは，要求された時だけしか印刷出力されない。
- 指定された部分だけを出力する。
- プログラム中の変数名と関連付けて印刷されるので，高級言語でもデバックしやすい。

スナップ・ショットでは，出力ののち引続きプログラムが実行され，何回でも出力と実行が繰返されるので，スナップ・ショットの頻度が多く，出力指定の量も多いと，多量の出力となる。プログラム・ループ内での要求の時などでは，良く考慮しておかなければならない。

7.11 トレース

トレース（trace）の使い方について説明する。

トレースは，プログラムの実行に伴なって情報を出力していく機能で，プログラム・コードがアセンブラか高級言語かによって，出力する内容も異なる。

アセンブリ・プログラムでは，レジスタの内容や，分岐命令で分岐した時の記録などを出力する。高級言語では，文に付けられた名札やセクション名，変数の値，サブルーチン呼出しの時のパラメタ値などを出力する。

トレースは，スナップ・ショットとは，次の点が異なる。

- 範囲指定などによって指定された範囲内のすべての文がトレースの対象となる。個々の文を単位に対象とすることはできない。
- 出力する情報は，トレースを取る変数名を指定するか（高級言語），または，あらかじめ決められた中から選択する（アセンブラ）機能となっていることが多い。
- 出力の指定方法によって，広い範囲に渡っての制御の移行や，または狭い範囲内の詳細な情報を得ることができる。
- スナップ・ショットのように，デバッグ文を加える必要がない。

トレースの出力は，スナップ・ショット以上に多量になりすぎることがある。他のデバッグ手段によって範囲を絞り，必要な部分を限定してからトレースをかけることが望ましい。

7.12 デバッグ・モード

デバッグ・モード(debug mode)機能の説明をする。

デバッグ・モード機能とは，この機能を使用することが指定された時，コンパイラがデバッグのためのコードを自動的に生成する機能である。たとえば，配列へのアクセスを含む文には，配列の下限と上限を越えたアクセスが行なわれるかどうかをチェックするためのコードが生成される。この機能によって不注意による許されないアクセスが，実行時に検出される。

プログラムが使われる時にまで，これらのチェック用コードを残していたのでは，プログラムの実行効率が悪くなる。したがって，デバッグ時だけチェック用コードを生成し，デバッグが終った時，コンパイルしなおして，チェック用コードを含まない目的プログラムを作る。

7.13 オンライン・デバッギング機能

オンラインの会話型デバッギング機能を持つシステムでは，この機能はデバッグのための極めて有効な支援となる。

ここでは，オンライン・デバッギング機能とその利点を説明する。

オンライン・デバッギング機能とは，端末からタイムシェアリング機能を使って，会話型にプログラムのデバッグが行なえる機能である。小さなコンピュータでは，タイムシェアリングでなく，コンピュータを専有する形で会話型デバッグを実現したシステムも多い。

デバッギング機能としては，たとえば次の機能をもつ。なお，機能の詳細については，参考文献〔5〕,〔6〕を参照するとよい。

- 区切り点(break point) を設定することで，プログラムを任意の場所で止めて，レジスタや変数の内容を見ることや，値を変えることができる。
- ある範囲を与えて，その範囲内を1ステップ単位でトレースすることができる。
- 任意の点から，何回でも，プログラムを再実行させることができる。

オンライン・デバッギング機能は，バッチ型の非会話型デバッギングに比べて，次の利点があることを理解させる。

- 非会話型デバッギングでは，あらかじめプログラムの動作を予想して情報集収のための機能を使うので，不必要な情報まで出力され，出力量が膨大になるか，必要な情報が欠けることが多い。
- バグを発見した時，仮修正ができるので，デバッグの収束が早い。
- ターン・アラウンド時間(turn around time)が極めて短かいので，短かい期間に効率の良い作業ができる。
- 実行結果を見ながら，次の実行のための決定ができるので，無駄な実行や，無駄な情報の集

収がない。

・局所的なトレースを利用して，バグ原因の絞り込みができるので，デバッグ効率が良い。

7.14 実時間プログラムのテスト

実時間プログラムのテストは，通常のプログラムのテストよりも，高度な技術を必要とする。ここでは，実時間プログラムの特性が，そのテストにどのような影響を与えるかを考察できるように指導し，他の特殊なプログラム（たとえば障害復旧用プログラムなど）でのテストについても類推できるようにする。

実時間プログラムは，通常のプログラムと，次の点が異なる。

・入出力に端末や回線を用いる。

・特別な技術が必要な機器を用いる（たとえば通信制御装置など）。

・多重プログラミング・システムであり，多重プロセサ・システムである。

・非同期に多くのメッセージが入って来るので，処理順序などの予測が難かしく，思わぬ所にエラーが発生する。

・より高い信頼性が要求される。

・プログラム間の関係が複雑である。

このような違いがあってテスト方法も難かしいので，テスト計画はより重要であり，場合によっては，プログラムを一時的に変更したり，扱う装置を別の装置に置き替えてテストする計画を立てなければならないこともある。

実時間プログラムのテストは，次のような支援のための道具を使って行なうことを説明する。

・シミュレーション——特殊な装置を模擬する。

・テスト用のスーパバイザ

・エラー検出用の支援——トレース，割込みのロギング，デバッグ用マクロ，ダンプ用の高優先度ルーチンなど。

・過負荷テストのための支援——テスト用データ生成プログラム，テスト出力解析プログラム，疑似多端末入出力用シミュレータなど。

実時間プログラムのテストについては，参考文献［7］を参照するとよい。

7.15 シミュレーション

この節では，シミュレーションをテストのための支援として使うことの有用性と問題点について理解させる。

シミュレーションがテストに有効であるのは，制御可能なプログラム環境を作ることができる

からである。シミュレーションは，二つの目的で使われる。

①通信回線などの特別な環境や，コンピュータ自身をシミュレートする。

②大型コンピュータの中に，小型コンピュータをシミュレートする。

①は，実時間プログラムのテストや制御プログラムのテスト（仮想コンピュータシステム（virtual machine system））などに使う。②は，小型コンピュータ用のプログラムをテストするために使う。

シミュレーションによって疑似環境を作ると，次の利点があることを説明する。

・主記憶の大きさによる限度を越えられるので，テスト用のプログラム等を自由に付加できる。

・入出力装置を扱いやすい仮想のものにすることができる（たとえば紙テープを磁気ディスクに置替えるなど）。

・デバッグのための区切り点を任意に設定できる。

・割り込みを疑似的に起こしたり，割込みが起きる状態を捕えることができる。

しかし，シミュレーションには，次の問題点もある。

・シミュレートしている環境が実際の環境と一致していることを保証することが難かしい。

・シミュレーション・プログラムをテストするのが難かしい。

7.16 テスト用データの自動生成

テスト用データを自動的に作り出すプログラムについて，その概要を説明する。

テスト用データの自動生成の方法は，次の二つの種類に分けることができる。

①対象とするプログラムの論理とは独立に，テスト用データを生成する。

②対象とするプログラムを入力として，そのプログラムをテストするためのデータを生成する。

①は，パラメタを与えることによって，そのパラメタに従ったテスト用データを生成するもので，多量のデータを使う時などに利用される。この種のプログラムの一つとして，数学関係のプログラムのテスト用に，正しい関係値を出力するサブルーチンのパッケージなどもある。

②は，対象プログラムの入力定義部分だけを参照して，対象プログラムへの入力となるテスト・データを生成するプログラムから，対象プログラムの論理構造を分析し，論理構造に合わせたテスト用データを生成するプログラムまである。前者は容易であるが，後者は，対象となるプログラムが複雑な構造をもっている場合には難かしく，まだ完全なプログラムは実現していない。

7.17 プログラムの自動検証

現在，プログラムの正当性を自動的に検証することが種々試みられている。この節では，自動検証の考え方について説明し，将来への方向付けができればよい。

プログラムの検証は，プログラムのテストとは異なった一面をもっている。たとえば，A，B，Cの三つの変数の値が互いに等しいかどうかを判定するのに，ＩＦ　Ａ＝（Ａ＋Ｂ＋Ｃ）／３とコード化した時，この誤りをテストによって見つけるのは難かしく，プログラムの検証が必要になる。

プログラムの自動検証には，次の方法があることを説明する。

・入力条件と出力結果を与え，ステップごとに数学的に検証していく（参考文献〔８〕）。

・変数に値を入れず，変数名をそのままの形でステップごとにインタプリティブに実行させていく。この結果，入力変数と出力変数との関係が確立される（参考文献〔６〕）。

・プログラムの自動生成（automatic program synthesis）で，プログラマが入力表明（input assertion）と出力表明（output assertion）を与えれば，自動的にコードを生成し，正しさの検証もする（参考文献〔９〕）。これは，「何を」プログラムにさせるかを与えれば，「如何に」それを行なうかを生成するものである。

一般的な言語で書かれたプログラムを，完全に自動的に検証できるシステムはまだ無い。

指導上の留意点

(1) テストについての，統一され，理論付けられた考え方や方式は少ない。ここでは，テストについての一般的な考え方の基礎を主として理解させる。

(2) テストは経験に基づいて実施されていることが多い。それらをなるべく系統立てることができるようになればよい。

(3) テスト・ケースの選択についての理論的な位置付けや，プログラム・エラーの類別，テスト・ケース選択の実例などは，参考文献〔１〕に記されている。これらについても説明できればより良いので，参照するとよい。

(4) テストについて書かれた本として，参考文献〔２〕，〔３〕などがあり，またテストについて書かれた章を含んだ本として参考文献〔４〕があるので，参考にするとよい。

(5) デバッグが主体となる部分（7.6～7.13）では，それぞれの例題を示しながら説明する。

(6) 全ダンプやトレースなどでは，コンピュータからの印刷出力の見本がある方がわかりやすいので，できるだけ用意する。

(7) どの場合にどの機能を使うかの選択は，一般的に説明するのはむずかしい。デバッグは経験に負う所が多いので，実習との組合せで理解させることが望ましい。

参考文献

〔１〕 J.B.Goodenough and S.L.Gerhart,"Toward a theory of test data

selection", Proceedings International Conference on Reliable software, SIGPLAN Notices, vol.10, pp.493-510, 1975

〔2〕 W.C.Hetzel 編 鳥居宏次訳「プログラム・テスト法」近代科学社, 1978

〔3〕 A.R.Brown and W.A.Sampson, "Program Debugging", Macdonald & Co.Ltd., 1973

〔4〕 G.J.Myers著 有沢誠訳 「ソフトウェアの信頼性――ソフトウエア・エンジニアリング概説」 近代科学社, pp.193-297, 1977

〔5〕 J.Blair, "Extendable Non-Interactive Debugging", Debugging techniques in large system, Prentice-Hall., pp.93-115, 1971

〔6〕 J.C.King, "A New Approach to Program Testing", Proceedings International Conference on Reliable Software, SIGPLAN Notices, vol.10, pp.228-233, 1975

〔7〕 J.Martin, "Design of Realtime Computing System", Prentice-Hall, INC, 1968

〔8〕 D.I.Good, R.L.London and W.W.Bledsoe, "An Interactive Program Verification System", IEEE Transactions on Software Engineering, vol. SE-1(1), pp. 59-67, 1975

〔9〕 Z.Manna and R.J.Waldinger, "Knowledge and Reasoning in Program Synthesis", Programming Methodology(Lecture Notes in computer Science 23), Springer, pp.236-277, 1975

第8章　プログラムの設計

キーワード

モジュール化（modularization），抽象化（abstraction），精練（refinement），段階的精練（stepwise refinement），情報の隠蔽（information hiding），上昇型設計（bottom－up design），下降型設計（top－down design），ネスト型仮想機械（nested virtual machines），構造的プログラミング（structured programming），高水準言語（higher level language），データ型（data type），抽象データ型（abstract data type）

目　標

この章の目標は，プログラムの設計ができるようになることである。設計の良否は，設計者の設計に対する考え方に負う部分が大きい。そのためには，設計の技法を学ぶだけでなく，技法の背景となる考え方を理解しなければならない。技法は，その考え方を十分に理解した時にだけ生かされる。

この章では，ソフトウェア工学の一部分として位置付けられている，モジュール化や構造的プログラミング，データの抽象化，それらを支える高水準言語などについて説明し，プログラムの設計やプログラミングについての現在の技術を理解するように指導する。そして，さらにソフトウェア工学の他の分野や，将来の技術進歩を吸収するための基礎を作ることを目標とする。

内　容

8.1　プログラムのモジュール化

プログラムのモジュール化（modularization）について，次のことを説明し，モジュール化が理解できるようにする。

- モジュール化の目的
- 優れたモジュール分割の形態
- モジュール化で注意すること

大きなプログラムでは，大きな複雑な問題を扱かわなければならない。モジュール化とは，この大きな問題を扱かうときの複雑さを減らすために，それを個々に解決が可能な，小さな単純な問題の集まりになるまで，分割をすることである。

モジュール化には，次の二つの異なった方法がある。

① 機能の表面的な性質の違いに基づいて分割する。異なった機能は異なったモジュールになるので，各モジュールは他のモジュールとは独立に，または並行して設計できる。

② 機能の抽象化（abstraction）と精練（refinement）を繰返しながら分割する。抽象化（abstraction）とは，異なった要素の中から共通の概念を抽出してまとめることである。精練とは，まとめたものを，さらに詳細な，そしてより一般的になるような要素群に分解することである。この繰返しで，モジュール化を行う。

この二つのモジュール化の方法について，例を挙げて説明する。例題は，参考文献〔1〕の中のプログラムがよい。

①は，従来から行われている流れ図を主体としたモジュール化の方法であり，②は，情報の隠蔽（information hiding）という考え方を基礎にしたモジュール化である。情報の隠蔽（information hiding）は，良いモジュール化の原則となることであるから，②によるモジュール化の利点を挙げながら，その考え方を説明する。

情報の隠蔽とは，情報を必要とする部分以外には見せず，設計者が意図した情報以外は使えないようにすることである。このように，情報の利用を限定することによって，モジュール間のインタフェースは簡単になり，また設計者の意図と異なったプログラムの開発を防ぐことができる（参考文献〔19〕）。

②によるモジュール化には，次の利点がある。

- 理解しやすい
- 機能や処理方法を変更しやすい
- それぞれのモジュールを，独立に開発しやすい

しかし，良いモジュール化は，容易なことではないことも理解させる。良いモジュール化のための次の三つの主要な条件が，互いに競合関係にあることや，モジュール化が上手に行われなかった時の弊害についても説明する。

① モジュールは互いに独立し，モジュール間の関係はわかりやすく，各モジュールはそのモジュールに与えられたことだけを行う。

② モジュールは小さいこと。①の条件を満たすためには，モジュールの単位を大きくすれば良いが，それでは複雑さを増し，拡張や変更が難かしくなる。

③ 著しく効率を落してはいけない。モジュールの単位をあまりに小さくすると，モジュール間の制御の受渡しが多くなり，効率が著しく悪くなることもある。

モジュール化が上手に行われなかった時の弊害としては，次の点がある。

・一つの機能を，異なった機能をもつ多くのモジュールによって実現すると，その機能の

持つ論理が隠れてしまって，わかりにくくなることがある。

・共通な機能が設計段階で完全に抜出せないことがあり，結果として，共通な機能が一つにまとまらず，多くの異なったモジュールの中に分散してしまうことがある。

・各モジュールが，共通な，または共用しているデータに，不適切な方法でアクセスすることがある。

8.2 モジュール化の手法

モジュール化の手法と，それぞれの手法の特徴を説明する（参考文献〔19〕）。

与えられた問題は，その問題に合った概念で示されている。一方，プログラミング言語は，汎用的で基本的な概念を持っている。この二つの概念の間を，プログラムが結び付けなければならない。この間をいかに結び付けていくかによって，モジュール化の手法は，次の三つに分類できる。

- 上昇型（ bottom-up ）による手法
- 下降型（ top-down ）による手法
- ネスト型仮想機械（ nested virtual machines ）による手法

(1) 上昇型による手法

プログラミング言語を基礎にしてプログラムを組立てていく手法である。上昇型手法では，はじめに基礎的なモジュールを考え，それらの上に次のレベルのモジュールを積上げていく。これを繰返して，与えられた問題を解決するプログラムを構成する。

(2) 下降型による手法

与えられた問題の概念から出発して，モジュールの分割と組立を行う。初めに，問題の解決を直接示すモジュール群を考える。しかし，この各モジュールは，解決しなければならない，多くの，そして詳細化されなければならない問題を含んでいる。次にこれらの新たな問題群を解決するために幾つかのモジュールを考える。この個々のモジュールの中にも，新たな，より詳細化されなければならない問題（群）が含まれている。この新たな問題（群）を解決するために，さらに……と，このような操作を，プログラム言語だけで解決できるモジュールになるまで繰返して，プログラムを構成する。

(3) ネスト型仮想機械（ nested virtual machines ）による手法

問題と実際のコンピュータの，二つの異なった概念の間を幾つかのレベルに分ける。この分けられた各レベルに，一連の仮想的なコンピュータを対応付けて置き，それらの仮想コンピュータの間をつなぐ一連のプログラムを作っていく手法が，ネスト型仮想機械（ nested virtual machines）による手法である。この手法では，プログラミング言語も一つの仮

想コンピュータのレベルと考える。最下段のレベルは，実際のコンピュータと等しくなる。各レベルは，問題を抽象化し，精練していく時の，それぞれの抽象化のレベルに対応する。

これらの三つの手法には，次のような特徴があることを説明する。

(1) 上昇型手法の特徴

- 明瞭な概念をもつプログラム言語を基にして組立てていくので，考えやすく，プログラミングしやすい。
- すでに定義されたモジュールの上に上位モジュールを作るので，安定感がある。
- 作り上げたモジュールから順にテストができる。
- 設計が完了しないと，プログラミングできない。
- 各モジュールの最適化を計っても，システム全体の最適化にはなりにくい。
- 設計の手法としては，目的である問題の概念を捕えにくいので，不適切である。

(2) 下降型手法の特徴

上昇型の項とはほぼ逆の特徴をもつが，そのほか次の点が挙げられる。

- 問題の概念を基にしているので，その問題に適したプログラミングが行える。
- 下位モジュールが作られていないので，プログラミングの基礎が不安定で考えにくい。
- 設計が完了していなくても，上位モジュールから順にテストできるが，まだ作られていない部分を考慮してテスト・ケースを作ることや，作られていない部分の疑似モジュールを必要とするので，煩雑になる。
- 設計と並行してテストしても，実際の環境でなく，疑似モジュールの上でのテストなので，正しさの確証が得にくい。
- 設計の手法としては，全体的なモジュール分割のバランスをとることができ優れているが，以前に同種の設計の経験があり，何が妥当であるかを見つける直感が無いと難かしい。

(3) ネスト型仮想機械手法の特徴

- 適切な仮想機械を設定することによって，複雑な問題の設計が容易になる。
- 制御構造とデータ構造の両方を抽象化していくことができる。
- モジュールをレベル毎にクラス分けしてグループ化することができ，各レベル単位に分割して，独立に開発できる。また，各レベル単位に独立して検証できる。
- ある仮想機械より下の部分を他のコンピュータ用のプログラムで置きかえることによって，互換性がもてる。
- 適切な仮想機械の設定が難かしい。

これらの三つの手法は，問題の性質や場合に応じて使い分けられるが，一般的には，設計は下降型で，プログラミングは上昇型で行うのが良いようである。また，オペレーティング・シ

ステムのように特別な，または大きなシステムを作る時には，ネスト型仮想機械の手法も良い。

8.3 モジュール化によるプログラムの設計

実際のプログラムの設計について，例を挙げて説明し，モジュール化の概念を理解すると共に，実際のプログラムの設計でどう応用されるかを説明する。

設計手法の例としては，構造化設計（structured design）（参考文献〔3〕）や複合設計（composite design）（参考文献〔4〕）などの下降型設計手法による方法がよい。また，ネスト型仮想コンピュータの設計の例として，参考文献〔5〕の抽象機械（abstruct machine）について取上げるのもよい。

モジュール化をする時の基準を与えることは，良いモジュールとはどのようなモジュールであるかを説明することでもある。良いモジュールの条件として，次に挙げる各項を説明する（参考文献〔3〕，〔4〕）。

- モジュール間の関係が最小であること。
  すなわち，モジュール間の独立性が高いこと。モジュール間の独立性が悪いと，1モジュール内のバグや修正による派生的な影響が他のモジュールに現われ，デバッグしにくくなり，保守性も悪くなる。
- 一つのモジュール内の各要素間の関連性が強いこと。すなわち，そのモジュールに明確な機能が与えられていて，その機能を遂行するためだけの要素から成っていること。機能が明確でなかったり，他の要素を一緒に含んでいると，他のモジュールのバグや修正による影響を受けやすい。
- モジュールの機能が簡潔に表現できること。これは，プログラム全体の構成をつかみ，モジュール構成の正しさからプログラム全体の正しさを計るためにも必要である。
- モジュールの大きさが適当であること。モジュール単位に，そのモジュールの正しさを人が検査でき，内容を把握できなければならない。

モジュール化によるプログラムの設計の一つに，モジュールの定義をより明確に与え，そして，その定義から可能となる検証系も含んで一つのシステムとしたH. O. S.（High Order Software）システム（参考文献〔6〕）がある。このシステムでのモジュールの定義を参考として説明するとよい。これは，モジュール化によって，信頼性のより高いシステムを設計，開発する場合の考え方の一つとして有益である。

オペレーティング・システムなど，機能の関係が複雑なシステムでは，仮想機械によるモジュール化も，その一つの方法である。この時，モジュールの概念やレベルと，機能の階層とは一致しないので，機能の階層構造だけに基づいて設計を行い，設計とモジュール化などの実現方法と

は分けて考えなければならないという意見もある（参考文献〔7〕）。

この方法では，仮想機械の各レベルは，レベル内に閉じたモジュールだけで構成されるのではなく，モジュールによっては，一つのモジュールが幾つかのレベルに渡って作られることもある。また，機能的な階層を主体にすると，モジュール化のレベルは，従来のモジュール化のレベルよりも，より細かい階層を作ることになる。この考え方を，モジュール化の一つの方向を示すものとして説明する。

8.4 プログラムの開発

システムが大きくなり，システムの持つ機能が複雑になるに従って，設計の初期の段階ではプログラムの完全性や，これから起るであろうプログラムの変更などを見通すことは，より難かしくなる。ここでは，このような場合に対処するための次の三つの手法について説明する。

- 下降型による開発
- 改良法（iterative enhancement）による開発
- 疑似言語による開発

これらの方法は，いずれも，まずはじめに容易で簡単な方法でプログラムを作り，そのプログラムの正しさを確かめながら，より詳細に，またより効率の良いプログラムへと改良していく方法である。

(1) 下降型開発

下降型の手法を，プログラムの設計から開発やテストの段階まで含めて行う方法である。まず初めに最上位のレベルのモジュールを設計し，プログラミングする。このモジュールが呼出している次のレベルのモジュールは，未定義モジュールとして呼出し手順だけを決めておく。次に，これらの未定義モジュールを作る。それらのモジュールが呼出している，その下のレベルのモジュールは，同じように未定義モジュールとしておく。この方法を繰返して順に下位のモジュールを作っていき，プログラムを完成させる。テストは，作られたモジュールを順に結合させて，プログラミングと同時に実施していく。

(2) 改良法による開発（参考文献〔2-0〕）

初めに，完全ではあるが簡単に，効率などは全く考えないでプログラムを作り，実行させる。そして，正しく実行することを確かめたのちに，そのプログラムを実行させながら効率等の測定を行い，目的に合うプログラムになるまで，順次改良していく。

(3) 疑似言語による開発

コンピュータによるコンパイラを持たない言語でよいから，高度な水準をもつ，記述性のよい言語で，簡単にプログラムを作る。そのプログラムを，コンピュータによるコンパイル

が可能な他の言語に，手でコンパイルし，対応するアルゴリズムに置きかえていって，実行可能なプログラムにする。その後プログラムを実行させながら，効率が良くなるように改良していく。

8.5 構造的プログラミング

構造的プログラミングの考え方と実現の形について理解させる。はじめに，構造的プログラミングが提案された背景を説明し，次に構造的プログラミングの形を，終りに構造的プログラミングの一応の定義を説明する。すなわち，次の各項を説明する。

- 背景 ── プログラムの信頼性について
- 背景 ── ＧＯＴＯ文無しについて
- ＧＯＴＯ文無しを実現するためのプログラムの基本構成要素
- 抽象化について
- 構造的プログラミングの幾つかの定義

(1) プログラムの信頼性について

現在，プログラムの信頼性が重要視されている次の理由を説明し，プログラムの品質がいかに大切であるかを認識させる。

- ハードウェアの大型化，高性能化に伴なうプログラムの規模の大型化と複雑化から，プログラム開発に於ける人件費の増加と，開発過程での信頼性が及ぼす費用への影響の増加。
- プログラムの長寿命化と，その間での保守や改造の費用がプログラムの品質に依存すること。
- 社会環境の中でのシステムの占める役割の増加により，プログラムの誤りによるシステムの停止や誤動作が与える影響の増大。

(2) ＧＯＴＯ文無しについて

ＧＯＴＯ文無しを良いプログラムの条件とする理由を説明する。その中で，次のことを解説する。

- プログラムのコード化された状態の構造（静的構造）と，プログラムの実行時の構造（動的構造）が一致していることが，わかりやすいプログラムの条件として重要であることについて
- ＧＯＴＯ文がプログラムの動的構造に与える影響によって，有害なＧＯＴＯ文と無害なＧＯＴＯ文に分けて考えられることについて
- 無害なＧＯＴＯ文が高級言語の中で，いかに構文的に他の文の中に吸収されるかについてと，それゆえ高級言語が人とプログラムを結ぶ役割りをもつことについて。

• どうしたらスパゲティ・プログラムを作らずにすむか ―― プログラムの構造化について。

ＧＯＴＯ文問題については，参考文献〔９〕を参照すると良い。なお，この文献中にあるＧＯＴＯ文問題の歴史に触れておくのもよい。

(3) ＧＯＴＯ文無しを実現するためのプログラムの基本要素

図８－１に示す三つの基本要素，連結構造（sequence ／ succession），選択構造（choise ／ decision ），繰返し構造（ iteration ）を説明する。

一つの入口と一つの出口を持ち，実行されない部分や無限の繰返しを含まないプログラムならば，この三つの基本要素を再帰的に使うことによって，どのようなプログラムの制御構造でも表現が可能なことが示されている。この三つの要素によるプログラムの構成を，PL／1言語を使ったプログラムで，例を挙げて説明する。この時，再帰的に構成されたネスティングの関係を陽に示すために，各行のコードの開始位置の段階づけを行う。

プログラミングを容易にするために，三つの基本要素に，図８－２に示す二つの要素，ＤＯＵＮＴＩＬ文とＣＡＳＥ文を加えてもよい。この二つの要素は，三つの基本要素を使って構成することができる。

(4) 抽象化について

構造的プログラミングの目的は，わかりやすく，人が検証することが可能な構造をもったプログラムにすることである。このためには，人の持つ概念の構造に合わせた構造をもたなければならない。人の持つ概念の構造は，ある目的にとって不必要な部分は落し，必要な部分だけに注目する，という形をもつ。すなわち，抽象化による構造が，もっとも人の考え方に合わせた構造といえる。

構造的プログラミングとは，抽象化の考え方に基づいてプログラムの構造を作っていくこと，またはそのようにして作られた構造をもつプログラムであるということもできる。

ここでは，ＧＯＴＯ文無しや構造的プログラミングの三つの基本要素などは，構造的プログラミングの表面に現われた形であり，それらが構造的プログラミングのすべてではないことを，理解させる。

(5) 構造的プログラミングの幾つかの定義

現在，構造的プログラミングという用語は，開発や設計までを含めた広い範囲に渡っての考え方として捕えることと，単なるコード化のための技法として捕える考え方（構造的コーディング（ structured coding ））の両方の意味に使われている。ここでは，それらの幾つかを説明して，構造的プログラミングについてのまとめとする。なお，二つの定義を例として挙げておく。

① 次の二つの規則で定義する（参考文献〔８〕）。

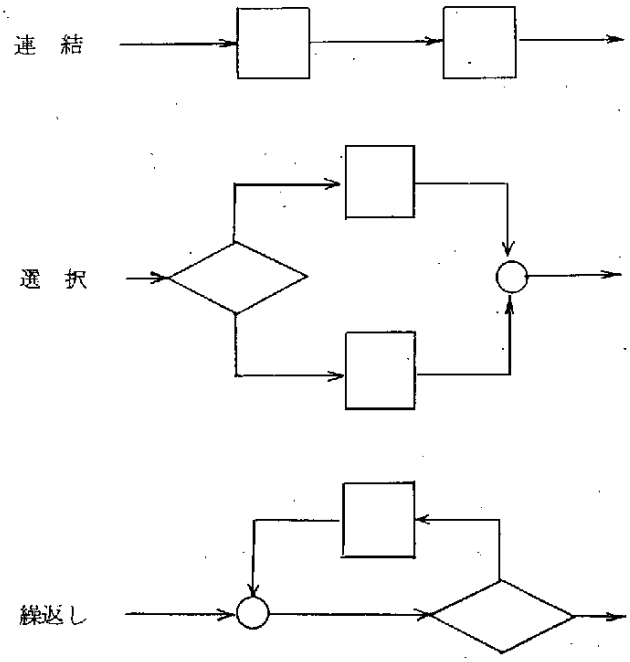


図８－１　　構造的プログラミングの三つの基本要素

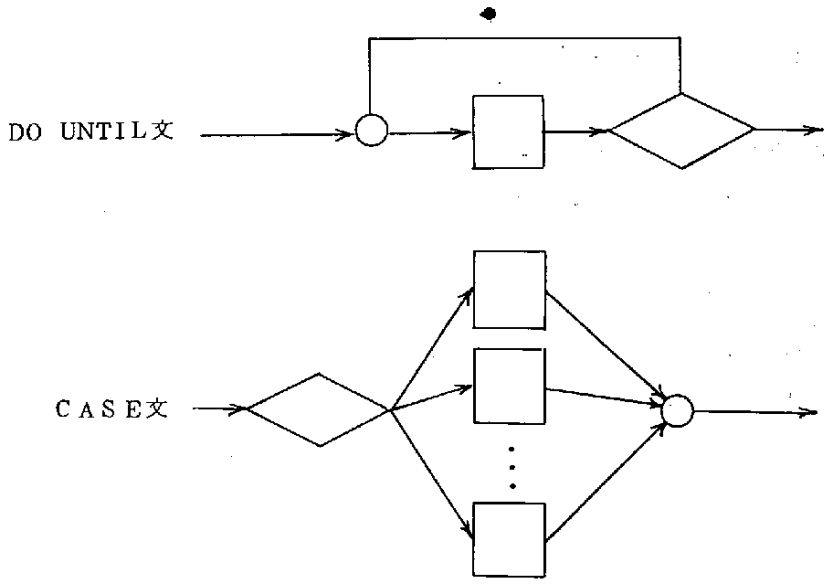


図８－２　　構造的プログラミングの二つの追加要素

- 構造的プログラミングは，レベル単位に上から下へと開発していく手法である。最上位のレベルでは，最上級の機能要素間の制御の流れを記述する。各機能要素には名前が付けられ，その名前がその機能要素を示す。名付けられた各機能要素は，次のレベルで，そのレベルでの機能要素間の制御の流れとして記述し，定義する。そこでまた要素の名前が現われる。これを繰返し，未定義の要素の名前が無くなった時終る。
- 各レベルの制御構造は，連結，選択，繰返しだけを許す。文は，より低い要素の名前を使って作ることができる。

② ①よりも具体的な定義である。構造的プログラミングとは，コンピュータとのコミュニケーションでなく，人とのコミュニケーションを指向してコード化する態度であるとして，次の条件を与える（参考文献〔10〕）。

- コードは，連結，選択，繰返しの三つの基本要素から構成されること。
- 可能な限りＧＯＴＯ文は使わない。とくにプログラム・リストの前方へ分岐するＧＯＴＯ文は最悪である。
- わかりやすい形で書くこと。
- リスト上の実行の断点間のつながりが簡単に追えること。
- 各モジュールは，一つの入口と一つの出口しか持たないこと。
- コードは読みやすいように，リスト上でセグメント化されていること。１モジュールの
- 実行文の部分は，１ページに入り切ること。
- 単純化され，問題の解が直ちにわかるように表現されていること。

8.6 構造的プログラミングによるコード化

構造的プログラミングによるコード化の手法として，段階的精練（ stepwise refinement）法を説明する。

構造的プログラミングが，E. W.ダイクストラの言う，プログラムの知力による扱いやすさ（intellectual manageability）を主体とするものであるならば，これを手法として実現したのが段階的精練（ stepwise refinement）の手法である。

段階的精練は，プログラムの構造をわれわれ人間の概念の構成に合わせようとしたもので，8.1節でモジュール化の方法としてすでに触れた，抽象化と精練を繰返して行く方法を，コード化の手法にしたものである（参考文献〔11〕）。

ここでは例題を挙げて，実際に段階的精練を行いながら解説する。例題としては，参考文献〔11〕，〔12〕 など多数あるが，プログラミング的には，参考文献〔11〕 の８個のクイーンの問題がおもしろい。

8.7 プログラムのスタイル

プログラム構造の基本的な部分として，またはより具体的なテーマとして，プログラムのスタイルの良さについて理解させる。

ここでは，①良いスタイルのための基本的な原則，②その原則に従がわないとどうなるか，③プログラムを改良する方法，などを，それぞれについて例を挙げながら説明する。原則や一般論だけを話すよりも，例題に基づいて進めた方が理解しやすく，また充実した内容となる。

良いスタイルのための基本的な原則を次に挙げる。

・コードの表現が明瞭で読みやすいこと。

・コードの構造が明瞭で読みやすいこと。ＧＯＴＯ文の代りにＩＦ－ＴＨＥＮ－ＥＬＳＥ文やＤＯ文，ブロック，サブルーチンを使い，またデータの表現にも注意深い配慮をすること。

・不当な入力や，境界域にある入力も含めて，どのような入力に対しても正しく働くこと。

・効率よりも明瞭さが優先すること。効率が悪ければ，アルゴリズムを変更する方が良い。

・プログラム自身が文書となるコードにすること。

例題と，より詳細な原則は，参考文献〔13〕を参考にするとよい。

8.8 データの抽象化

これまでにも抽象化について触れて来たが，ここで一度整理して説明する。はじめに，制御構造の抽象化を例として説明し，引続いてデータの抽象化について説明する。

抽象化とは，その時点で本当に必要とする詳細だけがわかり，それよりも詳しいレベルの，不必要な詳細事項は見せないようにするということである。たとえば，あるものを使う時に，それに何ができて，どう使えばよいかだけが必要であり，どのようにできているかは不要である。これは情報の隠蔽であり，制御構造を抽象化することがモジュール化の基本であった。すなわち，関数やサブルーチンとして一つのモジュールにした時，そのモジュールを使う側では，そのモジュールの機能と呼出し方だけを知ればよく，どのように遂行されるかは知る必要がない。

制御構造の抽象化だけでなく，データ構造の抽象化も必要である理由を理解させる。

構造的プログラミングであっても，制御構造だけの抽象化では，共通なデータや共用，共有しているデータ構造が変更された時には，二ケ所以上のプログラムの変更が必要となる。これを避けるためには，共通なデータのデータ構造を保持しているモジュールを設け，他のモジュールは必ずこのモジュールを通してデータにアクセスすればよい。こうすれば，データ構造は知らなくても，データへのアクセスはできる。これは，モジュール化をデータ構造にまで適用した方法で，データをアクセス手段という形で抽象化したことになる。

制御の抽象化に対して，データの抽象化はまだあまり行われていない。次に挙げるその理由を説明する。

・特定の言語でしかデータの抽象化機能をサポートしていないので，一般の言語では，モジュール呼出しが多くなり，効率を落す可能性がある。

・制御が主体の考え方がされていたこと。データは値の集まりがデータ構造になるとして捕えて，データの持つ性質からデータ構造が決まり抽象化できるということが，制御構造の抽象化ほど明示的でなかったこと。

データの抽象化の方法と，データ型（ data type ）と対象（ object ）について説明する。

データの抽象化をするには，特定の性質を持ったデータの集まりと，そのデータへの操作の集まりを一つの型（ type ）として定義すればよい。データの実現構造は，データをアクセスする側では知らなくてもよい情報となり，必要なのは特定の性質をもったデータの集まりに対する操作方法だけとなる。操作方法を定義する部分を作るときだけ，データの実現構造を意識すればよい。

同じ特定の性質をもったデータをデータ型，すなわち一つの抽象データ型（ abstract data type ）として定義した時，データの集まりの実際の値を対象（ object ）と呼ぶ。対象の識別は，そのデータ型のデータ操作を呼出す時，どのデータ値の集まりを扱かうかを決めるためのパラメタになる。

8.9 高水準言語

プログラム言語の中でも，構造的プログラミングや抽象データ型などの考えを積極的に取り入れた言語を，特に高水準言語（ higher level language ）と呼ぶことがある。ここでは，これらの言語の特徴的な部分だけを，その考え方を主体に説明する。個々の言語の詳細や使い方を説明するのでなく，プログラム言語の現在のあり方と，将来への方向付けができればよい。

高水準言語の特徴は，次の三つに代表される。

・制御（アルゴリズム）の構造化と抽象化が考慮されている。

・データの抽象化ができる。

・検証を可能にしている。

幾つかの代表的な言語によって，これらの言語仕様上での実現方法を簡単に説明する。なお，全体的な内容を捕えるには，参考文献〔14〕を参照するとよい。

(1) ＳＩＭＵＬＡ６７ （参考文献〔15〕）

クラス（ class ）の概念と対象（ object ）について簡単な説明をする。ＳＩＭＵＬＡ６７では，クラスという定義を使って，アルゴリズムとデータの両方の型を混在させて宣言する

ことができる。すなわち，クラスとはアルゴリズム付きのレコードと見なすことができる。クラスに対してサブクラス（ subclass）が何重にも定義でき，サブクラスを使うことによって，クラスをより精密に定義していくこと，すなわち段階的な精練ができる。

クラスに対してnewという命令を実行させると対象ができる。

ＳＩＭＵＬＡ６７は，シミュレーション・プログラムのように複雑で相互作用の多いプログラムを容易に作れるようにするために開発された言語であり，その中のクラスや対象の概念は，構造的プログラミングや抽象データ型の概念の基となっている。

(2) ＰＡＳＣＡＬ（参考文献〔16〕）

ＰＡＳＣＡＬはＡＬＧＯＬ６０を基に，データ定義機能を充実させた言語で，言語構造が単純で系統的なので，アルゴリズムの表現手段としてもよく使われている。ここでは，型について説明する。

ＰＡＳＣＡＬでは，使用する変数には必ず型宣言（ type declaration）が必要である。この型宣言を使って，新たなデータ型を自由に定義する機能があることが，ＰＡＳＣＡＬの特徴となっている。型宣言は，type 文で，たとえば，次のように行う。

type color = (red,yellow,green,blue,violet)

このように新たな型color を定義し，その型のとることのできる値を五つの色名（ red, …）することができる。変数はvarという宣言文で定義し，同時に type 文で宣言された型の名前を与える。すなわち，var card : colorのように宣言する。

この機能は，データへの操作の定義にはならないが，データ操作上のプログラミングの誤りを防ぎ，抽象データ型に近づいた機能と見ることができる。

(3) ＣＬＵ（参考文献〔17〕）

抽象データ型を実現した言語である。ＣＬＵの中のクラスタ（ cluster ）の概念について説明する。

ＣＬＵでは，抽象データ型を扱う基準として，次の三つを決め，この基準を満すクラスタと呼ぶモジュールが記述できる。

- データ型の定義には，そのデータ型の対象に適用される，すべての操作命令の定義も同時に含んでいなければならない。
- 抽象データ型の利用者は，そのデータ型の対象が，主記憶上でどのように表現されているかを知る必要はない。
- 抽象データ型の利用者は，そのデータ型に属す操作命令を使ってだけしか，その対象であるデータを操作することはできない。したがって，主記憶上に直接的にアクセスすることはできない。

プログラマが抽象データ対象を利用する時には，その対象への操作命令だけが与えられる。データ表現がどのように実現されるかは，クラスタモジュールの作成者だけに閉じられていて，クラスタを利用する側からは分離されている。すなわち，クラスタは，データ型をデータとその上で実行できる操作命令群だけで定義しようとするものである。

なお，データ型として特徴付けられていないデータを扱う命令（たとえば，実数をパラメタとする正弦関数ルーチン）は，関数的抽象（functional abstruction）として，procedure文で定義するモジュールとなる。

モジュールからの外部参照が可能なのは，クラスタ名かprocedure名だけに限られ，共通データや共用データを定義することはできない。

(4) ＡＬＰＨＡＲＤ（参考文献〔18〕）

検証を可能にした言語としてＡＬＰＨＡＲＤを取上げ，簡単に説明する。

ＡＬＰＨＡＲＤでは，抽象化の定義としてフォーム（form）という宣言を用いる。フォームは，制御構造とデータの両方の抽象に用いることができる，広い機構を持っている。このフォームは，仕様（specifications），表現（representations），実現（implementations）の三つの部分から成り，それぞれ次のような内容をもつ。

① 仕　様

そのフォームの仕様を記述する。すなわち，抽象化の領域（domain）の定義，各抽象型の初期値，各命令の実行前と実行後の条件などを記す。

② 表　現

そのフォームの構造を表現する。すなわち，そのフォームが使われた時の初期設定や，抽象化されている仕様を具体化して詳細化することなど，仕様に示された抽象化の内容を具体的に表現する。

③ 実　現

仕様に示された機能の主体となる部分である。各機能の入力と出力の具体的な表明（assertion）も含めて，そのフォームに属す機能の実現法（アルゴリズム）を記述する。

ＡＬＰＨＡＲＤでは，仕様や表現の部分をもつことによって，フォーム単位に，その実現の正当性を検証することができるようにしている。

指導上の留意点

(1) この章では，制御構造の抽象化であるモジュール化から，データの抽象まで広い範囲のテーマを含んでいるが，抽象化とその必要性が全体を通しての重点である。これを基準に，各節が関連をもって理解できるように説明する。

(2) それぞれの考え方を理解することが必要であり，プログラム設計の実習は，この章では，時間が許せば実施した方が良い程度である。

(3) できる限り例題を通して説明すること。例題を使うことによって，わかりやすく，身近なものとなるであろう。例題は，それぞれの所に示した参考文献の中から拾うことができる。

(4) 高水準言語の個々についての詳細な説明は必要ない。特徴的な部分をわかりやすく解説する。ただし，説明のための例題としてプログラムの一部を挙げることが望ましく，そのため，言語の主要な部分の説明が多少必要となるであろう。

参 考 文 献

〔1〕 D. L. Parnas, "On the criteria to be used in decomposing system into modules」, communications of the CACM, vol.15, pp.1053−1058, 1972

〔2〕 D. L. Parnas, "Information distribution aspects of design methodology", Information Processing 71, North−Holland Publishing Co. pp.339−344, 1972

〔3〕 W. P. stevens, G. J. Myers and L. L. Constantine, "Structured design", IBM System Journal pp.115−139, 1974

〔4〕 G. J. Myers 著 久保未沙他訳 「高信頼性ソフトウェア── 複合設計」近代科学社, 1976年

〔5〕 E. W. Dijkstra, "The Structure of the THE Multiprogramming system", communications of the CACM, vol.11, pp.341−346, 1968

〔6〕 M. Hamilton and S. Zeldin, "High Order Software — A Methodology for Defining Software", IEEE Transactions on Software Engineering, vol. SE−2, pp.9−32, 1976

〔7〕 A. N. Harbermann, L. Flon and L. Cooprider, "Modularization and Hierarchy in a Family of Operating Systems", communications of the CACM vol.19, pp.226−272, 1976

〔8〕 B. H. Liskov, "A desigh methodology for reliable software systems", Fall Joint Computer Conference pp.191−199, 1972

〔9〕 D. E. Knuth, "Structured programming with goto statements", Computing Surveys, pp.261−301, 1974

〔10〕 G. J. Myers 著 有沢誠訳,「ソフトウェアの信頼性 ソフトウェア・エンジニアリング概説」 近代科学社, 1977

〔11〕 N. Wirth 著 野下浩平他訳「系統的プログラミング〈入門〉」近代科学社, 1975

〔12〕 J. K. Hughes and J. I. Michtom,"A Structured Approach to Programming", Prentice－Hall,1977

〔13〕 B. W. Kernighan,P. J. Plauger 著 木村泉訳 「プログラム書法」共立出版, 1976

〔14〕 和田英一他 「パネル討論会 構造的プログラミング 昭和50年度第16回大会報告」情報処理, 第17巻第11号, pp.1073－1087, 1976

〔15〕 O.－J. Dahl, B. Myhrhaug and K. Nygaard,"SIMULA information Common Base Language", Norwegian Computing Center (オスロ), 1970

〔16〕 K. Jensen and N. Wirth,"PASCAL, User Manual and Report", Lecture Notes in Computer Science 18, Springe－Verlag, 1974

〔17〕 B. Liskov 他, "Abstraction Mechanisms in CLU」, Communications of the CACM, 第20巻第8号, pp.564－576,1977

〔18〕 W, A. Wulf,R. L. London and M. shaw, "ABSTRACTION and VERIFICATION in ALPHARD: Introduction to Language and Methodology", Carnegie-Mellon University, 1976

〔19〕 P. J. Denning,"A Hard Look at Structured Programming", STRUCTURED PROGRAMMING (Infotech State of the Art Report), Infotech International, 1976年

〔20〕 V. R. Basila and A. J. Turner,"Iterative enhancement:A practical technique for Software development", First National Conf.on Software Engineering, (IEEE Computer Society), 1975

第9章　総　合　演　習

目　標

総合演習は各章で機能単位に学習したものの総集編であって，具体的ソフトウェアの作成を通して，一連のプログラミング作業を遂行するための基本的技術を習得させる。

また，ソフトウェアの作成に関しての新しい手法について実務レベルで評価ができるよう諸手法を体験させると同時に，その過程で発生する実務的問題の把握および，それらの問題に対処ができるようにする。

内　容

規模として2～5kの文からなるソフトウェアを実務のステップに合わせて，新しい手法により作成させる。

9.1　演習用システム

演習で使用されるコンピュータについて，演習で必要な最低限の知識を次の項目について習得させる。

(1)　コンピュータの構成

(2)　オペレーティング・システムの概要

(3)　ジョブ制御言語

(4)　デバッギング・ツール

(5)　操作法

(6)　記述言語（演習プログラムの）

記述言語は記述能力（ビット／文字操作，記憶域管理，構造化のし易さ等）から考えてPL／Iが望ましい。

PL／Iが使用できない場合は他の高水準言語で代替可能であるが，アセンブリ言語レベルのものは避けるべきである。

記述言語は，オペレーティング・システム等に依存した事項，言語の標準仕様からはみ出した部分，コンパイラの仕様に依存した事項について説明する。該当項目として以下のものが考えられる。

(1)　ファイルの編成および入出力手続

(2) 特殊入出力装置の入出力手続

(3) コンパイラの制限事項

演習は高水準言語でプログラミングされるので，ジョブ制御言語，操作法等は典形的な例（たとえばコンパイル，連けい編集，実行）について説明し，作成過程で必要に応じて必要な部分を説明してもよい。

9.2 ソフトウェアの作成

(1) 演習プログラム

演習で作成させるプログラムの内容は2～5kの文から構成される程度のもので受講者の身近にあり問題の概要が容易に把握でき，受講者自身で外部仕様の決定ができるものが望ましい。

この場合，すでに稼動しているプログラムについて行わせるものと新規の問題に関して行わせるものとの二通りが考えられる。

(a) 新規の場合

問題としては会話型言語インタープリタ（ミニBASIC程度），簡単な情報検索，テキスト・エディタなどがある。これらは目的が明確であり規模も手頃であると考えられる。

これらのソフトウェアは実用性を必ずしも重視しなくてよい。むしろ作成工数上の制限から，重要でない例外事項の処理の省略，制限事項等の導入は止むをえないであろう。

(b) 既存のソフトウェアの場合

問題としては，(a)に準ずるが，外部仕様が確定しているので問題の選択範囲も広くなること，および新しい文書化の手法等を採用して旧手法と比較できること等の利点がある。

この場合はソフトウェアの機能（利用者から見た仕様）を変えず，開発手法，文書化仕様，設計仕様は旧プログラムにこだわらず他の新しい手法で行なう。

(2) チーム編成

演習は3～5名を1チームとし，チーム単位で行う。

受講者の知識レベルは同一と考えられ，実務に於けるチーム編成とは異なるが，責任者，作業分担は明確にして行う。文書化の責任者，テストの責任者と別々の人間が行なってもよい。チームの運営は自主性にまかすべきであるが，指導者はその進捗状況を十分に把握し，適切な助言を行なえるようにしなければならない。

(3) ソフトウェアの作成

(a) 方針の確定

演習の目的は単にプログラム作成の経験を得るためだけでなく，作成に関する問題点の把握，またそれらの解決策の一つである新手法の習得であり，新手法の導入に際して発生が予

想される諸問題への対処の仕方，新手法を評価する能力を涵養することである。

その意味で設計に対する方針を事前に明確化し，それに基づいて作成することが必要である。

したがって開発法，文書化，コーディング，テストなどについてどの手法で行うかを決定周知させる。

チームの各メンバーの分担を決め，分担分に対し研究テーマを与え，終了後そのテーマに対して報告書を作成させることも必要であろう。

(b) 作成作業

前項に述べた通り，方針に基づき新手法を用いて一連の作業を実施する。

作成に当って作業別の工数，コンピュータ使用時間等は，手法の比較，評価のために記録して置くことが望ましい。

特に既存のソフトウェアに基づいて作成する場合は旧手法との比較データが採れるよう配慮が必要である。

9.3 報告書

演習終了後報告書を作成する。

報告書はチームで作成するものと個人で作成するとの二通りが考えられる。

(1) チームで作成するもの

演習に関する経過および，工程，製品に対する考察に関してまとめたものである。内容としては次のようなものになるであろう。

- 演習の概要
- ソフトウェアの概要および作成に関する方針
- 工程および製品の性能に関する諸データおよび考察
- 結言
- 添付資料……製品

(2) 個人で作成するもの

個人別に与えられた課題に対する考察である。内容は単に自分が演習の中で経験した範囲にとどまらず，参考文献等の調査を含め，広い視野のもとで考察するよう指導する。

上記(1)(2)については，発表会を開いて，指導者，受講者，その他関連した人の前で発表させることも有効である。

指導上の留意点

(1) 演習はチームで行わないと意味がない。すなわち，最近のソフトウェアは規模が大きく，個人の単なるプログラミング能力ではこれに対処できない。

したがって演習もチームで，共同作業の形で行ない，チーム内でのコミニュケーション等の重要性を認識させる。

また他人の書いたプログラムは一般に理解しにくく，他人のプログラムに接しその長所を見出す機会を作るのは実務では非常にむづかしいが，これを利用して，その機会を作るようにする。

(2) 演習ソフトウェアの作成方針は指導者が示しても良いが，むしろチーム内での討議により決めるべきである。

受講者が多く複数のチームに分かれる場合には方針が重複しないよう調整する必要がある。

(3) プログラム仕様書や流れ図が完成した時点でチームでレビューを行い，信頼性の向上と同時に他人のプログラムに接する機会を作る。

(4) 演習で作成された製品は，以降の教育での文書やコーディング標準の例として使用に耐えうるものであること。

(5) 演習に当ってメーカー提供のマニュアルはチームごとに1組ずつ用意する。

- 言語仕様書（演習プログラムの記述用）
- オペレーティング・システム（OS）
- ファイル管理（データ管理）
- サービス・プログラム

  連けい編集

  分類・併合

  デバッギング・トゥール

  テキスト・エディタ

  他ユーティリティ・プログラム

本書は，日本自転車振興会の機械工業振興資金による補助金の交付を受けて実施した「昭和 52 年度情報処理教育に関する調査研究等補助事業」の一環として作成したものです。

昭和53年3月発行

財団法人　日本情報処理開発協会
情報処理研修センター

〒105　東京都港区浜松町2丁目4番1号
（世界貿易センタービル　7階）

TEL　03（435）6511（代）

返 却 期 限 票

○ 最後にある日付があなたの返却期限です。
○ 遅れないように期限内に返却しましょう。
○ 続いて借りたいときは届け出てください。

返却期限票　　MARUZEN