■第２編　ソフトウェア応用編■

第１章　アルゴリズムに関すること

指導目標

第１編においても、第３章・第３節の中で計算アルゴリズムについて極入門程度のことを学習した。もちろん、問題の解を得るまでの過程を分析し、そこに至るまでの処理手順としてアルゴリズムを導入したのであるが、そこではコンピュータ表現に慣れることを最大の目的として、直感的で分かりやすい流れ図を対応させた。

ここではさらに複雑な問題をコンピュータ処理することになり、アルゴリズムを流れ図で表現するすることは同様におこなうが、幾通りもありうる問題の解法を比較する上でも、問題を抽象化してアルゴリズムを考え出す訓練を積むことにも重きを置くこととする。

アルゴリズムという言葉から、技術計算における数値処理や数値解法を思い浮かべるであろうが、本来は、問題をどう捉え、どう定式化すれば問題の本質や論理の組立が明確になるかを考えれば、ファイルを扱うような事務処理、文字ストリング（テキスト文字列）を扱うような文章処理にも応用できる。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
複数ファイル処理	・分類処理 ・併合処理 ・突合せ処理 ・更新処理 ・維持管理 について説明	↑ 特になし ↓	一般参考図書の問題、情報処理技術者試験の問題	ＰＣ，ＷＳ
配列処理の技法	・分類問題 ・探索問題 について説明		一般参考図書の問題、情報処理技術者試験の問題	ＰＣ，ＷＳ
文字列処理の技法	・テキストの編集 ・文字列照合 について説明		一般参考図書の問題、情報処理技術者試験の問題	ＰＣ，ＷＳ

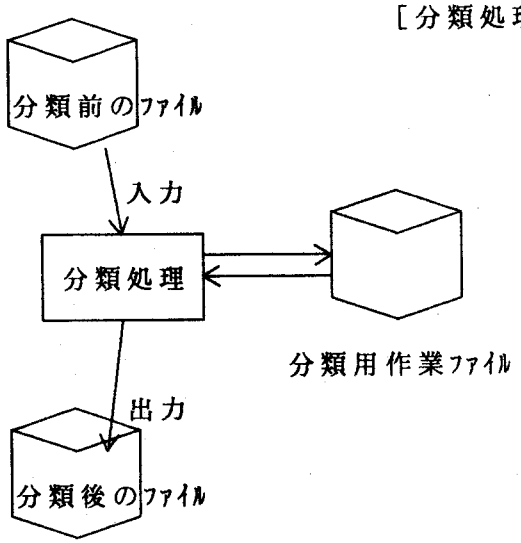
1．複数ファイル処理

第１編の第３章・第６節ではただ１種類のファイルを入力し、それを対象としたファイル処理について学習した。ここでは、複数個のファイルを入力し、それに対する処理論理について学ぶ。以下でまずファイル処理の基本的な作業について概要を整理する。

（１）ファイルに関する基本的な処理

①　分類処理（ソート：[Sort]）　〔整列処理ともいう〕

［分類処理］　指定された特定のキー項目に関して一定の順序でレコードを並べ替えることを分類という。



社員#	氏名	給料
1572	高田	250
1346	目黒	600
1448	佐伯	480
1399	石田	570

＝＝＞ 社員#順

1346	目黒	600
1399	石田	570
1448	佐伯	480
1572	高田	250

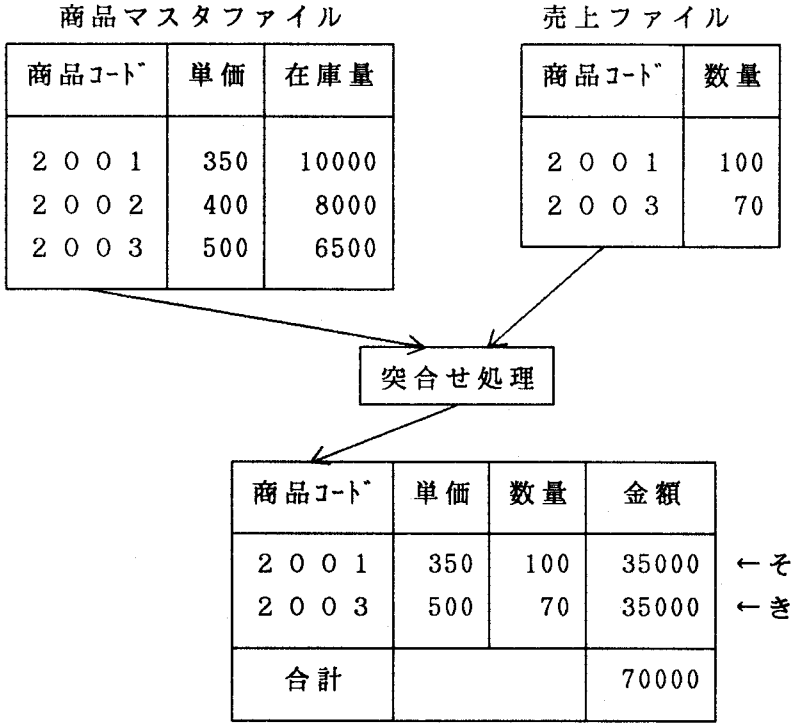
②　突合せ処理（マッチング：[matching]）

ファイル同志をキー項目によって突合せを行い、キーの合致性に従って各種の処理を行うことを“突合せ処理”という。突合せされる複数のファイルはそれぞれ１つ以上のキー項目を持っており、また各ファイルのレコードキー項目は昇順、降順いずれかで分類されている必要がある。

突合せは、どのようなファイルをどのような形態で処理するかにより処理が分類される。

(a) 併合処理
(b) 更新処理
(C) 維持処理
(d) 突合せ処理

マッチングの例として、商品マスタファイルの商品コードと売上トランザクションファイルの商品コードを突合せて売上金額を計算する処理がある。



商品ｺｰﾄﾞ	単価	在庫量
２００１	350	10000
２００２	400	8000
２００３	500	6500

商品ｺｰﾄﾞ	数量
２００１	100
２００３	70

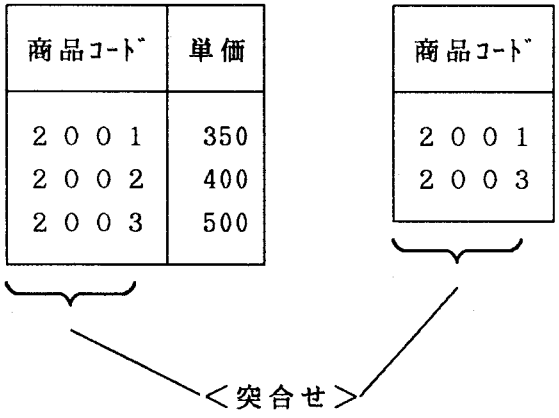
（注）商品マスタファイルの在庫量を更新する必要はあるが、ここではそれについては言及しない。

商品ｺｰﾄﾞ	単価	数量	金額
２００１	350	100	35000
２００３	500	70	35000
合計			70000

←それぞれは、請求書として発行で
←きる

● １対１の突合せ処理

突合せを行う各ファイルのキー項目が唯一（同じ値のレコードが他にない）の場合、“１対１の突合せ”という。

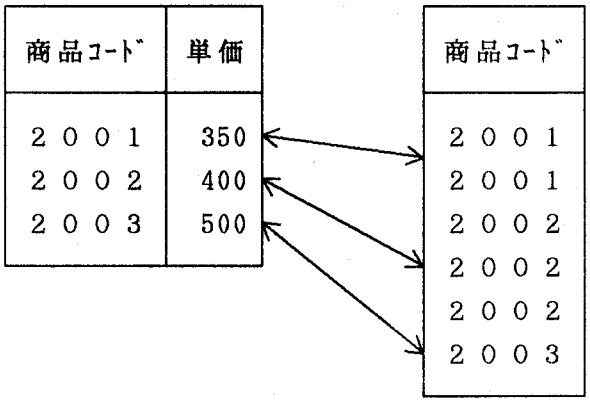


商品ｺｰﾄﾞ	単価
２００１	350
２００２	400
２００３	500

商品ｺｰﾄﾞ
２００１
２００３

● １対Ｎの突合せ処理

一方のファイルにキー項目が唯一のレコードがあり、他方のファイルに複数個の同一キーを持つレコードがある場合の突合せを、１対Ｎの突合せという。

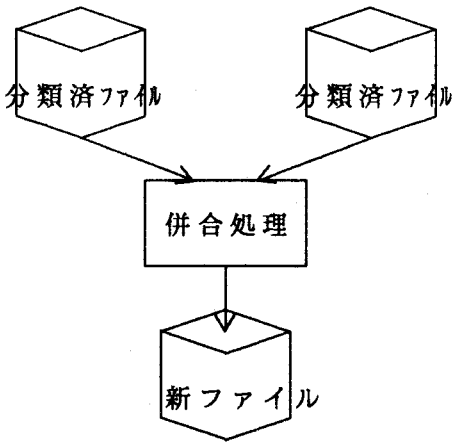


商品ｺｰﾄﾞ	単価
２００１	350
２００２	400
２００３	500

商品ｺｰﾄﾞ
２００１
２００１
２００２
２００２
２００２
２００３

③　併合処理（マージ：［merge］）

［併合処理］　指定された特定のキー項目に関して分類済の複数個のファイルのレコードをまとめて組合せ１つのファイルにする。



売上マスタ1

商品ｺｰﾄﾞ	出荷量
２００２	1000
２００３	500

売上マスタ2

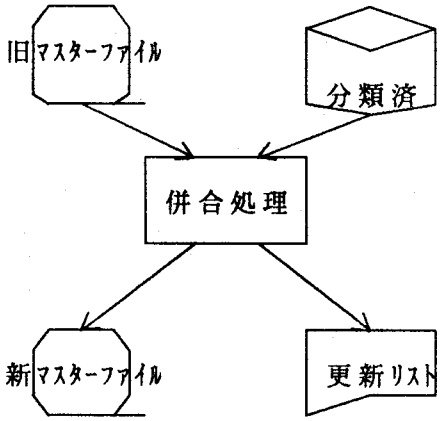
商品ｺｰﾄﾞ	出荷量
２００１	1200
２００２	300
２００４	900

＝＝＝＞ 商品ｺｰﾄﾞで併合

商品ｺｰﾄﾞ	出荷量
２００１	1200
２００２	1000
２００２	300
２００３	500
２００４	900

④　（マスタ）更新処理（アップデート：［update］）

［更新処理］　マスターファイルにトランザクションファイルを適用し、レコードの追加、修正、削除を行い、新マスターファイルを作る。



旧在庫ﾏｽﾀｰﾌｧｲﾙ

商品ｺｰﾄﾞ	在庫量
２０００	15000
２００２	12000
２００３	250
２０１１	13000
２０２０	20000

ﾄﾗﾝｻﾞｸｼｮﾝﾌｧｲﾙ

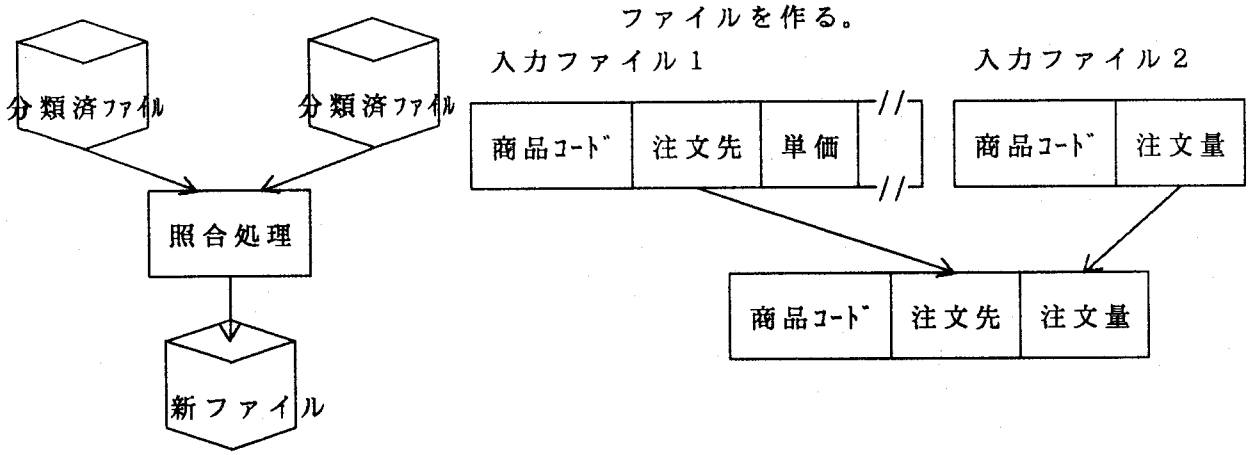
商品ｺｰﾄﾞ	区	数量
２０００	1	300
２０００	1	150
２００１	2	10000
２００３	3	0
２０２０	0	500

０：入庫
１：出庫
２：新製品
３：廃棄品

商品ｺｰﾄﾞ	在庫量
２０００	15450
２００１	10000
２００２	12000
２０１１	13000
２０２０	20500

⑤　照合処理（コレーティング：［collating)］）

［照合処理］　複数のファイルの間で同一なキー項目のレコードを組み合わせたり、仕分けをして新しいファイルを作る。



⑥　ファイルの維持

トランザクション情報でマスタファイルのレコードの修正、追加、削除を行う点で更新と同じであるが、更新では変化の状況をマスタファイルに反映するのに対し、維持管理では必要なデータがファイル内に記録されていることを確認する。ファイルを常に最新の状態に保つことが目的である

広義には、ファイルの更新途中などに障害が発生したとき、旧ファイルを使って正しいファイルを回復することもファイルの維持に含まれる。

⑦　データチェック

データに誤りがあり、それが発見されずに計算処理が進められると、単に一度限りの計算間違いでは済まず、永久的に誤りが残るかあるいはどんどんと他の計算処理にも大きな悪影響を及ぼしていく恐れがある。こうなるとその計算処理の意味がなくなるだけでなく、社会的な脅威に発展し始める。

このチェックについては色々な検査の視点から行わなければならない。まず、データの入力の時点では、伝票の発行、キー・ツー・フロッピー等人手がからむことが多く、厳重なチェックが必要である。また、データ処理の時点においても、同一データが作られたり、値がオーバフローしたりすることに対するチェックも必要である。さらに、データの出力時点においてもバランスシートなどで論理的なミスも発見されうる。

これらのデータチェックは、それが必須な時点で、計算目的に合わせて必要な種類を組み合わせて行わなければならない。

(a) データ発生時点チェック

チェック名称	チェックの方法
サイトチェック (Sight Check) (目視検査)	原始伝票における、記入ミス、記入洩れ。伝票とファイルのプリントとを見比べる。また、穿孔カードや穿孔テープの内容を印字してその内容を読合わせる。
ベリファイチェック (Verification) (検孔検査)	穿孔カードや穿孔テープ、キー・ツー・フロッピー等のデータを２度同じことを行い、両者が一致しているかどうかチェックする。

(b) データ入力時点チェック

チェック名称	チェックの方法
ニューメリックチェック (Numeric Check) (数値検査)	数値フィールドに数値以外のものが入っていないかどうかチェックする。

チェック名称	チ　エ　ッ　ク　の　方　法
ｱﾙﾌｧﾍﾞﾃｨｯｸﾁｪｯｸ Alphabetic Check （英字検査）	英字フィールドに英字以外のものが入っていないかどうかチェックする。
リミットチェック (Limit Check) （限界検査）	数値がとりうるべき値の範囲にあるかどうかチェックする。例えば、月日として１～１２、１～３１以内かどうか。
シーケンスﾁｪｯｸ (Sequence Check) （順序検査）	注文伝票のように、データが一連のキー項目順に並んでいるかどうかをチェックする。重複や紛失も発見できる。 順番検査ともいう。
フォーマットﾁｪｯｸ (Format Check) （書式検査）	指定された形式通りに入力データ（文字種、項目桁数など）が作られているかどうかをチェックする。例えば桁ずれがあると、入力フィールドの間違いにつながる。
カウントチェック (Count Check) （件数検査）	入力件数と原票件数を比べ、正しい関係にあるかどうかをチェックする。原票の入力洩れ、重複も分かる。
バリディティﾁｪｯｸ (Validity Check) （妥当性検査）	データ項目の内容が論理的に妥当であるか、処理に不適切なデータ値になっていないかをチェックする。

(c) データ処理（計算処理）時点チェック

チェック名称	チ　エ　ッ　ク　の　方　法
ﾀﾞﾌﾞﾙﾚｺｰﾄﾞﾁｪｯｸ Double Record Check（重複検査）	同じ内容のレコードが２件以上作られていないかチェックする。 二重検査ともいう。

チェック名称	チ　ェ　ッ　ク　の　方　法
サインチェック (Sign Check) （符号検査）	データに対する演算の結果、値の符号が明かな場合、当該項目の符号が正しいかどうかをチェックする。
オーバフローﾁｪｯｸ (Overflow Check) （桁あふれ検査）	データに対する演算の結果、その値が指定の桁数よりあふれがないかどうかをチェックする。
マッチチェック (Match Check) （照合検査）	例えばトランザクションレコードが、マスタファイルに存在するかどうか照合を行う。登録済みかどうかチェックできる。

(d) 出力時点チェック

チェック名称	チ　ェ　ッ　ク　の　方　法
バランスチェック (Balance Check) （平衡検査）	簿記の貸借対照表の貸し方金額合計と借り方金額の合計は一致しなければならない。このように、対になる項目の値や合計が一致しているかどうかをチェックする。
クロスフットﾁｪｯｸ Cross Foot Check （交差合計検査）	集計表のように縦合計と横合計が一致しているかどうかをチェックする。
ﾊﾞｯﾁﾄｰﾀﾙﾁｪｯｸ Batch Total Check	入力データの特定の項目の合計値を事前に計算しておき、入力データとともにコンピュータに入れておく。項目としては、数量、金額等総合計ということで何らかの意味を持つものを対象とする。データ処理において、コンピュータでもこの合計値を計算し、手計算等により求めた値と比較し、一致するかどうかをチェックする。 バッチおよびハッシュのトータルチェックを“集計検査”という

チェック名称	チ　ェ　ッ　ク　の　方　法
ﾊｯｼｭﾄｰﾀﾙﾁｪｯｸ Hash total Check	バッチトータルチェックとチェックの原理は同じであるが、集計項目の選び方が異なる。 ハッシュトータルでの場合、項目として、日付、商品コード、単価、社員番号などのように合計を求めても何の意味も持たないものを選ぶ。

(e) その他のチェック

チェック名称	チ　ェ　ッ　ク　の　方　法
ﾘﾀﾞﾝﾀﾞﾝｼｰﾁｪｯｸ (Redundancy Check)　(冗長検査)	データに、ある規則に従って作られた文字やビット等の冗長なデータを付加して入力する。コンピュータでも同じ規則により検算を行い、データの誤りをチェックする。
パリティチェック (Parity Check) (奇偶検査)	データ通信などにおいてデータを伝送する場合に、伝送エラーを発見し易いようにする操作。転送単位に1ビット分つけ加え、全ビットの“1”のビットの個数が奇数または偶数になるように、そのビットの値を決める。コンピュータに入力されたとき、途中で転送誤りがなかったかどうかをチェックする。
ﾁｪｯｸﾃﾞｨｼﾞｯﾄﾁｪｯｸ (Check Digit Check) (検査数字検査)	コードの先頭や末尾に、コードの各桁の数字に対して加工（ある計算処理）をして作成した検査数字を付加する。コンピュータに入力されたとき同じ検査数字を作成して、入力された検査数字を比較して誤りがなかったかどうかをチェックする。 チェックディジットは、全コードの和に対してモデュラスｎを使って求める方法が一般的である。

2．配列処理の技法

配列に対する基本的な操作として、参照、探索、挿入、削除、分類等がある。以下で、主な操作について解説をする。

（１）分類（整列またはソート）問題

ファイルのレコードを指定キー項目に従って、小さい順（昇順［ascending order］）、大きい順（降順［descending order］）に並べかえる作業を分類と呼ぶ。勿論データは必ずしもファイルにある必要はない。この順番としては、アルファベット、日本語の五十音順などの並べ替えもありうるが、基本はキー項目の計算機内部での表現値による大小である。

また、レコード中のキー項目は一種類だけでなく、複数個のキー候補が入っており、分類する時点で特定することになる。意味的には、従業員ファイルなら、社員番号順に並べるとか、氏名の五十音順に並べる、あるいは年令順に並べるなどが考えられる。

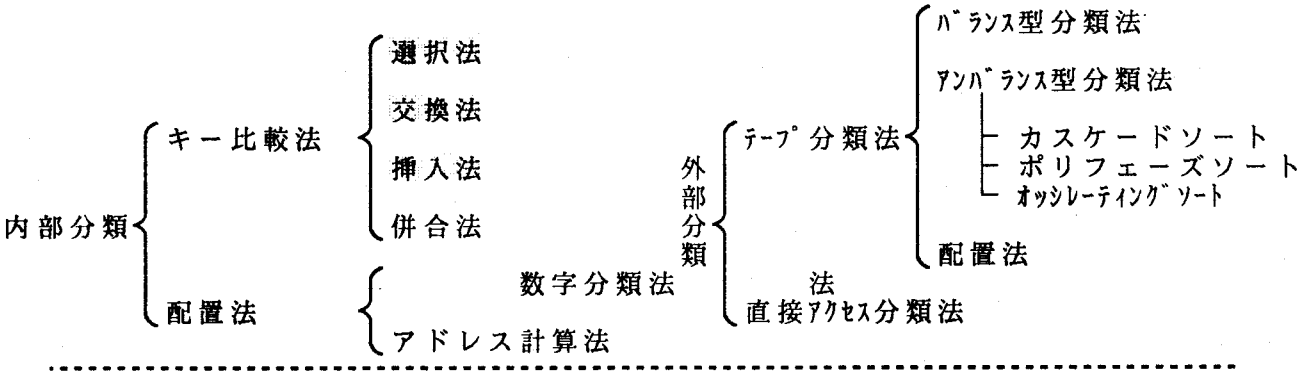
分類はバッチ処理の中で最も重要な手法の一つであり、高速性、メモリの有効な利用性を実現できるアルゴリズムが望まれ、現在までに各種の方法が研究されてきた。

よく用いられる手法として、交換法、挿入法、選択法がある。また、データの記憶場所により、それにふさわしい分類の方法もある。例えば、主記憶の配列に置かれているような場合内部分類法を用いる。但し、当然分類の対象となるレコード数には制限が設けられる。

ここで極簡単に、分類法の特徴とよくプログラミングで用いられる技法についてまとめておく。ここでは分類の対象となるデー多量の多少による内部分類とが外部分類について記述する。

- 内部分類法　－　主記憶域内に展開されたデータをキー項目順に並べる
- 外部分類法　－　補助記憶装置を用いて、多量のデータを分類する

下記に、内部分類、および外部分類を図示する。



前頁の分類において、内部分類では［選択法、交換法、挿入法］についてはよく理解しておく必要がある。第２種情報処理技術者試験にもしばしば出題される。その他の方法については、考え方について知っておくと良い。

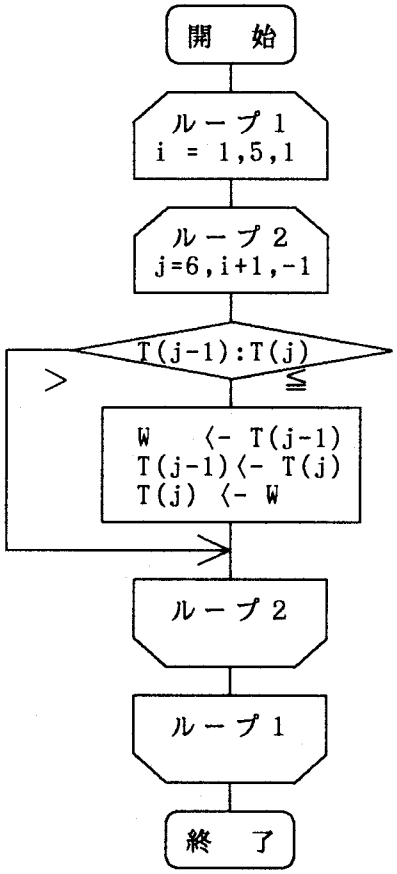
例えば、内部分類法に属すアドレス計算法は、キーの値に対してある計算上の操作を適用してデータのアロケーションを決めるアルゴリズムである。計算されたアドレスが空いていればそこにデータを格納し、空いていなければその近傍に格納先を決定する。

また、外部分類に属すカスケードソート法では、Ｎ本のテープを１本にまとめる方法である。テープがＮ＋１本あると１パスはＮウェイマージ、Ｎ－１ウェイマージ、・・・、１ウェイマージを行い、１つになるまでこのパスを続ける。

以下で、内部分類の代表的な方法に関して例示する。

① 交換法（Exchange sort）

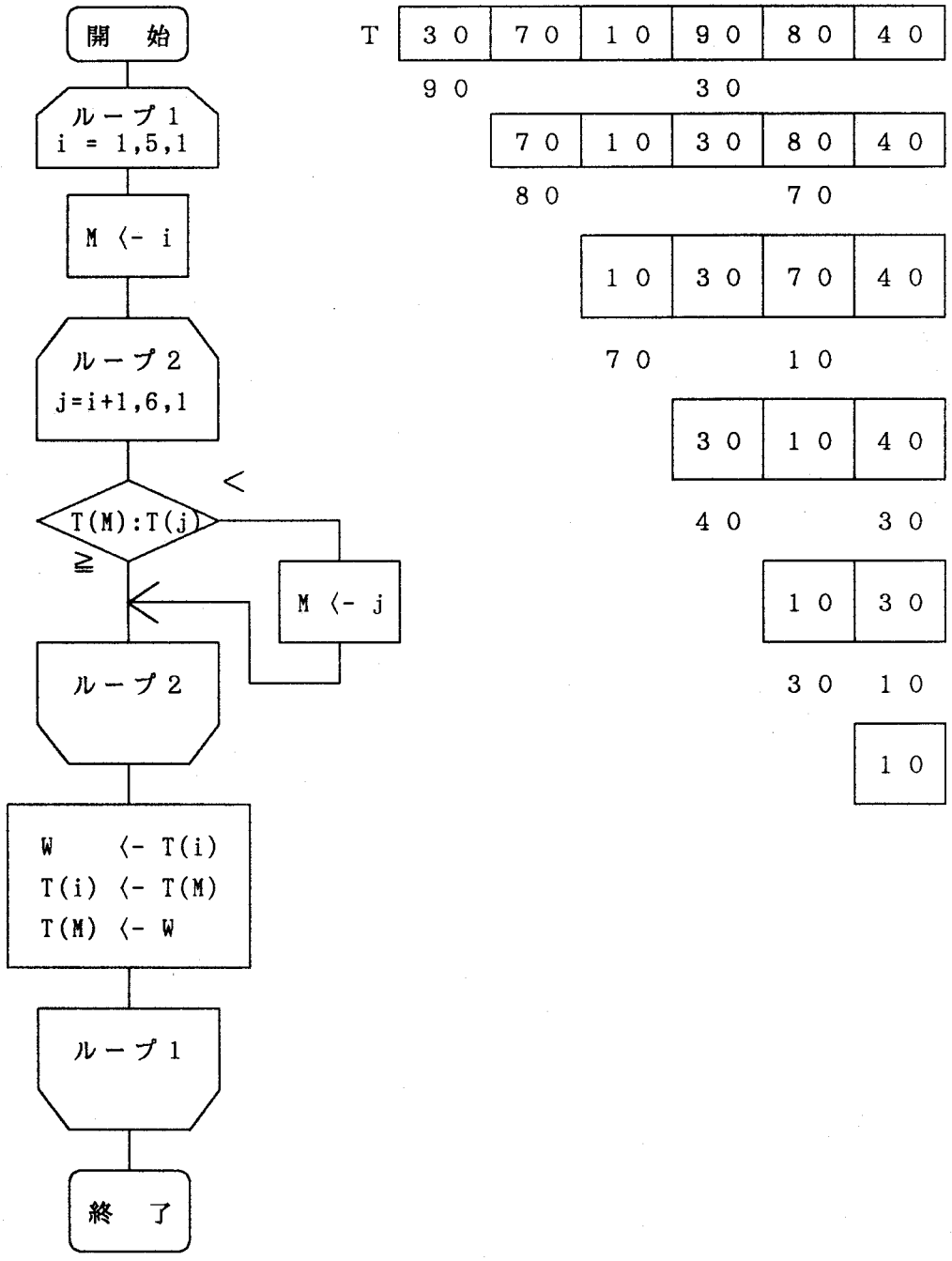
隣合う二つのレコードのキーを比較し、順序が逆であれば交換を行うことを繰り返しつつ分類する方法である。



T						
	70	80	90	10	30	40
					40	30
				40	10	30
			90	40	10	30
		90	80	40	10	30
	90	70	80	40	10	30
					30	10
				40	30	10
			80	40	30	10
		80	70	40	30	10
	90	80	70	40	30	10
	90	80	70	40	30	10

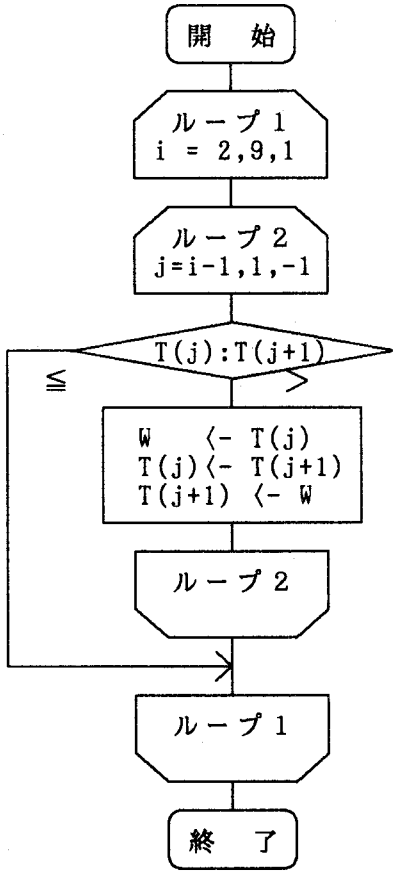
② 選択法（Selection sort）

レコードのキーの最小または最大なものから順に取り出す方法。



③　挿入法（Insertion sort）

挿入とは、配列に並んでいる一連の配列要素の途中に、いくつかのデータを挟み込むような操作を行うことを意味している。この方法はキー項目の値が順番として適切な位置にそのデータを挿入していく方法である。



T	24	02	16	88	79	19	35	09	51

24								
02	24							
02	16	24						
02	16	24	88					
02	16	24	79	88				
02	16	19	24	79	88			
02	16	19	24	35	79	88		
02	09	16	19	24	35	79	88	
02	09	16	19	24	35	51	79	88

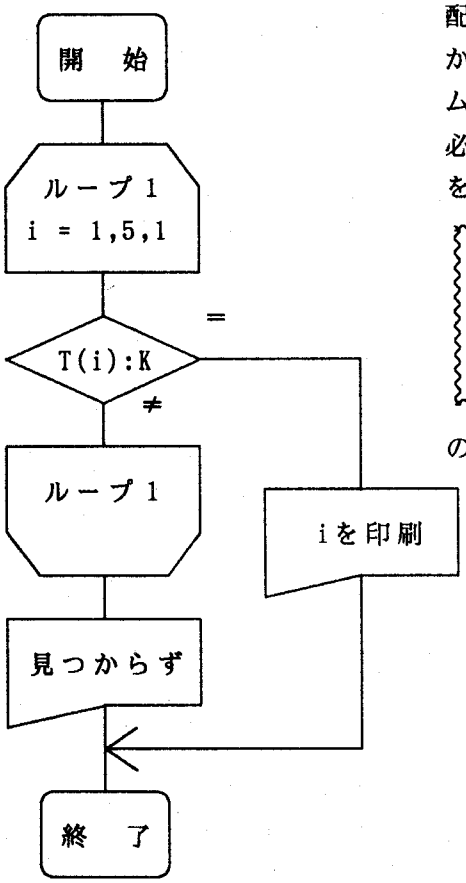
この方法では、挿入すべき位置よりも後に入っている要素は後ろにずらされる。

（２）探索（サーチ）問題

ファイルや配列などに格納されている大量のデータレコードの中から、指定キー項目を持つデータを探し出すことを探索という。探索の効率は、目的データを探すためにデータレコードを何回照合したかによって決まる。

代表的な探索の方法としては、データを順次調べ条件を満たすデータを探し出す“逐次探索法”（または“順次探索法”）と、ファイルや配列を中央で二つに分割し、前半か後半のどちらかを探す操作を繰り返して探す“２分探索法”がある。

①　逐次探索（sequential search）



配列ＴをＴ（１）から順番に見ながら、見つかったらこの操作を終わる。データがランダムに入っている場合この方法を用いる。
必ず見つかるものと仮定すると、Ｎ個の要素を持つ配列を探すような場合、

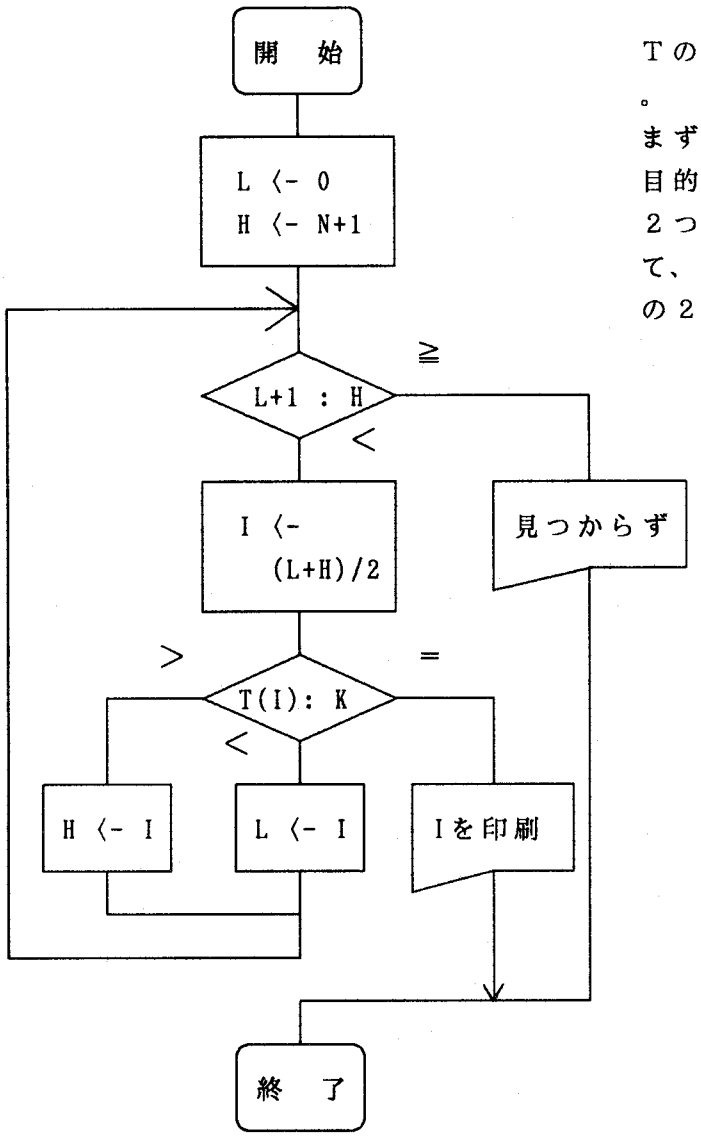
平均：　（Ｎ＋１）／２回

最大：　Ｎ　回

の繰り返しで見つけられる。

② 2分探索法（Binary search）

ファイルや配列の中のデータが特定キー項目の大きさの順に並んでいるときは、2分探索法が有効である。



Tの要素は小さい順に並んでいるものとする。

まず、この配列の真ん中の要素と比較する。目的データがこの配列要素より大きければ、2つに分割したうち、大きい方の部分について、小さければ小さい方の部分について同様の2分割と比較を行う。

このプログラムで、N個の要素に対する探索では、

$[\log_2 N]$ をKと置くと、比較回数の

平均： $(K+2-2^{K+1})/N+K+1$

最大： $(\log_2 N)+1$

の繰り返しで見つけられる。

3．文字列処理の技法

文字列の処理は、文字の集まりとしての“文字列”や文字そのもの（１文字分）に対する演算を施すことである。一般に高水準言語では文字や文字列を扱うことが得意な言語と、得意でないものとがある。

ＦＯＲＴＲＡＮは技術計算処理向きに作られた経緯もあり、文字列の扱い能力は低い。ＣＯＢＯＬでは文字列はしばしばデータ処理の対象とはなるが、やはりあまり凝った操作はできない。

これに対して、ＰＬ／ＩやＣ言語では、コンパイラや各種言語プロセッサの記述言語としてしばしば用いられることもあり、文字の扱いは得意な方である。

一般に、文字の操作としては、高水準言語の機能として持っているものとして、
・文字や文字列の代入
・文字列の連結
・文字や文字列の値としての大小、等値比較
等があり、また文字列処理用ライブラリ関数として、文字列の長さの計算、文字列の探索等色々と用意されている。

（１）テキストの編集

①　テキストの行揃え処理

事例：文字列の中に入れ子構造になりえる開き括弧’（’と閉じ括弧’）’、（と）間には１６文字の文字列［空白を含まない連続する文字］がある。
この文字列を桁を揃えてプリントしたいが、
・開き括弧が出てくるたびにインデント（１６文字分右寄せ）して改行する
・途中に’｜’の文字が現れたら、開き括弧のインデントの位置に復帰する
ようにしたい。プリントの処理は、元の文字列のあるバッファの終わりに到達したら終わりにする。なお、最初の非空白文字は“（”、最後の非空白文字は“）”であるものとする。

（例１）　文字列　ＢＦ

```
  (  A1   (  B1
 (C1   (D0)
  C2  )   B2)
A2  )
```

＝＝＝＝＞

プリント結果

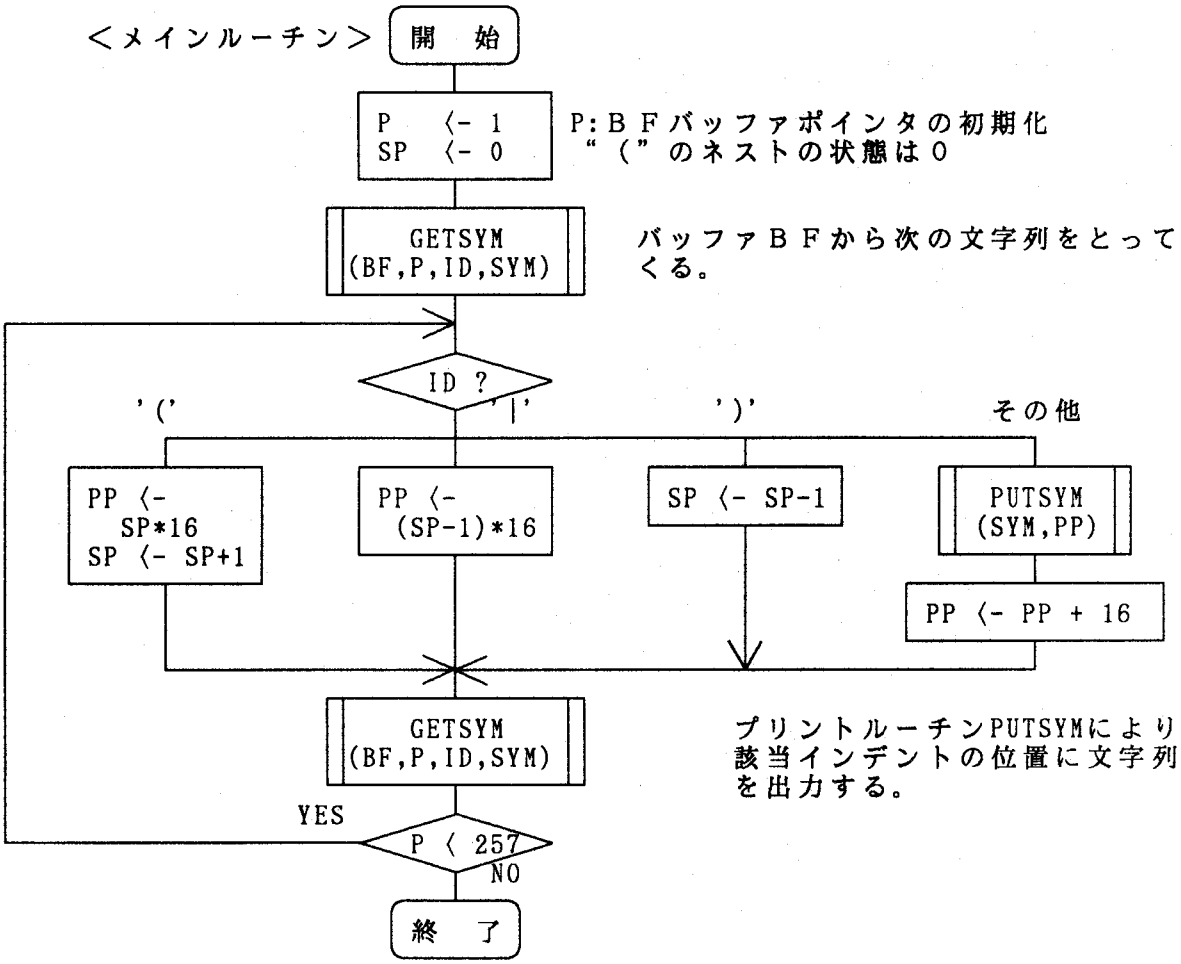
```
A1
    B1
        C1
            D0
        C2
    B2
A2
```

（注）上記Ａ１～Ｄ０は１６文字の文字列であるものとする

（例２）　文字列　　プリント結果

```
   (A1   A1   (          A1   A1
    B1   B1                    B1   B1
  )   |   A2   A2   ====>  A2   A2
   (   C1   C1                C1   C1
|   D1   D1   )             D1   D1
```

処理：文字列バッファの大きさをＢＦ（２５６）とする。
ＢＦ内の次の文字位置を示すポインタをＰ、その位置の文字が区切り［（，｜，）］でなければ、それに続く１６個の文字列をＳＹＭ（１６）に入れる。また、プリントする位置をＰＰに、“（”の深さをＳＰに持つこととする。

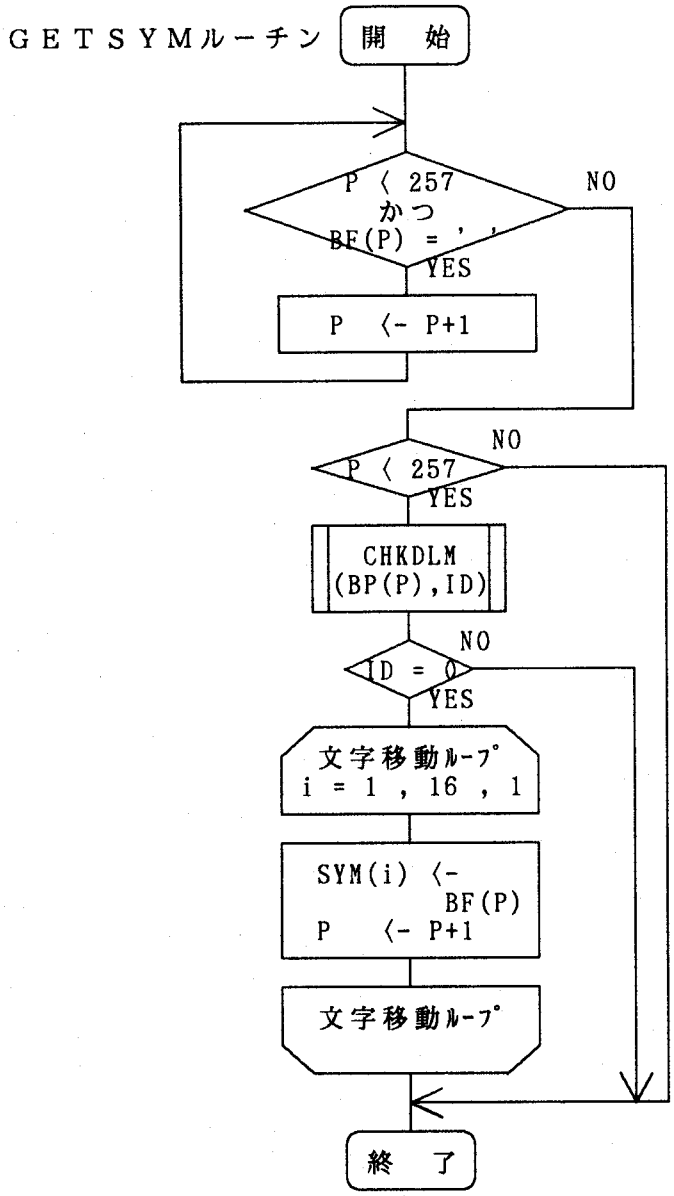


［サブルーチンＧＥＴＳＹＭの処理］

(1) 空白以外の文字が得られるまで、バッファポインタＰを進める。（バッファオーバーとなったらリターンする）

(2) その文字が、［（，｜，）］のいずれかでなければ、１６文字の文字列として取り出し、ＳＹＭ（１６）に移してリターンする。

(3) ［（，｜，）］のいずれかであればＩＤに［１，２，３］をセットしてリターンする。



文字が［（，｜，）］に対応し、ＩＤに［１，２，３］が返される。

② テキストの置き換え　　インクルード

事例：文字列Ａの中にときどき“＃”マークが現れる。これに出会ったときは、別の文字列Ｂから８文字分の部分文字列を取り出し“＃”をそれで置き換える。元の内容および置き換えた結果をバッファＳに順次格納する。文字列Ａの終わりにきたら、バッファＳの内容をプリントする。

（例１）　文字列　Ａ

```
ＸＸＸ　＃　ＹＹＹ
ＺＺＺ　＃　ＳＳＳ
＃　＃　ＴＴＴ
```

文字列　Ｂ

```
ＡＡＡ　ＢＢＢ
ＣＣＣ　ＤＤＤ
```

＝＝＝＝＞

バッファＳのプリント結果

```
ＸＸＸ　ＡＡＡ　ＹＹＹ
ＺＺＺ　ＢＢＢ　ＳＳＳ
ＣＣＣ　ＤＤＤ　ＴＴＴ
```

処理：文字列バッファの大きさをＡ（１２８）、Ｂ（６４）、Ｓ（２５６）とする。
バッファポインタをそれぞれＰＡ，ＰＢ，ＰＣとする。
Ａのバッファの最後にきたら、Ｂに残りの文字があってもそこで処理を終える。

（２）文字テキストの圧縮

事例：文字型の配列S(128)の中の文字列を、以下の規則により他の文字列C(128)に移し、それをプリントする。（配列Ｓの上限内に文字列の終端があれば、それは判定できるものとする）

(1) 同じ文字が３個以上連続している場合（空白文字もまた文字とみなす）、連続する旨を示すエスケープコード（’＃’を用いる）とその個数及び当該文字に置き換える。
例：ＡＡＡＡ　→　＃４Ａ　［４は８ビットのバイナリ値である］

(2) 文字列中に“／＊　・・・・・・　＊／”の文字列があらわれたら、その部分は捨てる。

（例）　文字配列　S

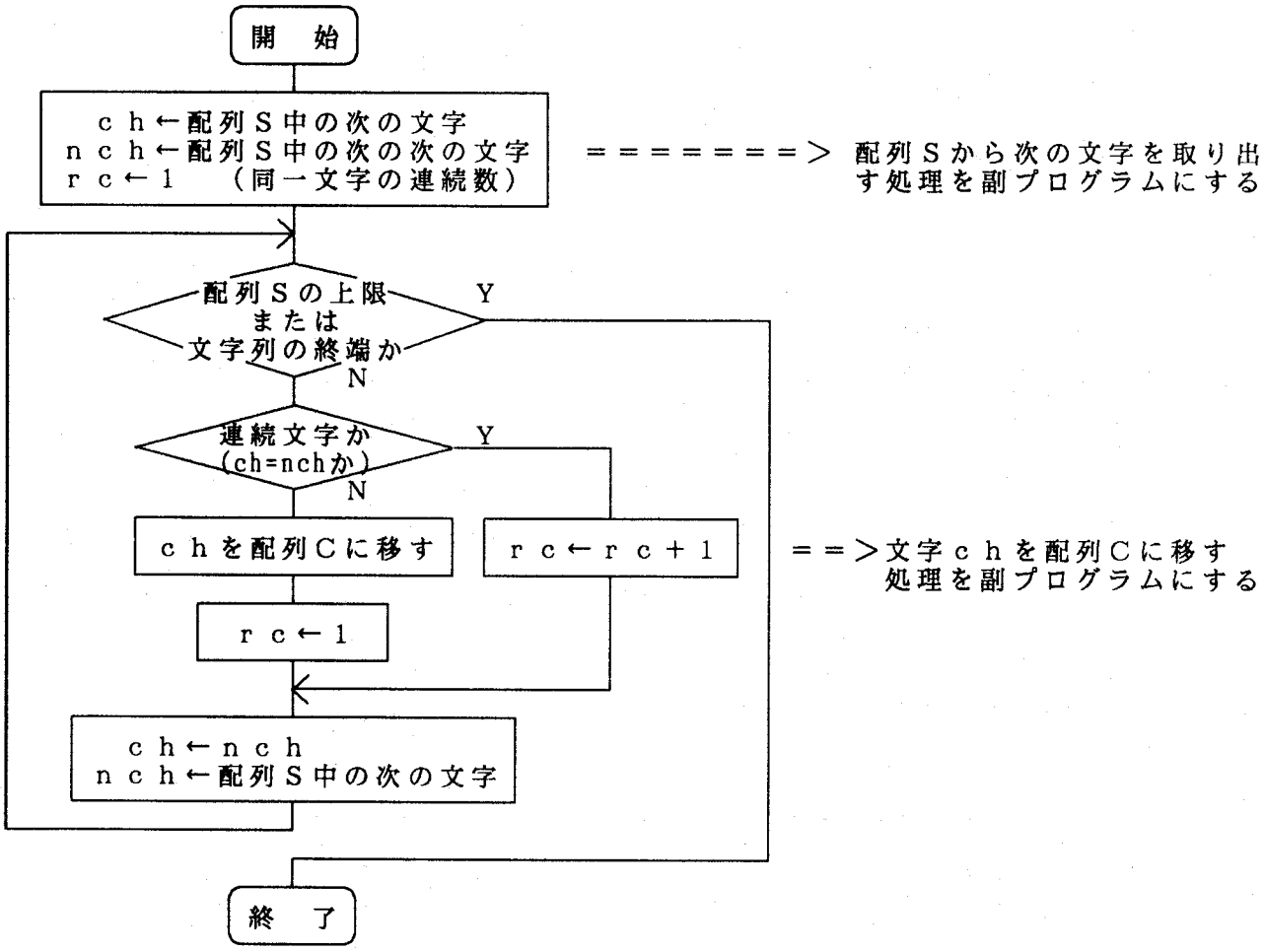
A				B	C	D	X	／	*		?	?	?		*	／	X	X		D		／	*
	?	／	*		%		*	／	?		*	／		E	E	E	Y	F	F	F	G	／	*
	?	?		*	／	／	*		?	?	?		*	／	A	A	／	*		?		*	／
A	←文	字	列	の	終	わ	り																

↓

文字配列　C　（圧縮後の文字列：数値は文字でなくバイナリ値）

A	#	3		B	C	D	#	3	X		D		?		*	／		#	3	E	Y	#	3
F	G	#	3	A	←文	字	列	の	終	わ	り												

処理：



[Cバージョン]

```
#define NUM     128
main() {
   char getch();
   char ch,nch;
   char *s = " 原始文字列 ";
   char c[NUM+1];
   int  repcnt, /* 連続文字数    */
        i,      /* 配列S用添字 */
        j;      /* 配列C用添字 */

   for ( j=0; j<=NUM; c[j++]='¥0');      /* コピー先配列Cをnullクリアする */
   repcnt=1, i=j=0;                      /* 連続文字数、配列添字の初期化   */
   ch=getch(s,&i); nch=getch(s,&i);      /* ２つの隣接する文字を取り出す   */

   while ( i<=NUM && ch!='¥0' )          /* 配列Sの上限内かつ文字列の終端以内 */
   {
      if ( ch==nch )                     /* 連続する同一文字数をカウントアップ */
         ++repcnt;
      else
      {
         chrcpy(repcnt,c,&ch,&j);        /* ２つの文字が異なるとき最初の文字を
         repcnt = 1;                        配列Cにコピーする               */
      }

     ch  = nch;                          /* 隣接位置を１つ進める */
     nch = getch(s,&i);
   };

   printf("%s ¥n",c);
}
```

```
/*
    コメントであればそれを捨てる
    配列の最新の位置の文字を返す
 */
char   getch(s,i)
       char s[];
       int  *i;
{
       char c;

       do
       {
          if ((s[*i]=='/') &&
              (s[*i+1]=='*'))
          {
                *i += 2;
             for (                    ;
                 !((s[*i]  =='*')&&
                   (s[*i+1]=='/'))  ;
                 ++*i               );
                 *i += 2;
          }
          else
            break;
       } while ( (c=s[*i])!='¥0');

       if ((c=s[*i])!='¥0')
          ++*i;
       return(c);
}
```

```
/*
    文字をコピーする
    同一文字が３文字以上連続する場合は
      それを圧縮する
 */
shrcpy(rep,c,ch,j)
       int  *rep,  /* 連続文字数 */
            *j;    /* ｺﾋﾟｰ先配列添字 */
       char c[],   /* コピー先配列 */
            *ch;   /* コピー文字 */
{
   switch (*rep)
   {
      case 1: c[*j]=*ch; ++*j;
              break;
      case 2: c[*j]=*ch; ++*j;
              c[*j]=*ch; ++*j;
              break;
      default:c[*j]='#'; ++*j;
              c[*j]=(char)*rep;
                    ++*j;
              c[*j]=*ch; ++*j;
   }
}
```

指導上の留意点

この章の指導内容は、第２種情報処理技術者試験では、午後の部の問題と非常に関連が深い。最近数年間の問題は非常に高度化しており、集中して十分な時間をかけて学習しておかないと問題を解くことができない。

ただし、選択性となっており、技術計算からみたアルゴリズムと、事務処理からみたアルゴリズムとの２つのアプローチをとっている。どちらかを選択すればよく、コンピュータカレッジの学習方針に沿って、あるいは生徒の特性に併せてどちらかの方法で進めることが望ましい。

２つのアプローチがある反面、ＦＯＲＴＲＡＮ，ＰＬ／Ｉ，ＣＯＢＯＬのどの言語を選択しても点差が大きくならないよう類似した問題が出題されている。言語選択の如何に関わらず、重点学習が必要な項目として以下がある。

○　表操作、探索、ハッシュ、ソート
○　２分木構造、リスト構造、スタック構造

これらについて、本書では十分扱っていないため、他のプログラミング授業などで十分時間を割いていただきたい。

また、最近はＣ言語の普及もあり、文字列処理が増えていることに注目する必要がある。

用　語

分類、整列、ソート、突合せ、マッチング、併合、マージ、
マスタファイル、トランザクションファイル（取引ファイル）、更新、
照合、コレーティング、
データチェック、サイトチェック、目視検査、ベリファイチェック、検孔検査、
ニューメリックチェック、数値検査、アルファベティックチェック、英字検査、
リミットチェック、限界検査、シーケンスチェック、順序検査、
フォーマットチェック、書式検査、カウントチェック、件数検査、
バリディティチェック、妥当性検査、ダブルレコードチェック、重複検査、
サインチェック、符号検査、オーバフローチェック、桁あふれ検査、
マッチチェック、照合検査、バランスチェック、平衡検査、
クロスフットチェック、交差合計検査、バッチトータルチェック、
ハッシュトータルチェック、リダンダンシーチェック、冗長検査、
パリティチェック、奇遇検査、チェックディジットチェック、検査数字検査、

内部分類法、外部分類法、交換分類法、交換法、選択法、挿入法、
探索、２分探索法、バイナリサーチ、

第２種情報処理技術者試験

○　アルゴリズム（算法）に関すること
　－表操作
　－突合せ
　－整列（分類）

第２章　プログラミングに関連する事項

指導目標

企業における実務レベルでの良いプログラミングの条件としては、ユーザの仕様を満足している（機能性）、正しい結果を得られる（信頼性）、効率がよい、など品質の高さの他に、開発においては、テストが容易にできる（テスト容易性）、解読し易い（解読容易性）、拡張性や柔軟性が高く保守がし易い、他機種への移植も容易である、等々が挙げられる。また、開発時に生産性が高く、再利用性も高いことなども重要な点である。

本指導書による養成の対象者に対しては、大規模であったり、長い年月利用されるようなプログラムを書くことはない。当養成施設における良いプログラムとしては、最小限次のようなものであろう。

○　簡潔なプログラムであること　（適切なモジュール化）
○　他人が読み易いプログラムであること（シンプルな構造）
○　他人にも参考となるようなテクニックを用いていること（適度な技巧）
○　テストが充分されていること

上記基準を持ってしても、良いプログラムかどうかなかなか分からないものではあるが、大体次のような心がけをすれば次第に実力が向上するであろう。

○　できるだけコーディングステップ数が短いこと
○　テスト期間が適度なものであること
○　見て美しい構造化されたプログラミングであること
○　重要な箇所にはコメント文が存在すること
○　変数名の名前の付け方が、プログラムの目的に沿っていること

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
プログラムテスト法	・ウォークスルー ・インスペクション ・ブラックボックステスト ・ホワイトボックステスト	特になし	特になし	
構造化プログラミング	構造化プログラミングの意義、構造化プログラミングの基本、構造化プログラミングの必要性について理解させる。	生徒同志で互いのプログラミングについて評価し合う中で構造化プログラミングの重要性を議論する。		汎用コンピュータ ＰＣ，ＷＳ

1．プログラムテスト技法

プログラムテストの基本的な技法については既に第１編で述べているので、ここでは、実用的なプログラム開発において行われるやや高級な手法について簡単に触れる。

（１）コードレビューとウォークスルー

プログラムを書いた当事者と部外者（同一企業の他の開発チームや他の部署、品質管理部隊など）とが一堂に会して、ソースプログラムの読み合わせを行いながら、エラーを発見することをコードレビューという。

他人が、自分の視点とは異なる観点で内容をチェックすることに意義があり、エラーの発見率は高い。

エラー発見は、他人によるばかりでなく、レビューのためにプログラム作成者が内容説明を行う際に、ポイントなどを整理するときにもしばしば自ら見つけることがある。

①　ウォークスルー（walk through）

一種のコードレビューであるが、あるテストケースを想定しての内容追究であり、レビューの参加者の頭の中で処理が正しいかどうかを確認する。あらゆるテストケースを洗い出してウォークスルーを行うのではなく、部分的に適用することになる。

②　インスペクション（inspection）

これも一種のコードレビューであるが、全体的なレビューというよりは、評価の観点をいくつかに絞って、それについて詳細な仕様評価を行う。

（２）テストの仕方

テストの対象プログラムに関して、内部処理を見てテストデータを作るか、外から機能だけを見てテストデータを作るかにより、テストの仕方が異なる。

①　ホワイトボックステスト

主に、プログラムの実行経路に対応させてテストデータを作る。テストの徹底性が追究できる。

②　ブラックボックステスト

プログラムが提供する機能について正しく実行しているかどうかを見るテストデータを作る。テスト方法としては自然なやり方であるが、テストの程度を保証しにくい。

2．構造化プログラミング

構造化プログラミング（structured programming）を整構造プログラミングともいう。

きれいな見やすいプログラムを書くと、以下のような多くの利点を得ることができる。

○　自分がみても、他人がみても内容を理解し易い
○　プログラム論理がよく分かる（プログラムの意図がよく分かる）
○　誤りがあっても、比較的早く気がつく
○　機能拡張など、手入れが容易に行える
○　他人にも保守が容易になる
○　独立性の高い適切なモジュール分割により、プログラムの再利用性も向上する
○　プログラムの生産性が向上する
○　欠陥を早く修復できる
○　誤りの少ないプログラムを作ることができる

このようなことを実現するために、プログラミング上どのような工夫をすればよいか、また、どのようなコーディングルールを決めたらよいか。

第２種情報処理技術者試験・平成２年度第２回での午前の部問７において、整構造プログラミングに関する試験問題が出題されたので、それをベースにその意義について説明しよう。

整構造プログラミング（構造化プログラミング）は、対象や問題を概念化して、プログラムを作りやすく、読みやすく、理解しやすく構成することを目的としている。これは**順番**、**反復**、**選択**などの基本的な制御構造を用いて手続きを表現し、構成する。

順番は、一連の手続きが逐次に実行される構造を表す。

反復は、ある手続きを繰り返して実行する。繰返しの条件を検査するのに、多くのプログラム言語では、**前判定**の方式を規定している。これは、手続きの最低実行回数が０回であり、手続きの実行回数が１回の場合の記述も容易なので、より汎用的であるといえる。それに対して、繰返しの条件を検査するのに**後判定**の構文をも有するプログラム言語もある。これは、手続きの最低実行回数が１回である。

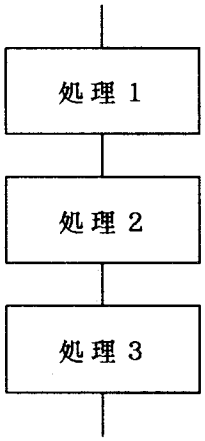
反復に関連して、手続きの途中での**抜け出し**の構文や、いわゆる“無限”繰返しなどの構文を持つ言語もある。

選択は、条件によって二つの手続きのいずれか一方を実行する。これを拡張して、変数や式の値に応じて複数の手続きのうちの一つを実行する**多岐選択**の構文を持つ言語もあり、ｃａｓｅ文やｓｅｌｅｃｔ文などで表現する。

前頁のようなプログラムを書くためにはわかりやすい論理を組み立てなければならないが、逆に論理や処理の流れがわかりにくくなる要因としては、
・処理の入り口が複数個あったり、出口が複数個あるような場合
・処理の流れが複雑に入り組んでいるような場合
が典型である。

次に、構造化プログラミングにおける３種類の基本構造について簡単に説明する。

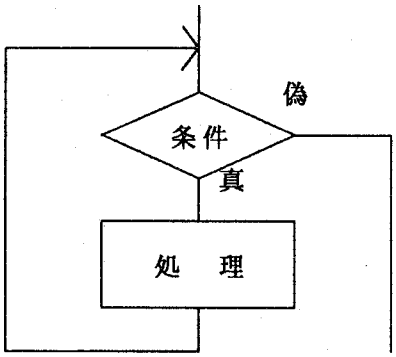
［順番（Sequence）］構造　（連続または逐次構造などともいう）



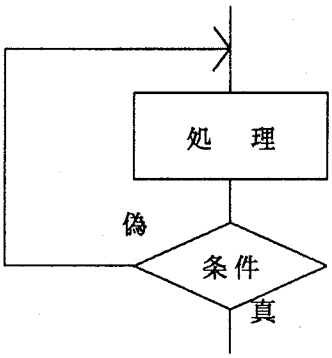
直線的に連続して行う処理である。

［反復（Repetition）］構造　（繰返し型ともいう）

(a) ＤＯ ＷＨＩＬＥ 型
ある条件が満足する間だけ処理を繰返す。（実行が０回もありうる）

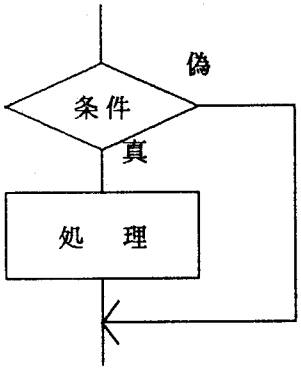


(b) ＤＯ ＵＮＴＩＬ 型
ある条件が満足されるまで処理を繰返えす。（実行は１回以上）

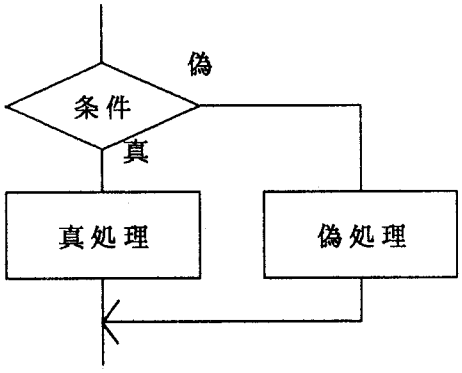


［選択（Selection）］構造

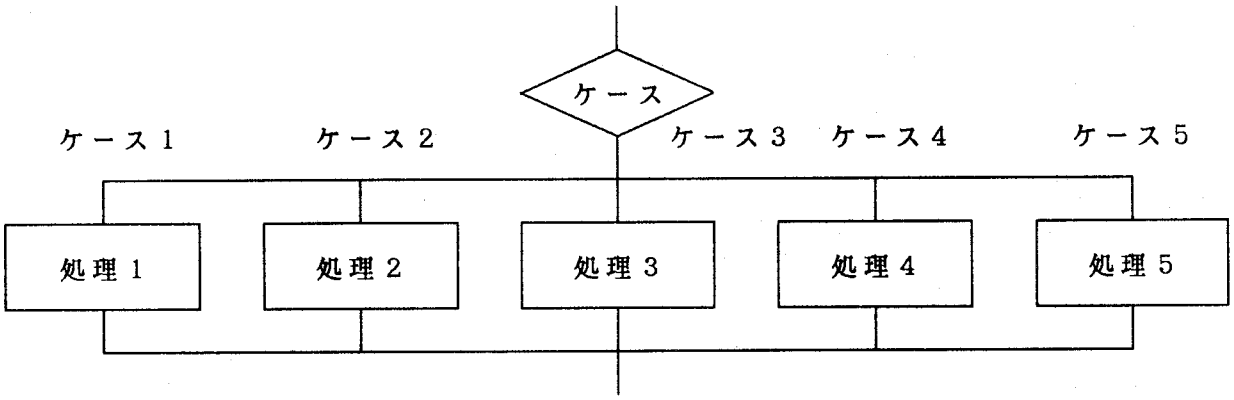
(a) ＩＦ　ＴＨＥＮ　型



(b) ＩＦ　ＴＨＥＮ　ＥＬＳＥ型



(c) ＣＡＳＥ　型（多分岐型）



ところで、一般の第２種情報処理技術者クラスの参考図書では、構造化プログラミングについての説明はこの程度に留まっている。しかし、これだけで構造化プログラミングとは何かを語り尽くせれば話は簡単であるが、実際には生徒にとって十分な理解を得られないことも多かろう。そこで、もう少しこの技法が必要となった背景について触れておこう。

構造化プログラミングとは、広義には、プログラミングにおける全プロセスにおいて、一貫した作業の枠組みを与えようとするものである。したがって、システムの要求定義書、機能仕様書、プログラム設計書、プログラミングの全ての作業過程に対して、アウトプット（ドキュメントやプログラムソースコード）の作成方式に関する標準的な規則を設け、

開発に関係する全員がその規則に準じて作業を進め、よってチーム全体のプログラム生産効率の向上、および、プログラム品質の向上を確保することを目指すことになる。

また、それとともにもう少しプログラムの書き方に着目してみると、構造化プログラミングが重要視されるに至った背景が明らかになる。

高水準プログラミング言語の歴史を振り返ると、７０年代後半には記述性の欠陥が大きく指摘はされたものの、科学技術計算系では、ＦＯＲＴＲＡＮ、事務処理系ではＣＯＢＯＬが６０年代初期から研究用にも、商用にも使われソフトウェア生産に大きな貢献をし始めた。また、ほぼ同時期にヨーロッパにおいて科学技術計算用にアルゴリズムを明快に記述できる構造化の概念を持ったＡＬＧＯＬも登場し、さらに６０年代後半には強力な言語仕様を持つＰＬ／Ｉが商用化され、わかりやすいプログラムを作る素地は出来上がった。

残念であることは、優れた言語であるＡＬＧＯＬを生んだヨーロッパが、米国や日本ほどコンピュータの利用の勢いが大きくなく、また先進国である米国、及びそれに追随した日本ではＦＯＲＴＲＡＮ、ＣＯＢＯＬが長らく主要言語であったため、センスのよいプログラムを書く風土が形成されにくかった点にある。

また、米国ＩＢＭの考案したＰＬ／Ｉは、ＦＯＲＴＲＡＮ，ＣＯＢＯＬ，ＡＬＧＯＬの優れた部分を集大成し、またオールマイティ性を狙ったこともあり、結果的に大仕様言語となってしまい、この言語を使いこなせる技術者はあまり増えなかった。

こうした状況下で、ＦＯＲＴＲＡＮ言語の記述性の悪さ、プログラムの増大にともないソースコードの内容が理解しにくくなる点が大きくクローズアップされ始めた。この主たる原因は、処理の流れをきれいに表現する文（ステートメント）が言語機能としてサポートされてなかったことにある。ＧＯＴＯ文が多用された非常に入り組んだわかりにくいプログラムが蔓延することとなった。特に１つのプログラムを何人もの技術者が保守を繰り返すとそれが顕著になっていく。

折しも、ダイクストラが［ＧＯＴＯ文有害論」を唱え、プログラムの制御には基本的に３種類のシンプルな構造で必ず現せることを強調し、これを受けてプログラミングスタイルに格段の改善をみるようになったのが７０年代中頃である。日本において構造化プログラムが根付くのは８０年代に入ってからであった。

構造化プログラミングの優れた点は、

○　プログラム手続きは、処理の集合を形成し、かつ処理は連続した流れとなる

○　どの処理も、それが開始する場所、終了する場所は１カ所に決められている

にあり、これにより処理を構造的に簡潔明瞭に表現できる。

プログラミングレベルでいえば、

○　永久ループや、孤立した命令を作ることがなくなる

○　無用のＧＯＴＯ文がなくなる

○　プログラムは上から下の方向に流れる

ことになり、コーディングミスも著しく減少することになる。

このようなプログラミングに関する反省もあり、ＦＯＲＴＲＡＮが１９７７年に、ＣＯＢＯＬが１９８０年に、構造化プログラム機能を導入し、プログラムの記述性を大いに高めることとなった。

指導上の留意点

あまりむずかしいことを教えすぎず、プログラムは基本制御構造に準じて書くくせを付けさせることが基本である。

テストについては、色々な角度から実施する方法、テスト技法があることを理解させる。

用　語

コードレビュー、ウォークスルー、インスペクション、
ブラックボックステスト、ホワイトボックステスト
構造化プログラミング、整構造プログラミング、基本制御構造（順番、反復、選択）、

第2種情報処理技術者試験

○　プログラムの構成に関すること
　－プログラムのモジュール構成
　－モジュール結合に関する用語
○　プログラム技法に関すること
　－プログラムを効果的に作る手法
○　プログラムテストに関すること
　－デバッグ、テストに関する手法、考え方など

第３章　ファイルに関連する事項

指導目標

ワークステーションやパソコンの普及とともに、コンピュータ“ファイル”が一般業務に極自然に受け入れられるようになった。個人用コンピュータ環境下では、例えばパソコンソフトの利用者という立場で、その処理の対処となる情報の保管を主目的としてファイルと接している。この状況下では、ファイルがどのように作られ、情報が記録されるか、中味の構成はどうなっているかについて利用者は知っている必要はない。これは、ワープロ、電子メール、表計算、データベース等のアプリケーションソフトが内部処理として扱う範囲であるため、外側からは関知する対象とはならないことにある。

しかし、その一方で、ワープロソフト、データベースソフトを開発する立場になったらどうなるであろうか。開発されたソフトは開発者個人が使うよりは、不特定多数の利用者が、色々な環境下でそれを利用することになる。そのようなソフトを開発する場合、スペース効率のよいファイル設計を行わなければならない、アクセス効率のよいデータ格納を考慮しなければならない、また、他のソフトで作られたファイルを読んだり、逆に自ソフトで作ったファイルを他ソフトで読めるようにしなければならない等、色々な条件を満足する必要がある。

なかでも、汎用コンピュータのソフトウェア開発においては、大規模な大容量ファイルについて、高効率な、経済的な利用方法を実現しなければならない。このため、情報処理技術者にとっては、データの利用目的、データに求められる要件に最もふさわしい、ファイルの設計とそのアクセス方式を採用する能力が必要とされてくる。

本章において、専門的・技術的なファイルの利用方法について学ぶことになる。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
ファイルの概念	コンピュータデータの保管、アクセスの単位としてのファイル概要につき理解	物理レコードはデータの保管上どのような役割を果たすか考える	特になし	特になし
媒体上のファイル	磁気テープ上のファイルの記録方式、およびディスク上の記録方式を理解する	媒体の種類の違いにより、どのような利点があるか考える		汎用コンピュータ
ファイルの形態と用途	利用の主体者、データの内容、使用目的、使用期間、記録媒体、編成法からみた区分、割当てについて理解	どのようなソフトウェアを組むときにどのような種類のファイルを使用する必要があるかを考える		汎用コンピュータ
ファイルアクセスの方式	利用者からみた入出力要求から物理的な入出力制御が行われるまでのアクセス方式につき理解	何故ファイルに対する論理的なアクセスと物理的なアクセスがあるかを考える		ＰＣ，ＷＳ
順編成ファイル	順編成ファイルに関する物理的特徴、アクセス方法、編成としての長短			汎用コンピュータ ＰＣ，ＷＳ

内　　容	説　明	議　論	机上実習	計算機実習
直接編成ファイル				汎用コンピュータ
索引順編成ファイル				汎用コンピュータ ＰＣ，ＷＳ
相対編成ファイル				汎用コンピュータ
区分編成ファイル				汎用コンピュータ
ＶＳＡＭファイル	従来のファイル編成法に比較してＶＳＡＭファイルの汎用性、柔軟性について理解する			汎用コンピュータ
階層構造とディレクトリ	ファイル管理の構造、管理上の利便性について理解する			ＰＣ，ＷＳ

1．ファイルの概念

ある目的をもって作られた一まとまりの情報にそれを代表する名前を付けたものをファイルと呼ぶ。このファイルをどこにどのように保管するかは、それを取り出したり、しまったり、中身に情報を追加したり、削除したり、修正したり、あるいは整理したりする人が決めることになる。

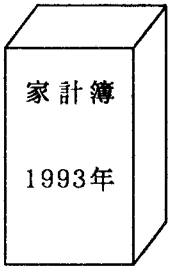
もし特定個人だけの情報であれば、該当する人が、情報を表す形式や保管場所を、また、秘密の情報であるか、開示されてもよいものかを決めることができる。つまり、その人が最も扱い易い形で、便利だと思う媒体に収納しておけばよい。

複数人の共有の財産であれば、共同で扱い易い形式で皆が取り出し易い場所に置いて管理すればよい。

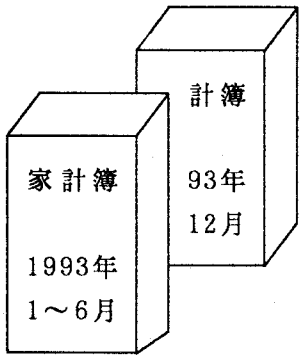
断っておくが、人の頭の中にしまったとか、口で喋って内容を取り出すようなことに対しては、ファイルとはいわないこととする。

このようなファイルをどこにしまうかは自由である。また、それを電子化した場合、いわゆる事務用品としてのファイルとを区別して、コンピュータファイルと呼ぶことも可能ではあるが、通常情報処理の世界では単に後者のことを“ファイル”と呼んでいる。

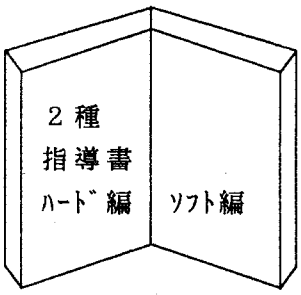
ここで、ファイルの概念を仮にノートに記述される情報と対応づけたとしよう。ノートは以下のように整理されるであろう。



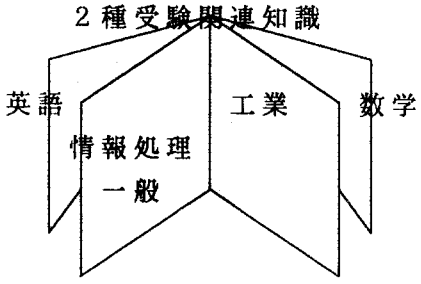
1冊の家計簿



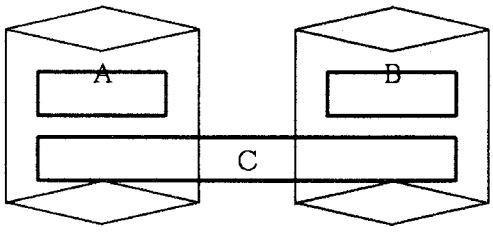
2冊一まとまりの家計簿



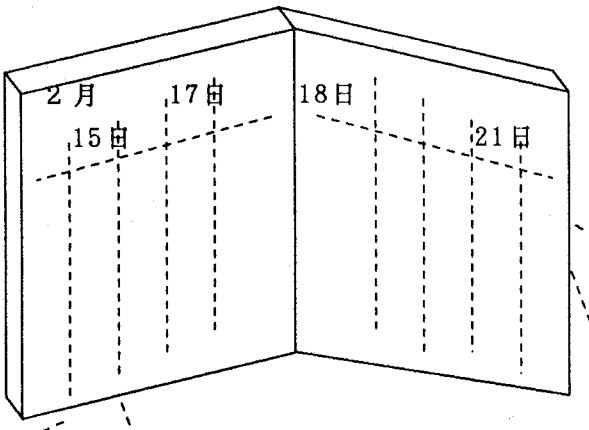
1冊で別の目的の情報



1冊で4つの目的の情報



コンピュータファイルでは上例もある



両開きページで1週間分の収支情報
（日単位の情報を束ねた（ブロック化）したものと考えることができる。）

収　入
肉・野菜
調味料
主　食
外　食
衣　服
交　際
交通・通信
貯蓄・保険
支出合計
現在高

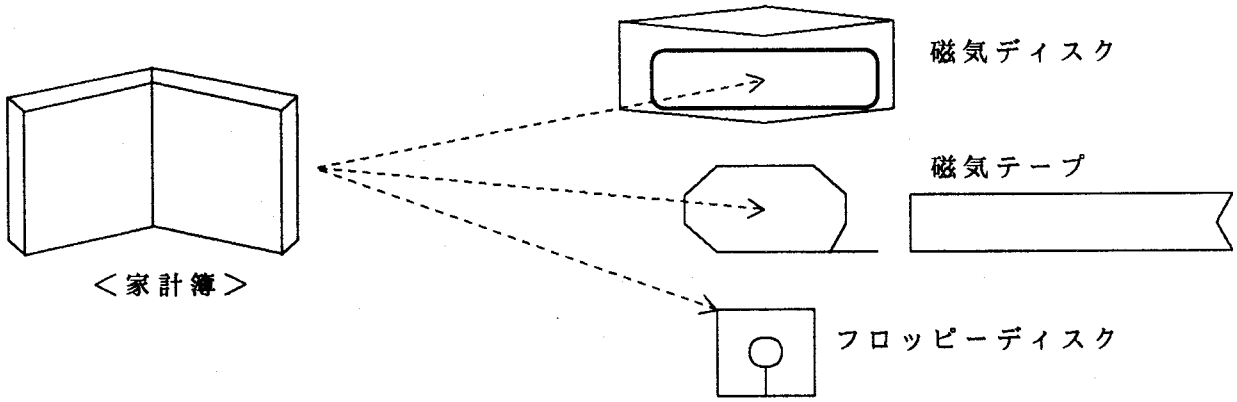
	月	火	水	木	金	土	日	週計	累計	予算残
肉・野菜										

情報としては、日単位の収支情報と捉えることも可能であり、また、支出費目別週間単位の支出状況情報と捉えることもできる。

（１）電子化されたファイル

情報処理でいうファイルとは、各種の情報を電子化したもの、即ち、磁気テープ、磁気ディスク、フロッピーディスク等の電子記録媒体上に格納したものを指している。

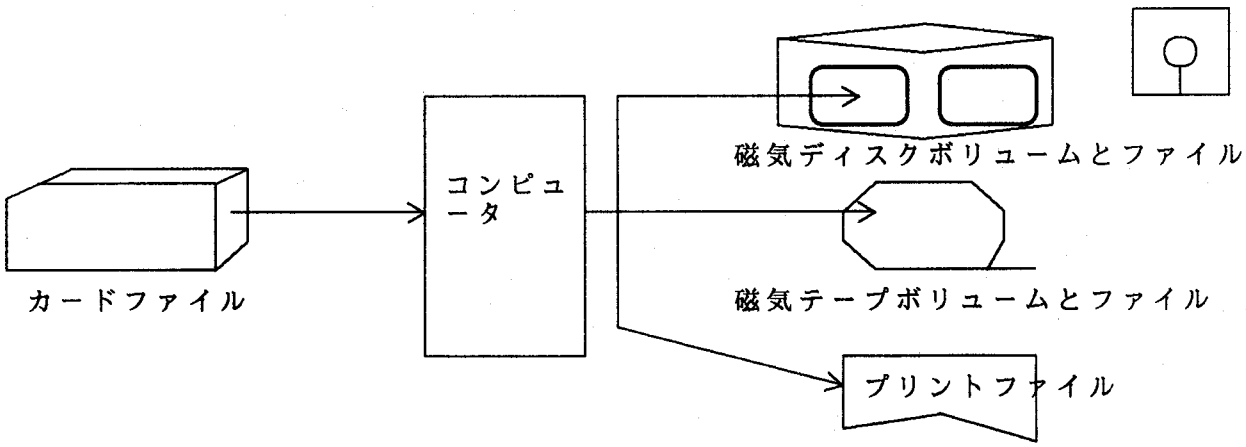
通常、一つの電子媒体上には複数個のファイルを作ることができる。それはそれぞれのファイルを完全に識別できるよう唯一の名前を付け、媒体からファイルの内容を取り出したり、情報を書き込んだりするときに、対象となるファイルを探し出すために、ファイルの内容の他に登録されているファイルに関する辞書をもって管理されることになる。

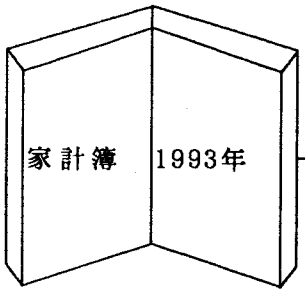


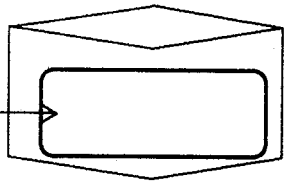
（２）ファイルとボリューム

電子記録媒体を総称して、ボリュームと呼んでいる。ボリュームとは、ファイルの上位の管理対象であり、そこに複数個のファイルが収納されている。

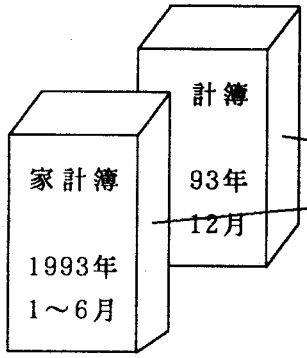
下図のように、ファイルとボリュームが１対１対応する場合もある。例えば、パンチカードや、プリント出力されたリスト等がそれに当たる。

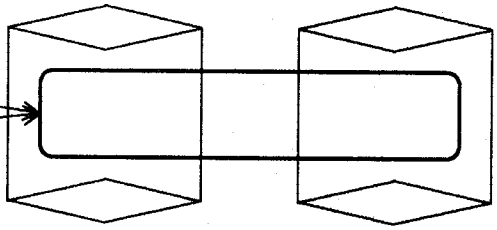




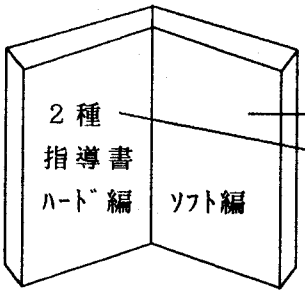


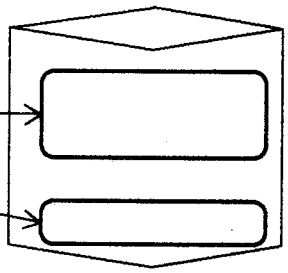
＜単一ボリューム単一ファイル＞



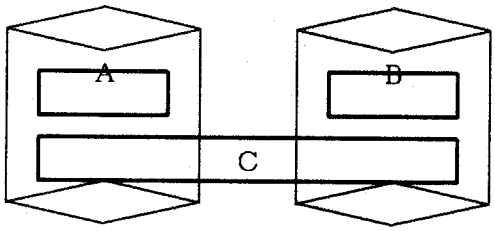


＜複数ボリューム単一ファイル＞





＜単一ボリューム複数ファイル＞



＜複数ボリューム複数ファイル＞

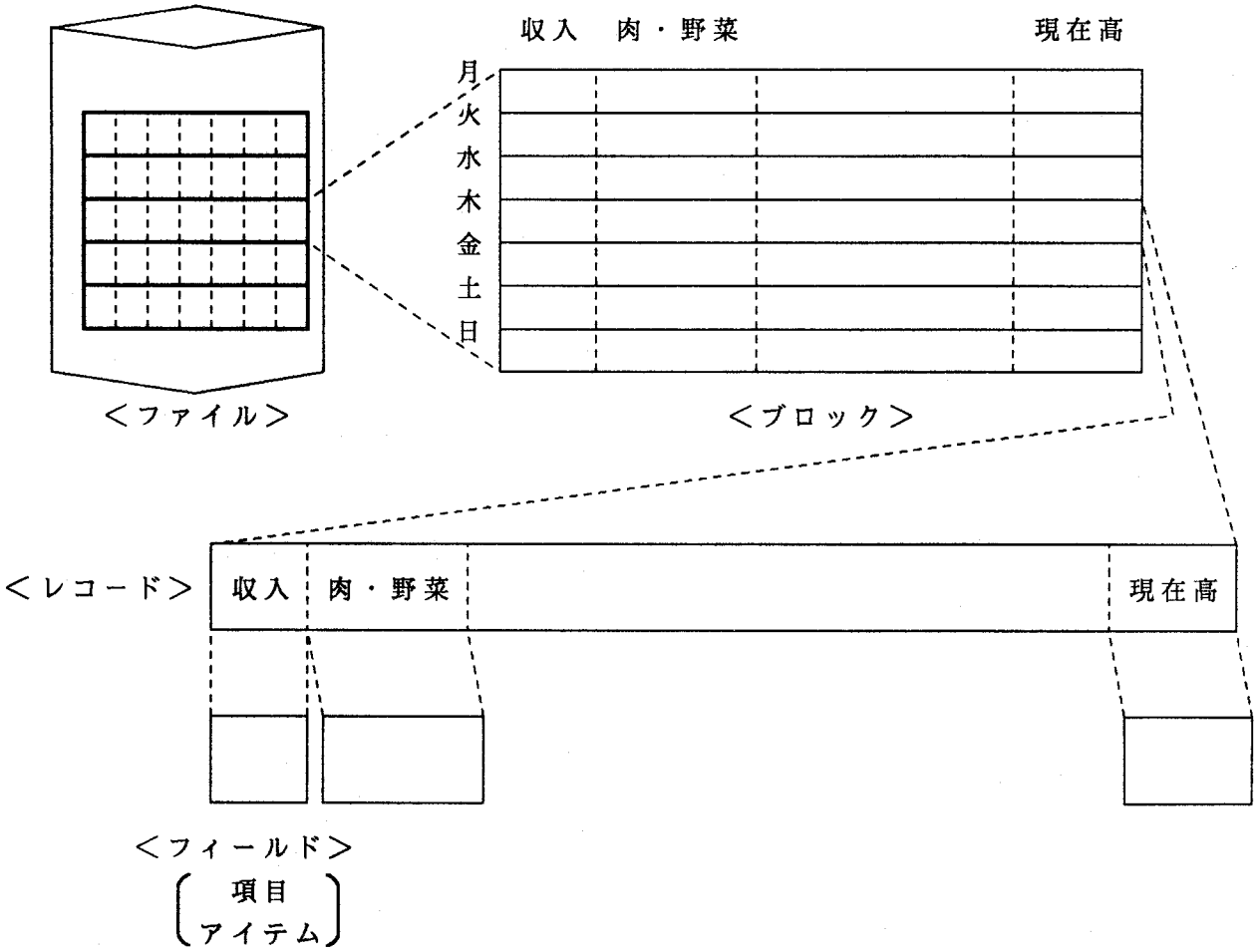
（３）ファイル、ブロック、レコード、フィールド

ファイルはある目的を持った情報の集合ではあるが、その情報の内容形式は無秩序なものではない。一定の規則を持ってまとめられた情報が複数個集まったものである。この単位をレコードと呼び、ファイルへの収納の最小の単位である。例えば、ノートの１ページ、あるいは１行等と対応する。

レコードはさらに、いくつかの細かい情報を集めたものである。その情報をフィールドと呼び、フィールドは通常、人間が意味を解釈する最小の単位である。但し、コンピュータ処理の世界ではさらに細かい単位もある。

このように、情報の管理の立場からみると、ボリューム＞ファイル＞レコードのレベルで記録され、利用の立場からみると、ファイル＞レコード＞フィールドというレベルの認識となる。

（注）ファイルの種類によっては、全く規則を持たない文字情報の塊である場合もある。



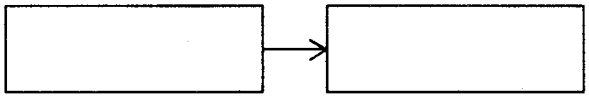
（４）論理レコードと物理レコード

前記のような、複数個のフィールドをある概念でまとめた単位を“論理レコード”と呼ぶ。コンピュータ利用者から見た場合、プログラムにより取り出し、書き込みを行うときの最小の単位でもある。それが、コンピュータ媒体上に置かれた場合、“物理レコード”と呼ばれる。

論理レコードと物理レコードの関係は、１つの論理レコードと１つの物理レコードが１対１の関係にある場合と、複数個の論理レコードをまとめて１つの物理レコード（１対ｎの関係）として媒体に置く場合とがある。

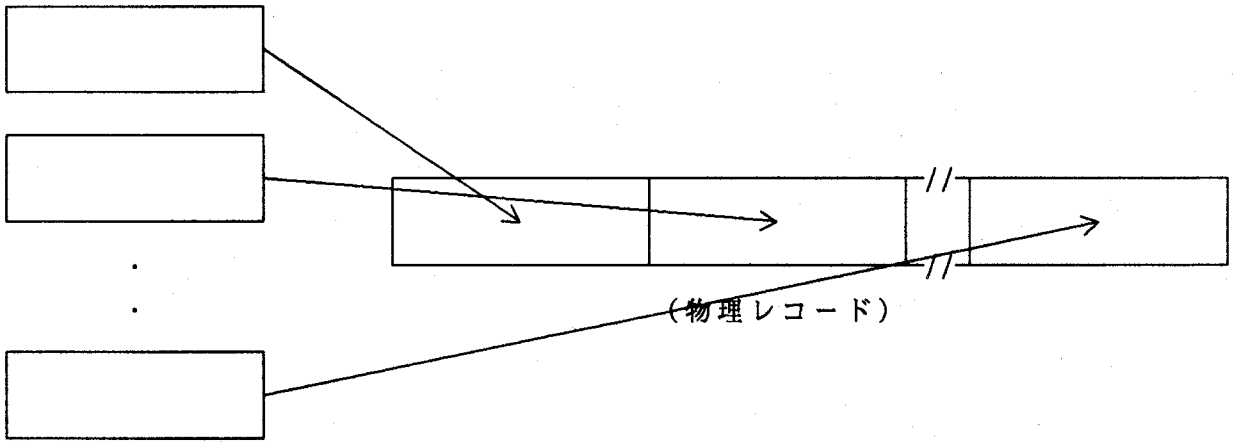
例えば、家計簿ファイルの場合、日単位のレコードを１論理レコード、週単位でまとめた論理レコードの集まりをブロックとした場合、論理レコードと物理レコードの関係を以下のように捉えることができる。

①　１論理レコード　＝　１物理レコード　のケース



（論理レコード）　　（物理レコード）

②　複数個の論理レコード　＝　１物理レコード　のケース



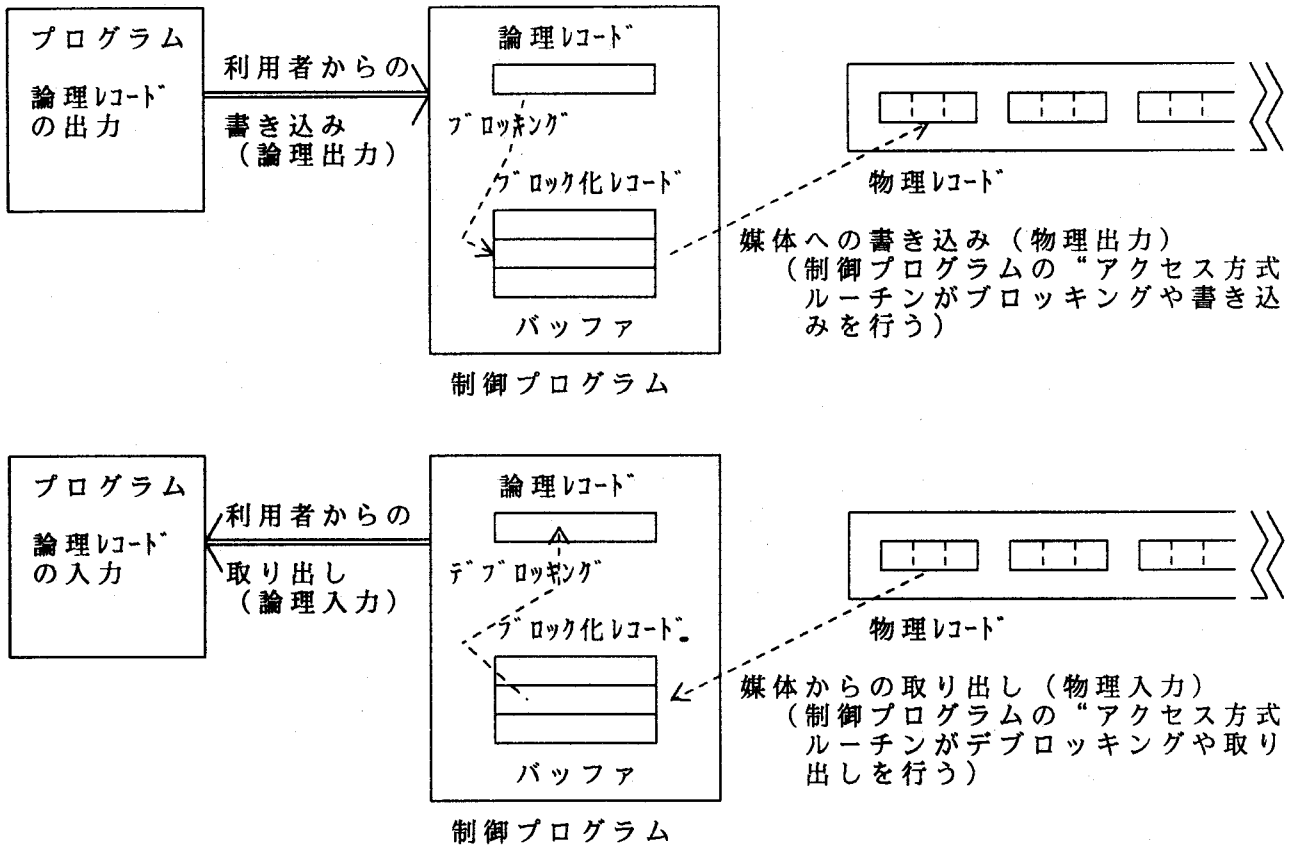
（論理レコード）

②の場合は、まとめの単位はある固定の数であり、これをブロック化因数（Ｂlocking Ｆactor）という。但し、通常プログラムからはこのブロック化因数は意識しない。

（５）物理レコードについて

ブロック化された物理レコードは、通常利用者のプログラムからはそのブロック状態を意識することは殆どない。この目的は、記憶媒体と主記憶装置との間での転送オーバヘッドをできるだけ効率よく行うことである。

この仕掛を下図に示す。



高水準言語でプログラムを書く場合、入出力ステートメントを用いてレコードの取り出しや書き込みを行う。この場合、プログラマは入出力の対象となる論理レコードだけに興味があり、どのような方式で物理的な入出力が行われているかは分からない。もっとも、ブロック化因数を大きくしないと入出力に時間がかかることは次第に分かってくるだろう。

このブロッキング／デブロッキング、および物理入出力を行うのは、入出力ライブラリであり、アセンブラなどの低水準言語で書かれていることが多い。

（６）レコード形式

レコードについて、コンピュータ処理の立場からもう少し詳細に見てみよう。１つの見方としては、論理レコードの中身についての分析である。どのようなデータ項目の、どのような構成によりレコードが成り立っているか構造について規定することである。

もう一つの見方としては、レコードの物理的な形式についての規定である。長さはどのくらいか、どのようにブロッキングするか等について、ボリューム上にファイルを定義するときに決める必要がある。

①　レコードの論理的な構造

各フィールドの細目を規定する。意味、バイト数（桁数）、表現形式、値のとる範囲などである。

市町村情報ファイルについて、下記のようなレコード形式を考えることができる。

県コード	市町村名	市町村長名	面　積	人口数	世帯数	選挙権保持者数
２桁 ２進数	８桁 文　字	１６桁 文　字	１０桁 小数	５桁 ２進数	４桁 ２進数	５桁 ２進数

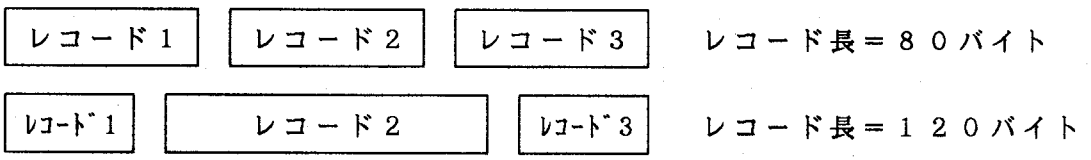
社員の健康保険ファイルについて、下記のようにレコードを定義できる。

社　員 番　号 ５桁 ２進数	社　員 氏　名 １６桁 文　字	被扶養者１			被扶養者10		
		続柄 １桁 ２進	生年月日 ６桁 ２進数	氏　名 １６桁 文　字	続柄 １桁 ２進	生年月日 ６桁 ２進数	氏　名 １６桁 文　字

②　レコードの形式

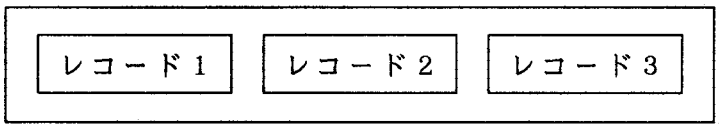
レコード長、固定長／可変長、ブロック長、スパン属性、不定長等につき規定する。

(a) 論理レコードの長さ（バイト数で設定）

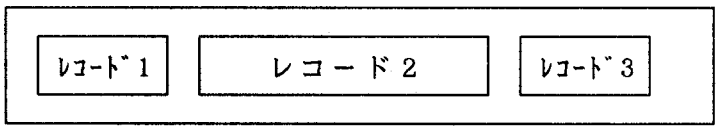


(b) 論理レコードの長さが可変か、固定か（可変の時はレコード長は最大長となる）

(c) ブロッキングするかどうか（固定長レコードの時、論理レコード長の倍数である）

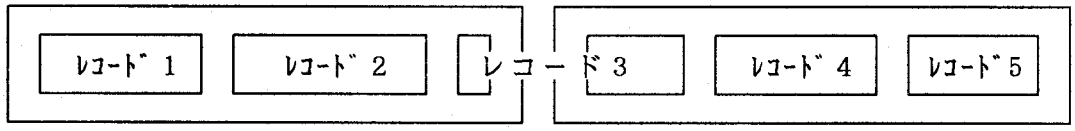


固定長レコードのブロッキング
レコード長80、ブロック長240



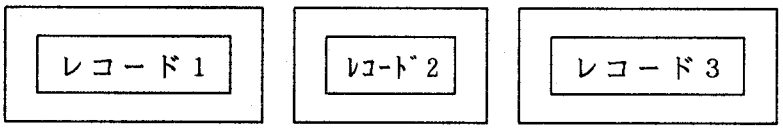
可変長レコードのブロッキング
レコード長100、ブロック長240

(d) スパンするかどうか（可変長でブロッキングするとき、ブロック間にまたがって論理レコードを置くかどうか）



スペース効率を上げるかどうか。レコード処理としては、スパンしない方が効率は高いが、スパンするとディスクスペースの使用効率が良くなる。

(e) 不定長レコードは、論理レコードと物理レコードとが1対1対応する



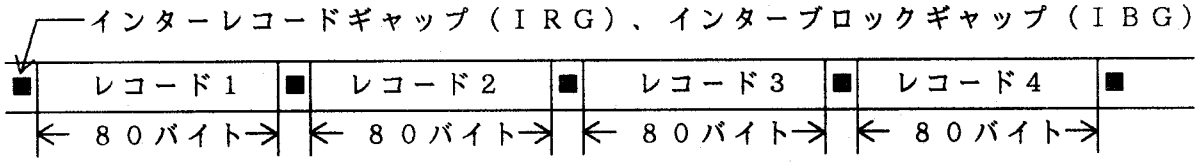
2．媒体上のファイル

磁気テープ、磁気ディスク、フロッピーディスク等にファイルを作成する場合、レコードの読み込み、書き込みのためにレコードの形式に基づいて記録情報を以下のように管理する。

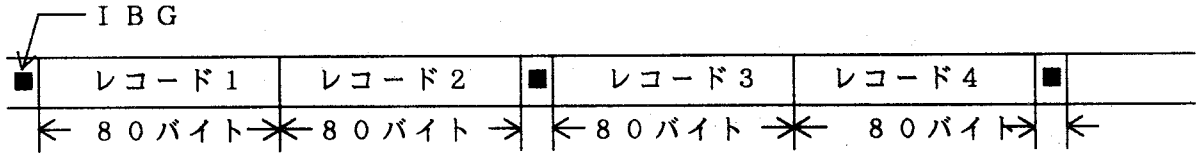
① 磁気テープファイル

レコード長、ブロック長、レコード形式によって媒体上に以下のような記録を行う。

(a) 固定長、非ブロック化

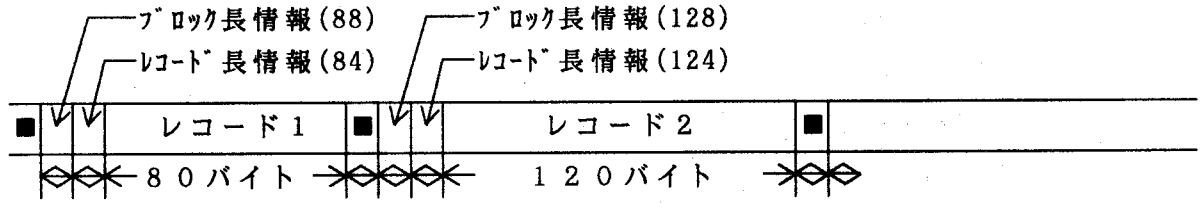


(b) 固定長、ブロック化



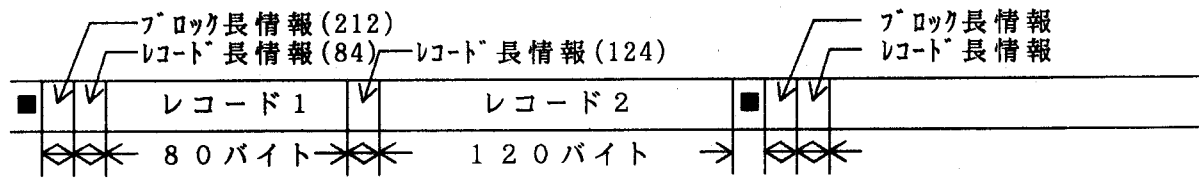
(c) 可変長、非ブロック化

各レコードの直前に、４バイトのブロック長情報と４バイトのレコード長情報を持つ

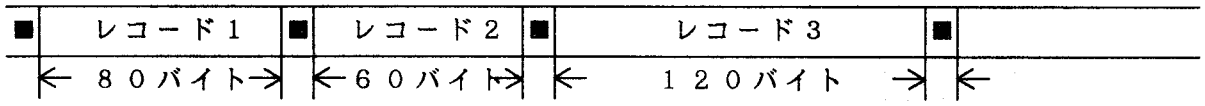


(d) 可変長、ブロック化

各ブロックの先頭に、４バイトのブロック長情報、各レコードの直前に４バイトのレコード長情報を持つ



(e) 不定長形式レコード



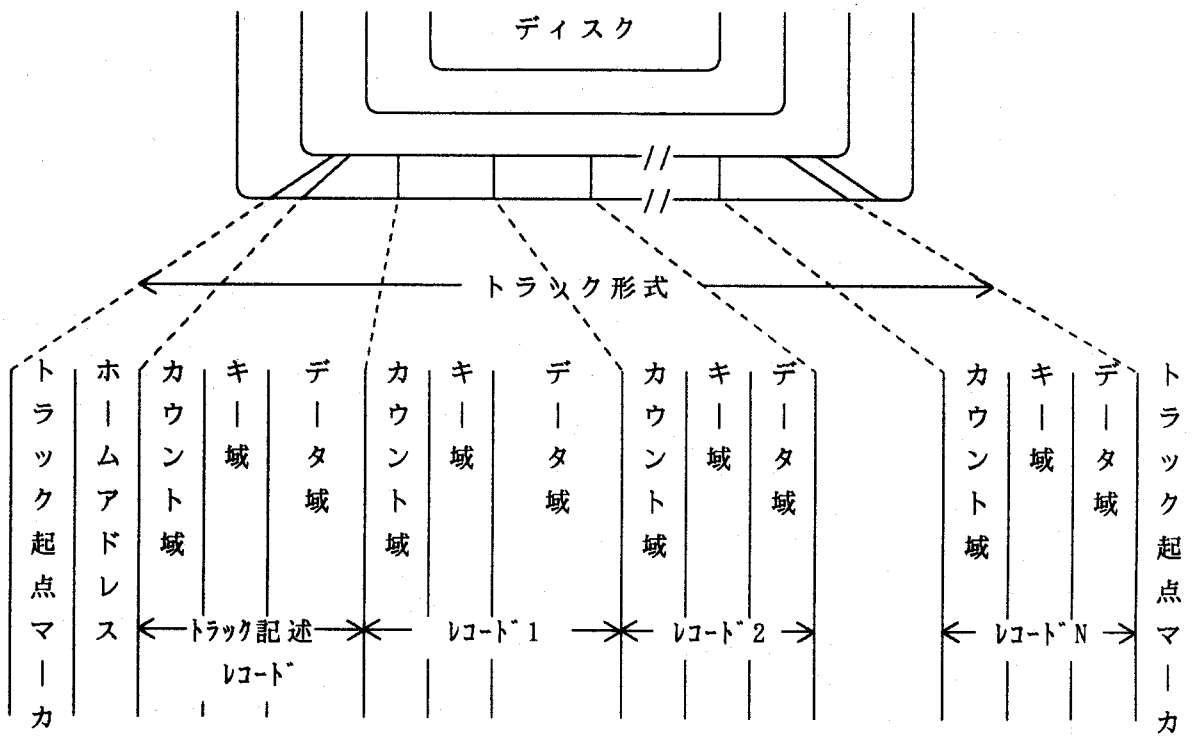
② ディスクファイル

磁気ディスクボリュームは、同心円状の複数個のシリンダ（円柱イメージ）からなり、各シリンダには等間隔で複数個のトラック（円柱の縦方向に）が並んでいる。

フロッピーディスクも同じようなイメージで捉えてよいが、同心円状のシリンダがあるのみで複数個のトラックは存在しない。

トラックの管理方式には、1トラック全体を一括管理する“トラック方式”と、トラックをいくつかのブロックに分割し、ブロック毎に管理をする“セクタ”方式とがある。

(a) トラック方式



上図は、可変長方式のトラック形式を表している。トラック記述レコードは他のデータレコードと異なり物理的な位置情報を持っている。

(b) セクタ方式

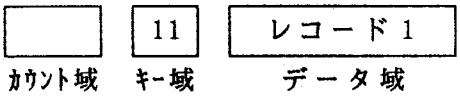
セクタ方式では、セクタを単位としてアドレスを指定する。セクタ方式でのトラック形式は、以下のようである。

◆セクタマーク　－　セクタの先頭を示す
◆アドレス域　　－　シリンダ番号、ヘッド番号、セクタ番号等を持つ
◆データ域　　－　実際のデータ

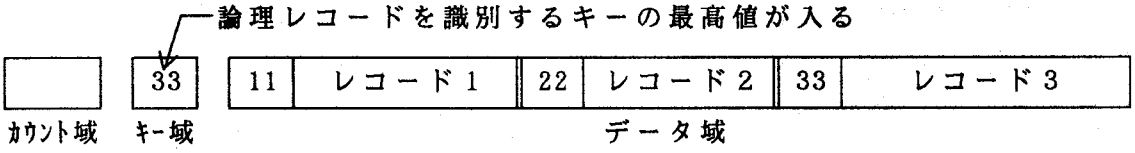
(c) レコード形式

ディスクボリュームのレコード形式には、カウント・キー・データ形式と、キーを持たないカウント・データ形式とがある。キーを用いてのレコード探索が多い直接アクセス処理の場合、前者の形式が便利である。論理レコードの形式は磁気テープの場合に準ずるが、物理レコード毎にカウント領域を、論理レコード毎にキー領域を持つ。

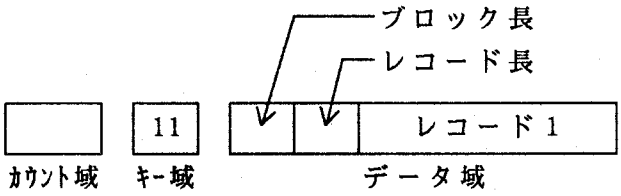
① 固定長、非ブロック化



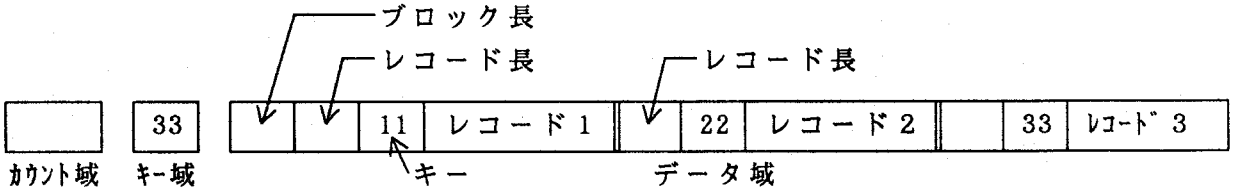
② 固定長、ブロック化



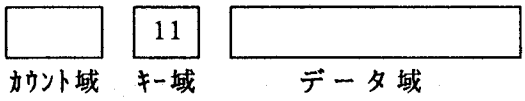
③ 可変長、非ブロック化



④ 可変長、ブロック化



⑤ 不定長

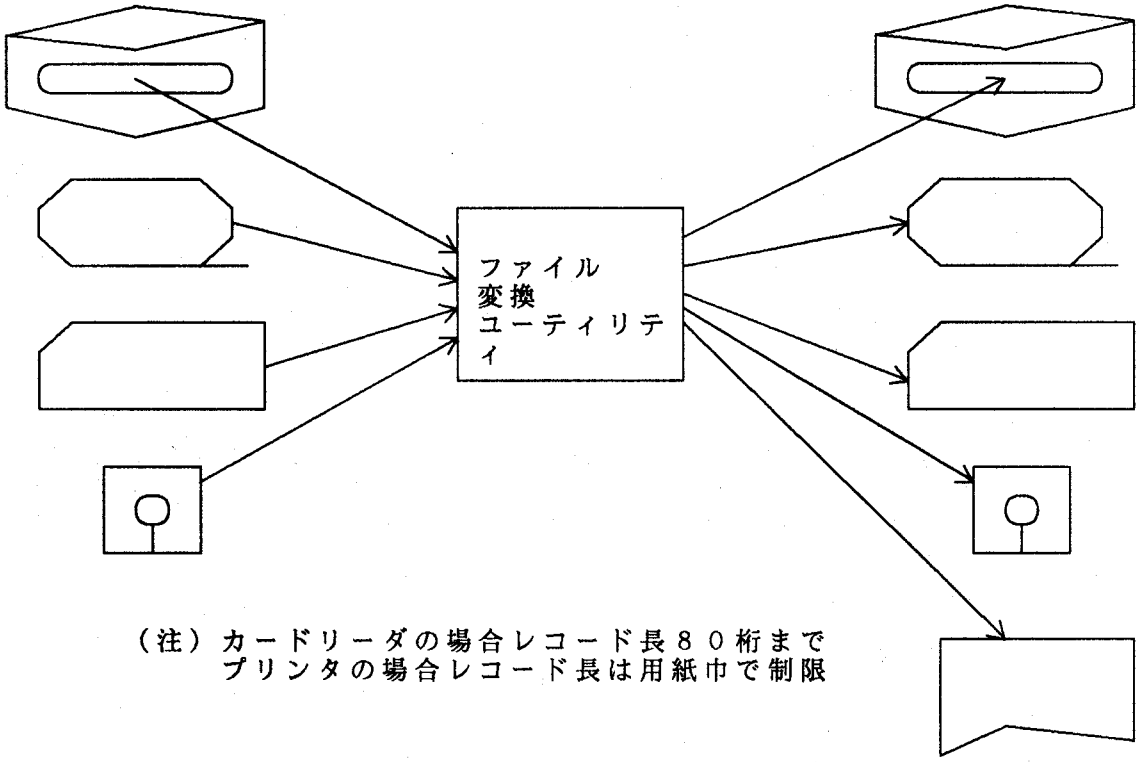


③媒体変換

ファイルはコンピュータの各種媒体に記録することができる。ファイル自身は通常それがどのような媒体上に作られたかに依存しない形式で保持されるため、作成した元とは異なる媒体に移動することができる。これをファイルの媒体変換という。

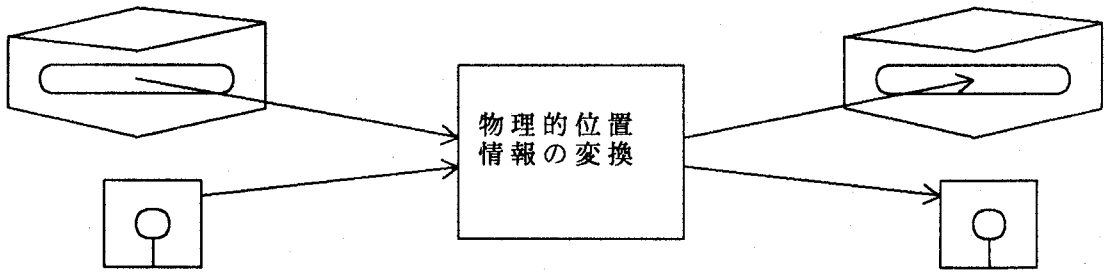
(a) テキストファイルの変換

(i) 順編成ファイル



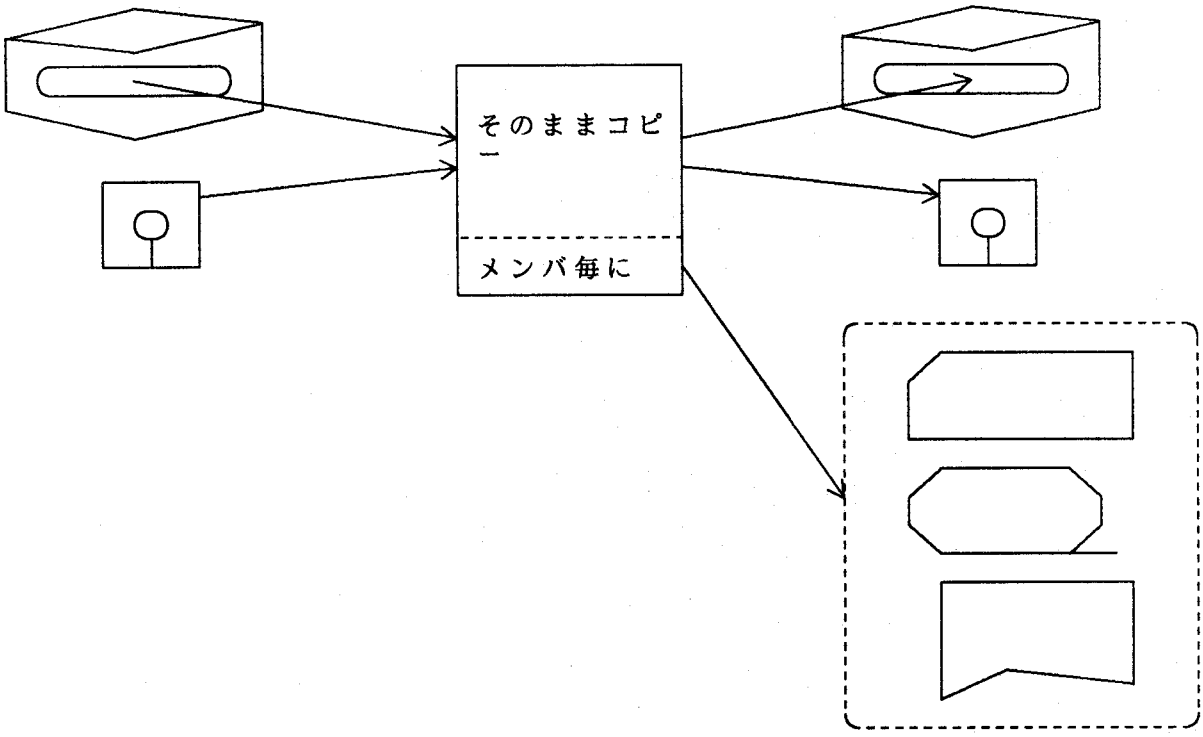
（注）カードリーダの場合レコード長８０桁まで
プリンタの場合レコード長は用紙巾で制限

(ii) 直接編成ファイル

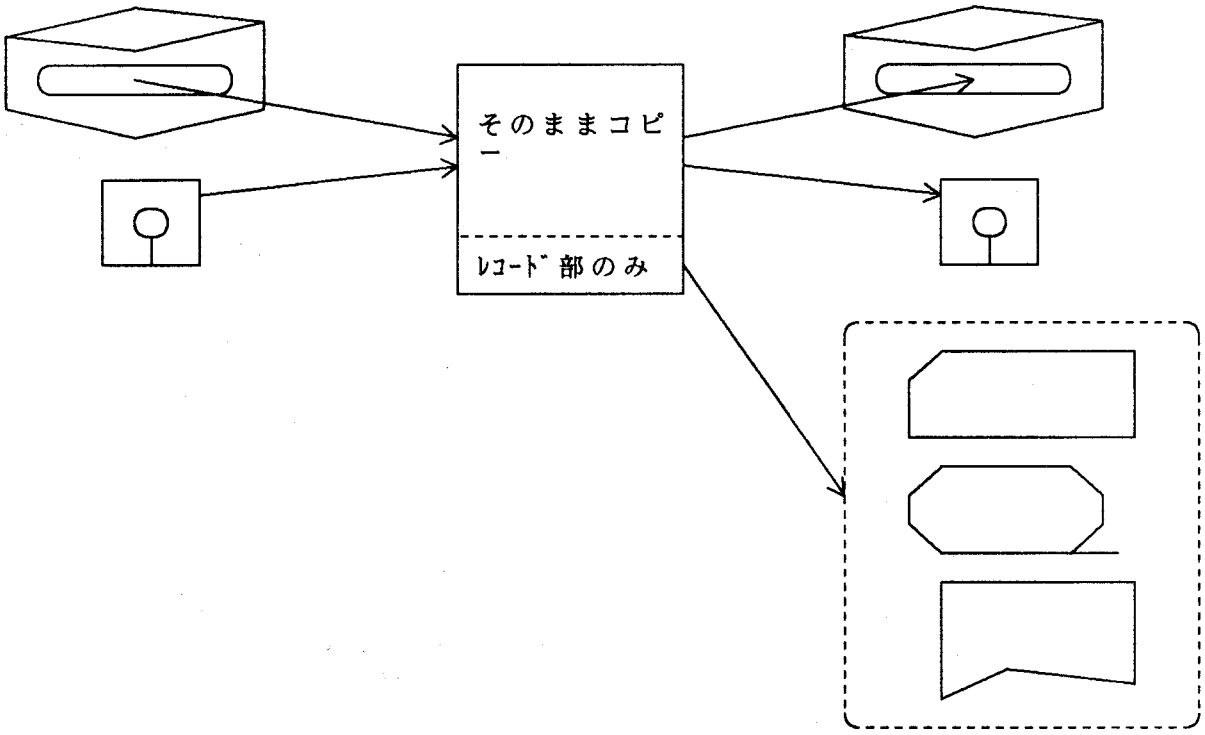


（注）入力データをシーケンシャルに取り出せれば何にでも変換できる

(iii) 区分編成ファイル



(iv) 索引順編成ファイル



(b) バイナリファイルの変換

基本的にはテキストファイルに準じるが、以下に留意

・カードにはコード表現上制限がある（縦方向１２欄パンチ孔）
・プリント出力は無意味

(c) 磁気テープの媒体変換上の注意事項

ラベル付きファイルの場合、メーカによってラベル内容に互換性がない場合がある。

3．ファイルの形態と用途

誰が、どのような目的で、どのように使うかまた、内容的な特徴は何かにより分類される。

（１）利用主体による分類

①　システムファイル

オペレーティングシステムの制御プログラムが、コンピュータシステムの運用・維持のために使うシステムサイドのファイルである。

(a) システム保守用のログファイル

(b) 使用料金課金用のファイル

(c) 共用プログラム、利用者プログラムを管理するためのファイル

②　ユーザファイル

ユーザが、特定の業務のために用意するファイルである。

（２）データの内容による分類

①　プログラムファイル

ソースプログラム、オブジェクトプログラム、ロードモジュールなど。

②　データファイル

制御プログラムやユーザプログラムが各種処理のために用いるデータファイル。

また、アプリケーション・プログラムの視点からファイルの用途を分類することができる。

（３）使用目的による分類

①　マスターファイル

基本ファイルともいい、相当長期にわたり使用されることから、永久ファイルとも呼ばれる。プログラム処理においては最重要なデータが格納されており、一般事務処理における帳簿の“台帳”に相当する。

このファイルは、変動条件の発生によりレコードの修正、追加、削除が行われる。データ処理においては、キー項目を頼りに参照・引用、そして維持される。

②　トランザクションファイル

変動ファイル、発生ファイル、更新ファイル、取引ファイルなどと呼ばれ、マスタファイルを更新するために使用されるファイルである。このファイルには取引データなどが記録されており、一時的な（伝票的な性格の）ファイルである。即ち、マスタファイルの更新などが終われば不要になる。

例えば、在庫マスタファイルに対する、日々の入出庫データとして使われる。

③　ワークファイル

処理の途中結果を一時的に保存するファイルのことで、作業用ファイルあるいは中間ファイル、スクラッチファイルとも呼ばれる。大抵、処理においてデータを主記憶域に置ききれなくなったときに、処理に直接関連する部分のデータを逐次出し入れするために用いる。分類（整列）・併合処理における中間結果の保存などが典型である。

④　バックアップファイル

予期しない障害、事故におけるファイル破壊に備え、適当なタイミングで別のファイルとして保存しておくことをバックアップをとるという。通常マスタファイルは適時予備がとられている。

⑤　ヒストリカル（ヒストリ）ファイル

過去から現在までの特定期間の履歴が記録されているファイルである。マスタファイルの内容変更［修正、追加、削除］に関連する事項を発生順にならべたもの。

マスタファイル破壊に対する内容回復（復元）に使用される。つまり、バックアップファイルとこのヒストリカルファイルとにより、破壊発生直前のファイルの状態に相当程度近づくことができる。また、ファイル破壊の発生の様子も分かる。

⑥　サマリファイル

ファイルを構成するレコードのある項目をキーにしてトランザクションレコードを分類する。さらに特定の項目だけを取り出してまとめ、先頭にキー項目を付けて１つの別のレコードにし、それらをまとめたファイル。

⑦　ログファイル

オペレーティングシステムの制御プログラムが出力したメッセージや処理内容の時系列的な記録ファイルである。システム障害からの復帰や、事故原因を調べるのに利用する。

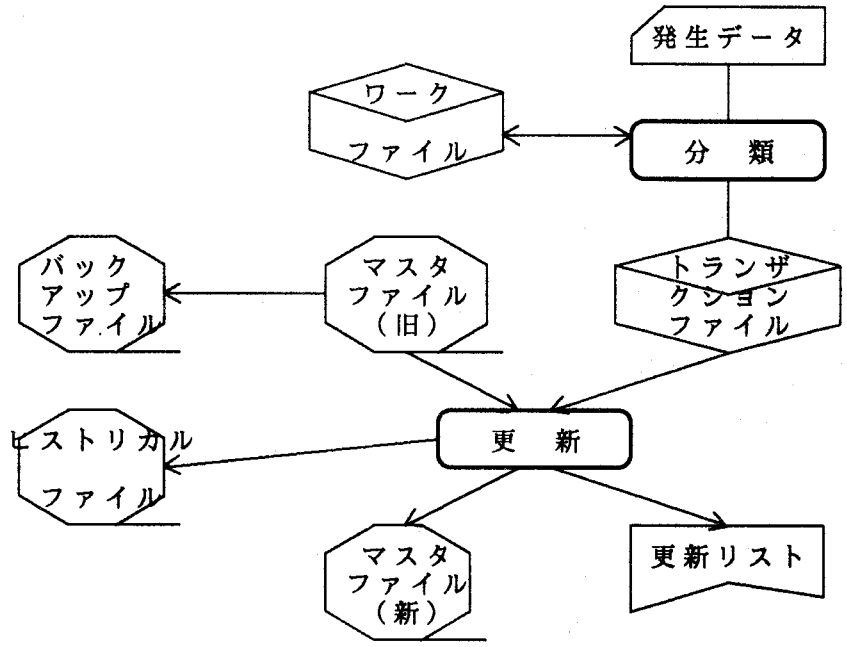
⑧　アーカイブファイル

性格上はバックアップファイルであるが、複数のファイルをまとめ、ファイル内容を圧縮しコンパクトにして保管するもの。システムファイルなどを定期的にバックアップし、磁気テープ等に格納し長期間保存する。

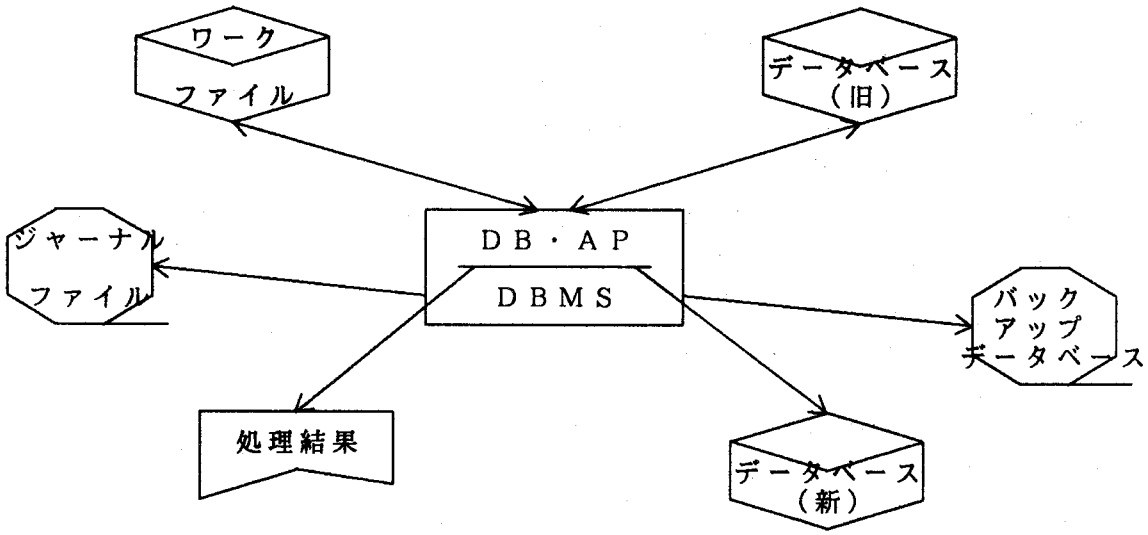
⑨　ジャーナルファイル

データベースの更新状態について、更新前後のレコードの状態を記録しておくファイル。データベースに障害が発生したときの回復に使用する。

≪マスターファイルの更新≫



≪マスターファイルの更新≫



（４）使用期間による分類

①　永久（パーマネント）ファイル

長期間継続的に使用、もしくは保存されるファイル。マスターファイルはこれに相当する。

②　一時（テンポラリー）ファイル

大量のデータを高速に処理するために、全体をファイルに、処理に必要な部分を主記憶域に置くもの。作業が終了すると不要となる。

（５）記録媒体による分類

①　カードファイル　　穿孔（パンチ）されたカードファイル。情報量が少ない。

②　紙テープファイル　　穿孔された紙テープファイル。読み取り速度が低速。

③　磁気テープファール　　単一リールファイル、複数リールファイルがある。

④　磁気ディスクファイル　直接アクセスが可能。大容量。

⑤　フロッピーファイル　　直接アクセスが可能。可搬性。

（６）編成法による分類

①　順編成ファイル

ファイルの先頭から順にレコードが記録されているファイル。レコードのアクセスはファイルの先頭から順に行う。大量のデータの順次一括処理に適する。バッチ処理ではこのファイルをしばしば用いる。

②　直接編成ファイル

レコードにキー（アドレス）を付けて記録されるファイル。レコードのアクセスはそのキーを基に行われ、キーに対応するアドレスにあるレコードを直接読み書きできる。トランザクションの高速処理に適する。

③　索引順編成ファイル

利用者によりレコードに索引（キー）が付加され、キーの値の順に記録されるファイル。レコードのアクセスは、順次の取り出しと、キー指定による直接取り出しが可能。

④　区分編成ファイル

順編成ファイルが“メンバ”という単位で記録される。メンバ名が登録簿（ディレクトリと呼ばれ、これも小さなファイルとなっている）に登録され、これを基に複数個のファイルが一括管理される。

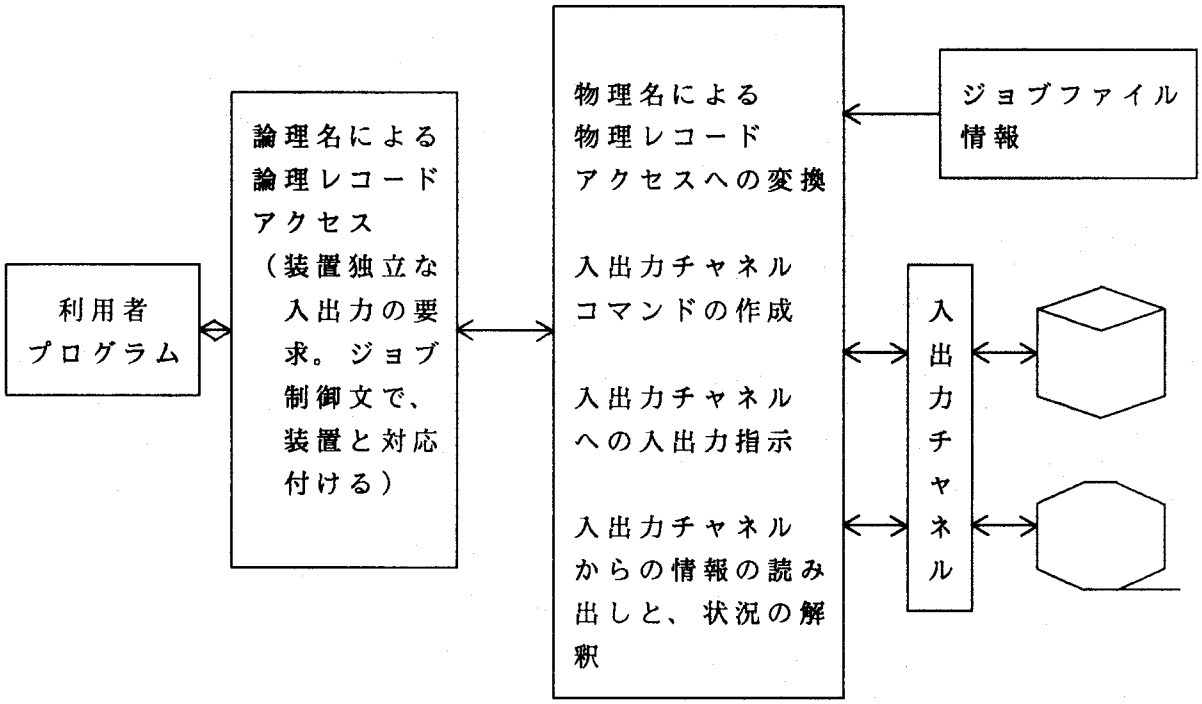
---

⑤　ＶＳＡＭファイル

直接アクセス記憶装置上に記録され、順次、索引順次、直接のいずれの方式アクセスをもサポートしている。直接アクセス記憶装置の物理的な情報を全く意識せずに仮想的なファイルとして、体系的で優れたアクセスを可能としている。

４．ファイルアクセスの方式

主記憶装置と補助記憶装置との間での低水準なレベル（実際の物理的な）のデータ転送を司り、入出力の制御を行う。一般に機械語レベルでの補助記憶とのやりとりを行うため、ユーザからの論理的なアクセス要求と対比して物理的な意味でのアクセスということで低水準のアクセスという。



アクセス方式には、ユーザの指定した論理レコードから物理的な入出力の単位となる物理レコードに変換またはその逆変換を行う機能を持つ。この処理プログラムは通常機械語で記述される。また、論理・物理の変換の方法により、基本アクセス方式（Ｂasic Ａccess Method）と待機アクセス方式（Ｑueued Ａccess Method）とがある。

① 基本アクセス方式

機械語でプログラミングする利用者は、論理レコードの入出力緩衝域（バッファ）へのブロッキング（もしくはデブロッキング）と、必要に応じ非同期（Asynchronous）入出力を行わなければならない。

② 待機アクセス方式

利用者は論理レコードを指定して入出力要求を出すと、このアクセス方式がブロッキング／デブロッキングを行ってくれる。また、非同期処理も自動的に行う。

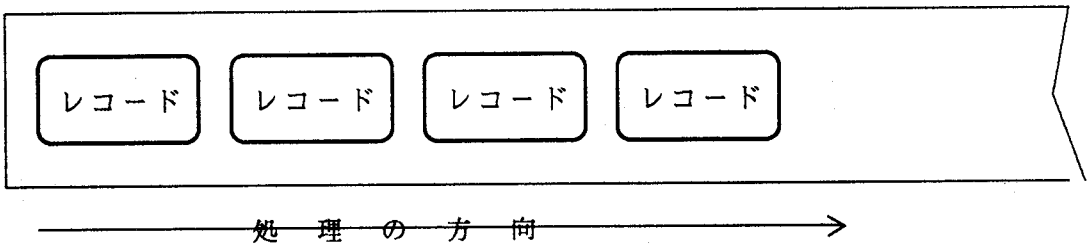
5. 順編成ファイル

ファイルの編成方法の説明にはいる前に、一般にサポートされているファイルの種類とその概要について簡単に整理しておく。

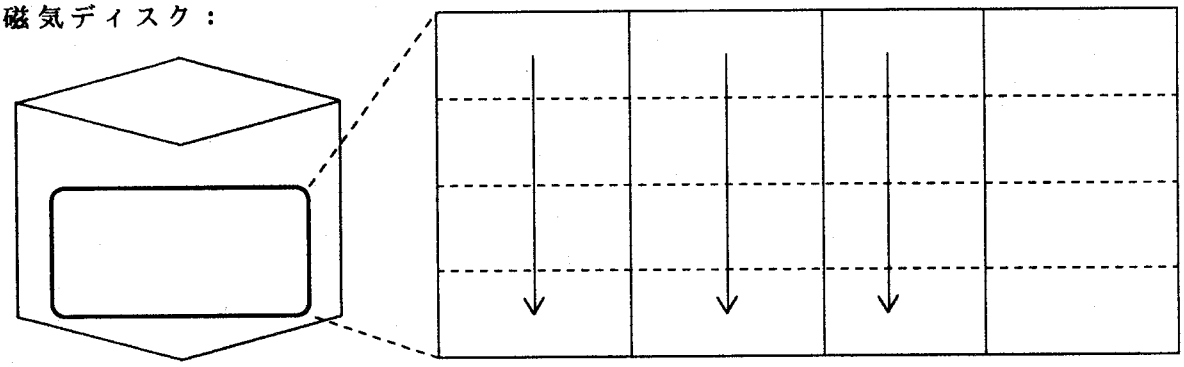
	物理的特徴	アクセス方法	サポート媒体	長　所	短　所
順編成ﾌｧｲﾙ	大量のデータをファイルの先頭から順にレコード単位に記録	ファイルの先頭レコードから順にアクセスする	カード 紙テープ 磁気テープ 印刷用紙 磁気ディスク ﾌﾛｯﾋﾟｰﾃﾞｨｽｸ	レコードの記録効率が最も高い。 ファイル全体の、登録順による一括処理に最適。	記録順のアクセスしかできず、効率が悪い。追加、修正、削除処理には物理的な工夫が必要
直接編成ﾌｧｲﾙ	ﾚｺｰﾄﾞのｷｰを基にﾃﾞｨｽｸの物理的ｱﾄﾞﾚｽにﾏｯﾋﾟﾝｸﾞされて該当位置に記録。間接アドレス、直接アドレスの２通りのﾏｯﾋﾟﾝｸﾞ法がある	キーを指定しての任意の位置にあるレコードのアクセスが可能	磁気ディスク ﾌﾛｯﾋﾟｰﾃﾞｨｽｸ	指定キーによる記録位置に無関係なアクセスが可能。 どのレコードも平均的な時間でアクセスでき、オンラインデータ処理に最適。	直接アクセス装置上に空領域ができ、媒体の使用効率が良くない。 また、シノニムレコードが多くなるとアクセス効率が落ちる
相対編成ﾌｧｲﾙ	ﾚｺｰﾄﾞのｷｰを基にﾌｧｲﾙの先頭からの相対ﾊﾞｲﾄ位置にﾏｯﾋﾟﾝｸﾞされて記録。直接編成に準じる	キーを指定してファイルの先頭からの任意の相対位置にあるレコードのアクセスが可能	磁気ディスク ﾌﾛｯﾋﾟｰﾃﾞｨｽｸ	指定キーによる直接アクセスとともに、順次、飛び越し順次、逆順アクセスが可能	
索引順編成ﾌｧｲﾙ	レコードのキーの昇順に記録。キーに関する情報が別ファイルとしてできる（インデックス領域）。	キーを指定しての直接アクセス、順次アクセスが可能	磁気ディスク ﾌﾛｯﾋﾟｰﾃﾞｨｽｸ	キーを指定しての任意のレコードからの順次アクセスと、キー値を持つレコードの直接アクセスが可能	レコード用ファイルの他に、インデックス領域、溢れレコード領域が余分に必要。順編成より記憶効率悪い
区分編成ﾌｧｲﾙ	メンバ名が登録簿に置かれ、そこで示される位置から順にレコード単位に記録される	メンバに割り当てられた位置の先頭レコードから順にアクセスする	磁気ディスク	メンバ単位にアクセスができる。 プログラムライブラリ管理などに利用できる。	メンバ管理のためのディレクトリ領域が必要、メンバ間に空き領域ができる。

レコードの入力順にデータが媒体に連続して記録されている。

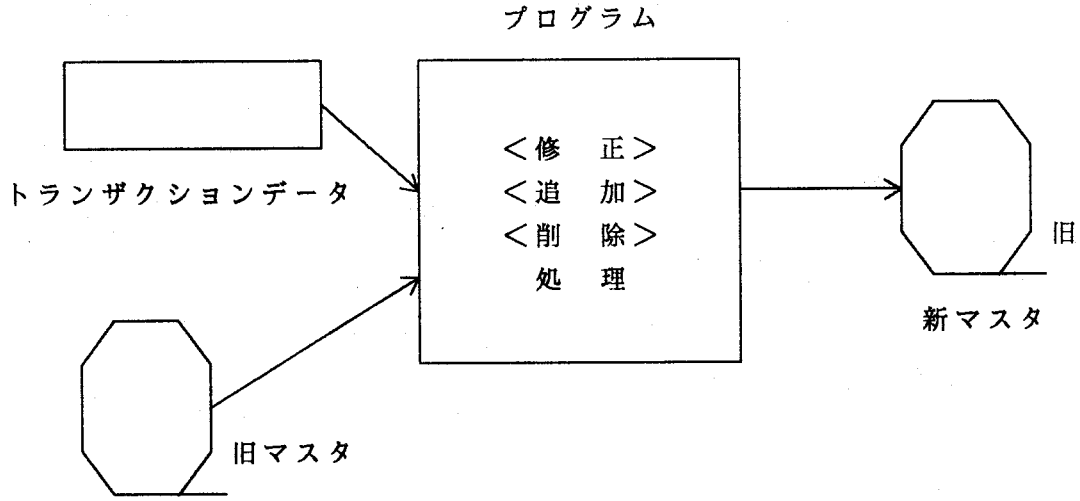
磁気テープ：

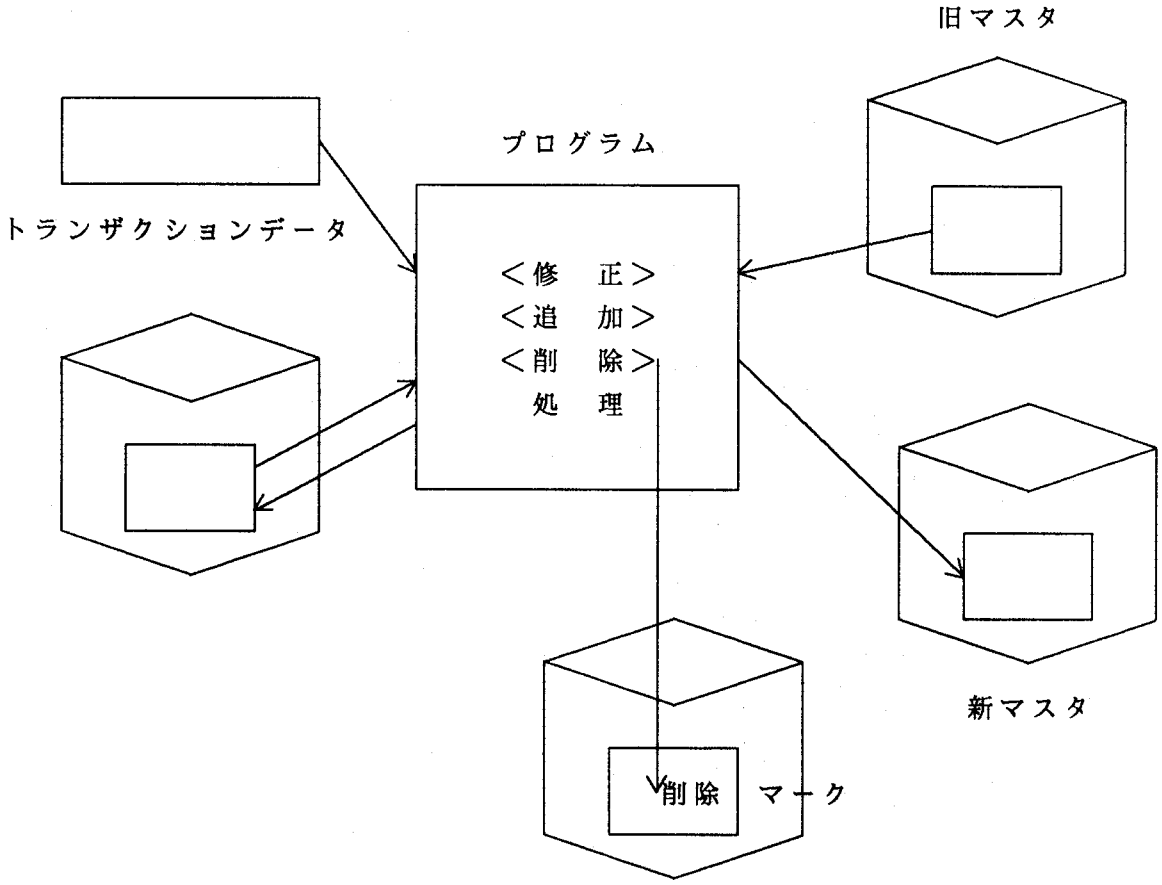


磁気ディスク：



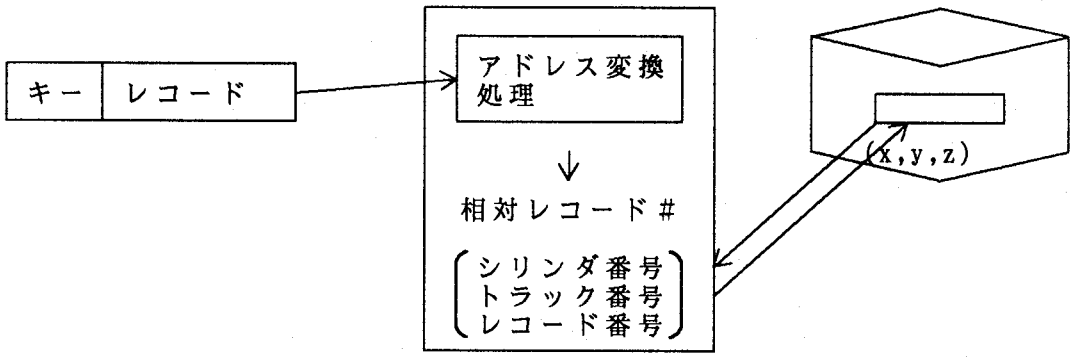
通常ファイルの先頭からの順次アクセスを行うが、［追加、修正、削除］などの処理を行う場合には、別のファイルに結果を納める。





6．直接編成ファイル

この編成ファイルは、順次アクセスには向かない。やはり直接アクセスに本来の高速アクセスの効果を果たす。



① 直接アドレス方式

キー値の発生状況により、ディスク上のスペースに空きができ、無駄な領域ができ易い。
均等に割り付けられるよう変換が必要。

相対ﾚｺｰﾄﾞ#	物理ｱﾄﾞﾚｽ

変換テーブル

② 間接アドレス方式

キー値に演算を施し、値のバラツキを最小にするように圧縮してアドレスの範囲を適切にする。

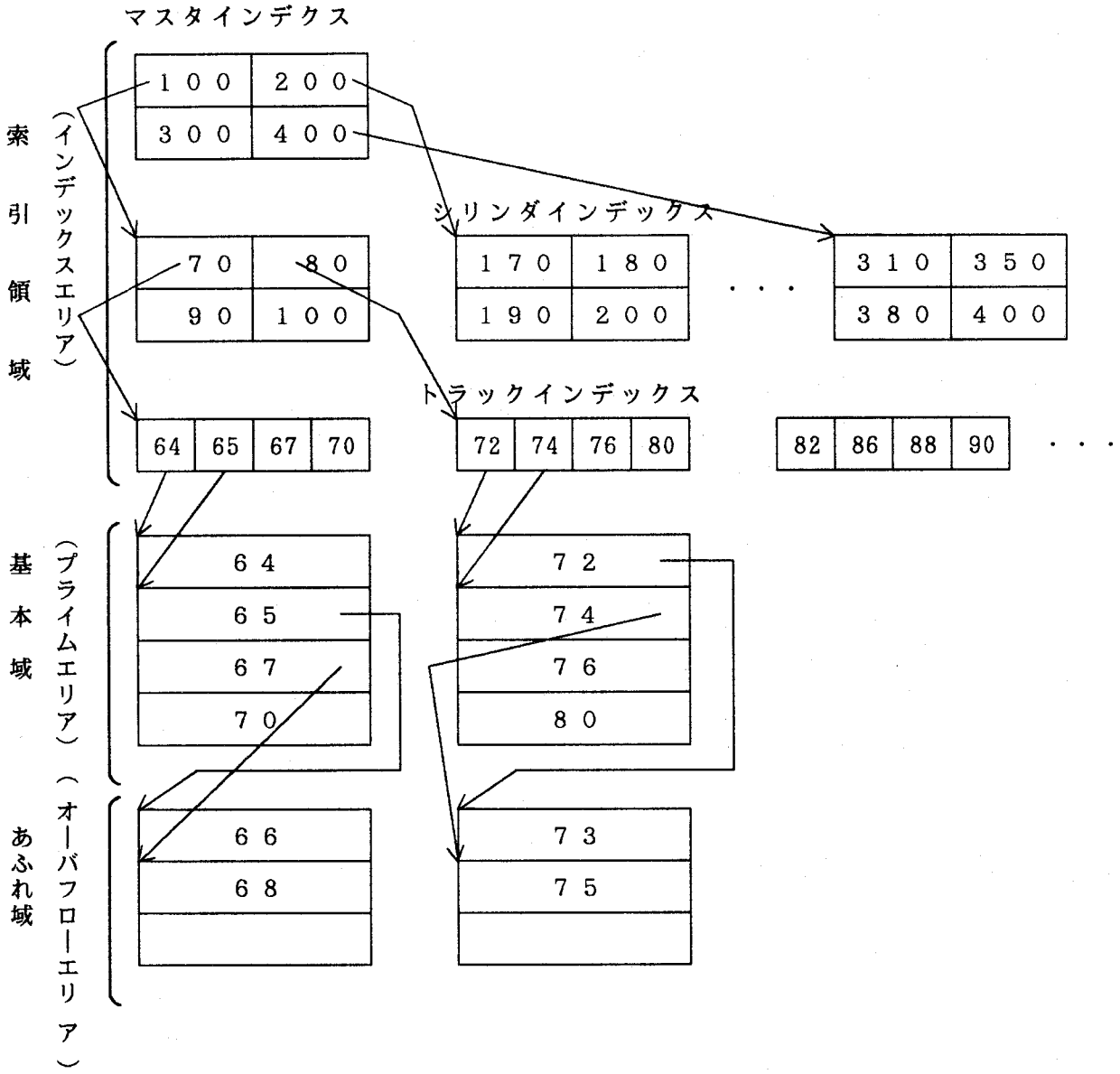
(A) 除算法：収容可能な総レコード数に最も近い素数でキー値を割り、その余りを求める。
(B) 重ね合わせ法：キー値を２部に分けてそれらを加算する
(c) 基数変換法：キー値を異なる基数に変換し、余分の桁を除く

≪シノニム（Ｓynonym）について≫
異なるキーを変換しても同一の相対レコード#になる場合、その#をシノニムという。
シノニムレコードはポインタでつながれシノニムレコード領域に置かれる。

７．相対編成ファイル

直接アドレス方式によるファイルを相対ファイルと呼ぶ。ＶＳＡＭ編成ファイルの項を参照のこと。

8. 索引順編成ファイル


マスタインデクス
索引領域
（インデックスエリア）
１００
２００
３００
４００
シリンダインデックス
７０
８０
９０
１００
１７０
１８０
１９０
２００
３１０
３５０
３８０
４００
トラックインデックス
64
65
67
70
72
74
76
80
82
86
88
90
基本域
（プライムエリア）
６４
６５
６７
７０
７２
７４
７６
８０
あふれ域
（オーバフローエリア）
６６
６８
７３
７５

9．区分編成ファイル（Partitioned File）

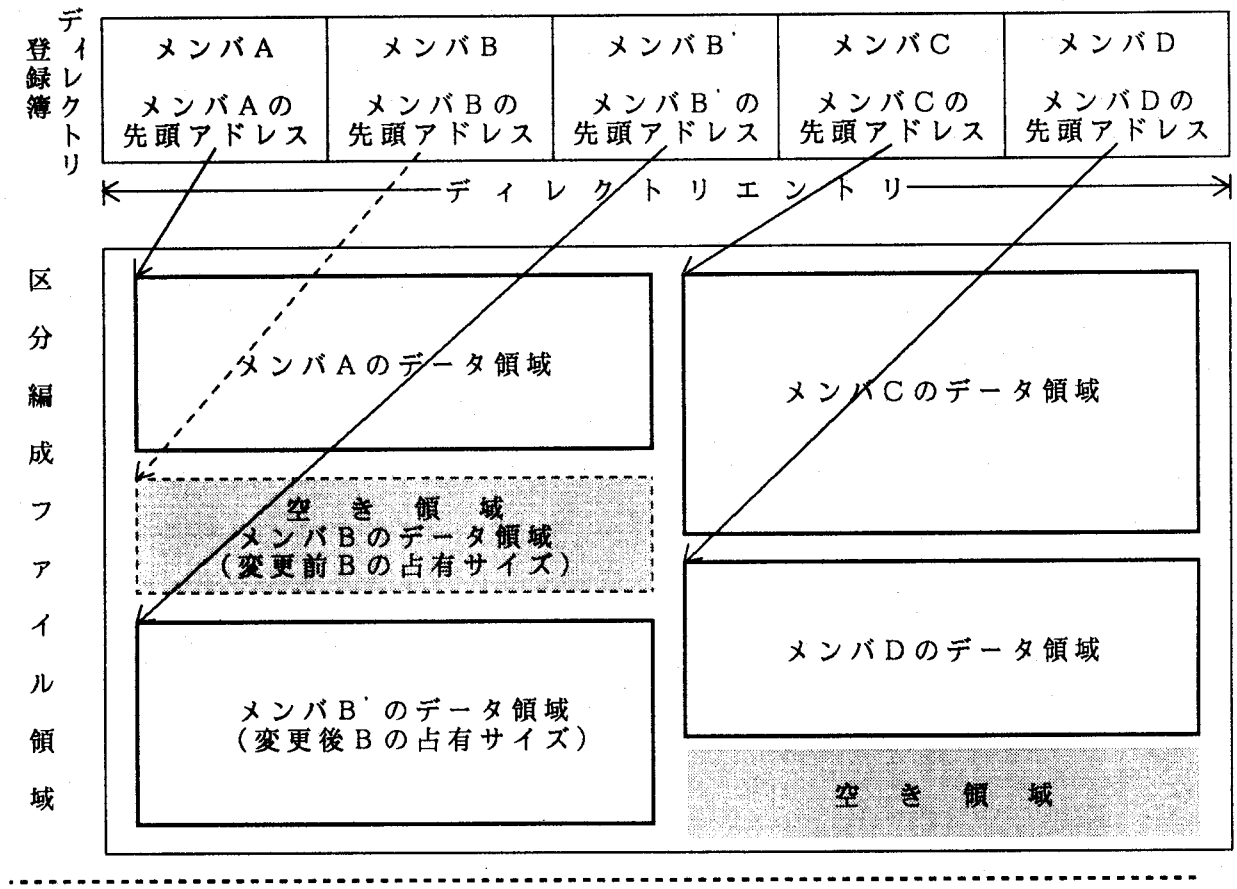
複数個の順編成データ（これをメンバという）を一括まとめて管理を行うファイルを区分編成ファイルという。メンバが開設されると、空き領域からデータ領域が確保される。

区分編成ファイルのメンバ用データ領域は、空き領域管理方式により連続する領域として必要サイズ分確保される。空き領域の管理、および空き領域を探すルールは以下のようである。

① 最初は全てのデータ領域が空き領域であるが、メンバの追加、修正、削除を繰り返すうちに、空き領域リストが作られる。

② メンバ用の必要領域確保の要求があると、空き領域リスト上で必要サイズに足りかつ最もそれに近い空き領域がとられ、メンバ領域としてディレクトリに登録される。ディレクトリの各エントリにはメンバ名、データ領域のアドレスやサイズが登録される。（メンバの追加処理）

③ メンバの削除要求に対しては、ディレクトリからエントリをはずし、使用中であったデータ領域は空き領域リストに返される。（メンバの削除処理）



④　メンバの修正に対しては、サイズに拡大があった場合、削除と追加の同時処理が行われる。（メンバの更新処理）

⑤　空き領域を探す場合、メンバとメンバとの間の隙間の数が増え、実際には必要サイズ分の領域がありながら、空き領域リスト上からはそれが確保できないことが発生する。このような場合には、各メンバについてファイル内で領域の移動を行って連続空き領域を作る。これを区分順編成ファイルの再構成というが、この処理では、ディレクトリエントリの修正、メンバの位置の修正等を伴う。

なお、区分編成ファイルの使い途として、順編成ファイルに対する管理の利便性に着目し、企業全体あるいは特定プロジェクトにおいて共用されるプログラムやデータのライブラリの管理にはしばしばこの編成法が使われている。

１０．ＶＳＡＭファイル（Virtual Storage Access Method）

直接アクセス記憶上に存在するＶＳＡＭファイルのレコードに対しては、順次アクセス、索引順次アクセス、直接アクセスのいずれの処理も可能である。従来ファイルの編成ではファイルの種類毎にアクセス方式があり、ファイルの使い方に自由度が少なく、また、ファイルの種類が多いときにはプログラムからのファイルアクセスの手続きが繁雑であった。

これに対し、ＶＳＡＭでは、一つの統合化されたアクセス方法により装置からの物理的な独立性の高いファイルアクセスを可能としている。

①　ファイル編成の種類

ＥＳＤＳ（入力順データセット：[Entry Sequencial Data Set]）、ＲＲＤＳ（相対レコードデータセット[Relative Record Date Set]）、ＫＳＤＳ（キー順データセット[Key Sequencial Data Set]）の３種類の形式がサポートされる。

	レコードの構成	アクセスの方法	特　　徴
ＥＳＤＳ	順編成ファイルに対応する。 キー順に整列したレコードを入力順に記録。	順次アクセスを行う。 修正、追加、削除も可能。 逆順アクセスもできる。	・追加、削除はデータセットの再編によって行う。 ・効率は落ちるが、直接アクセスも可能である。

	レコードの構成	アクセスの方法	特　　徴
RRDS	直接編成ファイルに対応する。 レコードのキーを基に変換を施して、データセットの先頭からの相対アドレスを求め、そこに記録	直接アクセスが基本となるが、順次アクセス、逆順アクセスも可能。	・レコードの相対位置をとびとびに指定して、順次アクセスもできる。
KSDS	索引順編成ファイルに対応する。 あふれ桁領域を持たない。	順次アクセスと直接アクセスが可能。 基本的にＲＲＤＳと同じアクセスが可能。	・特定キーをポイントした後、そこからの順次アクセスもできる。 ・索引順ファイルに比べ、オーバフロー領域がない分むだ領域が少ない。

② ＶＳＡＭファイルの構造

ＶＳＡＭでは、直接アクセス記憶装置の物理的な情報を必要としない仮想的なデータセットを想定する。このファイルでは、仮想データ空間上の任意の位置に配置されるデータは実際には直接アクセス記憶上につくられる。

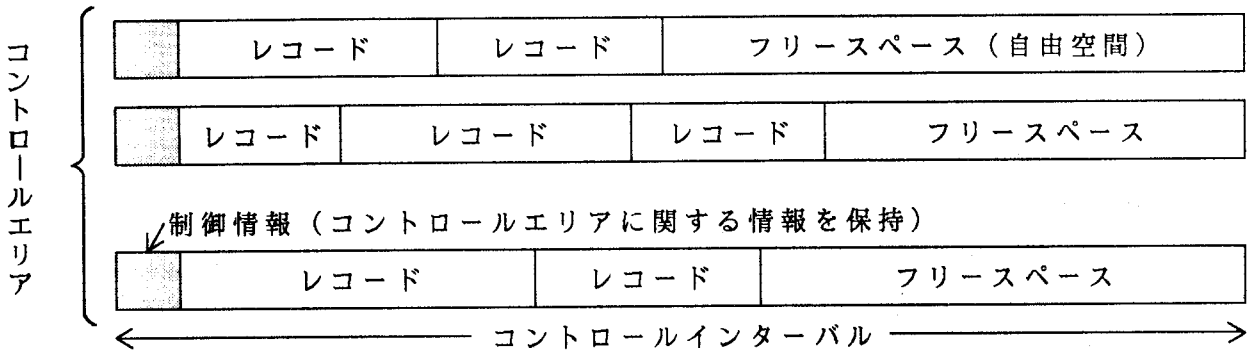
直接アクセス記憶上では、データはコントロールエリア（制御域）とよばれる領域に記録される。さらに、このデータコントロールエリアはコントロールインターバル（制御インターバル）と呼ばれる固定長のデータ空間の集まりとして構成される。

コントロールインターバルは、レコードを収容する空間であり、さらにインターバル内で空きになっている部はをフリースペース（自由空間）と呼ばれる。

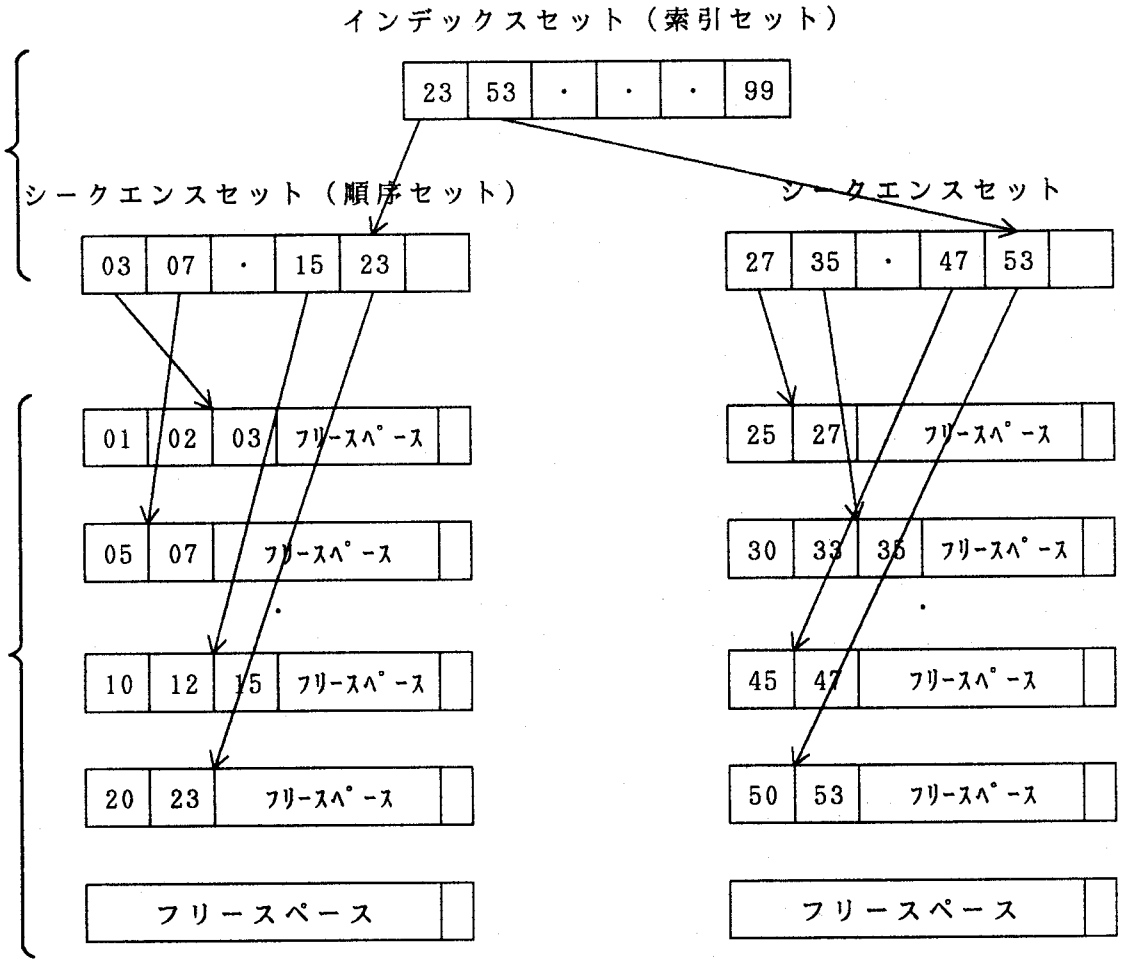
ＶＳＡＭデータセットのレコードは、データセットの開始点をアドレス０として、そこからの相対的なバイトオフセットで位置を捉えることができる。これを相対バイトアドレス（[RBA]Relative Byte Address）という。

ＶＳＡＭの物理的な構造を次頁に示す。

(a) ＥＳＤＳ，ＲＳＤＳ



(b) ＫＳＤＳ



１１．階層構造とディレクトリ

区分編成ファイルは、順編成ファイルの集合としてライブラリ管理などに重要な役割を果たしてきたが、これも第一階の階層構造であり便利さでは今一つである。

これに対して、ＵＮＩＸオペレーティングシステムでは、操作性の高いタイムシェアリングシステムの構築を目指してきただけあり、ファイルの管理に関しては非常に優れた体系をとっている。

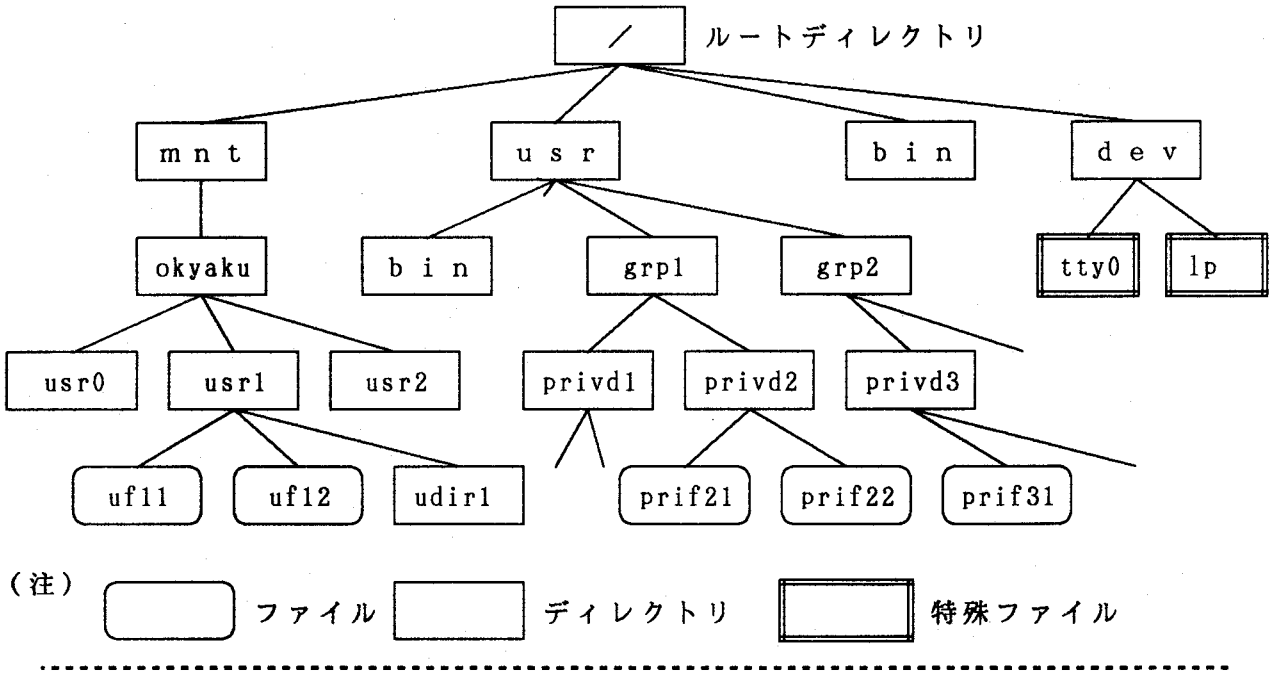
即ち、ｎ層の木構造階層型のファイル管理となっており、ディレクトリが階層構造的に設定できる。このため、システムに一つのファイル管理体系の中で、ディレクトリの機能を使うことにより、

・システム管理系　－　システムを維持するための機密性の高いファイル資産、システムの利用者に共通するファイル資産等を管理する部分
・グループ管理系　－　プロジェクトチームなどを組んだり、部署等を一つの単位としたり、特定のグループに共通するファイル資産を管理する部分
・個人管理系　－　グループの管理下あるいはシステムからの直接の管理下で個人のファイル資産を管理する部分

を独立的に管理・維持が可能である。

このファイル体系は、ＵＮＩＸの普及のみならず、パソコン関係の現在のオペレーティングシステムではほとんどがこの体系をとっている。

≪ディレクトリシステム≫



指導上の留意点

ファイルに関する諸事項に関して、理論的な解説だけで生徒に理解してもらうには色々と限界も出てこよう。また、プログラミングを通して試行錯誤を繰りかえさせて教え込むにも時間的な制約もあろう。

効率上は、できるだけ図解入りの資料を収集し直感的で分かりやすい教え方を行うことに注力が必要となろう。また、生徒の学習意欲をかき立てる上でも、第２種情報処理技術者試験の問題をできるだけ利用して、このレベルでの必須の事項について理解を深めるよう授業を展開することも一つの手であろう。

ファイルを概念的に理解するために、論理レコードと物理レコードについては、本質的に理解させる必要がある。ファイルに関する全ての基本はここにある。

ファイルを格納する物理装置については、日本規格協会編「ＪＩＳハンドブック・情報処理・ソフトウェア編」を参考とするとよい。

ファイル編成については、まず、順編成についてきっちりと理解する必要がある。分類されたレコードファイルは全てこの編成をとっている。

直接編成ファイルについては、アクセスの高速性については理解させ易いものの、その利用効能を教えるのは結構難しい。わかりやすい小利用事例を紹介する必要があろう。

ＶＳＡＭの便利さ、機能性については少々論じただけでは生徒に理解させることはできないであろう。実際に他の編成ファイルのアクセスを種々経験してみて、それらの不便さを感じるとともに、ＶＳＡＭの良さも分かってくる。

従来のファイル編成と比べた優位性をあまり強調せず、ありのままに教えることが得策であろう。実際の指導者にも概念的な良さ程度しか説明できないはずである。これに関しては完璧に説明できる人は少なく、また市販の参考図書の解説も感心できるものは殆ど見あたらない。

用語

ボリューム、磁気ディスク、磁気テープ、フロッピーディスク、単一ボリューム、
複数ボリューム、
ファイル、データセット、ブロック、レコード、フィールド、
物理レコード、論理レコード、ブロック化係数、ブロッキング、デブロッキング、
バッファ、
レコード長、固定長レコード、可変長レコード、不定長レコード、スパンレコード、
ＩＲＧ、ＩＢＧ、トラック方式、セクタ方式、
システムファイル、ユーザファイル、プログラムファイル、データファイル、
マスターファイル、トランザクションファイル、変動ファイル、発生ファイル、更新ファイル、取引ファイル、
ワークファイル、バックアップファイル、ヒストリカルファイル、サマリファイル、
ログファイル、アーカイブファイル、ジャーナルファイル、
永久ファイル、パーマネントファイル、一時ファイル、テンポラリーファイル、
アクセス方式、基本アクセス方式、待機アクセス方式、
順編成ファイル、入力順、昇順、降順、順次アクセス、
直接編成ファイル、アドレス変換、直接アドレス、間接アドレス、シリンダ番号、トラック番号、レコード番号、直接アクセス、シノニム、
相対編成ファイル、相対レコードアクセス、
索引順編成ファイル、基本データ域、プライムエリア、索引領域、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、あふれ域、オーバフローエリア、索引順次アクセス、
区分編成ファイル、登録簿、ディレクトリ、メンバ、メンバ記入項目
ＶＳＡＭファイル、ＥＳＤＳ、ＲＲＤＳ、ＫＳＤＳ、データコントロールエリア、コントロールインターバル、フリースペース索引セット、インデックスセット、順序セット、シークエンスセット、
ルートディレクトリ、特殊ファイル

第2種情報処理技術者試験

ファイルに関連する出題は、２種情報処理技術者試験の出題範囲に関連する事項は下記の通りであるが、“ソフトウェアの知識”に関する１２項目のうち２項目が関連しており、出題率は高くなろう。

さらに、出題されるどのような問題に関してもファイルの概念の理解なしで解けるものは少ない。その意味で本章の学習には十分な時間と、それに見合う生徒の理解度の確認が必要である。

③　ファイル編成に関すること
　　ファイルの定義、編成、アクセス方法に関する基本的な考え方及び各装置の容量計算、データ転送速度など。
⑦　データ構成、データ様式に関すること。
　　ファイルの項目、レコード、ブロック、コード設計、帳票設計など。

第４章　データベースについて

指導目標

適用業務プログラムが増え、規模が大きくなるにしたがってデータ処理をプログラムに依存したファイルだけで進める方式に限界があることをまず理解させる。

ファイルによるデータ管理の欠陥を大きく変えるデータベースについて、その出現の背景、メリット、利用普及促進があった状況を説明する。

次に、データベースシステムの中核となるデータベース管理システムについて説明する。特にスキーマの概念について確実に理解させる。また、データベースの定義、データベースの操作方法の概要についても触れる。

さらに、データベースのデータ構造の理解は、事務処理系のプログラミングにおいては非常に重要な点であり、３種類のモデルについては、それぞれの長所について十分説明する。

内容のあらまし

内　　容	説　明	議　論	机上実習	計算機実習
データベースの目的について	データの独立性、共有性、一貫性、仕様統一、機密性について優れた点を説明する。	ファイルだけで済ませる状況と、データベース化の必要な転換点について、比較論的に議論させる。	特になし	特になし
データベース管理システムについて	データベース管理の考え方、スキーマの作り方、データ操作の基本について解説する。	スキーマの役割、サブスキーマの役割について考えさせる。	ＣＯＢＯＬプログラムでのサブスキーマ定義の練習	汎用コンピュータ ＰＣ，ＷＳ
データベースモデルについて	階層構造モデル、ネットワークモデル、関係モデルについて例を挙げて説明する。	どのモデルがどのようなデータに対して適切かを検討させる。	ＳＱＬによる事例練習	汎用コンピュータ ＰＣ，ＷＳ

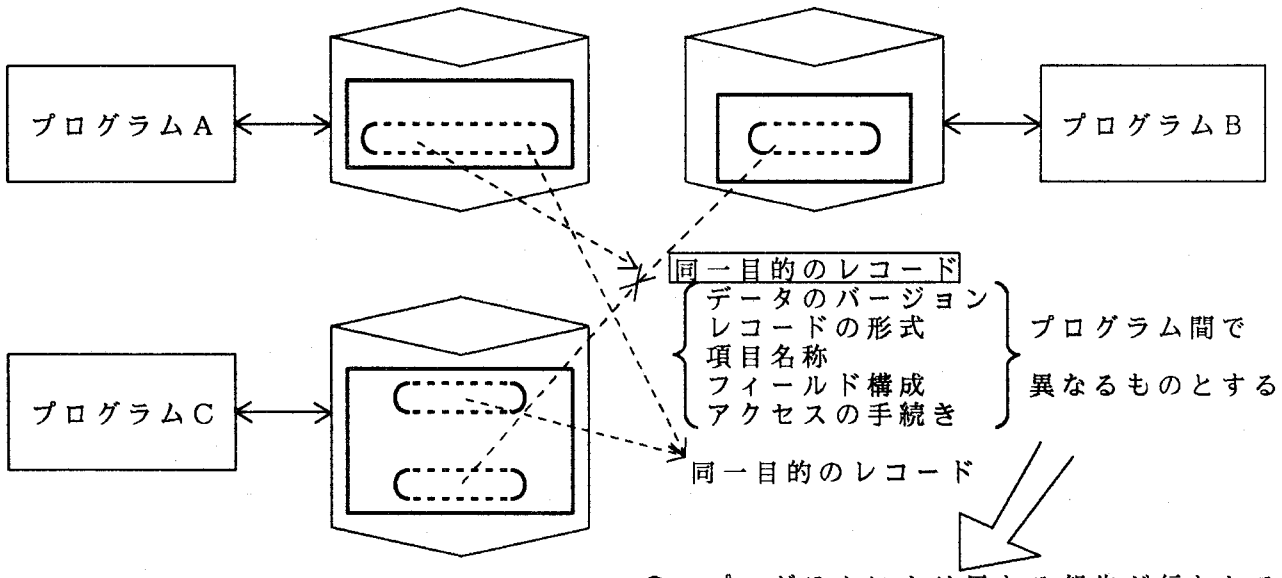
指導内容

1．データベースの目的について

電算化がさほど進んでなく、プログラムやデータの量も多くないときは、処理の対象となるデータはプログラムに依存してファイル化されればそれで十分事足りた。何も他のプログラムがそのファイルを利用したり、あるいは、当該プログラムの扱うデータの種類や形式に大きな変化がなければ、作り付けのデータ記録形式をとれば十分である。

しかしながら、電算化の進展とともにデータの種類は急カーブで増え続け、また、色々な適用業務プログラム（アプリケーションプログラム）が同じ様なデータを個々に作成し、別のファイルとして参照したり、更新したりするに至り、プログラムに依存した形でファイルを使用し続けることに、情報処理部門として、大きな限界にぶつかり始める。

データファイルを、全社で統一的に、またどのようなプログラムからも共通の手続きでアクセスできるようファイルの統合化とその管理体系が必要となったわけである。これがデータベース化の原点である。



○　プログラムにより異なる報告が行われる
○　報告書の分析がしにくくなる
○　プログラム間でファイルを相互利用したいができない
○　データのバージョン合わせも大きな負荷

電算業務におけるデータベース化は、いくつかの大きな利点を得ることを目的に行われる。すなわち、データをプログラムから独立させることにより、プログラム間での共同利用を図る。それはデータが大きくなるほど有効性が増す。また、データの一元的な管理によりどのようなプログラムからも同じ値を得ることができる。さらには、データには機密性の高いものがあり、それらに対して不正なアクセスを防止することができたり、データベースを企業の資産と考えることにより、それを利用した各種のアプリケーションプログラムを開発し、経営戦略や事業戦略に役に立てることも可能となる。

以下で、データベース化により得られる利益について整理する。

データの独立性	プログラムが最初にありその従属としてのデータではなく、まずデータが先に存在し、各種アプリケーションプログラムは必要なデータだけを組み合わせて取り出して、個々の目的に合うデータ処理を行うことができる。 使用の目的ごとにファイル仕様の設計、作成をしないで済む。 プログラムが不要になってもデータは生き残ることができる。
データの共有性	データの仕様、存在は唯一であり、複数の異なる処理を行うプログラム間でデータを共有できる。この共有性により、個々のファイルにあったデータの重複がなくなり、ファイルスペースも少なくて済む。
データの一元管理	データの管理は一カ所で統一的に行えるため、同じデータに対する個々のファイルでの別々の管理も不要となる。どのプログラムからも常に最新の状態のデータをアクセスすることができる。 ＤＢＭＳの機能により、ファイル構成が変わってもプログラムを変更する必要がない。ファイル保守要員の削減にもつながる。
データの一貫性	似たようなデータを個々に持つことがないため、データに書換があってもそれが全てのプログラムに反映される。したがって、どのプログラムからも同じバージョンのデータをアクセスできる。よって、同様なデータ処理については同じ結果を報告できる。
データ仕様の標準化	データの仕様が統一され、利用もしやすく、また維持管理も楽になる。レコードの設計、コード体系に一定の考え方を持つことができる。 この仕様はマシンに依存することがないため、異なるマシンからもアクセスすることが可能となる。
データの機密性	ＤＢＭＳの機能により、アクセス権限を持った端末やプログラムから、許されたアクセスだけを可能とすることができ、データの機密が保持される。

2．データベース管理システムについて

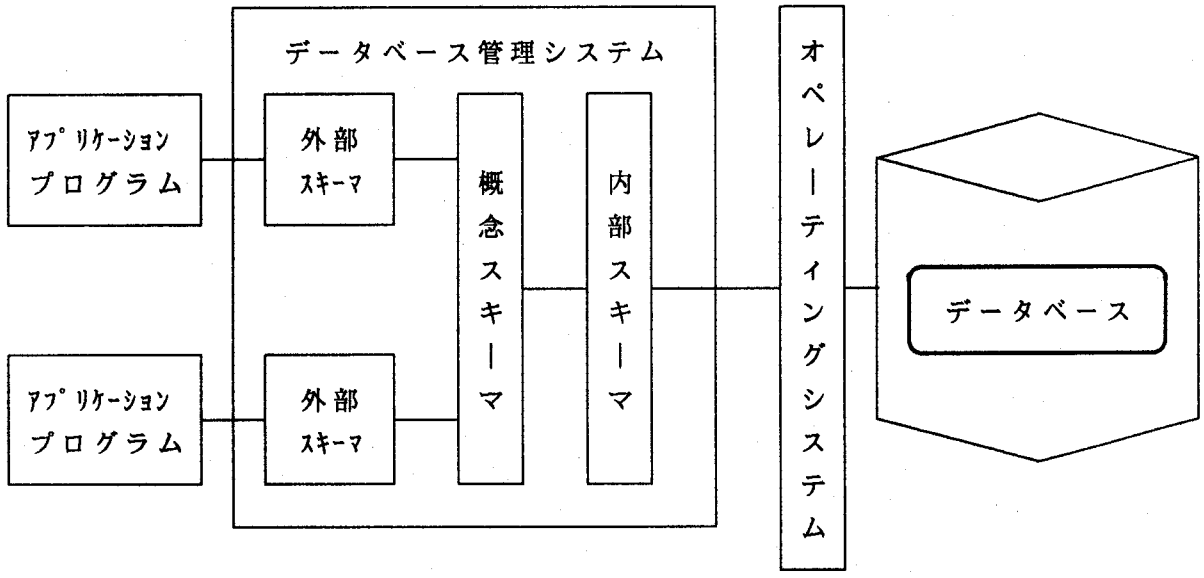
前節で述べたデータベースの利点を実現するソフトウェアをデータベース管理システム（ＤＢＭＳ：［Data Base Management System］）と呼ぶ。

データベース管理システムは、個々に目的を持つ各種アプリケーションプログラムからのデータアクセス要求（検索、更新、削除など）に対して適切なサービスを行う。

データベース管理システムを機能的に捉えると、「データを定義する機能」と「データを操作する機能」とから成り立っている。データベースの管理はこれだけで全てが行われるのではなく、データベースの維持、データ仕様やデータ構造の管理、アプリケーションプログラムとのインタフェースを統括するために、データベース管理者（ＤＢＡ：［Data Base Administrator］が必要になる。この管理者の判断や指示なくして、データの仕様や構造が勝手に変わるようなことがあってはならない。

データベース管理システムの機能に対応して、「データベース定義言語（ＤＤＬ：［Data Definition Language］）」と、「データ操作言語（ＤＭＬ：［Data Manipulation Language］）」と呼ぶ言語がある。前者は、データベース管理者がスキーマ（後述）というデータベースの定義を作成するもので、後者は、データベースの利用者がアプリケーションプログラムの中でデータの検索、更新を行うために使用する。

下図にデータベース管理システムとアプリケーションプログラムとの構造を示す。
（ＣＯＤＡＳＹＬ型データベースの場合）



データベース管理システムでは、データベースの構造をスキーマ（schema）によって定義する。このスキーマは3つの階層をとっており、それぞれ、データベース全体の論理構造を記述する概念スキーマ（conceptual schema）、データの格納構造を記述する外部スキーマ（external schema）、両スキーマをコンピュータで表現するための物理構造を記述する内部スキーマ（internal schema）からなる。

以下で簡単に各スキーマについて説明する。

外部スキーマ	アプリケーションプログラムからのデータの見方を記述するもの。データベースをアクセスするアプリケーションプログラムが、常にデータベース全体の構造を意識することは大変なことである。そのプログラムが扱う部分的なデータベースを定義することによりデータ操作対象の世界を描ける。 即ち、プログラムで処理するデータの論理構造を宣言することで、データベース全体構造を意識せずにすむ。外部スキーマはコンピュータの言語ごとに構造が異なり、言語側が概念スキーマのレコード形式に変換する。 このスキーマをサブスキーマ（sub schema）とも呼ぶ。
概念スキーマ	現実世界でのデータの見方を記述するもの。単にスキーマとも呼び、データベースイメージそのものを表現している。　データベースに唯一つ存在し、データモデルによるデータの論理的な構造をコンピュータによる物理的な管理の構造と対応づける。この情報はデータ操作時に参照される。
内部スキーマ	コンピュータ側からのデータの見方を記述するもので、物理的な構造を管理する。概念スキーマで定義したデータの論理構造に関してコンピュータによる物理的な記録方法を表現している。 データの実際の入出力操作時には、コンピュータはこの情報を用いる。

実際のデータベースの利用（データ構造の定義と、データの操作）においては、ＤＤＬとＤＭＬが使われる。

（１）データ構造の定義

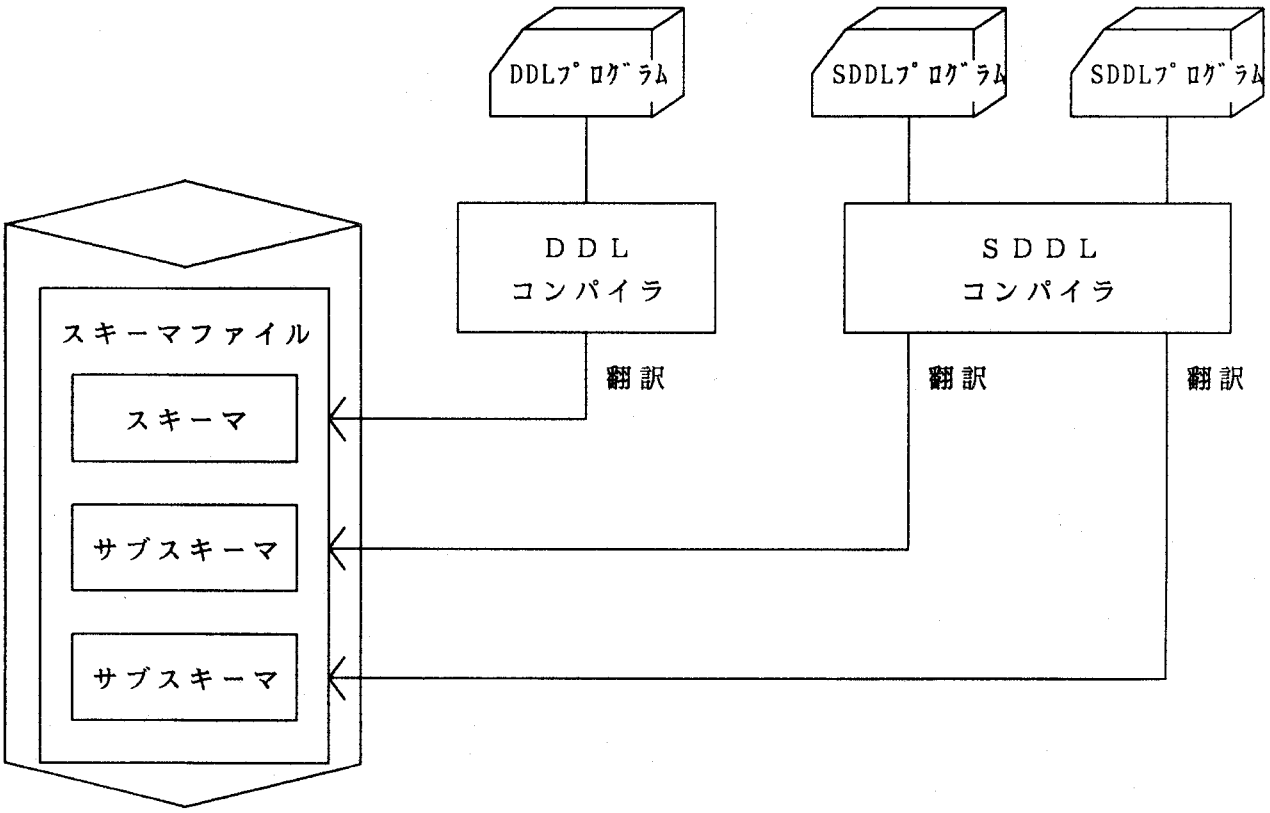
①　データベース全体の構造の定義

ＤＤＬ（データ定義言語）プログラムをＤＤＬコンパイラで翻訳して、スキーマファイルを作る。これはデータベースに１つだけ存在し、データの論理構造と物理構造とを対応づけるものであり、データのアクセス時には必ずこのスキーマが参照される。

②　アプリケーションプログラムにとってのデータ構造の定義

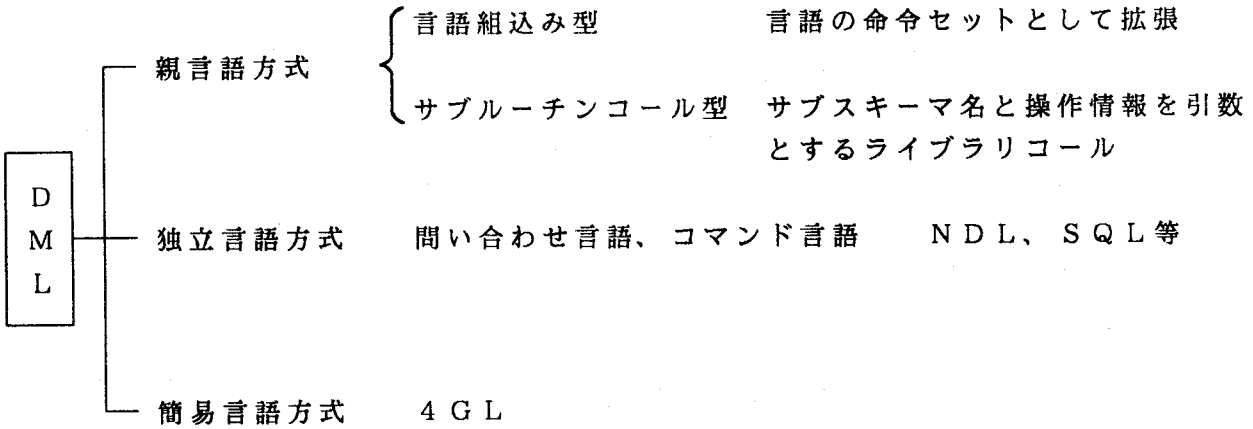
データベースをアクセスするアプリケーションプログラムごとにＳＤＤＬプログラムを書き、ＳＤＤＬコンパイラで翻訳してサブスキーマをスキーマファイルに作る。

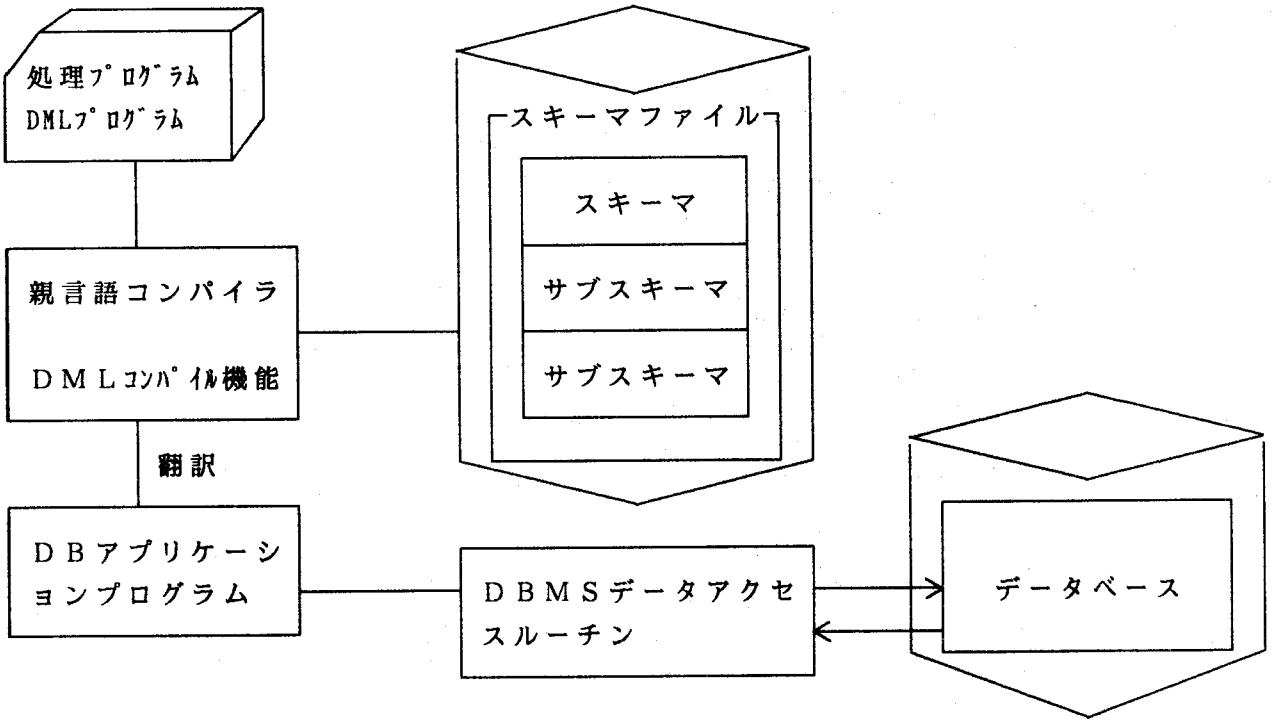
サブスキーマは、プログラムが操作するレコードの構造やデータベース全体とを対応づけるもので、アプリケーションプログラムをコンパイルするときに参照される。



（２）データの操作

アプリケーションプログラムはＤＭＬ（データ操作言語）を使って操作手続き（検索、更新、追加、削除）を記述する。





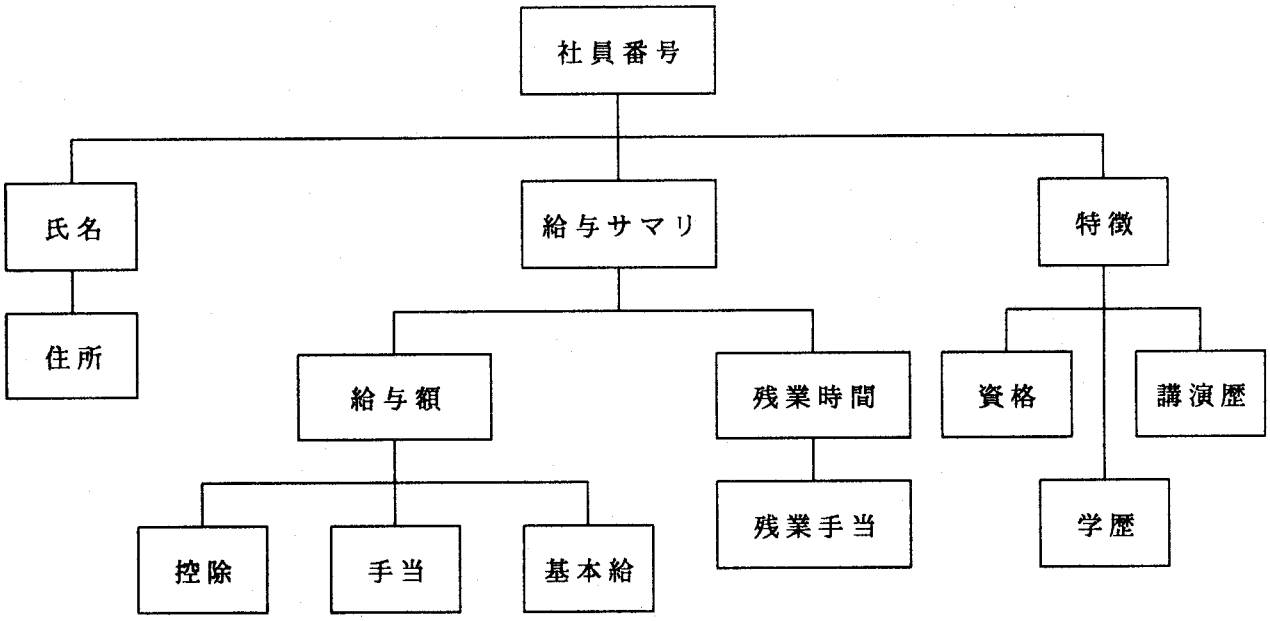
3．データベースモデルについて

データベースで使われるデータの論理的な関連付けを行うことをデータのモデリングという。これは、論理レコード中のフィールド間の関係を表す論理データ構造であり、階層構造（hierarchy structure）モデル、網構造／ネットワーク構造（network structure）モデル、関係（relational)モデルの３種類が代表的な例である。

（１）階層構造モデル

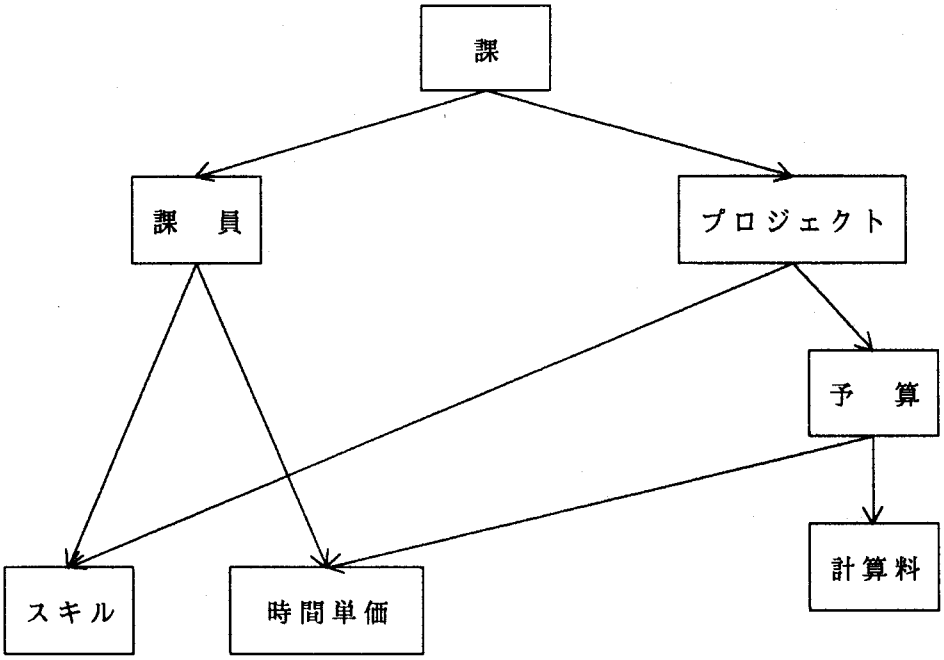
木構造（tree structure）をベースとしており、すべてのレコードは親子関係にあり、親からみた子レコードは複数個あるが、子からみた親レコードは唯一つであるような構造をとる。

このモデルでは、子レコードが複数の親レコードにつながることがないため、場合により同じレコードを持たなければならないことがありうる。



（２）網構造モデル

階層構造モデルの欠陥を補い、上下、水平、いずれも関連付けができる構造である。レコード間はポインタにより接続している。ＣＯＤＡＳＹＬ方式データベースはこのモデルの代表的なものである。



（３）関係モデル

リレーショナルモデルともいい、データを２次元の表形式で表すデータベースである。このモデルでは、この２次元表（relation）を複数個作る。また、それぞれの表間の関係付けを行う。データ構造を作るのではなく、各表に対して数学的な関係により、データモデルを表現するのが特徴である。

この表はスキーマと対比して考えることができる。また表自身をファイルに、行をレコードに、列をフィールドに対応することができる。行のことを組み（Tuple）、列のことを属性（Atribute）という。

関係モデルでは、表間に対する演算によって、新しい関係表を作ることができる。代表的な演算は以下の通りである。

［集合演算］

① 合併（union）
２つの表の全ての組の集合。

② 共通部分（intersection）
２つの表の共通する組の集合。

③ 差（difference）
ある表に属し、他の表に属さない全ての組の集合。

④ 直積（product）
ある表の組と他の表の組とを連結した組の集合。

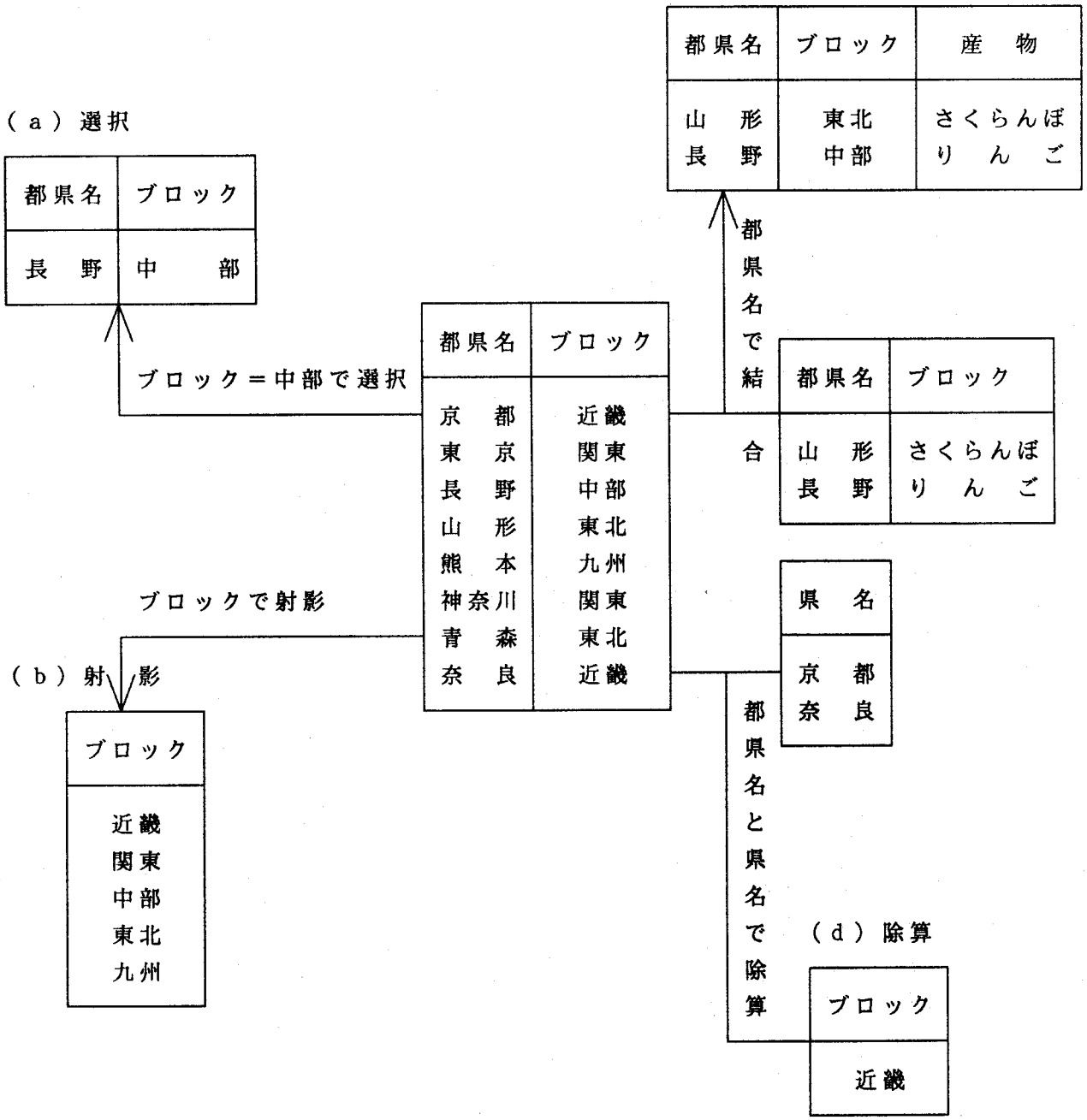
［関係演算］

① 選択（selection）
表中から、ある条件を満たすレコードを取り出し、新しい表を作る。

② 射影（projection）
表中から、ある条件を満たす項目（列）を取り出し、新しい表を作る。

③ 結合（join）
複数の表から条件に合致した組を結合して、新しい表を作る。

≪関係演算の例≫　　（出典：初級情報処理技術射育成指針：中央情報教育研究所）



指導上の留意点

まず、ファイルについて十分理解していない段階でこのテーマについて教えることは危険である。ファイルの限界についてある程度の認識を持たないとデータベースの意義を感じとることができない。

データベースがないと実習は困難であり、机上でできる演習に限定されよう。

この章については、まだ２種の試験には出題されていないので、あまり時間をかける必要もない。

用　語

ＤＢＭＳ、データベース管理システム、ＤＢＡ、データベース管理者、
ＤＤＬ、データ定義言語、ＤＭＬ、データ操作言語、ＣＯＤＡＳＹＬ型データベース、
スキーマ、外部スキーマ、概念スキーマ、内部スキーマ、サブスキーマ、
親言語方式、独立言語方式、コマンド方式、簡易言語方式、
データモデル、階層構造モデル、木構造モデル、網構造モデル、ネットワークモデル、
関係モデル、表、組、属性、
集合演算、合併、共通部分、差、直積、関係演算、選択、射影、結合